University of West Bohemia

Faculty of Applied Sciences

Doctoral Dissertation

2021 Michael HEIGL, M.Sc.

University of West Bohemia
Faculty of Applied Sciences

ENHANCING COMPUTER NETWORK SECURITY THROUGH IMPROVED OUTLIER DETECTION FOR DATA STREAMS

Michael HEIGL, M.Sc.

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in “Computer Science and Engineering”

Supervisor: doc. Ing. Dalibor Fiala, Ph.D.

Department: Department of Computer Science and Engineering

Pilsen 2021

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

ZVYŠOVÁNÍ BEZPEČNOSTI POČÍTAČOVÝCH SÍTÍ ZESÍLENOU DETEKCÍ ODLEHLÝCH HODNOT V DATOVÝCH TOCÍCH

Michael HEIGL, M.Sc.

disertační práce
k získání akademického titulu doktor
v oboru „Informatika a výpočetní technika“

Školitel: doc. Ing. Dalibor Fiala, Ph.D.

Katedra: Katedra informatiky a výpočetní techniky

Plzeň 2021

Declaration

I hereby submit for evaluation and defense, following the dissertation prepared at the end of the doctoral study program at the University of West Bohemia in Pilsen. I declare that I have written the present thesis independently, without assistance from external parties and without use of other resources than those indicated. The ideas taken directly or indirectly from external sources (including electronic sources) are duly acknowledged in the text. The material, either in full or in part, has not been previously submitted for grading at this or any other academic institution.

Aiterhofen, Germany, September 2021

Michael Heigl

Abstract

The omnipresent internetworking of embedded devices in all areas of life is driven by various trends such as the Internet of Things or Autonomous Vehicles and the emergence of disruptive technologies such as Artificial Intelligence or Quantum Computing. However, adversaries also benefit from these developments. Not only the way to fast progression of the rapidly advancing and pervading technologies is leading to a broad spectrum of security problems but also the increasing capabilities for adversaries in terms of tools, methods and technologies, which place great demands on innovative security solutions. The identification of network-based attacks or unknown behavior through an Intrusion Detection System (IDS) has established itself as a conducive and mandatory mechanism apart from the protection by cryptographic schemes in a holistic security eco-system. Nevertheless, these systems show various limitations when it comes to reliably identifying and reacting to cyber attacks. Over the past couple of years, machine learning methods - especially the Outlier Detection (OD) ones - have become anchored to the cyber security field to detect network-based anomalies rooted in novel attack patterns. Due to the steady increase of high-volume, high-speed and high-dimensional Streaming Data (SD), for which ground truth information is not available, detecting anomalies in real-world computer networks has become a more and more challenging task. Efficient detection schemes applied to networked, embedded devices need to be fast and memory-constrained, and must be capable of dealing with concept drifts when they occur.

The aim of this thesis is to enhance computer network security through improved OD for data streams, in particular SD, to achieve cyber resilience, which ranges from the detection, over the analysis of security-relevant incidents, e.g., novel malicious activity, to the reaction on them. Therefore, four major contributions are proposed, each divided into sub-novelties.

First, Feature Selection (FS) plays an important role when it comes to improved OD in terms of identifying noisy data that contains irrelevant or redundant features. State-of-the-art work either focuses on unsupervised FS for SD or (offline) OD. Requirements to combine both fields are derived and compared with existing approaches. The comprehensive review reveals a research gap in unsupervised FS for the improvement of off-the-shell OD methods in data streams. This gap is filled by proposing ***U**nsupervised **F**eature **S**election for **S**treaming **O**utlier **D**etection*, denoted as **UFSSOD**. A generic concept is retrieved that shows two application scenarios of UFSSOD in conjunction with online OD algorithms. Extensive experiments have shown that a promising FS mechanism for SD is not applicable in the field of OD. Moreover, UFSSOD, as an online-capable algorithm, yields comparable results with a state-of-the-art offline method trimmed for OD.

Second, a novel unsupervised online OD framework called ***P**erformance **C**ounter-**B**ased **iForest*** (**PCB-iForest**) is being introduced, which generalized, is able to incorporate any ensemble-based online OD method to function on SD. Carefully engineered requirements are compared to the most popular state-of-the-art online methods with an in-depth focus on variants based on the widely accepted Isolation Forest (iForest) algorithm, thereby highlighting the lack of a flexible and efficient solution which is satisfied by PCB-iForest. Therefore, two variants are integrated into PCB-iForest - an iForest improvement called Extended Isolation Forest and a classic iForest variant equipped with the functionality to score features according to their contributions to a sample's anomalousness. Extensive experiments were performed on 23 different multi-disciplinary and security-related real-world data sets in order to comprehensively evaluate the performance of our implementation compared with off-the-shelf methods. The discussion of results, including AUC, $F1$ score and averaged execution time metric, shows that PCB-iForest clearly outperformed the state-of-the-art competitors in 61% of cases and even achieved more promising results in terms of the tradeoff between classification and computational costs.

Third, consecutively applied Alert Correlation (AC) methods can aid in mining attack patterns based on the alerts generated by IDS. However, most of the existing methods lack the functionality to deal with SD and are mainly designed to operate on the output from misuse-based IDS. Although unsupervised OD methods have the ability to detect yet unknown attacks, most of the AC algorithms cannot handle the outcome of such anomaly-based IDS. Thus, a novel framework called ***S**treaming **O**utlier **A**nalysis and **A**ttack **P**attern **R**ecognition*, denoted as **SOAAPR** is being introduced. SOAAPR is able to process the output of various online unsupervised OD methods in a streaming fashion to extract information about novel attack patterns. Three different privacy-preserving, fingerprint-like signatures are computed from the clustered set of correlated alerts by SOAAPR, which characterize and represent the potential attack scenarios with respect to their communication relations, their manifestation in the data's features and their temporal behavior. Beyond the recognition of known attacks, comparing derived signatures, can be leveraged to find similarities between yet unknown and novel attack patterns. The evaluation - split into two parts - takes advantage of attack scenarios from the widely-used and popular CICIDS2017 and CSE-CIC-IDS2018 data sets. Firstly, the streaming AC capability is evaluated on CICIDS2017 and compared to a state-of-the-art offline algorithm, called Graph-based Alert Correlation (GAC), that has the potential to deal with the outcome of anomaly-based IDS. Secondly, the three types of signatures are computed from attack scenarios in the data sets and compared to each other. The discussion of results shows that SOAAPR can compete with GAC in terms of AC capability and outperforms it significantly in terms of processing time. Moreover, in most cases all three types of signatures seem to reliably characterize attack scenarios to the effect that similar ones are grouped together.

Fourth, until now, network security through cryptographic protection and intrusion detection has been regarded as separate disciplines. Thus, an ***U**ncoupled **M**essage **A**uthentication **C**ode* algorithm - **Uncoupled MAC** - is presented which builds a bridge between the two disciplines. It secures network communication (authenticity and integrity) through a cryptographic scheme with layer-2 support via uncoupled message authentication codes but, as a side effect, also provides IDS-functionality producing alarms based on the violation of Uncoupled MAC values. Through a novel self-regulation extension, the algorithm adapts its sampling parameters based on the detection of malicious actions. Sampling has become useful detecting malicious activity in a manageable amount of SD, especially in systems where resources are valuable goods and stand in contrast to the ever increasing amount of network traffic. The evaluation in a virtualized environment clearly shows that the detection rate increases over runtime for different attack scenarios. Those even cover scenarios in which intelligent attackers try to exploit the downsides of sampling.

Keywords: network security, machine learning, intrusion detection, outlier detection, streaming data, online learning, unsupervised learning, feature selection, alert analysis, alert correlation, attack scenario, message authentication, self-regulation

Abstrakt

Všudypřítomné propojení vestavěných zařízení ve všech sférách života je doprovázeno trendy typu internet věcí nebo autonomní vozidla a nástupem disruptivních technologií jako jsou umělá inteligence nebo kvantové počítače. Avšak z tohoto vývoje profitují i potenciální útočníci. Nejen že překotný vývoj těchto rychle postupujících a vše pronikajících technologií vede k širokému spektru bezpečnostních problémů, ale také ke zvyšování schopnosti útočníků užívat nástroje, metody a technologie, jež kladou velké nároky na inovativní řešení v oblasti bezpečnosti. Identifikace útoků nebo podezřelého chování v počítačových sítích systémem pro detekci vniknutí (IDS) se vedle ochrany kryptografickými prostředky etablovala jako napomáhající a povinný mechanismus holistických bezpečnostních ekosystémů. Nicméně, pokud jde o spolehlivou detekci a reakci na kyberútoky, tyto systémy narážejí na nejrůznější omezení. V několika posledních letech se metody strojového učení (zvláště ty zabývající se detekcí odlehlých hodnot - OD) v oblasti kyberbezpečnosti opíraly o zjišťování anomálií síťového provozu spočívajících v nových schématech útoků. Detekce anomálií v počítačových sítích reálného světa se ale stala stále obtížnější kvůli trvalému nárůstu vysoce objemných, rychlých a dimenzionálních průběžně přicházejících dat (SD), pro která nejsou k dispozici obecně uznané a pravdivé informace o anomalitě. Účinná detekční schémata pro vestavěná síťová zařízení musejí být rychlá a paměťově nenáročná a musejí být schopna se potýkat se změnami konceptu, když se vyskytnou.

Cílem této disertace je zlepšit bezpečnost počítačových sítí zesílenou detekcí odlehlých hodnot v datových proudech, obzvláště SD, a dosáhnout kyberodolnosti, která zahrnuje jak detekci a analýzu, tak reakci na bezpečnostní incidenty jako jsou např. nové zlovolné aktivity. Za tímto účelem jsou v práci navrženy čtyři hlavní příspěvky, z nichž každý obsahuje několik inovativních řešení.

Zaprvé, volba vlastností (FS) hraje důležitou roli při zlepšování OD identifikací zašuměných dat, která obsahují nerelevantní nebo nadbytečné vlastnosti. Současný výzkum se zaměřuje buď na FS pro SD bez učitele, nebo na (offline) OD. V této práci jsou zformulovány požadavky na kombinaci obou přístupů a dále jsou porovnány s existujícími řešeními. Obsáhlá rešerše odhalila mezeru v FS bez učitele pro zlepšování již hotových metod OD v datových tocích, která byla zaplněna navržením *volby vlastností bez učitele pro detekci odlehlých průběžně přicházejících dat* označované jako **UFSSOD**. Následně odvozujeme generický koncept, který ukazuje dva aplikační scénáře UFSSOD ve spojení s online algoritmy OD. Rozsáhlé experimenty ukázaly, že slibný mechanismus FS pro SD není v oblasti OD k dispozici. Navíc UFSSOD coby algoritmus schopný online zpracování vykazuje srovnatelné výsledky jako současná nejlepší offline metoda upravená pro OD.

Zadruhé představujeme nový aplikační rámec nazvaný *izolovaný les založený na počítání výkonu* (**PCB-iForest**), jenž je obecně schopen využít jakoukoliv online OD metodu založenou na množinách dat tak, aby fungovala na SD. Pečlivě zformulované požadavky srovnáváme s nejpopulárnějšími stávajícími uznávanými online metodami se zvláštním zřetelem na varianty široce přijímaného algoritmu izolovaného lesa (iForest) a ukazujeme při tom, že dosud neexistovalo flexibilní a výkonné řešení, které přináší až algoritmus PCB-iForest. Proto do tohoto algoritmu integrujeme dvě varianty – zlepšení izolovaného lesa, jež se nazývá rozšířený izolovaný les, a klasickou variantu izolovaného lesa vybavenou funkcionalitou k ohodnocení vlastností podle jejich přispění k anomalitě datového vzorku. Provádíme rozsáhlé experimenty na 23 multidisciplinárních datových sadách týkajících se bezpečnostní problematiky reálného světa za účelem podrobného srovnání naší implementace s již existujícími metodami. Diskuse našich výsledků zahrnující indikátory AUC,

$F1$ a průměrnou dobu zpracování ukazuje, že PCB-iForest jasně překonává už zavedené konkurenční metody v 61% případů a dokonce dosahuje ještě slibnějších výsledků co do vyváženosti mezi výpočetními náklady na klasifikaci a její úspěšností.

Zatřetí, postupně aplikované metody korelace poplachů (AC) mohou pomoci při dolování ze schémat útoku na základě poplachů generovaných IDS. Nicméně většina existujících metod postrádá funkcionalitu pro práci s SD a je převážně navržena pro zpracování výstupu z IDS založených na zneužití sítě. Ačkoliv OD metody bez učitele mají schopnost rozpoznat dosud neznámé útoky, většina algoritmů AC neumí zacházet s výsledky z takových IDS založených na detekci anomálií. Právě proto představujeme nový pracovní rámec nazvaný *detekce odlehlých hodnot a rozpoznávání schémat útoku proudovým způsobem* (**SOAAPR**), jenž je schopen zpracovat výstup z různých online OD metod bez učitele proudovým způsobem, aby získal informace o nových schématech útoku. Ze seshlukované množiny korelovaných poplachů jsou metodou SOAAPR vypočítány tři různé soukromí zachovávající podpisy podobné otiskům prstů, které charakterizují a reprezentují potenciální scénáře útoku s ohledem na jejich komunikační vztahy, projevy ve vlastnostech dat a chování v čase. Mimo rozpoznávání známých útoků může být porovnání odvozených podpisů využito k nalezení podobností mezi dosud neznámými a novými schématy útoku. Evaluace, jež je rozdělena do dvou částí, se opírá o schémata útoku ze dvou široce používaných a oblíbených datových sad CICIDS2017 a CSE-CIC-IDS2018. Nejprve se schopnost AC na průběžně přicházejících datech vyhodnocuje na CICIDS2017 a srovnává se stávajícím nejlepším offline algoritmem nazývaným korelace poplachů založená na grafech (GAC), který má potenciál zpracovávat výstupy z IDS založených na detekci anomálií. Potom se ze scénářů útoku v datových sadách vypočítají tři typy podpisů a porovnají navzájem. Diskuse výsledků ukazuje na jedné straně, že SOAAPR může soupeřit s GAC ve schopnosti AC a významně jej překonává z hlediska výpočetního času. Na druhé straně se všechny tři typy podpisů ve většině případů zdají spolehlivě charakterizovat scénáře útoků tím, že podobné seskupují k sobě.

Začtvrté, až do dnešní doby se zabezpečení počítačových sítí kryptografickými mechanismy a detekce vniknutí považovaly za oddělené disciplíny. Proto představujeme algoritmus *nepárového kódu autentizace zpráv* (**Uncoupled MAC**), který obě oblasti propojuje. Zabezpečuje síťovou komunikaci (autenticitu a integritu) kryptografickým schématem s podporou druhé vrstvy kódy autentizace zpráv, ale také jako vedlejší efekt poskytuje funkcionalitu IDS tak, že vyvolává poplach na základě porušení hodnot nepárového MACu. Díky novému samoregulačnímu rozšíření algoritmus adaptuje svoje vzorkovací parametry na základě zjištění škodlivých aktivit. Vzorkování se stalo užitečným nástrojem k detekci škodlivých aktivit ve zpracovatelném množství SD, zvláště v systémech, kde jsou zdroje ceněným zbožím a jsou v příkrém rozporu se stále vzrůstajícím množstvím síťového provozu. Evaluace ve virtuálním prostředí jasně ukazuje, že schopnost detekce se za běhu zvyšuje pro různé scénáře útoku. Ty zahrnují dokonce i situace, kdy se inteligentní útočníci snaží využít slabá místa vzorkování.

Klíčová slova: síťová bezpečnost, strojové učení, detekce vniknutí, detekce odlehlých hodnot, průběžně přicházející data, online učení, učení bez učitele, volba vlastností, analýza poplachů, korelace poplachů, scénář útoku, autentizace zpráv, samoregulace

Zusammenfassung

Die allgegenwärtige Vernetzung von eingebetteten Geräten in allen Lebensbereichen wird durch verschiedene Trends, wie das Internet der Dinge oder autonome Fahrzeuge, und das Aufkommen bahnbrechender Technologien, wie künstliche Intelligenz oder Quantencomputer, vorangetrieben. Doch auch die Gegner profitieren von diesen Entwicklungen. Nicht nur der rasante Fortschritt der sich stetig entwickelnden und alles durchdringenden Technologien führt zu einem breiten Spektrum an Sicherheitsproblemen, sondern auch die Fähigkeiten der Angreifer in Bezug auf Werkzeuge, Methoden und Technologien, die große Anforderungen an innovative Sicherheitslösungen stellen, nehmen zu. Die Identifizierung von Netzwerk-basierten Angriffen oder unbekanntem Verhalten durch Angriffserkennungssysteme (IDS) hat sich neben dem Schutz durch kryptografische Verfahren in einem ganzheitlichen Sicherheits-Ökosystem als förderlicher und obligatorischer Mechanismus etabliert. Dennoch weisen diese Systeme verschiedene Einschränkungen auf, wenn es darum geht, Cyberangriffe zuverlässig zu erkennen und darauf zu reagieren. In den letzten Jahren haben sich Methoden des maschinellen Lernens, insbesondere Methoden der Ausreißererkennung (OD), im Bereich der Cybersicherheit etabliert, um netzwerkbasierte Anomalien zu erkennen, die auf neuartigen Angriffsmustern beruhen. Aufgrund der stetigen Zunahme von hochvolumigen, schnellen und hochdimensionalen Streaming-Daten (SD), für die keine Lerndaten verfügbar sind, ist die Erkennung von Anomalien in realen Computernetzwerken zu einer immer größeren Herausforderung geworden. Effiziente Erkennungsverfahren für vernetzte eingebettete Geräte müssen schnell und speicherbegrenzt sein und mit Drifts in Daten umgehen können, sofern diese auftreten.

Ziel dieser Arbeit ist es, die Sicherheit von Computernetzwerken durch verbesserte OD für Datenströme, insbesondere SD, zu erhöhen, um Cyber-Resilienz zu erreichen, die von der Erkennung, über die Analyse sicherheitsrelevanter Vorfälle, z.B. neuartiger bösartiger Aktivitäten, bis hin zur Reaktion darauf reicht. Daher werden vier Hauptbeiträge aufgeführt, die jeweils in Unterbeiträge unterteilt sind.

Die Merkmalsauswahl (FS) spielt eine wichtige Rolle bei der Verbesserung von OD, wenn es darum geht, fehlerhafte Daten zu identifizieren, die irrelevante oder redundante Merkmale (Features) enthalten. Der Stand der Technik konzentriert sich entweder auf unüberwachte Verfahren von FS für SD oder für den Fall von (offline) OD. Es werden Anforderungen an die Kombination beider Bereiche abgeleitet und mit bestehenden Ansätzen verglichen. Im umfassenden Überblick zeichnet sich eine Forschungslücke im Bereich der unüberwachten FS zur Verbesserung von Standardmethoden für OD in Streaming-Daten ab. Diese Lücke wird durch den Vorschlag der *Unüberwachte Merkmalsauswahl für streamingverarbeitende Ausreißererkennung*, auch als **UFSSOD** bezeichnet, geschlossen. Es wird ein generisches Konzept vorgestellt, das zwei Anwendungsszenarien von UFSSOD in Verbindung mit online-fähigen OD-Algorithmen zeigt. Ausführliche Experimente haben erwiesen, dass ein vielversprechender FS-Mechanismus für SD im Bereich von OD nicht anwendbar ist. Darüber hinaus liefert UFSSOD als online-fähiger Algorithmus vergleichbare Ergebnisse wie ein modernes, auf OD getrimmtes Offline-Verfahren.

Als weiterer Hauptbeitrag wird ein neuartiges unüberwachtes online OD-Framework namens *Leistungszähler-basierender Isolation Forest* (**PCB-iForest**) eingeführt, das, verallgemeinert, in der Lage ist, jedes Ensemble-basierte online OD-Verfahren zu integrieren, um auf SD zu funktionieren. Sorgfältig ausgearbeitete Anforderungen werden mit den gängigsten Online-Methoden verglichen, wobei der Schwerpunkt auf Varianten liegt, die auf dem weithin akzeptierten iForest-Algorithmus basieren. Daher werden zwei Varianten in PCB-iForest integriert - eine iForest-Verbesserung namens Extended Isolation Forest und eine klassische iForest-Variante, die mit der Funktion ausgestattet ist, Merkmale entsprechend ihrem Beitrag zur Anomalität einer Probe zu bewerten. Umfangreiche Experimente wurden mit 23 verschiedenen, multidisziplinären und sicherheitsrelevanten realen Datensätzen durchgeführt, um die Leistung unserer Implementierung im Vergleich zu Standardmethoden umfassend zu bewerten. Die Diskussion der Ergebnisse, einschließlich AUC, $F1$-Score und gemittelter Ausführungszeitmetrik, zeigt, dass PCB-iForest in 61% der Fälle Konkurrenten des aktuellen Forschungsstands deutlich übertraf und sogar vielversprechendere Ergebnisse in Bezug auf den Kompromiss zwischen Genauigkeit und Ressourcenbedarf erzielte.

Konsekutiv angewandte Methoden der Alarmanalyse bzw. -korrelation (AC) können beim Mining von Angriffsmustern auf der Grundlage der von IDS generierten Alarme helfen. Den meisten der vorhandenen Methoden fehlt jedoch die Funktionalität, um mit SD umzugehen, und sie sind hauptsächlich auf die Ausgabe von Signatur-basierten IDS ausgelegt. Obwohl unüberwachte OD-Methoden in der Lage sind noch unbekannte Angriffe zu erkennen, können die meisten AC-Algorithmen nicht mit den Ergebnissen solcher, auf Anomalien basierenden IDS, umgehen. Daher wird ein neuartiges Framework namens *Streamingfähige Ausreißeranalyse und Angriffsmustererkennung* eingeführt, welches als **SOAAPR** bezeichnet wird. SOAAPR ist in der Lage, die Ausgabe verschiedener online unüberwachter OD-Methoden in einem Streaming-Verfahren zu verarbeiten, um Informationen über neuartige Angriffsmuster zu extrahieren. Aus der geclusterten Menge korrelierter Alarme werden von SOAAPR drei verschiedene datenschutzkonforme, fingerabdruckähnliche Signaturen berechnet, die die potenziellen Angriffsszenarien in Bezug auf ihre Kommunikationsbeziehungen, ihre Manifestation in den Datenmerkmalen und ihr zeitliches Verhalten charakterisieren und darstellen. Über die Erkennung bekannter Angriffe hinaus kann der Vergleich abgeleiteter Signaturen genutzt werden, um Ähnlichkeiten zwischen noch unbekannten und neuen Angriffsmustern zu finden. Die in zwei Teile gegliederte Evaluierung nutzt Angriffsszenarien aus den weit verbreiteten und beliebten CICIDS2017- und CSE-CIC-IDS2018-Datensätzen. Zunächst wird die Streaming-Fähigkeit von AC anhand von CICIDS2017 evaluiert und mit einem neuartigen graph-basierten Offline-Alarmkorrelationsalgorithmus namens GAC verglichen, der das Potenzial hat, mit den Ergebnissen von anomaliebasierten IDS umzugehen. Zweitens werden die drei Typen von Signaturen anhand von Angriffsszenarien in den Datensätzen berechnet und miteinander verglichen. Die Diskussion der Ergebnisse zeigt, dass SOAAPR mit GAC in Bezug auf die AC-Fähigkeit konkurrieren kann und diese bezüglich der Verarbeitungszeit deutlich übertrifft. Darüber hinaus scheinen in den meisten Fällen alle drei Arten von Signaturen Angriffsszenarien so zuverlässig zu charakterisieren, dass ähnliche Szenarien in Gruppen zusammengefasst werden.

Bislang wurden der kryptografische Schutz und die Angriffserkennung für Netzwerksicherheit als getrennte Disziplinen betrachtet. Daher wird als letzter Hauptbeitrag ein Algorithmus für *Entkoppelte Nachrichtenauthentifizierungscodes* - **Uncoupled MAC** - vorgestellt, der eine Brücke zwischen diesen beiden Disziplinen schlägt. Er sichert die Netzwerkkommunikation (Authentizität und Integrität) durch ein kryptographisches Verfahren mit Layer-2-Unterstützung über entkoppelte Nachrichtenauthentifizierungscodes, bietet aber als Nebeneffekt auch eine IDS-Funktionalität, die Alarme aufgrund der Verletzung von Uncoupled MAC-Werten erzeugt. Durch eine neuartige Erweiterung der Selbstregulierung passt der Algorithmus seine Sampling-Parameter auf der Grundlage der Erkennung bösartiger Aktionen an. Das Sampling hat sich als nützlich erwiesen, um bösartige Aktivitäten in einer überschaubaren Menge von SD zu erkennen, insbesondere in Systemen, in denen Ressourcen wertvolle Güter sind und im Gegensatz zu der ständig wachsenden Menge an Netzwerkverkehr stehen. Die Auswertung in einer virtualisierten Umgebung zeigt deutlich, dass die Erkennungsrate über die Laufzeit für verschiedene Angriffsszenarien ansteigt. Diese decken sogar Szenarien ab, in denen intelligente Angreifer versuchen, die Nachteile des Sampling auszunutzen.

Schlüsselwörter: Netzwerksicherheit, Maschinelles Lernen, Angriffserkennung, Ausreißererkennung, Streaming-Daten, Online Lernen, Unüberwachtes Lernen, Featureauswahl, Alarmanalyse, Alarmkorrelation, Angriffsszenario, Nachrichtenauthentifizierung, Selbstregulierung

Credits

Parts of this dissertation have previously appeared or are still under consideration in the following scientific publications. In all of those articles I am the first author, conceived the original idea as well as their conceptualization and contributed most of the work, including the writing of the manuscript. Subjective writing is used throughout this thesis (*we*-perspective) to reflect me and my co-authors.

Other parts have previously been published in a report capturing the state of the art & concepts of the doctoral thesis as part of my state doctoral examination. During the years of this study program, I also contributed to other related areas in the cyber security field and published papers written jointly with several collaborators as listed in Appendix A.

- Michael Heigl, *Network Communication Protection Based on Anomaly Detection's Incident Handling - State of the Art & Concepts of Doctoral Thesis*, Technical Report No. DCSE/TR-2019-03[1], University of West Bohemia, July 2019 **- Chapter 1 and Chapter 2**

- Michael Heigl, Laurin Doerr, Nicolas Tiefnig, Dalibor Fiala, Martin Schramm, *A Resource-Preserving Self-Regulating Uncoupled MAC Algorithm to be Applied in Incident Detection*, Computers & Security, vol. 85, pp. 270-287, May 2019, doi:10.1016/j.cose.2019.05.010 **- IF: 3.579 - Chapter 6**

- Michael Heigl, Kumar Anand, Andreas Urmann, Dalibor Fiala, Martin Schramm, Robert Hable, *On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data*, Electronics (Basel), 10(13), 1534, June 2021, doi:10.3390/electronics10131534 **- IF: 2.397 - Chapter 4**

- Michael Heigl, Enrico Weigelt, Andreas Urmann, Dalibor Fiala, Martin Schramm, *Exploiting the Outcome of Outlier Detection for Novel Attack Pattern Recognition on Streaming Data*, Electronics (Basel), 10(17), 2160, September 2021, doi:10.3390/electronics10172160 **- IF: 2.397 - Chapter 5**

- Michael Heigl, Enrico Weigelt, Dalibor Fiala, Martin Schramm, *Unsupervised Feature Selection for Outlier Detection on Streaming Data to Enhance Network Security*, Computers & Security, *revised version submitted 02 September 2021* **- IF: 4.438 - Chapter 3**

Most parts of this report were elaborated in the context of the research projects *Decentralized Anomaly Detection* (DecADe) and *New Multi-Layer Platforms for Security and Safety-Relevant Automated Driving Functions* (MLPaSSAD). Both projects have been supported by the German Federal Ministry of Education and Research (BMBF) under grant code 16KIS0539 (DecADe) and 13FH645IB6 (MLPaSSAD). This research work was also partially supported by the Ministry of Education, Youth and Sports of the Czech Republic under grant No. LO1506.

[1]`http://www.kiv.zcu.cz/en/research/publications/technical-reports/` (accessed on 04 September 2021)

Acknowledgements

Helen Keller's quote - *Alone we can do so little; together we can do so much.* - hits the nail on the head. Over the last couple of years, many people in my life have supported me whether from an intellectual, financial or personal standpoint. This support was invaluable on my journey achieving the Ph.D. degree. I faced a lot of challenges, both in work and life, but supportive individuals gave me the confidence to overcome these hurdles. Hence, this is an attempt at acknowledging their remarkable help and guidance. Without them, getting to this final stage would have been much more difficult.

My deepest appreciation goes to my supervisor doc. Ing. Dalibor Fiala, Ph.D. at the University of West Bohemia for his great guidance, his valuable feedback at all times, his expertise in the field of data mining and machine learning and his help in overcoming administrative obstacles especially with the Czech higher education bureaucracy. He successfully guided me through the study program to accomplish my research goals. Thank you also for constantly reviewing and validating my results.

A special and heartfelt thank you goes to all of my (former) colleagues of the ProtectIT institute at the Deggendorf Institute of Technology. In particular, those that supported me over all these years, contributed to my research work, both by realizing my conceptions and being co-authors of scientific publications. Thank you very much for all the scientific and technical discussions we had, for assisting me with all the implementation work and for the fun "team building" time. I also want to show gratitude to the wife of a highly respected colleague, a native English speaker, for all the laborious proofreading.

I am deeply grateful to Prof. Dr. Martin Schramm for enabling me to pursue a Ph.D. degree in cooperation with the University of West Bohemia. His comments, suggestions and reviews were always of inestimable value and made an enormous contribution to my work in the field of cyber security. His expertise in the field of embedded cryptography were particularly beneficial. I also want to express my gratitude to the Vice President of Applied Research and Technology Transfer, Prof. Dr.-Ing. Andreas Grzemba, of the Deggendorf Institute of Technology for granting me one of the coveted doctoral researcher positions and successfully guided me through my bachelor's and master's degrees. I was fortunate to have him as a mentor since he also gave me an insight into what it is like to work in the cyber security industry by letting me work as a freelancer in his spin-off.

I owe a very important debt to my parents, my sister and my grandparents for giving me a loving and safe childhood, for their moral support, for their warm encouragement throughout my life and their pride in my achievements. My education and academic career was always made a priority in all phases of life, including ups and downs. For this and so much more, I would like to express my deepest gratitude!

After becoming a husband and father, my young family became my constant drive and source of tireless motivation to achieve a long desired dream of mine - the doctoral degree. The birth of my son made me realize that the important things in life are not triumph and money but health and love which is aptly worded in the quote - *The most precious jewels, you'll ever have around your neck, are the arms of your children.* Finally, I would like to thank my wife that you decided to share your heart with me as well as your constant support and sacrifice during the tremendously time-consuming Ph.D. process. Thank you so much for your continuous, unconditional and unparalleled love as well as being such a wonderful mother. I love you both with all my heart!

Vergelt's Gott!

Contents

List of Acronyms xv

List of Figures xviii

List of Tables xix

List of Algorithms xx

List of Equations xx

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 2
  - 1.2.1 Non-Applicability of Artificial Neural Networks . . . 2
  - 1.2.2 Limitations of Intrusion Detection Systems . . . 3
  - 1.2.3 Limitations of Alert Correlation . . . 5
  - 1.2.4 Challenges of Streaming Data . . . 6
- 1.3 Research Objectives . . . 7
- 1.4 Thesis Statement and Contributions . . . 9
- 1.5 Organization of the Thesis . . . 14

2 Background **15**
- 2.1 Incident Detection . . . 15
  - 2.1.1 Taxonomy of IDSs . . . 15
  - 2.1.2 Aspects of Machine Learning . . . 21
  - 2.1.3 Outlier Detection . . . 27
  - 2.1.4 IDS Evaluation Metrics . . . 34
- 2.2 Incident Analysis . . . 37
  - 2.2.1 Pre-Processing . . . 38
  - 2.2.2 Processing . . . 41
  - 2.2.3 Post-Processing . . . 48
- 2.3 Incident Response . . . 60
  - 2.3.1 Taxonomy of Intrusion Response Systems . . . 60
  - 2.3.2 Intrusion Response Representation . . . 66
  - 2.3.3 Possible Response Measures . . . 68

3 Unsupervised Feature Selection for Outlier Detection on Streaming Data to Enhance Network Security **72**
- 3.1 Requirements Engineering and Comparison with Related Work . . . 72

3.1.1 Requirements with Respect to Feature Selection for Outlier Detection on Streaming Data . . . 72
3.1.2 Feature Selection for Streaming Data . . . 74
3.1.3 Feature Selection for Outlier Detection . . . 76
3.2 Unsupervised Feature Selection for Streaming Outlier Detection . . . 79
3.2.1 Operation Principle . . . 79
3.2.2 Operation Modes . . . 80
3.2.3 Model for Scoring and Clustering Features . . . 82
3.3 Evaluation . . . 85
3.3.1 Test Environment . . . 85
3.3.2 Data Source . . . 87
3.3.3 Evaluation Methodology . . . 88
3.4 Discussion of Results . . . 89
3.4.1 Comparison of FSDS, IBFS and UFSSOD with the Best 25% Features 89
3.4.2 Comparison of IBFS and UFSSOD with Different Feature Sets . . . 93
3.4.3 Application of UFSSOD, xStream and Loda Two Hist. in a Streaming Setting . . . 95

4 On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data 99
4.1 Related Work in Online Outlier Detection . . . 99
4.2 Requirement Specification & Validation . . . 102
4.3 Generic PCB-iForest Framework . . . 104
4.3.1 Drift Detection Method . . . 105
4.3.2 Performance Counter-Based Scoring . . . 106
4.3.3 Base Learner . . . 108
4.4 Experimental Evaluation . . . 109
4.4.1 Methodology & Settings . . . 110
4.4.2 Data Sources . . . 111
4.4.3 Evaluation Criteria . . . 113
4.5 Discussion of Results . . . 113
4.5.1 NDKSWIN Drift Detection . . . 114
4.5.2 Competitors-Based HSFS . . . 115
4.5.3 Multi-Disciplinary ODDS . . . 116
4.5.4 Security-Related UNSW-NB15 . . . 121

5 Exploiting the Outcome of Outlier Detection for Novel Attack Pattern Recognition on Streaming Data 123
5.1 Related Work for Streaming Alert Correlation and Outlier Detection . . . 123
5.1.1 Alert Correlation for Outlier Detection . . . 123
5.1.2 Streaming Alert Correlation . . . 125
5.1.3 Delimitation from SOAAPR . . . 126
5.2 Streaming Outlier Analysis and Attack Pattern Recognition . . . 130
5.2.1 Operation Principle . . . 130
5.2.2 Alert Generation & Preparation . . . 131
5.2.3 Streaming Alert Correlation & Clustering . . . 133
5.2.4 Signature Generation & Sharing . . . 140
5.3 Experimental Evaluation . . . 145
5.3.1 Methodology & Settings . . . 145

5.3.2 Data Source 146
5.3.3 Evaluation Criteria 147
5.4 Discussion of Results 148
5.4.1 SOAAPR Clustering 148
5.4.2 SOAAPR Signaturing 151

6 A Resource-Preserving Self-Regulating Uncoupled MAC Algorithm to be Applied in Incident Detection 162
6.1 Incident Detection Mechanisms 162
6.1.1 Adaptive Intrusion Detection Systems 162
6.1.2 Cryptographic Mechanisms for Incident Detection 164
6.2 Uncoupled MAC Algorithm Improvements 168
6.2.1 Master/Slave Negotiation 168
6.2.2 Integration in Networked Devices 169
6.2.3 Synchronization of MAC Phase Start 169
6.2.4 Static Communication Mode 170
6.2.5 IDMEF Extension 170
6.3 Self-Regulation Algorithm 171
6.3.1 Number of Packets n per MAC Phase 171
6.3.2 Waiting Duration α in the Idle Phase 172
6.3.3 Formula Verification 173
6.3.4 Algorithm Notation 173
6.4 Evaluation 174
6.4.1 Virtualized Environment 175
6.4.2 Attack Scenarios 176
6.4.3 Attacker Implementation 177
6.4.4 Evaluation Metrics 178
6.5 Measurement Results 179
6.5.1 Continuous Injection 179
6.5.2 Stochastic Injection 180
6.5.3 Bandwidth Low Injection 180
6.5.4 Weak Spot Injection 182
6.5.5 Uncoupled MAC Overhead 183

7 Conclusion 186
7.1 Summary of the Work 186
7.2 Revising the Research Questions 187
7.3 Future Work & Research Perspectives 190

Bibliography 194

Appendices I

Appendix A Author's Publications II

Appendix B Contents of the Enclosed SD-Card V

List of Acronyms

AC Alert Correlation iii, v, vii, 6–8, 12, 15, 38, 41–43, 45–48, 51, 56, 65, 123–125, 128–130, 137, 146, 147, 150, 187, 188

ACG Alert Correlation Graph 46

AI Artificial Intelligence 1, 3, 49, 186

ANDERS Anomaly-based Incident Detection and Response System xvi, 7–9, 186, 193

ANN Artificial Neural Networks 3, 19, 20, 25, 32, 34, 45, 46, 57, 108

APT Advanced Persistent Threat 4, 49

AUC Area Under the ROC Curve 34, 35, 188

AutoML Automated ML 23

CAN Controller Area Network 33, 34, 52, 59

CASH Combined Algorithm Selection and Hyperparameter 23

CVE Common Vulnerabilities and Exposures 51, 56, 68

DAG Directed Acyclic Graph 23, 45, 46

ECU Electronic Control Units 56, 59, 60

EIF Extended Isolation Forest 104, 108, 110

FN False Negative 5, 35, 37, 48, 126, 129, 131–133, 140, 141, 144, 150, 151, 189, 192

FP False Positive 5, 35, 37, 38, 41, 44, 48, 57, 65, 126, 128–130, 132, 133, 140, 141, 144, 150, 151, 189, 192

FPR False Positive Rate 35, 36, 113

FS Feature Selection ii, iv, vi, xvi–xviii, 8, 10, 11, 14, 24–26, 72–82, 86, 88, 89, 92, 186, 187, 191

GAC Graph-based Alert Correlation iii, v, vii, xvii, xix, 12, 123, 124, 127, 139, 145–152, 189, 192

HIDS Host-based IDS 16, 60

HMAC Keyed-Hash Message Authentication Code 164, 165, 171, 174, 176, 184, 185, 193

IBFS Isolation-Based Feature Selection xvii–xx, 11, 72, 78, 82, 85, 88–95, 109, 110, 187

IDEA Intrusion Detection Extensible Alert 12, 40, 171, 192

IDIP Intruder Detection and Isolation Protocol 39, 64, 67

IDMEF Intrusion Detection Message Exchange Format 20, 34, 38–40, 51, 65–67, 170, 171, 174, 192

IDS Intrusion Detection System ii–vii, 2–8, 12, 13, 15–20, 24, 25, 30, 32–39, 41, 42, 46–48, 51, 52, 55–60, 63–66, 69, 71, 72, 76, 87, 103, 111, 123–130, 142, 144–146, 162–164, 167, 168, 170, 171, 174, 178, 179, 186–190, 192, 193

IDXP Intrusion Detection eXchange Protocol 39, 40

iForest Isolation Forest ii, iv, vi, xviii–xx, 11, 12, 15, 28–32, 77, 78, 80–82, 85, 86, 98–101, 104–106, 108–111, 115, 116

IoT Internet of Things 1, 4, 5, 9, 15, 124, 186, 193

IPS Intrusion Prevention System 50, 51, 60, 61

IRMEF Intrusion Response Message Exchange Format 67

IRS Intrusion Response System 60–62, 66, 67

IT Information Technology 1, 5, 7, 34, 39, 52, 186

Loda Lightweight on-line detector of anomalies xvii–xx, 15, 28–32, 34, 80–83, 86, 88, 89, 95–100, 104, 106, 108, 109, 116–119, 121, 122, 131, 132, 187, 188, 191–193

MAC Message Authentication Code xviii, xx, 13, 165–174, 176–183, 185, 190, 193

ML Machine Learning 3–5, 9, 15, 19, 21–28, 32–35, 41, 42, 46, 124, 144, 147, 186, 193

NETCONF Network Configuration Protocol 67

NFV Network Function Virtualization 71

NIDS Network-based IDS 16, 17, 19–21, 28, 36, 58, 60, 69

OCSVM One-Class SVM 34

OD Outlier Detection ii–vii, xvi, xvii, xix, 3, 6–12, 14, 15, 19, 22, 27–30, 72–83, 86–90, 92, 93, 99–104, 108–113, 116–121, 123, 126–131, 133, 134, 137, 140–143, 145–147, 153, 186–189, 191–193

ODDS Outlier Detection DataSets xviii, xix, 87, 111, 115, 119, 188

ODE Ordinary Differential Equation 53, 54

PCA Principal Component Analysis 24, 25, 101, 105, 108

PCB-iForest Performance Counter-Based iForest ii–vii, xvii, xix, xx, 11, 12, 14, 31, 99, 101, 104–111, 113–122, 186, 188, 190–193

RC Research Contribution 10–15, 187, 188, 190

RF Random Forests 31, 124, 145, 153, 192

ROC Receiver Operating Characteristic 34–36

RP Random Projection 24, 25, 31, 88, 100, 105, 110

RQ Research Question 8, 10, 14, 187, 188, 190

SD Streaming Data ii–vii, xvi, xvii, xix, 6, 7, 10–12, 14, 27, 28, 31, 72–79, 81, 82, 86, 88, 89, 99–106, 108, 109, 112, 116, 123, 186, 187, 191

SDE Stochastic Differential Equation 53–55

SDN Software-Defined Networking 2, 8, 9, 15, 69–71, 163, 168, 193

SEIR Susceptible-Exposed-Infectious-Recovered 53, 54

SHAP SHapley Additive exPlanations 31, 145, 192

SI Susceptible-Infected 53, 58

SIEM Security Information and Event Management 38, 48, 49, 51, 170

SIR Susceptible-Infected-Recovered 53, 58

SNMP Simple Network Management Protocol 39, 67

SOAAPR Streaming Outlier Analysis and Attack Pattern Recognition iii, v, vii, xvii, xix, 12, 14, 123, 126–131, 133–135, 137, 140, 141, 145–151, 187, 189, 190, 192

SOC Security Operations Center 38, 48

STIX Structured Threat Information eXpression 12, 13, 130, 144

SVM Support Vector Machine 26, 33, 46, 51, 123, 124

TN True Negative 35, 133, 134, 150

TP True Positive 35, 37, 56, 128, 130, 133, 135, 150

TPR True Positive Rate 35, 36, 113

UFSSOD Unsupervised Feature Selection for Streaming Outlier Detection ii, iv, vi, xvi–xx, 10, 11, 14, 72, 79–85, 87–98, 186, 187, 190–192

Uncoupled MAC Uncoupled Message Authentication Code iii, v, vii, xviii, xx, 13, 14, 69, 162, 166–185, 187, 190, 192, 193

XML eXtensilble Markup Language 20, 40, 67, 170

List of Figures

1.1 Cyber defense life cycle 1
1.2 Incident handling process 2
1.3 Architecture of a generic ANDERS component including the four research questions of this thesis 8
1.4 Possible ANDERS application use case of malicious activity propagation in an SDN-based vehicular ad-hoc network 9
1.5 High level overview of improved outlier detection within the thesis leading to four major research contributions 10

2.1 Taxonomy/classification of IDSs 16
2.2 Taxonomy of detection approaches for IDSs 18
2.3 A typical Machine Learning Pipeline/Workflow 21
2.4 (a) Isolating an outlier, (b) Representation of a tree model 29
2.5 Taxonomy of evaluation measures 35
2.6 Example of an ROC-curve 36
2.7 Taxonomy of alert analysis fields 38
2.8 Application domains of exchange formats 39
2.9 Evaluation summary of exchange formats 40
2.10 Evaluation summary of exchange protocols 40
2.11 Different alert correlation architectures 43
2.12 Taxonomy of alert correlation techniques 44
2.13 Framework of real-time network attack intention recognition 50
2.14 Operation of a generic early warning system 51
2.15 Different types of behavior in compartmental models 53
2.16 Simulation of an SDE for malware propagation 55
2.17 Illustration of three categories of observation in networks 58
2.18 Taxonomy of source identification methods 58
2.19 Root cause identification framework 59
2.20 Taxonomy of intrusion response systems 62
2.21 Challenges for IRSs 63
2.22 Response selection process of REASSESS 66
2.23 Intrusion response message format - IRMEF 68
2.24 Intrusion response message format - IR-Message 69
2.25 Different intrusion response levels 70
2.26 List of common response measures 71

3.1 Block diagram and operation principle of UFSSOD 80
3.2 Conceptualization of module interaction for unsupervised FS for OD on SD (sequential approach) 81

3.3 Conceptualization of module interaction for unsupervised FS for OD on SD (parallel approach) 82
3.4 Visualization aid of ufssod_clustering with four exemplary clusters and semicircles around the cluster centroids 84
3.5 Flowchart of the evaluation measurements 88
3.6 Exemplary $F1$ plots for badly-performing IBFS and UFSSOD using top 25% feature subsets 91
3.7 Exemplary $F1$ plots for well-performing IBFS and UFSSOD using top 25% feature subsets 91
3.8 Exemplary plots for feature scores $\bar{s}_{fi(t)}$ of IBFS (a,c) and UFSSOD (b,d) applied on *musk* (a,b) and *satellite* (c,d) 92
3.9 Exemplary plots for feature scores $\bar{s}_{fi(t)}$ of IBFS (a) and UFSSOD (b) applied on *optdigits* 94
3.10 $F1/avg\_t$ and avg_t results for *optdigits* 94
3.11 $F1/avg\_t$ results for well-performing data sets *mnist* and *wbc* 95
3.12 $F1/avg\_t$ results for badly-performing data sets *cardio* and *optdigits* . . . 96
3.13 $F1/avg\_t$ results for well-performing data sets *mnist* and *satellite* 97

4.1 The workflow of the PCB-iForest incremental learning framework 105
4.2 Exemplary visualization of NDKSWIN gradual and abrupt drift detection with two different hyperparameter settings 115
4.3 $F1/avg\_t$ results for well-performing PCB-iForest$_{\mathrm{IBFS}}$ on data sets *mnist* (a) and *optdigits* (b) 117
4.4 avg_t results for the data sets *ionosphere* (a) and *wbc* (b) 118
4.5 Exemplary plots for feature scores $\bar{\boldsymbol{s}}_f$ on high-dimensional *arrhythmia* (a) and *musk* (b) including 25%, 50% and 75% subsets 119
4.6 $F1$ results for the high-dimensional data sets *arrhythmia* (a) and *musk* (b) utilizing different feature sets 120

5.1 Flowchart and Operation Principle of Streaming Outlier Analysis for Attack Pattern Recognition 131
5.2 Exemplary Loda outlier score values including the mean and median 132
5.3 Exemplary scenario of evolving alert clusters for two different observation times (a) and (b) 138
5.4 Generation and comparison process of the signature sig_{com} 141
5.5 Generation and comparison process of the signature sig_{attr} 143
5.6 Generation and comparison process of the signature sig_{temp} 143
5.7 Sunburst diagrams for the clustering performance of (a) GAC on FTP and SSH Brute Force, (b) SOAAPR on the web attacks, (c) GAC and (d) SOAAPR on the DoS-attacks 150
5.8 Hierarchical clustering of similarities between CICIDS2017 attack scenarios based on sig_{com} 152
5.9 Two exemplary sig_{attr} using the same set of 78 features for the attack scenarios (a) *Infiltration* of CICIDS2017 and (b) *Brute-Force-Web-0* in CSE-CIC-IDS2018 154
5.10 Dependency of γ on the Bhattacharyya distance (similarity) of two highly similar (a) and dissimilar (b) attack scenarios 155
5.11 Hierarchical clustering of similarities between a selection of CICIDS2017 and CSE-CIC-IDS2018 attack scenarios based on sig_{attr} for $\gamma = 30$ 155

5.12 Comparison of sig_{attr} for (a) SQL-Injection-0 of CSE-CIC-IDS2018 with Web Attack - Sql Injection of CICIDS2017 and (b) Brute-Force-Web-1 of CSE-CIC-IDS2018 with Web Attack - Brute Force of CICIDS2017 for $\gamma = 30$ 156
5.13 Smoothed γ-dependency-curves using Gaussian Filtering for a selection of similar and dissimilar attack scenarios 157
5.14 Hierarchical clustering of similarities between CICIDS2017 and CSE-CIC-IDS2018 attack scenarios based on sig_{temp} 159
5.15 Comparison of Δt histograms of four distinct *Infiltration* attack scenarios (a)-(f) present in CSE-CIC-IDS2018 160
5.16 Comparison of Δt histograms of four distinct web attack scenarios - *XSS* (a)-(b) and *Brute Force* (c)-(d) - present in CICIDS2017 and CSE-CIC-IDS2018 160

6.1 Network scenario of the proposed concept 166
6.2 Combination of methods 168
6.3 Integration of Uncoupled MAC operation in networked devices 169
6.4 Timing diagram for a synchronized start of a MAC Phase 170
6.5 Self-regulating sampling approach for Uncoupled MAC 172
6.6 Verification of the self-regulation formulas 173
6.7 Structure of the virtual evaluation environment 175
6.8 Attacker bandwidth measurement and injected packets in detected bandwidth lows 178
6.9 Confusion Matrix parameters for some Uncoupled MAC phase k 178
6.10 Detection rate - continuous packet injection 179
6.11 Detection rate - stochastic attack mode 180
6.12 Packet detection per phase - single attack mode 181
6.13 Packet detection per phase - burst attack mode 182
6.14 Detection rate - weak spot injection 183
6.15 Bandwidth overhead in presence of an attacker 184
6.16 CPU (left) and memory (right) utilization of the Uncoupled MAC process . 184

List of Tables

2.1 Time/space complexity of iForest and Loda 31
2.2 Confusion matrix for IDS evaluation 35
2.3 Response and recovery action classification 68

3.1 Comparison of existing FS work for SD 76
3.2 Comparison of existing FS work for OD 78
3.3 Characteristics of the utilized and partially adapted data sets from ODDS 87
3.4 $F1/avg\_t$ results for FSDS (different k), IBFS and UFSSOD 90
3.5 $F1$ results for IBFS and UFSSOD for different γ (feature subsets) 93

3.6 Individual classifier performance in terms of the percentage increase/decrease of avg_t and $F1$ applying ckeans_ufssod on IBFS and UFSSOD compared to full dimension on *wbc* data set 95
3.7 $F1$ results for UFSSOD using different γ (feature subsets) for data sets with ID i in streaming setting with xStream and $\text{Loda}_{\text{Two Hist.}}$ 96
3.8 Performance of xStream and $\text{Loda}_{\text{Two Hist.}}$ in terms of the percentage increase/decrease of avg_t and $F1$ applying UFSSOD in a streaming setting compared to full dimension 98

4.1 Comparison of existing OD work for SD with the requirements specified . . 104
4.2 Characteristics of the partially truncated data sets from ODDS 111
4.3 Characteristics of the four preprocessed CSV files from the UNSW-NB15 data set 113
4.4 Classification performance of different iForest-based competitors aggregated from their respective original work with $\text{PCB-iForest}_{\text{EIF}}$ and $\text{PCB-iForest}_{\text{IBFS}}$ 116
4.5 $F1$ results for different online OD algorithms on data sets with ID i 117
4.6 $F1/avg\_t$ results for different online OD algorithms on data sets with ID i 118
4.7 $F1/avg\_t$ results averaged for each online OD using different feature sets for data sets with ID i 120
4.8 Individual classifier performance in terms of the percentage increase/decrease of avg_t and $F1$ when applying a feature subset compared to full dimension on the *arrhythmia* and *musk* data set 121
4.9 $F1$ and avg_t results for different online OD algorithms on the four preprocessed CSV files from the UNSW-NB15 data set 121

5.1 Exemplary meta-alert generation based on five online OD algorithms classifying five data instances (3x normal, 2x abnormal) 134
5.2 The 14 different attacks with respective characteristics of the CICIDS2017 data sets 135
5.3 Proposed comparison functions f_k to compute the similarity between alert $\boldsymbol{a}_t$ and $\boldsymbol{a}_j$ utilizing the alert attributes $\{ip_{src}, ip_{dst}, port_{src}, port_{dst}\}$ 139
5.4 Clustering performance results of SOAAPR and GAC on 11 attack scenarios of the CICIDS2017 data sets 148
5.5 Upper triangular matrix of sig_{com} similarities between CICIDS2017 attack scenarios 151
5.6 A selection of attack scenarios with respective characteristics of the CSE-CIC-IDS2018 data set 153
5.7 Average processing time to generate sig_{attr} for a selection of attack scenarios of CICIDS2017 and CSE-CIC-IDS2018 158
5.8 Average processing time to generate sig_{temp} for a selection of attack scenarios of CICIDS2017 161
5.9 Average processing time to generate sig_{temp} for a selection of attack scenarios of CSE-CIC-IDS2018 161

List of Algorithms

1 High-level operation principle of UFSSOD 80
2 Scoring functionality of UFSSOD - ufssod_scoring() 83
3 Feature clustering of UFSSOD - ufssod_clustering() 85

4 Operation Principle of PCB-iForest 107
5 Feature Scoring in PCB-iForest utilizing IBFS 110

6 Operation Principle of *Streaming Alert Correlation / Clustering* 136
7 Monitoring of *Trigger Signature Generation* and *Discard Alerts & Clusters* . 138

8 Calculation of n for the next phase $k+1$ 174
9 Calculation of α for the next phase $k+1$ 174

List of Equations

2.1 Machine Learning Pipeline - Performance 23
2.2 Machine Learning Pipeline - Loss Minimization 23
2.3 iForest - $score(x, \psi)$ 30
2.4 iForest - $c(\psi)$ 30
2.5 iForest - $H(n)$ 30
2.6 Loda - $f(x)$ 30
2.7 Loda - $\hat{p}(x)$ 30
2.8 Similarity function - $Sim(X^i, Y)$ 44
2.9 Infection function - $dI(t)$ 54
5.1 Outlier score normalization formula 132
5.2 Similarity function between an alert $\boldsymbol{a}_t$ and a cluster C_i 139
5.3 Alert similarity between two alerts $\boldsymbol{a}_t$ and $\boldsymbol{a}_j$ 139
5.4 Similarity between two network motifs. 142
5.5 Similarity between two signatures sig_{attr} using the Bhattacharyya distance measure 142
5.6 Completeness evaluation metric for alert correlation 147
5.7 Soundness evaluation metric for alert correlation 148
5.8 Jaccard evaluation metric for alert correlation 148
6.1 Number of packets to be examined in the next MAC Phase 172
6.2 Percentage of packets to check in the next MAC Phase 172
6.3 Duration of the next Idle Phase 172
6.4 True Positive Rate for efficiency of Uncoupled MAC's detection performance179

1 Introduction

This chapter provides a short introduction to this thesis by firstly explaining its motivation. Further, the problem statement is provided and research questions are formulated. Finally, the chapter gives an outline of this research with four major contributions and a brief organization of the thesis.

1.1 Motivation

The constantly growing number of computer components, the spatial extent of networks, their interaction with their environment (e.g., automotive: "Always and Everywhere On") boosted by trends such as Internet of Things (IoT), Connected Cars, Smart Cities, Industry 4.0, 5G or Software-Defined Everything and the use of new technologies, e.g., Artificial Intelligence (AI), not only lead to the improvement of processes or customer comfort, but also increase system complexity and create new hazard potentials and risks with regard to the information and operational security of these systems. This also offers adversaries a broad spectrum of attack vectors and capabilities such as the usage of cloud or distributed computing, quantum computation for breaking contemporary public-key cryptography [1] or smart AI-powered malware that enables highly sophisticated and stealth attacks. The rise of next generation threats demands "adopting new methods of automated prevention methods" stated by John Samuel, senior vice president and global chief information officer at CGS[1] [2]. Constant monitoring of components, early detection and handling of attacks, and comprehensive continuous assessments of the security level of the overall system are therefore unavoidable for securing future-oriented IT-systems. The cyber defense life cycle [3, 4, 5, 6] shown in Figure 1.1 helps to reduce and better manage cyber risks.

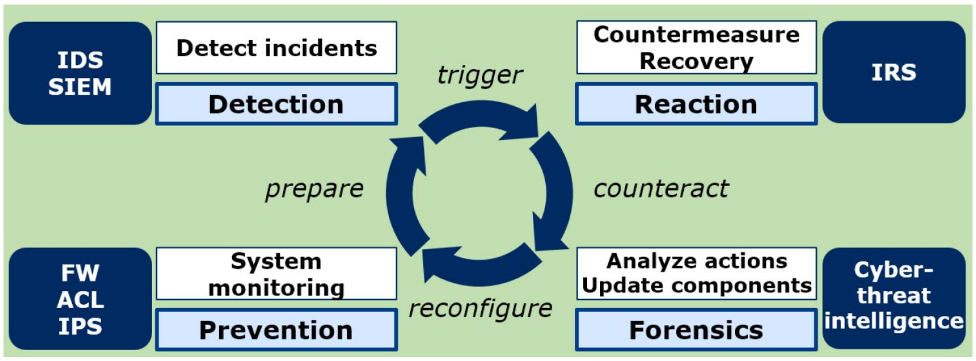


Figure 1.1: Cyber defense life cycle [7].

The cyber defense life cycle specifies the phases of a continuous incident handling process, mainly covering incident detection, incident analysis and incident response, in which malicious behavior, for instance in networks, can be detected, analyzed and appropriate reaction mechanisms can be planned and executed. Figure 1.2 shows the process with

[1]`https://www.cgsinc.com/en` (accessed on 05 September 2021)

the involved components. After detecting anomalous or intrusive behavior through the *Incident Detection* module, the resulting alarms will be logged or visualized within the *Incident Post-Processing* module and analyzed for a later reaction by the *Incident Response* module in the *Incident Analysis/Control* module. Incident analysis techniques, for instance, could comprise performing correlations or similarity functions on raised alerts. Based on the intelligence of this module, a proper reaction to the detected intrusion or anomaly can be planned and executed. Such a reaction could include reconfiguring the network through Software-Defined Networking (SDN) techniques, generating new configurations for firewalls, creating new misuse-based Intrusion Detection System (IDS) rules or adapting the parameters for other incident detection mechanisms. Apart from the aforementioned functions of the *Incident Post-Processing* module, it can further be used for monitoring other incident handling components.

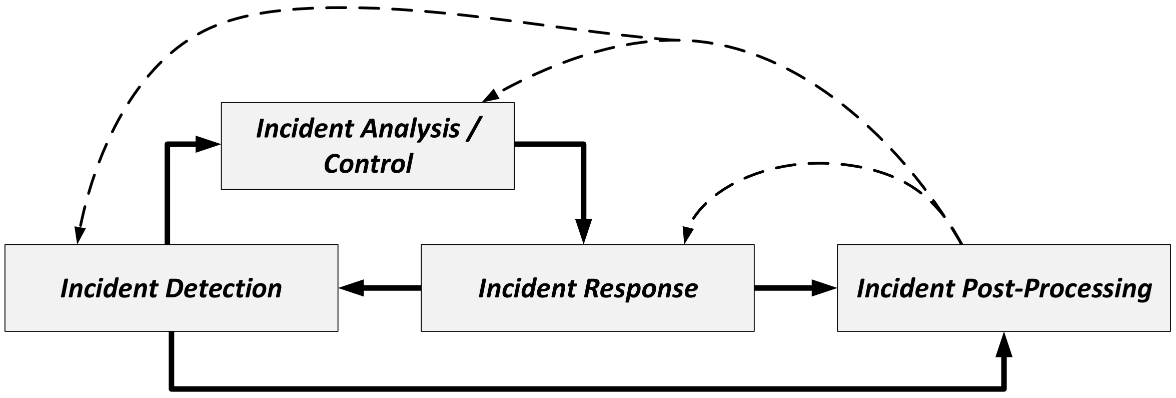


Figure 1.2: Incident handling process.

To defend against various types of cyber attacks, there are two main streams of security solutions for which a number of techniques have been designed [8]: cryptographic schemes and intrusion detection. The former ensures secure communication (confidentiality, integrity, availability, authenticity and non-repudiation of data) in the presence of malicious third-parties, known as adversaries. The latter is an efficient possibility to detect malicious behavior even when cryptography is broken [9]. Nevertheless, for a comprehensive security solution a defense-in-depth concept must include cryptographic procedures as well as IDS components [1]. In particular, security mechanisms are needed that can dynamically as well as flexibly detect and respond to network attacks in an automated manner. Methods, for instance, that use already known attack patterns to merely detect incidents are no longer sufficient in securing network infrastructures.

1.2 Problem Statement

1.2.1 Non-Applicability of Artificial Neural Networks

According to Michael Roytman, chief data scientist at Kenna Security, “automation is the name of the game in security” and “machine learning will help filter out the noise” [2]. Automation not only benefits the detection and analysis of incidents but can also select and execute appropriate responses without human intervention.

Over the past years, the main attention in research focused on the application of network-based IDSs, due to the advent of anomaly-based ones, which gained a huge momentum for their capability of meeting the challenges of the rapid advancements in the field of AI by industry. Many solutions focusing on the incident handling process have adopted Artificial Neural Networks (ANN) including deep learning approaches which are also a branch of AI. A thorough overview of recent literature regarding ANN-usage in the IDS area can be found in [10, 11]. However, ANNs are, according to the authors, usually treated only as a small part of available solutions compared to Machine Learning (ML), a core branch of AI. Although deep learning, often classified as a branch of ML [12], incorporates neural networks within its architecture and both disciplines are deeply intertwined, there are slight differences between them. For instance, the backpropagation algorithm used to train an ANN cannot be used to train deep networks [12]. However, both terms are used interchangeably throughout this thesis since they are not the main focus of this research. Rather, we focus on traditional machine learning models, so-called shallow methods.

There are various reasons why ANN-based approaches, including deep learning, are not addressed in this work. They mostly operate in a supervised fashion, especially in the context of detecting malicious activity, e.g. [13, 14]. Other promising approaches utilize ANNs to automatically learn features in an unsupervised way such that no feature engineering is necessary at all [15, 16]. Anyhow, this approach reinforces the intransparency, which is often criticized with AI, in such a way that it is no longer possible to perform root cause analysis. Deep learning models are especially seen as black boxes [17, 18], as their results are almost uninterpretable, while traditional shallow ML-algorithms have strong interpretability [12]. This is because features that contributed the most to causing the anomaly cannot be identified with ANNs. Moreover, the training process for ANN is considerably more time-consuming compared to ML and is more resource-intensive in terms of computational power and storage capacity. Deep models, in particular, have a higher runtime regarding both training and testing due to their high complexity [12]. This often demands for high-performance systems utilizing parallel computing or GPUs, especially when using deep learning approaches [13, 14, 19, 20, 21]. Although [22] proposed an ANN-based intrusion detection model, which can reduce the computational resources in the training phase and is suitable for real-time deployment, it needs a feature selection and daunting parameter tuning to achieve better performance [23].

With the advent of ML, the detection of novel network-based attacks has been revolutionized. Unsupervised Outlier Detection (OD) algorithms, especially, can help to identify indicators of sophisticated attacks, that are capable of changing their behavior dynamically and autonomously to avoid detection, or they can help to uncover policy violations or noisy instances in data without *a priori* knowledge. As part of anomaly-based IDS methods, they detect abnormal behavior by monitoring significant deviations from what has previously been seen under the viewpoint that outliers, indicators rooted from attacks, are statistically different from normal data with a significantly smaller ratio and do not happen frequently. According to [24], the most dangerous attacks occur only rarely such that OD will seemingly play a crucial role in future network security. In contrast to ANNs, OD methods are (still) more efficient in terms of time and space complexity and allow feature interpretability as well as model transparency.

1.2.2 Limitations of Intrusion Detection Systems

IDSs have crystallized as an essential core component in a holistic security solution by identifying malicious activity in the case of compromising. It monitors the events occurring in a computer system or network in order to analyze them for indications of malicious activity that aims at compromising the common goals of information security. Apart from the distinction of the architecture for IDS, *Host-based IDS* or *Network-based IDS*, in general, two types of detection methods exist. Besides *Anomaly-based* systems, *Misuse-based* ones, also denoted as signature- or knowledge-based, detect attacks based on already known patterns (signatures). Being quite fast and reliable in terms of detecting known attacks, they are not designed to uncover new ones whose signatures are not available, thus misuse-based detection is no longer enough [25]. The anomaly-based detection method creates a model of trusted activity from collected data samples and then compares new behavior with that model. Although it allows for the detection of novel, unknown attacks, it could lead to false negative and false positive alarms, in which malicious/trusted but previously unknown activities could be classified as trusted/malicious.

The ever-increasing and more advanced attack capabilities and strategies pose an enormous challenge to classical network security measures such as IDSs and demand for more sophisticated and comprehensive solutions in the near future. According to a recent study, current IDS research only covers approximately 33% of the threat taxonomy presented in [26]. The actual percentage might even be lower considering the multitude of degrees of freedom for IDSs while encountering real-world threats regarding, e.g., architecture, detection type, operation mode or their scope of functionality. A selection of major challenges and open issues in the field of network communication protection, in particular the detection, analysis and response to cyber attacks (using among others anomaly-based ML algorithms) is listed in the following (cf. [7, 27, 28, 29, 30]):

- insufficient performance of applied detection systems, especially with regard to anomaly-based IDSs, by
  - missing ability in handling a massive amount of throughput, e.g., due to high computational complexity
  - lack of internetworking in a decentralized/distributed fashion; limited scalability, interoperability
  - producing poor predictive values with respect to the binary classification (e.g., too many false positives and false negatives which waste the time of network administrators)
  - no dynamic and adaptive learning working on minimum knowledge for novel attack classification in real-time
  - inefficient pre-processing and poor feature selection
  - limited protection against attacks, e.g., IoT-based Distributed Denial-of-Service;
- novel (zero-day exploits) and highly sophisticated (distributed, stealth, more targeted, long-term, multi-step) attacks, e.g., Advanced Persistent Threats (APTs) cannot reliably be detected;
- increasing amount of alerts (alert flooding) produced by various detection methods with multiple formats cannot reliably be analyzed or today's applied analysis methods are very limited to, e.g., simple correlation methods;
- safe (with respect to functional safety) implementation of static and dynamic response measures whereby a defined and secure system state always remains in order to avoid hazards to life and limb (e.g., attack when driving a vehicle) or high financial losses (industrial plant is paralyzed) even when using cyber defence mechanisms;

- lack of incorporation of anomaly-based IDS outputs into new attack identification, classical systems assume nearly 100% confidence in alerts to map an attack;
- proactive (predictive) and immediate (real-time) response to cyber attacks instead of reactive and "a posteriori" response necessary as skilled IT security staff should exist to monitor IDS operations and respond as needed;
- lack of automation since a high degree of human interaction is often necessary, e.g., system administrators analyze (i.e. verify, correlate) alerts, update signatures, etc.;
- contextual information often not taken into account, e.g., type of systems, applications, users, networks, etc.;
- inadequacy in finding the actual root cause of the incident;
- inherent resource limitations (in terms of memory size, processing speed, energy consumption in the embedded domain) restrict the usage of complex machine learning based methods.

The application of various protocols and devices with different resource requirements (bandwidth, computation), often accompanied in deterministic timing environments by increasing traffic amounts, makes a holistic security solution difficult, especially when outdated legacy components are still active in the network. What is more, the preservation of computing resources, e.g., for IoT-enabled devices, stands in conflict with the resource requirements of detection mechanisms when applied in high-volume networks. To cope with the increasing amount of traffic within networks while reducing large memory and CPU processing requirements, sampling turned out to be a promising scalable data aggregation technology for IDSs since the processing capacity of such systems are typically much smaller than the amount of data to be inspected. The advent of pervasive computing will not only increase the amount of data in cloud systems but also in embedded environments such as for smart sensors in automobiles struggling with increasing data traffic. Adapting detection methods as part of a response measure could be an efficient countermeasure for devices having only limited processing capabilities. The term adaptive in the context of this work describes techniques of sampling and self-regulation, which means that the detection method is adjusted in runtime based on the detection's output.

Nowadays, cryptographic schemes are broadly applied to secure, for instance, the authentication and integrity of communication flows between communicating network parties. These could be used as a potential source for detecting security-related incidents since alerts might be raised if characteristics of the mechanisms fail (e.g., drifting sequence counters, incorrect decryption of messages or mismatching hash digests). However, not enough attention is paid to their interoperation with intrusion detection mechanisms.

1.2.3 Limitations of Alert Correlation

The growth of ML has led to a boost in *anomaly-based* detection methods that create a model of trusted activity from a set of collected data samples and identify malicious activity by analyzing behavior deviations. Although this type of method is predestined to detect novel, yet unknown, attack patterns without requiring *a priori* knowledge, they are accompanied by a high ratio of FP and FN detections. This limits their utilization in real-world scenarios due to their producing an unmanageable amount of alarms, overwhelming for human experts, especially when they are erroneously classified as attacks.

Coping with the increasing data volume challenge and enhancing the detection capability from a more global perspective, the concept of collaboration for IDS emerged, which deploys a multitude of collaboratively communicating IDS with a central Alert Correlation (AC) unit or with each other. AC is a common practice with aims such as false alert reduction, attack pattern identification, root cause detection of attacks or prediction of future attack steps by processing alerts from the heterogeneously applied IDS. Furthermore, they aid to reduce the sheer unmanageable number of events generated (alarm flooding), particularly considering the continuously growing amount of high-dimensional data which can no longer be handled by human analysts.

Yet, limited work exists for AC that is able to efficiently process alerts in a streaming fashion. In addition, most of the existing solutions assume a nearly 100% confidence in alerts mapping an attack, which mainly applies for misuse-based IDS. This means, that generated alerts based on a signature-match contain very precise information with high confidence about the attack or single attack step (of a multi-stage attack) due to the incorporation of existing knowledge. The reason for this is that alerts are typically composed of intrinsic attributes such as timestamp, source and destination IP or port information and an alert type field, often referred to as attack class or intrusion type, which specifies or encodes the attack. In this thesis, we denote an attack scenario as a single-stage attack, while attack campaigns are multi-stage attacks. Similar attack scenarios are called an attack category. Attacks detected by misuse-based IDSs are denoted as intrusion type or class. However, by only detecting such outliers as deviations from data attributes, e.g., a newly occurring IP-address in a computer network, the attack recognition ability is significantly limited since processed data, in the worst case, is only assigned a simple label of normal or abnormal. Since alerts consist of various attributes, further information must be gained from OD methods to be applied in AC. Nevertheless, due to the missing knowledge of known attacks, the alert type information cannot be provided by OD methods.

1.2.4 Challenges of Streaming Data

The ubiquity of massive continuously generated data streams across multiple domains in different applications poses an enormous challenge to state-of-the-art offline OD algorithms that process data as a batch. Data streams in real-world applications are encountering an evolving nature of data of a huge size, potentially infinite, that is continuously streaming in a record-by-record manner in almost real-time. Therefore, efficient and optimized schemes, in terms of processing time and memory usage, in intelligently designed systems, are required. These should be able to process the time-varying and rapid data streams one-pass at a time, while only a limited number of data records can be accessed. Furthermore, legitimate changes in data can occur over time, called concept drift, which require updates to the model in order to counteract less accurate predictions as time passes. Recently, many OD solutions have been developed that are able to compute anomaly scores while dealing with data streams. Data streams can be subdivided into *Streaming Data* (SD) and *Streaming Features*. In this work, we do not focus on the latter case in which the amount of features changes over time. Moreover, we do not focus on SD in the context of time-series data as it is the object of many other research works, such as [31, 32]. With the ubiquity of SD, however, some online unsupervised OD algorithms have been developed in recent years. These are able to efficiently process the time-varying data streams one-pass at a time while dealing with phenomena such as concept drifts, which require timely model updates to counteract accuracy loss if data changes as time passes. With online OD algorithms, alerts can be generated in a streaming manner and lead to a dynamic, huge, infinite and fast changing alert stream for which conventional offline AC methods are not designed [33]. Thus, apart from the above mentioned challenges with regard to online OD solutions on SD, limited work exists for AC that is able to efficiently process alerts in a streaming fashion. What is more, to the best of our knowledge, no work so far exists that process alerts in an online manner generated from OD mechanisms. Similarly to the statement in [34], when it comes to some critical streaming applications, whereby a fast but less accurate OD model is preferred, we strongly support the claim by [33] that it is more significant to detect an on-going attack in a timely manner than to analyze it afterwards in an offline fashion. Detecting attacks at an early stage significantly reduces damage since, even when applying advanced detection systems, sophisticated attackers can nest undetected for up to 100 days [35].

1.3 Research Objectives

The novel principle of "cyber resilience" goes far beyond pure cyber security and takes a comprehensive approach to protecting IT-infrastructures from cyber attacks by securing and restarting operations after attacks have occurred. IDSs are no longer sufficient in fulfilling the needs as well as the challenges stated above and demand for a more holistic solution. The measures and concepts of cyber security, computer forensics, information security, disaster recovery and business continuity management are components of this approach. Cyber resilience bridges the gap between a desired (semi-)automated incident handling and the topics surrounding detection, prevention, prediction and response apart from the protection via cryptography. The term incident, hereinafter, includes intrusions or failures and is an event that negatively affects the protection goals of a system. An event is an observed or detected change to the normal behavior of a system which typically leads to the generation of an alert. An alert notifies about a particular event or a series of events sent to responsible parties. For a detailed definition of an information security event and an information security incident refer to ISO/IEC 27000 [36]. A further goal is the reaction to incidents based on available knowledge within networks and related incidents (expert knowledge, risk analysis, etc.). This also comprises the output of incident detection systems based on a common format. Especially with regard to anomaly-based detection methods, the output of such algorithms do not provide much context for single anomalies, which makes the identification of attacks difficult. Information from multiple detection mechanisms in conjunction with historical data and expert knowledge creates a detection framework, to prediction appropriate proactive/reactive reactions. In the context of anomaly-based detection mechanisms such reactions could include the proper adjustment of features or parameters in order to reduce false positives or to even prevent highly sophisticated multi-staged attacks.

Most of the state-of-the-art research targets the office IT or cloud environment which has a large amount of resources available. To the best of the authors' knowledge, no system with the focus of this work is aimed to, e.g., automotive environments which poses special requirements, such as resource-saving or SD. Novel attack detection based on the output of anomaly-based detection methods is still an unsatisfactorily solved problem and will be tackled in the thesis. Thus, a generic framework, called **An**omaly-based Incident **De**tection and **R**esponse **S**ystem (**ANDERS**), is proposed, that consists of components that leverage different capabilities in terms of incident detection, analysis and response. An exemplary architecture of an ANDERS component is shown in Figure 1.3.

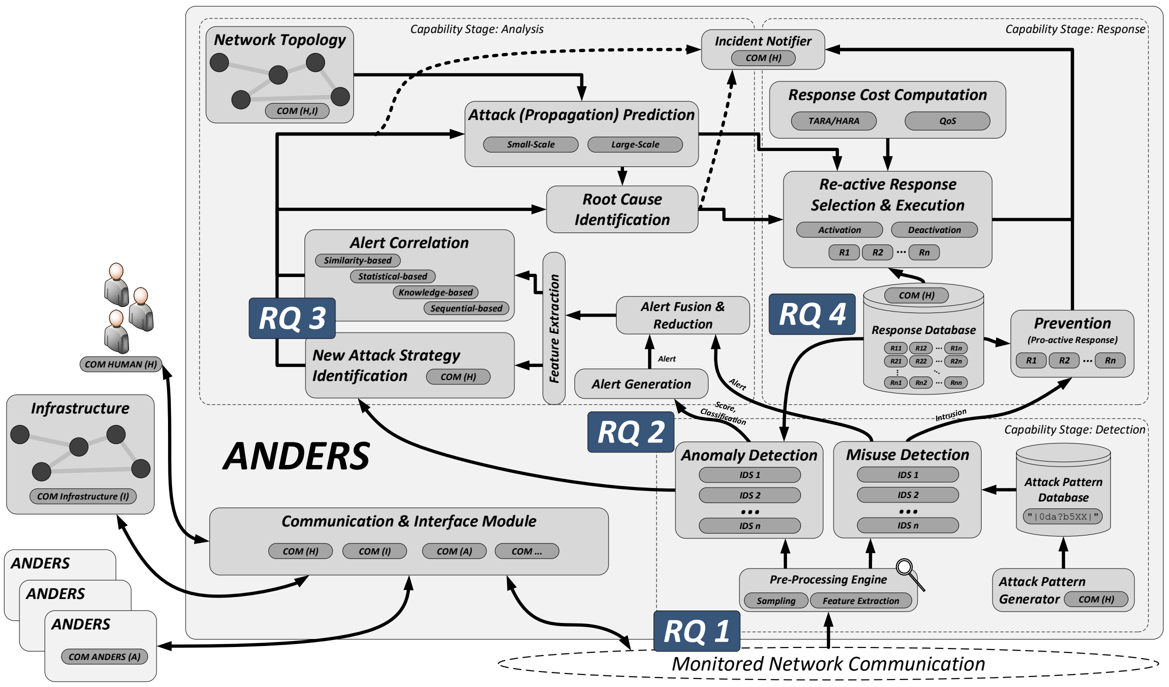


Figure 1.3: Architecture of a generic ANDERS component including the four research questions (RQ 1 - RQ 4) of this thesis.

Some of the components' functionality is already considered state-of-the-art and described in detail in Chapter 2. Among others, this includes the application of misuse-based IDS, or AC, as well as the prediction of the propagation of malicious activity in order to select a suitable response based on these IDS types. Applying methods for root cause identification and response cost, e.g., risk assessment, helps to select appropriate countermeasures. Since ANDERS might have different capability stages, the communication with, e.g., human experts (security administrators), the infrastructure or other ANDERS components is a core feature. However, the main focus with respect to this thesis lies in enhancing computer network security through improved OD for data streams. This includes several disciplines such as Feature Selection (FS) or AC that have not yet been tailored for unsupervised online OD mechanisms. Thus, this thesis aims to answer the following research questions (RQ), also presented in Figure 1.3.

RQ 1: *How can unsupervised feature selection be applied on streaming data for the purpose of outlier detection?*
RQ 2: *How can a flexible framework for unsupervised online outlier detection be designed to provide an online scoring functionality for feature importance?*
RQ 3: *How can the output of online outlier detection mechanisms be exploited to characterize and compare novel attack patterns?*
RQ 4: *How can a cryptographic scheme be leveraged to function as a detection mechanism and have its feedback be incorporated in improving performance over runtime as part of response functionality?*

A resulting prototype system might be embedded in an SDN-infrastructure, similar to [37]. It allows an immediate and scalable implementation of the next generation incident handling framework in the network infrastructure by integrating an ANDERS component directly in the network forwarding elements as a kind of software function for detection, analysis and response. SDN is a key future technology. Not only can it be used for future in-vehicle network communication, e.g., [38, 39, 40], or for vehicular ad-hoc networks [41, 42, 43, 44, 45, 46], but it can also be used in combination with other trends or benefiting from them such as for IoT, 5G, or Industry 4.0 [47, 48, 49]. A possible use case for the application of ANDERS components in a hierarchical SDN-based vehicular ad-hoc network communication is shown in Figure 1.4 (cf. mist-fog-cloud hierarchy [47, 50]). Here, a novel malicious activity propagating over various network hops is detected by a low-level ANDERS component, utilizing anomaly-based methods, which provides alert information to the next ANDERS components in the hierarchical topology. These then perform attack strategy identification, predicting the propagation and identifying the root cause in order for, at the latest, an ANDERS component close to the victim to be able to respond to the attack. It is desired that the ANDERS framework mitigates malicious activity as close as possible to the root cause.

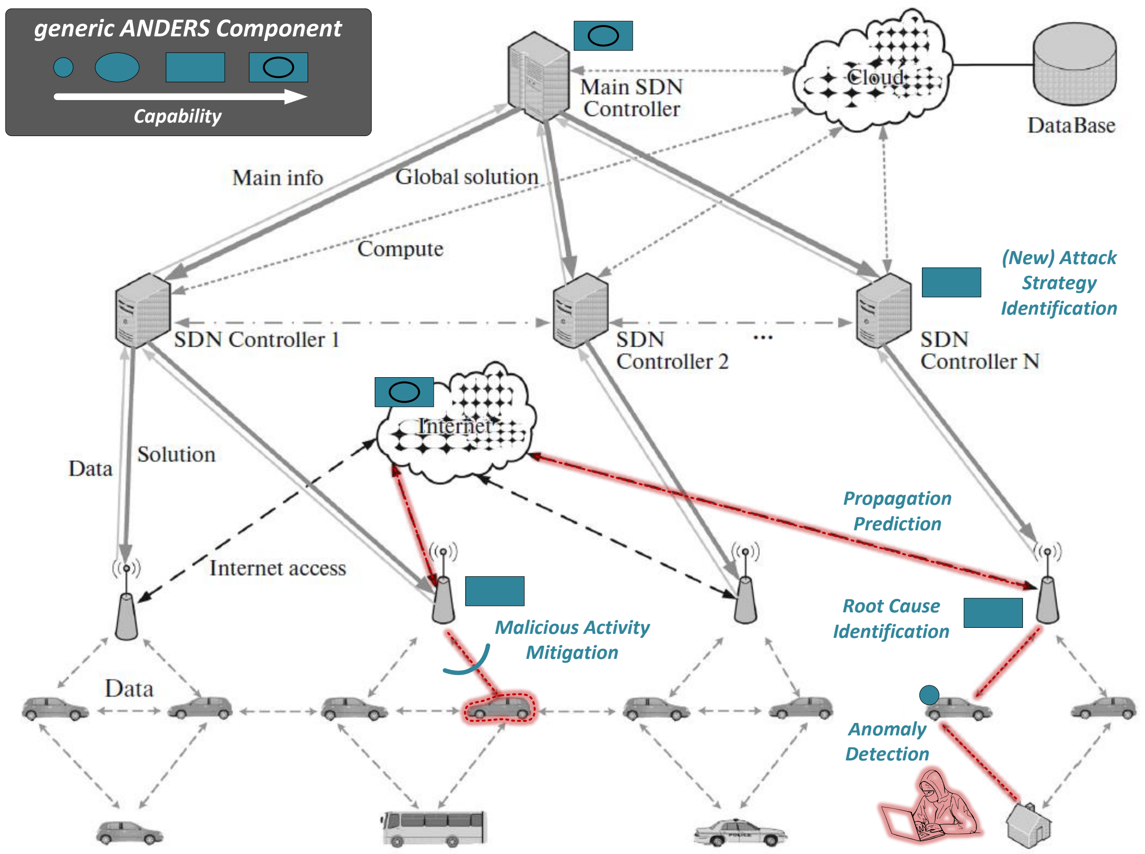


Figure 1.4: Possible ANDERS use case of malicious activity propagation in an SDN-based vehicular ad-hoc network (cf. [46]).

1.4 Thesis Statement and Contributions

The permeating application of ML has favored the detection of novel sophisticated network-based attacks changing their behavior dynamically and autonomously. In particular, unsupervised OD algorithms can help uncover policy violations or noisy instances as indicators of attacks by observing deviations in high-dimensional and high-volume streaming data without requiring *a priori* knowledge. These network infrastructures demand an increasing need for streaming analytics in a record-by-record manner, processing in (near) real-time in a resource-preserving fashion (time, memory) and whereby improvements be leveraged at different levels. Thus, the main goal of this thesis is as follows:

Research Goal: *Improve Outlier Detection for Data Streams to Enhance Computer Network Security.*

In order to achieve the research goal and with respect to answering the four research questions RQ 1 - RQ 4 of Section 1.2, four perfectly matching improvements for OD can be identified, which are depicted in a high level perspective in Figure 1.5. Hence, improvements can take place to either enhance input quality by FS, to design an intelligent online OD algorithm, to exploit the OD's output in order to mine for novel attack pattern or to incorporate feedback for enhancing detection capability.

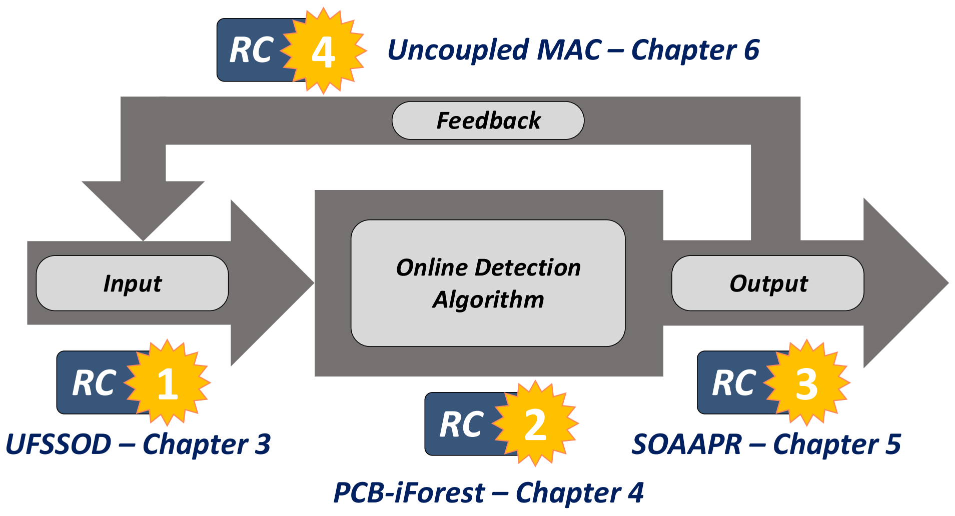


Figure 1.5: High level overview of improved OD within the thesis leading to four major research contributions (Research Contribution) (RC 1 - RC 4).

In turn, these improvements lead to the following four major Research Contribution (RC), RC 1 - RC 4, also shown in Figure 1.5, which are discussed in detail in Chapters 3, 4, 5 and 6. Likewise, these can be divided into sub-contributions as listed after each contribution. All four research contributions have been published or are submitted journal articles as mentioned in the credits section of this thesis.

RC 1: *Unsupervised Feature Selection for Outlier Detection on Streaming Data to Enhance Network Security.*

FS can be used to remove noisy, redundant or irrelevant features leading to better prediction values as well as a reduction in computational costs. Thus, the aim, with respect to improving online OD, is to identify unsupervised FS algorithms targeting the application in FS and the task of OD. To the best of the authors' knowledge no work so far has aimed to bring both sides together, therefore a conceptualization, along with a novel approach, called **UFSSOD** (***U****nsupervised* ***F****eature* ***S****election for* ***S****treaming* ***O****utlier* ***D****etection*), is proposed to achieve unsupervised FS for OD on SD (RC 1). It is discussed in Chapter 3 and offers the following sub-contributions:

- Substantial requirements for FS in the field of (i) SD and (ii) OD which are compared with a comprehensive review of the state-of-the-art pointing out the lack of any unsupervised FS method for (i) on (ii).
- Novel conceptualization of unsupervised FS for OD on SD including the discussion of two operation modes and introducing UFSSOD, the first (to the best of our knowledge) unsupervised FS algorithm for OD on SD that is also able to determine the amount of top-performing features by clustering their score values.
- Extensive evaluation of three FS methods on 15 data sets and 6 off-the-shelf online OD algorithms and the first application of an FS algorithm (UFSSOD) in line with two online OD classifiers operating in two settings in a truly online fashion while achieving better results than their bare versions using all features.

RC 2: *On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data.*

As for RC 2, we propose the so-called ***P**erformance **C**ounter-**B**ased **iForest***, denoted as **PCB-iForest**, a generic and flexible framework that is able to incorporate basically any ensemble based OD algorithm. We first highlight the most popular OD solutions for SD with an in-depth view of promising online variants of one of the most widely accepted (offline) OD algorithms, called Isolation Forest (iForest) [51]. Moreover, substantial requirements are derived to compare the existing state-of-the-art solutions. However, in this thesis, we focus on two iForest-based variants: an improvement/extension of classical iForest applied to a streaming setting and classical iForest being able to score features according to their contribution to a data record's anomalousness in a completely unsupervised way. PCB-iForest is able to deal with concept drifts by being equipped with a dedicated drift detection method. Moreover, we here take advantage of the recently proposed NDKSWIN algorithm [52]. Our solution is hyperparameter-sparse, meaning that one does not have to deal with complex hyperparameter settings which often demands *a priori* knowledge. In extensive experiments involving various multi-disciplinary but also security-related data sets with different characteristics, we show that the proposed PCB-iForest variants outperform off-the-shelf state-of-the-art competitors in the majority of cases. PCB-iForest is explained in detail in Chapter 4 and offers the following sub-contributions and it mainly differs from others because, to the best of our knowledge:

- Carefully engineered and specified requirements for an agile, future-oriented online OD algorithm design are derived, which are compared with state-of-the-art solutions pointing out the need for a more flexible solution which, is presented in this work.
- Contrary to other adaptions of iForest for SD, a flexible framework called PCB-iForest is proposed that "wraps around" any iForest-based learner (and might even be generalized to other ensemble-based approaches) and regularly updates the model in the case of concept drifts by only discarding outdated ensemble components.
- Two iForest-centric based learners are integrated into PCB-iForest providing (i) the first application of the improved iForest solution - Extended Isolation Forest - on SD denoted as PCB-iForest$_{\mathrm{EIF}}$ and (ii) online feature importance scoring by utilizing a recent iForest-based feature selection method for static data - Isolation-Based Feature Selection (IBFS) - into our online proposal denoted as PCB-iForest$_{\mathrm{IBFS}}$.

- Extensive evaluation conducted on PCB-iForest and off-the-shelf online OD algorithms on numerous, multi-disciplinary data sets with diversity in the number of dimensions and instances which no other iForest-based competitor for SD has yet performed.

RC 3: *Exploiting the Outcome of Outlier Detection for Novel Attack Pattern Recognition on Streaming Data.*

The so-called ***S****treaming* ***O****utlier* ***A****nalysis and* ***A****ttack* ***P****attern* ***R****ecognition* framework, denoted as **SOAAPR**, is proposed as RC 3. It is able to process the output of locally and autonomously or distributively operating online OD methods by equipping them with an alert generation functionality using the Intrusion Detection Extensible Alert (IDEA) [53][2] representation enriched with information that aid AC. Aggregated alerts (of the same outlier event) are fused into meta-alerts, for the sake of alert reduction, which are subsequently correlated and clustered in a streaming fashion. Clusters that are considered complete are immediately forwarded to ensure a very low response time for security analysts. A recent and innovative approach [54] is incorporated that, when triggered, transforms the reduced alert clusters, potentially representing an attack scenario, into a graph representation to derive motif signatures that capture the attack's communication relation, denoted as sig_{com} in this work. Two more signatures proposed by us, sig_{attr} and sig_{temp}, capture an attack's expression in the data's features and the time sequence pattern of the attack-related alerts. Those fingerprint-like characteristics allow a privacy-preserving sharing of novel attack patterns, e.g., utilizing the Structured Threat Information eXpression (STIX)[3], similar to shared signatures of misuse-based systems. A common problem with this type of IDS is that if similar attacks slightly change, the knowledge base may not be able to detect it anymore [55]. However, the benefit of our signatures is that they represent the attack behavior instead of specific intrusion types / classes and can thus be used to identify completely novel attacks with similar behavior from the same attack category. Our experiments with the popular, security-related CICIDS2017 and CSE-CIC-IDS2018 data sets indicate that SOAAPR is able to reliably correlate the alerts of a variety of attack scenarios and, in contrast to the offline competitor Graph-based Alert Correlation (GAC) that is also able to deal with the outcome of OD algorithms, while taking notably less processing time. Furthermore, we compare the three signatures sig_{com}, sig_{attr} and sig_{temp} in terms of their computational requirements and capability to characterize attack scenarios. Details of SOAAPR can be found in Chapter 5. It offers the following sub-contributions and mainly differs from others because, to the best of our knowledge:

- SOAAPR is the first framework that exploits the output of online OD algorithms to mine attack pattern information in a streaming fashion.

- Online OD algorithms are equipped with alert generation functionality that can be used for alert processing including timestamp, feature scoring causing the outlierness (root cause) and equally normalized outlier score results utilizing a common format - IDEA.

[2]`https://idea.cesnet.cz` (accessed on 05 September 2021)
[3]`https://oasis-open.github.io/cti-documentation` (accessed on 05 September 2021)

- Two more novel types of fingerprint-like signatures, apart from sig_{com}, are introduced, sig_{attr} and sig_{temp}, which can be used to characterize and compare attack patterns.
- Attack scenario information in the form of a signature-tuple can be shared in a privacy-preserved way generated from a novel attack pattern utilizing the STIX language.

RC 4: *A Resource-Preserving Self-Regulating Uncoupled MAC Algorithm to be Applied in Incident Detection.*

RC 4 introduces an ***Uncoupled M**essage **A**uthentication **C**ode* algorithm called **Uncoupled MAC** that is extended to work as an IDS, generating alarms if the authenticity and integrity of network communication get violated. This especially benefits resource-constrained environments where protection goals must be guaranteed and cryptographic schemes are necessary. With respect to the lightweight and resource-aware security concept of [56], Uncoupled MAC even provides simple IDS capability to a resource-aware cryptographic layer. No additional complex IDS, providing, e.g., behavioral-based detection by machine learning algorithms, must be applied if they are not feasible due to energy limitations and static communication [57]. However, Uncoupled MAC is not a replacement for a sophisticated and powerful IDS solution. While only monitoring a predefined ratio of sampled packets, a decent overhead on computational resources as well as network traffic can be preserved. By introducing a self-regulating extension, the Uncoupled MAC parameters defining its sampling mechanism can be automatically adjusted in a dynamic manner according to the detection of Uncoupled MAC violations. Uncoupled MAC is discussed in Chapter 6 and offers the following sub-contributions:

- Uncoupled MAC unites protection and detection mechanisms by equipping a cryptographic scheme with IDS-functionality and mitigating the lack of data integrity and data origin authenticity as with an IDS alone.
- Message Authentication Codes (MACs) are decapsulated from the original messages such that Uncoupled MAC is a retrofitted cryptographic protection for protocols without *a priori* security mechanisms and thus can be applied to legacy components.
- The decapsulation ensures that the underlying communication is not negatively influenced meaning that, e.g., real-time and safety aspects will be preserved.
- Uncoupled MAC can be applied to event- and time-triggered communication by the combination of a packet and time-driven mechanism.
- The self-regulation and sampling functionality allows for application in communication intense systems (e.g., in switches or cloud platforms) through an adjustable security level as well as overhead by dynamically adapting Uncoupled MAC parameters (e.g., in bandwidth-critical systems).

1.5 Organization of the Thesis

This dissertation contains seven chapters. In Chapter 1, the introduction, we have presented the motivation for this dissertation and the conceptual background of the research context, followed by the statement of the main research goal and resulting RQs. Finally, the main RCs of this thesis have been addressed and the remaining structure as a whole is presented.

Chapter 2 is an attempt at capturing the state-of-the-art in the domain of network communication protection by leveraging techniques involved in the incident handling process. Split into three sections, it provides an overview and general background on incident detection, incident analysis and incident response. Readers that are familiar with these topics can skip the chapter and proceed to the four independent main parts focusing on the contributions of our research.

Each RQ is answered by an explicit RC, which in turn, is dedicated to a separate chapter. Thus, Chapter 3 (RQ 1 - RC 1) provides details on the ***U**nsupervised **F**eature **S**election for **S**treaming **O**utlier **D**etection* (**UFSSOD**) algorithm, an FS algorithm on SD for the purpose of OD. Chapter 4 (RQ 2 - RC 2) discusses ***P**erformance **C**ounter-**B**ased **iForest*** (**PCB-iForest**), which is a generic and flexible framework that is able to incorporate basically any ensemble-based OD algorithm. Chapter 5 (RQ 3 - RC 3) provides details on the ***S**treaming **O**utlier **A**nalysis and **A**ttack **P**attern **R**ecognition* (**SOAAPR**) framework that exploits the output of online OD algorithms to mine attack pattern information in a streaming fashion in order to characterize and compare them using fingerprint-like signatures. Chapter 6 (RQ 4 - RC 4) discusses the ***Uncoupled M**essage **A**uthentication **C**ode (**Uncoupled MAC**)* algorithm, a cryptographic scheme that is equipped with an incident detection functionality and incorporates the detections' feedback in order to improve its performance.

The four core chapters (Chapter 3-6) are presented in a standalone and self-explanatory manner. Therefore, the relevant contexts including related work, the description of the solution, experimental results and discussion of results are presented in each of these chapters separately. Finally, Chapter 7, the conclusion, summarizes the work, revises the research questions and provides some perspectives for future work.

2 Background

This chapter provides relevant background information for the reader with regard to the incident handling process, covering incident detection, incident analysis and incident response. Thus, the chapter starts by providing information about incident detection mechanisms in Section 2.1 including classical IDSs with their various characteristics, methods and specificities in order to detect security-relevant incidents. A focus in this section lies on anomaly-based ML methods which are mandatory in order to detect novel malicious behavior. Two examples for network-based OD methods, iForest and Lightweight on-line detector of anomalies (Loda), are presented which satisfy the requirements stated. Furthermore, the combination of learning methods is discussed which also intersects with the proceeding Section 2.2, which mainly deals with the analysis of detected incidents. Apart from a taxonomy of alert analysis fields, the section deals with alert pre-processing including the discussion of appropriate alert exchange formats, processing by, e.g., presenting AC techniques and post-processing covering among others the prediction of malicious activity. Section 2.3 discusses incident response systems including their taxonomy, appropriate exchange formats and possible response measures. For those, an example for the reconfiguration of the network infrastructure applying SDN is given. Readers, familiar with certain topics, may skip some sections and proceed with the main part, the first major RC, beginning as of Chapter 3.

2.1 Incident Detection

Cryptographic mechanisms alone cannot provide a holistic security solution for network communication protection in the future. For instance, if an adversary compromises a sensor node, it is easy to inject malicious data. A possibility to detect attacks is to apply an IDS which is a component that monitors the events occurring in a computer system and/or network such that malicious actions attempting to compromise security primitives can be detected. At the beginning of the 1990s, Todd Herlein was invited to an IEEE conference in Oakland where he first introduced an IDS that was based on network traffic [58]. Since then, some progress has been made with regard to IDSs. Especially in the embedded sector, there has been an enormous increase in research activities in recent years. This is due to (1) the increasing networking of systems, (2) the development of more powerful and complex embedded systems and (3) an increasing demand due to an almost uncountable number of new and high sophisticated attacks. The permeation of connecting everything in various fields, inspired by the so-called “IoT-ification”, cannot only be seen in the industrial automation and avionics sectors but also in the automotive domain which are common application fields for embedded IDSs.

2.1.1 Taxonomy of IDSs

There is plenty of literature available providing an overview and a taxonomy of IDSs such as [12, 59, 60, 61, 62]. An exemplary taxonomy is depicted in Figure 2.1. In the following subsections, a selection of details on the taxonomy is provided including architectures, detection methods and other typical characteristics for IDSs.

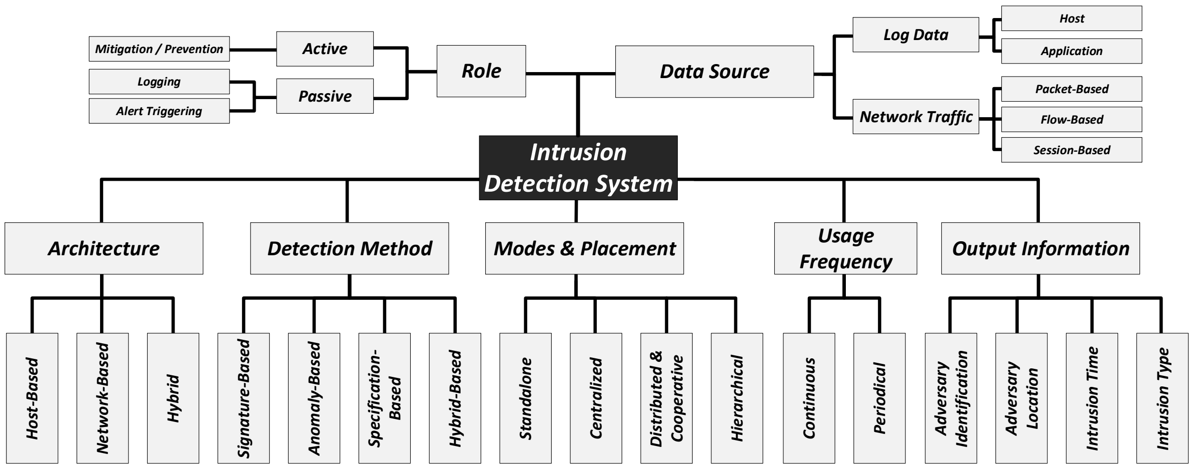


Figure 2.1: Taxonomy/classification of IDSs (cf. [12, 63]).

Architectures

The research and solutions for attack detection mechanisms are wide-ranging and manifold. However, from a higher perspective there are two main architectures of IDSs: *Host-based IDS (HIDS)* and *Network-based IDS (NIDS)*. HIDSs are applied on a single host to monitor all events for malicious actions, for instance, event logs, system logs, file access, running processes. In [64] an example of a distributed denial-of-service attack detection in cloud computing utilizing a HIDS is being presented. To detect network-based attacks in a network consisting of multiple computers, a HIDS must be installed on each of them. If a computer has been compromised or disabled by an attacker, the attack detection system is no longer trustworthy. Monitoring mainly takes place via four types of parameters. Those are the file system, the log files, the operating system kernel and the network connections. For systems that monitor changes to the file system, an attempt is made to detect when an attacker wants to gain control over the file system. For this purpose, changes to file sizes or file permissions are checked and evaluated cyclically. This method becomes problematic if the file system often undergoes changes, for example through the installation of software. Another possibility is to monitor log files of installed programs. If these reports contain information about possible attacks, the responsible administrators or users of the affected device can be warned. In addition, attacks can be detected directly at the operating system kernel level. The kernel has the ability to monitor all activities of a computer, so it can draw conclusions about malicious activities based on system calls and their parameters. As a countermeasure, the operating system can terminate the affected processes and thus render them harmless. In contrast to a NIDS, the content of network packets is not taken into account by HIDSs. Instead, connections to unauthorized ports are monitored and reported. In addition, port scans and an excessive number of connection attempts can be detected.

NIDSs, in contrast, reside on the network level to monitor and analyze the network traffic or application protocol activity. Those can either be deployed as dedicated sensors/agents either leveraged as specialized hardware or applied as software on a networking element. They record network packets and evaluate them according to, e.g., previously described rules or patterns. The former is called *Blacklisting*. Here, rules are defined that describe the properties of network traffic during an attack. In contrast, the other approach, *Whitelisting*, defines a set of rules containing information about the usual network traffic. If a rule applies, the network traffic in question is not evaluated as an attack. In contrast to host-based attack detection systems, attacks can still be detected if several computers in a network have failed or have been taken over. Since today's networks are built with switches that send incoming network packets only to dedicated ports, the sensors of a NIDS must be connected to the mirroring port. In addition to connecting the target device, all network packets are sent to this port. Another difficulty is the maximum bandwidth of the sensor. The data throughput of modern networks can exceed the processing capability of a sensor and it must discard network packets. This means that a complete monitoring of the network traffic is no longer possible. If these two disadvantages do not occur, an entire computer network can be monitored with a single sensor. The main attention over the past years focused on the application of NIDSs, due to the advent of anomaly-based NIDSs [65] which can be placed either centralized, decentralized or distributed within networks on either network switching entities (router, gateway, etc.) or dedicated hosts.

Detection Methods

Detection methods for IDSs can be categorized into anomaly-, signature-, hybrid-, and specification-based approaches [66, 67, 68] as shown in Figure 2.2. However, one may distinguish between two major ones: *misuse-based* and *anomaly-based*.

The misuse-based method, also called signature- or knowledge-based, refers to the detection of attacks whose patterns are already known, such as byte sequences in network traffic. Thus, it is founded on a set of rules or patterns describing network attacks which are either pre-configured by the system or manually by an administrator. Although signature-based IDSs easily detect known attacks, it is impossible to detect unknown attacks whose patterns are not available. Therefore, a main drawback is the lack of signatures that describe all possible variations and non-intrusive activities in network environments [69]. The anomaly-based detection method creates a model of trusted activity from collected data samples and then compares new behavior with that model. Although it allows to detect novel, unknown attacks, it could lead to false negative and false positive alarms, in which trusted but previously unknown activities could be classified as malicious.

Both approaches have their merits and demerits. For instance anomaly-based IDSs have a great potential in detecting novel attacks but they tend to be computationally intensive and are prone to false alarm generation. In contrast, misuse-based IDSs are fast in detecting known attacks with a very high accuracy and low false alarm rate but are limited in detecting new attacks. Thus, in recent years, hybrid approaches, e.g., in [70], have crystallized as the trend towards sophisticated IDS solutions. Hybrid IDSs usually combine the properties of anomaly- and misuse-based IDSs. Thus the advantages of both systems can be used by combining the methods sequentially or in parallel. Another possibility is to combine several anomaly-based methods in order to achieve better detection rates and reduce the number of false alarms. Various techniques have been proposed for detecting incidents based on misuse- and anomaly-based methods, e.g., discussed in [71]. A comprehensive overview of anomaly-based techniques with a focus on statistics-based, classification-based, clustering and outlier-based techniques and systems, soft computing-based and knowledge-based techniques and systems as well as techniques and systems based on combination learners can be found in [29].

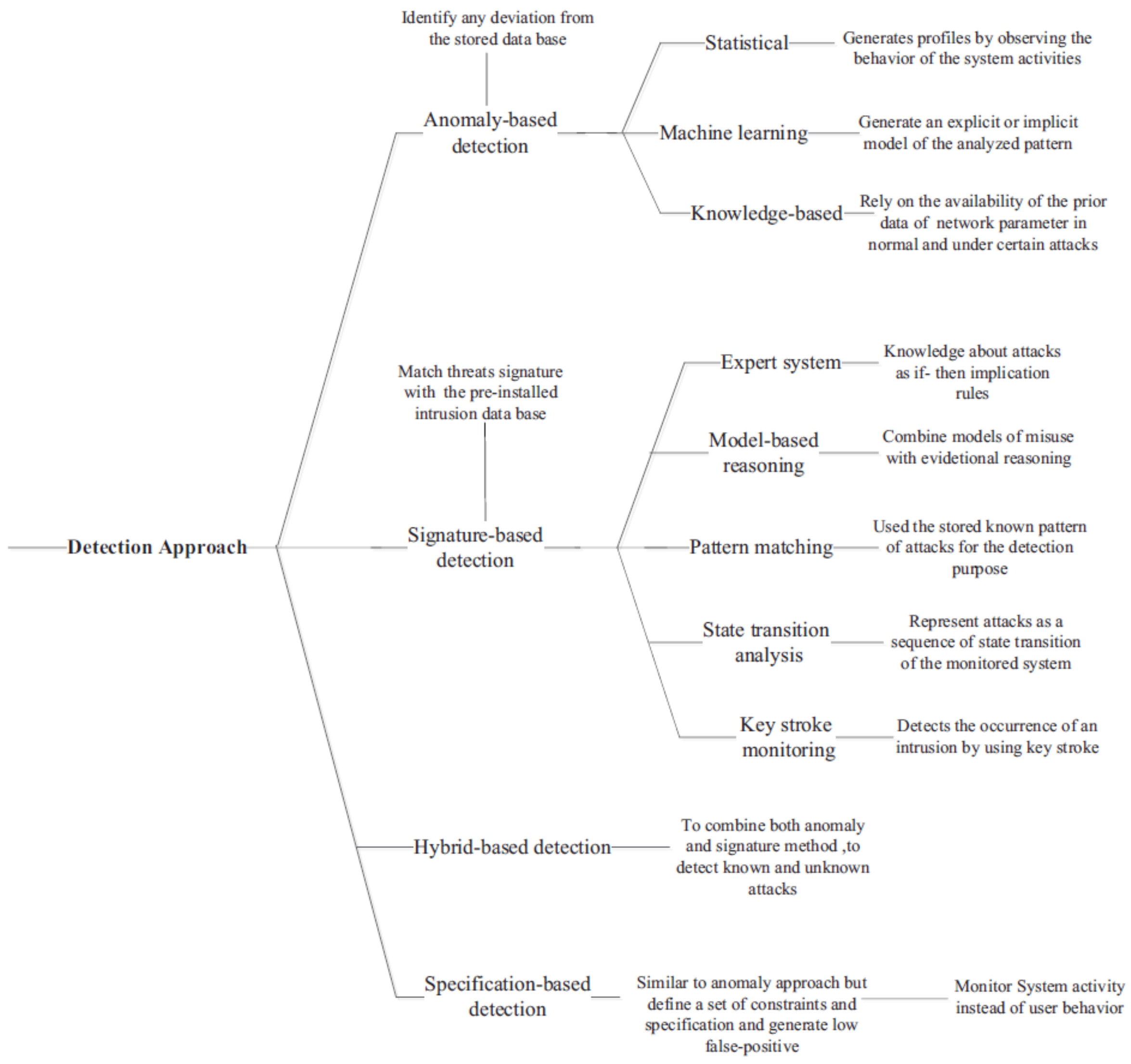

Detection Approach
Anomaly-based detection
Identify any deviation from the stored data base
Statistical
Generates profiles by observing the behavior of the system activities
Machine learning
Generate an explicit or implicit model of the analyzed pattern
Knowledge-based
Rely on the availability of the prior data of network parameter in normal and under certain attacks
Signature-based detection
Match threats signature with the pre-installed intrusion data base
Expert system
Knowledge about attacks as if- then implication rules
Model-based reasoning
Combine models of misuse with evidentional reasoning
Pattern matching
Used the stored known pattern of attacks for the detection purpose
State transition analysis
Represent attacks as a sequence of state transition of the monitored system
Key stroke monitoring
Detects the occurrence of an intrusion by using key stroke
Hybrid-based detection
To combine both anomaly and signature method ,to detect known and unknown attacks
Specification-based detection
Similar to anomaly approach but define a set of constraints and specification and generate low false-positive
Monitor System activity instead of user behavior

Figure 2.2: Taxonomy of detection approaches for IDSs [66].

Misuse-based Techniques

Techniques for misuse-based IDSs can, according to [72], be based on

- signatures (monitored events are matched against a database of known attack signatures),
- rules (set of "if-then" implication rules to characterize attacks),
- transitions (IDS is composed of a finite state machine where state transitions are used to monitor the system behavior) or
- data mining methods (learning algorithm is trained over a set of labelled "normal" or "intrusive" data).

Anomaly-based Techniques

Techniques for anomaly-based IDSs can, according to [29, 72], be based on

- statistical methods (measure certain system variables over time and derive statistical values, e.g., average or standard deviation),
- rules (normal behavior is summarized by a set of rules and anomalous behavior as a deviation from them),

- knowledge-based approaches (rule and expert systems, ontology and logic based),
- soft computing (e.g., genetic algorithms, fuzzy sets),
- distance-based approaches (attempt to overcome limitations of statistical OD approaches in higher dimensional spaces where it becomes increasingly difficult and inaccurate to estimate the multidimensional distributions of the data points and detecting outliers by computing distances among points),
- model/classification-based approaches (anomalies are detected as deviations from a model that represents the normal behavior by using data mining / ML techniques or ANNs), or
- profiling methods (profiles of normal behavior are built for different types of network traffic, users, programs, etc., and deviations from them are considered as intrusions utilizing data mining techniques or heuristic-based approaches).

Modes and Placement

Operating modes of IDSs can either be *online* (system learns and/or detects anomalies online [close] to real-time) or *offline* (system learns and/or detects anomalies offline) by working *passively* (system is configured to only monitor and analyze network traffic and alert an operator to vulnerabilities and attacks) or *reactively* (system works as in the passive mode and additionally takes pre-defined proactive actions to respond to the threat). If an IDS is operated in online mode, it can detect anomalies during operation and can be partially updated at runtime to create new models. This is a mandatory requirement for systems that are used in real scenarios. In some areas, even real-time recording of incidents is desirable. Systems that work offline are usually applied to existing data sets. By evaluating an IDS solution using existing data sets, one has better comparability with other (competing) ones. This is particularly popular for scientific work. According to [26, 73, 68], IDSs can be categorized as follows. This especially applies to NIDSs placed in network environments.

- **Centralized**: The centralized computation location works on data collected from the whole network. In the centralized IDS placement, the IDS is placed in a centralized component, for example, in the border router or a dedicated host. A disadvantage of this architecture is that it is difficult (especially with larger networks) to collect all important data at a central instance. This makes a systemic approach almost impossible to implement.
- **Distributed**: Unlike the centralized, the stand-alone computation location works on local data, disregarding decisions from other nodes. In this placement strategy, IDSs are placed in every physical object of the network.
- **Decentralized**: Similar to distributed but the placement follows a certain strategy, e.g., the network topology/hierarchy. An advantage here is that all important data can be captured. Disadvantage can be that not all participants in the network have enough resources available in order to have a complete distributed approach of the methods and to leave certain parts out of the overall system.
- **Hybrid**: Hybrid IDS placement combines concepts of centralized and distributed or decentralized placement to take advantage of their strong points and avoid their drawbacks. A combination of both, centralized and stand-alone, can be achieved through cooperative computation, such that each node can detect an intrusion on its own but also contributes to the overall decision. Here, more powerful components

are used at central locations, e.g., to create models, which usually requires more resources. The generated models are then distributed in the network to perform detection. These tests can also be performed by weaker systems, as they require far fewer resources.

Approaches which analyze the data traffic of an entire network with a central component are not target-oriented, since the amount of data to be considered requires a high computing power and thus binds resources. In addition, it is necessary to forward the data to be analyzed to the analysis component, which leads to an additional network load. Furthermore, networks are divided into subnets and an analysis of this kind initially provides little information about the network area in which an anomaly occurs. An alternative, novel concept is the decentralized use of anomaly detection systems. Here network sensors are used which are placed in the subnets of a network. Instead of analyzing the entire network traffic, the data traffic within each subnet is considered separately and only the anomalous alarms are passed on to a central component that correlates them. In addition, having more and more distributively connected devices, collaborative IDSs therefore promise to even detect highly advanced distributed attacks. Especially decentralized NIDSs are popular and a selection of research work dedicated to them is presented in the following. Already in 2001, an architecture is proposed in [74] that collects decentralized network traffic and sends it to a central server which classifies it. The drawback of this approach is the increased network overhead. Statistical methods are first applied to the monitored data sets in order to classify the outputs with an ANN.

Jahnke [75] has defined requirements for a decentralized attack detection architecture. These include the ability to work continuously without human interaction, to detect attacks on the IDS itself, or to adapt the IDS to the system or network behavior over time. Jahnke is also proposing the use of the Intrusion Detection Message Exchange Format (IDMEF) [76], a data format based on the eXtensilble Markup Language (XML), as a structure for communication between the components of his architecture. Six components are proposed: The sensor is defined as a process that collects or generates measurement data. For each sensor, an adapter is required that monitors the work of the sensor and processes the aggregated data and transforms it into the required form of the IDS. The message distributor is required to receive or send command messages. This is done via the communication channels, of which there are basically two: on the one hand the already addressed command messages, on the other hand one for special events, such as a detected attack. Event processing evaluates the events and determines the behavior of the last component, the reaction unit. However, there is no separation of command and event messages. Even a proposed response measure is not part of this work. A response action requires knowledge of the underlying network and the available response units. Lupu et al. [77] have developed a decentralized architecture for attack detection and implemented a communication framework based on IDMEF. The existing libraries for IDMEF, *libprelude-dev* and *libidmef*, have been discussed but an improved software library adapted to the decentralized architecture has been developed. In order to define the reaction to received alarms, an own syntax is used which is based on the developed functions. A programmer is offered the possibility to register his own callbacks which are called at every event, i.e. an incoming message. Therefore, a programmer can influence the role of a client himself. The use of IDMEF as an alarm format is suggested because of the suitable structure and the meaningful definition of fields. The presented architecture, however, mostly refers to the evaluation of these alarms and less defines the structure of the sensors. Nevertheless, it represents a suitable basis for the evaluation of generated alarms in a decentralized structure. Hu et al. has presented a method in [78] for detecting anomalies in network traffic. They are specifically designed to address the problem of the frequently changing structures of today's computer networks. In addition to the investigation of several different algorithms for the detection of anomalies, a new architecture of NIDSs is presented. The data of each packet passes through several stages of anomaly detection. First, a local model is used that is only locally present in the respective node. Then further global models, which are present in all nodes of the network, are applied. The approach demands many computing resources. However, those are divided between sensors and a server.

2.1.2 Aspects of Machine Learning

ML, a relative to computational statistics, data mining and data science disciplines, has an upscaling trend in the field of cyber security [79]. It has the ability to find similarities within a large amount of data such that intrusions creating distinguishable patterns within the network traffic that can be detected efficiently [30]. A major benefit is that ML can be used to identify anomalies in the data without prior knowledge even within high-dimensionality and massive amount of data which humans would never be able to recognize. Although often classified under a branch of ML, deep learning approaches are due to the reasons mentioned in Section 1.2.1 not considered in the following. Therefore, we mainly focus on so-called shallow models, traditional ML methods, that contain none or only one "layer" [12]. However, a taxonomy of common ML algorithms with the classification into shallow and deep learning models can be found in [12].

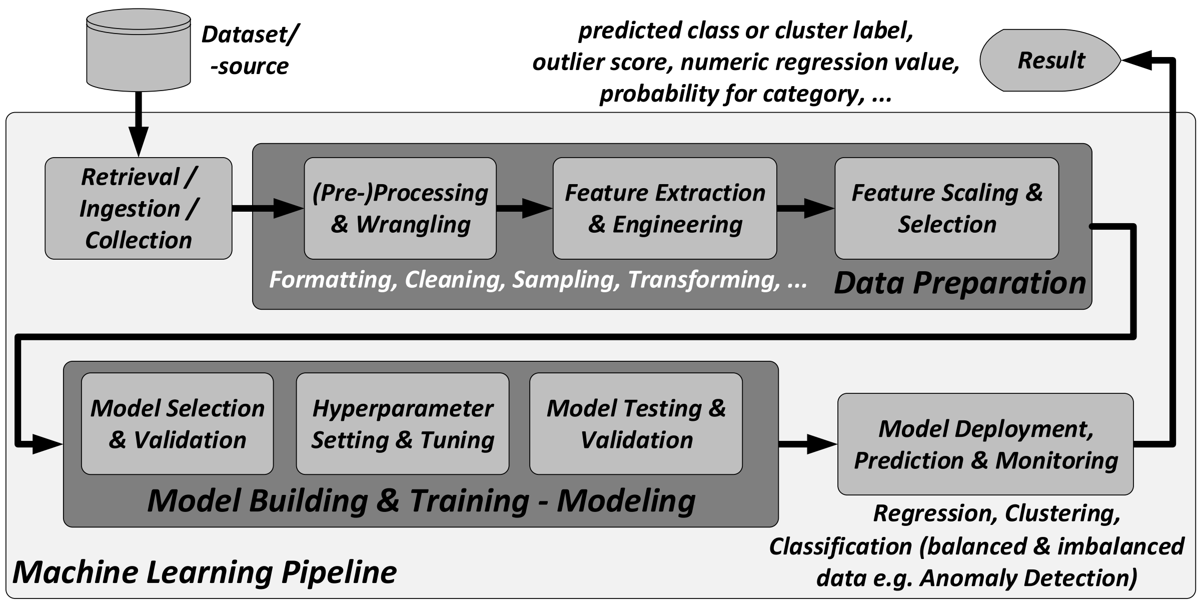

Dataset/ -source
predicted class or cluster label, outlier score, numeric regression value, probability for category, ...
Result
Retrieval / Ingestion / Collection
(Pre-)Processing & Wrangling
Feature Extraction & Engineering
Feature Scaling & Selection
Formatting, Cleaning, Sampling, Transforming, ...
Data Preparation
Model Selection & Validation
Hyperparameter Setting & Tuning
Model Testing & Validation
Model Building & Training - Modeling
Model Deployment, Prediction & Monitoring
Regression, Clustering, Classification (balanced & imbalanced data e.g. Anomaly Detection)
Machine Learning Pipeline

Figure 2.3: A typical Machine Learning Pipeline/Workflow.

Figure 2.3 shows a ML pipeline, often also referred to as ML workflow, which is a sequential combination of different mechanisms transforming data instances (data points) from a set of n objects $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}\}$ where each instance consists of a d-dimensional real-valued vector $\boldsymbol{x_i} = \{x_{i1}, x_{i2}, ..., x_{id}\}$ with $\boldsymbol{x} \in \mathbb{R}^d$ into a target value y, e.g., a class label. Each dimension is also called a feature. For network-based features, one may distinguish basic features (derived from packet headers [meta data] without inspecting the payload, e.g., ports, Media-Access-Control or IP addresses), content-based features (derived from payload assessment having domain knowledge, e.g., protocol specification), time-based features (temporal features obtained from, e.g., message transmission frequency, sliding window approaches) and connection-based features (obtained from a historical window incorporating the last n packets) [29, 80].

Learning Phases and Approaches

Many algorithms used in ML work in two (or three) phases: the *training phase*, (the verification phase) and the *evaluation phase.* In the training phase, a state or *model* is build by the selected algorithm. This model is then used by the algorithm in the evaluation phase to obtain the result. An additional verification phase might help to verify and optimize a built model before the actual operating (evaluation) phase. However, there are also algorithms that work in one step. In most cases, this involves obtaining information from the existing training data set (data mining). Basically, ML-based algorithms can be classified into three learning methods. The biggest difference can be seen in the already known basis of information. The learning method can already provide information about which problem the algorithm can solve, e.g., labels for the classification problem. In *supervised learning* (1), e.g. [81], the labels for the training data set are required. These are included in the models during the training phase. This means that the algorithm adapts the values of the model to the known labels. The goal is to approximate an unknown function $f(x)$ with the resulting value Y. Here x represents the data points and Y the known labels. If the function $f(x)$ is approximated, the resulting value can be calculated for each additional data point. The *unsupervised learning* (2), e.g. [82], is classified by algorithms that do not require labels in the training phase. Mostly, these algorithms are used to analyze the data set more precisely and to model it. Clustering, i.e. the division of similar data points into groups, is a well-known representative of this learning method. *Semi-supervised learning* (3), e.g. [83], is a trade-off between supervised learning and unsupervised learning. Here a label is only available for a subset of the data points. This procedure is used if there are many data points and a label cannot be assigned to them completely manually. *Hybrid* approaches, e.g. [84], try to exploit the benefits of the aforementioned. For instance the better detection rate when having labelled training data and mitigate their demerits, e.g., having a false alarm rate when assuming that normal data points are far more frequent than anomalies.

Problem Types

The selection of the appropriate model within the *Model Building & Training* stage mainly depends on the ML problem type. Different types of problems, that should be solved with methods of ML or data mining, are distinguished. In *classification*, data should be divided into already known classes or groups. The *regression* problem is very similar to the classification problem, but the result is not a class but a numerical value. With *clustering*, a set of data points shall be divided into classes. Each class should only contain points that are similar. Here it can be further differentiated whether the number of contained classes is already known before or an unknown number of classes are contained in the data set. The task *OD* belongs to the class of algorithms which can detect anomalies with the properties that these are only a few compared to the normal data and differ substantially from it. Such algorithms detect outliers during the evaluation phase without previously known information about the data, e.g., clusters.

Machine Learning Pipeline

For security analysts it is often difficult to properly set up a ML pipeline due to the multitude of possibilities to choose from at each pipeline stage. Domain experts are expected to have a very high level of multidisciplinary expertise from data science. This ranges from a meaningful data (pre-)processing, building of a domain-driven feature engineering and selection of the best-performing features with respect to the *Data Preparation* stage of Figure 2.3 in order to enable an efficient data analysis and to improve the performance of the actual ML task [85, 86]. This process is necessary since normally a data set contains raw data, which must be translated into an understandable format and optimized for a downstream applied algorithm.

Furthermore, the security analyst is required to select the best algorithm associated with its corresponding optimal hyperparameters with respect to the *Modeling* stage of Figure 2.3. Recent developments in the field of Automated ML (AutoML) aims to support those domain experts to properly set up a ML pipeline without extensive knowledge of statistics and ML. The authors of [87], for instance, have provided a mathematical formulation covering the ML pipeline construction modeled as a Directed Acyclic Graph (DAG) and benchmark existing frameworks including algorithms towards the Combined Algorithm Selection and Hyperparameter (CASH) optimization problem. Therefore, for an optimal selection of an algorithm and its corresponding hyperparameters, the authors formulated a ML pipeline denoted as $P_{g,\vec{A},\vec{\lambda}}$ as the triplet $g, \vec{A}, \vec{\lambda}$ with $g \in G$ as a valid pipeline shape and $|g|$ the length of a pipeline referring to a DAG in which the node represents a basic algorithm and the edges the flow of an input data set through different algorithms. The vector $\vec{A} \in \mathcal{A}^{|g|}$ consists of the selected algorithm for each node of the algorithm set $\mathcal{A}$ and $\vec{\lambda}$ a vector comprising the hyperparameters of all selected algorithms from the domain $\Lambda^{(\cdot)}$. A pipeline trained on a data set $D = (\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)$ is given by $P_{g,\vec{A},\vec{\lambda},D}$. The pipeline performance π is given a data set D' of size m and a loss metric $\mathcal{L}(\cdot,\cdot)$ is calculated as shown in Equation 2.1. The pipeline creation problem can then be formulated of finding a structure together with an algorithm and hyperparameter selection that minimizes the loss (Equation 2.2). The popular CASH optimization problem referring to [88] can be derived setting $|g| = 1$.

$$\pi(P_{g,\vec{A},\vec{\lambda},D}, D') = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(P_{g,\vec{A},\vec{\lambda}}(\vec{x}_i), y_i) \tag{2.1}$$

$$g^*, \vec{A}^*, \vec{\lambda}^* \in \underset{g \in G, \vec{A} \in \mathcal{A}^{|g|}, \vec{\lambda} \in \Lambda}{\arg\min} \quad \pi(P_{g,\vec{A},\vec{\lambda},D}, D) \tag{2.2}$$

However, the application of AutoML is an offline supervised learning setting. To solve the pipeline problem, multiple iterations are necessary while incorporating the feedback from a benchmark data set, e.g., KDD'99 [89] or NSL-KDD[1], which is quite resource consuming in terms of time and computation. The focus lies in solving the CASH optimization problem but transparency of ML is important such that automatic selection and adjustment of algorithms might lead to a misunderstanding by the user and the selected algorithm with its hyperparameters might perform insufficiently with other data sources or in real-world applications. Although there are existing methods that are able to optimize hyperparameters even for anomaly detection in the unsupervised case, e.g., by

[1] `https://www.unb.ca/cic/datasets/nsl.html` (accessed on 05 September 2021)

exploiting a concept called Mass Volume Curves [90]. Nevertheless, they require training and testing for each hyperparameter value, thus they do not operate in the online case.

Data Preparation

Data Preparation is a significant part of ML and essential in order to enable an efficient data analysis and to improve the performance of the algorithm [85, 86]. Data must first be collected and cleaned as well as corresponding information must be extracted from the raw data (*feature extraction*). Furthermore, a selection of the relevant data for the analysis must be carried out. This is followed by the actual pre-processing of the data. Possible goals of the data pre-processing are, for example, to convert the data into an optimal form for the analysis in order to increase the performance of the anomaly detection or to reduce the amount of data to be analyzed and thus to preserve resources. Data pre-processing and analysis must take place during the operation of the underlying network with the intention of detecting anomalies promptly. In general data pre-processing describes the necessary steps before an analysis of data can occur. Data pre-processing methods (with their respective functions) can be divided into the categories *data cleaning* (handling of anomalous, missing, erroneous and inconsistent data), *data integration* (merging of multiple data sources and handling of redundant data), *data transformation* (scaling, normalization and categorization of data) and *data reduction* (dimensionality and quantity reduction, discretization and compression of data) [91, 92]. Methods for data scaling or normalization are among others [93] the min-max or unit scaling (L1- or L2-norm). Two popular methods exist for dimensionality reduction [94]: Principal Component Analysis (PCA) and Random Projection (RP). PCA originates from statistics and is used to locate patterns in high-dimensional data. After such a pattern has been found in a set of data, its dimension can be reduced (compression). Decisive is, that with the reduction, the information represented by the data is largely preserved. Due to the Johnson-Lindenstrauss lemma described in [95] several possibilities were developed to map higher order matrices into lower dimensions. This lemma states that at data points in a high dimension only a small distortion occurs by mapping into a lower dimension with a certain probability. Random projection is taking advantage of this lemma. Clustering is another method of pre-processing data. Here, data is assigned to clusters according to its nature, such that the data is categorized. This classification can be used as a pre-processing measure to implement various measures: (1) The categorization of data may assist the analytic process in the classification of data. (2) The type of further processing can be selected on the basis of the cluster membership of data. (3) Depending on the cluster, data sampling is possible. (4) Each cluster can be further processed by a separate algorithm which allows a detailed analysis of the data. Sampling in statistics refers to the selection of a subset of instances from within a statistical population in order to estimate characteristics of the whole population. In pre-processing, sampling selects a subset, a representative, of a set to allow an analysis of only the subset while loosing as little information as possible. By this measure, the amount of data for analysis is reduced, and therefore fewer resources are required. For a discussion on various sampling methods refer to Chapter 6.

The disciplines "Feature Engineering" [96] and "Feature Learning" [97] play an important role in building ML-based IDSs since the chosen feature set (the collection of selected features - FS) highly affects the performance of the IDS. For instance, Wang et al. propose an automated feature learning approach in [15]. They abstract network traffic such that for spatial features traffic is transformed into "traffic images" to exploit the advantages of image processing, e.g., image classification based on geometric features to classify the traffic images, which also indirectly achieves the goal of identifying malicious traffic. For temporal features, the time series analysis method is applied to detect malicious behavior on extracted temporal features. The so-called Hierarchical Spatial-Temporal Features-based IDS is divided into two major steps: first the low-level spatial features of network traffic are learned using deep convolutional ANNs and then high-level temporal features are learned using long term short memory networks. For dimensionality reduction instead of the well-known PCA, the t-SNE algorithm is used. Different feature representations can be used to address different fields of anomaly detection. Some of them are considered naïve when they contain basic information about the software or network (e.g., IP source and destination address of a data packet), while others are considered rich when they represent deeper details (e.g., temporal relations of payload content) [96]. According to [26], features can be obtained by the following processes: feature construction creates new features by mining existing ones by finding missing relations within features. While extraction works on raw data and/or features and apply mapping functions to extract new ones. Selection works on getting a significant subset of features. This helps reduce the feature space and reduce the computational power [26].

Feature Selection

Especially when applying ML algorithms on high-dimensional data sets, one has to deal with the curse of dimensionality referring to the phenomenon that data becomes sparser in high-dimensional space. This adversely affects the storage requirements and computational cost of the algorithms. The process of choosing a subset of significant features $\mathcal{F}_S \subset \mathcal{F}$ from a data set $\boldsymbol{X}$ with the descriptive features $\mathcal{F} = \{f_1, f_2, ..., f_d\}$ is called FS or attribute selection. The subset divides the data set $\boldsymbol{X}$ into $\boldsymbol{X}^* = \boldsymbol{X} \cap \mathcal{F}_S$, thus reducing the data set's dimension and volume of information processed by the consecutive ML algorithm. Generally, it formulates a criterion to evaluate a set of features in order to identify redundant or irrelevant (non-informative) features for the ML task that needs to be removed which are deteriorating the performance of the ML algorithm. The output of FS is either a ranked list of the features or a subset of them. Therefore, in the case of classification, on one hand a more precise classification result can be achieved while on the other hand the computational effort can be limited by minimizing the cardinality of the selected feature set. This results in a faster, more cost-effective and improved prediction performance. In practice, FS still mostly depends on expert knowledge. However, with high-dimensional and high-volume data sets and its complex interwovenness this is no longer a humanly manageable task. As discussed, dimensionality reduction could also be achieved by methods such as PCA and RPs, where higher order matrices are mapped into ones with lower dimension since, at data points, in a high dimension only a small distortion occurs by mapping into a lower dimension with a certain probability. However, a major disadvantage of this process is that after classification a root cause analysis, e.g., which features contributed the most for the classification result, is not possible any more. This is because of the creation of new synthetic features from a linear or nonlinear combination of the original ones and then discarding the less important. As the physical meaning of the features are no longer retained by this projection, further analysis is impeded. FS, in contrast, is simply selecting and excluding given features without changing them such that for a root cause analysis one can still refer to expert domain knowledge by maintaining their physical meaning (feature interpretability).

Methods for FS can generally be categorized into *wrapper*, *embedded* or *filter* methods. Wrapper approaches are seeking for their subset by “wrapping around” FS over a ML algorithm following the iterative procedure that an original set repeatedly is divided into a subset which then gets evaluated by calling the subsequent classifier. Depending on the goodness of the subset either a new one needs to be generated or the result yields the best feature subset (stopping criteria). Since the wrapper approach includes a specific induction algorithm to optimize FS, it often provides a better classification accuracy result than the other methods. However, this method is time-consuming considering the search space for d features of 2^d while d is typically very large and strongly coupled with the classifier which makes it impractical to be applied to large data sets containing numerous features and in online settings. Embedded approaches include FS in the training process of the ML algorithm. Thus, operational costs are being reduced due to the classification process needed for each subset [98]. The filter approach does not require knowledge of the subsequent ML algorithm and measures the intrinsic statistical properties of the data set. It can be grouped into feature ranking methods assigning weights to features based on their relevancy and feature-subset-evaluating methods that also involve relationships between features finding redundancy [99]. However, statistical measures must be carefully chosen based on the type of input variable and the model outputs. From an online application perspective, filter approaches seem to be the most promising candidates, since they do neither demand offline training nor rely on multiple iterations. Similar to labeling in ML, FS can be broadly classified from a supervision perspective into *supervised* and *unsupervised* methods with respect to use or ignore the target variables. Over recent years, unsupervised approaches have gained attention since acquiring labeled data is particularly expensive in both time and effort. Furthermore, sufficient label information is usually not available in real-world applications. With respect to the data perspective, FS can be classified according to [100] into *static* and *streaming* data.

Unsupervised FS tries to find a relevant subset of features that preserves the inherent structure as much as possible [101]. This means that it tries to reduce the number of features without complicating the detection of anomalies. Since FS is a non-deterministic polynomial acceptable problem [102], there are many approximation solution methods. Luo et al. [103] are following a way to select features based on “Adaptive Reconstruction Graphs”. This led to the realization that omitting features can improve the result. If some features can be ignored, the calculation time is also reduced. Wieland et al. [102] have presented an approach based on a Support Vector Machine (SVM). This is used to model the relationship between the distribution of a particularly invasive mosquito species. The work was able to identify new features that improve the underlying detector. However, the method used is extremely computationally demanding and therefore not suitable for the usage on low-powered devices. Aljawarneh et al. [104] have developed a hybrid model using the following classifiers: J48, Meta Pagging, RandomTree, REPTree, AdaBoostM1, DecisionStump and NaïveBayes. An information gain detector based on mutual information was used to assign an information score to all possible features. The hybrid model was then applied to the best eight features. The calculation of the information gain detector is unfortunately dependent on information that is not always available when using unsupervised learning.

2.1.3 Outlier Detection

OD, also referred to as anomaly, deviation or novelty detection as well as exception mining, is an important issue for many real-world application domains, especially detecting indicators of malicious activity in computer networks. Manifold definitions in the literature exist for OD. In the following, three exemplary definitions are given, regarded general enough to cope with various types of data and methods [105].

- **Hawkins (1980)**: "An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism." [106]
- **Johnson (1992)**: "An outlier is an observation in a data set which appears to be inconsistent with the remainder of that set of data." [107]
- **Barnett and Lewis (1994)**: "An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs." [108]

In general it can be described as an atypical pattern or observation that significantly deviates from the norm based on some measure and thus attracts suspicion. Outliers are mainly characterized by three assumptions: (i) the majority of data is normally having only a small portion of outliers (imbalance) which (ii) are statistically different from the normal data (distinction) and (iii) do not happen frequently (rarity). Outliers are often classified into three categories referred to as Type I-III outliers: *point outliers*, *contextual outliers* and *collective outliers*. Point outliers show exceptional behavior compared to all other values in the data, whereas contextual outliers depend on the context, meaning, their abnormality depends on other contextual attributes even if the value itself seems normal. Collective outliers depend on the consecutive series of values whereby a single value might be normal but their consecutive set shows exceptional behavior. Referring to typical ML tasks, outliers are often seen as noise and are removed during the data preparation stage. However, for some applications, especially for detecting completely novel malicious activity in network security, the data points containing outliers carry the significant information. In terms of classification, OD constitutes a special form of imbalanced data where the outlier class show the properties (i)-(iii) stated compared to the normal data class. The output can either be a binary class label $y \in \{normal, abnormal\}$ or a score value $y \in \mathbb{R}$ that describes the strengths of anomalousness. The score itself can be used to derive a class label by utilizing a threshold value. In a later root cause analysis, outlier score values carry more information than a simple binary value.

Thus, in this thesis, we focus on OD algorithms in which $OD(\cdot) : \boldsymbol{x_i} \rightarrow \mathbb{R}$ assigns an outlier score for each data object in $\boldsymbol{X}$. This divides $\boldsymbol{X}$ into a set of outliers $\boldsymbol{X}^+$ and inliers $\boldsymbol{X}^-$ ($\boldsymbol{X} = \boldsymbol{X}^+ \bigcup \boldsymbol{X}^-$). Numerous methods have been introduced for OD. The most common ones are statistics-, distance-, clustering-, or density-based techniques. Other methods, including their properties, are discussed in [109, 110].

As already pointed out, in particular, the missing ground truth values in evolving (theoretically infinite) data that demands real or almost near real-time processing, taking the evolution and speed of data into account, requires unsupervised OD methods capable of dealing with SD. In the streaming setting, $\{\boldsymbol{X}_t \in \mathbb{R}^{n_t \times d}, t = 1, 2, ...\}$ is a continuous transmission of data records which arrive sequentially at each time step t. The count of features is denoted as d (dimension) and $\boldsymbol{x}_t$ the n_t-th d-dimensional most recent incoming data instance at time t. Widely accepted and popular solutions, such as Hoeffding Trees [111] or Online Random Forests [112], achieve good accuracy and robustness in data streams [113] but are not designed to operate on unlabeled data. Over the past couple of years, methods have been proposed that satisfy the unsupervised and online requirement, such as [114, 115, 116], but just a few, iForest [51], HS-Trees [117], RS-Hash [118] and Loda [119], have been shown to outperform numerous competitors and are therefore regarded as the state-of-the-art [120, 121]. Even if iForest was originally intended as an offline algorithm, a handful of variants, such as [122, 52, 123, 124, 113], have been proposed that are adapting it or are taking advantage of its concept to operate on SD.

Example of Two Outlier Detection Algorithms

In the following two representative exemplary algorithms for anomaly detection using ML, iForest [51] and Loda [119], based on unsupervised OD, are presented. Those have mainly been chosen according to the following criteria for the application as an anomaly-based NIDS. Furthermore, the sections discussing iForest and Loda serve the understanding of solutions presented in Chapter 3 and 4. More detailed requirements with regard to online unsupervised OD can be found in Chapter 4. The criteria are:

- operation without knowledge of data labels,
- possibility of online (real-time) detection of outliers,
- detection of previously unknown and advanced attacks,
- modeling and operation in environments which might contain anomalous data,
- applicability on lightweight devices with little available resources,
- coping with high dimensional data sets,
- providing outlier score values instead of simple binary values, and
- allowing feature interpretability, meaning the identification which feature contributed most to cause the outlierness.

The **iForest** algorithm is predestined for anomaly detection because it does not use distance or density methods which makes it much less computational intensive compared to methods such as k-Nearest-Neighbor [122] and also because it is well suited for real-time usage unlike most other algorithms [119]. In addition, it belongs to unsupervised learning and does not require a labelled training data set, such as Hoeffding Trees [125], as it recognizes patterns and does not sort out packets based on their label. Furthermore, it is independent of the scaling of the data set dimensions since its threshold for determining anomalies is based on the tree depth [126]. This algorithm attempts to separate outliers from other data points by isolating them by taking advantage of the fact that data points, that differ from other data points, require fewer steps to be isolated from them as shown in Figure 2.4 (a). In addition, the algorithm uses the observation that when a data set is represented in a binary search tree, anomalies are inserted in a tree at a shallower depth than normal values as depicted in Figure 2.4 (b).

The iForest algorithm forms several *Isolation Trees* in the training phase. These Isolation Trees are then the model for classification in the evaluation phase. Each tree is a real binary tree, which nodes are provided with different information. In the training phase, the training data set X is available with n data points. The training data set is divided into t subsets X' and X. It applies $X' \subset X$. Each subset X' contains ψ data points. An Isolation Tree is formed from each X'. This is done by recursively dividing

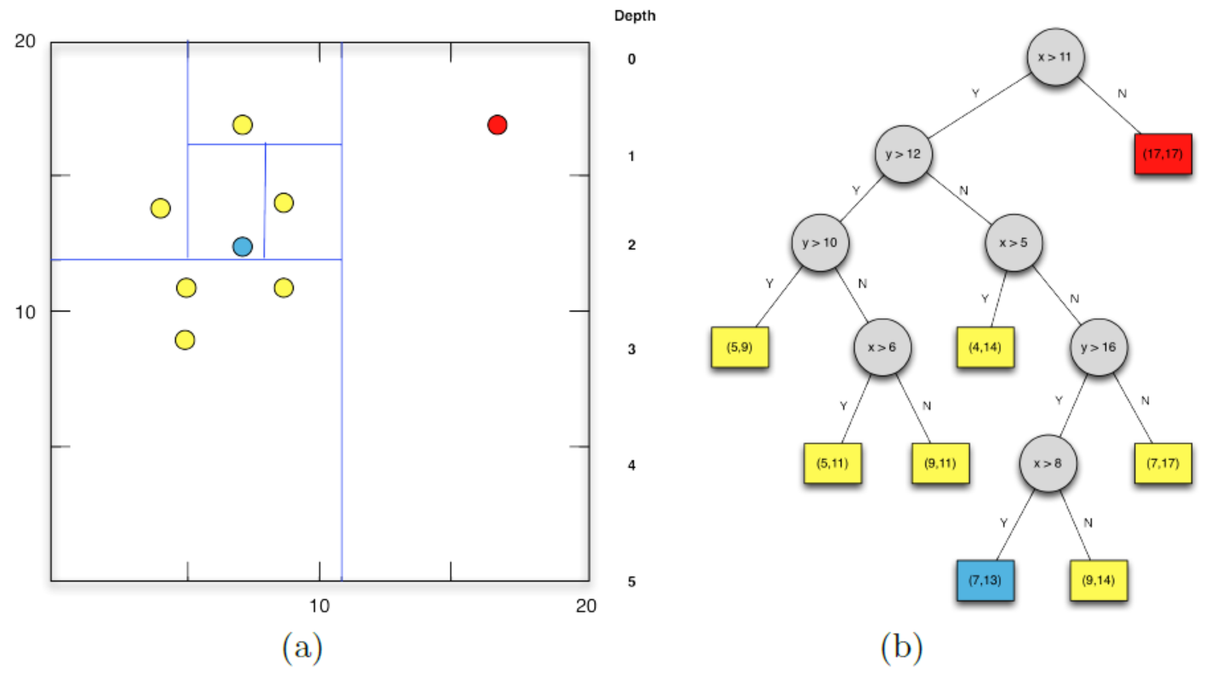


Figure 2.4: (a) Isolating an outlier, (b) Representation of a tree model [126].

X' by randomly choosing a feature q and a value p. p is a random value between the minimum and the maximum of the feature q of all data points at a node of the tree. New child nodes are formed by processing all data points for which $p_q < p$ applies, where p_q is the value of the feature q of a data point, in the left child node. All data points to which $p_q \geq p$ applies are processed further in the right child node. The recursion ends when fewer than two data points have arrived in a node or all data points are equal. The values q and p become attributes of the current node. Each node of a tree has either no child nodes or two. This is repeated for each X'. After the training phase, t Isolation Trees exist.

To classify a data point x, all Isolation Trees are traversed from the data point during the evaluation phase. The data point travels through the nodes to the previously trained values for q and p. If the data point $p_q < p$ applies, if p_q is the value of the feature q of the data point, the data point moves to the left child node. If $p_q \geq p$ applies, the data point moves to the right child node. This happens until an end node is reached. The result of this migration is the tree depth. From all reached depths $h(x)$ the average $E(h(x))$ is computed. The calculation of the resulting *score* is shown in Equation 2.3. First, $c(\psi)$ must be calculated. This equation is borrowed from the number of unsuccessful searches in a binary search tree. It represents the average depth reached by a binary tree when it contains ψ data points. n is the number of data points used to build a model (Equation 2.4). H represents the "$\psi - 1$"-th subsequent element of the harmonic sequence. This can be calculated approximately by Equation 2.5. The variable γ represents the Euler-Mascheroni constant ($\approx 0,57721$). The main advantage of the iForest algorithm is its low time complexity. This is $O(t\psi^2)$ in the training phase and $O(nt\psi)$ in the evaluation phase. It should be noted in particular that ψ can and should be kept small to avoid the effect of *swamping*.

Loda [119] has been presented as another algorithm besides iForest with similar properties. It belongs to the OD algorithms with unsupervised learning as well and consists of

$$score(x, \psi) = 2^{-\frac{E(h(x)))}{c(\psi)}} \tag{2.3}$$

$$c(\psi) = \begin{cases} 2H(\psi - 1) - \frac{2(\psi-1)}{n} & \text{for } \psi > 2, \\ 1 & \text{for } \psi = 2, \\ 0 & \text{for } \psi < 2 \end{cases} \tag{2.4}$$

$$H(n) \approx ln(n) - \gamma \tag{2.5}$$

a collection of k one-dimensional histograms, each histogram approximates the probability density of the input data projected onto a single projection vector. Projection vectors diversify individual histograms which is a necessary condition to improve the performance of individual classifiers. To train the algorithm, projection vectors w_i are first generated and histograms initialized. Each projection vector is generated during the initialization of the associated histogram by first randomly selecting $d^{-\frac{1}{2}}$, different from zero, features and then randomly generating non-zero values according to $\mathcal{N}(0,1)$. The histograms of each projection vector are updated with $z_i = x_j^T w_i$, where x^T is the transposed sample vector. The features used must be of approximately the same order of magnitude. Loda's output $f(x)$ on a sample x is the average of the logarithm of probabilities estimated on a single projection vector (Equation 2.6).

$$f(x) = -\frac{1}{k}\sum_{i=1}^{k} \log \hat{p}_i(x^T w_i) \tag{2.6}$$

Loda is especially useful in domains where a large amount of samples have to be processed because its design achieves a very good weighting between accuracy and complexity. The algorithm exists in different variants for batch and online learning. With the batch variant, data instances are collected and collectively used in the training routine. In this routine the projection vectors and the histograms are generated. In the online version, a histogram is continuously updated which makes it possible to use it even on devices with very low resources and thus eliminates the splitting of the modeling and evaluation phase (as necessary with iForest). However, in the early running time of the online variant, the algorithm will probably produce more false positives than the batch version, since the histograms need a certain amount of time to be fully updated. Loda can handle missing variables and can sort features according to their contribution to the anomaly score. Also the anomaly detection does not fail completely if single sensors are missing. In its original form, the algorithm returns a score value. The larger the value, the more likely it is an indication of an anomaly. However, this score value can be reduced to a probability by Equation 2.7). Here $f(x)$ is the score value of Loda from Equation 2.6.

$$\hat{p}(x) = 1 - e^{-f(x)} \tag{2.7}$$

With respect to Chapter 5, we are mainly interested in two functionalities of online OD algorithms. Firstly, for instance, to be able to deal with false positives, OD should provide outlier score values instead of simple binary values. Secondly, finding the actual root cause of incidents is still an open challenge for IDS. The importance of the features of the input data can play a major role when it comes to analyzing detected outliers. Thus, OD algorithms are required that are able to score or rank features according to their contribution to a data instance's anomalousness. The former criterion is fulfilled by all of the aforementioned algorithms although their scoring range differs. For instance, while iForest's scoring takes values from $[0, 1]$, Loda yields values from $[0, \infty)$. Referring to the second criterion, to the best of our knowledge, only Loda and the adaption of iForest for SD, PCB-iForest$_{\text{IBFS}}$ (refer to Chapter 4) provides by design the functionality to measure the statistic significance of each feature to its contribution of a data instance's scoring result in an unsupervised manner. From a supervised perspective, the Random Forests (RF) [127] algorithm, for which an online variant also exists [112], can provide feature importance scoring functionality using the SHapley Additive exPlanations (SHAP) method [17]. This method is founded on the so-called Shapley values which provide an explanation of a prediction by computing the contribution of each feature to the prediction - a method from coalitional game theory.

The complexity theory provides a measure for the representation of the differences between iForest and Loda. Table 2.1 shows the effort of resources (time and memory complexity) for the execution of both anomaly detection algorithms according to [119]. In the table, n denotes the number of samples for the training phase, d the number of features (dimensions), k the number of trees (iForest) or the number of histograms (Loda), l the number of samples for the construction of a single tree (iForest) or the length of an observation window for the continuous histograms (Loda) and b the number of histogram classes (Histogram bins) for Loda. In Table 2.1 a distinction is made between Loda with two alternating histograms (1) and the implementation with a continuously updated histogram (2). The construct of a binary tree contained in the iForest can be created in two ways. If all elements for the creation are not known in advance, then each element of the l elements must be added one after another. In the worst case, the complexity of the insertion is $\mathcal{O}(l)$ and so $\mathcal{O}(l^2)$ results. In the more likely case (also called average case), all l elements are known in advance and could be sorted by $\mathcal{O}(l \log l)$ and inserted afterwards. For this, one takes the middle element, insert it as root node and proceed recursively for the remaining elements. At the end one gets a so-called "balanced" tree, where l elements were inserted with $\log l$. Thus, a time complexity of $\mathcal{O}(l \log l)$ can also be achieved for the creation. The memory complexity when learning the model is $\mathcal{O}(n)$, where n is the number of elements in the data set to be learned. Once the model has been trained, the memory complexity is reduced to the number of memory complexities per binary tree $\mathcal{O}(k \log l)$ and is significantly less than $\mathcal{O}(n)$. In contrast to the learning phase, the time complexity of the classification is reduced by the number of subsamples l and results for a binary tree in $\mathcal{O}(l)$ in the worst case and $\mathcal{O}(\log l)$ in the average case. Thus, the number of k trees for the time complexity is $\mathcal{O}(kl)$ in the worst case and $\mathcal{O}(k \log l)$ in the average case. For the Loda variant (2), the time complexity for the training phase is $\mathcal{O}(nkd^{-1/2})$. The complexity results mainly from the nested loops with the limits n and k. The use of "Very Sparse RP" [128] yields a speedup of $\sqrt{d}$ from which the factor $d^{-1/2}$ results.

	Time complexity		**Space complexity**
	Training	**Classification**	
iForest	$\mathcal{O}(kl \log l)$	$\mathcal{O}(k \log l)$	$\mathcal{O}(kl)$
Loda (1)	$\mathcal{O}(nkd^{-1/2})$	$\mathcal{O}(k(d^{-1/2} + b))$	$\mathcal{O}(k(d^{-1/2} + b))$
Loda (2)	$\mathcal{O}(nkd^{-1/2})$	$\mathcal{O}(kd^{-1/2})$	$\mathcal{O}(k(d^{-1/2} + b + l))$

Table 2.1: Time/space complexity of iForest and Loda (cf. [119]).

Combining Classifiers

Apart from an incident analysis (refer to Section 2.2 - aggregation, alert fusion), which combines the alarms of different detection measures (*algorithm external combination*), there are also mechanisms that can be used for consensus finding within an ML algorithm (*algorithm internal combination*). The combination of learners has been categorized by Bhuyan in [29] to ensemble-based techniques (algorithm internal) utilizing bagging, boosting and stacking [129] and fusion-based techniques (algorithm external) combining several disparate data sources at the data level, feature level or decision level. Sadighian in [27] is categorizing fusion approaches into "Winner-take-all" and "Weight-based" ones. With the former, the final decision over the outputs from various IDSs is made based on the decision of the IDS that has the highest measurement value, e.g., majority vote, weighted majority vote, behavior knowledge space, naive-Bayes combination, and Dempster-Shafer combination. The latter assign weights to each IDS as its importance indicator on the final decision which is then made based on the weighted sum of the measurement values of all the IDSs, e.g., using ANNs or weighted average.

The concept of ensemble-based approaches combining several "weak" classifiers in order to gain a "strong" one, for instance in terms of a higher predictive performance, is becoming more and more popular. With respect to network-based attacks, attack characteristics significantly differ from each other. Thus, it is a common practice to have different sets of features as well as ML algorithms to detect different types of attacks [30], since single IDSs cannot cover all types alone [130]. These methods weigh the individual outputs and combine them (ensemble) to obtain a better result. Loda, as well as iForest, is based on the principle of producing a strong classifier by combining multiple weak ones (trees/histograms). However, there exists further work exploiting this concept. Amudha et al. [131] are investigating bagging and boosting as two possible methods for ensemble learning methods. Bagging performs random sampling, whereas boosting performs sampling based on a continuously updated distribution. Hu et al. have introduced the AdaBoost algorithm in [132], which like Loda, generates a strong classifier from weak classifiers (decision stumps). Kitsune, as proposed in [115], uses an ensemble of simple ANNs (in particular Autoencoders) to distinguish between anomalies and normal behavior. The research of Mirsky et al. [115] is showing that Kitsune works comparably to offline anomaly detectors and uses few resources. All this research work shows that an ensemble of weak classifiers provides better results than individual classifiers and works at the same level as strong classifiers, while even preserving resources.

Kittler et al. [133] are presenting many basic considerations for the combination of classifiers. These include many rules, such as the product, sum, minimum, maximum and median rules, as well as majority voting. A surprising finding is that the comparatively restrictive sum rule even produced better results than other rules. Voting in general is used to generate a collective decision. The four main components of a voting algorithm are input data, output data, input votes and output votes. Exact and inexact in this context indicates whether input objects are regarded as inflexible values or flexible neighborhoods, i.e. whether discrete or non-discrete values exist. There are different types of voting algorithms, e.g., consensus and compromise voting. Compromises are mainly voting variants based on the median or mean. Presets and adaptives indicate whether weightings are set or can change over runtime. Other variants are called threshold and plurality. Threshold voting means that the output weight exceeds a value, where plurality identifies an output that has maximum support from the inputs. [134]

With consensus voting, an anomaly is only recognized if all classifiers recognize it. In majority voting, an anomaly is detected when the majority detects an anomaly. Consensus voting prefers false negatives compared to false positives [135]. Gao et al. [136] are describing the use of consensus voting for multiple atomic detectors to improve detection rates. Lin et al. [137] are describing a creditability-based weighted voting system that assesses the creditability of each anomaly detection algorithm. This is done by comparing the results of the algorithms with known results of the network trace, in particular the information of the confusion matrix parameters. Unfortunately this is not compatible with unsupervised learning. Thus, a way must be found to obtain information whether an anomaly has been correctly detected or not. This is difficult to be achieved with unsupervised learning as there is no such information available. Based on this comparison, the weightings of the individual algorithms are then determined. Giacinto et al. [138] were investigating different approaches of disparate classification to obtain a single result. They are judging the "Dynamic Classifier Selection" algorithm as the best one. It selects for each pattern the classifier that finds the correct classification, if such a classifier exists. Aburomman et al. [139] have been introducing several ways to combine different classifiers and are stating that voting-based systems are the common method. Errors introduced by one classifier can be corrected by another if all classifiers have a similar performance. If the reliability of each classifier can be estimated in advance, it is possible to increase the accuracy by weighted voting. Weighted voting can be used more generally than simple majority voting and is therefore useful in a broader context. If all weightings are set to 1, simple unweighted voting results. It is important to note that the classifiers must be sufficiently different, otherwise there will be no significant improvements.

Many recent publications are dealing with the use of ML algorithms for anomaly detection. In [140, 141, 142] different algorithms and methods are tested. In some cases, multi-stage methods are being presented. Several algorithms are concatenated to achieve better results. A disadvantage is that the computational complexity of this method is higher by the application of several algorithms than with single-staged ones. Therefore, this approach is less suitable in environments characterized by less available resources without any modification or combination with other methods like sampling. The doctoral thesis of Taylor [70] presents a hybrid automotive anomaly-based IDS with a two-staged detector. Special attention is paid to frequency and sequence-based detection, which are specified for their application in order to identify Controller Area Network (CAN) frames which deviate from their normal transmission frequency or from their order in transmission (sequence). A so called anomaly score is calculated for consensus finding of the different systems and as a decisive feature for alarm generation. The authors of [143, 144] proposed a lightweight IDS for wireless sensor networks based on the combination of the anomaly- and misuse-based technique to offer a high detection rate. The approach is integrated in a cluster-based topology, to reduce communication costs, which leads to improving the lifetime of the network. The incoming data is first provided to the faster signature-based component and, if indicated abnormal, provided to the anomaly-based SVM. A decision making model combines then the outputs of both techniques, determines whether an intrusion occurred and classifies the type of the attack. The incident is then reported to an administrator for supervision. Guo et al. are presenting a two-staged hybrid approach in [145] that deploys an anomaly detection component in the first stage and its output in a second stage either forwarded to a second anomaly detection component (in the abnormal case of stage 1) or forwarded to a misuse detection component (in the normal case of stage 1). The misuse-based component is able to classify between an attack or not and the anomaly-based component between normal and abnormal connections. Since misuse-based techniques are typically less complex than anomaly-based ones a better approach would be to apply the misuse-based component in stage 1 similarly to the work in [146]. Thus, static checks are used which correspond to misuse-based (specification-based) detection by applying simple rules based on known communication matrices used in the automotive sector (CAN message catalogue). Those filter out inappropriate communication, e.g., exceeding payload values in a first place before features are extracted for a common basis to apply anomaly-based ML algorithms. A simple anomaly analyzer evaluates the outputs of, e.g., recurrent ANNs, One-Class SVM (OCSVM) and Loda in order to filter out false positives before logging detected anomalies.

Maglaras et al. are proposing IT-OCSVM in [147], a distributed intrusion detection system in a Supervisory Control and Data Acquisition network characterized by a three layer hierarchical abstraction into field, operation network and IT-network. It is using a central OCSVM and a cluster of automatically produced ones, one for each source that induces significant traffic in the system, an embedded ensemble mechanism, an aggregation method and a k-means clustering procedure that categorizes aggregated alerts using IDMEF messages. The detection functionality of the IT-OCSVM is composed of preprocessing (feature extraction from raw data containing all forms: continuous, discrete and symbolic mapping to numeric-values), the selection of the most appropriate features (divided into content and time-based features), the creation of cluster of OCSVM models (trained on discrete sources), testing of the traffic data set (containing malicious attacks), the ensemble of classifiers (combining the output of the different OCSVM modules using mean majority voting), social analysis (technique using Spearman rank correlation coefficient to add weight to alerts produced from different sources, e.g., the difference between mainly used protocols during the normal and abnormal operation of a node), the fusion of information/alarms (multiple anomaly outcomes are gathered and classified in terms of importance by k-mean clustering; groups alerts per source node and gives final scores to aggregated alerts based on the initial values and the number of similar initial alerts) and communication of the mechanism (IDMEF file exchange for alerts in terms of, e.g., importance, position, time). The ensemble based mechanism for the outcome of the central and the split OCSVMs is computed with $q_e(i,j) = \sum_{n=1}^{N} w_i d_t(i,j)$ where $d_t(i,j)$ is the outcome of each classifier n for the sample data i originating from node j with the assigned weight w_i.

2.1.4 IDS Evaluation Metrics

According to [29], metrics for IDS evaluation can be divided into data quality (quality, reliability, validity, completeness of, e.g., data source, selection of samples, sample size, time of data), correctness and efficiency as shown in the taxonomy of Figure 2.5.

Evaluation metrics to compare performance (efficiency) and effectiveness (correctness) can be generally classified into cost-based metrics, information-theoretical metrics [148], binary classification and resulting from binary classification, Receiver Operating Characteristic (ROC) [27, 149]. Efficiency deals with the resources needed by the system executing the IDS including, e.g., CPU cycles or memory demands. Further the timeliness is a metric that defines how quickly a response is performed after an incident has been detected. Correctness represents the ability of the system to distinguish between malicious and non-malicious behavior (classification performance) by measures such as ROC-curve, Area Under the ROC Curve (AUC), precision, recall, F-measure, confusion

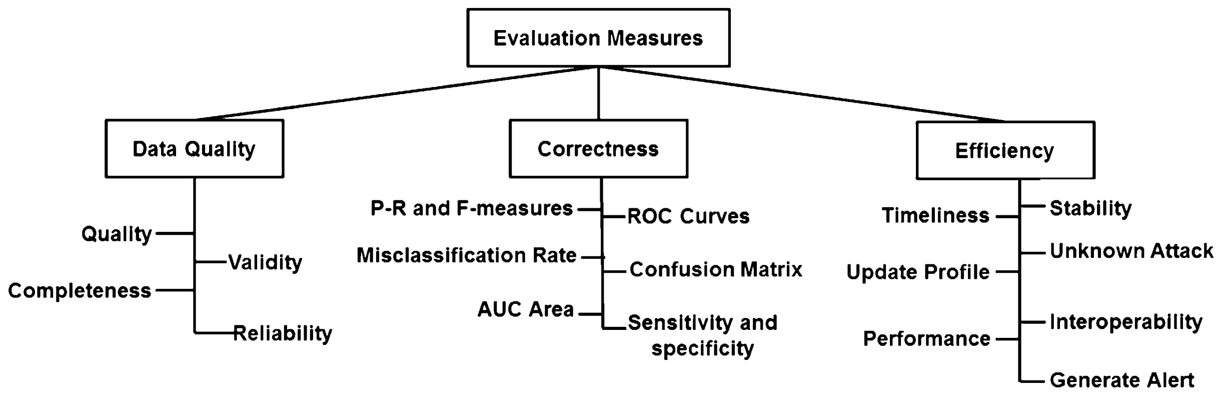


Figure 2.5: Taxonomy of evaluation measures [29].

matrix, misclassification rate, sensitivity, and specificity. Cost-based metrics assign a cost measure to weight false positive and false negative rate to consider a trade-off between the cost of a damage by a successful attack and the costs for impacts of false alarms. Especially for ML-based IDSs, a high detection rate is essential. However, when measuring the accuracy of IDSs, particularly for the problem of statistical classification, different characteristic values are used. A so-called confusion matrix is utilized to compare the performance of such algorithms. The focus of the performance lies on the predictive power of a model and not on the speed the model performs classifications into normal or abnormal classes (binary classification). The confusion matrix is represented by Table 2.2, in which each row represents the instances of a predicted class, while each column represents an actual class.

	Actual Non-Anomaly	**Actual Anomaly**
Predicted Non-Anomaly	True Negative (TN)	False Negative (FN)
Predicted Anomaly	False Positive (FP)	True Positive (TP)

Table 2.2: Confusion matrix for IDS evaluation.

Where:

TN: normal event/behavior classified as a normal event/behavior
FN: intrusion/anomaly classified as a normal event/behavior
FP: normal event/behavior classified as an intrusion/anomaly
TP: intrusion/anomaly classified as an intrusion/anomaly

Many other characteristic values (sensitivity, specificity, positive/negative predictive value, F-Score, Matthews correlation coefficient, etc. [27, 29, 150]) can be derived from the parameters of Table 2.2. Two examples, the False Positive Rate (FPR) and the True Positive Rate (TPR) computed by $FPR = \frac{FP}{FP+TN}$ and $TPR = \frac{TP}{TP+FN}$ are used to derive the ROC metric. The ROC-curve is a visual representation of the diagnostic ability of a binary classifier. An example of a ROC-curve is shown in Figure 2.6 in which the blue curve represents a random classifier whose output is completely random. The curve of a well-performing IDS is above the blue curve. This means that the top left corner of the plot is the “ideal” point with a FPR of zero, and a TPR of one. This is not very realistic but it does mean that a larger AUC is usually better. The “steepness” of ROC-curves is also important, since it is ideal to maximize the TPR while minimizing the FPR. With the help of such metrics results of different anomaly detection algorithms can be reliably compared or the anomaly detection algorithm under test can be optimized.

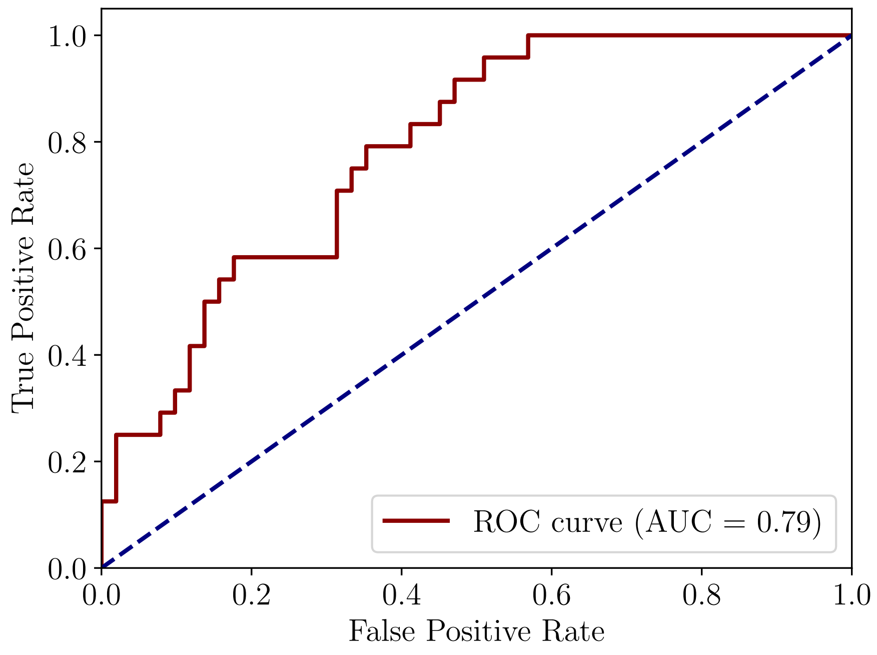


Figure 2.6: Example of an ROC-curve.

Further performance metrics and capabilities that typically characterize IDSs are listed in the following. However, there is plenty more available in the literature [26].

- **Resource requirements (efficiency)**: Resources needed to be allocated by the system including memory usage, CPU load/cycles and disk space.
- **Overhead:** Computation and communication overhead - especially considering collaborative IDSs, a reasonable overhead of communication effort and computation must be achieved.
- **Throughput**: This metric defines the level of traffic up to which the IDS performs without dropping any data instance, e.g., a packet.
- **Timeliness**: Average/maximal time between an intrusion's occurrence and its reporting.
- **Resilience**: States how resistant an IDS is to an attacker's attempt to disrupt the correct operation of the IDS or malfunctions of the component.
- **Ability to correlate event**: States how well an IDS correlates attack events from, e.g., routers, firewalls, or application logs. This already refers to incident analysis functionality.
- **Detection of "zero-day" intrusions**
- **Capacity verification for NIDSs**: Ability of inspection into deeper levels of, e.g., network packets.
- **Stress Handling**: The point of breakdown is defined as the level of network or host traffic that results in a shutdown or malfunction of IDSs.

- **Depth/Coverage of detection capability**: It is defined as the number of attack signature patterns and/or behavior models known to it. (What attacks can be detected?)
- **Reliability of attack detection**: It is defined as the ratio of FPs to total alarms raised - accuracy.
- **Error reporting and recovery**: The ability of an IDS to correctly report errors and recover from them.
- **Self-configuration:** Ability to automatically adjust itself without manual intervention.
- **Interaction capability with other systems**: The ability of an IDS to interact with other systems such as firewalls or anti-virus systems.
- **Attack analysis/identification**: It is the ability to report the extent of damage and compromise due to intrusions and to identify an attack based on common names or exploits (assumes 100% confidence).

2.2 Incident Analysis

Many works, e.g. [151], make the assumption for response planning that each raised alarm (output of an IDS) is treated as one attack (100% confidence of the alerts). However, this might be true when applying misuse-based IDSs which commonly have a high TP and low FN rate. In order to detect new attacks with high accuracy, the input of various detection mechanisms (including anomaly-based ones) might be important but operating for instance in safety-critical environments, cross-evaluation or plausibility checks of various inputs is essential before performing a reaction in order to reduce FPs to a minimum. A comparison of supervised, semi-supervised and unsupervised learning methods for anomaly-based IDSs has been examined in [152], each having its particular strengths but their detection capability differ significantly. Not only this circumstance but also (1) the handling of a massive amount of alerts from various applied detection sources is a requirement towards incident analysis and (2) the safe selection and execution of a following incident response measure. Hence, according to [153], IDSs are not enough to detect complex attacks over a network. Even they are able to detect some basic attacks, e.g., fabrication and suspension attacks, they fail to detect more sophisticated ones such as the masquerade attack [154]. An intelligent incident (alert) analysis is therefore necessary in order to

- gain knowledge of multiple detection sources by using a unified format,
- identify the root cause of an incident,
- recognize pattern between the alerts and historic events,
- reduce the number of alerts, cluster and correlate them in order to prepare the essential information for an administrator, and
- predict the propagation of malicious action (in this context referred to malware and cyber attacks).

Alert analysis techniques and methods help to manage and diagnose, e.g., to deal with (a huge amount of) alerts gathered from (various) incident detection components by filtering out alerts, grouping and correlating them or prioritize important ones. Alerts typically incorporate information (alert feature) regarding the creation/detection time of an attack or suspicious event, its description and severity etc., which is defined by the alert format used. According to [29], alert management contains three major components: *alert correlation* (AC), *alert merging* (aggregation) and *alert clustering*. For the sake of generalization, alert analysis can be broken down into three main fields [28] as shown in Figure 2.7: *pre-processing* (e.g., alert normalization, redundancy elimination, FP reduction), *processing* (e.g., AC techniques, new attack scenario detection) and *post-processing* (e.g., alert prioritization metrics and intention recognition, prediction). For further literature to each component in Figure 2.7 refer to [28]. It is noted that the boundaries of the categorization into pre-processing, processing and post-processing might become blurred since for instance a system incorporating AC might feature prediction capability as a processing and only visualization as post-processing functionality.

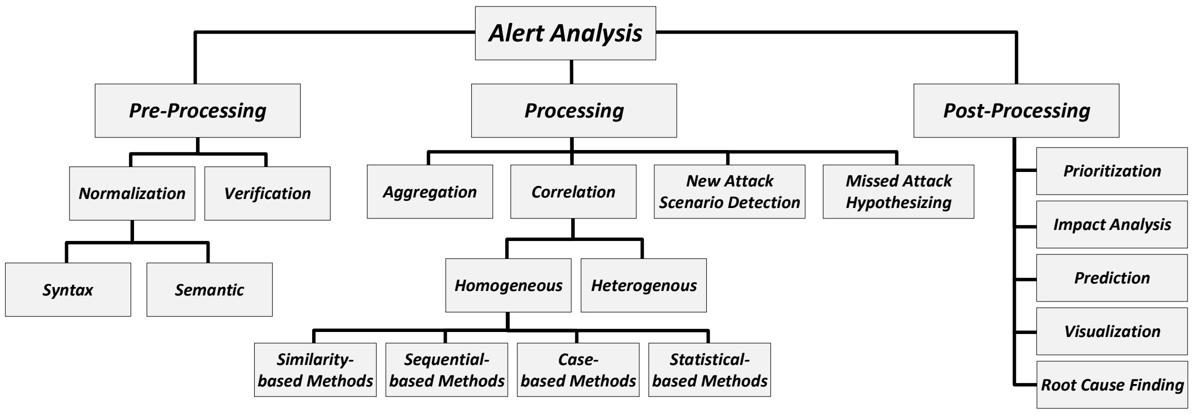


Figure 2.7: Taxonomy of alert analysis fields (cf. [28]).

2.2.1 Pre-Processing

Pre-processing is the process performed before an attack scenario construction. It is composed of *normalization* and *verification* which are fundamental steps before, e.g., a correlation can be accomplished.

Normalization

Security Information and Event Management (SIEM) systems are designed to help network administrators, typically working in a Security Operations Center (SOC), to manage security tools, e.g., IDSs operating in the network infrastructure. Typically, the work of SIEM systems is to aggregate, standardize and correlate alarms. Today, SIEM systems mainly use internal proprietary formats to describe alerts. Most of those are inspired of log management tools such as Splunk[2] and based on syslog with a simple but limited key-value paradigm. However, the heterogeneity and diversity of existing security tools pose a significant challenge to SIEM and SOC due to the multitude and diversity of alert sources demanding the need for a common format. Normalization is used for the translation of a raw alert into a standardized alert format, e.g., IDMEF as proposed in [29]. With the growth of semantic technology and the inability of, e.g., IDMEF only presenting a syntax for formatting, new studies try to introduce new data models for handling

[2]`https://www.splunk.com` (accessed on 05 September 2021)

alerts semantically in order to provide a robust solution [28]. A structured overview of various existing exchange formats ("describes a structure for the processing, storage, or display of data" [155]) and protocols ("a set of rules defining how to interconnect network devices and establish a channel to transmit network datagrams, representing exchange formats, across a computer network" [156, 157]) targeted to the IDS domain is provided in [155, 158]. Application domains of other exchange formats in IT security is depicted in Figure 2.8. For a comprehensive overview of standardization attempts for security automation refer to [7].

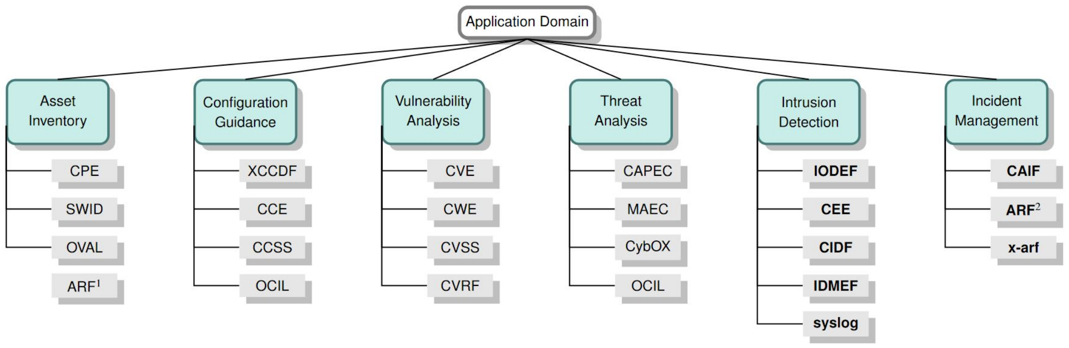
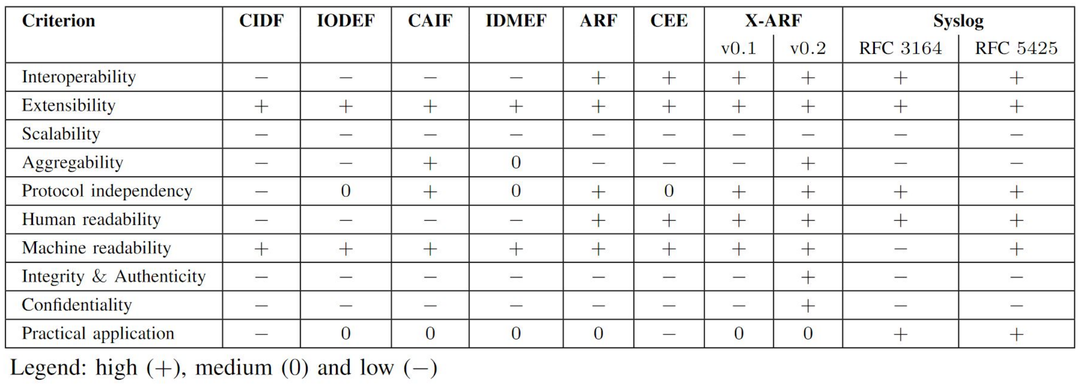


Figure 2.8: Application domains of exchange formats [155].

Koch et al. are stating in [158] several technical requirements for data formats and exchange procedures for sharing information of interest to IDSs, response systems and to management systems including vendor independence, near real-time capability and scalability, e.g., for decentralized approaches. Message exchange protocols discussed are (1) proprietary protocols, (2) Simple Network Management Protocol (SNMP), (3) Common Intrusion Detection Framework (CIDF), (4) Intrusion Detection Message Exchange Format (IDMEF) including IDMEF Communication Protocol (IDP) and the newer and recommended Intrusion Detection eXchange Protocol (IDXP), (5) Incident Object Description and Exchange Format (IODEF), (6) Format for Incident Report Exchange (FINE) and (7) Intruder Detection and Isolation Protocol (IDIP). However, the authors of [155] are stating that Koch et al. do not differentiate between a high-level description of functional requirements, an exchange format or an exchange protocol. Thus, Steinberger et al. in [155] have reviewed 10 exchange formats and 7 exchange protocols that can be used to share security event related information in context of intrusion detection and incident handling with respect to their use-case scenario. They further provided *inter alia* an assessment of the exchange formats for the interoperability and a qualitative evaluation and comparison of the formats and protocols in context of high-speed networks. Apart from the aforementioned formats and protocols, the authors introduce further formats (8) Common Announcement Interchange Format (CAIF), (9) Common Event Expression (CEE), (10) Messaging Abuse Reporting Format (ARF), (11) x-arf, (12) Syslog Message Format - IETF RFC 3164 and further protocols (13) Real-time Inter-network Defense (RID), (14) Extensible Messaging and Presence Protocol (XMPP), (15) CEE Log Transport Protocol (CLT), (16) Simple Mail Transfer Protocol (SMTP) and (17) Syslog protocol. The results of the evaluation carried out by the authors regarding the discussed exchange formats and exchange protocols are depicted in Figures 2.9 and 2.10.

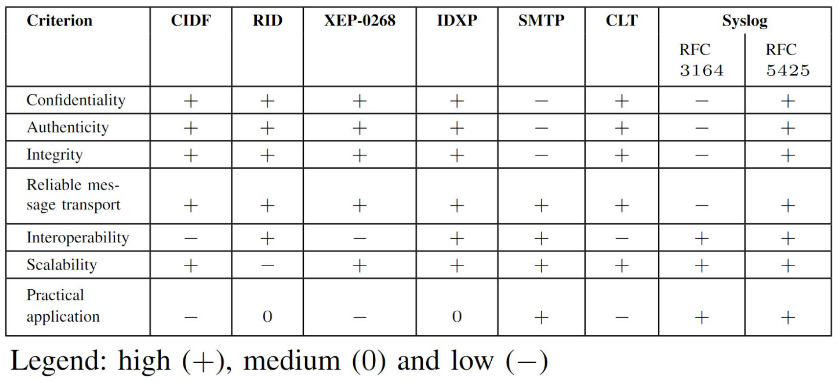
Even if Steinberger et al. have provided a comprehensive survey, they did not mention IDIP which seems a promising candidate for automated incident response execution and a few other recent formats, e.g., the JavaScript Object Notation (JSON) serialized



Criterion	CIDF	IODEF	CAIF	IDMEF	ARF	CEE	X-ARF		Syslog	
							v0.1	v0.2	RFC 3164	RFC 5425
Interoperability	−	−	−	−	+	+	+	+	+	+
Extensibility	+	+	+	+	+	+	+	+	+	+
Scalability	−	−	−	−	−	−	−	−	−	−
Aggregability	−	−	+	0	−	−	−	+	−	−
Protocol independency	−	0	+	0	+	0	+	+	+	+
Human readability	−	−	−	−	+	+	+	+	+	+
Machine readability	+	+	+	+	+	+	+	+	−	+
Integrity & Authenticity	−	−	−	−	−	−	−	+	−	−
Confidentiality	−	−	−	−	−	−	−	+	−	−
Practical application	−	0	0	0	0	−	0	0	+	+

Legend: high (+), medium (0) and low (−)

Figure 2.9: Evaluation summary of exchange formats [155].



Criterion	CIDF	RID	XEP-0268	IDXP	SMTP	CLT	Syslog	
							RFC 3164	RFC 5425
Confidentiality	+	+	+	+	−	+	−	+
Authenticity	+	+	+	+	−	+	−	+
Integrity	+	+	+	+	−	+	−	+
Reliable message transport	+	+	+	+	+	+	−	+
Interoperability	−	+	−	+	+	−	+	+
Scalability	+	−	+	+	+	+	+	+
Practical application	−	0	−	0	+	−	+	+

Legend: high (+), medium (0) and low (−)

Figure 2.10: Evaluation summary of exchange protocols [155].

IDEA. According to [158], IDMEF and IDXP can have a likewise effect on research and deployment of intrusion detection technology what HTML and HTTP did for the Internet growth. However, Steinberger et al. concluded that it is still a challenge to find a standardized exchange format and protocol that is thoroughly validated and tested in full scale of industry trials [155].

A significant drawback of IDMEF is the usage of XML which makes it easier to develop and deploy, but it comes with a performance cost. Due to the structure of XML, the data encoded is typically very large (for instance in comparison to JSON), mainly because of XML's closing tags. Therefore parsing XML messages is still a relatively slow task today [158]. The Warden [159] and MISP [160] projects also bring further interesting approaches. Warden uses a client-server architecture and offers two different types of clients: the receiving-client and the sending-client. With the sending-client events can be sent to the Warden server, with the receiving-client events received at the Warden server are also sent to the client side. The Warden team designed the IDEA format for the transmission of events. It is based on already existing data formats for the transmission of security relevant information (mainly IDMEF) and aims to eliminate weaknesses that these data formats bring to their system. For example, messages in the AbuseHelper format consist of any number of keys and associated values which makes this format easily extensible but can lead to inconsistencies in automated message processing. The IDEA format should lie between the complexity and depth of IDMEF and the loose structure of the AbuseHelper format [53].

Verification
Verification or alert validation is necessary since many problems can occur, such as misconfiguration, low accuracy of applied detection methods, and lack of attention to contextual information during the alerts analysis [161]. Verification tries to recognize if any changes have taken place in the system monitored. If any new device has been installed in the system, it may produce irrelevant alerts. Thus, verification helps to filter out alerts with low-interest, irrelevant alerts or some known FPs. Validation can also refer to post-processing. In [162] an approach is presented that defines three major dimensions to recognize attacks and identify the target by the validation of AC systems: prioritization (assigning weight to each alert based on the probability that it may indicate an attack and dispensation of the target), multi-step correlation (alert correlator can reconstruct a multi-step attack scenario by correlating different individual attack steps which is important to infer attack intention and their effective response) and multi-sensor correlation (combines multiple alerts received from different sensors to create an overall picture of the system) [29].

2.2.2 Processing

Processing mainly copes with the attack scenario construction and contains *aggregation*, *correlation*, *new attack scenario detection* and *missed attack hypothesizing*. The outcome of processing serves as a basis, for instance, to recognize the intention of an adversary which is performed in post-processing.

Aggregation
Aggregation, or alert merging, respectively alert fusion, refers to the reduction of alerts by combining multiple (and possibly heterogeneous sources) of them to yield a more precise and descriptive result of IDSs. Alert fusion is a special case (sometimes a sub-process) of AC that collects and analyzes alerts independently generated from the same potentially malicious event by different IDSs, in order to make an appropriate final decision about the event [163]. The main idea behind aggregation is to provide clustering and grouping similar alerts based on their features in order to eradicate duplicates having the same root cause. Especially by the application of distributed or decentralized IDSs, the exploitation of alert fusion enhances the overall detection efficiency, improves detection accuracy, fault-tolerance, stability and reliability of IDSs and helps to make appropriate decisions [27]. The process is closely related to Subsection 2.1.3 (algorithm external combination). Finally, with the aggregation of duplicate and redundant alerts, the uninteresting ones are eliminated and a big view of the security situation is provided by fusing the same events [28]. Weng et al. for instance are proposing in [164] an alarm reduction for distributed IDSs using edge computing, exploiting the strengths of cloud computation while offloading only a limited amount of information by processing data at the edge for shorter response time and energy saving. Their proposed framework consists of three layers which are structured hierarchically from the infrastructure to the cloud side: IDS layer (performs traffic inspection and false alarm reduction by exploiting the strengths of distributed IDSs), edge layer (aggregate data from IDS layer and select most appropriate ML algorithm from a predefined pool), cloud layer (providing sufficient computation resources for deploying intelligent alarm filters).

Correlation

The homogeneous AC refers to a case that each of the monitoring devices like IDSs examine the same type of data, whereas in the other one, various deployed sensors examine different types of events and raw data sources [27]. Salah et al. [165] and Hubballi et al. [166] are providing a comprehensive overview in the field of AC which is defined as a measure of the relation between multiple alarms such that new meanings can be assigned to them. Thus, not only the verification of the alerts' validity can be verified but also complex attack scenarios can be identified. The AC process comprises different approaches available in the literature [166, 167, 168, 169, 170, 171] and has been classified in [29, 172, 173]. However, it must be noted that due to different types of attacks with different sophistication levels, there might be limitations in the handling of the multitude of alerts with equal importance. Hence, it might not be sufficient to rely on a single component but rather on different ones to concentrate on various aspects of the general correlation problem. Several factors that can be used to assess correlation algorithms are stated in [173] composed of algorithm capability (e.g., alert verification, attack sequence detection), algorithm accuracy, algorithm computation power, required knowledge base and algorithm extendibility and flexibility.

AC techniques try to reconstruct the attack scenarios from alerts which may exhibit an attack that involves multiple stages in compromising a network [29]. A taxonomy of AC techniques is provided by [165] (Figure 2.12) including types of applications and architectures for the correlation process. In [165, 174] AC architectures are categorized into *centralized* (data collection performed locally and reported as alerts to a central server executing correlation analysis), *distributed* (alerts or high-level meta-alerts are exchanged, aggregated, and correlated in a completely cooperative and distributed fashion between equally weighted agents; communication is performed using a peer-to-peer protocol) and *hierarchical* (referred to as decentralized by the author of this work; separated correlation into hierarchical layers of local analysis, regional analysis and global analysis) as shown in Figure 2.11.

The number of data sources - single or multiple - with respect to Figure 2.12 state that the AC method is sourced either by a single data source, e.g., a database or a single security measure, or by a collaborative set which allows a more precise and coherent view about the observed system. The authors further subdivide the correlation methods into *similarity*, *sequence* and *case-based methods*, whereas the authors of [29, 173] are introducing - apart from similarity-based and case-based (referred to as *knowledge-based*) - *statistics-based* methods and *hybrid* approaches (both not shown in Figure 2.12). However, it must be noted that the categorization is not completely precise and methods from each class may show similar behavior or rely on comparable mechanisms.

Similarity-based methods correlate alerts based on similarities of selected features such as the source/destination IP address, time or protocol information and are designed to reduce the total number of alarms through aggregation or clustering [165]. Therefore, they can be further subdivided into attribute-based (similarities between attributes/features) and temporal-based (temporal time relations) techniques. Mirheidari et al. categorize similarity-based algorithms into ones that are based on simple rules, hierarchical rules or ML [173]. Usually, similarity functions are defined for the individual alarm attributes and applied to two alarms. Together with appropriate attribute weightings, a similarity value is determined that reflects how well two alarms match. These values are useful because it can be assumed that alarms with high similarity can be part of the same attack or suspicious event. The resulting similarity between two alerts can

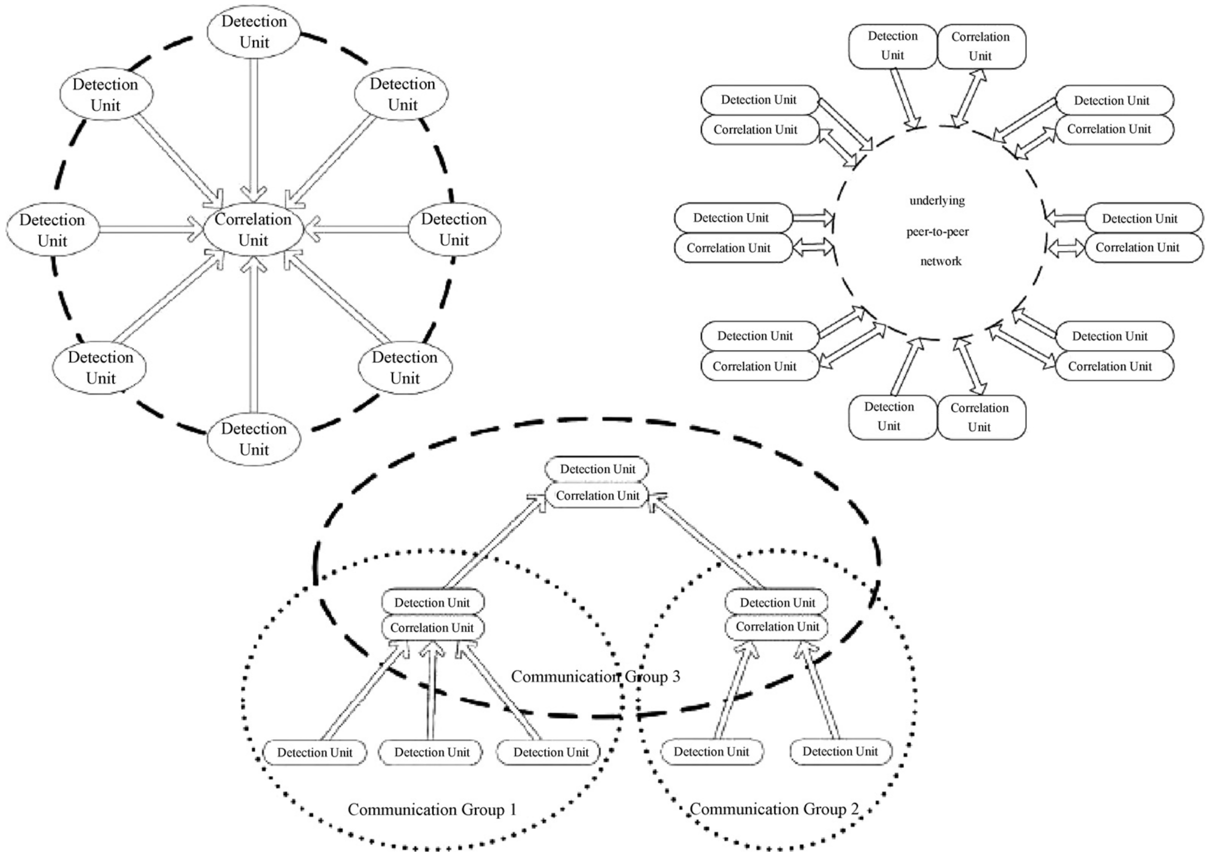


Figure 2.11: Different alert correlation architectures - centralized (left), distributed (right) and hierarchical (bottom) [174].

be calculated according to Equation 2.8 in which X^i is the candidate meta alert i for matching; Y is the new alert; j is the index over the alert features; E_j is the expectation of similarity for feature j; and X_j as well as Y_j are the values for feature j in alerts X and Y [29]. Further, attribute-based similarity measures can be computed using metrics such as Euclidean, Mahalanobis, Minkowski and/or Manhatten distance functions [165]. Temporal-based methods, in order to find temporal relationships between alerts, typically rely on time-windows such that only alerts observed in a short time are to be correlated. A benefit is to reduce the number of alerts generated by the same event in a certain period of time [165].

Valdes and Skinner presented a probabilistic approach in [175] that falls into the area of similarity-based methods which extends the idea of multi-sensor data fusion for AC. This method shall find its use in the handling of alarms generated by heterogeneous sensors. The correlation algorithm expects features from reported alarms in a self-defined alert template. For comparable features, suitable similarity functions are defined, whereby the features of incoming alarms will be compared with a list of already existing meta-alerts and result in values between 0 (mismatch) and 1 (exact match). The similarity value is composed of a weighted average of the features, but if one does not exceed the minimum similarity threshold, the complete alarm is not considered similar. The alarm is correlated with the most similar meta-alert, otherwise the alarm forms a new meta-alert thread. The alert fusion considers feature overlap (new and existing alerts may share some common features), feature similarity (value of similarity scores of same type of feature), minimum similarity and the expectation of similarity. Since sensors can classify attacks differently,

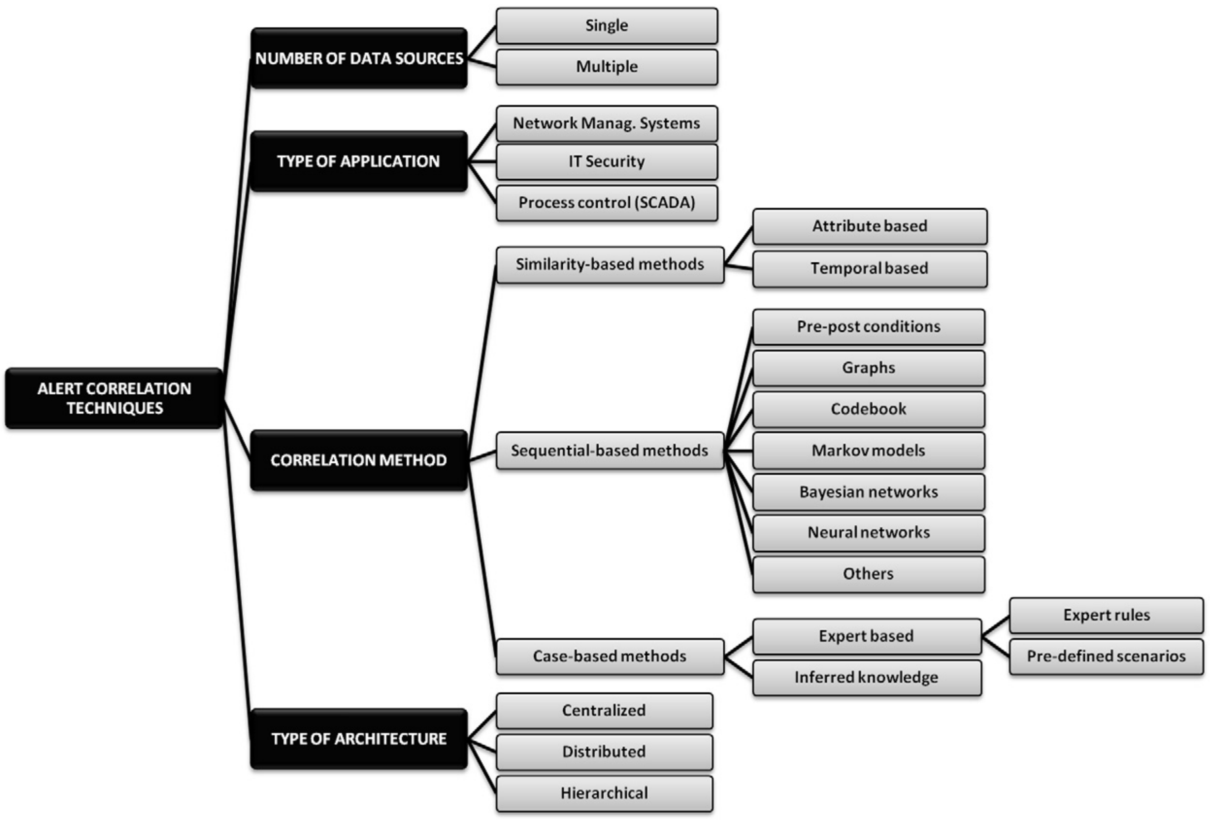


Figure 2.12: Taxonomy of alert correlation techniques [165].

$$Sim(X^i, Y) = \frac{\sum_j E_j Sim(X^i_j, Y_j)}{\sum_j E_j} \tag{2.8}$$

a matrix of similarities between attack classes is used to compare them. The correlator checks whether the sensor identification and the incident class match exactly. Then it checks if all overlapping features at least match the minimal similarity and calculates the similarity values. If this is the case, the overall similarity is calculated.

Zhuang et al. [176] rely on the work of [175] and are extending it with a rule-based knowledge base. A correlation system architecture consisting of alert collection, alert verification, data fusion and correlation is proposed and explained. The correlation process takes the features IP addresses, port numbers and time stamps into account. Alarms from different sources are collected by the alert collection module and the features are passed on to the alert verification. In, order to support heterogeneous sensors, different plugins are used which process the respective alarms, e.g., Snort[3] alarm $\rightarrow$ Snort plugin. The alarm information is additionally appended with information about the plugin that processed it. The transmitted alarms are checked for FPs by the alert verification based on information regarding the network topology as well as the hosts and a confidence value is determined. Legitimate alarms are then grouped by the data fusion component using similarity functions. Using the knowledge base, the last step in the correlation process is to classify the alarms into an attack scenario, which is also referred to as a schema. A schema consists of a number of rules which use the information from the previous steps such as the confidence value to describe the state of the monitored system. The description of a schema by the rules resembles a tree structure and attacks are detected

[3]`https://www.snort.org/` (accessed on 05 September 2021)

if the rules of a schema are met from root to leaf. Similarity-based methods prove to be suitable for alert clustering and reducing the number of alarms as well as discovering simple attacks with a small number of features. In addition, they are easy to implement and work well for a known set of alerts with a known feature set, but find their weakness in recognizing causal and other statistical relationships between alarms, limited to known alerts only and incapable of identifying complicated attacks [29, 165, 174].

Statistics-based methods, according to [27, 29, 173], rely on statistical causality analysis to correlate alerts that are related to some specific attacks in order to reconstruct attack scenarios. Similar attacks have similar statistical attributes, and so, they can be categorized easily corresponding to different attack stages. Statistical computation can be categorized into (1) detection of repeated and repetition patterns; (2) estimation of causal relationships between alerts, predicting next alert occurrence, and detecting attacks; and (3) combining reliability by mixing completely similar alerts. Since these methods are based on statistics, pre-defined knowledge about attack scenarios is not required. However, they lack in discovering dependencies, structural cause relationships between alerts and it is difficult to estimate correlation parameters. Exemplary work using statistics-based methods is provided in [171, 177].

Sequence-based methods attempt to determine causal relationships between alarms using defined *preconditions* and *consequences*. This is done by describing events or states, which are necessary for an attack step, as preconditions and describing the respective effects or states, which result from successful execution of this attack step, as consequences. Thus, they are not limited to known attacks but the correlation may result in many false correlations due to the misconfiguration of the relationships mainly represented as logical operators such as AND/OR or the inadequate quality of sensors. According to [165], they can further be subdivided into, e.g., pre/post conditions (using the concept of hyper-/meta-alerts as a tuple of prerequisites and consequences), graphs (DAGs in which the set of nodes represent alarms and the edges represent the temporal relation), codebook (matrix representation of alerts (rows) and problem symptoms as columns), Markov models (stochastic model composed of discrete states and a matrix of state transition probabilities trained by sequence of events), Bayesian networks (probabilistic DAG model of alerts representing their probabilistic inference), ANNs (a collection of connected units or nodes called neurons working interconnected to perform AC).

Ning et al. [178] are presenting a possibility to reconstruct attack scenarios based on prerequisites and consequences by correlating alarms. For the representation of prerequisites and consequences, the use of predicates is suggested which can be linked by logical combinations if necessary. The prerequisites and consequences of an event are represented by so called *hyper alerts* which encode the knowledge about an attack. A hyper alert consists of three components: fact, prerequisite and consequence. Fact states what information is reported with the alert. This consists of a set of attributes, each with its own range of possible values. Prerequisite is a logical combination of predicates whose variables occur in fact and specify which criteria must be met for an attack to be successful. Consequence describes the effects if the attack is actually successful. A correlation occurs when the consequences of one hyper alert fulfill the prerequisites of another one. This is also referred to intuitively as a “prepares-for” relationship, since two hyper alerts h_1 and h_2 are correlated if h_1 is prepared for the following hyper alert h_2. By describing a hyper alert type, hyper alert instances are created when events occur that are described in the prerequisites of the hyper alert types. From the prepare-for relationship and thus the relationships between hyper alerts, the authors create an Alert Correlation Graph (ACG) based on hyper alerts to represent attack scenarios step by step. The ACGs consist of a number of nodes (hyper alerts) and edges, which represent the connection between nodes and thus the relation. The result is a DAG that corresponds to the detected scenario. Zhu and Ghorbani also use the concept of hyper alerts to determine attack scenarios in [179]. Their correlation engine is based on ANNs using Multilayer Perceptrons (MLP) and SVM. MLP and SVM learn the desired behavior with one training set, using a total of 6 features. The networks are used to decide whether two alarms should be correlated and, if so, provide a correlation value between 0 and 1. Determined correlation values between two alarms are stored in an AC matrix and later updated by the correlation engine. In addition, alarms with the best matching alarms are grouped into hyper-alerts using thresholds. Based on this, graphs are created to show the attacker's approach.

The authors of [180] present an approach to reconstruct attack scenarios from alarms coming from heterogeneous sensors. The process is divided into two steps: semantic-based alert clustering and causality-based attack analysis. For the semantic analysis of alarms, an ontology is introduced using classes and explicitly defined relationships that contain relevant information regarding the intrusion detection environment. The emerging relationships between intrusion alerts can be used to determine how relevant semantic alarms are. Semantically related alarms are converted into ACGs. The ACG is a non-directed weighted graph with nodes representing alarms and edges representing the relationships between alarms based on the previously determined semantic relevance with numerical values ($[0, 1]$). Groups of nodes are determined from the created ACG, whereby all nodes of a group are connected to each other by edges. In graph theory such groups are called cliques. Here, cliques are regarded as candidate attack scenarios. Based on the time stamp information from the alarms, the sequence of the individual steps of the attack is determined. If the dependencies between preconditions and consequences are clearly defined, sequence-based methods are well suited for detecting known but also unknown attack scenarios allowing to detect zero-day and multi-step attacks. However, the weak point here is that individual steps of an attack can be overlooked by the applied IDSs which would not lead to the fulfillment of a consequence (refer also to hypothesizing). Furthermore, in a network with heterogeneous sensors describing an attack differently but with the same meaning, the alarms must be described for each sensor. A combined approach with a similarity-based method for clustering and normalizing alarms would be well suited here.

Case-/Knowledge-based methods, also referred to as AC based on known scenarios, usually rely on well-described attack definitions in a knowledge base. The knowledge can be based on either prerequisites and consequences, attack scenarios or case-based reasoning which is defined as the process of solving new problems based on the solutions of similar past problems [173, 181]. Those are typically described by rules such as in [176] or a correlation language such as LAMBDA [182], STATL [183] or CAML [184]. Methods from correlation languages, from data mining or ML search the knowledge base for the best fitting case and update it if the case is successfully solved. According to [165], examples for case matching algorithms are nearest neighbor, inductive, and knowledge-based indexing and case-based methods can be further subdivided into expert-based (knowledge base is build by human using expert rules or predefined scenarios) and inferred knowledge (symbolic classification rules are automatically constructed from some training cases - alerts or meta-alerts whose classification is known - by ML). According to [29], the main drawbacks of these methods are the manual definition of prerequisites, the limitation to deal with new pattern, the difficulty in updating the correlation knowledge, the inability to discover structure and statistical relationships and their impracticability for the use in large scale or real time environments due high computational expense.

To exploit the benefits of the different techniques, hybrid approaches are often proposed. Ahmadinejad et al. [185], for instance, are presenting a model consisting of two modules for alarm correlation. Received alerts are passed to the first module to check if it can be placed in an already known attack scenario. The analysis is done using attack graphs, here called "queue graph", by checking for incoming alarms whether a prepares-for relationship can be identified with already existing alarms. Using a depth search in the queue graph, alarms or steps of a known attack can be found that have been overlooked by the IDS. A threshold value is used to decide whether this belongs to a known attack scenario with missing detected steps or whether a potentially unknown scenario exists. Alarms that cannot be classified in the queue graph by the first module are forwarded to the second module of the model for similarity-based analysis. Selected features are taken into account and a similarity vector with values $[0, 1]$ is formed on the basis of similarity functions. Based on the similarity vector and the existing hyper alert, a "CorrelationThreshold" is used to decide with which hyper alert the new alarm should be correlated. The authors of [186] propose a hybrid approach that is based on hierarchical clustering composed of an offline correlator (aggregates historical data, extracts attack strategy graphs and uses hierarchical clustering to group similar attack strategy graphs – the attack characteristics of each cluster is then identified) and an online correlator (generates hyper-alerts which contain useful attributes for security analysts. Hyper-alerts are composed of different low-level alerts and are updated in real-time as the upcoming low-level alerts are triggered. Hyper-alerts are associated to the clusters generated by the offline correlator in order to understand the characteristic of an attack). Since the approach correlates historical alerts into clusters using data mining techniques and associates upcoming alerts to these clusters in real time, an efficient security alert analysis technique could be achieved and useful information from historical data can be discovered to assist the analysis of new alerts that reduces the time between the detection and response to an incident.

New Attack Scenario Detection

New attack strategy detection copes with the discovery of novel attack scenarios from the sequence of events and tries to overcome limitations of correlation methods that are typically unable to extract unknown malicious behavior of intruders [187]. This includes finding new multi-step attack scenarios from the analysis of alerts to which no "template" is available. Simple changes to those templates (or attack patterns) might result in a failure of attack detection. Especially, with the application of anomaly-based IDSs which output either simple classification values - anomaly or not - or scoring values representing only indicators of malicious activity, it is more difficult to analyze alerts than with misuse-based ones. Those are able to interpret events/indicators due to their existing knowledge base and provide more detailed information. Soleimani and Ghorbani are presenting an approach in [188] that aggregates alerts and generates episodes which represent a sequence or a partially ordered collection of events. After a learning phase, that includes learning real multi-step attacks, the framework is able to either filter critical episodes predicting future steps of attacks or to filter uncritical episodes which might correspond to new attack strategies. A three-phase AC framework called 3PAC is proposed by Ramaki and Rasoolzadegan in [189] which processes real-time alerts, correlates them utilizing causal knowledge discovery, constructs attack scenarios via the Bayesian network concept and is able to predict next attack steps. The authors state that the Bayesian inference model in conjunction with statistical data mining techniques has several merits for alert analysis, e.g., processing speed, incorporating expert knowledge, computation of a correlation output probability instead of a binary result. New attack strategies are extracted from the attack tree construction phase based on classified benign episodes. Those serve as a basis for the presented causal knowledge analysis algorithm to identify critical new episodes. ZePro [190] by Sun et al. is an approach targeted towards zero-day attack path identification. The approach assumes that a chain of attack actions, typically composed of both zero-day and non-zero-day exploits, forming an attack path is necessary by adversaries to achieve their malicious goal. ZePro builds a comprehensive network-wide graph based upon system calls from which it effectively and automatically identifies zero-day attacks.

Missed Attack Hypothesizing

Missed attack hypothesizing is dealing with the problem of FNs that might occur with IDSs which does not lead to the generation of an alert. Alert analysis, especially considering multi-step attack detection relying on each alert, must be able to cope with the missing ones. However, this functionality poses still a major problem in the field of alert analysis. According to [28], hypothesizing can be categorized into approaches with or without any predefined knowledge about attacks. The former require knowledge, e.g., in form of attack templates to compare the received attack types with existing attack patterns. Exemplary work is provided in [191, 192]. The more interesting and innovative hypothesizing approach uses data-mining techniques that can generate some artificial clusters that represent attack classes which are then validated whether the quality of the cluster is higher than a certain threshold representing a missed attack. According to [193], three main steps are necessary for the hypothesis process: cluster generation, cluster validation, and cluster tuning. Fatma and Limam are presenting a two-staged approach in [194] that firstly deals with FP alerts aggregated from multiple IDSs and secondly tries to identify potential FNs representing missed attacks. Therefore, the first stage clusters IDS alerts into a set of meta-alerts based on several attributes and identifies FPs using binary optimization. The second stage discards meta-alerts that have been created by the majority of IDSs. Remaining alerts are grouped and yield the set of potential FNs from which - via a binary classification algorithm - the set of FNs is identified.

2.2.3 Post-Processing

Post-processing is performed after AC, in a particular attack scenario construction, and is composed of *intention recognition*, *(propagation) prediction*, *alert prioritization*, *impact analysis* and *visualization* (presentation, representation and visualization of detected security events or corresponding attack scenarios in a SIEM and/or to a SOC).

Intention Recognition

Intention recognition is mainly a forensics topic and deals with the derivation of an adversaries' intention (typically unpredictable) caused by a malicious activity by analyzing the alert analysis results. It deals with the interpretation and judgment of the purpose, vision and intention of attackers according to their behavior and network environment by analyzing the alert information. However, in a broader spectrum, it tries to give a reasonable explanation of the real purpose of malicious activity and predicts the subsequent attack steps which is, according to [195], the premise and foundation of threat analysis and the important part of network security situation awareness. Malicious activity in this work is either referred to as (self-propagating) malware as it is mainly discussed in the literature and represented by a worm/virus software, by a human-controlled or an AI-driven attack. However, with the advent of artificial intelligence for cyber attacks the boundaries blur which might make it difficult to strictly distinguish between them. Malicious activity may contain parts of the attack phases (sometimes referred to as kill chain) itemized and summarized below. The main goal of malicious activity is, however, to infect/compromise victim systems even if the motivation/intention behind may differ.

- **Planning and Discovery:** Social Engineering, Permission/Authorization, Information Gathering, Scoping and Exploration, Service Identification, Scanning and Fingerprinting
- **Exploitation and Assessment:** Vulnerability Identification, Vulnerability Assessment, Enumeration, Gaining Access, Exploitation (External vs. Internal)
- **Post-Exploitation and Reporting:** Discovery and Forensics, Finding Analysis, Data Collection, Maintaining Access, Covering/Cleaning Tracks, Placing Rootkits/Backdoors, (Network) Spreading, Privilege Escalation, Reporting

A kill chain describes the ability to disrupt the sequence of events an attacker must perform in order to achieve success during an attack. Thus, it breaks down attacks into levels which represent the structure and procedure of an attack. For each level, the model indicates which activities attackers undertake, so that one can set up the defense including reaction mechanisms accordingly. Although the cyber kill chain is already several years old, it can not only be applied to classic malware attacks. APTs can also be mapped to it and broken down into steps. While the attacker has to go through the complete process to reach the target according to the model, the defense can try at any level to interrupt the kill chain and thus stop the attack. However, it is necessary to build up the defense in several levels, because at each level the attacker can already do damage. The earlier the attack can be detected and stopped, the less damage can be expected. Several existing kill chain models are discussed and a new kill chain model is proposed in [196] for a remote security log analysis with a SIEM software.

Malware are meant to be programs that self-propagate across a network exploiting security flaws having the ability to propagate from host/network to host/network. The program first explores vulnerabilities in the network by utilizing various discovery techniques and infects entities by exploiting them. The infected entity serves then to spread itself automatically or through human triggering. Typical properties are the fast and various different automated scanning techniques, the functional range of either targeted to one or more (limited) number of vulnerabilities to exploit and spread as well as the aim of infecting as much vulnerable victims as possible (dedicated for large-scale networks). However, if vulnerabilities are once fixed, the certain type of self-propagating malware is no threat anymore. In contrast to an automated malware, the human-controlled attack performing the above listed attack phases has different properties. Scanning, especially in the sense of APTs, could take more time when performed stealthy. They have a large number of tools at their disposal with a broad spectrum of attack vectors to gain access. If a vulnerability has been fixed, there might exist various others such that the host can be reinfected again. Further, the latent time between the initial possible low-privileged access to gain system privileges which might be necessary to further reach other networks (e.g., pivoting on dual-homed machines or gateways) can be longer. Hackers might adapt the exploits for gaining access or privilege escalation which often result in a crash of a system depending whether the attacker has not tried the exploit in advance. This could translate into a slightly higher death rate of machines compared to the malware case. The goal differs from malware since maybe only certain targets are tried to be compromised, e.g., high value administrating machines, or all machines should be compromised depending on the initiator of the malicious activity (script kiddie, expert hacker, hacker organization) with their motivation in mind. However, from the information gained from analysis it might be possible to draw conclusions on the initiator. With a dedicated motivation in mind the human-controlled attack is more targeted for small-scale networks, e.g., inside a companies industrial site. Hui and Kun have proposed a dynamic real-time network attack intention recognition method based on attack route graphs [195]. By correlating real-time network attacks and vulnerabilities in their framework, shown in Figure 2.13, the method determines spread routes and stages of an attack based on graph theory and probability theory. They are then able to dynamically reason the possible intrusion intention and its probability according to the attack behavior characteristics and the network environment.

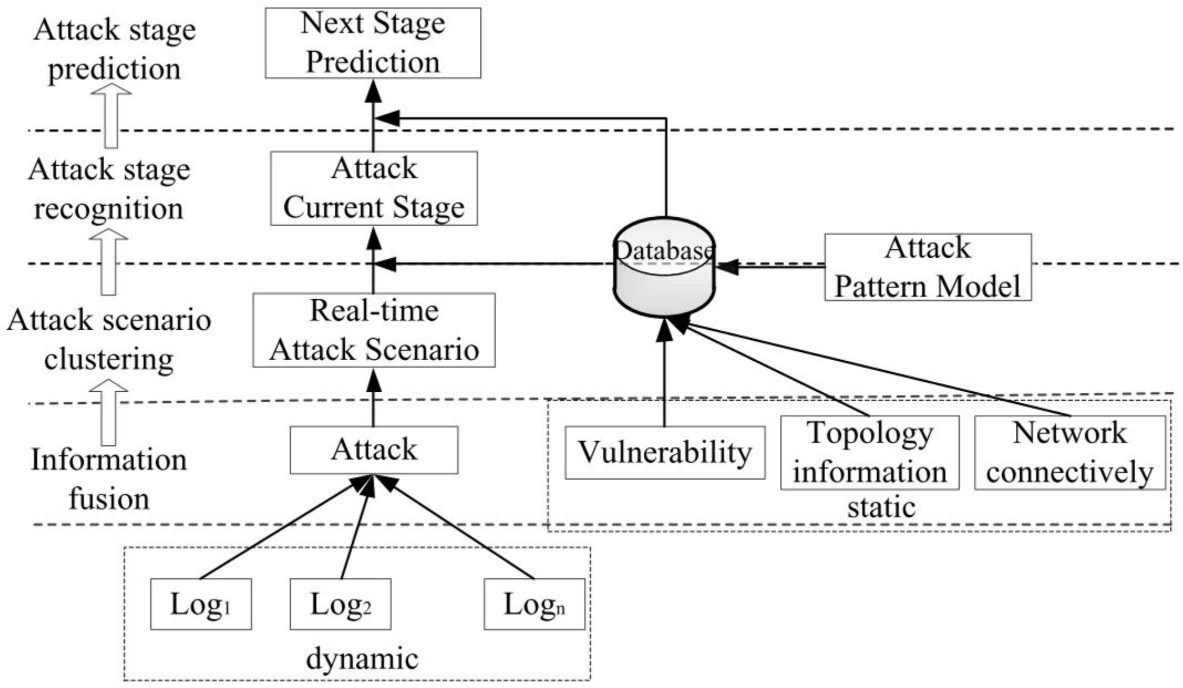


Figure 2.13: Framework of real-time network attack intention recognition [195].

Prediction

Intention recognition is closely related to prediction (or attack forecasting, attack projection) which tries to predict the next step of malicious activity during an ongoing (multi-step) attack with reference to the killchain model and how likely a next step will occur. Abdlhamed et al. are categorizing the prediction of cyber-attacks into three major methodologies: alert correlation, system call sequences as well as statistical methods and they are discussing prediction methods including hidden Markov models, Bayesian networks and genetic algorithms [197]. They state that the type of prediction is an important factor, for instance, solutions dedicated to predict attacks usually use hidden Markov models, while devoted for forecasting the intentions or the abnormal events mostly exploit the Bayesian networks. Anumol proposes in [198] an Intrusion Prevention System (IPS) which performs event analysis and predicts future probable multi-step attacks using a SVM based on network log files collected in the OSSIM[4] SIEM solution. Features consisting of, e.g., number of packets, number of bytes or packet rate are subject to formulas for information gain in order to get the best features with maximum gain ratio. However, the information provided by the article is quite sparse and its title therefore misleading in terms of predicting multi-step attacks based on raised alerts. As a proactive approach targeting to prevent attackers from reaching their malicious goals, Ramaki and Atani are providing a survey of early warning systems which acts beyond the scope of IDSs or IPSs in [199]. In order to proactively counteract new emerging high sophisticated threats, the aim of early warning systems complementary to intrusion detection/response is to detect potential malicious behavior in a system, evaluating its scope and implementing appropriate response mechanisms as early as possible (Figure 2.14). According to [200], the term early has two different meanings: (1) starting on-time to prevent or minimize the damage or damages and (2) the ability to process incomplete information. A discussion of characteristics (e.g., functionality, detection scope, challenges, data collection), architectures (centralized, hierarchical, distributed) and a comprehensive overview of existing early warning systems as well as those under research and development is provided by [199]. Further work of Ramaki et al. [201, 189] intensively deals with real-time AC and prediction using, e.g., Bayesian networks.

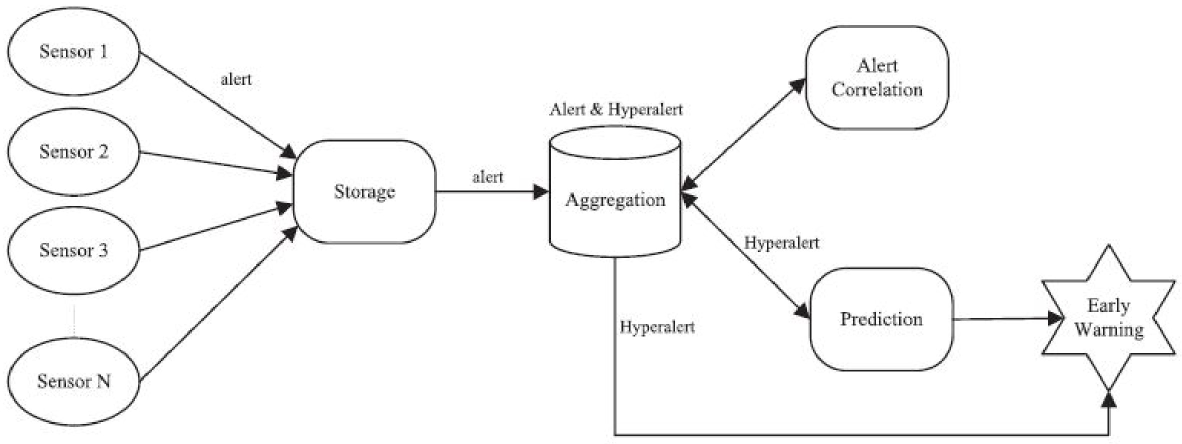


Figure 2.14: Operation of a generic early warning system [199].

In [172] Ghasemigol et al. are presenting an attack forecasting approach in form of an extended attack graph which is able to predict future network attacks based on information such as intrusion alerts, active responses, and service dependencies. Therefore, they combine the information acquired from an uncertainty-aware attack graph, a hyper-alerts graph, a multi-level response graph and a dependency graph into their proposed forecasting attack graph which increases its accuracy for predicting future attacks based on the additional information. The problem of a real-time multi-step attack prediction has been targeted by Holgado et al. in [202]. In this work, IDSs send their alerts using IDMEF to the system which turns them into a hidden Markov model capable of observations utilizing a clusterization process that incorporates a tag (inferred by matching the significant words in the alert description with the words occurrence frequency in the Common Vulnerabilities and Exposures (CVE) reports) and a severity (corresponding to the alerts severity

[4]`https://cybersecurity.att.com/products/ossim` (accessed on 05 September 2021)

parameter). The hidden Markov model states are representing the chain of different attack steps. By computing the mean number of alerts for each state, a final attack probability can be computed. The training of the hidden Markov model is performed by applying a supervised and an unsupervised (Baum-Welch) algorithm. An evaluation is performed using the LLDDOS1.0 attack scenario of the DARPA data set and the alerts provided by a Snort IDS. The authors of [203] are modeling vehicle states, e.g., door open/closed, vehicle moving/stationary, using a hidden Markov model extracted from the CAN network traffic. Considering the movement of a vehicle as a sequence of states that depend on the previous ones, the model can be derived and anomaly detection is performed within a sliding window of n previous observations that slide over the various states. If the posterior probability is 0 or less than a defined threshold, an alert is generated indicating an anomaly. In each case an observation would be a vector of different sensor values that generates a set of probabilities corresponding to each observation. The proposed approach seems quite promising for detecting anomalies in time series data. However, it could be seen as a utility for further appliances to work beside low-level detection mechanisms on a higher level of abstraction for cross-evaluation. Thus, for instance, if a car is driving with an implausible speed and the applied IDS sensors are detecting a huge number of alerts, the security state of the vehicle is critical and an immediate interaction by the driver needs to be taken. The prediction system of Abdlhamed et al. in [197] incorporates multiple sources of information and different methodologies. There are two modes of operation: if there are little security incidents, prediction is done by using statistical methods. If the information is sufficient, the system builds attack scenarios and constructs profiles for suspected users. A risk assessment is dynamically calculated when there is an abnormal behavior affecting the system. The prediction is produced when the profiled user actions and the dynamic risk assessment are indicating specific stages on the security master plan. However, the system is predicting attacks over multiple days as shown in the evaluation which does not satisfy real-time requirements.

Apart from the selection of aforementioned prediction approaches, malware propagation (prediction) models can be exploited to forecast the further network spreading of malicious activity. Not only attacks controlled by humans but also unassisted malware gains more sophistication. The classical separation of malware in the categories with their different flavours, e.g., fuzzers, backdoors, denial-of-services, exploits, shellcode execution, worms, viruses, or trojans, is not possible anymore since new malware shows various characteristics or functionality of other. Therefore classical worms can show behavior/characteristics/functionality (e.g., stealthiness, polymorphism, context awareness) of other malware and turn into self-spreading programs. Their propagation - characterized by random effects - can be modeled using a stochastic process. Those effects can be malware-related (e.g., scanning strategy), network-related (e.g., bandwidth, topology), system-related (e.g., vulnerable hosts), policy-related (e.g., intrusion prevention) or human-related (e.g., removal, patching, isolating, restoring) and leads to an overall complexity. Especially when considering the human-related portion which mainly affects the response/reaction to malware, their actions could be performed (semi-)automated. Understanding the behavior (propagation) of malicious activity inside an IT-network could help an incident management to be a step ahead. The model of malicious propagation based on information gained from detection and analysis components could help to individuate and describe symptoms of malicious activity such that useful data could be provided to trigger emergency responses or the implementation of automatic reactions. Mathematical models in general can be categorized into three different characteristics [204]: (1) deterministic or stochastic, (2) continuous or discrete, and (3) global or individual. Those models depend on whether the variables (and parameters) are random or not, if the variables take an infinite or finite number of values and aims either to simulate the behavior of a complex system providing the global evolution or, in contrast, only focuses on the dynamics of individual nodes. Malware propagation models are based on those initially developed for the spreading of infectious diseases. The epidemiological models are compartmental, that is, the population (through which the infectious disease is propagated) is divided into different types of behavior bearing in mind the characteristics of the disease: susceptible, exposed (with or without symptoms), infectious, recovered, quarantined, vaccinated, isolated, and so on [204]. Possible states and interactions of the compartment models, e.g., Susceptible-Infected (SI), Susceptible-Infected-Recovered (SIR), or Susceptible-Exposed-Infectious-Recovered (SEIR) are shown in Figure 2.15.

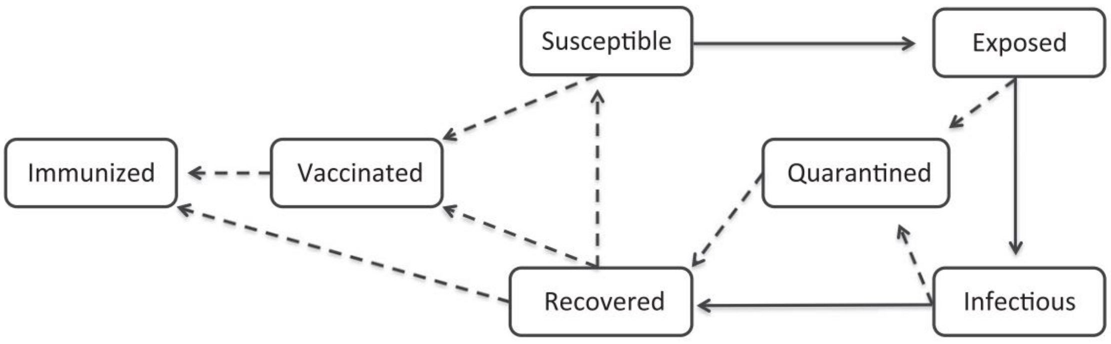


Figure 2.15: Different types of behavior in compartmental models [204].

Epidemic models in general can be classified into *deterministic* ones e.g. [205], *stochastic* ones e.g. [206] which typically refer to global models representing the dynamic of the overall population without taking into account the local interactions between individuals and, according to [207], *spatial-temporal* ones considering individual-based models, e.g., cellular automata [208] or agent-based [209]. Deterministic models are usually based on Ordinary Differential Equations (ODE) or difference equations and perform better in large-scale networks. Stochastic models can be subdivided into three types differing in their underlying assumptions regarding time and state variables: Markov chains (discrete time with discrete state variable), Markov chains (continuous time with discrete state variable) and Stochastic Differential Equation (SDE) with continuous time and discrete state variables. Stochastic models show their strength considering small-scale networks. However, both global models offer a good insight in the dynamics of networks providing characteristics, e.g., stability or equilibrium [210] but do not consider individualized infections because the global parameters are fixed. Thus, changes of individual computers by, e.g., an updated anti-virus software are not taken into account. Furthermore, global models assume a homogeneous distribution of malware in a network meaning that each system is connected to another. This is true considering large-scale networks, e.g., malware spreading in the internet but in microscopic environments, e.g., local networks, the results are less reliable since the dynamics strongly depend on individual interactions. Individual-based systems overcome those drawbacks by including other information, e.g., update policy, applied operating systems and the topology by, e.g., utilizing directed or undirected graphs, but demand a high computational cost when considering large-scale networks or they require a comprehensive gathering of information from the system components. A comprehensive introduction in the field of graph theory, network topologies, community structures and diffusion models with regard to malicious attack propagation is given by [211].

A selection of research activities regarding the use of SDE for malware propagation is presented in the following. The authors of [212] are providing a survey and comparison of worm propagation models by categorizing them into scan-based (propagation without dependence on topology possible by various scanning techniques) and topology-based (spreading through topological neighbors) worms based on their characteristics. The authors provide information about propagation models on these categories but, however, they do not consider stochastic models, e.g., in form of SDE. In the first part of the dissertation [213], stochastic epidemic models and inference for the propagation of computer viruses are studied. A comprehensive literature review on deterministic and stochastic models is provided with a focus on Markovian- and SDE-based SEIR models. With respect to SDE, the author developed a new model for multi-group stochastic SEIR. It is stated that, although ODEs can be safely used to approximate a stochastic process when the population size is large, no probabilistic event is considered. Moreover, the ODEs only describe the average tendency of virus propagation. Thus, deterministic models cannot represent rare events such as saturation and extinction of malicious activity. The two main findings of the work in [213] considering the multi-group based SDE approach are that a SEIR framework including a latent period is superior to other models and the impact of the network structure can be explored via multi-group variants with different parameterization within subnets/subgroups clustered of individuals which differ substantially in communication activity profiles and in their purposes. For future work, the authors propose to consider the integration of human countermeasures in the model since appropriately including the effects of such countermeasures can substantially further improve such a models's predictive ability and the impact of the malicious activity propagation. Another point is to consider the infection rate which is normally constant as a function of time. This can especially be the case when an adversary is invading deeper inside a network trying to reach a high value target such as a domain admin machine. The author also states the investigation of the effects of the network topology on computer malware propagation must be included in further work for instance by tying together the ideas and tools of random graph dynamics to describe the stochastic behavior of the topological structure of large computer network.

In [206], the authors build on a stochastic worm propagation model based on SDEs modeling random effects during worm spreading. Derived from the paper, the essential SDE for the modeling of infected hosts is given by Equation 2.9 in Itô notation in which $I(t)$ is the function representing the infected hosts. N are the unique hosts in a network scanned by the worm where N_S is the number that could potentially become infected. $< \beta >$ is the mean of the total infection rate incorporating randomness including the worm's decisions, e.g., scanning strategy, scan rate or changes in the network environment, e.g., bandwidth or congestion. Using the Euler-Maruyama method, Figure 2.16 shows 10 plotted paths in red including the computed mean (500 simulated paths) for the infection function $I(t)$ of Equation 2.9 where $N = 254$, $N_s = 100$, $< \beta >= 1.4$ and $I_0 = 5$.

$$dI(t) = \frac{< \beta >}{N}(N_S - I(t)) \cdot I(t) \cdot dt + \frac{1}{N}(N_S - I(t)) \cdot I(t) \cdot dB(t) \tag{2.9}$$

The authors state that the size of the network, small-scale or large-scale, e.g., Internet, needs to be considered since a small network size reduces the time for early detection but increases false alarms because large-scale networks will diffuse the network heterogeneity's and better describe the phenomenon. In order to counteract this, the authors provide a theoretical estimation of a critical network size which is sufficient to be monitored. For network monitoring and intrusion detection this information for critical size in subnetworks can be useful. Thus, worm projection basically involves collecting data and then estimating the infection rate and expected damage caused by the worm. Early projection results, paired with a well-established early warning policy, may lead to robust response strategies against fast-spreading, unknown worms. Since, in an unknown network the size could either be far smaller, far greater or close to the critical size, a hierarchical distributed early warning system is proposed by the authors. Each network domain k monitors n subnetworks with variable sizes and internal characteristics (e.g., bandwidth or topology). An early detection component which is able to detect the presence of a fast-spreading worm and is able to define the worm propagation model parameters. Each domain is locally monitored by one Local Monitoring Center. A local agent runs in each LMC and is programmed to act as a communication interface between the LMC and the root of the hierarchy, namely a Global Monitoring Center (GMC). The operation of the local agent is practically the basic requirement in order to participate in the warning system. Finally, the GMC receives infection information from the LMCs and sends back warning information for an emergency response. The LMCs could, as a response measure, adapt their network or host-level firewall policies, automatic quarantine policies or disconnect/isolate particular hosts or services. The authors state that an early warning system is meant to complement current systems and they suggest the deployment of an anomaly-based IDS that will collect preliminary data from default locations, analyzes it and makes a decision on whether the examined traffic contains potential scanning worm behavior.

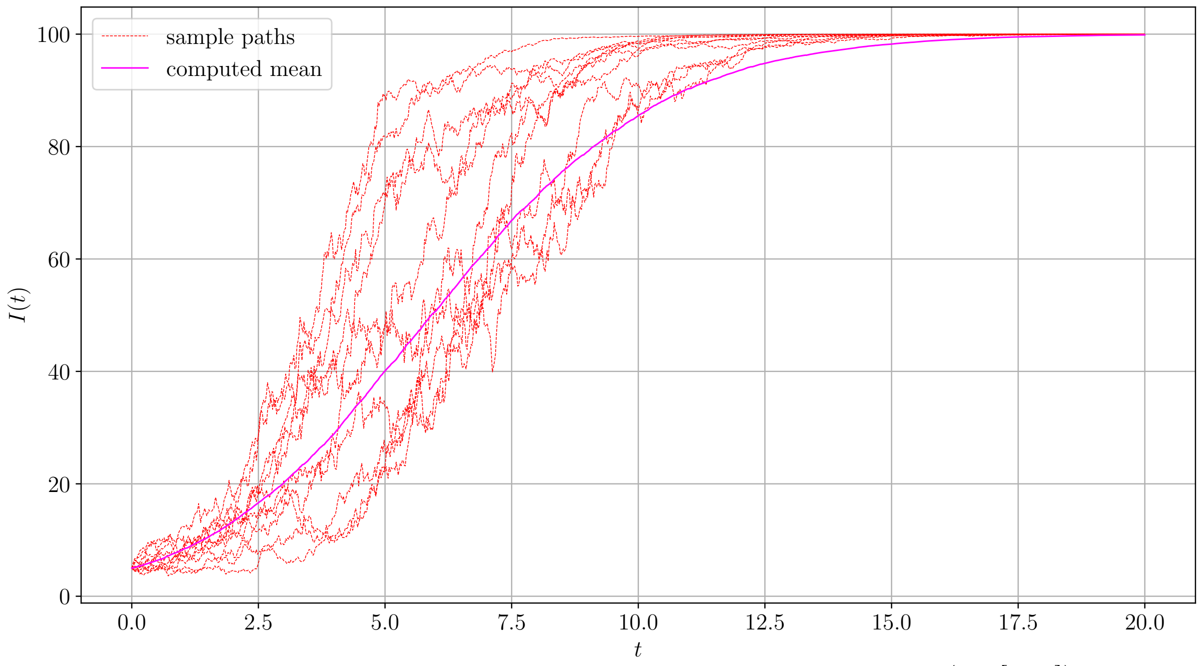


Figure 2.16: Simulation of an SDE for malware propagation (cf. [206]).

Impact Analysis

The characteristics of the malicious activities' impact may also be different (impact analysis) apart from the stated above. For instance, a malware exploiting a certain discovered vulnerability that effects all models of a series (either a certain vehicle [automotive] or Programmable Logic Controller (PLC) [industrial] type) has a more devastating impact on the whole vehicle fleet [automotive] or various companies applying the PLC type [industrial] than a single hacked/compromised vehicle [automotive] or compromised industrial plant [industrial] by an adversary. Along with this, the topology of single compromisable entities and reachability with others, e.g., Electronic Control Units (ECU) inside a vehicle and their connection between them vs. the vehicles of a whole fleet and their communication among them and their infrastructure, plays a major role and has to be considered. Apart from the close relation of impact analysis with topics such as incident response and forensics analysis, the most inferred is the threat and risk assessment. Its main goal is to identify the consequences of malicious activity which helps administrators to find out the implicit and explicit relations between the attacks and the assets of an organization using information sources such as asset database, topology database, and vulnerability database for analyzing the impacts [214]. Risk management describes the process of the consideration of potential risks in a certain domain. It is further composed of risk assessment and risk mitigation. The former is the process of determining, analyzing, and interpreting the risk analysis results, and risk mitigation is the process of selecting and implementing security controls to reduce risks to an acceptable level [172]. Especially in the automotive domain, the Hazard and Risk Analysis, abbreviated HARA, targeted for safety as well as the Threat and Risk Analysis, abbreviated TARA, targeted for security are widespread. Network risk assessment approaches can be categorized into network risk assessments based on (1) attack graphs, (2) dependency graphs and (3) on non-graph approaches, e.g., hidden Markov models or fuzzy logic [5]. An overview and discussion of available literature on these categories can be found in [172].

Prioritization

Prioritization focuses on categorizing and ranking alerts based on their importance since generated alerts by security components do, according to [215], not have equal importance. Salah et al. state that for analyzing the significance of suspicious events, a prioritization component needs to be added to the correlation system [165] such that it can be distinguished between moderate and devastating threats/attacks. Porras et al. propose an AC and ranking technique called M-Correlator [216] which ranks alerts based on the likelihood of the attack to succeed, the importance of the targeted asset, and the amount of interest in the type of attack [217]. Alsubhi et al. state in their work that these techniques are promising in the evaluation of alerts generated by signature-based IDSs, but cannot evaluate alerts raised by anomaly-based IDSs, since they heavily rely on the vulnerability knowledge base. Thus, in their work [217], they extend Porras et al.'s approach by offering a technique which works with both signature-based and anomaly-based IDSs and makes use of additional criteria, such as the sensor sensitivity, relationship between alerts, service stability, and social activity between source and target for a more accurate evaluation of the alerts. A few alert prioritization score metrics from [217] are categorized in [29] to applicability metric (applicability of a raised alert to the current environment based on information from various knowledge bases, e.g., vulnerability knowledge base), victim metric (specifying the properties of critical machines, services, applications, accounts, and directories in the current network environment), sensor status metric (based on the Bayesian detection rate formula estimating the TP probability that an alert is raised when an attack is detected), attack security metric (measuring the risk level of the vulnerability based on known attack metrics such as MITRE or CVE), service vulnerability metric (representing the strengths and weaknesses of a host based on the targeted services) and social activity metric (exploiting features of a social network to find hidden participants in a communication session). McElwee et al. are proposing a deep learning based approach for prioritizing and responding to alerts in [218]. Their FASTT (Federated Analysis Security Triage Tool) concept uses a TensorFlow deep ANN classifier to automatically categorize IDS alerts and determine which type of security analyst should review the alerts. In addition, FASTT uses an Elasticsearch indexed data store to speed the retrieval of IDS alerts and a Kibana user interface to allow flexible display of visualizations and dashboards that can be tailored to meet the security analysts' workflow. ACSAnIA (A Comprehensive System for Analysing Intrusion Alerts) is proposed by Shittu et al. in [219] which is a post-correlation framework that consists of seven components. The *Offline Correlation* uses a set of historic alerts to build a correlation model which gets updated periodically by the *Online Correlation* module. This module analyzes it for every incoming alert against a set of past alerts of a certain time window. The *Meta-alert Comparison* measures differences between meta-alerts that are produced by the correlation process and *Meta-alert Prioritisation* maps, a prioritizing value to each meta-alert based on the degree to which it is an outlier computed by the local outlier factor algorithm. The *Meta-alert Clustering* receives the set of meta-alerts and groups them into clusters. The *Attack Pattern Discovery* receives the clusters of meta-alerts and attempts to extract a set of representative features for each cluster using frequent pattern mining. The *Reporting System* receives the outputs and prepares them for human interaction. The results of alert prioritization are useful for countermeasure selection, where the system is able choose a suitable response automatically based on the assigned values for detected attacks (Section 2.3).

Root Cause Finding

Alerts are typically indicators of problems representing the symptoms of incidents but do not provide explicit information regarding the actual root cause of them. However, identifying the "culprit" of malicious activity is not only important for forensic purposes but also for finding the causation as quickly as possible to establish countermeasures as nearly located as possible. Simple mechanisms such as IP traceback, a method for identifying the origin of a packet on the internet based on its source address, seem insufficient for root cause finding since for the propagation of malicious activity, the source of packets is almost never the source but just one of the many propagation hops. Furthermore, IP-spoofing prevents its application. Even the detection of stepping-stones, by techniques stated and discussed by Kumar and Gupta [220], only focuses on the IP level of packets which make finding the root cause difficult. Furthermore, they are vulnerable to time delays, chaff perturbation, and have a high FP rate. Root cause finding methods are tractable estimators performing on multiple topologies to find propagation sources in higher level of networks, e.g., an application level to find logical structures. Approaches of identifying malicious attack sources can be divided into three main categories of source detection: the complete observation based (requires a complete observation of the attacked network after a certain time of the malicious propagation), the snapshot based (requires a partial observation of the attacked network at a certain time), and the detector/sensor based (requires the observation of a small set of nodes but all the time in the attacked network) [211]. Figure 2.17 illustrates the three categories and Figure 2.18 a taxonomy of source identification methods based on this categorization. For a detailed description of the following refer to [211].

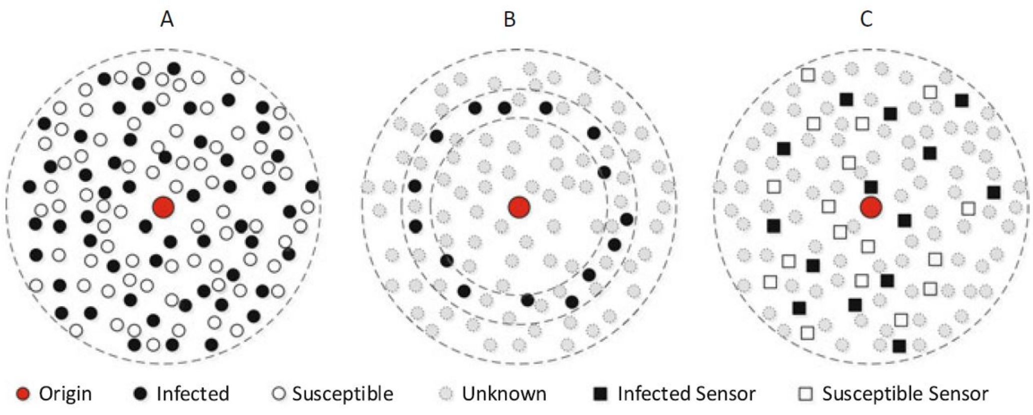


Figure 2.17: Illustration of three categories of observation in networks. (a) Complete observation; (b) Snapshot; (c) Sensor observation [211]

Examples for the former are the Rumor Center of the work in [221] or the Dynamic Age found in [222]. In their works [221, 223, 224], Shah and Zaman provided models for rumor spreading in a network based on the SI model and then constructed an estimator for the rumor source based on a novel topological quantity, called rumor centrality. They established a maximum likelihood estimator for a class of graphs: the regular tree graph, the general tree graph, and the general graph. The second category deals with the problem that in real-world networks a complete observation might not be possible. Thus, a given snapshot at a certain time serves as a basis for source identification. Representative for this class is the work of Zhu and Ying [225], which base their solution on the SIR model and are providing a reverse infection algorithm based on a given network snapshot. A node with the minimum infection eccentricity, called Jordan center, is the estimated source node by the algorithm. Approaches using sensor observations are, among others, based on Bayesian [226], Gaussian [227] or Moon-Walk [228] (Figure 2.18).

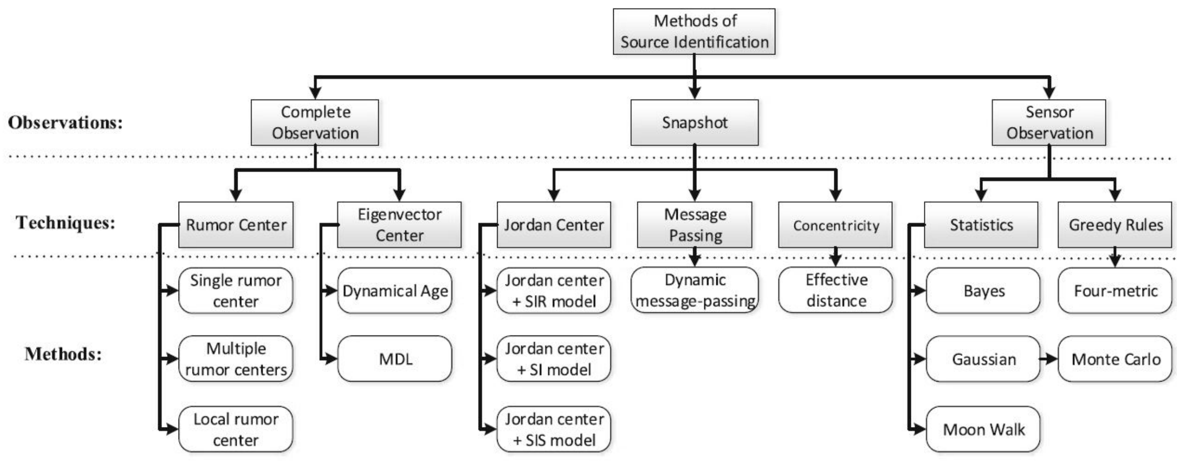


Figure 2.18: Taxonomy of current source identification methods [211].

The aforementioned approaches for a root cause identification mentioned in [211] are, however, not relying on the output of IDSs in form of alerts. Julisch [229] initially proposed a different approach for source identification using a set of unlabeled NIDS alerts and generating clusters of similar types. The work discusses alarm clustering as a method that supports the discovery of root causes. Another clustering based approach is Y-AOI [230] that bases on the Y-means anomaly detection method [231] and uses an attribute-oriented induction algorithm. Firstly, Y-means divides the alerts into several clusters based on their occurrence time and secondly an adopted oriented induction algorithm inducts the clusters into short, highly comprehensible and more informative summary tables which help administrators to more easily find the root cause. Al-Mamory and Zhang are using the root cause analysis in [232] to discover the sources that make IDSs trigger a large number of alerts by supposing that most of the root causes are no attacks. Similar alarms are clustered by their semi-automated clustering system, also basing on the attribute-oriented induction algorithm, helping the security analyst in specifying the root causes behind these false alarms and in writing accurate filtering rules. Kechadi et al. present an approach called Behavioral Proximity Discovery which is a framework for root cause analysis that consists of three complementary clustering algorithms based on alarm behaviors. The first algorithm (SM) identifies periodic alarm behaviors. The two other algorithms (FECk and CUFRES) correlate events leading to the identification of faults by the network operator [233]. An automated root cause identification approach is proposed by Cotroneo et al. in [234]. Their framework (Figure 2.19) consists of a filter and a decision tree to address a large number of alerts and to support the automated identification of root causes by adopting term weighting and conceptional clustering approaches to fill the gap between the unstructured textual alerts and the formalization of the decision tree.

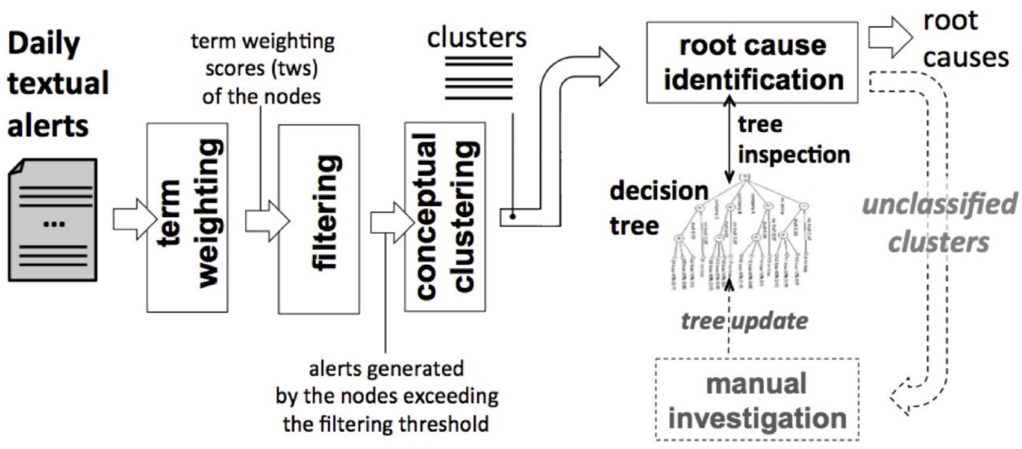


Figure 2.19: Root cause identification framework [234].

Another promising possibility for identifying the root cause comes from the field of exploiting the physical properties of network participants. Physical device fingerprinting builds fingerprints to uniquely identify a machine by, e.g., measuring the differences in machine internal clock signals. In [235] a time-based device fingerprinting technique is proposed that is generic and can work with different functions, making the method adaptable to different environments. Cho and Shin proposed an anomaly detection approach called Clock-based IDS (CIDS) that is based on the unique clock skews of an ECU in [154]. Since the CAN frame in an in-vehicle communication does not deliver any information about its source ECU, the downside of many IDS approaches is that anomalies cannot be traced back to its source. By associating a clock skew to each ECU in the CAN network, every message can be backtracked to its source ECU. CIDS first creates a norm baseline based on clock skews which is done by measuring the intervals of periodic messages constructed on the recursive least squares technique. After the construction of the norm baseline, each ECU is associated with its own clock behavior or fingerprint. The detection of anomalies is conducted from using the cumulative sum method which is used to detect abnormal shifts in identifying errors. With Viden (Voltage-based attacker identification), Cho and Shin are providing another possibility in [236] to identify the attacker ECU by measuring and utilizing voltages on the in-vehicle network. Viden exploits the voltage measurements to construct and update the transmitter ECUs' voltage profiles as their fingerprints and uses these profiles to pinpoint the attacker ECU with a low false identification rate. A major benefit of Viden compared to CIDS is the ability to function even in the presence of event-triggered messages. CIDS rely on the timing information from periodic messages but since Viden determines the transmitter ECU based on voltages, it is able to pinpoint attacker ECU regardless of the communication technique.

2.3 Incident Response

Incident response or reaction implies the set of actions a system executes after security-relevant incidents have been identified. An IPS, as a proactive solution, works as an IDS but in the case of an detection it drops malicious traffic automatically before it causes any harm to the network rather than raising an alarm afterward. Similar to the term countermeasure defined in RFC 2828 [237] as "an action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken", a response refers to a specific action that is triggered by an alert or an alert analysis after an intrusion detection phase (reactive). Evaluating the severity of attacks, identifying the cause of incidents, and selecting an appropriate response under considerations of, e.g., the correct time or the available resources are typical tasks of an Intrusion Response System (IRS). Intrusion risk assessment is closely related to the field of IRSs. This process helps to determine the probability that a detected incident is a valid, action demanding attack towards an important compromised target that requires a certain form of a response. Properties and characteristics that influence the response model, e.g., incident response time are provided in [238].

2.3.1 Taxonomy of Intrusion Response Systems

A taxonomy for IRSs is provided by Shameli-Sendi et al. in [5] and shown in Figure 2.20. The characteristics and criteria of their taxonomy resulted from a comprehensive literature review and are listed in the following. Some additional requirements that should be fulfilled for the development of an IRS stated by [238, 239] are added in *italics*.

- ***Activity:*** The activity of a triggered response can be categorized into passive (do not attempt to minimize damage already caused by the attack or prevent further attacks - main goal is notification) and active (aim to minimize the damage done by the attacker and/or attempt to locate or harm the attacker).
- **Level of Automation**: An important feature of an IRS is whether it can be fully automated or requires administrator intervention after each incident. Therefore, the level of automation can be categorized into notification systems (alert information is used by an administrator to select applicable response measures), manual response systems (preconfigured set of responses based on the type of attack that an administrator has to trigger) and automated response systems (automated execution of responses without human intervention). *Cooperation and autonomy for response systems are two features used respectively in a NIDS and HIDS. Thus, it*

is necessary to have both features in a single system to be more accurate. According to [66], automated IRSs can further be divided into: adaptive (feedback loop to evaluate previous response), expert (response decisions are based on one or more metrics) and association (simple decision table approach in which a specific response is associated with a specific attack).

- **Response Cost**: Knowing the power of responses to attune the response cost with attack cost plays a critical role in IRSs. The evaluation of the positive effects and negative impacts of responses are very important to identify response cost. *Thus, the selected response should not be more costly than the attack.*
- **Response Time**: This criterion refers to whether the response can be applied with some delay after an intrusion is detected (delayed, reactive) or before the attack affects the target by applying an intrusion prediction mechanism or IPS (proactive). *Thus, proactive response mechanisms should be included within the intrusion response system to enable early responses to intrusions.*
- **Adjustment Ability (*Adaptiveness*)**: Usually, an IRS framework is run with a number of pre-estimated responses. It is very important to readjust the strength of the responses depending on the attacks. *Adaptiveness is a powerful feature required to ensure normal functionality while still providing effective defense against intrusive behavior, and to automatically deploy different responses on the basis of the state of the current system.* Adjustment models can be categorized into non-adaptive and adaptive. The former keeps the order of responses during the life of the IRS, whereas the adaptive has the ability to adjust the order based on their history. Non-adaptive adjustment models can be converted into adaptive ones. Response goodness refers to the history of success and failures of each response mitigated over time.
- **Response Selection (*Effectiveness*)**: The task of an IRS is to choose the best possible response. Existing techniques vary in the way response selection is achieved. *The response should be tailored to the type of attack and incident context. In fact, the response could be different for the same attack affecting two kinds of resources.*
- **Applying Location**: There are different locations in the network to mitigate attacks. Using information from an "attack path" (consisting of (1) adversary's start point, (2) firewalls/routers, (3) intermediary devices and (4) target device) appropriate locations with the lowest penalty cost for implementing response measures can be identified.
- **Deactivation Ability**: Another distinguishing feature that separates IRSs is the response deactivation (response lifetime), which can take the users' needs in terms of Quality of Service (QoS) into account. Most countermeasures are temporary actions which have an intrinsic cost or induce side effects on the monitored system, or both. Thus, there must be an ability to deactivate response measures.

A phylogenetic tree showing the history and advancement of intrusion response approaches is provided by [66]. The authors further propose characteristics (adaptive nature, cost-sensitive, semantic coherence, manage false alarm, and response metrics policy) for designing proper response systems since poorly designed ones may lead to a generation of a large number of false alarms and degrade the performance of the system. They state that existing IRSs are unable to provide a real-time optimum response because of the absence of semantic coherence and dynamic response metrics features. A survey of research activities with respect to IRSs highlighting their novelty and core characteristics

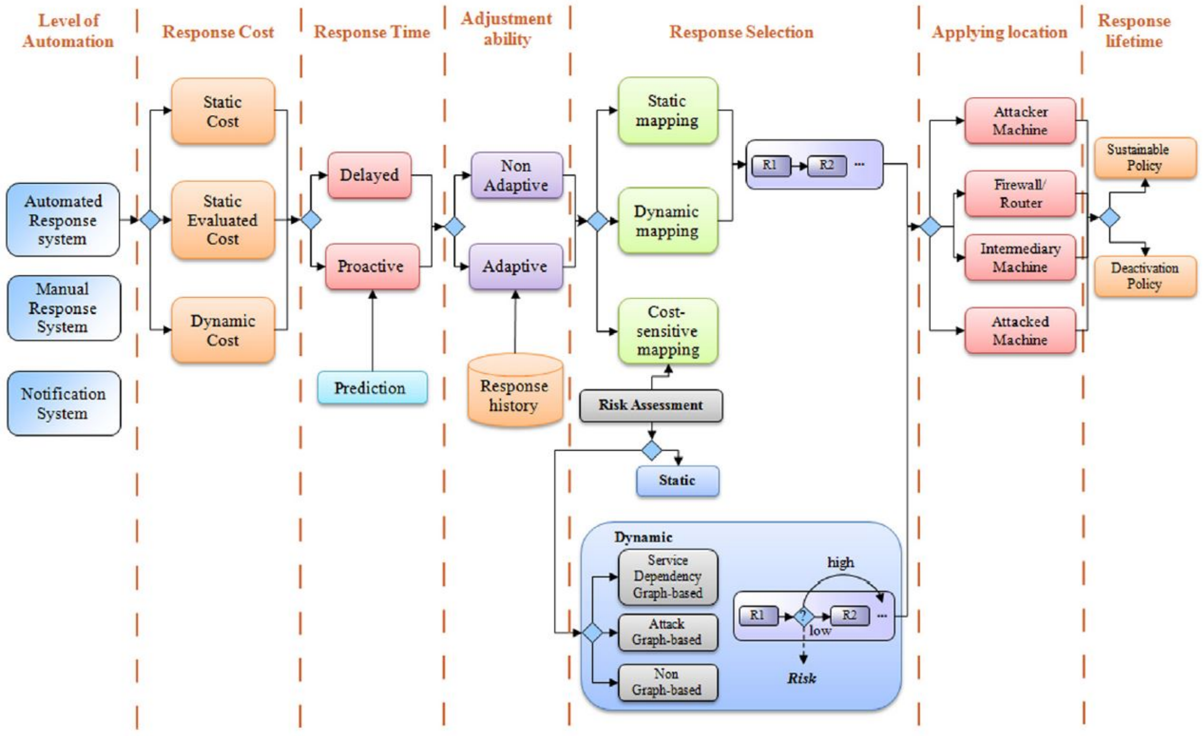


Figure 2.20: Taxonomy of intrusion response systems [5].

of relevant ones is provided in [7]. Challenges and future direction of IRSs are discussed in [7, 66] and a selection is illustrated in Figure 2.21. Further, open issues and some proposed solutions in the field of IRSs in general and with respect to cloud-environments for smart mobile devices are provided in [3, 4].

Response Cost

Response cost describes the impact of implementing responses on the system's protection goals. This may not only include security-relevant ones, such as data confidentiality, integrity and authenticity, but also safety-critical ones, e.g., availability or performance. For instance, switching off a service to mitigate an attack results in the loss of its function. An evaluation of the response cost makes only sense when considering possible attack scenarios. A dynamic response model (static cost, static evaluated cost, dynamic evaluated cost) offers the best response based on the current situation of the network, and so the positive effects and negative impacts of the responses must be evaluated online at the time of the attack [5]. For the static cost model an expert assigns a static value to a response ($R^{s}_{cost} = const.$). If an evaluation function is applied, typically incorporating the impacts on the protection goals, associated with each response, a statically evaluated cost is obtained ($R^{se}_{cost} = f(x)$). In the dynamic evaluated cost (R^{de}_{cost}) the systems' situation is evaluated in an online fashion such that certain responses are only implemented if other entities, e.g., other services, processes and resources are not threatened.

Herold presents possibilities to assess the costs of a response as well as the potential damage of a security incident in [238]. The author categorizes them into basic approaches, e.g. [240, 241, 242, 243], that select a response if the cost of it is lower or equal to the damage cost, approaches including system importance, e.g. [151, 244, 245], approaches using probabilities based on the success likelihood of an ongoing attack, e.g. [245, 246], and

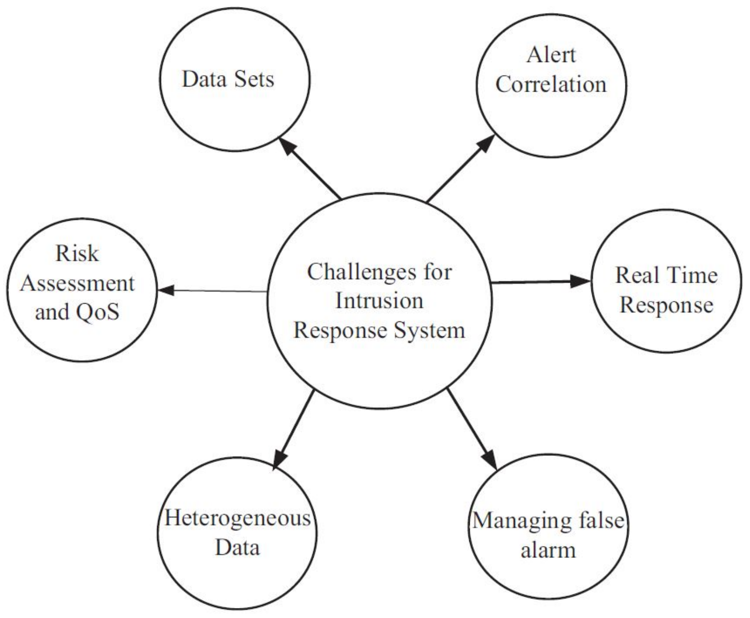


Figure 2.21: Challenges for IRSs [66].

approaches including IDS capabilities e.g. [247, 248, 249]. Fessi et al., for example, present an approach for automated reaction measures with a multi-attribute decision model and a cost-benefit analysis of the selected reaction in [250]. They present an intrusion response architecture composed of a collection module, a detection module and a response module, with the actual focus of the paper on the response module. The response module contains information regarding the known intrusions, possible reaction measures in case of an intrusion and a response mechanism for deciding on a suitable reaction measure. The problem of finding a suitable response to an intrusion is defined as a multi-attribute decision making problem. It is based on the three parameters: *financial cost* (loss for the company in financial terms), *enterprise reputation loss* (reputation of the company which is related to its survival and existence) and *processing resources* (regarding personnel and information assets). These attributes are normalized using a weighted linear combination method such that the values can be used later in the evolutionary algorithm's fitness function. Using the evolutionary algorithm, the best possible response measure for intrusion is then determined by a cost-benefit analysis. A matrix with n columns, which represent the resources in the system, and m rows, representing the reaction measures, is used. This means that one row indicates the effects of the reaction measure m on the overall system and is regarded as a chromosome for the algorithm. The evolutionary algorithm follows the usual operations (random initial population, selection, crossover, mutation and natural selection) and, after scheduling, provides the optimal reaction measure in terms of cost-benefit for the overall system. Shameli-Sendi and Dagenais proposed a practical framework in [241] for online response cost evaluation to encounter the problem that typically a good response decreases the service quality. Thus, a balance between response cost and the cost for an attack in a cost-sensitive fashion can be achieved. A service dependency graph is utilized to consider the negative effects that consist of impacts on, e.g., the network or hosts. Their framework includes information of affected resources determined by an expert's opinion.

Response Selection

Security administrators often face multi-criteria decision making problems when it comes to select an optimal response in a timely and cost-effective fashion. According to [5], there are three response selection models namely *static mapping* (an alert is mapped to a predefined response), *dynamic mapping* (response mapping to an attack differs depending on, e.g., system state, attack metrics such as frequency or severity, or network policy) and *cost-sensitive mapping*. Static mapping itself can be exploitable since an attacker may predict response measures. Dynamic mapping lacks on intelligence by improving itself with every new responded attack without any dedicated upgrade. Cost-sensitive mapping attempts to attune intrusion damage cost which represents the amount of damage to an attack target when the IDS is either unavailable or ineffective. It is closely related to risk assessment which itself can be categorized into *static* and *dynamic*. Dynamic (real-time) risk assessment approaches can be subdivided into (1) *attack graph-based* (referring to Chapter 2.2 by constructing attack steps based on correlation methods), (2) *service dependency graph-based* (impact on confidentiality, integrity, and availability on a service interacting with or depending on others), and (3) *non graph-based* (risk assessment carried out independent of the attack by performing risk analysis on alert statistics and other information provided by alerts) [5, 172]. An overview of attack modeling techniques including their strengths and weaknesses is presented in [7] structured into attack graph, attack tree, service dependency graph and Markov decision process models. A method for risk assessment based on the dynamic Bayesian network (static Bayesian network extended in time) is proposed by Wang et al. in [251].

Already in 2000, Schackenberg and Djahandari proposed an architecture based on the Intruder Detection and Isolation Protocol (IDIP) in [252] which represents an early work in the area of automated intrusion response. IDIP's objective is to track intrusions by sharing information between neighboring devices to attempt responding or tracing back the attack along it's path. Local IDIP agents are equipped with detection, audit or response functionality and reports are not only distributed among each neighboring agent for local decision making but also sent to a discovery coordinator which is able to correlate reports and to gain a better overview. For response functions, IDIP defines trace messages (including a description of the anomaly, a value indicating how certain the detector is of this attack, a severity value based on the potential services lost from this attack, and a requested response) and directives (messages sending "do" and "undo" messages in order to, e.g., block or unblock network traffic). With the Cooperative Intrusion Traceback and Response Architecture (CITRA) presented in [253], the authors extended their work to trace attacks to their source and block them as close as possible to it. CITRA allows to use immediate responses and utilizes a simple cost model to select a limited number of responses which are based on thresholds and can not be adapted. However, CITRA's main focus is on the traceback of an attacker to the source of the security incident.

Zonouz et al. are proposing a game-theoretic intrusion response engine in [254], called the Response and Recovery Engine, which allows to analyze security incidents and to take their optimal countermeasures using attack-response trees modeling a two-player Stackelberg stochastic game. The engine considers inaccuracies associated with IDS alerts and chooses an optimal response action by solving a partially observable competitive Markov decision process derived from the attack-response trees. An extraordinary approach has been presented by Sharma et al. in [255, 256] proposing an intrusion response mechanism inspired by plant-based biology. The PIRIDS (Plant-based Inspiration of Response in IDS) is a three-layered bioinspired detection and response method composed of the layers

PRR (Pattern Recognition Receptors), Guard Agent & SAR (Systematic Acquired Resistance) and HR (Hypersensitive Response). With their approach the authors were able to cope with known attacks, zero-day ones and the infection spread of malicious activities could be stopped.

Kholidy et al. are presenting ACIRS (Autonomous Cloud Intrusion Response System) in [257], an effective attack and vulnerability detection and response framework to accurately identify the attacks targeted to cloud environments. Its architecture integrates both, behavior and knowledge based techniques, and considers different service models and user requirements, collects and correlates security events and user behaviors from all environments in the cloud system. It provides a security measure to assess the system risk to select an appropriate response to mitigate the risk consequences. Alert integration, correlation and risk assessment is based on IDMEF to include multiple detection mechanisms such as Snort (network-based) and OSSEC[5] (host-based) for instance. Thus, as normalization process, all alerts are brought into IDMEF to simplify their analysis and correlation in the next layer. A prioritization process ensures that each detector (analyzer/sensor) has its own prioritization system. AC and summarization correlates a large number of normalized alerts from different detectors to highlight the few critical ones. This technique looks for evidences of an alert to discover if it signals a true attack and it correlates logically related alerts. Logically related alerts denote the same attack signature, have the same source and destination addresses and are close in time. These alerts may also denote a step of a multi-stage or compound attack that consists of a set of steps performed by one attacker. Beside reducing FPs alerts and summarizing the huge number of alerts to the cloud administrator, this results in a correlation process that deals efficiently with multi-stage attacks and facilitates the risk assessment and mitigation processes. The authors propose to use OSSIM, an open source correlation engine that uses a tree of logical conditions (rules) or AND/OR a tree in the correlation process. In each correlation level of the correlation tree (or respective to a "tree" of hyper-alerts) a risk value is updated according to an equation in the paper. The respective correlation and risk assessment flowchart with N correlation levels, each with different number of rules, is shown in [257].

In [151], Ossenbühl et al. are presenting REASSESS (Response Effectiveness Assessment), a response selection model that evaluates negative and positive impacts associated for countermeasures to mitigate network-based attacks. Requirements are automatic deployment, scalability, adaptability, system independency, calculation efficiency, usability and protection preventing unauthorized access by deploying security mechanisms. The negative effects lead from possible service degradation and penalty costs can occur from service level agreement violations. Furthermore, alert priorities are taken into account. REASSESS is based on several stages related to the NIST incident response cycle for the response selection process as shown in Figure 2.22 and follows a sequential execution. The assumptions, namely aggregation and confidence which assume that each alert raised by a detection engine is treated as one attack and a strong confidence with a hundred percent certainty of alerts, however, strongly mitigate the usage of anomaly-based IDSs which might raise a large number of false alerts. Furthermore, REASSESS in its current state is working sequential, not able to cope with multiple IDS nodes, does not use common standards for the exchange of incident related information and is not capable of coping with more advanced attack scenarios due to the lack of an AC mechanism.

An optimal metric-aware response selection strategy using mixed integer linear pro-

[5]`https://www.ossec.net/` (accessed on 05 September 2021)

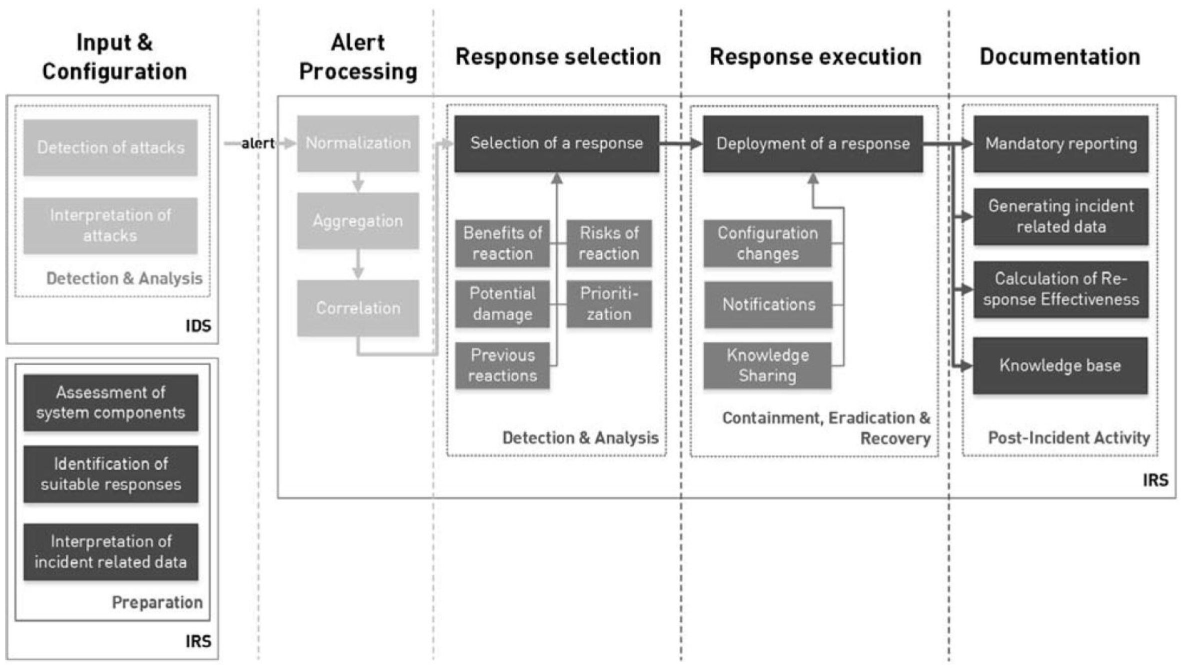


Figure 2.22: Response selection process of REASSESS [151].

gramming to obtain an optimal subset of responses is presented by Herold et al. in [258]. Thus, they were able to provide response strategies much faster and with higher quality than using simplistic heuristics allowing, e.g., a larger number of responses or entities. GhasemiGol et al. proposed a network attack forecasting strategy in [172] by using an attack graph and a dependency graph approach enriched by analyzed IDS alerts (hyper-alerts graph) to identify possible risky nodes and IRS activated responses (multi-level response graph). The utilized attack graph handles the uncertainty of an attack probability by measuring the probability of vulnerability exploitation. Instead of using IDMEF for alert analysis, the authors use their E-correlator, which is a similarity correlation system that functions on raw alerts and outputs a directed hyper-alert graph. They define a response graph with eight levels (refer to Figure 2.25). By the combination of various graph-based approaches, the authors were able to handle uncertainties of current attack graphs and predict future network attacks by the inclusion of additional information. In their consecutive article [259], GhasemiGol et al. extended their work to a foresight model for intrusion response management that includes a response cost estimation based on whether the impact of a response is negative or positive on each level of their multi-level model considering the confidentiality, integrity, and availability parameters. In a recent work [260], Shameli-Sendi et al. are proposing a framework for selecting and deploying optimal countermeasures to intrusions dynamically. Therefore, an optimal countermeasure is identified by evaluating the current and potential damage cost, the accuracy of the countermeasure risk reduction, the impact on QoS and the balance between countermeasure and attack damage cost. An advantage of this framework is that countermeasures are not predictable for an adversary since they are not statically defined.

2.3.2 Intrusion Response Representation

With respect to Herold [238], responses may be reconfigurations of hosts or network elements. Targeting the implementation of reactions on network elements, Herold presents the SNMP and Network Configuration Protocol (NETCONF) defined in RFC 6241 [261]. NETCONF makes it possible to read out, install and manipulate the configuration of devices and is based on a simple RPC layer (Remote Procedure Call). It uses XML-based encryption to transfer both configuration data and protocol messages. Among the advantages are the human readable representation of the data, the reusability of the message structures as well as the easy extensibility. The data model belonging to NETCONF is called YANG (Yet Another Next Generation). This model explicitly and unambiguously defines the structure, syntax and semantics of the configuration and operational data of network devices. It thus offers a uniform interface to their manipulable and operational data. NETCONF has not been defined to inherit information of intrusion responses. However, it could be applied when it comes to network device reconfiguration triggered by an IRS.

The IDIP application and message layer is defined in [262] providing details on the objectives, specification and operations of IDIP and has been used by [252, 253]. IDIP applications use trace messages to describe network-based intrusions which are passed to neighboring devices to trace the path of the intrusion and provide the information necessary for each device along this path to determine an appropriate response. Other IDIP application messages are used to support this tracing and response mechanism. As stated in the specification, IDIP is designed to minimize the size and number of messages required to support intrusion response. Application layer messages are primarily sent only after an intrusion has been detected. Once the response has been initiated, the protocol attempts to only send messages to components that potentially could have witnessed parts of the attack. In addition, IDIP components send reports of the responses to a centralized management component called the discovery coordinator. [262]

Since IDMEF does lack in messages for specifying responses, Klein et al. are introducing the Intrusion Response Message Exchange Format (IRMEF) in [263]. It extends IDMEF as shown in Figure 2.23 by a response class that uses elements that are already defined in the IDMEF message class. Those are the CreateTime, set to the time the response message is created, the DetectTime to schedule the time of response execution, the Source issuing the response, the Target(s) to which the response should be applied, an Assessment field containing the action that should be performed, e.g., kill process and an AdditionalData field containing optional elements, e.g., parameters for the response, e.g., process ID for the process to be killed. The proposed communication protocol between those agents and a console is SNMP in version 3. This scheme does not allow to schedule more actions in an appropriate manner, for example, neither sequential or otherwise timed actions or the interconnection of multiple actions are possible nor the specification of the control flow in more detail [238].

An intrusion response message format similar to the IDMEF is proposed in [259] and represented by XML based on the multi-level model information from [172]. Figure 2.24 shows the corresponding data model consisting of the Intrusion Response Message (IR-Message) and its subclasses providing further response information. The attributes of the IR-Message are a response identification number (Response-ID), a Response-status indicating the response condition being active or inactive and a Response-cost containing the impact of the response on the network entities. The Response-Target contains information about the target with respect to the authors' multi-level response model. Response-Type indicates the kind of responses in terms of impact level and Response-Location, the place of applying responses such as a Firewall. The Response-Action shows the kind of actions that can be applied by responses, e.g., shutdown, reset, block, or notify.

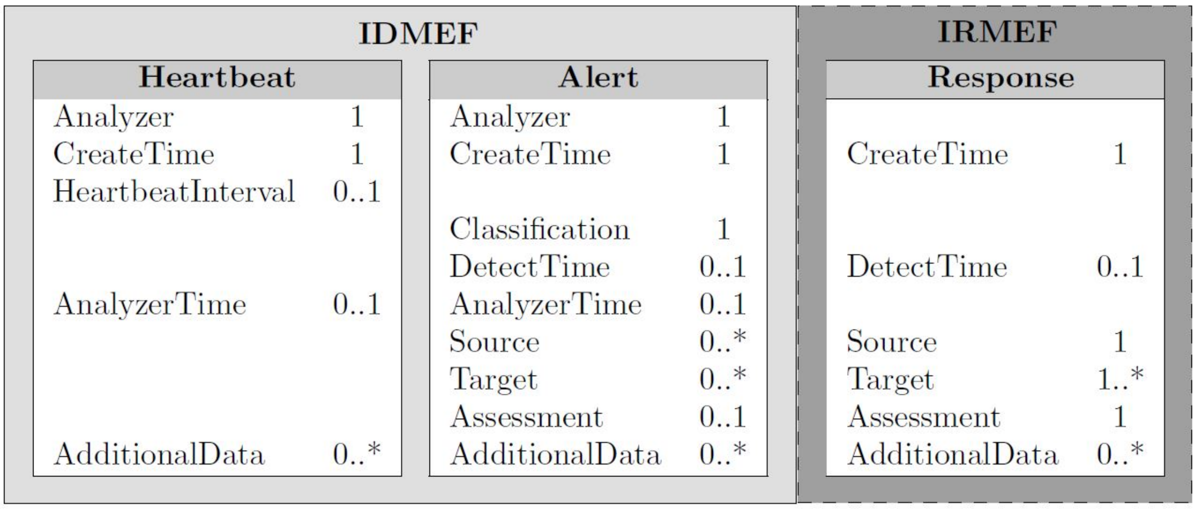


Figure 2.23: Intrusion response message format - IRMEF [263].

2.3.3 Possible Response Measures

Table 2.3 illustrates different classes of response actions with examples that might be applicable, depending on the type of consequence and the involved assets. Those measures are used to neutralize an individual attack or execute preventive measures to secure the system against future attacks of the same type. GhasemiGol et al. define different levels in [259], as shown in Figure 2.25, that can also be applied to map response measures to different levels. Those are composed of *notification-level* (generation of a report or alarm), *attacker-level* (targeting the attacker machine by, e.g., blocking the attacker IP), *vulnerability-level* (patching or updating software to remove vulnerabilities, e.g., CVE countermeasures), *file-level* (block file or change file permissions), *user-level* (block user or reduce user privilege), *service-level* (block affected processes, services or ports), *host-level* (affects victim machine by, e.g., shutting it down) and *unclassified-level*.

Action class	**Description**	**Examples**
Rollback	bring the component back to a saved secure state	restart virtual machine or software
Rollforward	find a new state from which the component can operate	restore or update process
Isolation	perform physical/logical exclusion of the “faulty” components	block attacker’s IP address
Reconfiguration	reconfiguring a system, component or reassign tasks to others, activate spare components	reconfigure network routing
Reinitialization	check/record new configuration and update system tables	restart a TCP connection

Table 2.3: Response and recovery action classification (cf. [264]).

According to [265], responses with respect to virtual cloud systems can be categorized into filtering (e.g., updating upstream filters to block traffic), rate limiting (attempts to relieve the pressures on bottlenecks), adapting the use of virtual machines (e.g., increase or decrease its number) and identifying the attack source (using trace-back techniques).

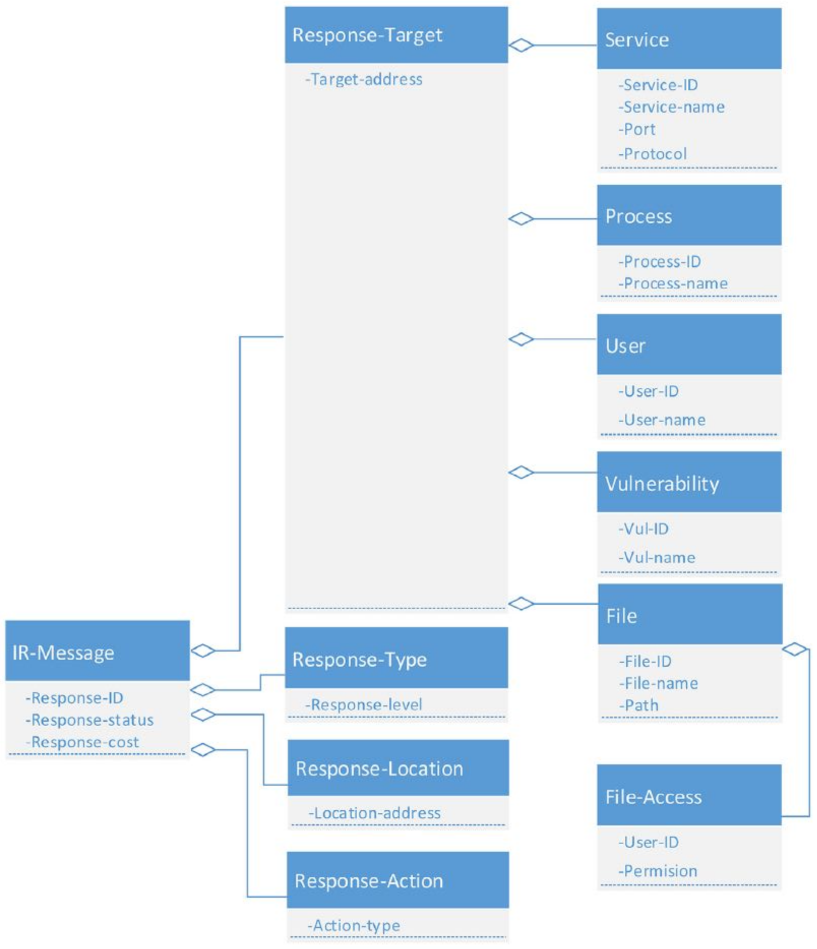


Figure 2.24: Intrusion response message format - IR-Message [259].

A more generic categorization into passive and active responses with lists of common responses per category based on the work of [4, 239, 266] and inspired by the multi-level categorization of [259] is illustrated in Figure 2.26. The dotted arrow indicates that some response measures (unclassified-level) categorized under passive might also be active ones depending on their interpretation and level of automation.

Chapter 6 introduces Uncoupled MAC providing adaptive IDS-functionality by utilizing sampling and self-regulation that can be used to adapt IDSs as a response measure. The Uncoupled MAC algorithm is a cryptographic scheme that applies message authentication codes decapsulated from the original packet and is able to dynamically adjust its sampling parameters depending on the detection of both, a message's data integrity as well as its authenticity violation. However, in the following, an exemplary response measure for the reconfiguration of a network infrastructure applying SDN, e.g., isolate a malicious host and reroute its traffic, is provided.

Network Reconfiguration Leveraging SDN

Note: The following paragraph has been published by the author in [6] presenting a generic incident handling framework. Exploiting SDN technology to reconfigure the networking environment is a further reaction possibility. Controlling network flows dynamically enables to separate malicious (or suspicious) network flows from benign ones dynamically. For example, supposed that a NIDS detects some suspected flows, the flows can be

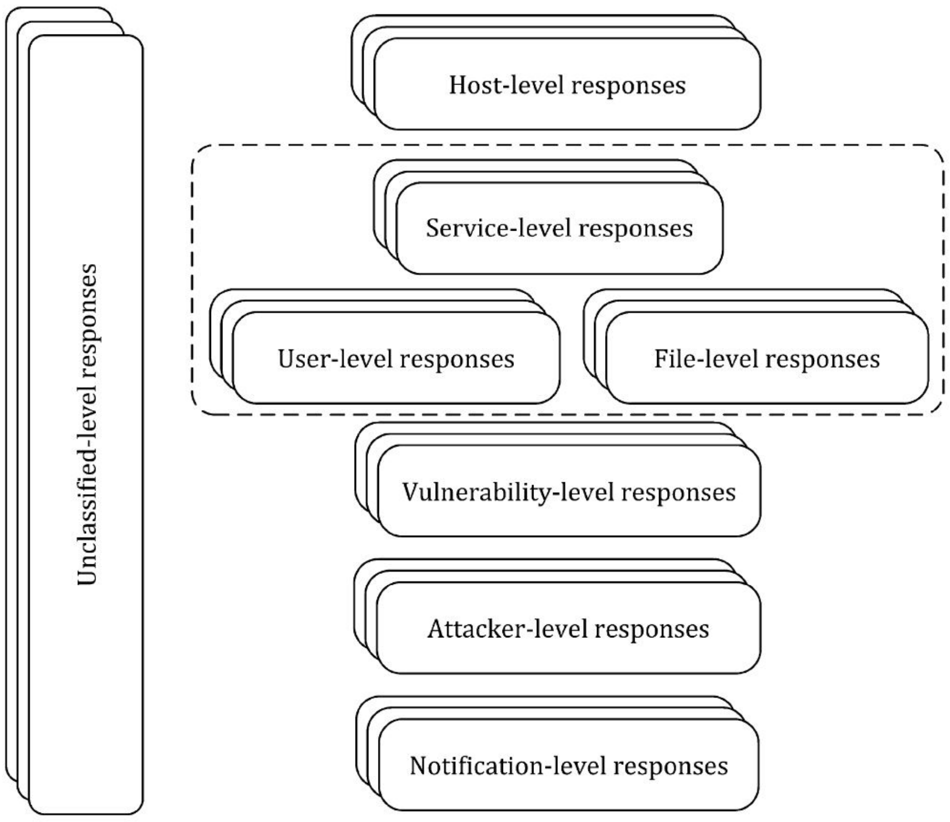
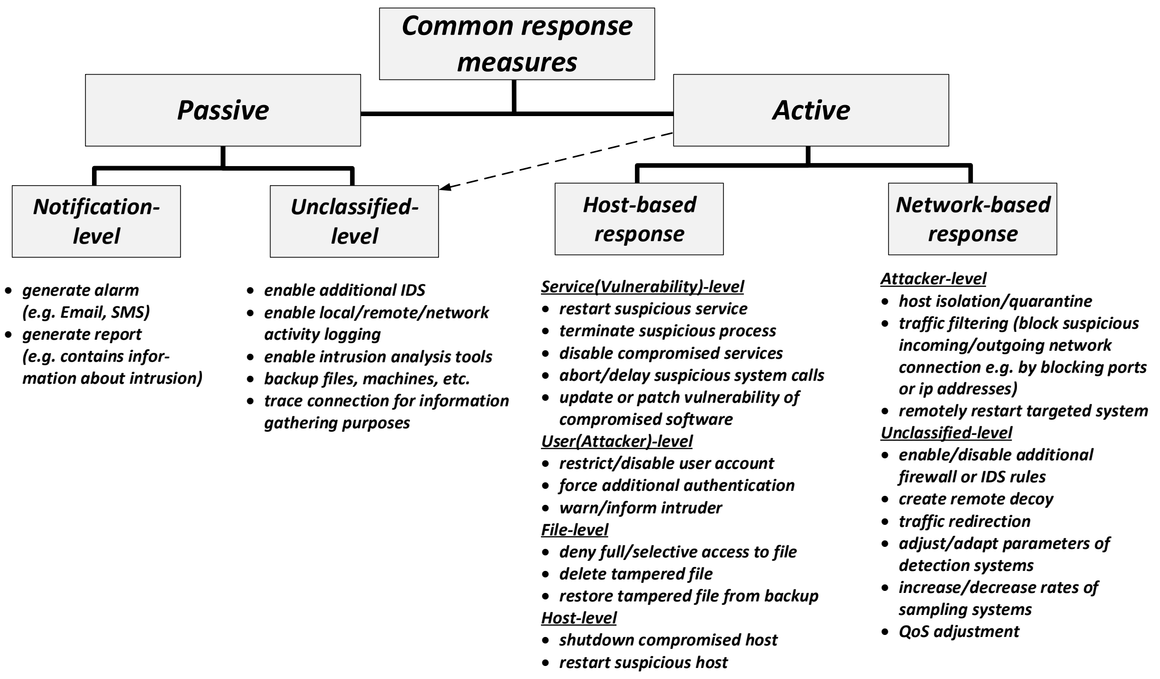


Figure 2.25: Different intrusion response levels [259].

rerouted for in-depth investigation, e.g., in a honeypot [267]. Further firewall functionality can be implemented using SDN. When a switch receives a new packet and there is no rule matching this packet in the flow table, it reports it to the SDN controller which forwards the packet to the firewall application. The firewall checks whether the incoming packet violates security policies or not and enforces a new flow rule accordingly. This rule is delivered to the switch by the controller and all future packets from the flow of the first packet would be handled directly in the switch without the need to interact with the controller again [268]. Another possibility applying SDN to respond to a specific incident is through network separation. In traditional networks, the common way to separate a network is employing Virtual LAN (VLAN) technique, which adds specific IDs in a packet header (12-bits VLAN ID field). However, VLAN technology incurs scalability limits in large-scale networks, since it can only assign 4,096 different virtual networks. SDN-based separation solutions provide the capability of different level abstractions with desired security properties, which not only separates the network segments efficiently at scale, but also veils the physical view of networks to users [269].

A “pluggable” software platform aimed to provide centralized administration and experimentation for anomaly detection techniques in software defined data centers is provided by [270]. The proposed SDN-PANDA consists of three controller-centric application modules responsible for (1) data collection and pre-processing of switch aggregated flow statistics, (2) anomaly detection based on a flexible interface and (3) performing response actions defined on standard policies independent of the anomaly detection method. The response policies must be accurately defined such that they only mitigate the identified attack and not cause a loss of service by, e.g., dropping packets. To address this issue they



Common response measures
Passive
Active
Notification-level
Unclassified-level
Host-based response
Network-based response
• generate alarm (e.g. Email, SMS)
• generate report (e.g. contains information about intrusion)
• enable additional IDS
• enable local/remote/network activity logging
• enable intrusion analysis tools
• backup files, machines, etc.
• trace connection for information gathering purposes
Service(Vulnerability)-level
• restart suspicious service
• terminate suspicious process
• disable compromised services
• abort/delay suspicious system calls
• update or patch vulnerability of compromised software
User(Attacker)-level
• restrict/disable user account
• force additional authentication
• warn/inform intruder
File-level
• deny full/selective access to file
• delete tampered file
• restore tampered file from backup
Host-level
• shutdown compromised host
• restart suspicious host
Attacker-level
• host isolation/quarantine
• traffic filtering (block suspicious incoming/outgoing network connection e.g. by blocking ports or ip addresses)
• remotely restart targeted system
Unclassified-level
• enable/disable additional firewall or IDS rules
• create remote decoy
• traffic redirection
• adjust/adapt parameters of detection systems
• increase/decrease rates of sampling systems
• QoS adjustment

Figure 2.26: List of common response measures (cf. [239, 4, 259, 266]).

propose to apply redundant services in different spots of the infrastructure and reroute the traffic in the case of incidents which allows, e.g., the investigation of the compromised host while still guaranteeing the availability of the service. In [37] the authors leverage SDN and Network Function Virtualization (NFV) technologies for incident response in industrial control systems. They provide potential response use cases including rerouting attacker traffic to a honeypot, changing forwarding rules to drop communications or transfer services from compromised devices to redundant ones using virtualized resources. In a proof of concept they are extending MiniCPS providing SDN functionality and deploy an IDS in the SDN controller which is working threshold-based and compares actual and estimated sensor values. Incident response functionality is quite limited following pre-configured policies such as discarding packets from the compromised sensor and replacing them with the estimated values of the IDS. Afterwards, the SDN controller modifies the flow table of the SDN-capable switch to leverage the response. The authors state that although their application is simple, the model can be extended to include different detection mechanisms and various incident-response policies for different types of attacks. A similar work [271], utilizing SDN and NFV, proactively detects (low-level deep packet inspection) and mitigates botnets in 5G mobile networks by a dynamic reconfiguration (isolating botnet communication).

3 Unsupervised Feature Selection for Outlier Detection on Streaming Data to Enhance Network Security

This chapter provides details on the ***U**nsupervised **F**eature **S**election for **S**treaming **O**utlier **D**etection* (**UFSSOD**) algorithm and is organized as follows — An extensive review of the state-of-the-art for unsupervised FS from two different viewpoints (SD, OD) is provided in Section 3.1 and compared with thoroughly engineered requirements for this tasks. In Section 3.2, details on the conceptualization, operation modes and operation principle of UFSSOD can be found to achieve unsupervised FS for OD on SD. In Section 3.3, the performance of two representative algorithms for FS with a focus on SD and OD, FSDS and IBFS is evaluated, and compared with UFSSOD using extensive measurements. The discussion reveals that UFSSOD is comparable to (offline) IBFS while working online and shows that FSDS cannot be applied for the purpose of OD.

3.1 Requirements Engineering and Comparison with Related Work

FS has become a mighty tool in the field of network security to enhance the performance of IDS. For better clarity of the plenty of available solutions in this field, this section has been structured in such a way that first requirements for FS are engineered. Second a thorough review of state-of-the-art work which is structured in two parts is compared with the requirements manifesting the research gap of FS for OD in a SD setting.

3.1.1 Requirements with Respect to Feature Selection for Outlier Detection on Streaming Data

Much attention has been paid towards OD on SD in the field of network security monitoring over the past years since data is increasingly generated, e.g., with high velocity, in tremendous volume and afflicted with the phenomenon of concept drift due to their dynamic. Many state-of-the-art works try to come up with new algorithms or try to tune the algorithm setting in order to improve their classification performance to the best. Most of them compete on the same (outdated) benchmark data set but might perform insufficiently with other data sources or in real-world applications. Due to the technological advancement in the last years, the amount of unlabeled data generated across many scientific disciplines, such as text mining, genomic analysis, social media, and intrusion detection has steadily increased which demands unsupervised learning leading to the first requirement **[R-FS01]**. Apart from its supervision, the application of AutoML methods is not possible due to its non-applicability on SD. We state that due to those circumstances a certain amount of false positives and false negatives is acceptable for the sake of a subsequent applied root cause analysis which is conducted on details provided by the ML algorithms. This position contradicts to the general approaches that tailor their algorithms to specific data sets by highly tuned parameter sets or by a specific system architecture. However, the high complexity by various intervolving or complex algorithms, the high degree of freedom in the ML pipeline and the coherent resource requirements oppose the application in real-world scenarios.

If FS is considered in the same setting as online OD, requirements roughly coincide. Compared to models trained in batch (offline) learning, methods for online OD and for FS respectively must be more sophisticated in a few points. Especially, training from imbalanced data from an infinite data stream in a pass-efficient way, meaning one pass at a time **[R-FS02]**, is a huge challenge since some approaches need to temporarily store the incoming data. Also the time-varying nature of an infinite data stream compared to a fixed set might lead to concept drifts. With the different types of concept drifts (sudden, gradual, incremental, recurring, blips) [272], online processing, especially unsupervised, will be impeded. Models in batch approaches would quickly be outdated and lead to a performance degradation such that online algorithms must be able to continuously retrain or update their model to handle this situation **[R-FS03]**. Due to the steady growth of high-dimensional data volumes, offline approaches storing the entire data for training or scanning the data set multiple times (many passes over the data) lead to considerable memory limitations and demand lightweightness **[R-FS04]**. The major downside of online methods in general, especially unsupervised methods compared to their batch opponents, is the poorer performance when it comes to classifying abnormal and normal data instances. However, we strongly support the hypothesis in [34], when considering critical streaming applications as for detecting network-based malicious activity, a fast model, even with less accuracy, is preferred. However, applying FS shall at least improve the classification performance of OD methods **[R-FS05]**.

In this work, the focus of FS and OD on SD is not in the context of time-series data as it is the object in many other research works [31, 32]. SD within this context is any flow of information characterized by incoming instances of data chunks that might be processed in near real-time. The information of time, which might be a specific piece of data received, may not represent the time of measurement and may thus not be an important feature at all. Regarding time-series data, the order and time are fundamental aspects with a central meaning of the data such that, based on past observations, predictions towards the future time can be made. It is the goal to detect outlying time-series patterns based on temporal dependencies, rather than independent outliers in data [120]. More generally, data streams, becoming more and more prevalent in real-world applications, can broadly be classified into *streaming data* and *streaming features*. The first-mentioned defines incoming data in a record-by-record manner having a static number of features d while in a feature stream new features can be generated dynamically. Thus, the stream $\{\boldsymbol{X}_t \in \mathbb{R}^{n_t \times d}, t = 1, 2, ...\}$ is a continuous transmission of data instances (data points) which arrive sequentially at each time step t. The count of features is denoted as d (dimension) and $\boldsymbol{x}_t$ the n_t-*th* d-dimensional most recent incoming data instance at time t. In the field of cyber security and the monitoring of network data the number of features is fixed **[R-FS06]** and can be defined *a priori* by an expert since incorporating domain knowledge can help select relevant features to improve learning performance greatly. For network-based features, one may distinguish between basic features (derived from raw packet headers (meta data) without inspecting the payload, e.g., ports, MAC or IP addresses), content-based features (derived from payload assessment having domain knowledge, e.g., protocol specification), time-based features (temporal features obtained from, e.g., message transmission frequency, sliding window approaches) and connection-based features (obtained from a historical window incorporating the last n packets) [80]. Those features are streaming from a single source (single-view) **[R-FS07]**, e.g., a raw network interface, as statistics from network switching elements or in form of log-files from devices. However, a human expert cannot be expected to recognize correlations from the variety and multitude of multivariate features **[R-FS08]** occurring in high-dimensional real-world applications. Typically, this manual feature engineering is a time-consuming ad hoc process composed of trial and error to determine the features most relevant to the detection problem which inhibits the application of ML to network security [30, 273]. Thus, FS is needed whose output can either be a ranked list of the input features or a dedicated subset. The former allows a better insight into which features contribute the most in order to select the top-performing-features **[R-FS09]**. The requirements are summarized in the list below.

- R-FS01: operation without knowledge of data labels (unsupervised)
- R-FS02: processing incoming data sequentially record-by-record (SD)
- R-FS03: adaptability in highly dynamic networks (dealing with concept drift)
- R-FS04: lightweight in terms of space and computational complexity
- R-FS05: applicable for the classification task on imbalanced data i.e. OD (no clustering)
- R-FS06: incoming data has the same cardinality (static amount of features - no streaming features)
- R-FS07: single data source with the assumption that the data is independent and identically distributed (single-view)
- R-FS08: dealing with multivariate features (identifying redundant and irrelevant (univariate) features)
- R-FS09: providing a ranked list of features to select (feature scoring)

3.1.2 Feature Selection for Streaming Data

For the purpose of anomaly detection we state that the cardinality of features is known beforehand by a domain expert's feature engineering process but the number of data instances is not known or even infinite. Thus, it is impractical to wait until all or a significant amount of data is available to perform FS. A batch-mode method especially in highly dynamic networks with concept drifts is not able to select relevant features in a timely manner. In contrast, solutions like OSFSMI [274], UFSSF [275] or DOFS [276] are facing, independent of their supervision, feature streams in which the number of instances is fixed and the features arrive one-by-one or in groups. In domains where the global feature space is defined by a domain expert, these methods might only be applicable if they can fix the number of features and are capable to operate on streaming data.

Much work is dedicatedly tailored towards the clustering task [277, 278, 279] which can immediately be excluded for the application of anomaly detection as a classification task on imbalanced data sets. However, other work does not explicitly focus on clustering but evaluate their approaches applying those algorithms. Therefore, the question arises to what extent such algorithms can be used for the task of OD. From a supervision perspective, a supervised FS approach for binary classification on SD called OFS has been proposed in [280] that is able to learn either on all features of a training data instance as well as only a small number to select the relevant feature subset. The authors of [281] focus on data stream mining classification algorithms often applied in machine learning with applications in network intrusion detection. Their real-time FS method is called MC-NN which describes a concept drift detection method for data stream classification algorithms including the capability of feature tracking. The method can either be applied as a real-time wrapper or a batch classifier independent of any learning algorithm. However, the authors state that their study was not concerned with the classification capabilities of MC-NN, but with the behavior of the underlying model during concept drift. A similar approach, we denote it as EaSU, hypothesizes that features can be dynamically weighted in order to augment the importance of relevant features and diminish the importance of those which are deemed irrelevant according to observed feature drifts [282]. Therefore they introduce the concepts of entropy and symmetrical uncertainty in a sliding window approach to track the relevance of features and could enhance k-Nearest Neighbor, naive Bayesian and Hoeffding Adaptive Tree algorithms in a supervised setting.

With a focus on unsupervised FS, a comprehensive and structured review towards the most referenced state-of-the-art work has been given in [283]. However, the review does not give any details for the specific domains such as text data, link data nor to SD. An online unsupervised multi-view FS method, called OMVFS, is proposed in [284], which in addition to the features of FSDS is also capable of handling multi-view data. It performs unsupervised FS using the concept of nonnegative matrix factorization with sparse learning and processes incoming multi-view data into several small matrices by using a buffering technique to reduce the computational and storage cost. FSDS as proposed in [285] is, to the best of our knowledge, the only unsupervised FS scheme truly operating in single-view on SD for clustering and classification applications. It utilizes the ideas of matrix sketching where a sketch of a smaller matrix still approximates its original to maintain a low-rank approximation of the entire observed data at each time step. Hence, they modified the so called Frequent Directions algorithm proposed by Liberty [286] and use regression analysis to generate a feature importance score with each incoming data. It is space and time efficient requiring only one pass over the data.

The related work regarding approaches towards unsupervised FS for SD is compared to the requirements from Section 3.1.1 in Table 3.1. Most of the approaches can be neglected due to their supervision perspective, their design towards streaming features rather than SD, their focus on clustering task or their need for training instances. Two approaches, FSDS and OMVFS, even if evaluated on clustering algorithms, seem promising to function on the classification class. However, it must be evaluated if they are able to produce valid results in imbalanced data for the purpose of OD. Although OMVFS might be capable of functioning in single-view, it is designed for multi-view. Therefore, as of now, we set our focus on FSDS.

Table 3.1: Comparison of existing FS work for SD with the requirements defined in Section 3.1.1 (✓ and ✗ denotes the requirement is either fulfilled or not, ∅ denotes missing information to analyze the respective requirement, (++/+/-) for R-FS04 and R-FS08 denotes, as objectively as possible, how well the requirement is fulfilled, R-FS02 and R-FS03 are combined since R-FS03 is a phenomenon associated with R-FS02 and the existing work in this table fulfills both equally).

Work	R-FS01	R-FS02\R-FS03	R-FS04	R-FS05	R-FS06	R-FS07	R-FS08	R-FS09
OSFSMI [274]	✓	✓	-	✗	∅	✓	++	✓
UFSSF [275]	✓	✓	++	✗	✗	✓	-	✗
DOFS [276]	✓	✓	-	✗	✗	✓	+	✗
OFS [280]	✗	✓	++	✗	✓	✓	++	✗
MC-NN [281]	✓	✓	∅	✗	✓	✓	++	✓
EaSU [282]	✗	✓	++	✗	✓	✓	+	✓
OMVFS [284]	✓	✓	++	✗	✗	✓	++	✓
FSDS [285]	✓	✓	++	✗	✓	✓	++	✓

3.1.3 Feature Selection for Outlier Detection

FS for IDS must be able to determine the most relevant features of incoming data, e.g., network traffic, to minimize the cardinality of the set of selected features which affects their effectiveness without dropping features indicating abnormal behavior. A review on FS algorithms for anomaly-based network IDS can be found in [287] dividing them into bio- and non-bio-inspired ones. However, the authors do not focus on the supervision perspective, thus, their reviewed work only leverages different supervised FS methods. None of the reviewed approaches seem to address the requirements defined in Section 3.1.1. This is discussed by the authors since most of the methods are based on the wrapper nature or on a combination of FS algorithms which increases the computation and time complexity.

A lot of FS methods [288, 289, 290, 291, 292] within the context of OD, even recent ones [293, 294, 295], are operating supervised. Those approaches are mainly performing multiple evaluations due to their wrapper nature on a dedicated data set typically KDD'99 or NSL-KDD to test different feature subsets using a certain classifier. Within their setting, they achieve the best subset of features resulting in an optimal classification performance. However, facing highly dynamic networks with large volumes of data and high velocity, an analysis is necessary that is beyond the aforementioned approaches. Some of the authors even state that their approaches take a lot of time to train [292] or to use other classifiers and verify them under more realistic environments [290]. However, with the improvements mentioned in the outlook of [292], it is the only work in the field of FS for OD considering support for online processing even though still being of supervised nature. For the rest of this section, we set our focus on unsupervised approaches since those are more meaningful for real-world applications of OD methods.

A limited amount of papers deal with FS approaches for the purpose of OD in an unsupervised nature. The algorithm in [296], denoted in this chapter as FS-CVI, uses a genetic algorithm to optimize a cluster validity index, a measure of how well a clustering algorithm manages to identify and assign clusters in a data set, over a search space consisting of feature subsets. Classification based on the proposed subsets from the genetic algorithm is performed in such a way that clusters are being built which are representing legitimate traffic and others more prevalent pertaining to intrusions. An unsupervised backward elimination FS algorithm based on the Hilbert-Schmidt independence criterion, named BAHSIC-AD, is proposed in [24]. The authors compare their approach with a recent Spectral Ranking for Anomalies algorithm and show that a combination of this algorithm and BAHSIC-AD is able to detect point, collective and contextual anomalies. Their evaluation is performed on real-world data sets taken from various application domains. However, the field of cyber security with intrusion detection is not covered. With a focus on categorical features, [297] proposed a method that we denote as MI-FS based on mutual information measure and entropy computation for FS that is employed using two OD methods: AVF and Greedy method. Thus, within this filter approach categorical features are selected that expect to highlight the deviations characterizing the outliers with minimum redundancy among them by performing feature-wise entropy computation. Also aimed for categorical data, [298] proposes CBRW_FS, a coupled biased random walks approach providing feature weights for categorical data with diversified frequency distributions, and in [109] a Coupled Unsupervised Feature Selection (CUFS) framework instantiated into a parameter-free Dense Subgraph-based Feature Selection (DSFS) method. With their instantiation called CINFO in [299], the authors further developed CUFS by using unsupervised discretization methods like equal-width and equal-frequency to adopt the methods to numeric data rather than categorical data. Another approach is given in [300] with Unsupervised Feature Selection and Cluster Center Initialization, denoted in this work as UFS_CCI. It derives feature scores as the difference of feature entropy from unlabeled data by computing the ratio of the maximum count of occurring values by the total amount of samples. Assuming that the value of the result is in between a threshold range, UFS_CCI considers the feature to be non-redundant. However, the main focus of this work lies on clustering samples with reduced dimension for intrusion detection and less on feature relevancy for OD. In CINFO an outlier detector generates scores in an unsupervised fashion that are fed into a supervised learning sparse Lasso regression based on pseudo-labels from outlier candidates. Those steps are executed in an iterative manner to build a sparse sequential ensemble OD framework and is even further improved by bagging. Although working unsupervised overall to obtain a dependent set of an OD and FS model, it requires all of the data objects beforehand. Similarly, ODEFS proposed in [301] integrates OD and FS into a combined framework by also using ideas from [299]. The ODEFS framework consists of multiple feature learning components whose individual outlier scores based on diverse feature subsets are weighted together in a final score. Therefore, (i) outlier thresholding based on Cantelli's inequality using the results from the outlier detector, in this example the distance-based LeSiNN, is applied to obtain possible outlier candidates. Each feature learning component (ii) randomly chooses unlabeled examples from the initial data and from the set of outlier candidates to be fed into a pairwise ranking formulation that embeds FS into OD. The training process (iii) utilizes the thresholded self-paced learning approach to learn example and feature weights which will in (iv) be used to compute the final score by the idea of boosting to combine the outlier score vectors with associated loss as weights. Although, the approach by ODEFS seems promising, it is designed to know the input data beforehand and heavily relies on a training phase. Thus, an application in the context of SD is forfeited. The work of [302] exploits the strengths of the widely known iForest OD algorithm [51] to handle large data size and high dimension with many irrelevant features while having the potential to identify those to reduce dimensionality. Thus, their proposed method IBFS calculates feature imbalance scores using an entropy measure during the training process of iForest and is immune to feature scaling. Features are ranked in a descending order according to their feature scores. However, the authors did not discuss how to select the top-features rather evaluating their approach on multiple settings in an offline fashion.

The related work regarding approaches towards unsupervised FS for OD is compared to the requirements from Section 3.1.1 in Table 3.2. Most of the approaches can be ignored due to their supervision perspective, their demand for training data or their wrapper-like nature that are highly tuned and optimized for a data set and the ML classifier. Only a few test their approach on different data sources or with different classifiers. Good subsets of features for the purpose of OD are obtained after multiple iterations in a supervised manner or offline with training data which are often computationally costly. Most of them lead the argument to use FS in order to reduce the performance costs for OD but do not discuss the significant costs necessary to obtain a relevant subset that is tailored to one data set and a corresponding classifier. Despite its supervision, the only work considering the application in online processing in their future work is [292]. In real-world applications with highly dynamic networks, FS must work online to deal with concept drift, tailored to OD without multiple iterative validation rounds and must function unsupervised because of the lack of labeled data. IBFS is the only approach that might be able to satisfy those requirements since it works unsupervised and is tailored for the purpose of OD by exploiting the nature of iForest. Numerous recent advancements of iForest in the streaming setting [124, 123, 52], in particular, let IBFS constitute a promising candidate for FS on SD.

Table 3.2: Comparison of existing FS work for OD with the requirements defined in Section 3.1.1 (✓ and ✗ denotes the requirement is either fulfilled or not, ∅ denotes missing information to analyze the respective requirement, (++/+/-) for R-FS04 and R-FS08 denotes, as objectively as possible, how well the requirement is fulfilled, R-FS02 and R-FS03 are combined since R-FS03 is a phenomenon associated with R-FS02 and none of the existing work in this table fulfills both).

Work	**R-FS01**	**R-FS02\R-FS03**	**R-FS04**	**R-FS05**	**R-FS06**	**R-FS07**	**R-FS08**	**R-FS09**
FS-CVI [296]	✓	✗	∅	✓	✓	✓	-	✗
BAHSIC-AD [24]	✓	✗	-	✓	✓	✓	∅	✓
MI-FS [297]	✓	✗	++	✗	✓	✓	+	✓
UFS_CCI [300]	✓	✗	∅	✗	✓	✓	++	✗
CBRW_FS [298]	✓	✗	-	✗	✓	✓	+	✓
CUFS-DSFS [109]	✓	✗	-	✗	✓	✓	+	✓
CINFO [299]	✓	✗	+	✓	✓	✓	++	✓
ODEFS [301]	✓	✗	+	✓	✓	✓	++	✓
IBFS [302]	✓	✗	++	✓	✓	✓	++	✓

3.2 Unsupervised Feature Selection for Streaming Outlier Detection

Existing work targeting FS for SD is mainly supervised and for the clustering, regression or classification task on balanced data. FS for OD is either supervised or in the unsupervised case, where even fewer publications are available [109], not capable of coping with SD. Most of the latter requires a dedicated training phase thus making it impossible to function for SD in near real-time. However, there is limited work of FS for OD because it is challenging to define feature relevance given its unsupervised nature. Some work defines relevancy in the context of correlation between features which is not really useful for detecting outliers since some features can be strongly relevant for OD but do not correlate to others. In addition, if there is no dedicated batch training phase, as it is the case with SD, FS becomes a challenging task. To the best of the authors' knowledge unsupervised FS for OD on SD is a completely new field. Thus, we propose **UFSSOD** referring to ***U**nsupervised **F**eature **S**election for **S**treaming **O**utlier **D**etection* that can be applied to improve off-the-shell online OD methods.

However, the two following assumptions are made in order to deal with the problem that is mostly neglected in the literature due to either the supervised nature or the batch approach but is present with FS on SD within the context of OD. Discarding irrelevant or redundant features in time t due to unsupervised and streaming FS might contain outliers in time $t + i$. Therefore we assume that (i) attacks are complex and indicators of attacks resulting in outliers might affect not only the one feature being identified as irrelevant or redundant, thus dropped, and (ii) attacks and their affecting outliers predominantly correspond to the same features. Particularly the second could be shown in [303] where some features have been more significant over multiple attack scenarios, e.g., *B.Packet Len Std*, *Flow Duration* or *Flow IAT Std*. Most of the state-of-the-art work of FS within the context of OD does not discuss this dilemma since they either work supervised and know which features contain the outliers or work in an offline setting having training data available. Therefore they know all the data including those containing outliers beforehand. In the next two sections we present the conceptualization alongside operation modes for UFSSOD and provide its internal functionality with regard to cluster feature scores in order to obtain the top features for OD on SD.

3.2.1 Operation Principle

The operation principle of UFSSOD is shown in the block diagram of Figure 3.1. Data instance (data point) $\boldsymbol{x}_t$ with dimension d of the data stream $\{\boldsymbol{X}_t \in \mathbb{R}^{n_t \times d}, t = 1, 2, ...\}$ streams into the module UFSSOD at each time step t composed of the global feature space $\mathcal{F} = \{f_1, f_2, ..., f_d\}$. UFSSOD is capable of computing a ranked list of features contributing the most for the purpose of OD leading to a subset $\mathcal{F}_S = \{f_{i1}, f_{i2}, ..., f_{i\gamma}\}$ of the top-γ-features for each time step t where a higher value of f_i indicates a highly contributing feature for the purpose of OD. Using $\mathcal{F}_S$, $\boldsymbol{x}_t$ is reduced to $\boldsymbol{x}_t^*$ where $\boldsymbol{x}_t^* = \boldsymbol{x}_t \cap \mathcal{F}_S$ applies. Besides, if desired, UFSSOD is able to produce possible outlier candidates by providing scaled outlier scores $\tilde{y}_0$ for each $\boldsymbol{x}_t$ without increasing the overall complexity in big $\mathcal{O}$ notation referring to [119]. Algorithm 1 provides a high-level overview of UFSSOD's operation principle consisting of the scoring and clustering functionality for feature scores, denoted as *ufssod_scoring* and *ufssod_clustering*. The internal working of both key modules is described in detail in Section 3.2.3.

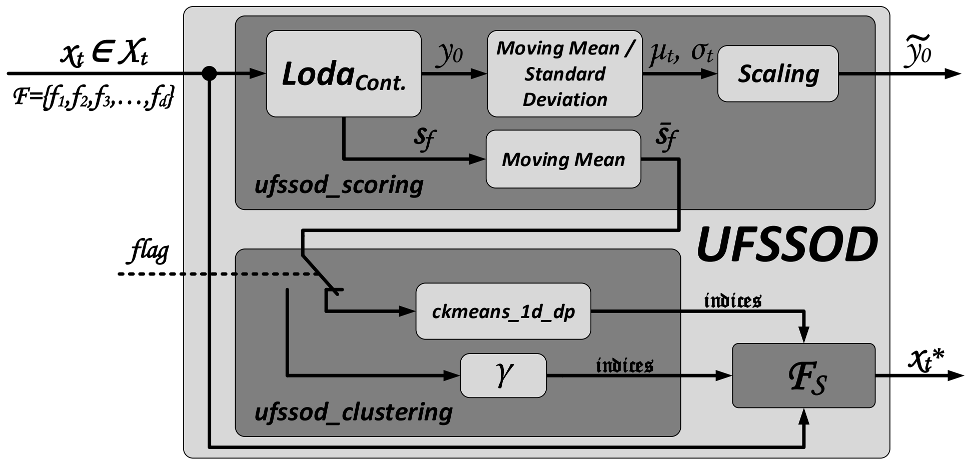


Figure 3.1: Block diagram and operation principle of UFSSOD

Algorithm 1: High-level operation principle of UFSSOD.

```
Input: A sample $\boldsymbol{x_t}$ and a $flag$ to control ufssod_clustering
Output: A reduced sample $\boldsymbol{x}_t^*$ based on feature subset $\mathcal{F}_S$
1 $\tilde{y}_0, \bar{\boldsymbol{s}}_f \leftarrow$ ufssod_scoring($\boldsymbol{x}_t$)        ▷referring to Algorithm 2
2 if flag is set then
3     indices ← ufssod_clustering($\bar{\boldsymbol{s}}_f$)        ▷referring to Algorithm 3
4 else
5     indices ← the lastly acquired indices
6 $\boldsymbol{x}_t^* \leftarrow \boldsymbol{x}_t$[indices]
7 return $\boldsymbol{x}_t^*$
```

3.2.2 Operation Modes

The conceptualization for an online unsupervised FS in an application for streaming OD is depicted in Figure 3.2 showing the sequential operation mode for UFSSOD. Independent of the applied operation mode, in the starting point t_0, where no FS has been performed, or the initialization of each algorithm takes place, γ is set to d in order to work with all features. The module *Unsupervised Online OD* operates an online capable unsupervised OD algorithm, e.g., xStream [120], iForest_ASD [122] or Loda [119], and uses the proposed subset $\mathcal{F}_S$ to boost its performance yielding a more precise outlier score y_i based on $\boldsymbol{x}_t^*$. One might also apply multiple classifiers in parallel in the Unsupervised Online OD module to exploit the power of a combination of m individual classifiers depending on the available resources. For this, one might define a certain resource budget r comprising of computational resources such as CPU or wallclock time as well as memory usage and let $m = f(r)$ be a function of r. Thus, even during runtime, some classifiers might be turned off for the sake of resource preservation. The outlier scores y_0 and y_i are normalized using *Gaussian Scaling* proposed by [304] to $\tilde{y} = max\{0, erf(\frac{y-\mu_t}{\sigma_t\sqrt{2}})\}$ in which $erf()$ is the Gaussian Error Function which is monotone and ranking stable. The moving average μ_t and standard deviation σ_t of the outlier scores until time t are obtained by applying, e.g., the well-known Welford's algorithm [305] or one of the methods proposed in [32]. Scaling is necessary when combining algorithms of different nature with different characteristics. For instance operating Loda$_{\text{Two Hist.}}$ with different window sizes will result in different averaged score values due to the different state of knowledge on normal data. With increasing window sizes the model becomes more accurate while seeing larger amounts of normal data which are then scored less compared to models obtained from smaller window sizes. With the mean and standard deviation proportion of the *Gaussian Scaling* this difference will be eradicated and the Error Function the scores are tailored to the interval between 0 and 1. With the normalized outlier scores, a final score can be computed that might also be weighted. The confidence level of the Unsupervised Online OD module is typically higher than those of UFSSOD since it operates on the reduced representation of a sample $\boldsymbol{x}_t^*$ and thus one might give a higher weight to its score. We use $\tilde{y} = g_0\tilde{y}_0 + \sum_{i=1}^{m} g_i\tilde{y}_i$ depending on the weights g_0 for UFSSOD and g_i for each classifier (typically $g_i > g_0$) where $g_0 + \sum_i g_i = 1$ and $\tilde{y} \in [0, 1]$ applies. Using domain expertise, a reasonable threshold can be determined over runtime yielding a decent classification performance that assigns a binary value from the set $\{normal, abnormal\}$ for each $\tilde{y}$.

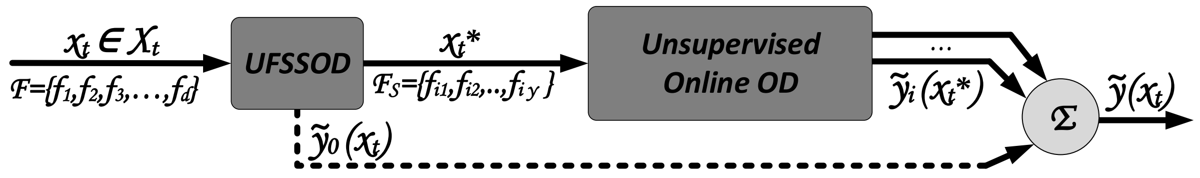


Figure 3.2: Conceptualization of module interaction for unsupervised FS for OD on SD (sequential approach).

Since the UFSSOD module is able to provide different feature scores at each time step t, resulting in possible other top-γ-features than at time $t - i$, the downstream applied Unsupervised Online OD module might be capable of dealing with those changing features. If this is the case, the setup can truly work online, meaning that UFSSOD provides an updated version of the feature subset F_S with a potentially changing cardinality that is used by the Unsupervised Online OD module to detect outliers in each t. To the best of our knowledge, only xStream is able to handle this situation. Moreover, xStream not only handles a varying cardinality of the feature subsets but is also able to deal with completely newly occurring features. However, the latter is not the focus of this work since for domain specific applications features are assumed to be known beforehand. Since most of the state-of-the-art unsupervised OD algorithms operate on a fixed set of features with a fixed cardinality, a windowed approach can be utilized if the subsequently applied online OD algorithm is able to dynamically change its model during runtime. Thus, it can replace the old feature subset with the newly proposed one after the current time window elapsed. Possible algorithms would be iForest_ASD, which rebuilds its forest after a certain condition is met, thus also allowing to rebuild it with a different feature subset, or $\text{Loda}_{\text{Two Hist.}}$ and HS-Trees using windowing where models are replaced with the ones currently built and the ones that will be built can use the current feature subset. With respect to $\text{Loda}_{\text{Two Hist.}}$, the old histograms built and classified with $\mathcal{F}_{S(t-i)}$ will be replaced with the new histograms using $\mathcal{F}_{S(t)}$, also allowing to change the cardinality (different top-γ-features).

If this sequential operation limits the throughput, a parallel approach can be utilized where both modules UFSSOD and Unsupervised Online OD work in parallel. As depicted in Figure 3.3 the online OD algorithm for each t operates with the previously proposed subset $\mathcal{F}_{S2}$, while the UFSSOD module continuously proposes a new $\mathcal{F}_S$, which is stored within $\mathcal{F}_{S1}$. The Unsupervised Online OD module can now decide whether it changes its feature subset for each t by using the one proposed by UFSSOD in $t-1$ switching between

$\mathcal{F}_{S1}$ and $\mathcal{F}_{S2}$ or switching subsets again in a windowed fashion. The latter seems more convenient if the setup must not truly work online since the OD algorithm can work with the full throughput using the static feature subset until its model gets replaced or updated with the new subset, which was generated within the current time window. Furthermore, it should be noted that ufssod_clustering with respect to Algorithm 2 must not necessarily be performed for each sample in order to preserve resources while operating in the windowed mode. Thus, continuously ufssod_scoring is performed but ufssod_clustering is only performed if the subsequent classifier demands a new feature subset. With respect to Loda$_{\text{Two Hist.}}$ and Figure 3.3, the old histograms built and classified with $\mathcal{F}_{S2}$ will be replaced with the new histograms switching to $\mathcal{F}_{S1}$ when the window size is reached, whereas UFSSOD continuously updates $\mathcal{F}_{S1}$ until switching.

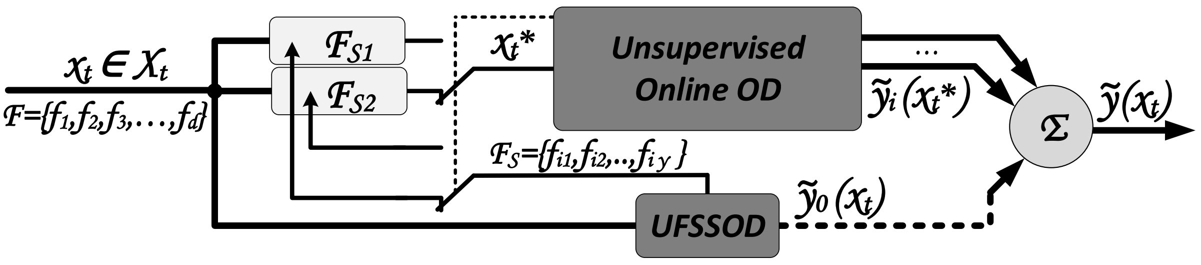


Figure 3.3: Conceptualization of module interaction for unsupervised FS for OD on SD (parallel approach).

3.2.3 Model for Scoring and Clustering Features

In this section, details of the internal working of UFSSOD (Figure 3.1) with regard to the core modules, ufssod_scoring() and ufssod_clustering(), is provided. Motivated by the functionality of Loda to rank features according to their contribution, UFSSOD leverages an adapted version of Loda$_{\text{Cont.}}$ that continuously processes the feature scores for ranking and proposing the top-γ-features. Besides, it is able to provide outlier scores for each incoming $\boldsymbol{x}_t$. It fulfills all the requirements of FS for SD and OD stated in Section 3.1.1. Also encouraged by IBFS, which exploits the training process of iForest for FS, we aimed to bring the nature of an online OD algorithm into this field as well but for the SD context. However, Loda seems more appropriate since it handles high-dimensional data more efficiently and is able to handle concept drifts. We see ourselves validated in our approach, since the evaluations of [120] prove that projection-based methods, as is the case with Loda, are advantageous in high dimensions with many irrelevant features, because the features relevant for OD are less likely to be selected by other methods that operate with $\mathcal{F}$. Therefore, sorting out those irrelevant features from high-dimensional data will significantly aid other OD methods by increasing their performance. Furthermore, the author of Loda already showed the capability of the algorithm to identify meaningful features in his experiments.

Our basis for Loda$_{\text{Cont.}}$ is the *Appendix: online histogram* stated in [119]. However, it must be noted that both of its Algorithms 3 and 4 contain minor mistakes. In the former z_{max} is computed by the *min* instead of the correct *max* function and referring to the latter, the formula for the probability $p(z)$ should depend on a weighted average of the bin counts. With the formula given, one obtains negative results in the case for negative z_i and z_{i+1}. A corrected version has also been proposed by [306] as $p(z) = \frac{(z_{i+1}-z)m_i+(z-z_i)m_{i+1}}{M(z_{i+1}-z_i)}$ such that for instance if z is closer to z_i, $p(z)$ gets weighted closer to $p(z_i)$ accordingly.

The one-tailed two-sample t test has been proposed by the author of Loda, referring to Equation 4 in [119] to measure the statistic significance of each feature to its contribution of a sample's anomalousness. Apart from the complexity of $\text{Loda}_{\text{Cont.}}$ in time with $\mathcal{O}(hd^{-\frac{1}{2}})$ and space $\mathcal{O}(h(d^{-\frac{1}{2}}+b))$, referring to the naïve implementation of continuously updated histograms where h is the number of histograms and b the number of histogram bins, the identification of relevant features does not increase the complexity. This is because the statistical test performed is linear with respect to the number of projections h and number of features d and only increases the complexity in big $\mathcal{O}$ notation by a negligible constant [119]. We integrated the functionality of relevant feature identification of $\text{Loda}_{\text{Cont.}}$ in UFSSOD as a fundamental part and are able to produce feature scores $s_{fi(t)}$ for the i^{th} feature f_i at each time step t for one sample $\boldsymbol{x}_t$, resulting in a one-dimensional array $\boldsymbol{s}_f = \{s_{f1}, s_{f1}, ..., s_{fd}\}$ of d feature scores per sample. Similarly to the continuous updating of the histograms, we propose to continuously update the feature scores for each sample by incremental averaging. There are various approaches discussed in [32], e.g., Exponentially Weighted Moving Average, that are better able to cope with occurring concept drifts in the feature scores while incrementally averaging them. As of now, however, we apply the incremental average $\bar{s}_{fi(t)} = \frac{1}{n_t}(\bar{s}_{fi(t-1)}n_{t-1} + s_{fi(t)})$ with a continuous counter value n_t for each new data instance, in order to obtain the averaged array of feature scores $\bar{\boldsymbol{s}}_f$ but reserve the right to use more advanced moving methods in future work. While only preserving d values for the current average scores, one value for the continuous counter n_t and performing d updates of the scores, both the space and time complexity for each feature score averaging is $\mathcal{O}(d)$ when applying Welford's algorithm. This does not significantly increase the overall complexity of UFSSOD since d is fixed. A summary of the scoring functionality of UFSSOD for both, the outlier score and the feature scores, inclusive of their processing, is shown in Algorithm 2.

Algorithm 2: Scoring functionality of UFSSOD - ufssod_scoring().

```
Input: A sample $\boldsymbol{x}_t$
Output: The scaled outlier score $\tilde{y}_0$ and the averaged feature scores $\bar{\boldsymbol{s}}_f$
1 $y_0, \boldsymbol{s}_f \leftarrow$ Loda_cont.update_score($\boldsymbol{x}_t$)
2 $\mu_t \leftarrow$ outlier_score.moving_mean($y_0$)
3 $\sigma_t \leftarrow$ outlier_score.moving_sd($y_0$)
4 $\tilde{y}_0 \leftarrow$ argmax(0, erf($\frac{y_0-\mu_t}{\sigma_t\sqrt{2}}$))
5 $\bar{\boldsymbol{s}}_f \leftarrow$ feature_scores.moving_mean($\boldsymbol{s}_f$)
6 return $\tilde{y}_0, \bar{\boldsymbol{s}}_f$
```

Since the features within the subset not only influence the efficiency of the OD task but also the cardinality of the set significantly, the number γ referring to the top-features is crucial. However, this is an optimization problem where the intention is to optimize the solution in such a way that the highest accuracy can be reached together with the lowest execution time achieved, e.g., by a smaller number of features. According to [100] it is still a challenging and known problem to determine the optimum number of features to select. Since this mostly applies for the supervised and offline case, we state that finding an optimal solution and therefore a point of equilibrium between the best classification with the lowest computational performance is impossible in the online setting.

When the Unsupervised Online OD module demands a new subset of the top-γ-features, UFSSOD clusters $\bar{\boldsymbol{s}}_f$ to obtain the cluster(s) with the top scoring features. Even if clustering in one-dimensional space is not as trivial as it sounds, some algorithms are capable of solving this task, e.g., by applying Kernel Density Estimation (KDE), having a strong statistical background and seeking for local minima in density to split the data into clusters. However, we propose the k-mean adaption for one-dimensional clustering Ckmeans.1d.dp [307, 308]. It achieves $\mathcal{O}(kd)$ by the optimization proposed in [308] for both time and space complexity, where k refers to the number of clusters built. Three improvements have been made for Ckmeans.1d.dp for the application within UFSSOD. We (i) increased the factor of the Log-Likelihood within the BIC (Bayesian Information Criterion) computation, relevant to finding the optimum number of clusters k in order to better cluster the feature scores by more distinct spacing between data points. To avoid the case of too few feature scores in the top cluster yielding a small γ, we (ii) add a minimum constraint of having at least γ_{min} feature scores in the top cluster. Thus, Ckmeans.1d.dp is performed as long as $|cluster_1| \geq \gamma_{min}$ by successively reducing k starting from the initial optimum number of clusters k_{opt}. This adds a negligible constant to big $\mathcal{O}$ notation in the worst case where $|cluster_1| = 1$ of $\gamma_{min} - 1$. For γ_{min}, we propose a minimum number of $\lceil\sqrt{d}\rceil$ features which is often used as a rule of thumb for selecting features and achieved promising results in our evaluation. However, γ_{min} can also be freely set by the domain expert. Since γ_{min} only states the least requirement and the second, third, etc. best clusters might have promising feature scores as well, we (iii) also propose to check the distances between the cluster centers and their radii which are defined by the utmost feature score from the cluster center as depicted with semicircles in Figure 3.4. Thus, not only the features of the cluster with the top scoring features $cluster_1$ are returned, with $|cluster_1| \geq \gamma_{min}$, but also those where the distance between the centers of $cluster_i$ and $cluster_{i+1}$ is less than the sum of radii of $cluster_i$ and $cluster_{i+1}$. Within the example of Figure 3.4 for $d = 20$, $\lceil\sqrt{d}\rceil = 5$, $|cluster_1| = 6$ the feature scores of $cluster_1$ are returned with (ii) and $\gamma = 6$ whereas those of $cluster_1$ + $cluster_2$ are returned with (iii) and $\gamma = 11$.

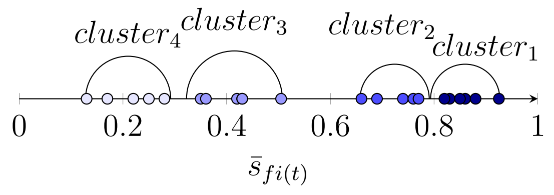


Figure 3.4: Visualization aid of ufssod_clustering with four exemplary clusters and semicircles around the cluster centroids.

It is again noted that the cardinality has a significant impact on the classifier since some even perform well with only a few features [283, 303] whereas others operate best in high dimensions [120] or fluctuate with a varying number of relative features [119]. However, depending on the classifier used, one might adapt γ_{min} or not use the cluster approach at all by setting γ to a certain value. UFSSOD then ranks the feature scores in descending order, e.g., by the widely accepted Quick Sort or Merge Sort algorithm, and returns the top-γ-features. Depending on the chosen setup, in terms of complexity, we propose Merge Sort since the space and time complexity of d feature scores, even in the worst case, is $\mathcal{O}(d)$. A summary of the clustering functionality of UFSSOD is shown in Algorithm 3.

Algorithm 3: Feature clustering of UFSSOD - ufssod_clustering().

```
Input: The averaged feature scores $\bar{\boldsymbol{s}}_f$ and optional=$(\gamma, \gamma_{min}, distance)$
Output: The set of $\mathfrak{indices}$ of $\mathcal{F}_S$ for $\boldsymbol{x}_t$ to obtain $\boldsymbol{x}_t^*$
1  if $\gamma$ is set then
2      return $\mathfrak{indices}$ ← first $\gamma$ of argsort($\bar{\boldsymbol{s}}_f$, descending)
3  if $\gamma_{min}$ not set then
4      $\gamma_{min} \leftarrow \sqrt{|\bar{\boldsymbol{s}}_f|}$
5  result ← ckmeans_1d_dp($\bar{\boldsymbol{s}}_f$)
6  while result.$|cluster_1| < \gamma_{min}$ do
7      k ← result.k$_{opt}$ - 1
8      result ← ckmeans_1d_dp($\bar{\boldsymbol{s}}_f$, k)
9  $\mathfrak{indices}$ ← result.cluster$_1$
10 if $distance == true$ then
11     $cr_1 \leftarrow \varnothing\, cluster_1/2$
12     $cr_2 \leftarrow \varnothing\, cluster_2/2$
13     $i \leftarrow 1$
14     while $cr_i + cr_{i+1} >$ center_distance$(cluster_i, cluster_{i+1})$ do
15         $i \leftarrow i + 1$
16         $cr_{i+1} \leftarrow \varnothing\, cluster_{i+1}/2$
17         $\mathfrak{indices} \leftarrow \mathfrak{indices} +$ result.cluster$_{i+1}$
18 return $\mathfrak{indices}$
```

3.3 Evaluation

This section gives a glance at the experimental setup used for evaluation. First the test environment is presented, followed by information on the data set collection used and a description of the evaluation methodology.

3.3.1 Test Environment

Experiments were conducted on a virtualized Ubuntu 20.04.1 LTS equipped with 12 x Intel(R) Xeon(R) CPU E5-2430 at 2.20 GHz and 32 GB memory running on a Proxmox server environment. All programs are coded in Python 3.7. Unless otherwise stated, commonly used Python libraries, e.g., numpy, are used for instance with the argsort function applying Merge Sort to sort the feature scores. We implemented ufssod_clustering in Python according to Ckmeans.1d.dp [308] with only the relevant functions for BIC and the one-dimensional cluster-routine and adapted it with respect to Section 3.2.3. As of now, however, we did not implement the return of additional clusters based on the cluster center distance verification.

For FS, we integrated the code[1] for FSDS [285] and implemented IBFS according to the original paper [302] in the code[2] of iForest. UFSSOD has been implemented according to Section 3.2.3 in Python as well.

[1]`https://github.com/takuti/stream-feature-selection` (accessed on 05 March 2021)
[2]`https://github.com/mgckind/iso_forest` (accessed on 05 March 2021)

For online OD on SD, we have chosen 6 off-the-shelf ensemble algorithms. Most of them have been shown to outperform numerous standard detectors [121]. The Python framework called PySAD[3] (Python Streaming Anomaly Detection) proposed in [309] is used to provide multiple implementations for online/sequential anomaly detection. It contains among others RS-Hash [118], HS-Trees [117], iForest_ASD [122], Loda [119], Kitsune [115] and xStream [120]. However, some of them are not yet fully implemented, e.g., iForest_ASD, or their offline (batch) implementation is included, taken from PyOD [310], rather than their online counterparts as for Loda and xStream. Thus, we integrated RS-Hash, HS-Trees and Kitsune from PySAD and used our own Python implementation of $\text{Loda}_{\text{Two Hist.}}$. We avoided the use of the Rozenholc formula, stated by [119], to obtain the optimum number of bins per histogram and used default numbers of histograms and bins for all runs. Furthermore, iForest_ASD from scikit-multiflow [34] and the iMForest implementation[4] provided along with [124] was added for the online case. However, iForest_ASD was not included in the measurements due to (i) the lack of properly setting the drift threshold depending on the used data sets, which led to intense time-consuming measurements, and (ii) our desire to rely on the more advanced and recent iForest streaming competitor iMForest. xStream was taken from the StreamAD library[5] providing the streaming version rather than the static one, as with PySAD. Unless otherwise stated the default hyperparameters of the algorithms have been used and outlier thresholds have been fixed for all measurements. In order for the classifiers not to start their classification with empty models, all measurements are first initialized by processing 40% of the same input data later used for classification. This approach seems legitimate to us since we do not focus on the actual performance of each individual online OD classifier but rather want to evaluate the impacts of FS to them.

In terms of evaluation metrics, for each measurement, we computed the typical parameters of the confusion matrix for binary classification, True Negatives (TN), False Negatives (FN), False Positives (FP) and True Positives (TP) to derive the harmonic mean of precision and recall, denoted as $F1$ score. It represents the classification performance of an algorithm and can be computed by $F1 = \frac{TP}{TP+\frac{1}{2}\times(FP+FN)}$. Compared to other work that relies on the ROC (Receiver Operating Characteristic Curve), AUC (Area under the ROC) or only accuracy metric, e.g. [291, 293, 302], we see the $F1$ score is more appropriate for OD since, e.g., the false positive rate used in the ROC metric depends on the number of TN whose proportion in OD is typically quite large. Thus, the ROC curve tends to be near 1 when classifying imbalanced data and thus is not the best measure for examining OD algorithms. A good $F1$ indicates low FP and FN and is therefore a better choice to reliably identify malicious activity in the network security domain without being negatively impacted by false alarms.

Furthermore, we measured the average runtime per OD algorithm, denoted as avg_t, as a representative metric for the computational performance. Thus we accumulated the elapsed time for individual steps necessary to perform, e.g., partial fitting or prediction to derive the average runtime after multiple iterations for processing a particular data set. Having a tradeoff between the classification and computational performance, the third metric is the ratio of $F1/avg\_t$.

[3]`https://pypi.org/project/pysad/` (accessed on 05 March 2021)
[4]`https://github.com/bghojogh/iMondrian` (accessed on 05 March 2021)
[5]`https://pypi.org/project/streamad/` (accessed on 05 March 2021)

3.3.2 Data Source

Rather than relying on a security-domain specific single data set such as KDD'99, NSL-KDD or a predestined younger one CSE-CIC-IDS2018[6], we have deliberately chosen real-world candidate data sets from the Outlier Detection DataSets (ODDS)[7] Library [311] which are commonly used to evaluate OD methods for various reasons. In recent years, the majority of state-of-the-art IDS data sets have been criticized by many researchers since their data is out of date or do not represent today's threat landscape [303, 312, 313]. Even if CSE-CIC-IDS2018 overcomes some shortcomings, it was not optimal for the extensive number of measurements performed (Figure 3.5) due to its enormous number of data instances in multiple files. Furthermore, we wanted to stress test UFSSOD with very dynamic data sets meaning that outliers might occur in various features, which are typically not the same. As could be shown with its predecessor, the CICIDS2017, only a subset of three to four features per attack is enough to describe it best [303], making it look quite static. With an ensemble of the 15 data sets from ODDS shown together with their characteristics in Table 3.3, we focused on a variety of data sets in terms of length, dimension and outlier percentage from multidisciplinary domains. Except for *lympho* and *vertebral*, which have been neglected due to their insignificant number of data instances or dimensions, those data sets are also used by PyOD for benchmarking. Furthermore, to reduce the processing runtime of each OD algorithm we truncated *mnist*, *musk*, *optdigits*, *pendigits*, *satellite*, *satimage-2* and *shuttle* while mostly maintaining its outlier percentage.

Table 3.3: Characteristics of the utilized and partially adapted data sets from ODDS [311].

ID	**Data set**	**⋇ Instances**	**⋇ Dimensions**	**Outliers (%)**
1	arrhythmia	452	274	14.6
2	cardio	1831	21	9.6
3	glass	214	9	4.2
4	ionosphere	351	33	35.9
5	letter	1600	32	6.25
6	mnist	2603	100	26.9
7	musk	1000	166	9.7
8	optdigits	2216	64	6.8
9	pendigits	2000	16	2.3
10	pima	768	8	34.9
11	satellite	3000	36	31.1
12	satimage-2	1750	36	1.0
13	shuttle	3000	9	7.9
14	vowels	1456	12	3.4
15	wbc	378	30	5.6

[6]`https://registry.opendata.aws/cse-cic-ids2018/` (accessed on 05 March 2021)
[7]`http://odds.cs.stonybrook.edu/about-odds/` (accessed on 05 March 2021)

3.3.3 Evaluation Methodology

For the comparison of FSDS and IBFS, as representatives of FS for SD and OD, with UFSSOD, our methodology is shown in a flow chart in Figure 3.5. For each data set of Table 3.3 we performed extensive measurements by first setting an FS algorithm. It must be noted that FSDS relies on the number of singular values k. The authors of FSDS state that this parameter should be equal to the number of clusters in the data set. However, since our focus is on classification rather than on clustering, we performed measurements with different values for k to test the applicability of FSDS for OD. After the FS algorithms yield their scores for each feature, the subset is obtained by different γ, e.g., the top 25% ranked features, randomly chosen γ features, or automatically proposing γ features by using ufssod_clustering to avoid the top-γ-problem. Each obtained subset has been used by 6 online capable OD algorithms per measurement yielding avg_t and $F1$ per classifier. Finally, the results have been averaged across 10 independent runs, since most of the methods are non-deterministic, e.g., negatively affected by Random Projection.

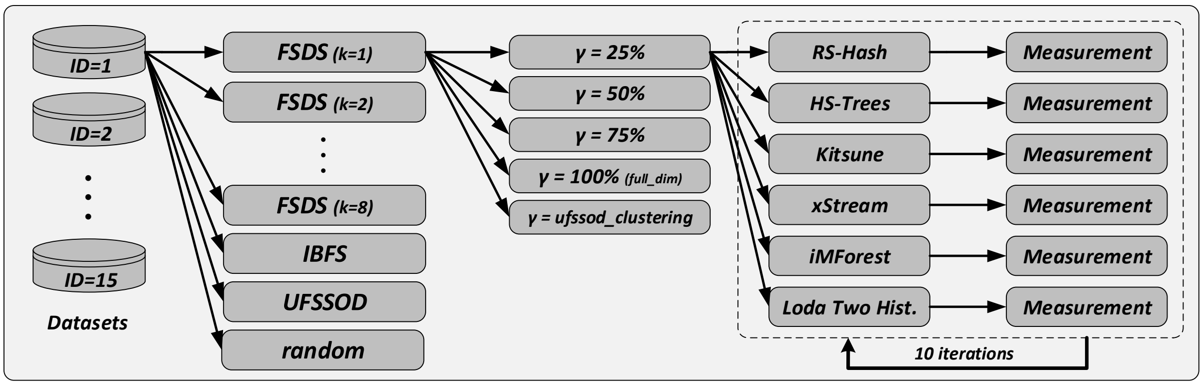


Figure 3.5: Flowchart of the evaluation measurements.

Since no other unsupervised FS algorithms for OD with regard to a streaming fashion exist, we tested our conceptualization in the two proposed settings of Figure 3.2 and 3.3. Thus, we let UFSSOD compute an outlier score and propose the top-γ-features for each data instance that will immediately be used and processed sequentially by xStream. For $\text{Loda}_{\text{Two Hist.}}$ we used the parallel approach and let Loda build histograms using the current feature subset while classifying with the old one. The window size was set to 200 samples rather than 256 used by [119] because of the limited number of data instances in the *glass* data set. After the window size is reached, the current classifying histograms are replaced with the ones built and the new histograms are built using the currently proposed subset by UFSSOD. The streaming setting was only performed on the data sets with IDs 2, 5, 6, 8, 9, 11, 12, 13, 14 due to their more meaningful number of data instances for a streaming setting. Since we stick to the default threshold values for most of the classifiers and do not extract the score values but their binary prediction, we have not utilized the scaled combination approach so far. Despite that, we combined the results of UFSSOD, xStream and $\text{Loda}_{\text{Two Hist.}}$ by a simple logical *or*-conjunction in three settings: UFSSOD with xStream, UFSSOD with $\text{Loda}_{\text{Two Hist.}}$ and UFSSOD with xStream with $\text{Loda}_{\text{Two Hist.}}$ as a combination of UFSSOD with two downstream applied OD algorithms. Although achieving a higher TP while also producing more FP, our approach is legitimate under the assumption that it is more important to detect attacks while coping with FP in a consecutive alert analysis. Albeit not within the scope of this work, it must be noted that most of the classifiers are not able to perform root cause analysis, e.g., Kitsune as stated in [314], which might be relevant for a subsequent alert analysis.

3.4 Discussion of Results

In this section we discuss some of the key results obtained by the comprehensive evaluation. We structured this section into three parts. First we discuss the applicability of FSDS as an FS algorithm for SD when utilized for the purpose of OD by comparing it with IBFS and UFSSOD. Then, IBFS as an FS algorithm for OD is compared with UFSSOD in different feature subset settings to discuss the comparable operational capability of UFSSOD as an FS algorithm for OD. Lastly, we prove the applicability of UFSSOD in conjunction with the online OD algorithms xStream and $\text{Loda}_{\text{Two Hist.}}$ in two different streaming settings.

3.4.1 Comparison of FSDS, IBFS and UFSSOD with the Best 25% Features

First, we compare the results for γ set to the 25% of the top-ranked features for FSDS, IBFS and UFSSOD. The reason behind only considering the top 25% is that if the FS algorithm is able to rank the features according to their contribution of anomalousness properly, the results should, even with this limited amount of features, be noticeable for the task of OD. If one of the FS algorithms yielded poor results even for the top features, we could show its non-applicability to the task of OD. The results of the $F1/avg\_t$ metric are shown in Table 3.4. We have chosen the $F1/avg\_t$ metric in this setup since, independently of IBFS being an offline FS algorithm, we wanted to include the information of the tradeoff between classification and computational performance, especially to compare FSDS with UFSSOD being of online nature. Since we also compared feature subsets of the same cardinality, this approach is legitimate. Nevertheless, the results for $F1$ show a similar behavior. It is noted that FSDS is able to process more than only one sample at each time t. The implementation used required 10 samples at each time to function properly. Thus, the results, with regard to avg_t, would even be worse for FSDS if it would only process one instead of 10 samples due to longer runtime.

Table 3.4: $F1/avg\_t$ results for FSDS (different k), IBFS and UFSSOD for γ set to 25% of d for data sets with ID i (values are scaled with $\times 10^{-3}$ and in unit $1/s$, best performing feature set in bold).

ID	**FSDS_k_1**	**FSDS_k_2**	**FSDS_k_3**	**FSDS_k_4**	**FSDS_k_5**	**FSDS_k_6**	**FSDS_k_7**	**FSDS_k_8**	**IBFS**	***random***	**UFSSOD**
1	10.01	10.35	10.28	9.92	9.97	10.11	9.97	9.99	**11.27**	8.18	9.89
2	2.41	2.03	2.21	1.55	1.62	1.71	1.77	1.71	**2.83**	2.24	2.58
3	7.30	7.92	6.40	9.36	7.21	11.00	8.80	9.39	**12.20**	6.88	3.30
4	17.19	19.52	18.76	18.26	18.17	18.94	18.34	17.61	18.55	18.47	**18.99**
5	1.36	1.42	1.41	**1.40**	1.06	1.16	0.88	1.21	1.39	1.15	1.32
6	2.01	2.11	2.07	2.18	1.99	2.02	2.31	**2.46**	1.77	1.69	1.81
7	6.43	6.95	8.56	**8.86**	7.12	6.20	5.02	5.10	7.75	6.62	6.49
8	1.39	1.36	1.16	1.41	1.30	1.31	1.41	**1.88**	0.68	0.54	0.68
9	**0.93**	0.85	0.68	0.63	0.67	0.49	0.51	0.55	0.63	0.78	0.61
10	7.02	7.06	6.15	7.32	6.11	5.45	6.39	8.81	7.84	7.41	**8.91**
11	1.58	2.20	2.01	2.01	2.40	2.12	2.17	2.05	**3.21**	2.00	3.17
12	0.54	0.39	0.51	0.60	0.59	0.62	0.67	0.56	**1.26**	0.58	0.95
13	1.08	0.70	0.59	1.93	1.65	1.40	1.66	1.24	1.24	0.76	**2.82**
14	0.60	0.66	0.73	0.80	0.86	1.09	1.00	0.64	**1.64**	0.97	1.23
15	13.39	12.40	11.17	10.55	11.05	12.97	12.73	10.51	**25.75**	11.47	24.84

Surprisingly, for some data sets FSDS performs comparably or even better than IBFS or UFSSOD. However, for those data sets even randomly choosing 25% of d features performs mostly comparably too and the better $F1/avg\_t$ is mostly explained by the better average runtime of FSDS compared to IBFS and UFSSOD, especially for data sets with a high dimension. For instance, with *musk* ($d = 100$), IBFS and UFSSOD have an approximately 29% higher runtime compared to all FSDS subset results. For better comprehensibility two exemplary plots of the $F1$ for *letter* and *musk* are shown in Figure 3.6 where IBFS and UFSSOD performed poorly and FSDS achieves better results. For *letter* in Subfig. 3.6a, the overall $F1$ is quite poor also showing that randomly choosing 25% of features achieves results comparable to the other subsets. It is due to the nature of both IBFS and UFSSOD, being based on an OD algorithm, that if the overall $F1$ is poor, to not reliably score the feature contributions with regard to their anomalousness. With an overall decent $F1$ achieved by the subsets in *musk* (Subfig. 3.6b), one can clearly see that IBFS and UFSSOD perform significantly better than the random subset and for most of the FSDS subsets with different k.

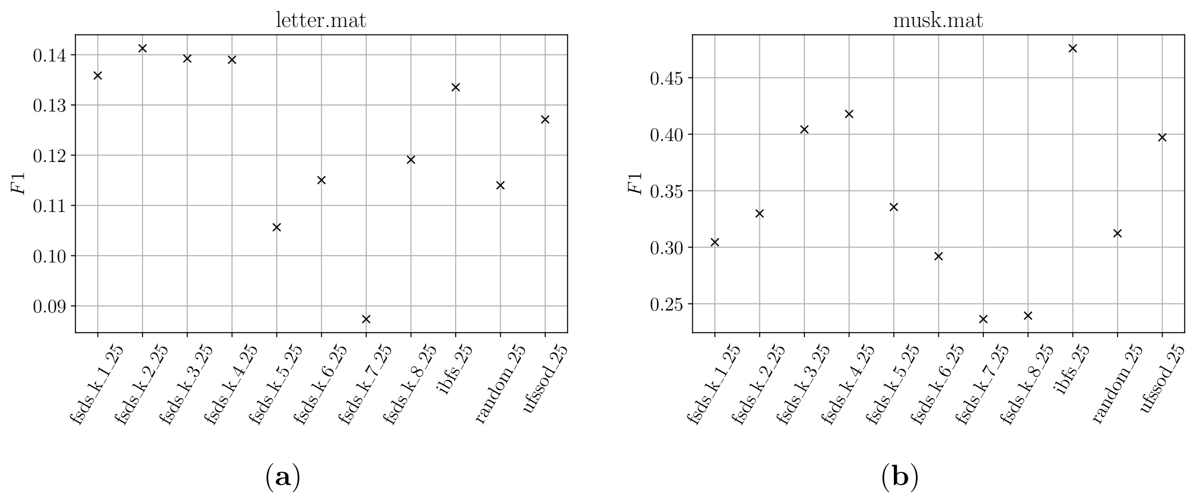


Figure 3.6: Exemplary $F1$ plots for badly-performing IBFS and UFSSOD using top 25% feature subsets.

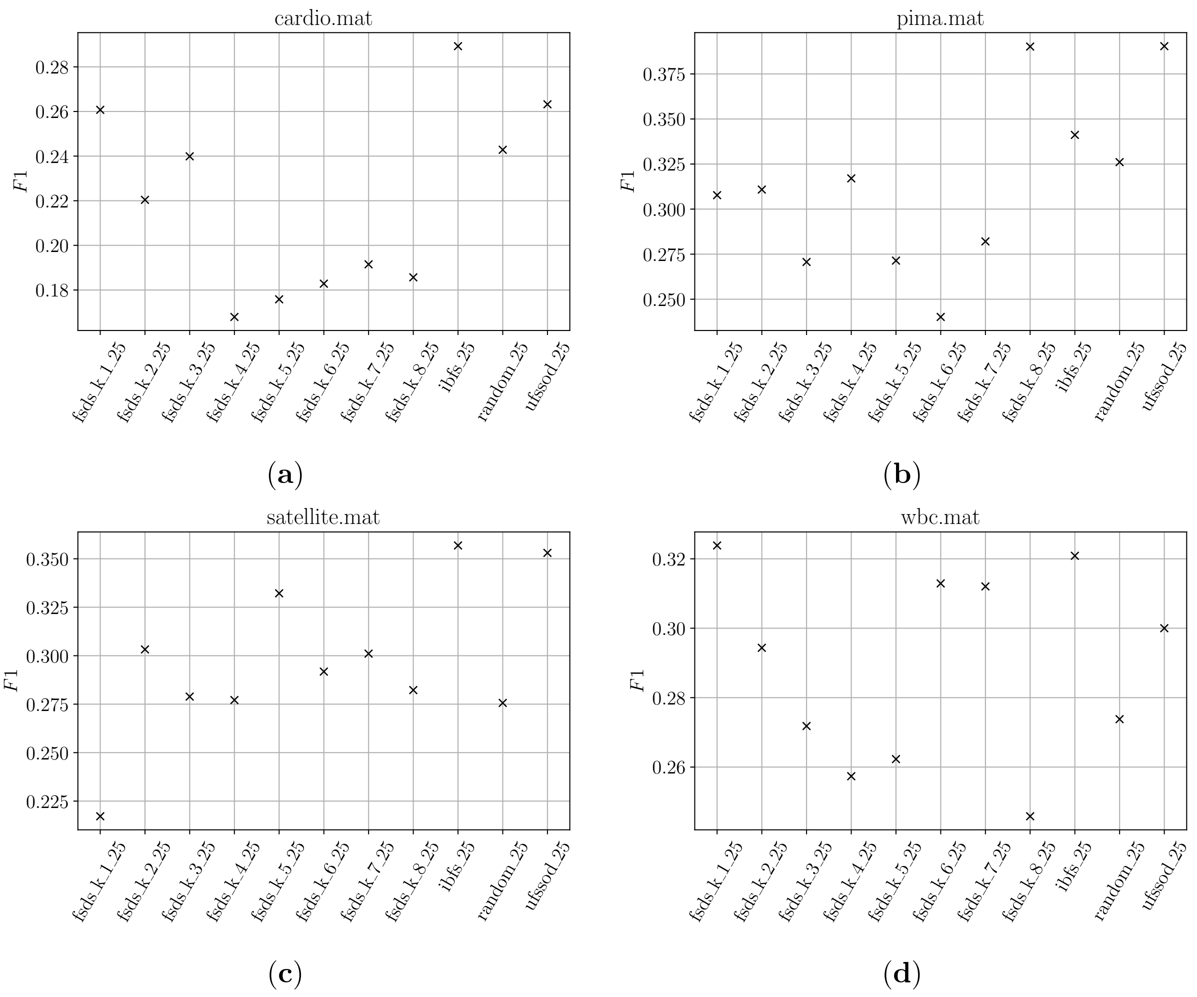
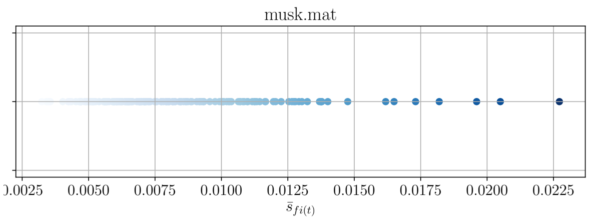


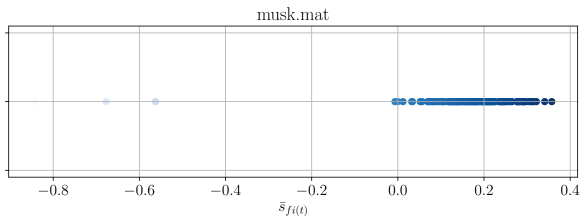
Figure 3.7: Exemplary $F1$ plots for well-performing IBFS and UFSSOD using top 25% feature subsets.

Figure 3.7 shows the $F1$ results of data sets where IBFS and UFSSOD performed well and an overall decent $F1$ could be achieved. In all subfigures it can be seen that both IBFS and UFSSOD achieved better results than randomly selecting 25% of features proving that even for this fixed small amount of features, IBFS and UFSSOD are able to score features reliably according to their contribution of anomalousness. Two key outcomes can be noted. First, in most of the cases FSDS could not achieve better results than randomly selecting features while IBFS and UFSSOD are able to produce better ones. Second, independent of the used parameter k, the results significantly vary for each k per data set without any pattern apparent, e.g., the smaller the dimension the smaller the k. Even if promising results can occasionally be obtained for some k, without any pattern behind, one is not able to properly set the parameter. Those individual cases, e.g., in Subfig. 3.7d with FSDS and $k = 1$, might be explained by FSDS' ability to find redundancy among features in a higher dimension which scored worse and coincidentally might not contain any outliers. Thus, their removal will positively affect the $F1$. However, this way of sorting out irrelevant features is not the actual intention of FS for the task of OD.

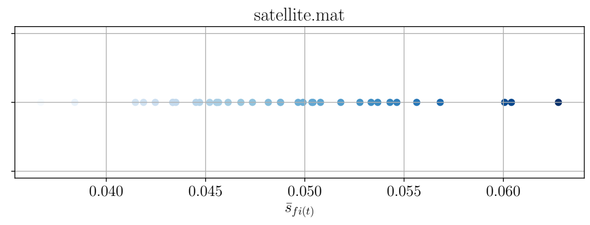
To find an explanation for the worse performance of IBFS and UFSSOD compared to FSDS in terms of $F1/avg\_t$ for some data sets, we examined two exemplary ones: *musk* (Subfig. 3.6b) as a representative for the badly-performing and *satellite* (Subfig. 3.7c) for the well-performing with decent $F1$ scores. We then examined the plotted feature scores of both, which are depicted in Figure 3.8. It can clearly be seen from Subfig. 3.8a and especially from 3.8b that the majority of scores lie densely within a certain range and are homogeneously distributed, which is an indicator that outliers tend to occur in any feature. Referring to Subfig. 3.8c and especially 3.8d, a more heterogeneous distribution of the feature scores for the *satellite* data set can be seen, reasoning that outliers might tend to occur in the same features, which generally contribute more to the outlier score.



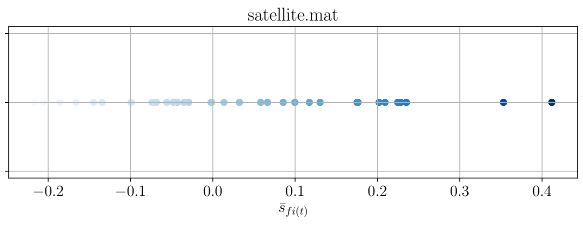
(a)



(b)



(c)



(d)

Figure 3.8: Exemplary plots for feature scores $\bar{s}_{fi(t)}$ of IBFS (a,c) and UFSSOD (b,d) applied on *musk* (a,b) and *satellite* (c,d) (the more intense the color, the higher the score and the more important the feature).

3.4.2 Comparison of IBFS and UFSSOD with Different Feature Sets

Since FSDS did not prove to be a promising candidate for OD, we proceed our result discussion by comparing IBFS and UFSSOD. Because of applying different feature set lengths and an offline with an online algorithm, we now focus on the results of the $F1$ metric for different subsets, as shown in Table 3.5, since we do not want to blur results with shorter average runtimes. For only two data sets, *optdigits* and *satellite*, using all features is more promising whereas, for *optdigits*, the feature subsets perform much worse. For *satellite* the $F1$ scores across all columns are comparable.

Table 3.5: $F1$ results for IBFS and UFSSOD for different γ (feature subsets) for data sets with ID i (full_dim refers to using all features $\gamma = d$, *_25,50,75 refers to setting γ to the top scoring 25, 50 and 75% features and *_ckm refers to the top-γ-features obtained by ufssod_clustering, best performing feature set in bold).

ID	full_dim	ibfs_25	ibfs_50	ibfs_75	ibfs_ckm	ufssod_25	ufssod_50	ufssod_75	ufssod_ckm
1	0.279	0.305	0.290	0.277	**0.308**	0.274	0.282	0.286	0.266
2	0.295	0.297	0.284	0.293	0.296	0.261	**0.298**	0.297	0.264
3	0.092	**0.158**	0.090	0.110	0.090	0.054	0.062	0.138	0.060
4	0.470	0.408	0.458	0.491	0.398	0.409	0.438	**0.472**	0.459
5	0.129	0.137	0.134	**0.141**	0.120	0.124	0.113	0.115	0.123
6	0.271	0.266	0.295	0.319	0.312	0.340	0.303	**0.344**	0.312
7	0.359	**0.482**	0.435	0.420	0.463	0.404	0.420	0.404	0.411
8	**0.158**	0.079	0.096	0.138	0.092	0.086	0.083	0.099	0.065
9	0.101	0.072	0.094	0.090	0.060	0.072	0.097	0.097	**0.102**
10	0.350	0.345	0.245	0.351	0.248	**0.390**	0.234	0.327	0.348
11	**0.364**	0.354	0.297	0.306	0.292	0.352	0.300	0.339	0.347
12	0.061	0.067	**0.075**	0.067	0.064	0.056	0.073	0.064	0.058
13	0.395	0.113	0.303	0.406	**0.447**	0.260	0.341	0.340	0.397
14	0.093	0.075	0.096	0.109	0.095	0.057	**0.098**	0.091	0.097
15	0.283	**0.321**	0.292	0.249	0.305	0.302	0.293	0.273	0.310

A better picture of the bad performance for *optdigits* can be made by examining the feature scores $\bar{s}_{fi(t)}$, as shown in Figure 3.9. The red crosses mark those features that are returned by ufssod_clustering applied on the scores of IBFS and UFSSOD. From a number of 64 features for full dimension, ufssod_clustering applied on IBFS and UFSSOD only marks 9 features to be the most important. Since for *optdigits* it generally applies that the higher the number of features in the subset the better the score, even the measurements with 75% of features seem not sufficient enough. Again, this supports the assumption that for *optdigits* outliers tend to occur in any feature. An overall $F1$ of only 0.162 by using all features shows that the online OD algorithms also performed quite poor on this data set.

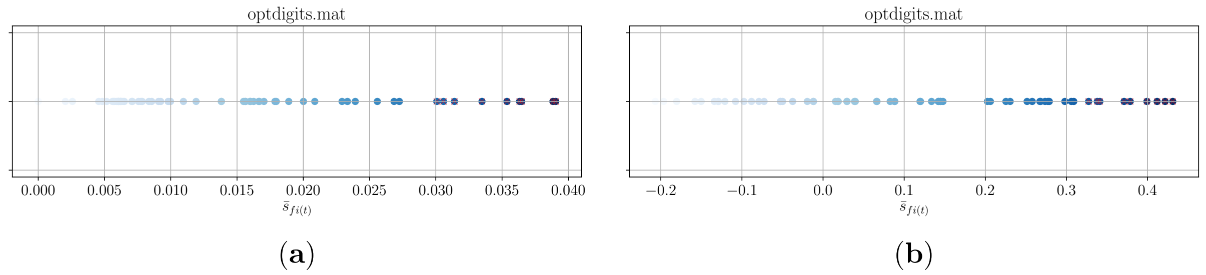


Figure 3.9: Exemplary plots for feature scores $\bar{s}_{fi(t)}$ of IBFS (a) and UFSSOD (b) applied on *optdigits* (the more intense the color, the higher the score and the more important the feature; red crosses mark the top-γ-features by ufssod_clustering).

Even if IBFS and UFSSOD with the ufssod_clustering subset performed approximately 7% faster with regard to avg_t, considering the $F1/avg\_t$ metric for all subsets depicted in Figure 3.10, it did not significantly influence the results at all.

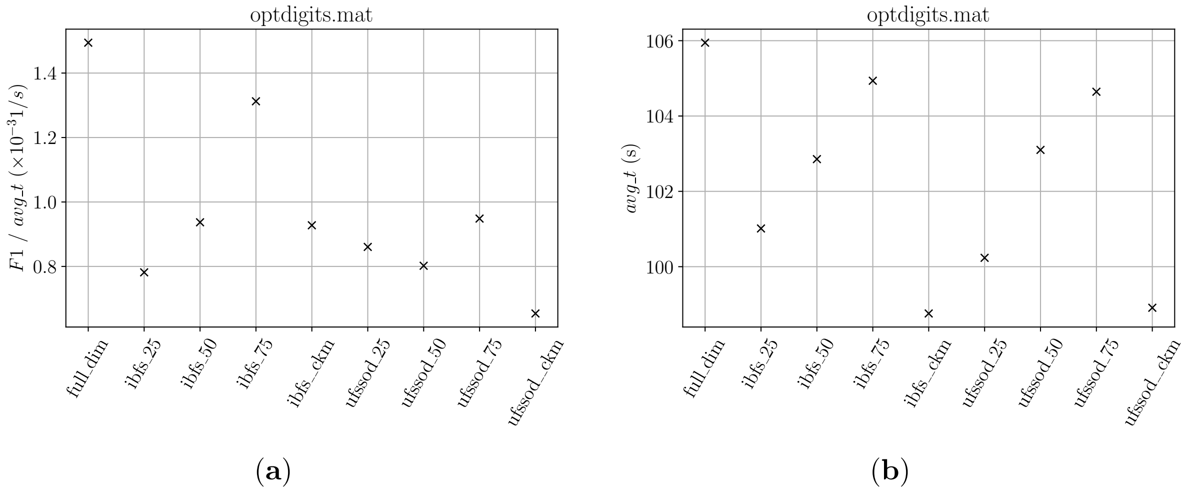
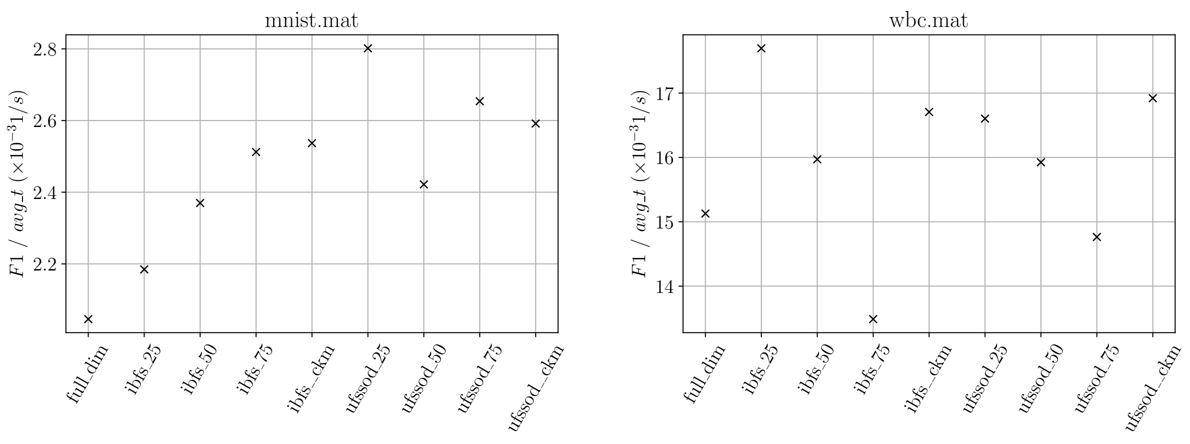


Figure 3.10: $F1/avg\_t$ and avg_t results for *optdigits* referring to Table 3.5.

Independently of whether using IBFS or UFSSOD, in the majority of cases using a subset of features performs better than using the full set. For better comprehensibility, two exemplary plots for data sets that achieved decent $F1$ scores are shown in Figure 3.11. In Subfig. 3.11a, it can be seen that using all 100 features of the *mnist* data set degrades the classification performance, whereas ufssod_clustering with 37 (IBFS) and 21 (UFSSOD) features achieved very good results in terms of $F1/avg\_t$. For UFSSOD, slightly more features as shown with the 25% measurement would have achieved even better results. Subfig. 3.11b shows the results for the *wbc* data set and that it is not always better to choose a high percentage of top scoring features. From the 30 features in total, using ufssod_clustering with 10 (IBFS) and 14 (UFSSOD), features achieved much better results than subsets with a higher number.



(a) (b)

Figure 3.11: $F1/avg\_t$ results for well-performing data sets *mnist* and *wbc* referring to Table 3.5.

In order to show the influence of applying ufssod_clustering for IBFS or UFSSOD for each classifier, we refer to Table 3.6. As a representative example with the *wbc* data set the percentage increase or decrease of the metrics avg_t and $F1$ compared to the measurements using the full feature dimension is shown. For each classifier, applying ufssod_clustering, whether on IBFS or UFSSOD, could decrease the runtime by approximately 12% on average considering the significant improvement on Kitsune. The $F1$ score could be increased by approximately 14% on average. HS-Trees and Kitsune show a significant improvement and a slight improvement could be noticed for $\text{Loda}_{\text{Two Hist.}}$, whereas iMForest and RS-Hash showed a classification degradation. For xStream the application of UFSSOD even yielded an $F1$ improvement compared to IBFS.

Table 3.6: Individual classifier performance in terms of the percentage increase/decrease of avg_t and $F1$ applying ckeans_ufssod on IBFS and UFSSOD compared to full dimension on *wbc* data set.

wbc ($d = 30$)	$\gamma = 10$ **(ibfs_ckm)**		$\gamma = 14$ **(ufssod_ckm)**	
	% avg_t	% $F1$	% avg_t	% $F1$
RS-Hash	-2.33	-1.04	-1.68	-4.24
HS-Trees	-1.14	64.97	-1.34	27.25
Kitsune	-68.89	56.86	-68.56	66.67
xStream	-1.90	-21.15	-1.62	3.23
iMForest	-0.08	-16.66	-0.34	-8.25
$\text{Loda}_{\text{Two Hist.}}$	-2.52	4.00	-1.33	1.20

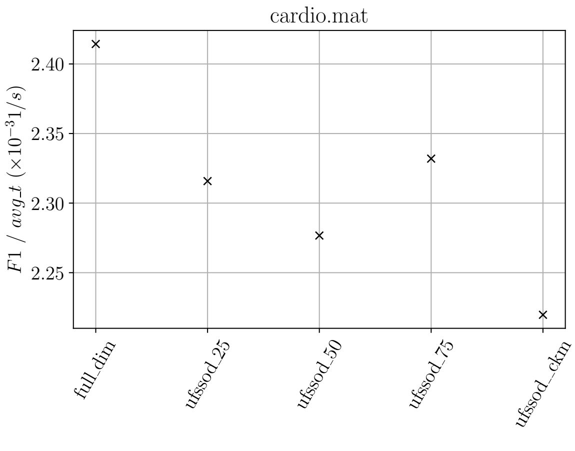
3.4.3 Application of UFSSOD, xStream and Loda Two Hist. in a Streaming Setting

Measurement results for the streaming setting with regard to the sequential and parallel approach (Figure 3.2 and 3.3) are shown in Table 3.7. The $F1$ score results are averaged by the results of UFSSOD, xStream, $\text{Loda}_{\text{Two Hist.}}$ and their logical *or*-combination stated in Subsection 3.3.3.

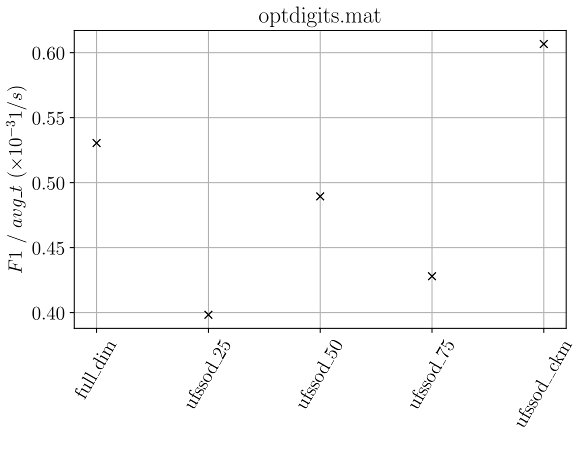
Table 3.7: $F1$ results for UFSSOD using different γ (feature subsets) for data sets with ID i in streaming setting with xStream and $\text{Loda}_{\text{Two Hist.}}$ (full_dim refers to using all features $\gamma = d$, *_25,50,75 refers to setting γ to the top scoring 25, 50 and 75% features and *_ckm refers to the top-γ-features obtained by ufssod_clustering, best performing feature set in bold).

ID	full_dim	ufssod_25	ufssod_50	ufssod_75	ufssod_ckm
2	**0.435**	0.404	0.402	0.416	0.390
5	0.083	0.083	0.088	0.076	**0.093**
6	0.161	0.282	0.229	0.204	**0.328**
8	**0.136**	0.097	0.122	0.108	0.113
9	0.154	0.169	**0.186**	0.159	0.170
11	0.349	0.391	0.389	0.384	**0.396**
12	0.267	0.247	0.247	**0.271**	0.254
13	0.425	0.350	0.455	**0.456**	0.394
14	0.039	0.075	**0.077**	0.075	0.056

For only two data sets, *cardio* and *optdigits*, using all features achieved better classification performance than using a subset. However, instead of yielding worse results, the subsets performed comparably to the full set. Taking into account the average runtime decrease using a subset, as shown in Subfig. 3.12b, for *optdigits*, UFSSOD with ufssod_clustering even achieved a better tradeoff than the other settings. For *cardio* the $F1/avg\_t$ could not provide better results. The reason why ufssod_clustering has a higher influence on the result of the $F1/avg\_t$ metric on *optdigits* than on *cardio* is due to the fact that with only 21 (*cardio*) and 64 (*optdigits*) the number of dimensions can be reduced more significantly in the latter case. Therefore, the average runtime can also be reduced more notably. Since the avg_t of *cardio* with its minor number of features can be reduced only marginally with ufssod_clustering, the dominating factor remains the $F1$ score.



(a)



(b)

Figure 3.12: $F1/avg\_t$ results for badly-performing data sets *cardio* and *optdigits* referring to Table 3.7.

In three data set measurements, *letter*, *mnist* and *satellite*, ufssod_clustering applied in a streaming setting to xStream and Loda$_{\text{Two Hist.}}$ yielded the best results. We neglect *letter* since the overall $F1$ score is poor across all feature sets and therefore the results are non-significant. For *mnist*, ufssod_clustering notably achieved the best $F1$ compared to the other feature sets, where even when using all features, it performed the worst. With *mnist's* 100 dimensions, the good result is improved, shown in Subfig. 3.13a, considering the significant reduction of features resulting in a runtime decrease of approximately 40% and thus a better $F1/avg\_t$ metric. Even for *satellite* having only 36 dimensions, this effect takes place, shown in Subfig. 3.13b, where the avg_t is decreased by approximately 20%.

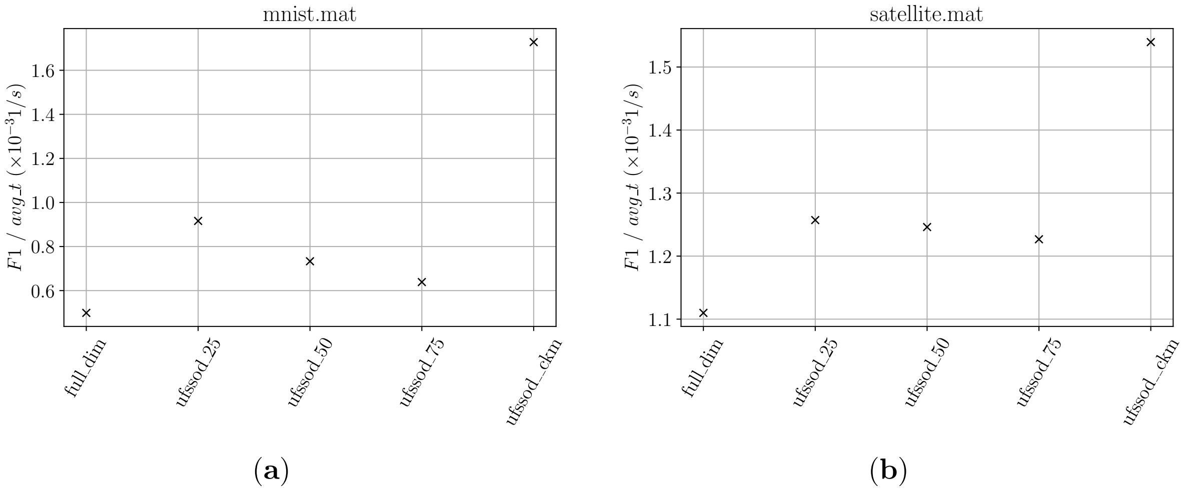


Figure 3.13: $F1/avg\_t$ results for well-performing data sets *mnist* and *satellite* referring to Table 3.7.

In order to show the influence of applying UFSSOD to xStream and Loda$_{\text{Two Hist.}}$, we refer to Table 3.8 showing the percentage increase or decrease of the metrics avg_t and $F1$ compared to the measurements using the full feature set. For all data sets, xStream achieves better results with UFSSOD in terms of avg_t with an overall improvement of approximately 11% compared to the measurements using all features. However, with an average runtime per data set (full dimension) of approximately 590 sec yielding 0.27 sec/sample across all data sets, xStream's Python implementation of StreamAD does not seem very efficient compared to our Loda$_{\text{Two Hist.}}$ implementation, with approximately 3 sec average runtime per data set and 1.4 msec/sample. Compared to the other classifiers in the measurements, xStream was the only one with this significant runtime also explaining why the authors of xStream provided a C++ streaming version rather than a Python pendant to their static one. Considering the fast runtime of Loda$_{\text{Two Hist.}}$, significant improvements of avg_t could only be achieved for data sets whose number of data instances and dimension ratio is higher (ID 6, 8 and 11) than those of the others and, on average, reduces avg_t by 7%. As an example, for *optdigits* with 2216 data instances and 64 dimensions, UFSSOD could reduce avg_t by approximately 22%. As for the percentage improvement of $F1$, excluding the statistical strays of data set 6 and 15 for xStream, an improvement of approximately 22% on average could be obtained. For Loda$_{\text{Two Hist.}}$, the improvement is even more notable with approximately 45% (ID 6 and 15 excluded).

Table 3.8: Performance of xStream and Loda$_{\text{Two Hist.}}$ in terms of the percentage increase/decrease of avg_t and $F1$ applying UFSSOD in a streaming setting compared to full dimension (N/A for ID 15 since no $F1$ score could be obtained with full dimension due to poor classification results).

	xStream		**Loda$_{\text{Two Hist.}}$**	
ID	% avg_t	% $F1$	% avg_t	% $F1$
2	-2.74	-30.72	0.49	-6.41
5	-2.29	60.29	3.18	-14.37
6	-42.9	2638.80	-36.38	232.54
8	-28.4	-47.85	-22.22	305.52
9	-1.34	1.50	1.39	41.81
11	-18.75	34.63	-15.21	3.46
12	-3.19	111.16	1.82	-1.34
13	-1.70	22.61	1.68	-16.07
15	-1.15	N/A	0.29	16.00

It should be noted that xStream worked in the truly online mode. Therefore similar performance boosts, as with Loda$_{\text{Two Hist.}}$, might also be achieved for xStream if using the proposed windowed approach mentioned in Section 3.2. In summary, applying UFSSOD in a streaming setting notably increases both the classification and computational performance if the data set is of high-dimensional nature and has a high number of data instances. In a real-world applications, the former depends on the applied domain but fits perfectly with current observable trends. The latter is only part of this evaluation setup since in the real-world the data stream has an infinite amount of samples. Although xStream and Loda are representatives that work better on high-dimensional data [120] and one might assume that a reduction of features by UFSSOD might degrade their performance, the experiments show the opposite is true, with an overall improved result. Thus, we assume that using UFSSOD with other online classifiers, such as ones based on iForest that handle changing feature sets during runtime, may boost the performance even more.

4 On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data

This chapter provides details on the ***P**erformance **C**ounter-**B**ased* ***iForest*** (**PCB-iForest**) algorithm and is organized as follows — Section 4.1 provides related work with the most popular state-of-the-art solutions for unsupervised OD on SD, especially existing online iForest adaptions. Substantial requirements for online OD algorithms are derived in Section 4.2 and compared with the related work. In Section 4.3, details on the conceptualization and operation principle of PCB-iForest can be found which is able to satisfy all requirements stated in Section 4.2. In Section 4.4, the test environment is presented and details on the extensive evaluation together with the discussion of results (Section 4.5) revealing the superiority of PCB-iForest on the state-of-the-art in most of the measurements.

4.1 Related Work in Online Outlier Detection

Concentrated on supervised or semi-supervised learning, widely accepted online anomaly detection algorithms such as Hoeffding Trees [111] or Online Random Forests [112] achieve good accuracy and robustness in data streams [113]. However, the main focus of this work lies on unsupervised approaches since the amount of unlabeled data with missing ground truth generated across many scientific disciplines, especially intrusion detection has steadily increased. In recent years, many methods have been proposed for unsupervised online OD such as [114, 115, 116] but only a few of them, apart from iForest, namely HS-Trees [117], RS-Hash [118] and Loda [119] have been shown to outperform numerous standard detectors and hence are considered the state-of-the-art [120, 121]. Since xStream proposed in [120] is as competitive as those detectors, particularly effective in high-dimensions, and revolutionized online OD algorithms by being able to deal with streaming features, we count it to the list of the state-of-the-art.

RS-Hash samples axis-parallel subspace grid regions of varying size and dimensionality to score data points by utilizing the concept of randomized hashing. Its adaption RS-Stream is able to operate on SD by computing the log-likelihood density model using time-decayed scores. Thus, compared to other work that applies sliding windows, for example, it uses continuous counting where points are down-weighted by their up-to-dateness. Compared to RS-Hash, the streaming variant requires greater sophistication in the hash-table design and maintenance, although the approach is quite similar [315].

Loda, a lightweight online detector of anomalies, is an ensemble approach consisting of a collection of h one-dimensional histograms, each histogram approximates the probability density of the input data projected onto a single projection vector. Projection vectors diversify individual histograms which is a necessary condition to improve the performance of individual classifiers in high-dimensional data. The features used must only be of approximately the same order of magnitude which is an improvement on other methods such as HS-Trees. Loda's output $f(\boldsymbol{x})$ on a sample $\boldsymbol{x}$ is the averaged logarithm of the probabilities estimated on a single projection vector. It is especially useful in domains where a large amount of samples have to be processed because its design achieves a very good weighting between accuracy and complexity. The algorithm exists in different variants for batch and online learning. For online processing a subdivision can be made. Similar to HS-Trees, two alternating histograms can be used in Loda, denoted as $\text{Loda}_{\text{Two Hist.}}$, where the older set of histograms is used for classification while the newer one is built in the current window. If the new set is built, it replaces the currently used histogram set. A floating window approach, $\text{Loda}_{\text{Cont.}}$, denotes an implementation of continuously updated histograms based on [316].

xStream is able to deal with data streams that not only are characterized by SD in terms of instances (rows) but also evolving, newly-emerging features can be processed while being constant in space and time per incoming data. Having a fixed number of bins for the one-dimensional histograms, a growing feature space cannot be handled by Loda. Thus, the authors of xStream overcome this limitation by so-called half-space chains where the data, independent of streaming features, is projected via sparse Random Projection into recursively constructed partitions with splits into small, flexible bins. This density-based ensemble handles non-stationarity, similar to HS-Trees and $\text{Loda}_{\text{Two Hist.}}$, by a pair of alternating windows.

Random forests are one of the most successful models used in classification and are known to outperform the majority of classifiers in a variety of problem domains [315, 317]. Due to their intuitive similarity, iForests as an unsupervised approach have been established as one of the most important methods in the field of OD. Much work has been done to improve iForest, e.g. [318, 319], or to adapt it to other application scenarios such as feature selection [302]. Even if it was initially not designed to work as an online algorithm, over the last years, manifold variants of online algorithms have been proposed that are either based on iForest's concept or adapt it to operate in a streaming fashion.

HS-Trees, a collection of random half-space-trees, is based on a similar tree ensemble concept as iForest. HS-Trees has a different node splitting criteria and calculates the anomaly scores based on the sample counts and densities of the nodes. Furthermore, the trees have a fixed depth (height) while iForest uses adaptive depths with smaller subspaces. For SD with concept drifts, HS-Trees utilizes two windows (batches) of equal size where simultaneously to the learning of HS-Trees in the current window, the HS-Trees trained in the previous one replace the old.

One of the first approaches adopting iForest for SD is iForestASD proposed in [122]. It utilizes a sliding window with fixed length to sample data on which the ensemble of trees is built. Based on a predefined threshold value, changes within the window can be detected. In the case of an occurring concept drift, this leads to a re-training of the whole ensemble based on the information of the current sliding window content. The authors themselves propose significant improvements in their future work. For instance, the predefined threshold relying on *a priori* knowledge should be replaced and partial re-training of only some trees is suggested rather than discarding the complete model. A description of the differences between HS-Trees and iForestASD can be found in [52].

Recently, iForestASD has been implemented on top of the open source ML framework for data streams scikit-multiflow [34] and improved in [52] to better handle concept drifts by extending it using various drift detection methods. Therefore, the authors extend ADWIN [320] and KSWIN [321] drift detectors and denote them SADWIN/PADWIN and NDKSWIN. Their OD solutions in this article are denoted as $\text{IFA}_{\text{(S/P)ADWIN}}$ and $\text{IFA}_{\text{NDKSWIN}}$. However, some major disadvantages of their proposals, such as partially updating the model rather than discarding the complete forest, are part of future work.

More recently, the work of [123] improves LSHiForest, a classifier based on iForest, to handle high-dimensional data while detecting special anomalies, e.g., axis-parallel ones, to handle SD and produce time-efficient results when processing large high-dimensional data sets. Their improvement, denoted as $\text{LSHiForest}_{\text{Stream}}$ in this article, consists of a combination of streaming pre-processing based on dimensionality reduction with PCA and a weighted Page-Hinckley Test to find suspicious data points. Furthermore, locality sensitive hashing is applied that hashes similar input items into the same branches with high probability and dynamic iForest is applied with efficient updating strategies. Thus, rather than exchanging the whole model as with iForestASD, this approach repeatedly checks if new suspicious data points exist and updates them into the tree structure.

Another hybrid method called iMondrian forest, denoted as iMForest, is proposed in [124]. Mondrian forest is a method based on the Mondrian processes for classification and regression on SD. The authors embed the concept of isolation from iForest by the depth of a node within a tree into the data structure used in Mondrian forest to operate for OD on SD.

The concept of Growing Random Trees or GR-Trees is proposed in [113] which is also capable of partially updating the ensemble of random binary trees. The GR trees approach is quite like the iForest with respect to the training process as is the approach for anomaly score assignment to the data instances. In an initial stage using the first sliding window content, the ensemble is built without explicit training data. Incremental learning is achieved by combining an update trigger deciding when to update the ensemble. Tree online growth and mass weighting ensure that the model can be adapted in time and is able to handle concept drifts.

Referring to Section 4.3, most related to our PCB-iForest approach, in order to only partially update the ensemble-based model rather than completely discarding it (as present in iForestASD, IFA-variants) are the solutions presented within $\text{LSHiForest}_{\text{Stream}}$, iMForest and GR-Trees. Both $\text{LSHiForest}_{\text{Stream}}$ and iMForest do not fulfill the requirement of algorithm agility because of being tailored to dedicated data structures for online learning and classification, rather complex in the case of LSHiForest. Since $\text{LSHiForest}_{\text{Stream}}$ is designed to deal with multi-dimensional multi-stream data, it is seemingly more computationally intensive than multi-dimensional single-stream solutions and, considering the inclusion of k-means clustering for anomaly detection in online mode, the same applies for iMForest. GR-Trees is similar to the iForest training and classification process and also their framework for SD. During online detection of an initially created ensemble, the classified normal instances are stored in a buffer which are used for the streaming update leading to tree growth and updating the trees in the ensemble. The trees are discarded based on the mass weight of the results evaluated by each tree for the sliding window. The tree online growth and mass weighting mechanisms ensure that the detection model can be adjusted in time as the data distribution changes to avoid misjudgments caused by concept drift. Apart from the sliding window, our approach does not need an additional buffer which preserves memory. Additional hyperparameters, like update rate and discard rate, other than the ensemble size and subsample size, could possibly require a bit of adjustment to suit the actual needs to obtain better results. Furthermore, apart from the replacement of discarded trees with trees obtained from a building window, existing trees are updated based on an update window. Both mechanisms source from the buffered normal instances which could possibly pose a slight performance degradation as stated in [51] referring to the section "training using normal instances only".

4.2 Requirement Specification & Validation

In this section, we specify the requirements OD algorithms for SD have to satisfy to be applied in real-world future-oriented scenarios. Additionally, some requirements are added that help a holistic incident handling process, for example, by providing functionality in assisting to identify the root cause of incidents rooted in outliers. We structure the requirements into *operation*-, *data*-, *performance*- and *functionality*-related ones.

As already pointed out in the introduction, in particular, the missing ground truth values in evolving (theoretically infinite) data that requires real or almost near real-time processing, taking the evolution and speed of data into account, that demands unsupervised methods leading to the requirement we will denote as **[R-OD01]**. In addition, those must be capable of dealing with SD **[R-OD02]**. Both requirements are categorized into the *operation*-related ones.

We see additional requirements, as follows, that play a major role in future systems. Another *operation*-related requirement is that the setting of hyperparameters should be of low complexity and especially no information of ground truth should be mandatory for the setting or, even better, no hyperparameters should be set at all **[R-OD03]**. This requirement is important since, nowadays, domain experts are expected to have a high level of multi-disciplinary expertise from data science, e.g., extensive knowledge in statistics, in order to properly set up a machine learning pipeline. Even if recent developments in the field of automated machine learning [87] tries to aid domain experts, they require extensive offline supervised training and testing for each hyperparameter value, thus do not operate in the unsupervised online case. Hence, aiding the domain experts by not burdening them with setting parameters seems important in manageable systems. Furthermore, the time-varying nature of SD, especially in highly dynamic networks, is subject to the phenomenon called concept drift. This means the data stream distribution changes over time. With the different types of concept drifts (sudden, gradual, incremental, recurring, blips) [272], algorithms must be able to efficiently adapt to these changes and continuously re-train or update their model **[R-OD04]** to prevent being outdated, leading to a performance degradation. Dedicated drift detection methods might come to the rescue for OD algorithms to deal with data's varying nature.

From a *data*-related perspective, in the field of network-based OD, data streams from a single source (single-view) and does not necessarily have to be normalized **[R-OD05]**, e.g., a raw network interface, as statistics from network switching elements or in form of log-files from devices. Features can be defined in advance by an expert since incorporating domain knowledge can help select relevant features to improve learning performance greatly. Thus, the cardinality of the feature set (meaning the number of dimensions) is fixed **[R-OD06]** not demanding algorithms that are able to deal with streaming features. For network-based features, one may distinguish between basic features (derived from raw packet headers (meta data) without inspecting the payload, e.g., ports, MAC or IP addresses), content-based features (derived from payload assessment having domain knowledge, e.g., protocol specification), time-based features (temporal features obtained from, e.g., message transmission frequency, sliding window approaches) and connection-based features (obtained from a historical window incorporating the last n packets) [80]. However, with the technological advancement and the increase of potential features, the algorithms must efficiently operate on high-dimensional and high-volumes of data **[R-OD07]** and must cope with missing variables that might occur due to unreliable data sources **[R-OD08]**.

Performance-related requirements can be subdivided into computational and classification performance. The former demands for lightweight algorithms **[R-OD09]**, in terms of time & space complexity, for both model-updating and classification, to be implementable in embedded software. SD is potentially infinite and algorithms must temporarily store as little data as possible and process it as fast as possible due to the time constraint of observing incoming data in a limited amount of time. The classification performance requires to be sufficiently good **[R-OD10]**, i.e. producing decent Area Under the *ROC* curve (*AUC*), in which *ROC* is the Receiver Operating Characteristics, or $F1$ score metrics, meaning to detect malicious activity in a reliable way. It should be noted that stream methods, compared to their batch competitors, typically perform worse in terms of classification. However, under the assumption of applying a subsequent root cause analysis, we strongly support the justification in [34] that when considering critical SD applications, an efficient method, even with less accuracy, is preferred.

Functionality-related requirements can be subdivided as follows. One still unresolved issue for IDS is the lack of finding the actual root cause of incidents. Instead of yielding simple binary values (*normal* or *abnormal*), requirement **[R-OD11]** demands for algorithms that provide outlier score values. Those carry more information and could help a subsequent root cause analysis, for instance, by dealing with false negatives. In addition, the importance of features can play an important role, thus demanding functionality to score or rank features according to their outlier score contribution **[R-OD12]**, providing information about which feature (mainly) caused the outlier. Reducing the data's dimensionality can help deal with the curse of dimensionality referring to the phenomenon that data becomes sparser in high-dimensional space and can still be represented accurately with less dimensions. This adversely affects both the storage requirements and computational cost of the algorithms. Reduction by methods such as PCA maps higher order matrices into ones with lower dimension with a certain probability. However, the physical meaning of the features is no longer retained by this projection and impedes root cause analysis (feature interpretability). Feature selection methods reduce the dimension by only selecting the most relevant features and, hence, preserve their physical meaning. Applying feature selection on SD would possibly lead to changing top-performing-features as time passes and thus demand for OD algorithms that are capable of changing feature sets during runtime **[R-OD13]**. Considering the multitude of recent work that is tailored to attack machine learning, e.g. [322, 323], we see, similarly to cryptographic agility, the flexibility to exchange the actual algorithm as a forward-looking requirement **[R-OD14]** in the case where the currently used algorithm (poisoning) or its model (evasion) get compromised. More likely it is the former, since an evasion of the model seems, due to the continuous updating, more irrelevant.

Over the past years, much attention has been paid to establish OD algorithms for SD in the field of network security, and is increasingly facing trends of massive generated amounts of data with high velocity and afflicted with the phenomenon of concept drift. Many existing works has tried to improve the algorithm setting in terms of *performance*-related requirements by competing on the same (often outdated) benchmark data set. For real-world applications, in which most of the algorithms might perform insufficiently, we expect that designing algorithms and finding a tradeoff between the stated requirements is more crucial. Thus, assuming the application of a subsequent root cause analysis enabled by, e.g., **[R-OD12]**, a certain amount of false positives and false negatives is acceptable referring to **[R-OD10]**. In particular, considering critical application domains, it might be preferred to quickly and efficiently detect outliers even with less accuracy. Table 4.1 validates state-of-the-art algorithms presented in Section 4.1 using the specified requirements. It can clearly be seen that none of the existing methods achieve good results across the majority of requirements.

Table 4.1: Comparison of existing OD work for SD from Section 4.1 with the requirements specified (✓ and ✗ denotes the requirement is either fulfilled or not, ∅ denotes missing information to analyze the respective requirement, (+/++/+++) denotes - as objectively as possible - how well the requirement is fulfilled).

Work	**R-OD01**	**R-OD02**	**R-OD03**	**R-OD04**	**R-OD05**	**R-OD06**	**R-OD07**	**R-OD08**	**R-OD09**	**R-OD10**	**R-OD11**	**R-OD12**	**R-OD13**	**R-OD14**
HS-Trees [117]	(✓)[1]	✓	++	+	✓	✓	+	∅	+	++	✓	∅	✓	✗
RS-Stream [118]	✓	✓	+	++	✓	✓	++	∅	++	++	✓	(∅)[2]	✗	✗
Loda$_{\text{Two Hist.}}$ [119]	✓	✓	+++	++	✓	✓	+++	✓	+++	++	✓	✓	✓	✗
Loda$_{\text{Cont.}}$ [119]	✓	✓	++	++	✓	✓	+++	✓	+++	++	✓	✓	✗	✗
xStream [120]	✓	✓	+	+	✓	(✗)[3]	+++	∅	++	+++	✓	∅	(✓)[4]	✗
iForestASD [122]	✓	✓	+	+	✓	✓	+	∅	+	+	✓	∅	✓	(✓)[5]
IFA$_{\text{(S/P)ADWIN}}$ [52]	✓	✓	++	++	✓	✓	+	∅	++	++	✓	∅	✓	(✓)[5]
IFA$_{\text{NDKSWIN}}$ [52]	✓	✓	++	+++	✓	✓	+	∅	+	++	✓	∅	✓	(✓)[5]
LSHiForest$_{\text{Stream}}$ [123]	✓	✓	+	+	✗	✓	++	∅	+	+	✓	✗	∅	✗
iMForest [124]	✓	✓	+	+	✓	✓	++	∅	+	++	✓	∅	✗	✗
GR-Trees [113]	✓	✓	+	+	✓	✓	+	∅	++	+	✓	∅	✗	(✓)[5]

[1] uses only normal data for training (semi-supervised); [2] "... results from the detector are interpretable and provide a good description of the outliers."; [3] additionally designed for streaming features; [4] feature set can be changed for each arriving data; [5] framework is "wrapped around" the base learner, thus would allow for exchange

4.3 Generic PCB-iForest Framework

In this section we focus on the design of an intelligent OD solution that is able to satisfy all of the aforementioned requirements. Thus, we carefully reviewed related work and combined the merits of the most promising approaches while alleviating their shortcomings. Our focus lies on the iForest-based approaches since iForest is (i) a state-of-the-art algorithm for OD, (ii) widely used by the community, (iii) efficient in terms of computational as well as classification performance and (iv) can easily be adapted for the SD application [52]. The wide acceptance in research is reflected in numerous improvements and extensions, for instance Extended Isolation Forest (EIF) [318], Functional Isolation Forest [319], Entropy iForest [324], LSHiForest [325] or SA-iForest [326] for different application domains or with focus on special problems such as dealing with categorical and missing data [327]. However, those adaptions are mainly tailored for static data sets rather than the application on SD. Thus, our aim is to provide a framework that is able to exchange the iForest-based classifier with respect to either the ability to exchange the model in the case of compromising or if the application domain, with its specific task, demands another base learner. Even more, the framework can be generalized to basically incorporate any ensemble-based algorithm consisting of a set of components like trees. The workflow of our so-called Performance Counter-Based iForest framework, denoted as PCB-iForest, is shown in Figure 4.1.

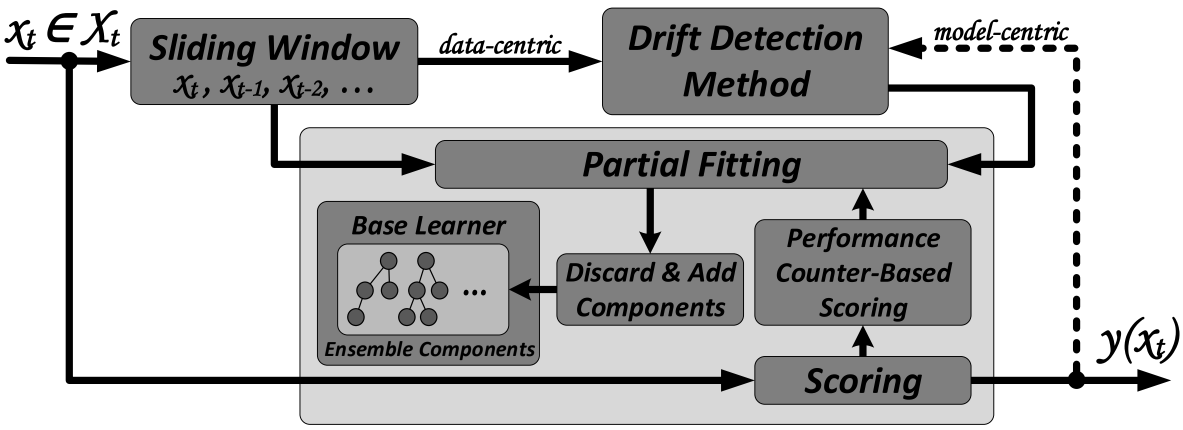


Figure 4.1: The workflow of the PCB-iForest incremental learning framework.

Data instance (data point) $\boldsymbol{x}_t$ with dimension d of the data stream $\{\boldsymbol{X}_t \in \mathbb{R}^{n_t \times d}, t = 1, 2, ...\}$ will be captured as the latest instance at each time step t in the count-based *Sliding Window* $\boldsymbol{W}$ and parallelly will be evaluated in the *Scoring* module which provides an outlier score y for each $\boldsymbol{x}_t$. The sliding window is composed of the latest w instances such that $\boldsymbol{W} = \{\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, ..., \boldsymbol{x}_{t-w}\}$. A dedicated *Drift Detection Method* is applied that triggers the *Partial Fitting* process to *Discard & Add Components*, denoted as C, of the ensemble E. The core of PCB-iForest is the *Performance Counter-Based Scoring* module which is able to identify well and badly performing components of an ensemble. *Partial Fitting* will then discard only the bad-performing and replaces them with newly created ones from the most recent instances contained in the *Sliding Window*. In the following, we provide more details on the main parts of our framework.

4.3.1 Drift Detection Method

Detecting changes in multi-dimensional SD is a challenging problem, especially when considering scaling with the number of dimensions. A sometimes applied solution is to reduce the number of dimensions by either performing PCA (cf. LSHiForest$_{\text{Stream}}$) or Random Projections. Furthermore, one might even reduce the number to one (or more) uni-dimensional statistics and apply well-known drift detection methods such as DDM [328] or ADWIN [329, 320]. For IFA$_{\text{(S/P)ADWIN}}$, drift detection will be performed on the one-dimensional statistic of either the binary prediction value (PADWIN) or the actual score value (SADWIN). Reduction is achieved by the learning model. Thus, it is referred to as a *model-centric* approach. However, generic approaches, such as [330, 331], exist that deal with multi-dimensional changes performed dedicatedly on the SD, referred to as *data-centric*.

We see model-centric approaches (indicated by the dotted line in Figure 4.1) might be prone to a phenomenon called *positive feedback*. This means that drift detection causing partial fitting will be negatively influenced by the actual classification results in such a way that the ensemble results tend to be the same by discarding "badly" performing components from the model point of view. Positive feedback is also present within iForestASD since drift detection depends on the anomaly rate computed by the model's scoring results. Furthermore, iForestASD's anomaly rate is dependent on *a priori* knowledge, which is hardly feasible in real-world applications. Therefore, we recommend the usage of data-centric solutions, which are unbiased of the applied model only relying on the SD characteristics. Since NDKSWIN in [52] has, as of now, proven to be a reliable drift detection method, we are applying it to PCB-iForest but our approach is open to any data- or model-centric solution. NDKSWIN adapts a recent one-dimensional method called KSWIN [321] based on the Kolmogorov-Smirnov (KS) statistical test which does not require any assumption of the underlying data distribution to be capable of detecting concept drifts in multi-dimensional data.

In KSWIN the sliding window $\boldsymbol{W}$ is divided into two parts. The first sub-window, called R, contains the latest data instances where a concept drift might have taken place. The length of R is predefined by the parameter r. The second sub-window, called L, contains uniformly selected data instances that are a sampled representation of the old data. The concept drift is detected by comparing the distance of the two empirical cumulative distributions from R and L according to $dist(R, L) > \sqrt{-r^{-1}ln(\alpha)}$ in which α is the probability for the statistical KS-test. NDKSWIN extends this test by declaring a concept drift if a drift is detected in at least one of the d dimensions. However, contrary to $\text{IFA}_{\text{NDKSWIN}}$, the application of NDKSWIN in PCB-iForest differs. We do not apply drift detection inline before scoring. Our parallel setting, that newly arriving data instances are immediately forwarded to the scoring function, allows us to detect anomalies in near real-time without losing time when performing an upstream applied drift detection. Although a possible concept drift might already afflict the new instance, thus legitimating the approach to first update the model before scoring, we again state that for network-based anomaly detection an accelerated but less precise model is favored. PCB-iForest's design seems obviously more performant, especially, if a high throughput is demanded. Our approach further improves the computational benefit with NDKSWIN since, contrary to iForestASD or $\text{IFA}_{\text{NDKSWIN}}$, we do not discard the whole model in case of detected drifts but are able to only partially update it. Consequently, even if NDKSWIN detects slightly more drifts, our approach is a good tradeoff between a resource-saving model up-to-dateness and a continuously updating model, e.g., HS-Trees or $\text{Loda}_{\text{Cont.}}$ that continuously fit their model with each arriving instance even if there is no need.

4.3.2 Performance Counter-Based Scoring

Performance Counter-Based Scoring monitors the performance of each component C (herein an iTree) in the ensemble E (herein the iForest) by assigning it with a *Performance Counter* (PC). In general, the approach favors or penalizes individual ensemble components over runtime referring to their contribution to the ensembles overall scoring result. Thus, the PC-value is changed for each data instance, i.e., increased or decreased depending on the component's scoring quality to the ensemble's anomaly score. The PC-values increase or decrease by 1 for well and badly performing components depending on whether each individual score is above or under the anomaly threshold s (herein 0.5 for iForest-based learners as discussed in [51]). For example, the ensemble scores a sample with $score\_E > s$, which indicates an anomalous sample. Each individual component's score contribution is verified such that if the score of the i-th component C_i is greater than s, $score\_C\_i > s$, the PC-value of C_i, pc_i increases. Respectively, if $score\_C\_i \leq s$, C_i is penalized by decreasing pc_i. However, one might even increase or decrease the PC-values in an even more granular fashion, e.g., $\pm 2, 3, ...$, depending on the confidence level of the ensemble score and each individual component's score contribution. For instance, if $1 > score\_E > 0.8$, the confidence level of the ensemble is high that the sample is anomalous. Thus, if any $score\_C\_i \ll 0.8$, component C_i might be penalized to a larger extent by decreasing *pc_i* with a higher value. For the sake of simplicity in this article, we apply the more simple binary approach in which each individual PC is increased/decreased by 1 if its score value is greater/less than the ensemble's score value. The counting goes until a drift is detected. Once this happens, the weaker performing components, as indicated by their negative PC values, are replaced with new ones built on data instances present in the current window $\boldsymbol{W}$. The PC values of all trees are set to zero after the partial update is finished, hence, even resetting the values for previously well performing trees clears the old bias (effect of previous scoring). Referring to Figure 4.1, Algorithm 4 shows the operation principle including the core of the Performance Counter-Based Scoring. Also, we neglected the initialization phase in which, once the sliding window is filled, the components are initially built and the PC values are set to zero.

Algorithm 4: Operation Principle of PCB-iForest.

```
   Input: Sample $\boldsymbol{x_t}$, Sliding Window $\boldsymbol{W}$, Anomaly Threshold $s$
   Output: Outlier score $y$
   Data: Ensemble $E$ of Components $C$
   ▷Scoring of Ensemble and each Component
1  for $i$ in $|E|$ do
2      score_C_i ← ComponentScore($i$)
3  $y \leftarrow \frac{1}{|E|} \sum_i$ score_C_i
   ▷Updating PC values
4  for $i$ in $|E|$ do
5      if score_E $> s$ then
6          if score_C_i $> s$ then
7              pc_i = pc_i + 1
8          else
9              pc_i = pc_i - 1
10     else
11         if score_C_i $< s$ then
12             pc_i = pc_i + 1
13         else
14             pc_i = pc_i - 1
   ▷Drift Detection & Partial Update
15 drift_detected ← NDKSWIN($\boldsymbol{W}$)
16 if drift_detected == $true$ then
17     for $i$ in $|E|$ do
18         if pc_i < 0 then
19             delete $C_i$
20             $C_i$ ← build($\boldsymbol{W}$)
21         pc_i ← 0
22 return $y$
```

4.3.3 Base Learner

The PCB-iForest framework is designed to allow exchanges of the base learner. Although being initially intended for iForest-based approaches, the conceptualization can easily be generalized for any ensemble method with its components such as trees, histograms or chains. With the partial updating, compared to iForestASD and the IFA-approaches, a higher throughput is possible since the complete model does not need to be updated. Rather, only a certain number of penalized trees are updated allowing it to not completely and abruptly forget previously learned information by flushing the whole model, similar to catastrophic interference known from the field of ANNs. Thus, with respect to non-iForest-based approaches, we see the potential of our framework to replace, e.g., the alternating windows of HS-Trees or $\text{Loda}_{\text{Two Hist.}}$ in which new ensembles are built and continuously replace those currently used - even if there is no necessity. Our approach would be more resource-preserving while keeping a set of ensemble components as long as there is no need to replace them, e.g., due to a concept drift. However, in this article we focus on iForest-based approaches for the reasons stated in the beginning of this section. In particular, we want to present two application scenarios underlining the fulfillment of crucial requirements from Section 4.2.

Algorithm Agility

SD is afflicted with a theoretically infinite flow of data. Thus, in some cases, it might be necessary to exchange the base learner as time passes. A possible application scenario would be if the currently used base learner has been compromised. This means it is vulnerable to, e.g., poisoning of the algorithm, and an adversary might bypass the detection of its malicious activity. Another non-malicious use case would be a major change of data due to the long term running time that is beyond a concept drift which requires a different type of base learner. Some iForest improvements might then need tailoring for specific application scenarios. In this article, apart from classic iForest, we prove the algorithm agility by incorporating EIF. It addresses drawbacks of iForest's branching using random horizontal and vertical cuts by substituting them with non-axis-parallel hyperplanes with random slopes. Thus, to the best of our knowledge, PCB-iForest is the first work that applies the improved version of iForest on SD. Since the PCB-iForest framework only "wraps around" EIF, no other specific adaptions are necessary except for adding the Performance Counter-Based Scoring. We denote this variant as $\text{PCB-iForest}_{\text{EIF}}$. However, it should be remarked that feature interpretability is irretrievably lost with the improvement of branching in EIF. Therefore, in addition, we are taking on the topic of feature importance measurement for OD on SD by a second variant explained in the next section.

Feature Scoring

Apart from popular dimensionality reduction algorithms such as PCA, feature selection for OD aims to only select relevant features for the classification task by discarding irrelevant and redundant features, which reduce dimensionality. This leads to more computationally efficient OD, all the while preserving feature interpretability. Especially in a consecutively applied root cause analysis, feature interpretability plays a crucial role for future forensics use cases. While some feature selection approaches only provide a subset of relevant features, others are able to score and rank features according to their contribution to a sample's anomalousness. Therefore, one is able to select the best performing features as indicated by their score values. In particular, since iForest is inferior to projection-based methods on high-dimensional noisy data sets [120], feature selection would significantly aid to reduce dimensions and, thus, amplify iForests classification performance in lower dimensions. This coincides very well with the suggestion from Togbe et al. in [52] mentioning that feature selection could mitigate the effect of choosing the most important dimensions for drift detection.

Much work has been done in the field of feature selection but, to the best of our knowledge, existing approaches either focus on feature selection for SD (but not with the focus on imbalanced data classification), e.g. [284, 281, 332], or focus on feature selection for OD, e.g. [24, 302, 301], (but in a supervised and offline fashion). Thus, we see it as crucial to contribute with feature scoring solutions on SD focusing on OD that might be exploited for feature selection. Pevný, in [119], proposed a one-tailed two-sample test for Loda to achieve feature scoring without increasing its overall complexity. This approach seems most related to the intention of scoring relevant features (for the task of OD) in a streaming fashion, which one might use to rank and select the top-performing features.

In order to achieve feature scoring, we take advantage of the unsupervised IBFS method recently proposed in [302] tailored for OD. The method exploits the training phase of the classical iForest, in particular the random selection of feature values, and computes score values for each feature by calculating imbalance scores using an entropy measure. Although this method is designed for offline iForest, it can easily be adapted to our PCB-iForest in a streaming fashion, denoted as PCB-iForest$_{\mathrm{IBFS}}$. Since it is designed for the training phase, we only obtain feature scores after each partial update (training). In order to receive representative feature scores as time passes, we will continuously update the score values with each partial update as shown in Algorithm 5. Once a partial update is triggered, we let IBFS compute feature scores $s_{fi(k)}$ for the *i-th* feature f_i based on the data instances in $\boldsymbol{W}$ resulting in a one-dimensional array $\boldsymbol{s}_f = \{s_{f1}, s_{f2}, ..., s_{fd}\}$ of d feature scores. With each partial update we continuously update the feature scores by incremental averaging. For the sake of simplicity, we apply the incremental average $\bar{s}_{fi(k)} = \frac{1}{k}(\bar{s}_{fi(k-1)}(k-1) + s_{fi(k)})$ with a continuous counter value k for each partial update, in order to obtain the averaged array of feature scores $\bar{\boldsymbol{s}}_f$. It must be noted that other methods exist, e.g., discussed in [32], that might be superior when concept drifts occur within the feature scores. While only preserving d values for the current average scores, one value for the continuous counter k and performing d updates of the scores, both the space and time complexity for each feature score averaging yields $\mathcal{O}(d)$ when applying the well-known Welford's algorithm [305]. This does not significantly increase the overall complexity of PCB-iForest$_{\mathrm{IBFS}}$ since d is fixed. A summary of the feature scoring functionality is shown in Algorithm 5. As time passes and feature scores are continuously computed, one might rank the feature scores and identify the top-performing ones. Thus, it might be necessary to change the feature set or reduce the number of features from the original set. PCB-iForest$_{\mathrm{IBFS}}$ is able to change the feature set during runtime. Once a partial update is triggered, instead of discarding only poorly performing components, the whole model can be discarded and the new ensemble can be built using the newly proposed feature set.

4.4 Experimental Evaluation

This section gives a glance at the experimental setup used for evaluation. First, the methodology will be explained, followed by information on the data set collection used and a description of the metrics as evaluation criteria.

Algorithm 5: Feature Scoring in PCB-iForest utilizing IBFS.

```
Input: Sliding Window $\boldsymbol{W}$
Output: Averaged feature scores $\bar{\boldsymbol{s}}_f$
1 drift_detected ← NDKSWIN($\boldsymbol{W}$)
2 if drift_detected == true then
3     $\boldsymbol{s}_f$ ← IBFS.compute_scores($\boldsymbol{W}$)
4     $\bar{\boldsymbol{s}}_f$ ← feature_scores.moving_average($\boldsymbol{s}_f$)
5 return $\bar{\boldsymbol{s}}_f$
```

4.4.1 Methodology & Settings

Drift detection plays a crucial role when in it comes to detecting changes that require partial updating. Thus, we (i) perform measurements to check whether NDKSWIN works within our model using dedicated data sets for drift detection. Since the most related work historically used the same set or subset of four specific data sets, we aggregate their results from the original work and (ii) perform measurements on those data sets utilizing both, PCB-iForest$_{\text{EIF}}$ (referring to Section 4.3.3) and PCB-iForest$_{\text{IBFS}}$ (referring to 4.3.3). A major drawback of the aforementioned is the insignificant number of multi-disciplinary data sets with an outdated small number of dimensions in terms of today's real-world applications. Furthermore, most of the related work from Section 4.1 ignore the existence of some competitors since they include rather insignificant algorithms in their evaluation (cf. [124]). Therefore, (iii) a selection of state-of-the-art off-the-shelf ensemble OD algorithms is compared with PCB-iForest$_{\text{EIF}}$ and PCB-iForest$_{\text{IBFS}}$ in a large collection of multi-disciplinary data sets with a variety in the amount of dimensions. Additionally, PCB-iForest$_{\text{IBFS}}$ performs online feature (importance) scoring. The scored features are then used in a second run on a selection of algorithms to evaluate the effects when utilizing a feature subset. Lastly, since the main application domain is network security, we (iv) evaluate our PCB-iForest variants and a selection of performant algorithms on an up-to-date data set for network intrusion detection. It should be remarked that LSHiForest$_{\text{Stream}}$ was not included in any of our evaluation since it is designed for multi-dimensional multi-stream data, thus, a fair comparison is not possible within the focus of this article.

Experiments were conducted on a virtualized Ubuntu 20.04.1 LTS equipped with 12 $\times$ Intel(R) Xeon(R) CPU E5-2430 at 2.20 GHz and 32 GB memory running on a Proxmox server environment. Overall, 9 off-the-shelf ensemble algorithms including 5 iForest-based competitors took part in the experiments. For equal conditions, all algorithms are coded in Python 3.7. Unless otherwise stated, the default hyperparameters of the algorithms were used and outlier thresholds fixed for all measurements. In particular, the latter seems legitimate since one important requirement is to not burden human domain experts with a complex hyperparameter setting especially to simulate the appliance in real-world applications. For PCB-iForest we used the default parameters for NDKSWIN as discussed in Section 4.5.1 and, in terms of iForest, we stick to the default parameter of an ensemble size with 100 trees and an outlier threshold of 0.5. For EIF, the fully extended version was used by setting the extension level to 1. Thus, PCB-iForest is hyperparameter-sparse mainly affected by hyperparameter window size. However, except for Section 4.5.2, the window size was set to 200 for all window-using or window-based algorithms. All results (except in Section 4.5.1 and 4.5.4) were averaged across 10 independent runs, since most of the methods are non-deterministic, e.g., negatively affected by Random Projection. The

UNSW-NB15 measurements in Section 4.5.4 were averaged across 3 independent runs due to the large amount of instances accompanied with an enormous evaluation-runtime for some inefficient classifiers.

4.4.2 Data Sources

To achieve high quality in our evaluation, we utilized four different data set sources. In order to measure the performance of NDKSWIN, we utilized the recently proposed ensemble of synthetic data sets for concept drift detection purposes from the Harvard Dataverse [333]. It contains 10 abrupt and 10 gradual data sets, each consisting of approx. 40,000 instances and the same three locations where abrupt and gradual drifts were injected. In particular, *rt_2563789698568873_abrupto* and *rt_2563789698568873_gradual* were used in our evaluation.

Since some of the iForest-based related work performed measurements on four data sets used in [122], we stick to this approach and performed measurements using PCB-iForest on *HTTP*, *SMTP* (from security-related KDD Cup 99[1]), *ForestCover* and *Shuttle* (from UCI Machine Learning Repository [334]) data sets. We truncated the data sets varying in the number of samples while keeping their outlier percentages as in the original set (*HTTP* - 0.39%, *SMTP* - 0.03%, *ForestCover* - 0.96%, *Shuttle* - 7.15%) in order to reduce processing runtime. In the following, the four data sets are denoted as HSFS.

In recent years, the majority of state-of-the-art IDS data sets including KDD Cup 99 or its improved successor NSL-KDD[2] have been criticized by many researchers since their data is out of date or do not represent the threat landscape of today [303, 313]. Therefore, we have chosen two improvements compared to the aforementioned evaluation data sets. First, we have selected fifteen real-world candidate data sets from the Outlier Detection DataSets (ODDS)[3] Library [311] tailored for the purpose of OD. Those will serve to benchmark PCB-iForest in terms of a variety of different amounts of features, contrasting outlier percentages and the application across multi-disciplinary domains. In particular, we included the ensemble of data sets from ODDS shown in Table 4.2 with their characteristics. To reduce the processing runtime of each OD algorithm, *mnist*, *musk*, *optdigits*, *pendigits*, *satellite*, *satimage-2* and *shuttle* were truncated while mostly maintaining its original outlier percentage.

Table 4.2: Characteristics of the partially truncated data sets from ODDS [311].

	arrhythmia	**cardio**	**glass**	**ionosphere**	**letter**	**mnist**	**musk**	**optdigits**	**pendigits**	**pima**	**satellite**	**satimage-2**	**shuttle**	**vowels**	**wbc**
ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
# Instances	452	1831	214	351	1600	2603	1000	2216	2000	768	3000	1750	3000	1456	378
# Dimensions	274	21	9	33	32	100	166	64	16	8	36	36	9	12	30
Outliers (%)	14.6	9.6	4.2	35.9	6.3	26.9	9.7	6.7	2.3	34.9	31.1	1.0	7.9	3.4	5.6

[1]`http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html` (accessed on 12 May 2021)
[2]`https://www.unb.ca/cic/datasets/nsl.html` (accessed on 12 May 2021)
[3]`http://odds.cs.stonybrook.edu/about-odds/` (accessed on 12 May 2021)

Even though CSE-CIC-IDS2018[4] overcomes most of the aforementioned criticism and is said to be well designed and maintained [313], in terms of a network security related data set, we have selected its competitor UNSW-NB15 [335] for the following reasons. It is also well structured, as well as labeled, and more complex than many other security-related data sets, making it a useful benchmark for evaluation [313]. However, apart from the proneness to the issue of high-class imbalance for CSE-CIC-IDS2018 mentioned in a recent publication [336], the main reason for choosing UNSW-NB15 is that the aggregated data sets contain the IP address (source and destination), as well as the respective port features, which we deem essential for a consecutively applied root cause analysis. Especially, similarity-based alert analysis approaches such [337] cannot be utilized when those features are missing since they mainly operate on this information.

UNSW-NB15 incorporates nine types of attacks, namely, *Fuzzers*, *Analysis*, *Backdoors*, *DoS*, *Exploits*, *Generic*, *Reconnaissance*, *Shellcode* and *Worms* created by the IXIA PerfectStorm tool. In terms of feature generation, Argus and Bro-IDS tools among others were utilized to generate 49 features which can be divided into five feature sets: flow, basic, content, time, and additional. Since the four CSV files available only contain the raw features, further data preparation steps had to be performed including handling inconsistent values, dropping irrelevant features and dealing with categorical attributes. Sanitizing had to be performed for source and destination port features containing some values that are not conform with others. Hexadecimal values were converted to integers and for ICMP-protocol-based features, some values containing the character "-" were set to zero. Empty string values are replaced by 0 and typecasted according to its column-type defined in [335]. We dropped the *timestamp* feature since it has no added value for OD, albeit important for a consecutive root cause analysis. However, in real-world scenarios the timestamp will be added after the detection of outliers since it is not contained in the incoming data.

Except for *srcip*, *sport*, *dstip*, *dsport*, *proto*, *state* and *service* (we do not count the actual binary prediction label and the attack category), the data sets contain only numerical and binary data which can be processed by the applied OD algorithms. Various methods can be applied to handle IP-addresses such as converting them into their binary or integer representation (one-to-one), splitting them into four numbers (one-to-four) or applying the widely-used one-hot encoding for categorical features (one-to-many). However, the latter is not feasible in the real-world online setting since (i) all possible IP-addresses that might occur must be known in advance and (ii) one-hot encoding leads to a significantly higher number of features. We chose to convert the IP-addresses into integers since it is an acceptable option for network intrusion detection [338] and does not increase the number of dimensions. The port feature can easily be converted into an integer and for *proto*, *state* and *service* one-hot encoding was used in a way that the number of features does not grow too large and could be utilized on SD. Since, e.g., *proto*, only TCP and UDP have the largest proportion, categories have been limited to only the most important ones in terms of frequency by merging the least occurring values into one category (beneath 1% occurrence). A domain expert could also predefine those categories based on domain knowledge. By this method, the generated feature space of the whole data set could be reduced from 207 to finally 57 features. The four sanitized CSV files with its characteristics are summarized in Table 4.3.

[4]`https://registry.opendata.aws/cse-cic-ids2018/` (accessed on 12 May 2021)

Table 4.3: Characteristics of the four preprocessed CSV files from the UNSW-NB15 data set [335].

UNSW-NB15	**#1**	**#2**	**#3**	**#4**
# Instances	700k	700k	700k	440k
# Dimensions	57	57	57	57
Outliers (%)	3.17	7.53	22.49	20.20

4.4.3 Evaluation Criteria

The confusion matrix, consisting of the parameters True Negatives (TN), False Negatives (FN), False Positives (FP) and True Positives (TP), is the most intuitive and widely-used performance measure for binary classification of machine learning algorithms. Further parameters can be derived such as Accuracy, Precision, Recall, or Specificity. Most widely used as the standard metric for score-wise evaluation of outlier detectors is AUC of the Receiver Operating Characteristics (ROC) curve. ROC is created by plotting the True Positive Rate (TPR) meaning the Recall against the False Positive Rate (FPR) which corresponds to 1 - Specificity. TPR is computed by $\frac{TP}{TP+FN}$ and FPR by $\frac{FP}{FP+TN}$. The AUC metric is used for the sake of comparing related work results in Table 4.4. It is computed as $AUC = \frac{1}{2}(1 + TPR - FPR)$ as proposed in [339].

The harmonic mean of Precision and Recall, denoted as $F1$ score, is used for representation of the classification performance for all other measurements. It can be computed as $F1 = \frac{TP}{TP+\frac{1}{2}\times(FP+FN)}$. Compared to ROC, we deem the $F1$ score more appropriate for OD since, e.g., the FPR used in the ROC metric depends on the number of TN whose proportion in OD is typically quite large. Thus, the ROC tends to be near 1 when classifying imbalanced data and, thus, is not the best measure for examining OD algorithms. A good $F1$ indicates low FP and FN and is therefore a better choice to reliably identify malicious activity in the network security domain without being negatively impacted by false alarms.

Furthermore, we measured the average runtime per OD algorithm, denoted as avg_t, as a representative metric for the computational performance. Thus, we accumulate the elapsed time for individual steps necessary to perform, e.g., partial fitting or prediction, to derive the average runtime after multiple iterations for processing a particular data set. Providing a tradeoff between the classification and computational performance, the last metric is the ratio of $F1/avg\_t$.

4.5 Discussion of Results

In this section we discuss some of the key results obtained by the comprehensive evaluation. It is structured into the following parts. We discuss the capability of PCB-iForest's drift detection by examining NDKSWIN. Then, PCB-iForest$_{\text{EIF}}$ and PCB-iForest$_{\text{IBFS}}$ are extensively evaluated against related work in the following sections. This includes three different types of data sources and includes the evaluation of the feature importance scoring functionality by PCB-iForest$_{\text{IBFS}}$.

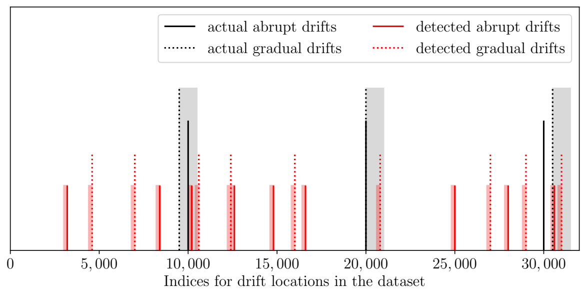
4.5.1 NDKSWIN Drift Detection

NDKSWIN drift detection is mainly affected by the hyperparameters, sliding window size w, the number of samples to be tested n_sample as a percentage value of the window size, sub-window size for the latest samples r, the number of dimensions to be evaluated n_dim and the α-value of the KS-test. For the sake of low hyperparameter complexity, we will rely on the default values $n\_sample = 0.1$, $r = 30$, $n\_dim = 1$ and $\alpha = 0.01$ used in [52]. Since the sliding window size is a crucial parameter, not only for NDKSWIN but also for PCB-iForest, this section examine possible effects of slightly varying hyperparameters of NDKSWIN and discuss potential impacts on PCB-iForest. Since marginal changes in α did not remarkably influence the results, we will vary parameters with the sets $w = [100, 200, 500]$, $n\_sample = [0.1, 0.2]$, $r = [30, 50]$ and $n\_dim = [1, 2]$.

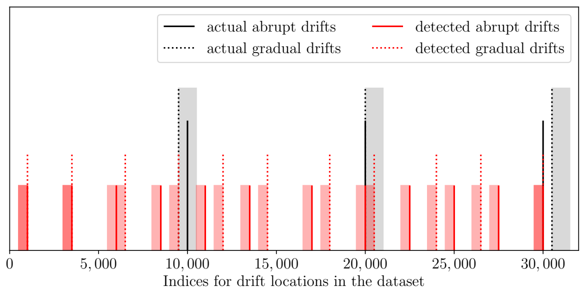
Across all measurements, NDKSWIN was mostly able to detect the gradual and abrupt drifts but was afflicted to a high number of false detections as exemplary shown in Figure 4.2 with two different settings. The duration of a gradual drift lasts for 1,000 samples which is indicated by rectangles in gray. The sliding window size is indicated by rectangles in red which apply to gradual and abrupt drift detection. The detected drift location can only be pinpointed within the range of a window. Thus, detected gradual drifts are counted if either the gray or the red rectangle are intersecting and the abrupt drift must be detected near the red rectangle. From the 24 measurements with the varying settings, NDKSWIN was able to detect 61 out of 72 gradual and 40 out of 72 abrupt drifts.

Generally, increasing NDKSWIN's hyperparameters, multiple tests performed for many dimensions lead to an increasing probability of falsely detected drifts for one dimension, although none is present. This phenomenon might be reasoned due to the widely known problem in statistics called multiple comparison or multiple testing. Independent of the window size, this problem is more distinct for higher values n_sample, r and n_dim as shown in an exemplary way in Subfigures 4.2a and 4.2b. To strengthen our assumption, we have varied n_sample, r and n_dim each at a time while observing the change in false detections. Its number was higher every time we increased each value. In particular, increasing n_sample from 0.1 to 0.2, while keeping the other hyperparameters stable, resulted in a doubling of false detections. Respectively, increasing r (30 to 50) and n_dim (1 to 2) each led to an approximate increase of 40% of false detections. The problem can be alleviated by choosing small hyperparameter values. Thus, the default parameters proposed in [52] $n\_sample = 0.1$, $r = 30$, $n\_dim = 1$ and $\alpha = 0.01$ achieved decent results, mostly mitigate the multiple comparisons problem for its small values, and are therefore used in our further measurements.

Since, the performed measurements are only non-deterministic snapshots without averaging the results over multiple runs, more intensive measurements for drift detection are necessary in future work. We deliberately choose to not average the results since this would blur the results in terms of a better actual drift detection and would not show the effects in real-world applications. Future measurements will include the comparison to other data-centric drift detection methods and take into account higher dimensional data sets. Furthermore, we want to focus on improving the NDKSWIN method by tackling the proneness to the multiple comparisons problem. Although drift detection plays a crucial role in our model, it is not the main focus of our article. We conclude from the measurements that NDKSWIN is able to reliably detect actual drifts and regularly updates our model when triggered by falsely detected drifts. However, it should be remarked that the used data sets lack information regarding the drift characteristics, meaning how distinct drifts are to be detected. For the rest of our evaluation, we set the default parameters (except for the window size) to achieve a good tradeoff between updating our model in regular times and not demanding extensive resources by continuously updating it with every sample.



(**a**) $w = 200, n\_sample = 0.1, r = 30, n\_dim = 1$



(**b**) $w = 500, n\_sample = 0.2, r = 50, n\_dim = 2$

Figure 4.2: Exemplary visualization of NDKSWIN gradual and abrupt drift detection with two different hyperparameter settings (rectangle in gray - duration of gradual drift, rectangle in red - sliding window size).

4.5.2 Competitors-Based HSFS

In this section, PCB-iForest$_{\text{EIF}}$ and PCB-iForest$_{\text{IBFS}}$ are evaluated on the HSFS data sets used by some of the iForest-based competitors: iForestASD, IFA$_{\text{(S/P)ADWIN}}$ and IFA$_{\text{NDKSWIN}}$ as well as GR-Trees. Table 4.4 shows the results where the best-performing values from the competitors' original work has been included together with the measurement results for the two PCB-iForest versions using different window sizes w. Although Togbe et al. in [52] used all four data sets to compare HS-Trees and iForestASD, they only performed measurements on *Shuttle* and *SMTP* utilizing IFA$_{\text{(S/P)ADWIN}}$ and IFA$_{\text{NDKSWIN}}$. What is more, contrary to iForestASD and GR-Trees, the authors chose the $F1$ instead of *AUC* metric. Since we included *Shuttle* in our ODDS measurements in the next section and also rely on the $F1$, only the results from iForestASD and GR-Trees are presented. It must be noted that the authors of GR-Trees constrained the abnormal proportion of each data set to 10% in their evaluation. This step can be seen as critical since the data sets normally have an outlier percentage between 0.03% and 7.15% so that classification results can be blurred by this step. Despite, GR-Trees performed inferior to iForestASD and both PCB-iForest versions as shown in Table 4.4. For this reason, GR-Trees will be excluded from the rest of our measurements. As expected, since being an improvement for the outlier scoring of iForest, PCB-iForest$_{\text{EIF}}$ yielded the best *AUC* results in most of the cases except for *HTTP*. PCB-iForest$_{\text{IBFS}}$ achieved very good results on *HTTP* across all window sizes. Albeit being outperformed by both PCB-iForest variants on three data sets, iForestASD achieved decent *AUC* results. However, it is firstly remarked that the predefined threshold parameter used to trigger concept drifts, and thus updating the iForestASD model, depends on *a priori* knowledge, thereby limiting the application in real-world scenarios. Secondly, the measurements did not take into account computational performance since we assume that iForestASD performs much worse than our PCB-iForest, which is able to partially update its model. We discuss the results of our tradeoff measurements between classification and computational performance in the next section utilizing multi-disciplinary data sets.

Table 4.4: Classification performance of different iForest-based competitors aggregated from their respective original work with PCB-iForest$_{\text{EIF}}$ and PCB-iForest$_{\text{IBFS}}$ (*best setting with $w = 2048$, best performing values in bold).

	iForestASD*	**GR-Trees**	**PCB-iForest$_{\text{EIF}}$**				**PCB-iForest$_{\text{IBFS}}$**			
			$w=128$	$w=256$	$w=512$	$w=1024$	$w=128$	$w=256$	$w=512$	$w=1024$
HTTP	0.95	0.95	0.86	0.89	0.90	0.91	0.92	0.95	**0.96**	**0.96**
SMTP	0.85	0.83	0.84	0.90	0.93	**0.95**	0.70	0.67	0.71	0.70
ForestCover	0.84	0.58	0.62	0.71	**0.93**	0.50	0.75	0.83	0.92	0.50
Shuttle	**0.98**	0.89	0.97	**0.98**	0.94	0.66	0.95	0.96	0.96	0.95

4.5.3 Multi-Disciplinary ODDS

Since in real-world applications hyperparamater-optimization is difficult, especially on SD afflicted with concept drifts, we compared the results of several off-the-shelf state-of-the-art OD algorithms with our proposed PCB-iForest versions using the default parameters proposed in the respective original work and across multi-disciplinary data sets with varying characteristics.

Full Dimension

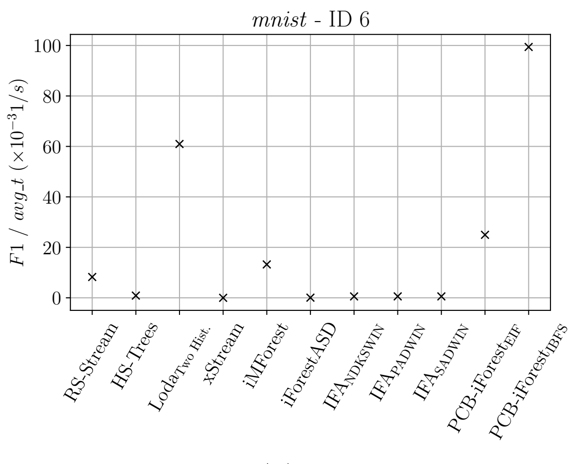
First, we run each classifier on full dimension. Later, the effects of PCB-iForest$_{\text{IBFS}}$'s feature importance scoring using a subset of the best-performing features applied on different algorithms are evaluated. The results of the $F1$ metric are shown in Table 4.5. Approximately in half of the data sets PCB-iForest$_{\text{EIF}}$ outperforms the other online OD algorithms by achieving the best classification result and, hence, achieves the first rank averaged over all data sets. Except, for data sets with ID 7, 10 and 11, PCB-iForest$_{\text{EIF}}$ performs at least comparably with the other data sets and PCB-iForest$_{\text{IBFS}}$ only marginally performs worse to it with an overall rank 3, only slightly outperformed by iMForest. All iForest-based competitors achieved similar averaged outlier scores taking the rankings 6-9.

However, considering the slightly better average runtime, PCB-iForest$_{\text{IBFS}}$ outperforms PCB-iForest$_{\text{EIF}}$ in terms of $F1/avg\_t$ in almost all measurements, as depicted on two exemplary data sets *mnist* and *optdigits* in Figure 4.3, thus presenting itself as an OD algorithm with a good tradeoff between classification performance and computational costs. Except for those two data sets, Loda$_{\text{Two Hist.}}$ achieved the best $F1/avg\_t$ results across all online OD algorithms as shown in Table 4.6, summarizing the $F1/avg\_t$ results. Except for the efficiently performing Loda$_{\text{Two Hist.}}$ on rank 1, our proposed PCB-iForest$_{\text{IBFS}}$ and PCB-iForest$_{\text{EIF}}$ show consistently remarkable results with rank 2 and 3, even superior to the well performing iMForest with respect to its $F1$ results. Table 4.6 also shows the inefficient processing of the iForest-based competitors, in particular the IFA-variants take ranks 7-9 evenly and iForestASD with the last rank.

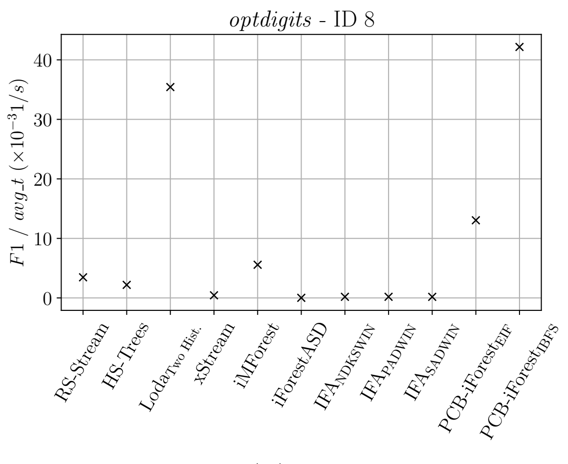
In most of the measurements, iForestASD yielded the longest avg_t value followed by IFA$_{\text{NDKSWIN}}$ as well as IFA$_{\text{(S/P)ADWIN}}$ and xStream. The avg_t results for two exemplary data sets *ionosphere* and *wbc* are shown in Figure 4.4. It should be remarked that for data sets *letter*, *satellite* and *shuttle* the avg_t exceeded 4 hours per data set iteration due to the extensive runtime of those four classifiers. Therefore, we excluded them from the remainder of our evaluation.

Table 4.5: $F1$ results for different online OD algorithms on data sets with ID i (best performing values in bold, '-' measurement aborted after 42 hours runtime, *avg* denotes the average outlier score over all data sets, *rank* from best (1) to worst (11) performance).

ID	***RS-Stream***	***HS-Trees***	***Loda$_{\text{Two Hist.}}$***	***xStream***	***iMForest***	***iForestASD***	***IFA$_{\text{NDKSWIN}}$***	***IFA$_{\text{PADWIN}}$***	***IFA$_{\text{SADWIN}}$***	***PCB-iForest$_{\text{EIF}}$***	***PCB-iForest$_{\text{IBFS}}$***
1	0.248	0.207	0.358	0.178	0.256	0.380	0.413	0.400	0.401	**0.419**	0.268
2	0.143	0.152	0.317	0.353	0.552	0.391	0.365	0.521	0.510	**0.642**	0.526
3	0.029	0.239	0.134	0.014	0.088	0.204	0.205	0.196	0.213	**0.271**	0.220
4	0.422	**0.605**	0.239	0.430	0.601	0.496	0.478	0.488	0.487	0.482	0.479
5	0.001	**0.199**	0.196	0.087	0.145	-	-	-	-	0.141	0.138
6	0.424	0.040	0.229	0.003	0.493	0.409	0.445	0.451	0.446	**0.670**	0.582
7	0.425	0.480	0.253	0.264	**0.570**	0.138	0.222	0.009	0.007	0.007	0.032
8	0.142	0.083	0.104	0.189	0.134	0.149	0.151	0.155	0.142	**0.275**	0.241
9	0.038	0.139	0.143	0.034	0.063	0.095	0.094	0.095	0.091	**0.170**	0.095
10	0.000	**0.584**	0.259	0.397	0.288	0.269	0.265	0.250	0.259	0.274	0.314
11	0.000	0.218	0.385	0.348	**0.543**	-	-	-	-	0.187	0.244
12	**0.105**	0.044	0.084	0.005	0.104	0.078	0.068	0.037	0.037	0.037	0.033
13	0.000	0.430	0.548	0.069	0.487	-	-	-	-	**0.904**	0.828
14	0.109	0.123	0.093	0.001	**0.162**	0.091	0.096	0.092	0.092	0.094	0.091
15	0.101	0.158	0.482	0.130	0.544	0.527	**0.550**	**0.550**	0.523	0.522	0.470
avg	0.146	0.247	0.255	0.167	0.335	0.215	0.223	0.216	0.214	0.340	0.304
rank	**11**	**5**	**4**	**10**	**2**	**8**	**6**	**7**	**9**	**1**	**3**

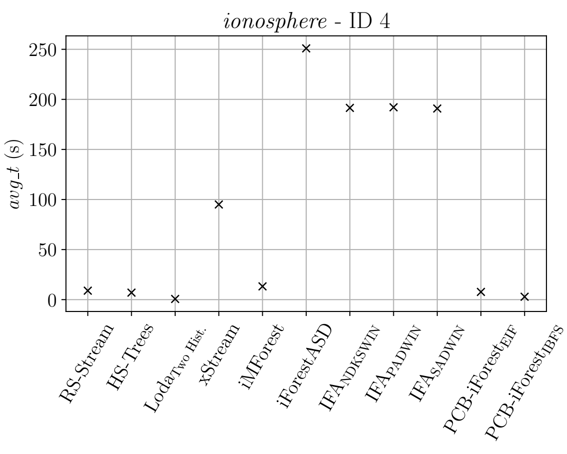


(**a**)

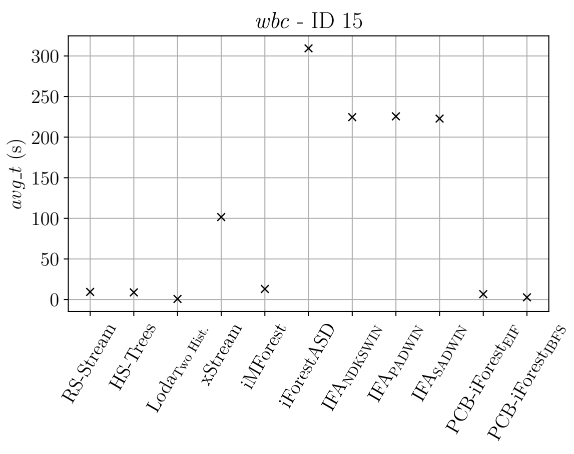


(**b**)

Figure 4.3: $F1/avg\_t$ results for well-performing PCB-iForest$_{\text{IBFS}}$ on data sets *mnist* (a) and *optdigits* (b) referring to the results of Table 4.5 and 4.6.



(a)



(b)

Figure 4.4: *avg_t* results for the data sets *ionosphere* (a) and *wbc* (b) referring to the results of Table 4.5 and 4.6.

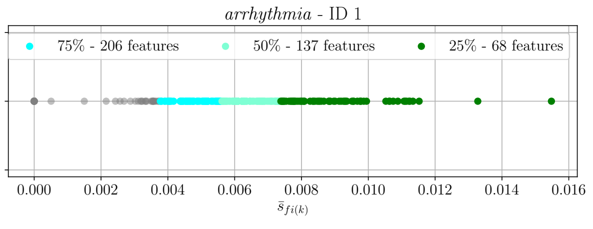
Table 4.6: $F1/avg\_t$ results for different online OD algorithms on data sets with ID i (best performing values in bold, '-' measurement aborted after 42 hours runtime, *avg* denotes the average outlier score over all data sets, *rank* from best (1) to worst (11) performance, values are scaled by a factor of $\times 10^3$ in units of 1/s).

ID	***RS-Stream***	***HS-Trees***	***Loda$_{\text{Two Hist.}}$***	***xStream***	***iMForest***	***iForestASD***	***IFA$_{\text{NDKSWIN}}$***	***IFA$_{\text{PADWIN}}$***	***IFA$_{\text{SADWIN}}$***	***PCB-iForest$_{\text{EIF}}$***	***PCB-iForest$_{\text{IBFS}}$***
1	21.29	18.87	**369.03**	0.25	11.52	1.02	1.76	1.71	1.74	4.85	57.62
2	3.89	4.75	**118.89**	0.98	29.01	0.06	0.47	0.62	0.63	35.17	95.53
3	5.48	38.42	**362.71**	0.23	8.21	3.49	3.53	3.34	3.66	88.98	191.21
4	47.23	86.57	**355.57**	4.52	45.42	1.98	2.49	2.54	2.55	62.55	166.02
5	0.02	6.15	**65.84**	0.23	6.85	-	-	-	-	7.39	8.42
6	8.26	0.88	60.96	0.01	13.23	0.07	0.56	0.57	0.57	24.97	**99.41**
7	17.43	22.54	**127.22**	0.89	21.92	0.09	0.37	0.02	0.01	0.44	8.21
8	3.46	2.19	35.43	0.44	5.57	0.02	0.19	0.19	0.18	13.05	**42.16**
9	1.04	4.3	**55.59**	0.1	3.68	0.01	0.1	0.1	0.1	9.63	16.06
10	0	38.25	**178.64**	2.05	19.87	0.18	0.54	0.51	0.53	27.29	90.4
11	0	3.52	**67.08**	0.48	16.6	-	-	-	-	4.67	6.57
12	2.91	1.4	**30.82**	0.01	4.92	0.02	0.07	0.05	0.05	2.16	6.2
13	0	6.83	**95.85**	0.1	18.43	-	-	-	-	21.58	21.12
14	3.21	4.03	**35.8**	0	8.66	0.05	0.12	0.13	0.13	5.93	19.92
15	10.79	17.94	**671.91**	1.28	41.87	1.7	2.45	2.44	2.35	78.35	173.65
avg	8.33	17.11	175.42	0.77	17.05	0.58	0.84	0.81	0.83	25.80	66.83
rank	**6**	**4**	**1**	**10**	**5**	**11**	**7**	**9**	**8**	**3**	**2**

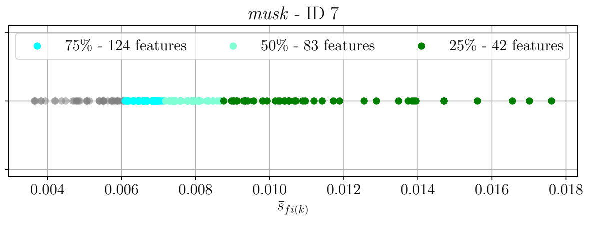
Feature Subsets

Due to the extensive runtime of some classifiers, we continued our evaluation using RS-Hash, HS-Trees, iMForest, Loda$_{\text{Two Hist.}}$ and our two PCB-iForest variants to evaluate the online-capable feature importance scoring functionality of PCB-iForest$_{\text{IBFS}}$. Thus, we used the scored feature values $\bar{\boldsymbol{s}}_f$ obtained by PCB-iForest$_{\text{IBFS}}$, ranked them and supplied subsets of 25%, 50% and 75% to each online OD algorithm. Ideally, if feature scoring operates correctly, the subset of features on the one hand yields a more precise classification result, while on the other hand, the computational effort can be limited by minimizing the cardinality of the selected feature set. Thus, the $F1/avg\_t$ metric should tend to achieve better results using a subset compared to the full dimension measurements with all features in average across all algorithms. This can also be seen from the results in Table 4.7. The full feature set is only top performing for 5 data sets while applied feature subsets yield the best result for 10 data sets. Since, to the best of our knowledge, (as of now) no other online unsupervised feature selection method for OD exists, we could not cross-check our results. In addition no ground truth information is available for ODDS revealing details on which features mainly cause outliers. Thus, with respect to the 5 badly-performing data sets (ID 10-13 and 15), outliers tend to occur in a wide range of dimensions leading to a higher $F1$ degradation using a subset than benefiting from the avg_t reduction, thus, resulting in a worse $F1/avg\_t$. This effect is strengthened by Table 4.7 showing that the higher the number of features in a subset for the badly-performing data sets,it generally results in higher the $F1/avg\_t$. Likewise, without ground truth information, it cannot be shown that for the 67% well-performing data sets, subsets from 25-75% of the features cause the majority of outliers and thus result in good $F1/avg\_t$ values. Nevertheless, the focus of the feature selection results should be set on the high-dimensional data sets, such as *arrhythmia*, *mnist* and *musk* with IDs 1, 6 and 7, since as stated and discussed in [120] most of the classifiers can efficiently handle lower dimensions. For those data sets all feature subsets achieved better results than full dimension.

A better picture of the good performance, shown in an exemplarily way for high-dimensional *arrhythmia* and *musk*, can be obtained by examining the feature scores $\bar{\boldsymbol{s}}_f$, as shown in Figure 4.5. Having a high number of dimensions, feature selection can significantly aid to reduce them, which in turn increases a classifier's classification performance while reducing its computational cost only processing relevant features for the purpose of OD. With regard to the $F1$ score results for *arrhythmia* and *musk*, as shown in Figure 4.6, 50% of the 274 features from *arrhythmia* are already sufficient for achieving better results than utilizing the full feature set. For *musk*, the same applies at the 25% mark of the 166 features.



(a)



(b)

Figure 4.5: Exemplary plots for feature scores $\bar{\boldsymbol{s}}_f$ on high-dimensional *arrhythmia* (a) and *musk* (b) including 25%, 50% and 75% subsets in different colors.

Table 4.7: $F1/avg\_t$ results averaged for each online OD using different feature sets for data sets with ID i (full_dimension refers to using all features, *_25,50,75 refers to setting the top scoring 25, 50 and 75% features obtained by PCB-iForest$_{\text{IBFS}}$, top performing values set in bold, values are scaled by a factor of $\times 10^3$ in units of 1/s).

ID	full_dimension	PCB_IBFS_25	PCB_IBFS_50	PCB_IBFS_75
1	14.49	**21.10**	18.34	18.32
2	15.34	13.04	15.75	**15.97**
3	33.52	**33.86**	30.66	27.81
4	67.95	60.71	67.08	**68.89**
5	5.22	6.41	**6.62**	5.58
6	11.87	13.55	13.12	**14.09**
7	16.24	**19.63**	17.61	17.86
8	5.49	2.54	3.14	**6.22**
9	4.15	3.60	**5.18**	4.86
10	**30.01**	27.91	20.18	28.89
11	**7.69**	6.22	7.23	7.59
12	**2.76**	2.16	2.58	2.71
13	**16.04**	3.15	11.48	15.05
14	5.44	3.24	5.32	**5.51**
15	**52.78**	46.25	45.82	50.06

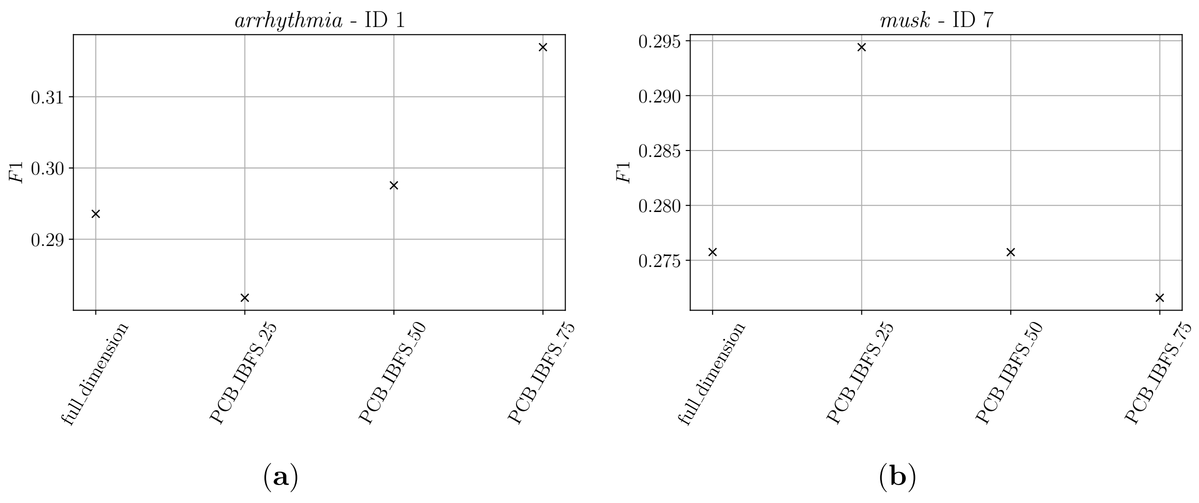


Figure 4.6: $F1$ results for the high-dimensional data sets *arrhythmia* (a) and *musk* (b) utilizing different feature sets (full_dimension refers to using all features, *_25,50,75 refers to setting the top scoring 25, 50 and 75% features obtained by PCB-iForest$_{\text{IBFS}}$).

In order to show the influence of applying a feature set to each classifier, we refer to Table 4.8 showing the percentage increase/decrease of avg_t and $F1$ when applying a feature subset compared to full dimension on the *arrhythmia* and *musk* data set. Except for PCB-iForest$_{\text{IBFS}}$ on both data sets and HS-Trees on *musk*, the avg_t could be reduced for all classifiers. Generally, the average runtime could be reduced by approximately 6% on *arrhythmia* and 9% on *musk*. Contrary to PCB-iForest$_{\text{IBFS}}$, as expected, PCB-iForest$_{\text{EIF}}$ significantly benefits by the dimensionality reduction for both data sets. In terms of $F1$, again we see a performance degradation for PCB-iForest$_{\text{IBFS}}$ but a significant increase for PCB-iForest$_{\text{EIF}}$ on both data sets, especially *musk*. Generally, the $F1$ performance could be increased by approximately 11% on *arrhythmia* and 30% on *musk*. Excluding the high value achieved by PCB-iForest$_{\text{EIF}}$ on *musk*, the percentage increase is still notable with an increase of 5%. The poor performance of PCB-iForest$_{\text{IBFS}}$ utilizing a feature subset only takes place for those two data sets and *wbc*. On all other data sets, PCB-iForest$_{\text{IBFS}}$ could increase the $F1$ while additionally reducing the avg_t with at least one of the subsets. Although being a projection-based method that is designed to operate on high-dimensional data sets, Loda$_{\text{Two Hist.}}$ achieves performance boosts on both high-dimensional data sets, *arrhythmia* and *musk*, for avg_t and $F1$. In summary, it can be said that each individual method reacts differently to feature subsets and although the performance partially degrades for some of them, the overall benefit by a combination of classifiers is clearly evident.

Table 4.8: Individual classifier performance in terms of the percentage increase/decrease of avg_t and $F1$ when applying a feature subset compared to full dimension on the *arrhythmia* and *musk* data set.

	***arrhythmia* - ID 1 (75%)**		***musk* - ID 7 (25%)**	
	% avg_t	% $F1$	% avg_t	% $F1$
RS-Stream	-1.30	-11.52	-0.63	10.91
HS-Trees	-0.23	69.45	0.14	-2.72
iMForest	-8.55	-0.88	-30.00	3.88
Loda$_{\text{Two Hist.}}$	-3.29	10.35	-5.29	27.42
PCB-iForest$_{\text{EIF}}$	-22.65	7.40	-18.87	153.89
PCB-iForest$_{\text{IBFS}}$	1.92	-10.40	2.07	-13.92

4.5.4 Security-Related UNSW-NB15

The results of the time intensive processing - running approximately 38 h - of the four UNSW-NB15 CSV files for only 3 iterations utilizing 6 online OD methods are presented in Table 4.9. Again, Loda$_{\text{Two Hist.}}$ achieves the best results across all measurements for the avg_t resulting in a notable 1.0 ms to process one data instance. This is faster by a factor of approximately 15 compared to the slowest algorithms, iMForest and RS-Stream, achieving approximately 15 ms. For HS-Trees it takes approximately 11.6 ms and for PCB-iForest$_{\text{EIF}}$ 5.6 ms to process one sample. Remarkably, PCB-iForest$_{\text{IBFS}}$ operates most competitively to Loda$_{\text{Two Hist.}}$ with 1.6 ms.

Table 4.9: $F1$ and avg_t results for different online OD algorithms on the four preprocessed CSV files from the UNSW-NB15 data set (best performing values in bold).

UNSW-NB15	**RS-Stream**		**HS-Trees**		**iMForest**		**Loda$_{\text{Two Hist.}}$**		**PCB-iForest$_{\text{EIF}}$**		**PCB-iForest$_{\text{IBFS}}$**	
	avg_t	$F1$	avg_t	$F1$	avg_t	$F1$	avg_t	$F1$	avg_t	$F1$	avg_t	$F1$
#1	10211	0.059	7892	**0.091**	9153	0.062	**645**	0.057	3165	0.084	943	0.080
#2	10522	0.132	8139	0.138	11341	0.140	**660**	0.111	4067	0.239	1102	**0.380**
#3	10489	0.353	8174	0.111	11249	0.367	**675**	0.202	4103	**0.594**	1209	0.342
#4	6767	0.326	5244	0.169	6292	0.336	**438**	0.180	2666	0.310	809	**0.597**

Albeit processing samples very efficiently, in particular with respect to the $F1/avg\_t$ metric, $\text{Loda}_{\text{Two Hist.}}$ could only outperform our proposed $\text{PCB-iForest}_{\text{IBFS}}$ for CSV files #1 and #3. In terms of $F1$, our proposed PCB-iForest variants outperform the other classifiers on all four CSV files except for #1 where all algorithms performed poorly. The reason behind this poor performance on #1 might be the slightly different distribution of attack categories as well as the high imbalance of normal and abnormal data. The *Generic* attack category has the highest proportion of all other classes, on all files. However, on file #2-#4 the proportion of the *Generic* class is higher with approximately 4% on #2, 17% on #3 and 14% on #3 compared to only approximately 1% on #1. Furthermore, the higher proportion of normal data with approximately 97% on #1 compared to 92% on #2, 78% on #3, 80% on #4 leads, in general, to poorer $F1$ values due to the extreme imbalance of positive and negative classes [340].

5 Exploiting the Outcome of Outlier Detection for Novel Attack Pattern Recognition on Streaming Data

This chapter provides details on the ***S**treaming **O**utlier **A**nalysis and **A**ttack **P**attern **R**ecognition* (**SOAAPR**) framework and is organized as follows — Section 5.1 provides relevant background for the reader regarding unsupervised OD on SD as well as aspects of AC and provides related work with the most popular state-of-the-art solutions for AC with respect to (i) outlier detection and (ii) streaming alerts. In Section 5.2, details on the conceptualization and operation principle of SOAAPR, for streaming outlier analysis to identify attack pattern, can be found. It contains a detailed description of the major modules for streaming AC and generation as well as comparison functionality for all three types of signatures. In Section 5.3, the evaluation methodology is described along with details on the data sources and the evaluation criteria. The discussion of results (Section 5.4) is split into two major parts. Firstly, the streaming AC from SOAAPR is compared to the competitor GAC and, secondly, the results, evaluating the signature generation and comparison, are discussed.

5.1 Related Work for Streaming Alert Correlation and Outlier Detection

5.1.1 Alert Correlation for Outlier Detection

Bolzoni et al. in [341] formulated the problem, when an anomaly-based IDS raises an alert, it cannot associate the alert with an intrusion type / class, mostly mandatory for AC. Anomaly-based IDS can only provide little information, such as the IP-addresses and port information, and in addition, a security analyst might add the intrusion type or class label, but only in a laborious manual analysis process. Thus, the authors proposed Panacea [341] in order to automatically classify attacks utilizing a supervised SVM learning model. It inspects the payload of data instances and searches for unusual novel patterns, e.g., byte sequences of certain intrusion types by leveraging previously learned information. The intrusion type can be assigned by finding the most similar alert payload. However, Panacea is payload-centered which hampers the application for certain attack categories such as Portscan or DDoS not involving malicious payload content. Further, it requires training data which, in particular for attack-payload, is typically not available and faces problems when dealing with payload-encrypted traffic.

With a different focus on filtering false alarms, the work in [342] addresses the issue that reported alarms from anomaly-based IDS also lack rich information. They may only identify the anomalous connection stream but cannot provide intrusion type or class information. Their proposed framework is composed of the feature constructor, the cluster constructor and the so-called simple best fit cluster in order to monitor the generated alerts from an anomaly-based IDS. The feature constructor extracts network traffic flow information and derives certain metrics used to construct clusters of normal alarm patterns in a training phase utilizing the cluster constructor. Incoming alerts from an anomaly-based IDS are then evaluated in the simple best fit cluster module incorporating information from their respective network traffic flow features if deviations from the trained model occurred. However, the main intention of this work is not the identification of novel attack pattern but rather the reduction of anomaly-based IDS outputs.

An approach that detects multi-stage attacks in an unsupervised way without details on single-stage attacks is proposed in [343]. Since the authors state that conventional multi-stage attack detection is designed for misuse-based IDS, which are leveraged for single-stage attack detection, their proposal is designed to operate on both, signature and anomaly-based IDS alerts. The main idea of the approach is that suspicious flows are generated, clustered and labeled in the rule generation phase. Labeling in this phase means that the intrusion type or class labels are assigned to each cluster. According to [344], the assignment of clusters to attack stages still remains to be investigated.

A recent work [344] proposes Adept, a distributed framework to detect individual attack stages in order to uncover a coordinated attack in the IoT security domain. Anomaly detection is performed on network traffic of IoT devices and potential anomalies are sent to a security manager. It will aggregate and mine alerts using a method called frequent itemset mining. The resulting alert- and pattern-level information will be supplied to a ML approach to identify the individual attack stages. However, the attribution of incoming alerts to different attack stages is performed using a supervised approach, in particular, leveraging k-Nearest-Neighbor, RF and SVM. Furthermore, their extraction and identification of attack patterns is founded on a simple anomaly detection method designed for the needs in the IoT domain. Thus, common state-of-the-art off-the-shelf anomaly detection methods are not compatible with Adept.

A promising generic graph-based AC solution, denoted as GAC, is proposed by Haas et al. in [337]. Since it only relies on the alert attributes IP and port of source and destination it can be exploited to correlate alerts generated by anomaly-based IDS. GAC is composed of three building blocks: alert clustering, context supplementation and attack interconnection. For alert clustering, similarities between each of the alerts are computed by attribute-specific comparison functions. Then, an attribute graph is derived with alerts as nodes and their similarity values as weighted edges. In leveraging community clustering, in particular the Clique Percolation Method, loosely coupled clusters can be extracted from the attribute graph which potentially contain alerts of a single-stage attack scenario. Context supplementation then transforms the resulting clusters into a graph - flow graph - that characterizes the communication patterns between the alerts. From the resulting flow graph four different attack categories can be identified depending on the communication relation between attacker(s) and victim(s). The last block, attack interconnection, aids to identify multi-stage attacks by revealing relations between individual attack scenarios by comparing the set of attackers and victims of each attack cluster.

In a subsequent work [54], the authors proposed a more flexible solution than context supplementation by assigning clusters to one of four attack categories: one-to-one (oto), one-to-many (otm), many-to-one (mto) and many-to-many (mtm). The so-called motif-based approach builds upon the alert clustering stage from GAC or any other method that groups alerts into clusters, potentially reflecting attack scenarios. Then, if clusters are obtained, a communication structure graph is derived by the IP-address and port information extracted from the alerts. The communication relation of who attacks whom and which ports are relevant for an attack are reflected in a directed graph structure. A fingerprint-like characteristic can be extracted from the graph by leveraging a concept called motif signatures. These are different characteristic sub-graphs, e.g., three nodes with 16 possible edge patterns among them, whose occurrence in the graph represents a motif signature. Network motifs were initially proposed by [345] and have been transferred to the security-domain as discussed in [54]. However, the application of network motifs in [54] is transferred for the characterization of attacks and allows a fine-grained, privacy-preserving and dynamic generation of signatures for a multitude of known and unknown attack scenarios, as well as the differentiation and comparison among them.

5.1.2 Streaming Alert Correlation

A real-time correlation of intrusion alerts is proposed by Wang et al. in [346]. It requires alerts that contain the intrusion type and relies on a vulnerability-centric correlation that maps exploit information, and its vulnerability relation, with alerts. Hence, the focus of the solution is on multi-stage attack detection and by its so-called Queue Graph approach it is able to counteract the limitations of sliding windows prone to be tricked by adversaries. Using an extension of the Queue Graph, the attack graph based approach is able to hypothesize missing alerts and predict future ones.

Ma et al. in [33] propose a real-time system that automatically discovers attack strategies from evolving alert streams. They leverage a well-known streaming clustering method, called CluStream [347], that is designed with an online and offline module and replaces the latter with an AC component. The online module is used to generate high-level alerts, hyper-alerts, that maintain statistics from the streaming alerts, summarizing their characteristics, over different time periods. Alerts must feature, among other attributes, the intrusion type denoted as SigID, which is obtained by misuse-based IDS. Signature-like characteristics can then be derived by the assumption that a multi-stage attack of the same type typically happens in a certain time span and the sequence of their hyper-alerts is similar.

A framework for incremental frequent structure mining is proposed in [348] that aggregates alerts into structured communication patterns depending on the connectivity-relation of involved hosts: mto, otm and mtm. The frequency of those patterns is mined from the streaming alerts and is considered finished if it is not changed for a user-definable amount of time. From those patterns, a so-called Frequent Structured Pattern Tree, FSP_Tree, is created that encodes the most significant patterns along with their time-sensitive information in a Pattern Tree.

Ren et al. in [349] propose online AC using two components. In an offline module, a Bayesian correlation approach is utilized to extract causal relations among alert features. Based on those patterns, the relevance of alerts for attack steps can be analyzed, which will be stored in a Correlation and Relevance Table. Those reference tables can be consulted if new alerts stream into the online module of the system to uncover multi-step attacks. Again the intrusion type / class field of an alert plays a crucial role and additional alert features must be derived in order to form hyper-alerts of the same type. However, as mentioned by Sundaramurthy et al. in [55], the approach by Ren et al. can only learn and detect the type of attacks that previously occurred. Therefore, Sundaramurthy et al.

proposed a slightly different approach, which is knowledge-based but works on knowledge of an attacker's intention and constraints rather than attack specifics. Thus, they use a semantic model which maps potential meanings to alerts without incorporating types of attack scenarios.

A real-time method, denoted RTECA, is proposed in [350] that extracts so-called critical episodes, which are sequences of alerts that could be part of multi-step attack scenarios. Thus, their framework aggregates alerts, including their intrinsic attributes with the intrusion type and attack severity information in order to generate hyper-alerts and merge similar alerts together. The timely sorted alerts are categorized in larger parts, batches, each divided into smaller parts, called episode windows. The framework is composed of an online and offline module. In the former, an online attack tree is generated based on the alerts and the steps of multi-step attacks are determined. In the offline phase, alert similarities are computed and an offline attack tree is generated by the alerts of critical episodes in order to learn multi-step attack scenarios.

Utilizing an offline and online module too, Daneshgar et al. in [351] proposed a method that clusters alerts as fuzzy events according to their similarities and historical events, obtained from the offline module, in an online manner. A fuzzy frequent pattern mining module in the offline phase mines for relations based on statistical characteristics between alerts to extract fuzzy patterns. The resulting correlation strength can, in turn, be taken into account for the similarity measure utilized in the fuzzy clustering.

Zhang et al. in [352] proposed a framework named IACF, which stands for Intrusion Action Based Correlation Framework. Its components, split into an extraction and modeling phase, cover alert normalization, action extraction, session rebuilding and building, as well as updating a correlation graph. Actions in this work refer to a set of alerts potentially indicating a single-stage attack. Subsequently, sessions are a sequence of actions that represent the association relation between actions based on temporal metrics. IACF is able to prune sessions in order to remove redundant actions, fuse them and construct a correlation graph, which can be utilized to predict future attack steps. In a consecutive work [24], the authors leverage the Hierarchical Temporal Memory (HTM) algorithm for online intrusion scenario discovery and prediction. Again, normalizing alerts is the first step before hyper-alert processing is performed by online clustering, session reconstruction and session encoding. Alerts are clustered by computing the similarity of the types field in an online manner. Similarly to IACF, sessions are reconstructed and encoded in order to feed the HTM online learning as part of the intrusion scenario discovery module. HTM learns the patterns, predicts potential next steps and scores anomalies. Outcomes of the HTM are used to update a correlation matrix representing the correlation strengths between actions from which potential future attack paths can be extracted.

5.1.3 Delimitation from SOAAPR

Inspired by the above literature, this article transfers and improves methods for aggregation, (streaming) clustering and attack categorization in the field of attack pattern mining by only utilizing the outcome of OD algorithms. To better guide the reader, a delimitation from SOAAPR to related work is given in this section.

Regarding the fusion of alerts of multiple IDS sensors in order to derive high-level alerts with strong confidence about the mitigation of FP and FN, our alert preparation module in SOAAPR works similarly to the aggregation component in RTECA [350]. Both approaches fuse similar alerts based on the concept of attribute similarity. However, RTECA strongly relies on alert type and attack severity information which is not present when operating on anomaly-based IDS. Thus, we take advantage of additional OD functionality incorporating outlier score and feature contribution of a data instance leading to an alert generation. Furthermore, as used by [33], similar alerts can be fused if they have only minor temporal differences. SOAAPR differentiates between two cases in which either alerts are obtained from multiple algorithms running in parallel on one system or from multiple distributively operating ones.

In terms of clustering for anomaly-based IDS, GAC's clustering solution [337] is most related to SOAAPR. However, our approach differs in various ways. Since working on a finite set of alerts, it might be difficult for a security operator to choose an appropriate point in time when to start and end recording alerts which will be fed to GAC. If the recording time is too short, an attack might not yet be finished and further attack-related alerts are not included. If the recording time is too large, the GAC approach consumes considerable resources. This has been discussed in their work and was mitigated by chunk-based processing in which the alert data set is divided into smaller chunks which can efficiently be computed by GAC. Doing this might split the alerts of an attack scenario into two chunks. Furthermore, if no chunks are considered and no notion of time through alert timestamps is incorporated, it is very likely that alerts with similar attributes might be clustered together which have no temporal dependency and are from different attack scenarios. SOAAPR only mines for temporally "unique" attack stages by taking the timestamp information into account. Furthermore, it automatically checks whether a cluster is completed such that no old alerts get correlated into a recent cluster. Considering parametrization, GAC suffers from choosing the parameter k, searching for fully connected sub-graphs of size k within the community clustering, and a suitable threshold τ providing a minimum similarity value between alerts. The former allows the assignment of alerts to multiple clusters such that may clusters result that potentially contain alerts of unrelated attacks. For SOAAPR, we assume that each alert can only be assigned to one cluster and we handle confidence of alerts in a designated alert preparation stage. Furthermore, instead of having a minimum similarity threshold on the alert level, SOAAPR assigns alerts into clusters with the highest overall similarity. Having a minimum similarity on a cluster level, makes it more robust to fluctuations in the alert attributes.

With respect to streaming alert clustering, competitors mainly focused on multi-stage attack detection by correlating alerts enriched with the intrusion type and assuming that raised alerts correspond to a single-step attack. Therefore, streaming solutions are mainly designed for the application of misuse-based IDS. The incremental frequent structure mining approach in [348] is similar to our SOAAPR by creating frequency patterns of alerts as potential attack scenarios, i.e. single step attacks, and only incorporating IP-address information. However, our approach differs in various ways. Largely comparable to those patterns, the clusters in SOAAPR are obtained by more flexible comparison functions allowing for the consideration of various alert attributes. Problems with the temporal handling of patterns, stated in the outlook of [348], are slowly developing patterns with large delays in between the steps and the fixed *Keep_Active* parameter used to decide when a pattern is considered "stable", i.e. finished. By assigning individual and cluster-characteristic time values, SOAAPR is more flexible when determining a cluster as "saturated", i.e. finished. Additionally, requiring a mining interval time in which every x minutes the pattern tree structure is updated, makes it possible for an adversary to perform an attack unnoticed within this time span.

Attempts for discovering novel single-stage scenarios as part of multi-stage attacks are given by RTECA in [350], mining for critical episodes. However, although stating real-time operation, processing is only started when a window is completed. In contrast, for SOAAPR each alert is processed immediately and no window needs to be filled. Aggregated alerts in RTECA are merged by utilizing intrusion type and attack severity attributes. Our approach dispenses with this information and leverages attributes that can be provided by OD algorithms. Although similarity functions in RTECA used to update the correlation matrix are similar to ours, the online clustering process in SOAAPR is more efficient compared to the online attack tree generation when considering the processing of each newly arriving alert.

The online fuzzy clustering module used in [351] operates most similar to the clustering in SOAAPR. It exploits a common approach that similar alerts belong to an existing pattern (in our approach a cluster) whose similarity degree is high or triggers the generation of a new pattern in the case that existing ones are not sufficiently similar (minimum threshold of similarity). The fuzzy clustering approach also considers the notion of time which strengthens our approach introducing a "time to live" for each cluster. However, the threshold used for the life time of patterns is one timing constraint, which might be exploited by a strong adversary performing its attack until the life time is not exceeded. SOAAPR provides two timing constraints preventing such attacks and also takes care of considering that new alerts are not added to an existing cluster if the cluster's temporal existence is outdated. However, contrary to SOAAPR, this time difference is part of the similarity functions, which makes its threshold less intuitive for the operator. The parameter tuning is discussed by the authors in their conclusion and some parameter improvement is part of further work. A membership degree in the fuzzy patterns can be compared to the outlier score values of each alert representing an alert's confidence as part of a cluster. Although not relying on the intrusion type attribute, and thus adaptable for OD, it is designed and evaluated using alerts from misuse-based IDS.

With their recent work, Zhang et al. in [24] and [352] proposed real-time intrusion scenario detection methods. With IACF alerts streaming in real-time are grouped into actions based on the similarity of the intrusion type and destination port. The former strongly requires existing knowledge such as using misuse-based IDS which is a major difference to SOAAPR and our clustering, where the comparison functions is less strict but does not allow duplicate actions because of duplicated alerts or FP. Timely formed sessions from a sequence of actions are then split by a similar method used in SOAAPR to determine saturated clusters. However, we compute the discreteness of time intervals between alerts instead of actions on a session level in IACF. Furthermore, for the sake of reducing the impact of FP, the pruning algorithm might also filter out single-stage attacks. We deem this step as critical since we support and transfer the statement in [34] for OD-based AC that, especially for critical streaming applications, it is more important not to miss critical TP anomalies forming a single-stage attack while accepting a certain rate of FP.

In terms of attack characterization, we see the huge potential of generic signatures derived from attack characteristics as proposed by the motif-based approach in [54]. Thus, we evaluate the applicability of those communication-related fingerprints and extend them by considering feature attributes and a temporal behavior also potentially characterizing attacks, since we deem that the communication relation of attacks is not the only and best option to characterize attacks.

Contrary to the mined patterns in [348], SOAAPR provides a more unpredictable, cluster-specific generation of signatures. The mentioned high number of FP patterns is not surprising when dealing with alerts that do not state high confidence and map to a single attack step. SOAAPR will also likely be suffering from FP clusters due to the impossible prevention of FP and FN alerts in real-world applications and its time-specific online clustering approach. However, SOAAPR provides signatures from clusters that can be compared among each other or with ones previously stored as knowledge base, subsequently allowing for a deeper analysis by a human operator.

Derived candidate attack sequence patterns from the approach in [33] are composed of hyper-alert sequences within a certain time span, which characterize a multi-stage attack. Due to the different lengths of those patterns, a comparison with others is impeded and hyper-alerts still contain privacy-relevant information such as IP-addresses. The intrusion scenario construction phase in IACF [352] derives correlation graphs extracted from pruned sessions. However, as stated by the authors in the conclusion, the generated graphs are not very intuitive for human analysts. This is because sessions are decomposed into binary correlations of sessions. SOAAPR generates comparable, fixed-sized signatures in which similarity scores can be computed. Furthermore, those signatures are free from privacy-relevant information and can be shared with others. Signatures in our approach represent single-stage attack scenarios and not multi-stage attacks. However, by chaining our signatures, multi-stage attack comparison is also possible. Also depending on the attribute intrusion type, the authors' more recent work [24] performs online clustering in a similar manner as SOAAPR but clusters actions into clusters with high similarity of the type field. This makes the framework highly dependent on existing knowledge, for instance obtained from alerts generated by misuse-based IDS.

We want to note that especially much of the AC work, e.g. [151], which relies on misuse-based IDS, assumes that raised alerts can be treated as an attack, i.e. single-stage attack or attack step. Based on those assumptions, much work exists that identifies multi-staged attacks by analyzing those alerts. However, SOAAPR significantly differs from those by assuming that raised alerts from anomaly-based IDS, in particular in this work by OD algorithms, only represent indicators of potential known or unknown attack scenarios, i.e. single-stage attacks. By equipping OD algorithms to produce enriched alerts, SOAAPR mines alerts to identify those attack scenarios. Additionally, OD algorithms have their limitations when identifying a broad spectrum of attack scenarios based on the assumption that outliers are rare, distinct and do not happen frequently compared to normal data. Thus, it is difficult to produce alerts for the whole duration of long-term attacks with a massive amount of data, such as DoS-like or Brute Force, since online OD algorithms might adapt to them by supposing a concept drift. Furthermore, for attack detection using OD, it is assumed that detected outliers are indicators for maliciously triggered events although potentially being only an anomalous event rooted in a non-malicious fault, for instance. Intentional masking and swamping that might blend OD, as discussed in [353], are also not considered. Rather, we would like to point out with this work that identifying attack pattern from OD can work to a certain extent but in real-world applications we strongly suggest to leverage a hybrid system of anomaly- and misuse-based IDS. Although the intention of SOAAPR is highly ambitious, we see the approach confirmed since real attacks are more likely small probability events, and the most dangerous attacks only happen rarely [24, 354]. This so-called "rare data problem" is the case for which OD algorithms are especially predestined.

5.2 Streaming Outlier Analysis and Attack Pattern Recognition

5.2.1 Operation Principle

The workflow and operation principle for **S**treaming **O**utlier **A**nalysis and **A**ttack **P**attern **R**ecognition, denoted as **SOAAPR**, is shown in Figure 5.1. One may utilize SOAAPR in conjunction with online OD algorithms in two different interaction modes depending on the network infrastructure and the available resources. On the one hand, online OD can be performed self-sufficiently on a single system applying multiple OD algorithms in parallel, denoted as *single system - multiple algorithms*. The amount of algorithms in parallel depends on the available resources of the single system. All algorithms perform OD on the same data instances (data points) $\boldsymbol{x}_t$ with dimension d of the data stream $\{\boldsymbol{X}_t \in \mathbb{R}^{n_t \times d}, t = 1, 2, ...\}$. The continuous transmission of data records arrives sequentially at each time step t in which the count of features is denoted as d (dimension) and $\boldsymbol{x}_t$ the n_t-th d-dimensional most recent incoming data instance at time t. The feature set of $\boldsymbol{X}_t$ is denoted as $\mathcal{F} = \{f_1, f_2, ..., f_d\}$. *Alert Generation* and *Alert Preparation* can be performed locally on the single system after each classifier yields its result, either a TP or a FP. For a greater extent of visibility across larger network infrastructures, collaboratively operating IDS sensors, have found their way into alert detection or AC. Therefore, on the other hand, SOAAPR is able to deal with multiple self-sufficiently working online OD algorithms, denoted as *multiple systems - single algorithms*. Each distributively or decentrally applied algorithm in the network infrastructure operates on data instances $\boldsymbol{x}_t$, $\boldsymbol{y}_t$ or $\boldsymbol{z}_t$ from different data streams $\boldsymbol{X}_t$, $\boldsymbol{Y}_t$ and $\boldsymbol{Z}_t$. *Alert Preparation* in this interaction mode is performed by SOAAPR. A combination of both interaction modes, denoted as *multiple systems - multiple algorithms*, is also possible. Thus, *Alert Preparation* must be performed on both the multiple single systems utilizing multiple online OD algorithms and within SOAAPR processing the streaming alerts.

Alert instances $\boldsymbol{a}_t$ from the stream $\{\boldsymbol{A}_t \in \mathbb{R}^{m_t \times l}, t = 1, 2, ...\}$, generated by either a TP or a FP, are streaming into SOAAPR at each time step t, in which the count of attributes of $\boldsymbol{a}_t$ is denoted as l and $\boldsymbol{a}_t = \{a_1, a_2, ..., a_l\}$ the m_t-th l-dimensional most recent incoming alert at time t. Within SOAAPR, a *Buffer* is used to temporarily store alerts for the *Streaming Alert Correlation / Clustering*. No longer required alerts, e.g., from unusable clusters will be flushed by the *Discard Alerts & Clusters* module. The *Trigger Signature Generation* component monitors the evolving clusters $C_i^{(t)}$ from the set $\boldsymbol{C}^{(t)}$ and triggers the *Signature Generation* from suitable clusters, ideally representing an attack or step of an attack, denoted as attack scenario $S_i^{(t)}$ from the set $\boldsymbol{S}^{(t)}$, in order to create three types of signatures denoted as $sig_{<type>}^{(t)}$.

The further processing of the generated signatures depends on two operation modes - *Runtime* and *Learning* - based on whether an existing knowledge base is available or not. In the *Learning* phase, generated signatures can - if desired - be clustered according to their similarity and are presented to a human analyst. Instead of a massive amount of alerts generated without applying SOAAPR, the expert must only analyze a reduced amount of alerts already preprocessed in a set clustered as $C_i^{(t)}$ and ideally corresponding to the attack scenario $S_i^{(t)}$. In the case of a true attack scenario, the expert condenses information about the attack scenario using STIX and attaches the respective signature $sig_{<type>}^{(t)}$ to it. Since the resulting knowledge of the attack scenario only consists of fingerprint-like attack-characteristic information, free of privacy-relevant or confidential

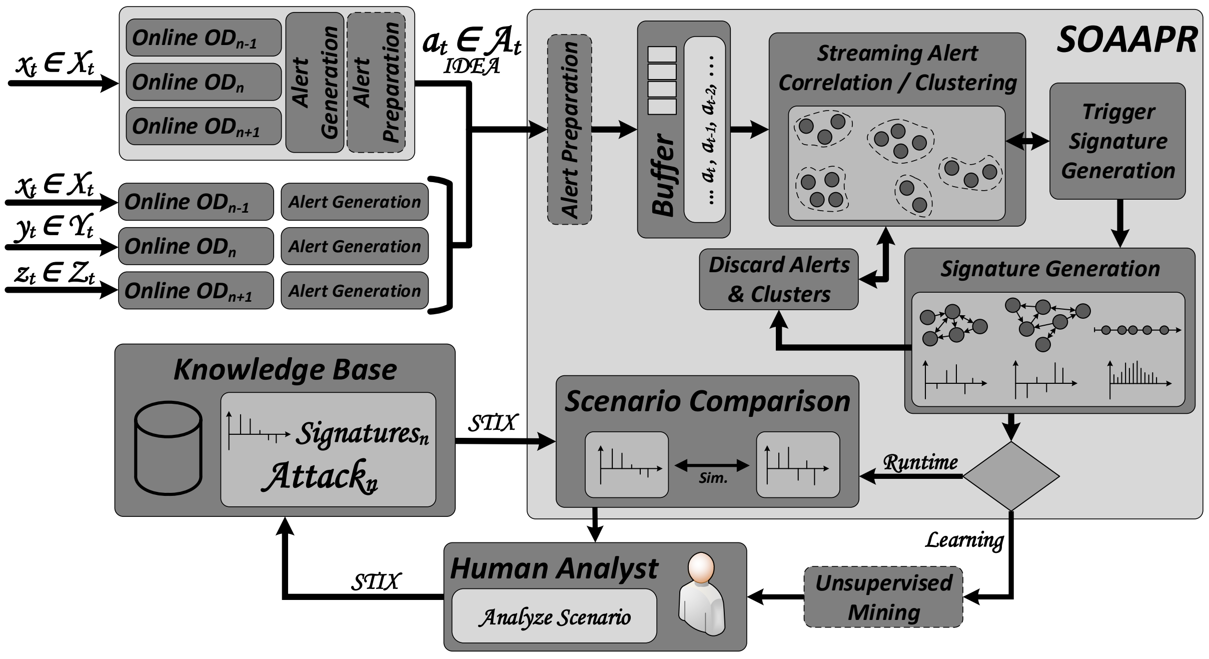


Figure 5.1: Flowchart and Operation Principle of Streaming Outlier Analysis for Attack Pattern Recognition.

data such as in-house IP-addresses. Thus, it can be shared in a cross-company manner. Having a knowledge base established, in the *Runtime* phase, generated signatures can be compared within the *Scenario Comparison* module by calculating similarity values. This way, one can take advantage of the strengths of misuse-based signatures with their fast, efficient and reliable attack identification capability but surpasses it via its ability to identify completely new, yet unknown, attack scenarios by being similar to known scenarios (from the same attack category) with a similar pattern. In the following, the core components of SOAAPR are discussed in more detail.

5.2.2 Alert Generation & Preparation

An OD algorithm $OD(\cdot) : \boldsymbol{x_t} \rightarrow f(\boldsymbol{x_t})$ assigns either a class label as given by $f(\boldsymbol{x_t}) \in \{normal, abnormal\}$ or a score value $f(\boldsymbol{x_t}) \in \mathbb{R}$, describing the strengths of anomalousness, for each data object in $\boldsymbol{X_t}$. Score values carry more information and aid the *Alert Preparation*, for instance, by dealing with FN. Thus, *Alert Generation* creates alarms utilizing the IDEA format for instance by enriching the intrinsic alert properties such as timestamp, IP-address and port source and destination information with (i) a normalized value of the outlier score and (ii) the respective top-γ-features mainly causing the outlierness as the subset $\mathcal{F}_S = \{f_{i1}, f_{i2}, ..., f_{i\gamma}\} \subseteq \mathcal{F}$.

We deem (i) crucial when combining multiple OD algorithms with different scoring functions, since, for instance, algorithms such as Kitsune [115] or Loda [119] score instances higher the more abnormal they are within their model ($f(\boldsymbol{x_t}) \in [0, \infty[$). This impedes setting a unified threshold value across all applied OD algorithms, in particular, when they operate on different hyperparameter sets. As an example, Loda utilizing two alternating sets of histograms with different window sizes will yield different averaged score values due to the different state of knowledge about normal data. With increasing window sizes, the models become more accurate while incorporating larger amounts of normal data instances which, in turn, are then scored less compared to models obtained from smaller window sizes. We suggest normalizing the outlier scores using an improved version of the *Gaussian Scaling* proposed by [304] in which the mean μ and the standard deviation σ are used to encounter the aforementioned problems. Since μ and σ work well for normally distributed values, i.e., assuming a normal distribution of the outlier scores, we replace them by the median *med* and the median absolute deviation *mad* because they are a better option for distributions with skewness.

Figure 5.2 depicts an exemplary distribution showing non-negative (positive) skewness of the Loda online algorithm utilized on the fourth CSV file of the security-related UNSW-NB15 [335] data set. It clearly points out the difference of mean and median caused by the unequal ratio of outliers. Thus, the normalization formula leads to Equation 5.1 in which $erf()$ is the monotone and ranking stable Gaussian Error Function and med_t as well as mad_t are moving or rolling variants of the median and median absolute deviation to be applied in a streaming fashion.

$$\tilde{f}(\boldsymbol{x_t}) = max\{0, erf(\frac{f(\boldsymbol{x_t}) - med_t}{mad_t\sqrt{2}})\} \tag{5.1}$$

Applying this formula will translate the arbitrary outlier score values in the range $[0, 1]$ as interpretable values describing the probability of a data instance being an outlier. Using domain expertise, a reasonable threshold can be determined over runtime yielding a decent classification performance that assigns a binary value from the set $\{normal, abnormal\}$ for each $\tilde{f}$.

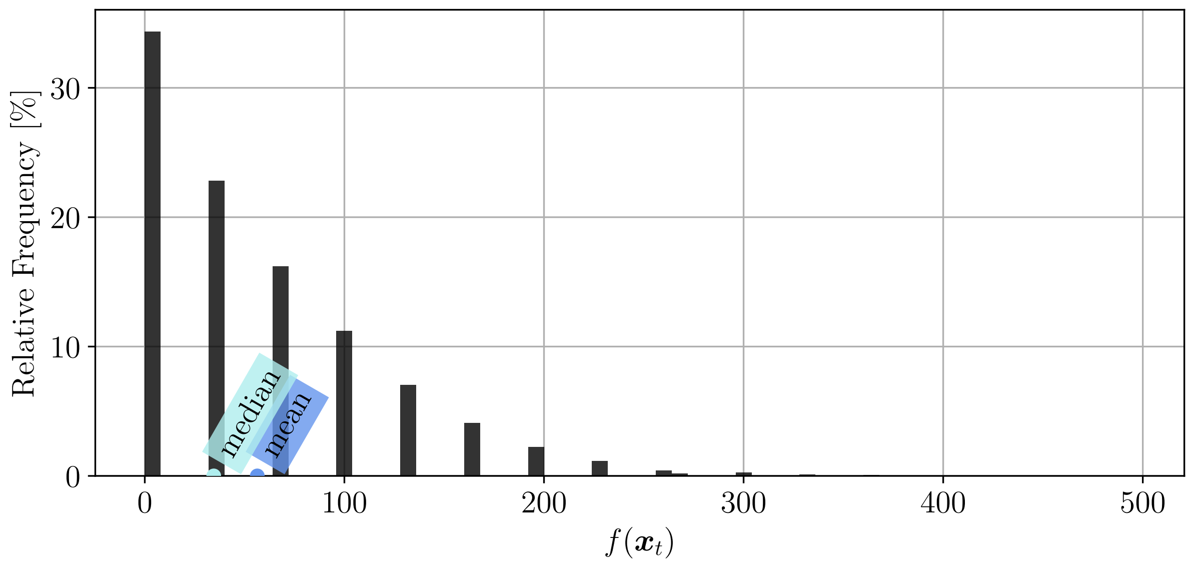


Figure 5.2: Exemplary Loda outlier score values including the mean and median.

In terms of *Alert Preparation*, it must be distinguished between the modes *single system - multiple algorithms* and *multiple systems - single algorithms*. In general, *Alert Preparation* helps to reduce FP as well as FN by exploiting the strengths of multiple classifiers sending their alerts. For *single system - multiple algorithms*, the classifier processes the same data instance from the stream $\boldsymbol{X}_t$ in parallel, thus feature interpretability (additional alarm attribute (ii)) is not mandatory since it can strongly be assumed that resulting outliers from the same data instance are caused by the same feature deviating too much from normal behavior.

However, having *multiple systems - single algorithms*, with the feature information causing the outlierness, alerts can be assigned to the same event but from different stream perspectives, e.g., $\boldsymbol{Y}_t$ or $\boldsymbol{Z}_t$, considering the timestamp information as well. Even in the distributed case, an attack might cause outliers close in time to be detected by the classifiers which subsequently generate alerts. If the classifiers are able to determine the features causing the outlierness, those alerts can be mapped to the same event with high confidentiality whose γ most contributing features have high similarity. Independent of the interaction mode, alerts generated from detected outliers of the same event (data instance) by multiple classifiers can be fused together to reduce the amount of alerts that SOAAPR needs to process (alert filtering). We suggest deriving meta-alerts that summarize the multiple alerts generated by the classifiers for the same event. Having the same alert attributes in terms of IP-address and port source and destination information with timestamps close in time (and highly similar respective features causing the outlierness for the *multiple systems - single algorithms* option) but with different normalized outlier score values, those alerts can be mapped to a single meta-alert. The normalized outlier score values (additional alarm attribute (i)) are combined by $\tilde{f}_{meta} = \sum_i g_i \tilde{f}_i$ into a meta outlier score depending on potentially additional weights g_i for each classifier where $\sum_i g_i = 1$ and $\tilde{f}_{meta} \in [0, 1]$ applies. For the sake of simplicity, we assume equal weights in the following.

Table 5.1 shows an example of five online OD algorithms classifying five data instances with ground truth $\{normal, normal, abnormal, normal, abnormal\}$. Further, we assume an anomaly threshold of > 0.5 which triggers the generation of alerts by each classifier. A normal data instance can either be a TN or FP and an abnormal data point can be either a TP or a FN. Having ground truth information available, even for misclassification of individual classifiers, meta-alerts would be generated with high confidence referring to the meta outlier score value, denoted as $\tilde{f}^{(GT)}_{i(meta)}$. Since alerts are only generated from TPs and FPs, $\tilde{f}_{i(meta)}$ can lead to falsely generated meta-alerts without incorporating information on how many classifiers generated alerts. Thus, the meta-alert outlier score can be penalized by some measure, e.g., utilizing majority voting or the ratio of alerts generated to the number of classifiers. Taking advantage of the latter leads to a penalized meta-alert outlier score, denoted as $\tilde{f}^{(penalized)}_{i(meta)}$. Setting a dedicated threshold for meta-alert generation based on the penalized outlier scores, for instance 0.3 in Table 5.1, significantly reduces FPs and potentially alleviates the number of FNs that limit the alert generation by obscuring the actual TPs.

It is noted that this measure only works for *single system - multiple algorithms* since those algorithms definitely operate on the same input data and the total number of classifiers processing the same event can be determined. In contrast, the number of classifiers that operate distributively in the *multiple systems - single algorithms* case, processing the same event, cannot be reliably determined.

5.2.3 Streaming Alert Correlation & Clustering

The core component of SOAAPR in order to group incoming alerts into clusters, potentially representing attack scenarios, is the *Streaming Alert Correlation / Clustering* component. It is again noted, that we are not yet interested in uncovering attack campaigns or multi-stage attacks but attack scenarios or attack steps that are rooted in timely

Table 5.1: Exemplary meta-alert generation based on five online OD algorithms classifying five data instances (3x normal, 2x abnormal) (yellow-colored background for alerts generated by each classifier for True Positives (TP) and False Positives (FP); red-colored background for critical False Negatives (FN); (GT) denotes if Ground Truth about the binary classification was available).

classifier	**normal**	**normal**	**abnormal**	**normal**	**abnormal**
#1	TN (0.2)	TN (0.3)	TP (0.9)	FP (0.6)	FN (0.5)
#2	TN (0.1)	FP (0.7)	FN (0.5)	TN (0.3)	TP (0.9)
#3	FP (0.6)	FP (0.6)	TP (0.9)	TN (0.3)	FN (0.4)
#4	TN (0.1)	TN (0.2)	FN (0.5)	TN (0.2)	TP (1.0)
#5	TN (0.2)	TN (0.2)	TP (0.8)	TN (0.1)	FN (0.5)
$g_i^{(GT)}$	0.20	0.20	0.20	0.20	0.20
$\tilde{f}_{i(meta)}^{(GT)}$	0.24	0.40	0.72	0.30	0.66
g_i	1.00	0.50	0.33	1.00	0.50
$\tilde{f}_{i(meta)}$	0.60	0.65	0.87	0.60	0.95
$\frac{\#alerts}{\#classifiers}$	1/5	2/5	3/5	1/5	2/5
$\tilde{f}_{i(meta)}^{(penalized)}$	0.12	0.26	0.52	0.12	0.38
Meta-Alert	✗	✗	✓	✗	✓

correlated outliers. Although so-called Advanced Persistent Threats are characterized by intelligent adversaries that might exploit the notion of time by stealthily prolonging their attack campaign, we are focusing on the preliminary steps adversaries have to undertake which are of a reasonable finite length of time. For multi-stage attack uncovering, we refer to other work such as [343, 355].

Table 5.2 shows the 14 different attack scenarios of the CICIDS2017 data sets together with their respective characteristics in terms of number of (anomalous) records associated for each attack and their duration. Furthermore, the outlier percentage of each attack within the data set is given. Since OD is tailored for highly imbalanced data, it won't work well for the *DoS Hulk*, *Portscan* and *DDoS* attack. However, especially for DoS-like attacks, plenty of work exists for detection and prevention [356, 357, 358]. It can clearly be seen from the table that, on average if we exclude the DoS type, attacks are typically not longer than 4 h. Furthermore, we assume that attacks are rooted in a reasonable amount of outliers such that - if clustered - meaningful signatures of those potential attack scenarios can be derived. The number of instances in Table 5.2 confirms this assumption by showing the average amount of (anomalous) instances per attack scenario and attack category (excluding the three above mentioned) is 2,830 with a minimum of 11 for *Heartbleed*. However, we want to note that each attack category has its unique characteristics in terms of average number of instances per attack scenario and their duration. Thus, it might be reasonable to apply multiple instances of SOAAPR each adjusted for a dedicated attack category. This would allow to apply SOAAPR with other detection mechanisms, except OD ones, which might also be tailored for DoS-like attacks.

Furthermore, we deem it more reasonable that each incoming alert is only assigned to one cluster. This stands in contrast to approaches such as [337] whose community clustering might assign alerts to several clusters under the assumption of dealing with

Table 5.2: The 14 different attacks with respective characteristics of the CICIDS2017 data sets ([*]Although labeled as one attack scenario in the data set, it actually consists of three short-term ones: Meta exploit Win Vista, Infiltration - Cool disk - MAC and Infiltration - Dropbox download - Win Vista).

Data set	Attack Type	# Instances	Outliers (%)	Duration (min)
Tuesday-WorkingHours	SSH-Patator	5,897	1.32	62
	FTP-Patator	7,938	1.78	73
Wednesday-WorkingHours	DoS Hulk	231,073	33.35	24
	DoS GoldenEye	10,293	1.49	9
	DoS Slowloris	5,796	0.84	467
	DoS Slowhttptest	5,499	0.79	22
	Heartbleed	11	0.002	20
Thursday-WorkingHours-Morning	Web Attack - Brute Force	1,507	0.88	45
	Web Attack - XSS	652	0.38	20
	Web Attack - Sql Injection	21	0.01	2
Thursday-WorkingHours-Afternoon	Infiltration[*]	36	0.01	86
Friday-WorkingHours-Morning	Bot	1,966	1.03	205
Friday-WorkingHours-Afternoon	Portscan	158,930	55.48	138
Friday-WorkingHours-Afternoon (2)	DDoS	128,027	56.71	20

uncertainty in clustering and, although clusters might contain alerts of unrelated attacks, they might include all TP alerts. This assumption can be seen as critical since alerts of unrelated attack scenarios, denoted in this work as "noisy", might lead to blurred signatures and similar attack scenarios in the future may lead to completely different signatures without those additional noisy alerts. SOAAPR operates on alerts with high confidentiality of TPs due to the *Alert Preparation* mechanism.

The *Buffer* component serves as a "First In - First Out" data queue which helps to cope with potential bursts of alert floods while relaxing the processing time of the *Streaming Alert Correlation / Clustering* module. The streaming alert processing is performed as provided with Algorithm 6 for each oldest (first) alert $\boldsymbol{a}_t$ in the *Buffer*. The timely order of alerts is assumed and to be given by the *Alert Preparation* component. As of now, we limit ourselves to five alert attributes for streaming clustering: IP-address (source, destination - $\{ip_{src}, ip_{dst}\}$), port information (source, destination - $\{port_{src}, port_{dst}\}$) and the timestamp indicating the time when the outlier was detected, denoted as t_a. Moreover, one is able to weight alert attributes, for instance, to give less weight to the source IP-address or source port as an adversary might spoof it during its attack scenario [352].

In order to achieve streaming clustering for our purposes, we extend each cluster $C_i^{(t)}$ with additional properties beyond the simple subset of alerts, which we deem mandatory to answer the following fundamental questions: Firstly, when is a cluster saturated, i.e., when is it ready for the process of signature generation? Secondly, when is a cluster with its alerts considered outdated and should be discarded? To answer those two questions, we refer to Table 5.2 and define two significant user-definable parameters: a maximum total time to live for a cluster, denoted as t_{ttl}, and a minimum number of alerts that should be clustered in order to reasonably represent an attack scenario for signature derivation, denoted as min_alerts. Based on those parameters, clusters are assigned the additional information of its create timestamp t_c, a last alert added timestamp t_{laa}, an alert counter cnt, an expiry date t_{expiry}, a flag *state* indicating a cluster's state *evolving*, *saturated* or *discard* and an alert adding frequency mean t_freq_{mean} and its standard deviation t_freq_{std}. The latter two characteristics are used to determine whether a cluster could be seen as completed in a timely fashion without stressing the very latest expiry date.

Algorithm 6: Operation Principle of *Streaming Alert Correlation / Clustering.*

```
Input: Alert $\boldsymbol{a}_t$ composed of attributes $\{ip_{src}, ip_{dst}, port_{src}, port_{dst}, t_a\}$, Minimum
       Similarity $min\_sim$, A cluster's total time to live $t_{ttl}$
Output: Updated set $C^{(t)}$ of clusters each $C_i^{(t)}$ composed of an alert subset
        $\boldsymbol{A}_i^{(t)} \subset \boldsymbol{A}_t$, a create timestamp $t_c$, a last alert added timestamp $t_{laa}$, an
        alert counter $cnt$, an expiry date $t_{expiry}$, an alert adding frequency mean
        $t\_freq_{mean}$ with its standard deviation $t\_freq_{std}$, a flag $state$ indicating
        a cluster's state "evolving", "saturated" or "discard"
 ▷Initialization - add alert to new cluster
1  if $C^{(t)} == \varnothing$ then
2      $C_0^{(t)} \leftarrow \boldsymbol{a}_t$    $t_c^{(C_0)} \leftarrow t_a$    $t_{laa}^{(C_0)} \leftarrow t_a$    $t_{expiry}^{(C_0)} = t_c^{(C_0)} + t_{ttl}$
3      $cnt^{(C_0)} \leftarrow 1$    $state^{(C_0)} \leftarrow evolving$
4  $best\_cluster \leftarrow 0$
5  $highest\_sim \leftarrow 0$
 ▷Iterate over all existing clusters
6  for $i$ in $C^{(t)}$ do
       ▷Clusters are regularly updated - check if cluster is still evolving
7      if $state^{(C_i)} == evolving$ then
           ▷Compute the similarity of the new alert with the cluster (Equation 5.2
           & 5.3)
8          $sim \leftarrow$ similarity($C_i^{(t)}$, $\boldsymbol{a}_t$)
9          if $sim > highest\_sim$ then
10             $best\_cluster \leftarrow i$
11             $highest\_sim \leftarrow sim$
12 $no\_clusters \leftarrow i$
13 $i \leftarrow best\_cluster$
 ▷Add alert to existing cluster or create new cluster
14 if $highest\_sim \geq min\_sim$ AND $state^{(C_i)} == evolving$ then
       ▷Add alert to existing cluster
15     $C_i^{(t)} \leftarrow \boldsymbol{a}_t$    $cnt^{(C_i)} \leftarrow cnt^{(C_i)} + 1$
16     $t\_freq_{mean}^{(C_i)} \leftarrow \mathbf{t\_freq_{mean}^{(C_i)}.moving\_mean}(t_a - t_{laa}^{(C_i)})$
17     $t\_freq_{std}^{(C_i)} \leftarrow \mathbf{t\_freq_{std}^{(C_i)}.moving\_std}(t_a - t_{laa}^{(C_i)})$
18     $t_{laa}^{(C_i)} \leftarrow t_a$
19 else
       ▷Add alert to new cluster
20     $i \leftarrow no\_clusters$    $C_i^{(t)} \leftarrow \boldsymbol{a}_t$    $t_c^{(C_i)} \leftarrow t_a$    $t_{laa}^{(C_i)} \leftarrow t_a$
21     $t_{expiry}^{(C_i)} = t_c^{(C_i)} + t_{ttl}$    $cnt^{(C_i)} \leftarrow 1$    $state^{(C_i)} \leftarrow evolving$
22 return $C^{(t)}$
```

Thus, as long as the minimum number of alerts per cluster is not reached, the time difference between each alert is computed and their moving mean and standard deviation computed by, e.g., the well-known Welford's algorithm [305]. If the minimum number of alerts per cluster is reached, alerts can still continuously be added to a correlated cluster as long as this process happens frequently. However, if no more alerts have been added to a specific cluster for some time, it is considered as *saturated* and ready for signature generation. A trivial but yet effective method is a one-dimensional (time) OD method based on the assumption that, if a value is a certain number of standard deviations away from the mean, that data point is identified as an outlier. The specified number of standard deviations is called the threshold whose value is user-definable, denoted as frequency exceeding threshold k.

With this set of information, we are able to answer the above questions. Therefore, a cluster is considered *saturated* and ready to *Trigger Signature Generation* when it has reached a minimum number of alerts and (from a time perspective) either the expiry date ($t_{expiry} = t_c + t_{ttl}$) has been exceeded or for a certain amount of time no more alerts have been added to it. Both time constraints are given by $t_{now} > t_{expiry}\ OR\ t_{now} > t_{laa} + t\_freq_{mean} + k \cdot t\_freq_{std}$, in which t_{now} is the current time. Both options are mandatory since the expiry date prevents, apart from the circumstance that falsely triggered alarms continuously keep clusters alive, that an adversary might unceasingly trigger alerts on purpose such that a cluster is not considered saturated for the duration of the attack concealing its actual one. The second part of the time constraint prevents an adversary from leveraging its attack until the expiry date is met, allowing a maximum attack time period of t_{ttl}. SOAAPR is also robust to attack scenarios that can be performed on sliding window based approaches as discussed in [346] in which an attacker can prevent two attack steps from falling into one window. In order to further lower the determinism for an attacker to not exploit either time boundary, a certain amount of jitter might be introduced. Thus, similar to the mechanism in [359], the attacker might not exactly guess both timing constraints.

Algorithm 7 is proposed to monitor the clusters in a regular time-triggered manner (each time step Δt) to check their conditions. This is necessary since, if no alert is streaming in SOAAPR for a longer time span, clusters for the event-triggered case might already be considered *saturated* or to be *discarded*.

For better comprehensibility, Figure 5.3 shows an exemplary scenario for two observation times (a) and (b) of (timely) evolving clusters in three dimensions - two hypothetical alert attributes and the dimension of time. Although C_{i+1} in (a) might have attribute correlation with C_i, C_i is already considered *saturated* since the alert within C_{i+1} is outdated due to the frequency constraint. A third cluster C_{i+2} is currently in the *evolving* state. Since C_i was ready for signature generation and C_{i+1} could not reach a minimum number of alerts till the expiry date, assigned the state *discard*, both have been removed as shown in state (b). Cluster C_{i+2} is assigned the state *saturated* in (b) since it reached the minimum number of alerts and no more alerts have been added for a certain amount of time. A fourth cluster C_{i+3} recently added two alerts and thus is in the *evolving* state. Furthermore, a cluster with its corresponding set of alerts can be discarded by the *Discard Alerts & Clusters* module, when a signature of a cluster has been computed (assuming that the above conditions are met) or if the minimum number of alerts per cluster could not be reached before the cluster is considered to be expired.

Many algorithms for data stream clustering exist that operate on some similarity measure such as the partition-based approaches incremental k-means, HPStream or CluStream, which has already been used for AC in [33], or DenStream, I-DBSCAN or LDBSCAN as density-based cluster-methods [360]. For instance, established clusters feature a cluster centroid each in l dimensions and a new alert, characterized as a l-dimensional data point, could be added to the cluster whose distance to the data point is the lowest.

Algorithm 7: Monitoring of *Trigger Signature Generation* and *Discard Alerts & Clusters.*

```
Input: Time step $\Delta t$, Minimum number of alerts $min\_alerts$, Frequency
       exceeding threshold $k$, Set of clusters $C^{(t)}$
Output: Set of clusters with state "saturated" $C^{(t)}_{saturated}$ and "discard" $C^{(t)}_{discard}$
▷Regularly update clusters
1  foreach $\Delta t$ do
2      $t_{now} \leftarrow$ current_time()
       ▷Iterate over all existing clusters
3      for $i$ in $C^{(t)}$ do
           ▷Check if cluster is "saturated" or needs to be "discarded"
4          if $(t_{now} > t^{(C_i)}_{expiry})$ OR $(t_{now} > t^{(C_i)}_{laa} + t\_freq^{(C_i)}_{mean} + k \cdot t\_freq^{(C_i)}_{std})$ then
5              if $cnt^{(C_i)} \geq min\_alerts$ then
6                  $state^{(C_i)} \leftarrow saturated$
7              else
8                  $state^{(C_i)} \leftarrow discard$
9          if $state^{(C_i)} == saturated$ then
10             $C^{(t)}_{saturated} \leftarrow C^{(t)}_i$
11         if $state^{(C_i)} == discard$ then
12             $C^{(t)}_{discard} \leftarrow C^{(t)}_i$
13 return $C^{(t)}_{saturated}, C^{(t)}_{discard}$
```

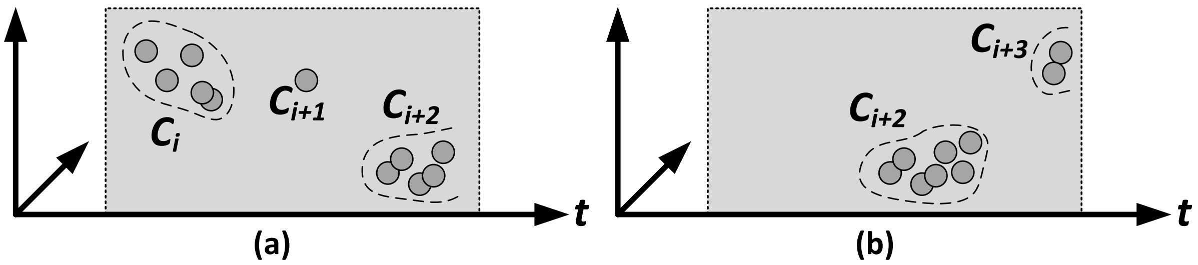


Figure 5.3: Exemplary scenario of evolving alert clusters for two different observation times (a) and (b) - indicated by the frames with gray background - with two hypothetical alert attributes and the timestamp attribute.

Adding the time t as an additional dimension, one could obtain moving (evolving) clusters in t which gets completed as time passes, as illustrated in Figure 5.3. When a new alert is very far away in time from the ones within the cluster, even if the other alert attributes are highly similar, it might not be added to it if the time dimension's weight lets the similarity fall below the minimum similarity. However, we have deliberately decided against this approach for two reasons. Firstly, alert attributes are coordinates in the l-dimensional space and similarity is only a measure for each coordinate. If one is additionally interested in the similarity between two different coordinates, such as the similarity of the source IP from one alert with the destination IP of another alert, this procedure cannot be utilized. Secondly, the setting of a minimum similarity value min_sim as a criterion, whether to add an alert to an existing cluster or not, depends on t which significantly decreases its interpretability. The graph-based approach in GAC allows arbitrary similarity functions, although the authors limited the attributes to only compare $\{ip_{src}, ip_{dst}, port_{src}, port_{dst}\}$ between any two alerts. It is not able to incorporate timing information and an edge in the graph between two nodes (alerts) is only added if a minimum similarity value is reached. This fine-granular setting significantly influences the resulting alert similarity graph and the final alert clusters obtained from it. Thus, we have chosen a more general approach by adding an alert to a cluster in a streaming fashion by measuring the similarity of a new alert $\boldsymbol{a}_t$ with the whole i-th cluster $C_i = \{\boldsymbol{a}_j | j \in \mathbb{Z}, 1 \leq j \leq |C_i|\}$ by utilizing Equation 5.2 in which $C_i[j]$ is the j-th alert $\boldsymbol{a}_j$ of C_i. The notion of time is, contrary to [337], included by the time constraints of expiry date and frequency transgression while not blurring min_sim in a timely manner.

$$sim(\boldsymbol{a}_t, C_i) = \frac{\sum_{j=1}^{|C_i|} alert\_sim(\boldsymbol{a}_t, C_i[j])}{|C_i|} \in [0,1] \tag{5.2}$$

The similarity between the new alert $\boldsymbol{a}_t$ and the j-th alert $\boldsymbol{a}_j$ of C_i can be computed by Equation 5.3 utilizing attribute-specific comparison functions, denoted as $f_k(\boldsymbol{a}_t^{(x)}, \boldsymbol{a}_j^{(y)})$, which might individually be weighted by w_k (typically $\sum_{k=1}^{K} w_k = 1$) for a total amount of K comparison functions and x, y not necessarily the same attributes of $\boldsymbol{a}_t$ and $\boldsymbol{a}_j$.

$$alert\_sim(\boldsymbol{a}_t, \boldsymbol{a}_j) = \frac{\sum_{k=1}^{K} w_k \cdot f_k(\boldsymbol{a}_t^{(x)}, \boldsymbol{a}_j^{(y)})}{\sum_{k=1}^{K} w_k} \in [0,1] \tag{5.3}$$

For network-related alerts, the attributes $\{ip_{src}, ip_{dst}, port_{src}, port_{dst}\}$ are the most important and common ones [337]. However, contrary to GAC with $f_k \in \{0,1\}$, whether the compared attributes are unequal (0) or equal (1), and $x == y$ for each k, we utilize comparison functions shown in Table 5.3. In addition, with regard to $x \neq y$, we check whether the source IP-address of the new alarm compares to the destination IP-address of an existing alert which might be an indicator that a host was already compromised taking into account that the victim communicates with the attacker. The same applies for the identity check of the source port of $\boldsymbol{a}_t$ with the destination port of $\boldsymbol{a}_j$.

Table 5.3: Proposed comparison functions f_k to compute the similarity between alert $\boldsymbol{a}_t$ and $\boldsymbol{a}_j$ utilizing the alert attributes $\{ip_{src}, ip_{dst}, port_{src}, port_{dst}\}$.

f_k	Computation
$f_1 \in \{0,1\}$	$ip_{src}^{(\boldsymbol{a}_t)} \overset{?}{=} ip_{src}^{(\boldsymbol{a}_j)}$
$f_2 \in \{0,1\}$	$ip_{dst}^{(\boldsymbol{a}_t)} \overset{?}{=} ip_{dst}^{(\boldsymbol{a}_j)}$
$f_3 \in \{0,1\}$	$ip_{src}^{(\boldsymbol{a}_t)} \overset{?}{=} ip_{dst}^{(\boldsymbol{a}_j)}$
$f_4 \in \{0,1\}$	$port_{dst}^{(\boldsymbol{a}_t)} \overset{?}{=} port_{dst}^{(\boldsymbol{a}_j)}$
$f_5 \in \{0,1\}$	$port_{src}^{(\boldsymbol{a}_t)} \overset{?}{=} port_{src}^{(\boldsymbol{a}_j)}$
$f_6 \in \{0,1\}$	$port_{src}^{(\boldsymbol{a}_t)} \overset{?}{=} port_{dst}^{(\boldsymbol{a}_j)}$

For iterating over all existing clusters of the current set $C^{(t)}$ with each new alert $\boldsymbol{a}_t$ and calculating the similarity of it with the subset of alerts in each $C_i^{(t)}$, one might assume high complexity in terms of time and space. However, the number of comparison functions K is fixed and the number of clusters in the current set $|C(t)|$ can be seen as fixed as well since it only fluctuates while new clusters occur but expired or saturated clusters disappear over time. Thus, with respect to streaming alerts $\boldsymbol{a}_t$, the *Streaming Alert Correlation / Clustering* module only has linear time and space complexity since only the number of alerts in an existing cluster can increase until it is expired, which demands space and time by computing the overall similarity of the new alert with all alerts of each existing cluster. As already mentioned, an attacker might take advantage of producing as many "decoy" alerts as possible to keep alive and fill a cluster until t_{expiry} is satisfied to stress out the time and space complexity. However, the attacker might only be able to trigger a limited amount of decoy alerts since the alerts are generated by the online OD algorithms and too many "outliers" would represent a concept drift in $\boldsymbol{X}_t$. Thus, malicious data might be seen as normal after a certain amount of time and no more alerts that stress SOAAPR are generated. SOAAPR achieves a decent tradeoff between a "real-time" detection by ideally analyzing each individual alert generated by online OD in order to immediately react to an attack (which is impossible in real-world scenarios by the massive amount of streaming alerts afflicted with FP and FN) and a "near real-time" detection with a certain delay by the clustering process to obtain a decent amount and human-manageable set of alerts representing potential attack scenarios.

5.2.4 Signature Generation & Sharing

Inspired by the idea in [54] to derive privacy-preserving signatures and fingerprint-like characteristics of novel attack patterns, by only utilizing the alert information commonly available with IP and port information, we extend it in SOAAPR. Clusters containing a huge number of alerts representing an attack scenario can be significantly reduced to a fixed-sized characteristic by transforming the communication relation of hosts, involved in an attack, into a directed graph-based structure to derive so-called motif signatures initially proposed in [345]. This enables a more fine-grained characterization of attacks compared to other work discussed in Section 5.1.3, such as [337], which differ between four types of communication patterns. However, with the ever-increasing complexity of novel attacks, we deem that (i) not only the communication relation is a mandatory attack characteristic but mandatory is also (ii) the data's attributes, features of the data $\boldsymbol{X}_t$, which are mainly responsible for shaping an attack and, thus, causing outliers, as well as, (iii) the temporal pattern between the alerts. Considering those three metrics for fingerprinting clusters by deriving three signatures denoted as sig_{com} (i), sig_{attr} (ii) and sig_{temp} (iii), we can achieve a more comprehensive and even more fine-grained characterization and comparison of attack scenarios, still satisfying the privacy-preservation benefit.

We see ourselves encouraged in our assumption of introducing the additional signature sig_{attr} since certain types of attacks and their affected outliers are predominantly caused by the same features. As could be shown in [303], some features have been more significant over multiple attack scenarios, e.g., *B.Packet Len Std*, *Flow Duration* or *Flow IAT Std* and certain types of attacks are more reflected by dedicated features referring to Table 3 in [303]. For instance, *Subflow Fwd Bytes* and *Total Length Fwd Package* are most influential for Infiltration and Bot attack types or the *Bwd Packet Length Std* is a typical feature whose outlierness indicates DoS-like attacks [303, 361].

Likewise, our assumption proposing sig_{temp} is strengthened by the statement in [352] that a series of intrusion actions performed by an attacker is more concentrated in the temporal domain than random FPs. As similarly stated in [352], sig_{temp} will likely not characterize a real attack with precision, since an attacker might try to manipulate the timing of its attack steps forging the time interval between triggered alerts or sig_{temp} might be blurred by FP alerts, it is nevertheless a reasonable approach for capturing the temporal behavior of attacks. In particular, it is easier for an attacker to manipulate timing of a multi-stage attack than for a single step (focus of SOAAPR) since some tools, e.g., Metasploit, are often used, whose execution, resulting in potential alerts, might not be tampered in a timely way.

Although each signature might not fully characterize an attack on its own, such as sig_{com} proposed in [54], the combination of the signature triplet, potentially weighted as well, better allows to characterize attacks and to find novel patterns that somehow share similarities with other signatures. Thus, deviations of one of the signatures from similar attack scenarios can be better compensated by the others. It must be noted that meaningful signatures can be derived when a cluster contains all relevant instances of an attack, ideally free from FPs and FNs, which is an especially ambitious intention utilizing OD. Having the knowledge about the communication relation of hosts, which features are the most important for a certain attack scenario and the timing behavior of a potential attack scenario, significantly provides a more intuitive root cause analysis process for a human expert than analyzing correlated alerts on their own.

Generation and Comparison of sig_{com}

Since it is described in detail in [54], we only provide the most important steps to perform signature generation for sig_{com} and comparison but strongly recommend the original work. In order to derive sig_{com}, we take advantage of the alert attributes $\{ip_{src}, ip_{dst}, port_{src}, port_{dst}\}$ representing the communication structure of two hosts communicating over a port ($ip_{src} : port_{src} \rightarrow ip_{dst} : port_{dst}$). Those attributes are contained in each alert $\boldsymbol{a}_j$ of the i-th cluster C_i as shown in Figure 5.4. To *transform* all alerts into a network graph $G_{com}(C_i)$, nodes of the graph either represent hosts via the IP-address $\{ip_{src}, ip_{dst}\}$ or ports which are bound to the respective IP-address $\{ip_{src} : port_{src}, ip_{dst} : port_{dst}\}$. Edges of the graph are reflected by information on who attacked whom, $\{(ip_{src}, ip_{src} : port_{src}), (ip_{src} : port_{src}, ip_{dst} : port_{dst}), (ip_{dst}, ip_{dst} : port_{dst})\}$.

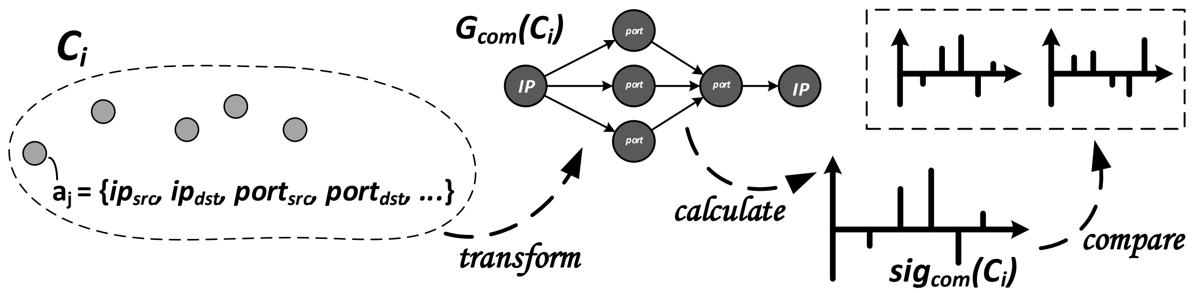


Figure 5.4: Generation and comparison process of the signature sig_{com} with respect to [54].

With this intuitive communication direction, a directed graph structure is obtained. In order to *calculate* the signature, all sub-graphs $G^*_{com} \subseteq G_{com}$ are enumerated, which are assigned to motif patterns m_i. The accumulated number of occurrences of every m_i will yield an absolute signature M^A. Since M^A depends on the graph size, a comparison can be achieved by utilizing the so-called Z-Score [345]. Briefly summarized, a random graph structure with the same size and the same number of edges as G_{com} is derived and utilized to generate a "relative" signature denoted as M^Z, which represents $sig_{com}(C_i)$ with respect to Figure 5.4. In order to *compare* two such signatures, M^Z_1 and M^Z_2, they are interpreted as two vectors of fixed length in a multi-dimensional space. The similarity between M_1^Z and M_2^Z, independent of their length, can then be derived by calculating the angle ϕ between the two vectors, as shown in Equation 5.4, in which $< \vec{M}_1^Z, \vec{M}_2^Z >$ is the inner product and $||\vec{M}_{1/2}^Z||_2$ the Euclidean norms.

$$sim(M_1^Z, M_2^Z) = \frac{cos^{-1}(\phi)}{\pi}; \quad cos(\phi) = \frac{< \vec{M}_1^Z, \vec{M}_2^Z >}{||\vec{M}_1^Z||_2 \cdot ||\vec{M}_2^Z||_2} \tag{5.4}$$

Generation and Comparison of sig_{attr}

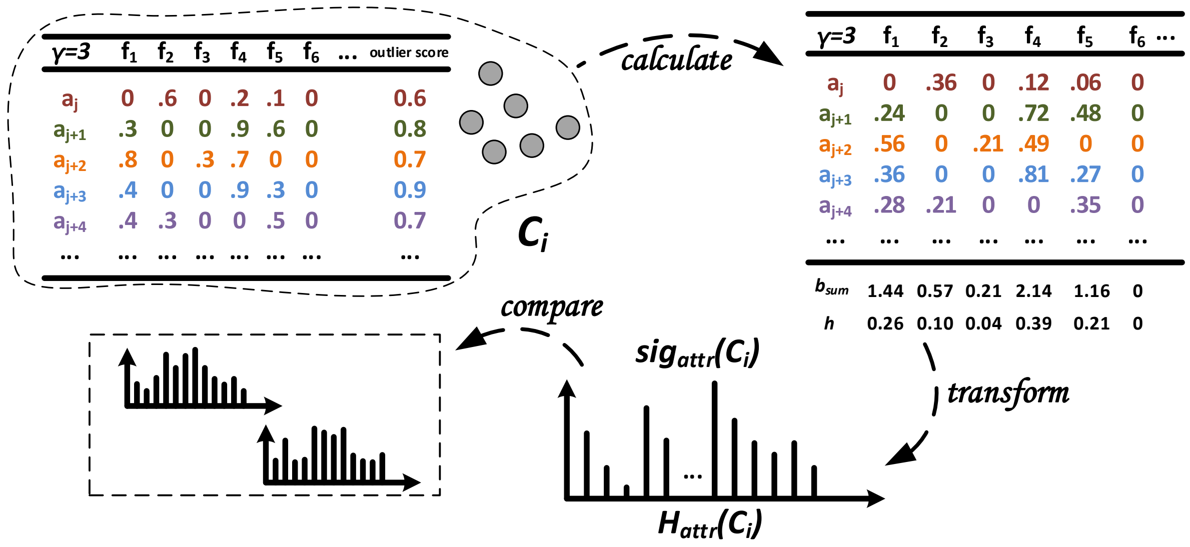
Considering that for different attack scenarios certain features are more distinctive, we take advantage of a different approach than the graph-based one for sig_{com} in order to *calculate* and *compare* signatures sig_{attr}. In general, we *transform* feature importance information into a histogram $H_{attr}(C_i)$, describing sig_{attr}, that characterizes the potential attack scenario clustered in C_i.

In the exmaple in Figure 5.5, each alert $\boldsymbol{a}_j$ of the cluster C_i provides feature importance score values for the top-γ-features of the set $\mathcal{F}$ consisting of d features (score values of other features are set to zero) and the corresponding penalized meta-alert outlier score $\tilde{f}_{j(meta)}^{(penalized)}$. Each feature f_i of $\mathcal{F}$ represents a histogram bin (bucket) and the count (frequency) of this bin is computed by adding up each score per feature $s_f^{(i)}$ weighted with the outlier score provided by each alert in the cluster. In the example, this leads to $b_{sum}^{(i)} = \sum_j s_{f_i}^{(j)} \tilde{f}_{j(meta)}^{(penalized)}$. Finally, we compute the relative frequency h of each bin by $h^{(f_i)} = \frac{b_{sum}^{(f_i)}}{\sum_i^d b_{sum}^{(f_i)}}$.

In order to *compare* two signatures, one can take advantage of methods computing the similarity between two statistical distributions which are represented by any two histograms $H_{attr}(C_i)$ and $H_{attr}(C_j)$ obtained from clusters C_i and C_j. Although the two-sampled Kolmogorov-Smirnov test can be modified to function on discrete data, as it applies for binned values in the histogram, it is normally used for continuous data [362]. Two of standard choices in the discrete case are the well-known chi-squared test and the Bhattacharyya distance measure [363]. Here we apply the Bhattacharyya distance $D_B(H_i, H_j)$ between the histograms $H_i = \{h_i^{(b)}\}_{b=1,...,B}$ and $H_j = \{h_j^{(b)}\}_{b=1,...,B}$ for B equi-width bins b with (relative) frequency h which is defined as given in Equation 5.5. It takes values in the range of $(0 \leq D_B \leq \infty)$. BC is the Bhattacharyya Coefficient (Equation 5.5) for which $0 \leq BC \leq 1$ applies. We obtain identical histograms and thus identical signatures sig_{attr} between two attack scenarios if $D_B == 0$ applies. The higher the D_B, the more different the signatures are.

$$D_B(H_i, H_j) = -ln(BC(H_i, H_j)); \quad BC(H_i, H_j) = \sum_{b=1}^{B} \sqrt{h_i^{(b)} h_j^{(b)}} \tag{5.5}$$

It is noted that the applied OD algorithms must be capable of providing the top-γ-features. Although only a limited amount of work is able to satisfy this requirement, it is an important functionality for the design of future anomaly-based IDS [364]. The higher the value for γ, the more information must be transmitted with each alert increasing the communication overhead. Therefore, a decent value for γ could provide a tradeoff between a meaningful signature and a reasonable communication overhead not stressing resources. Furthermore, in order to be able to accumulate a knowledge base of signatures sig_{attr}, they all must have been created by the same feature set $\mathcal{F}$, such that each feature within $\mathcal{F}$ represents the exact same bin in sig_{attr}. However, in many applications, OD algorithms are utilized on the same pre-processed data by systems such as Argus or Bro-IDS as utilized in [335] or CICFlowMeter-V3 as used in [303]. Nevertheless, a knowledge base should be built up providing signatures for a multitude of commonly applied feature sets such that the user can choose the set, which suits the application best. To what extent signatures obtained from a different amount of γ features but from the same feature set $\mathcal{F}$ can be compared, must be evaluated in future work.



γ=3	f1	f2	f3	f4	f5	f6	...	outlier score
aj	0	.6	0	.2	.1	0		0.6
aj+1	.3	0	0	.9	.6	0		0.8
aj+2	.8	0	.3	.7	0	0		0.7
aj+3	.4	0	0	.9	.3	0		0.9
aj+4	.4	.3	0	0	.5	0		0.7
...	...	...	...	...	...	...	...	...

γ=3	f1	f2	f3	f4	f5	f6 ...
aj	0	.36	0	.12	.06	0
aj+1	.24	0	0	.72	.48	0
aj+2	.56	0	.21	.49	0	0
aj+3	.36	0	0	.81	.27	0
aj+4	.28	.21	0	0	.35	0
...	...	...	...	...	...	...
b_{sum}	1.44	0.57	0.21	2.14	1.16	0
h	0.26	0.10	0.04	0.39	0.21	0

Figure 5.5: Generation and comparison process of the signature sig_{attr}.

Generation and Comparison of sig_{temp}

For generating a signature that represents the temporal (timing) characteristic of an attack, called sig_{temp}, we take advantage of the same procedure as for sig_{attr}. The two metrics, the duration of an attack and its number of respective alerts (with respect to Table 5.2), extracted from a saturated cluster could be utilized to compare a potential novel attack with a knowledge base consisting of signatures for different attack scenarios with an averaged duration per attack and number of events. However, averaged values do not ideally represent the ground truth. Therefore, a more fine-grained way is to characterize an attack by computing the difference in time, Δt, in between the temporally ordered alerts (events).

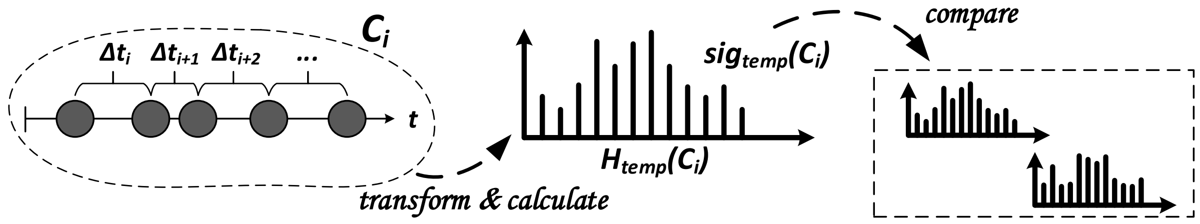


Figure 5.6: Generation and comparison process of the signature sig_{temp}.

With respect to Figure 5.6, we *transform* the Δt information of alerts from cluster C_i into a histogram with a fixed amount n_bins of equal-width bins b, denoted as $H_{temp}(C_i)$

which describes the signature $sig_{temp}(C_i)$. The histogram ranges from 0 to a subtle maximum value, mostly representing attack scenarios best, and the bin-width can be computed dividing the maximum value by the number of bins. If some outlying Δts occur, they are added to the last bin. In order to preserve information about the number of instances, a frequency histogram is proposed, otherwise, one might *calculate* a relative frequency histogram. One might further differentiate between histogram types such as *short*, *mid* or *long* to adjust the number and width of the bins, depending on the order of magnitude of Δts. To *compare* two signatures (histograms) of the same type, we again leverage the Bhattacharyya distance measure as discussed with sig_{attr}.

It is noted that the correctness of sig_{temp} critically depends on the correct order of alerts and assumes no strong variation in delays of alerts or a high number of FPs or FNs. However, if the notion of time of IDS sensors is not correctly loosely synchronized or an adversary may tamper with the time, imprecise temporal characteristics of alerts may cause incorrect or confusing results for sig_{temp}. We leave the discussion of this phenomenon as future work.

Handling of the Signatures

Having a certain set of reference scenarios, denoted in this work as *Knowledge Base*, one can identify novel attack scenarios in the *Runtime* phase by comparing the obtained signatures sig_{com}, sig_{attr} and sig_{temp} with existing ones utilizing the *Scenario Comparison* module. Signatures known from misuse-based IDS, anti-virus software or anti-malware systems are either one- or two-dimensional, as Blacklists or Whitelists are examples of the former and regular-expression functions is an example for the latter. A more recent method, multi-dimensional signatures can be seen as ML models that create a multi-dimensional model of input data applying mathematical techniques and score or classify observations. However, formats such as STIX are not compatible with those technologies as each class of threat classification may be founded on completely different trained models [365]. Thus, similar to the idea of conventional signatures, we transfer the idea of having such one-dimensional characteristics to anomaly-based ML systems. What is more, the signatures proposed, sig_{com}, sig_{attr} and sig_{temp}, must not necessarily match the exact signatures from the reference scenarios since the comparison measures can help to identify the similarity of novel patterns to existing ones from the *Knowledge Base*. Over time, the set of reference scenarios might grow very strongly. Thus, it might be time consuming to compare a novel attack pattern composed of the three signatures with each one in the *Knowledge Base*. Thus, the *Unsupervised Mining* module makes it possible to apply a hierarchical clustering in the *Learning* phase which clusters similar unknown attack pattern before presenting them to the *Human Analyst*. Hierarchical clustering was also proposed by [54] but only for similar sig_{com}. Having a tree-like structure within the *Knowledge Base*, attack scenarios characterized by the 3-tuple of signatures can be structured into clusters according to their similarities. For each of the hierarchical clusters, the signatures that represent the most each sub-cluster can be determined to compare novel attack pattern much faster by only comparing it with the representing signatures $sig_{com}^{(Ref)}$, $sig_{attr}^{(Ref)}$ and $sig_{temp}^{(Ref)}$. The overall best signature match can be obtained by $max(sig_{com}, sig_{com}^{(Ref)}) \wedge min(sig_{attr}, sig_{attr}^{(Ref)}) \wedge min(sig_{temp}, sig_{temp}^{(Ref)})$. Since none of the signatures contain privacy-relevant information, they can be enriched with additional attack information and, e.g., shared among companies using STIX in an automated manner [366].

5.3 Experimental Evaluation

5.3.1 Methodology & Settings

With respect to Figure 5.1, we split our evaluation of SOAAPR into two parts. Firstly (i), the *Streaming Alert Correlation / Clustering* module along with *Trigger Signature Generation* and *Discard Alerts & Clusters* is evaluated. Secondly (ii), the *Signature Generation* as well as the *Scenario Comparison* capability of SOAAPR is evaluated. For (i), the traffic labeled CSV files only of CICIDS2017 are parsed to derive the ideal number of instances per attack scenario serving as the ground truth clusters. Then, we iterate over the data sets and simulate an anomaly-based IDS by generating alerts which are fed into SOAAPR. Our proposed *Streaming Alert Correlation / Clustering* is used beside the competitor GAC to evaluate and compare their clustering capability. Since GAC does not rely on any intrusion type attribute for clustering, it could be leveraged for anomaly-based IDS and its outcome can be used to generate motif-signatures sig_{com} by design. Furthermore, its chunk processing could be regarded as a trivial form of online processing. For those reasons, we see a comparison of SOAAPR with GAC as reasonable.

For (ii), we generate signatures sig_{com} as part of the *Signature Generation* module leveraging ideally clustered attack scenarios from CICIDS2017 and compare them (*Scenario Comparison* module) in order to show their applicability for attack characterization based on OD. In order to derive signatures sig_{attr}, the supervised RF algorithm is used on a selection of attack scenarios from CICIDS2017 and CSE-CIC-IDS2018 each extracted into a separate CSV file. The reason for choosing the supervised RF algorithm instead of an online unsupervised OD variant is the more reliable feature importance scoring functionality providing better interpretability for sig_{attr} evaluation, which can be obtained by the SHAP method. Another benefit of the supervised approach is that the outlier percentages do not influence the feature importance scoring functionality. Therefore, we could extract arbitrary attack scenarios, even those that are not characterized with a low anomaly percentage. Thus, evaluations of unsupervised online OD methods along with the *Alert Preparation* module are part of future work. We then produce clusters from alerts utilizing the top-γ-features and the outlier scores per instance obtained by multiple RF-instances (*single system - multiple algorithms*) in order to generate and compare signatures sig_{attr}. For the third signature sig_{temp}, we compute the Δts from each ideal attack cluster within CICIDS2017 and CSE-CIC-IDS2018 to generate and compare signatures.

Experiments were conducted on a virtualized Ubuntu 20.04.1 LTS equipped with 12 Intel(R) Xeon(R) CPU E5-2430 at 2.20 GHz and 32 GB memory running on a Proxmox server environment. Programs are coded in Python 3.9 using the latest PyCharm 2021.1.3 environment. GAC for graph representation and clustering as well as sig_{com} for graph representation and motif comparison utilizes the *igraph*[1] and *networkx*[2] libraries. RF is taken from *sklearn*[3], SHAP from its respective library[4] and SOAAPR's sig_{attr} as well as sig_{temp} generation rely on histograms generated with the popular *numpy*[5] library. Across all clustering evaluations, GAC is configured with the default hyperparameters proposed by [337] with a clique size of 15, a similarity threshold between alerts of 0.25 and a chunk size of 5000 subsequent alerts since higher numbers of alerts increase the graph size and

[1]`https://igraph.org/python/` (accessed on 25 June 2021)
[2]`https://github.com/networkx` (accessed on 25 June 2021)
[3]`https://scikit-learn.org` (accessed on 25 June 2021)
[4]`https://pypi.org/project/shap/` (accessed on 25 June 2021)
[5]`https://numpy.org/` (accessed on 25 June 2021)

its computational complexity heavily. For equal comparison with SOAAPR, we did not take advantage of parallel processing. SOAAPR's hyperparameters are min_sim, t_{ttl}, min_alerts and the frequency exceeding threshold k. Although SOAAPR allows six comparison functions with respect to Table 5.3, to ensure equal conditions in comparing with GAC, only f_1, f_2, f_4 and f_5 have been used. Unless otherwise stated, SOAAPR's streaming clustering parameters are set to $min\_sim = 0.25$, $t_{ttl} = 1day$, $min\_alerts = 8$ and $k = 50$. The hyperparameters provided decent results across all data sets and attack scenarios. The minimum similarity min_sim is set equally to GAC and the total time to live t_{ttl} is set to one day since none of the attack scenarios present in CICIDS2017 and CSE-CIC-IDS2018 exceeds this boundary and we are able to capture all alerts. The minimum number of alerts is set to 8 to capture attack scenarios with even a low number of associated alerts, e.g., Heartbleed in CICIDS2017. The frequency exceeding threshold is set very high in order to capture fluctuations of Δts throughout all attack scenarios which is also amplified by the poor time stamping quality present in CICIDS2017 and CSE-CIC-IDS2018.

5.3.2 Data Source

For the evaluation of AC, much of the work - even recent works [367, 343, 368] - applies the outdated DARPA 2000 having two scenarios LLDOS 1.0 and LLDOS 2.0.2 based on the output of the (misuse-based) ISS RealSecure IDS due to the lack of labeled alert data or attack data that can be used for the purpose of attack detection. Haas et al. use, apart from their synthetically generated alert data set, a real-world data set from the Internet Storm Center[6] operating a platform called DShield for sharing data from security devices to evaluate their GAC clustering approach. However, this data source lacks ground truth information such that generated clusters by GAC or our SOAAPR approach cannot be evaluated for their accuracy. Since we are anyway interested in the complete processing pipeline starting from the detection of outliers in $\boldsymbol{X}_t$ by the online OD algorithms, we set our focus on recent IDS data sets such as CICIDS2017 [303] and CSE-CIC-IDS2018[7] [303] provided by the University of New Brunswick on AWS, or UNSW-NB15 since long-serving and still widely used data sets, such as KDD Cup 99[8] or NSL-KDD[9], have been criticized by many researchers over the past couple of years [303, 313]. Especially for the evaluation of anomaly-based IDS methods, the latest updated data sets like CSE-CIC-IDS2018 should be utilized [369].

Although CSE-CIC-IDS2018 is tailored for the evaluation of anomaly detection and consists of seven attack categories with a total of 14 different types of intrusions, e.g., SSH-BruteForce or DDoS-LOIC-UDP, it only provides statistical traffic features obtained by CICFlowMeter-V3 saved as a CSV file. However, in order to generate meaningful alerts for SOAAPR, the typical TCP/IP level network traffic header features, IP-address and port number, are mandatory. Furthermore, in some cases timestamps from data record $n + 1$ is older than the timestamp from data record n. Therefore, sorting the data set according to its timestamp feature is necessary in order to preserve the chronological sequence of events. UNSW-NB15 provides IP and port feature information and has a huge variety of attacks and subtypes of attacks, but the corresponding ground truth CSV file

[6]SANS Technology Institute, Internet Storm Center, `https://isc.sans.edu/` (accessed on 25 June 2021)

[7]License: `https://registry.opendata.aws/cse-cic-ids2018/` (accessed on 25 June 2021)

[8]`http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html` (accessed on 25 June 2021)

[9]`https://www.unb.ca/cic/datasets/nsl.html` (accessed on 25 June 2021)

cannot be properly used to map attack scenarios to each anomalous instance due to the unclear start and ending timestamps of events and cannot assign clear temporal ordered timestamps to the data set records. Although the ML CSV files from CICIDS2017 also does not feature IP and port information, the additional available labeled traffic CSV files do, and these can be used to gather the mandatory information since both CSV files have the same timestamps and number of records. However, there are two drawbacks with CICIDS2017. Firstly, each of the 14 attack scenarios, with respect to Table 5.2, only occur once, which hampers the signature comparison of similar attack scenarios. Secondly, the units of the timestamp are only provided in minutes. Since CSE-CIC-IDS2018 provides similar attack scenarios, e.g., two *Brute Force - Web* attacks, and timestamps are provided in a more fine-granular fashion, it is at least utilized for measurements where no IP and port information is required, such as for sig_{attr} and sig_{temp} evaluation. Thus, although no accurate timestamping is provided, we ambitiously tried to compare attack scenarios using different data sets of CICIDS2017 and CSE-CIC-IDS2018. In order to encounter the timestamp challenge in CICIDS2017, synthetic timestamps are inserted for data instances from an attack scenario assigned the same timestamp in minutes. However, in a real-world scenario, we might be able to assign more fine-grained units such as milliseconds. In order to be able to generate meaningful temporal signatures, we introduce smaller units by assuming equal Δt of alerts having timestamps from the same minute. We take into account blurred signatures for sig_{temp} on CICIDS2017 but see the potential of this fingerprinting using the more accurate results of CSE-CIC-IDS2018. Since our aim is to mine information from the outcome of OD algorithms, for the evaluation of SOAAPR's and GAC's clustering, we excluded the DoS Hulk attack scenario in the CICIDS2017 *Wednesday-WorkingHours* data set and completely neglected the *Friday-WorkingHours-Afternoon* data sets due to the high amount of outlier percentage with respect to Table 5.2. The streaming clustering of SOAAPR depends on timely sorted alerts, which is why all alerts in CICIDS2017 have been sorted.

5.3.3 Evaluation Criteria

In terms of evaluation metrics for AC, we rely on four different metrics. Two have been proposed by Ning et al. [370] called *completeness* and *soundness*. The former, also known as the true detection rate, is denoted as $COMP$ and calculated by the ratio of the number of correctly correlated alerts (CCA) divided by the number of related alerts RA, i.e., the real attack-scenario-related number of instances, which states the ground truth of each attack scenario (Equation 5.6). The latter, denoted as $SOUND$, is the ratio of CCA and correlated alerts (CA), i.e., all the clustered alerts in C_i (Equation 5.7). A further metric - the Jaccard index - will provide the similarity of the ideal cluster (ideal attack scenario) and C_i obtained by the AC system. It compares two sets of elements, in this case the ideal cluster and the obtained cluster, and provides information about which alerts are shared between the two sets and which are distinct. It is denoted as JAC and computed by Equation 5.8. It takes values in the range of $[0, 1]$ and yields a higher value the more similar two sets are. Further metrics including the compression rate, which might be interesting for systems afflicted with a high number of FP are available in [35].

$$COMP = \frac{\#CCA}{\#RA} \tag{5.6}$$

$$SOUND = \frac{\#CCA}{\#CA} \tag{5.7}$$

$$JAC = \frac{|RA \cap CA|}{|RA \cup CA|} \tag{5.8}$$

Furthermore, we measure the average runtime for GAC and SOAAPR's streaming clustering as a representative metric for computational performance. Thus we accumulated the elapsed time for processing each data set until the final clusters are obtained.

In terms of attack characterization, for each signature sig_{com}, sig_{attr} and sig_{temp} per attack scenario, we compute the similarity values in between each attack scenario with respect to the formulas given in Section 5.2.4. Furthermore, we measure the average runtime to generate each signature in order to compare the computational performance of sig_{com}, sig_{attr} and sig_{temp} of varying cluster sizes.

5.4 Discussion of Results

5.4.1 SOAAPR Clustering

Table 5.4 summarizes the clustering results of SOAAPR and GAC for the 11 selected attack scenarios of the CICIDS2017 data sets. It shows how many clusters were generated by each algorithm and how many clusters are assigned to each attack scenario. Furthermore, the number of associated alerts contained in the assigned clusters is provided together with the total number of ideal alerts that represents each attack scenario. With respect to Equations 5.6, 5.7 and 5.8, metrics are provided for the clustering performance and the overall elapsed time to cluster the attack scenarios for each data set.

Table 5.4: Clustering performance results of SOAAPR and GAC on 11 attack scenarios of the CICIDS2017 data sets (No. Clusters - assigned cluster(s) / generated cluster(s); No. Alerts - assigned alerts / actual number of alerts per attack scenario; Metrics - $COMP/SOUND/JAC$; Time denotes the total amount of processing time for each data set containing the respective attack scenarios).

Attack Scenario	**SOAAPR**				**GAC**			
	No. Clusters	**No. Alerts**	**Metrics**	**Time (sec)**	**No. Clusters**	**No. Alerts**	**Metrics**	**Time (sec)**
FTP-Patator	1/2	7,938/7,938	1.0/1.0/1.0	105.11	$2/3^1$	10,000/7,938	1.0/1.0/0.92	1,424.17
SSH-Patator	1/2	5,897/5,897	1.0/1.0/1.0		$2/3^1$	8,835/5,897	1.0/1.0/0.88	
WA - Brute Force	1/24	1,507/1,507	1.0/1.0/1.0	2.32	1/1	2,180/2,180	1.0/0.69/0.69	266.56
WA - XSS	21/24	652/652	1.0/1.0/1.0		1/1	2,180/2,180	1.0/0.30/0.30	
WA - Sql Injection	2/24	16/21	0.76/1.0/0.76		1/1	2,180/2,180	1.0/0.01/0.01	
DoS GoldenEye	$2/4^2$	4,065/10,293	0.39/1.0/0.39	63.62	$3/5^1$	11,588/10,293	1.0/1.0/0.96	12,107.17
DoS Slowloris	$1/4^2$	3,588/5,796	0.62/1.0/0.62		$2/5^1$	10,000/5,796	1.0/1.0/0.94	
DoS Slowhttptest	$1/4^2$	5,501/5,499	$1.0/1.0/1.0^3$		$2/5^1$	10,000/5,499	1.0/1.0/0.81	
Heartbleed	$0/4^2$	0/11	0.0/0.0/0.0		$0/5^1$	0/11	0.0/0.0/0.0	
Infiltration	1/1	36/36	1.0/1.0/1.0	~0.0	1/1	36/36	1.0/1.0/1.0	~0.0
Bot	2/2	1,926/1,966	0.98/1.0/0.98	2.51	2/2	1,962/1,966	$1.0/1.0/1.0^3$	75.09

[1] Due to "out of memory" error using 32 GB RAM, the alerts had to be processed in chunks of 5,000 as proposed in [337], [2] Changed hyperparameter set, [3] rounded values

SOAAPR perfectly clusters both brute force scenarios, *FTP-Patator* and *SSH-Patator*, each in a separate cluster while only needing approximately 105 sec. For GAC's graph-based clustering, in contrast, a high amount of alerts needed to be processed in chunks since 32 GB RAM on the evaluation machine were not enough. Nevertheless, the processing time was significantly higher by a factor of approximately 13. Three clusters have been generated by GAC which split each attack scenario into two halves. Thus, both attacks have been assigned to two clusters each. Due to the process of splitting, GAC even misses some alerts for both scenarios which yielded a JAC value lower than 1. For the sake of visualization, the sunburst diagram in Figure 5.7a is given showing GAC's clustering results.

GAC clustered the 3 web attack scenarios into one cluster completely since the alert attributes are highly similar and it cannot differentiate them in a timely way. In contrast, SOAAPR did generate 24 clusters and captures the *Brute Force* scenario into one cluster completely while splitting the *Sql Injection* into two clusters and *XSS* into 21 as depicted in the sunburst diagram of Figure 5.7b. The splitting of *XSS* is due to the fact that the first min_alerts of the attack scenario are timestamped in the same minute, thus, a meaningful frequency exceeding factor k cannot be built. Furthermore, all attack scenarios have similar alert attributes such that decreasing t_{ttl} would split the scenarios in a better manner. Setting $t_{ttl} = 45$ min (duration of *Brute Force* attack scenario) and increasing $min\_alerts = 500$ resulted in 2 clusters while capturing the *Brute Force* in one cluster with $COMP = SOUND = JAC = 1.0$ and *XSS* into the other with $COMP = 1.0$, $SOUND = 0.97$ and $JAC = 0.97$.

With the default hyperparameter setting, SOAAPR generates only one cluster for the DoS-like attacks and Heartbleed scenario. This is due to the fact that all of those attack scenarios have similar alert attributes. Slightly increasing the min_sim parameter to 0.5 instead of 0.25 generates 3 clusters in which all DoS-attacks are grouped together in one cluster and the Heartbleed attack is perfectly assigned into another cluster with $COMP = SOUND = JAC = 1.0$. Both hyperparameter settings result in an approximate processing time of 500 sec. However, Table 5.4 shows the results of a hyperparameter set adapted to capture each DoS-attack. It was already mentioned that in real-world applications it might be feasible to apply different GAC instances, each parameterized for mining a dedicated attack scenario. Thus, we decreased t_{ttl} to one hour, and set the min_alerts to 2,000 (DoS-attacks are typically characterized by a high number of events) and the frequency exceeding factor k to 300, since the Δts are typically quite low for DoS-attacks, in order to be sure that each attack scenario is completed in a timely manner. Since t_{ttl} was decreased, SOAAPR's processing time was significantly reduced to approximately 60 sec because it can discard clusters after t_{ttl} was exceeded, which is - in contrast to GAC, with over 3 h processing time - an impressive speed-up. GAC again needed to process the attack scenarios in chunks, which resulted in 5 total clusters. For better illustration, the clustering results of the DoS-attacks and Heartbleed utilizing GAC (c) and SOAAPR (b) is depicted as sunburst diagrams in Figure 5.7.

In terms of the *Infiltration* and *Bot* attack scenario, both algorithms performed almost similarly. Especially, with the low amount of alerts for *Infiltration*, GAC could compete with our approach. However, with the 1,966 alerts of the Bot attack, GAC's processing time is a factor of 30 higher. Just decreasing the frequency exceeding factor k for the *Infiltration* data set, SOAAPR generates 2 or 3 clusters instead of one, which better represents the ground truth of 3 individual infiltration attacks instead of the labeled single one referring to the footnote in Table 5.2.

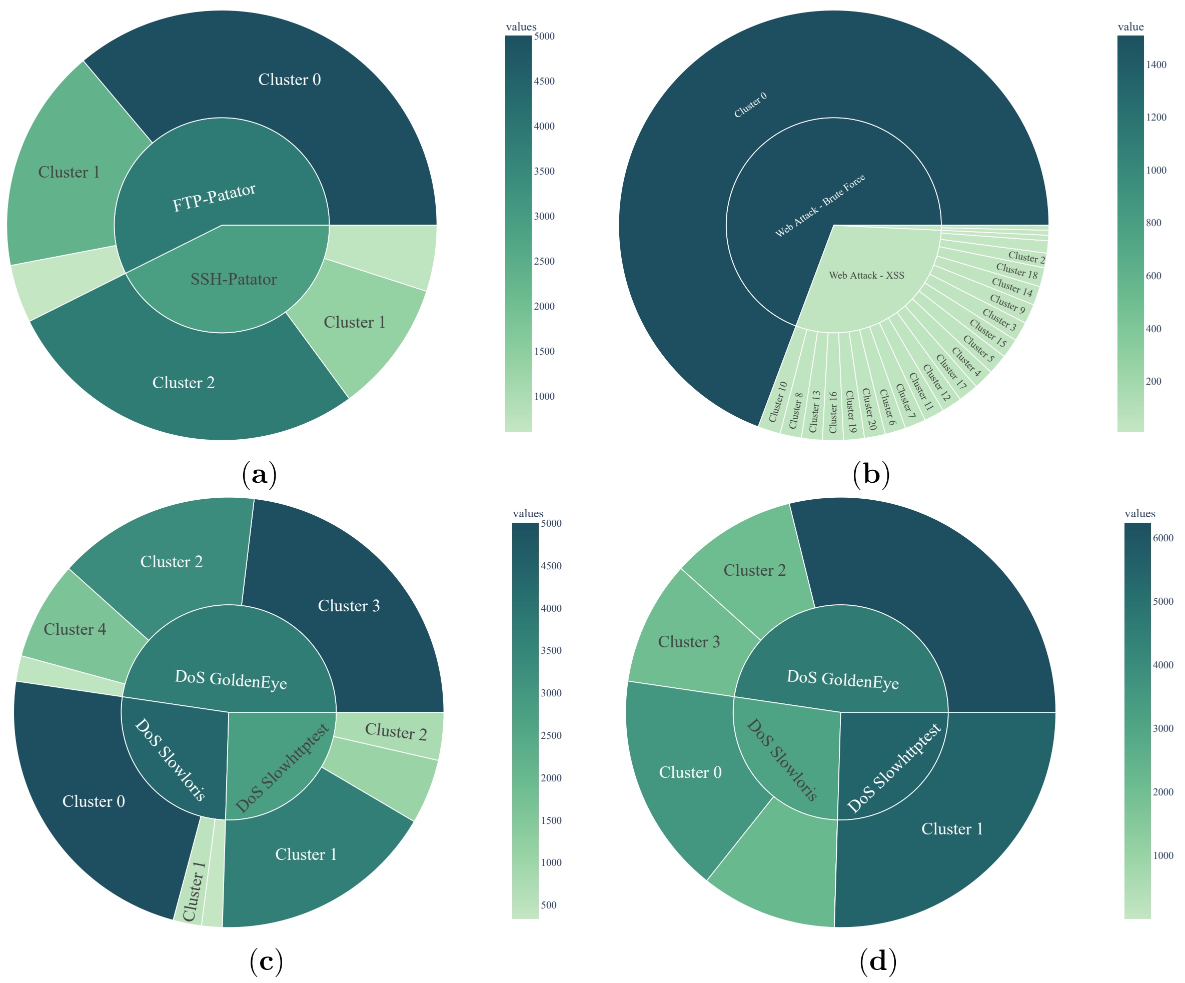


Figure 5.7: Sunburst diagrams for the clustering performance of (a) GAC on FTP and SSH Brute Force, (b) SOAAPR on the web attacks *Brute Force* (1 cluster), *XSS* (21 clusters) and *Sql Injection* (2 clusters), (c) GAC and (d) SOAAPR on the DoS-attacks.

As of now, only ideal alerts (only TPs) have been considered and fed into the AC methods. However, in real-world applications, AC has to deal with FPs and FNs. In preliminary measurements, we introduced a certain confidence level that steers the interspersion of FPs and FNs with a certain probability. From the binary classification result providing the amount of TPs, TNs, FPs, and FNs, we can derive the so-called $F1$-score as the harmonic mean of precision and sensitivity by $F1 = \frac{TP}{TP+\frac{1}{2}\times(FP+FN)}$ which is often used as a metric on imbalanced data [34]. The effects of FPs and FNs on the clustering result are exemplarily discussed for the *Bot* attack scenario for which both, SOAAPR and GAC, achieved good results and the number of alerts is more meaningful compared to *Infiltration*. Introducing approximately 190 FPs and FNs yields a $F1 = 0.90$ and SOAAPR, as well as GAC, perfectly capture alerts into two clusters as with only TPs (Table 5.4) but both generate additional clusters. Those are denoted as ghost-clusters (SOAAPR - 3, GAC - 2) and are mainly caused by the FPs. The FNs only reduce the amount of alerts inside the clusters and can be seen as less critical since the consecutive signature generation will also work if the cluster still contains a majority of representative alerts characterizing the attack. Injecting a higher number of FPs (approximately 1900) and FNs (approximately 400), yielding $F1 = 0.58$, will again cause the generation of the two *Bot*-related clusters (with the reduced amount of FN-alerts) but is associated with a significantly higher number of ghost-clusters (SOAAPR - 40, GAC - 28). However, SOAAPR can easily be adjusted to deal with those ghost-clusters, which in total have an average size of 40 alerts by increasing the *min_alerts* parameter to 100. Then, SOAAPR reduces the number of ghost-clusters to 2 while still capturing the 2 *Bot*-related clusters only taking approximately 4.6 sec. GAC, in contrast, has no possibility of reducing ghost-clusters and takes approximately 79 sec.

5.4.2 SOAAPR Signaturing

For each attack scenario in either CICIDS2017 (sig_{com}, sig_{attr} and sig_{temp}) or CICIDS2017 and CSE-CIC-IDS2018 (sig_{attr} and sig_{temp}) signatures are derived and the similarity in between each attack scenario's signature is computed. As a result, we obtained an upper triangular matrix of similarity values from which we applied hierarchical/agglomerative clustering using the average-linkage method yielding a dendrogram for the sake of better visualization. As of now, we reserve evaluations showing the effects of FPs and FNs on signature comparison for future work and only consider ideally clustered attack scenarios.

sig_{com}

Table 5.5 provides the sig_{com} similarity values between the 11 evaluated CICIDS2017 attack scenarios. It can clearly be seen from the table that attack scenarios that share the same typical communication relation between attacker and victim have high similarity. For instance, the brute force *FTP-/SSH-Patator* attacks have a strong correlation in terms of sig_{com} with a value of 0.9905 in the range of $[0, 1]$ since a single attacker IP with varying source ports attacks a single victim IP and a dedicated port (either FTP - 21 or SSH - 22). In contrast, *Heartbleed*, characterized by a single attacker IP from a single port attacking a single victim IP on a single port, differs significantly from all other attack scenarios and only shares a value of approximately 0.3 in similarity with the FTP and SSH brute force attacks. All three DoS-like attacks share a similar communication pattern having a similarity value of above 0.97 amongst each other.

Table 5.5: Upper triangular matrix of sig_{com} similarities between CICIDS2017 attack scenarios (WA - Web Attack).

	Bot	**Infiltration**	**WA – Brute Force**	**WA – XSS**	**WA – Sql Injection**	**FTP-Patator**	**SSH-Patator**	**DoS slowloris**	**DoS Slow-httptest**	**DoS Golden-Eye**	**Heartbleed**
Bot	-	0.7903	0.7477	0.7742	0.8068	0.6861	0.6955	0.6874	0.6868	0.6595	0.3085
Infiltration		-	0.8610	0.8731	0.9120	0.8210	0.8279	0.8220	0.8216	0.8006	0.3517
WA - Brute Force			-	0.9734	0.9251	0.9382	0.9477	0.9395	0.9389	0.9115	0.2908
WA - XSS				-	0.9448	0.9115	0.9210	0.9128	0.9123	0.8849	0.2870
WA - Sql Injection					-	0.8698	0.8787	0.8711	0.8705	0.8446	0.2807
FTP-Patator						-	0.9905	0.9987	0.9993	0.9734	0.3049
SSH-Patator							-	0.9918	0.9912	0.9639	0.3021
DoS slowloris								-	0.9994	0.9721	0.3045
DoS Slowhttptest									-	0.9727	0.3047
DoS GoldenEye										-	0.3134
Heartbleed											-

To guide the reader better, Figure 5.8 visualizes the results of Table 5.5 in a dendrogram. Attack scenarios sharing high similarity are clustered together, such as in the DoS-attacks or the *FTP-/SSH-Patator* attack scenarios. The web attacks, *Brute Force*, *XSS* and *Sql Injection* are extremely similar as well.

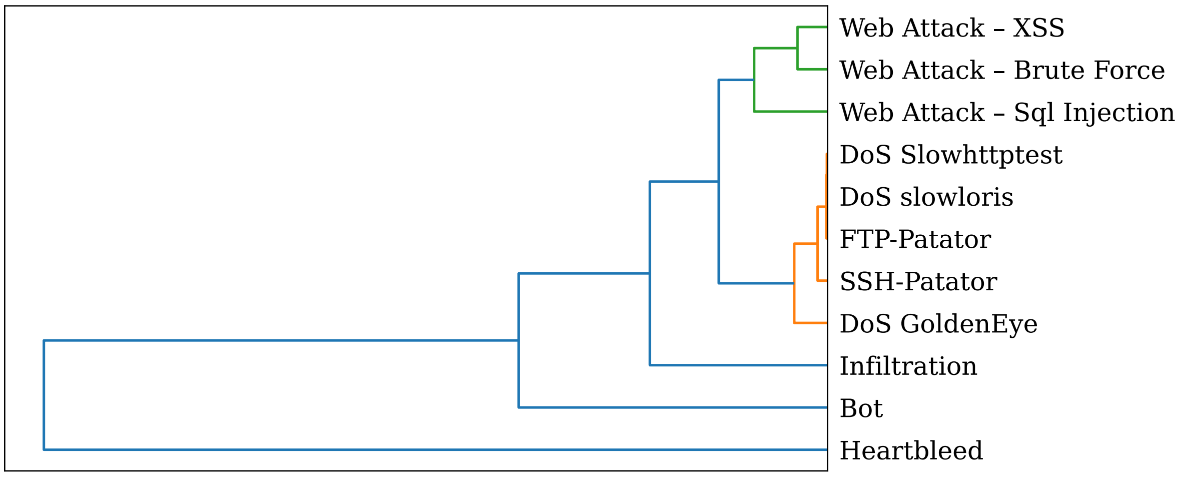


Figure 5.8: Hierarchical clustering of similarities between CICIDS2017 attack scenarios based on sig_{com}.

Most of the attack scenarios share a strongly similar communication relation with values approximately above 0.7 with respect to Table 5.5. In most of the attack scenarios, the attacker and victim share an oto-relation in terms of IP and an mto-relation (referring to [337]) in terms of port information. The main difference is the amount of varying source ports which is higher the more alerts are present. Thus, DoS-like attacks having a larger amount of varying source ports leads to more nodes within the G_{com} graph and thus a slightly less similar communication relation as for instance with web attack scenarios. In contrast, *Bot* and *Infiltration* differ from the rest of the attack scenarios and *Heartbleed*, especially, is the only attack scenario that is characterized by a oto communication relation in terms of IP and port information, thus being the least similar to all others.

One of the intentions of introducing a motif-approach by Haas et al. in [54] was to provide a most fine-grained attack characterization as possible with GAC only identifying four pre-defined cluster classes (oto, otm, mto, mtm) [337]. Although containing slightly more information within the graph-structure by indirectly incorporating the amount of varying IP or port information in the form of nodes in the graph, as assumed, sig_{com} alone does not characterize individual attack scenarios in a more fine-grained way. Excluding the privacy-preservation capability of the motif-approach, one might even leverage the pre-defined cluster classes and count the frequency of unique IP or port data values to obtain similar information gain as with sig_{com}, which - especially with a higher number of clustered alerts - might be far less computationally complex than the graph-based approach. The higher the number, the longer it takes to generate each attack's signature. Whereas *Infiltration* with 36 alerts takes only about 0.02 sec, *Bot* with approximately 2,000 alerts takes around 10 sec. *FTP-/SSH-Patator* brute force clusters with around 14,000 alerts take around 3 min and the three DoS-attacks including *Heartbleed*, with respect to Table 5.2, take approximately 10 min to be generated. The processing time for sig_{com} shows exponential behavior over the number of alerts.

sig_{attr}
In order for the RF classifier to generate feature importance scores, each attack scenario has been extracted into a separate CSV file only containing benign and no malicious data. Since DoS-like, Portscan or Brute Force attacks are typically not the scope of OD algorithms, we limit ourselves to the web attack (Brute Force, XSS, Sql Injection), Infiltration and Bot scenarios of CICIDS2017 (refer to Table 5.2) and CSE-CIC-IDS2018 (refer to Table 5.6).

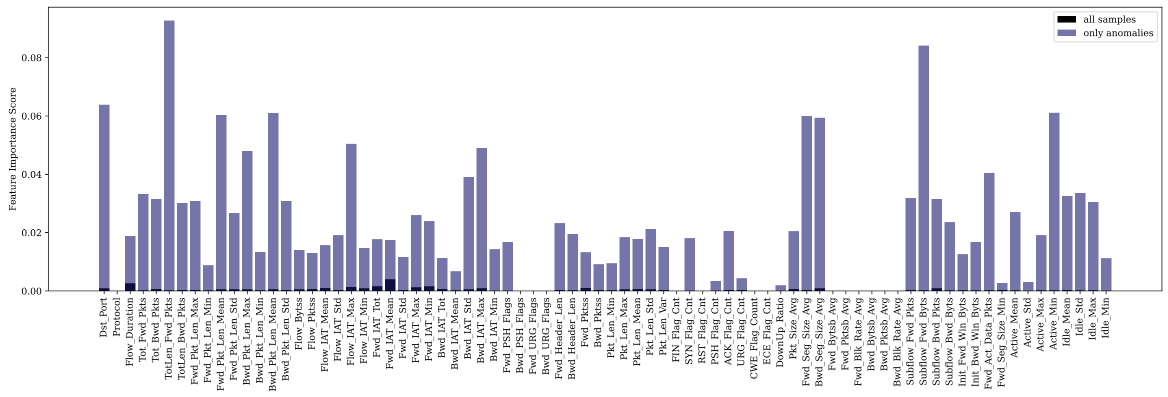
Table 5.6: A selection of attack scenarios with respective characteristics of the CSE-CIC-IDS2018 data set.

Data set	**Attack Type**	**# Instances**	**Outliers (%)**	**Duration (min)**
Thursday-22-02	Brute-Force-Web-0	250	0.023	56.01
	Brute-Force-XSS-0	81	0.008	0.90
	SQL-Injection-0	31	0.003	13.35
Friday-23-02	Brute-Force-Web-1	363	0.035	48.85
	Brute-Force-XSS-1	151	0.014	69.02
	SQL-Injection-1	50	0.005	0.97
Wednesday-28-02	Infiltration-1	42,760	6.974	22.25
	Infiltration-0	26,111	4.259	6.00
Thursday-01-03	Infiltration-3	54,311	16.403	96.98
	Infiltration-2	38,752	11.704	57.98
Friday-02-03	Bot-0	190,240	18.143	473.27
	Bot-1	95,951	9.151	89.82

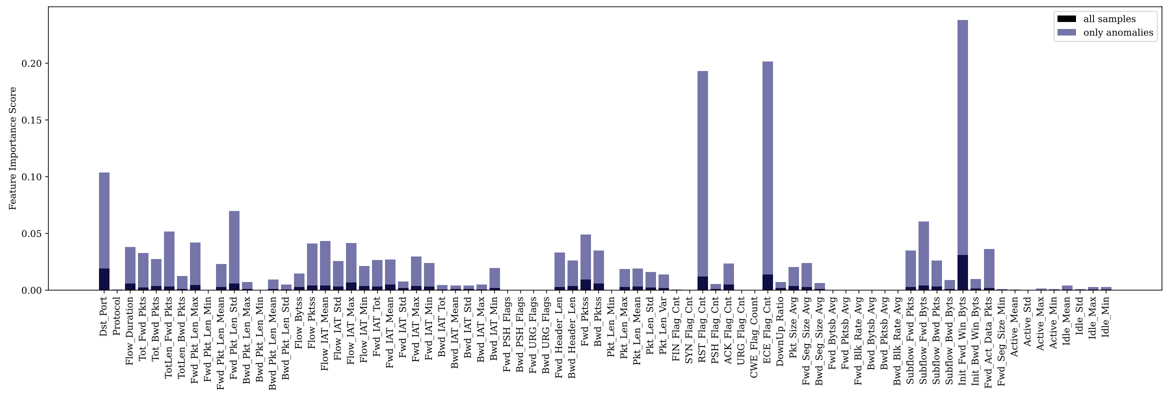
Since founded on the assumption that the feature importance is a representative characteristic for an attack scenario, as discussed by the authors in [303], two exemplary signatures sig_{attr} generated by RF's feature importance scoring applied on both data sources CICIDS2017 and CSE-CIC-IDS2018 are provided in Figure 5.9. In order to compare attack scenarios from both data sets, the same set of 78 features had to be applied. With respect to Subfigure 5.9a, the *TotLen_ Fwd_ Pkts* and *Subflow_ Fwd_ Byts* are the most important features, which is also stated in [303] for the *Infiltration* attack scenario. For the web attack *Brute Force* scenario, the most important feature is the *Init_ Fwd_ Win_ Byts*. Since [303] only provides feature importance for web attacks in general and not individually as done in this work, the other two highly important features *RST_ Flag_ Cnt* and *ECE_ Flag_ Cnt* (Subfigure 5.9b) seem characteristic for this sub-attack category.

Choosing an appropriate γ is not only crucial for the amount of information an alert has to carry, but also affects the similarity between the sig_{attr} of two attack scenarios. Therefore, we have computed similarity in the form of the Bhattacharyya distance over the top-γ-features between each two attack scenarios. Figure 5.10 shows the results of this measurement respectively for the two strongly similar attack scenarios of the CSE-CIC-IDS2018 data set *SQL-Injection-0* and *SQL-Injection-1*, as well as the highly dissimilar scenarios of *SQL-Injection-0* from CSE-CIC-IDS2018 and *Bot* from CSECIC2017.

From those measurements we obtain multiple insights on γ's effects. Firstly, with low γ values, attack scenarios are less similar in general but dissimilar attack scenarios have a higher magnitude for the Bhattacharyya distance. For instance, the Bhattacharyya distance for low γ in Subfigure 5.10a is approximately 0.5 and for the dissimilar attack scenarios in Subfigure 5.10b approximately 1.2. Secondly, the more similar two arbitrary attack scenarios are, the more sharply the curve will fall to lower similarity values. Thirdly, all attack scenarios reach a stationary point for the Bhattacharyya distance with



(**a**)

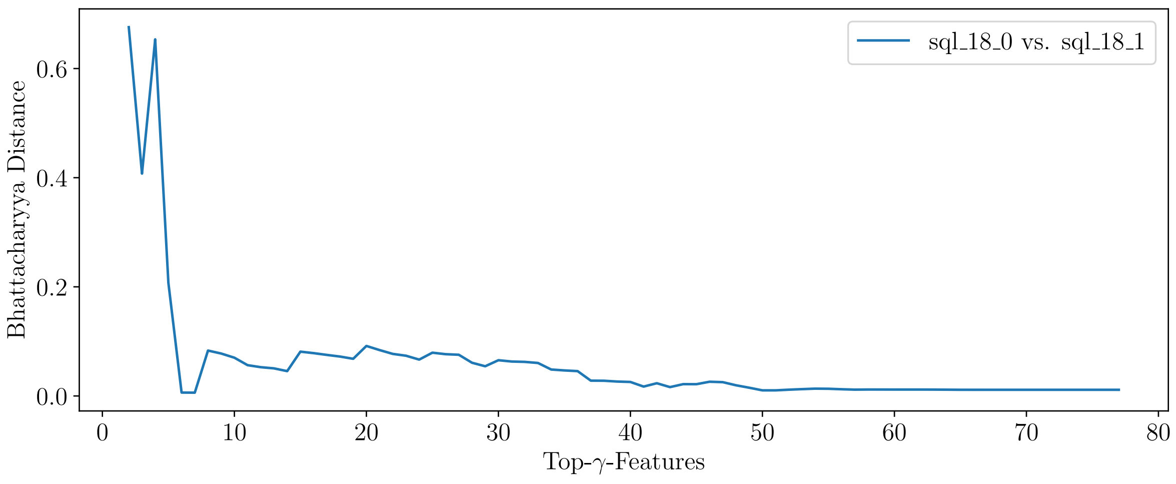


(**b**)

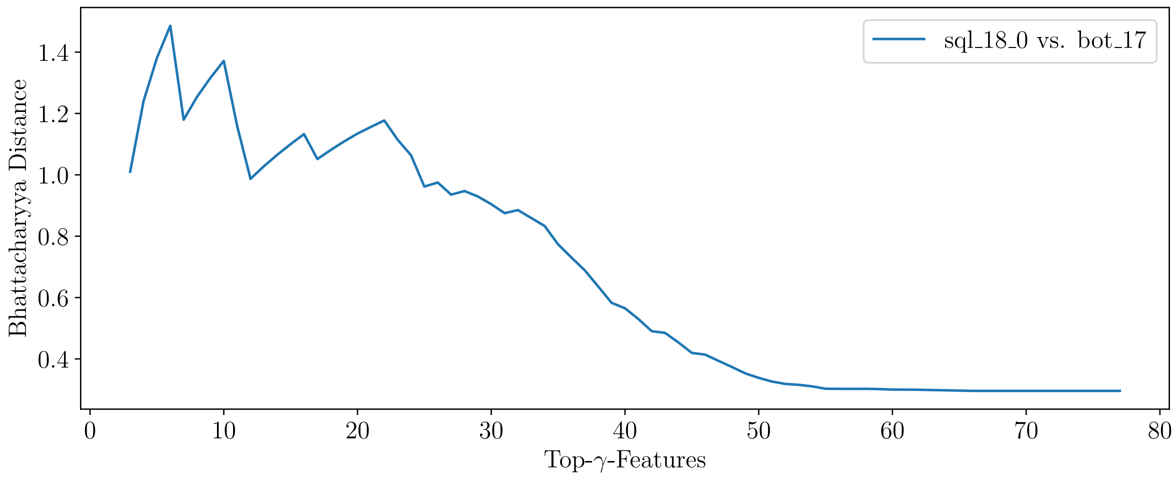
Figure 5.9: Two exemplary sig_{attr} using the same set of 78 features for the attack scenarios (a) *Infiltration* of CICIDS2017 and (b) *Brute-Force-Web-0* in CSE-CIC-IDS2018 (two colors for feature importance on all samples and only on anomalous samples - sig_{attr}).

a certain amount of features whereby increasing γ does not affect the similarity. In turn, this also means that a certain amount of information to be transferred with alerts is enough, and more information does not improve the result. However, this also means that the higher the γ, the more decisive the magnitude of the stationary point of the Bhattacharyya distance is in distinguishing between any two attack scenarios and a clear differentiation deteriorates. The reason behind this is that a mainly limited amount of features is representative for an attack scenario. Thus, if using more than these, the importance of the representative features is negatively influenced by the less important ones, meaning that feature importance of benign data takes over. Therefore, fourthly, a decent range for γ considering the feature set used in our measurements is approximately from 10 to 50, whereas if γ is set closer to 10, it will benefit highly from similar attacks such as a *XSS* attack with another *XSS* attack, while γ closer to 50 will benefit attack scenarios of the same category such as web attacks in general.

Figure 5.11 depicts the hierarchically clustered results of attack scenario similarity between a selection of CICIDS2017 and CSE-CIC-IDS2018 attacks based on sig_{attr} with γ set to 30. Related attack scenarios such as *xss_18_0* and *xss_18_1*, *sql_18_0* and *sql_18_1*, *brute_force_18_0* and *brute_force_18_1*, *bot_18_0* and *bot_18_1* or the Infiltration attacks from CSE-CIC-IDS2018 are highly similar.



(**a**)



(**b**)

Figure 5.10: Dependency of γ on the Bhattacharyya distance (similarity) of two highly similar (a) and dissimilar (b) attack scenarios.

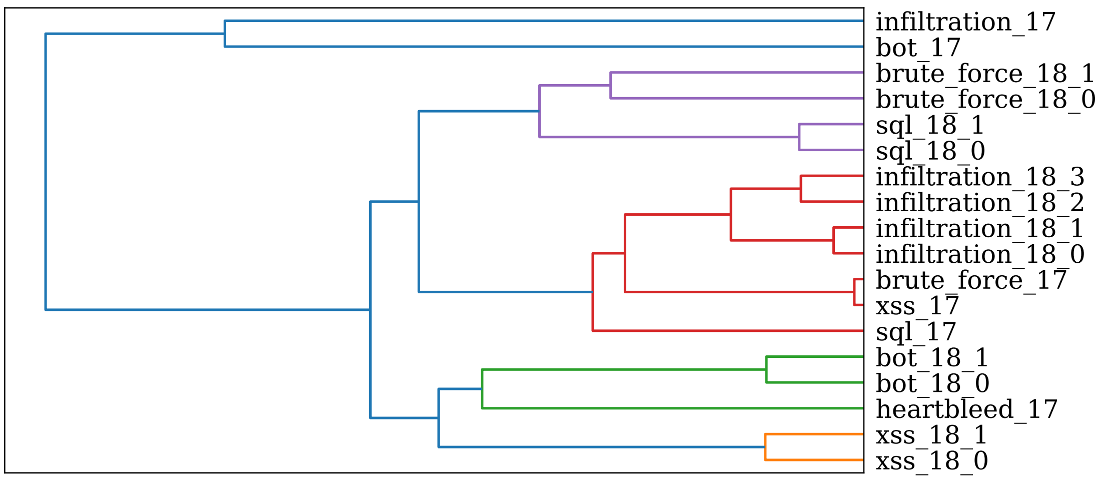
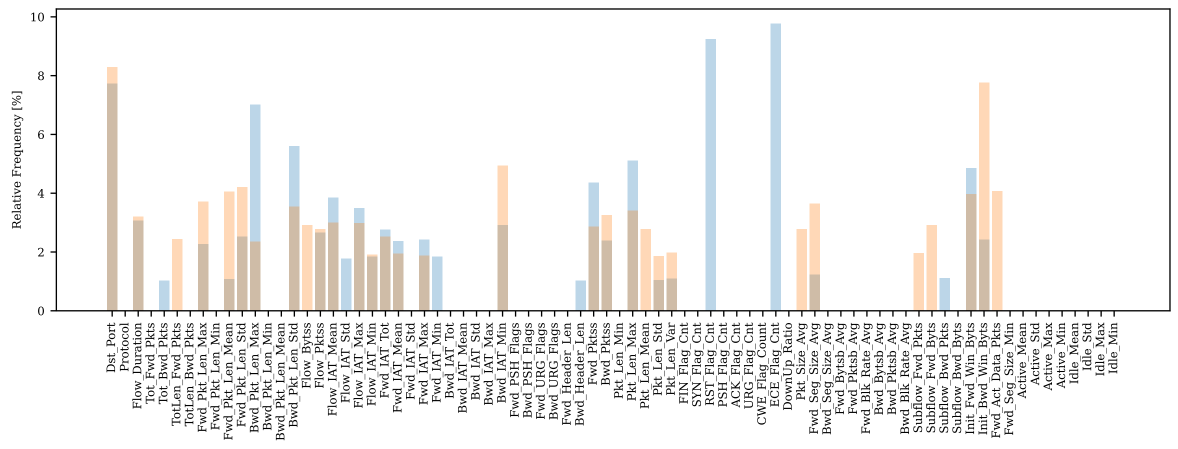
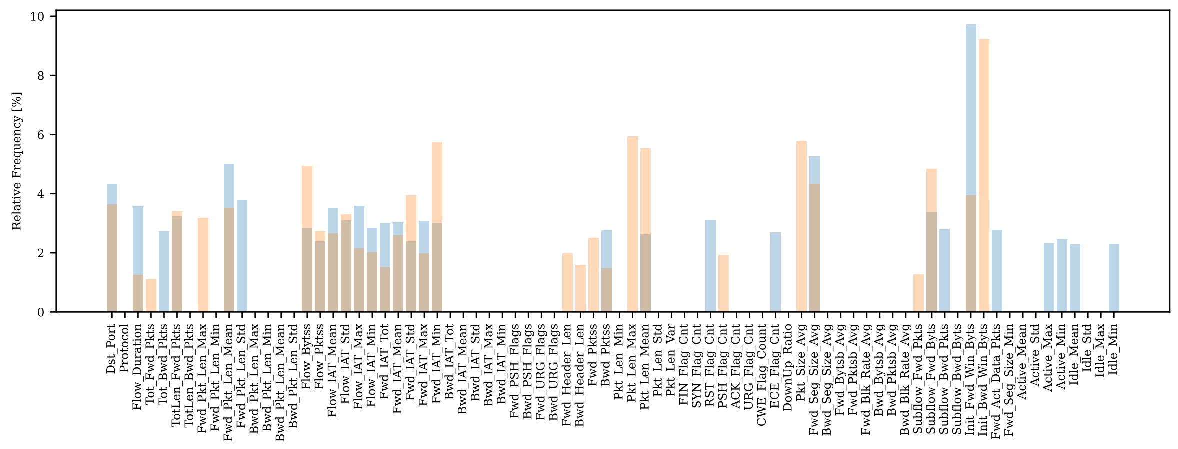


Figure 5.11: Hierarchical clustering of similarities between a selection of CICIDS2017 and CSE-CIC-IDS2018 attack scenarios based on sig_{attr} for $\gamma = 30$.

In general, strong similarity between similar CICIDS2017 and CSE-CIC-IDS2018 attack scenarios such as *infiltration_17* and *infiltration_18_X* is not provided. Taking a look at the histogram comparison of two dissimilar examples from both data sets, *sql_18_0* and *sql_17* (Subfigure 5.12a) as well as *brute_force_18_1* and *brute_force_17* (Subfigure 5.12b), reveals, that despite some feature similarity, both are afflicted with a high number of irrelevant features for the comparison. For instance, the two most important features of *sql_18_0*, *RST_Flag_Cnt* and *ECE_Flag_Cnt*, are completely irrelevant to *sql_17* despite the rest of the features showing similarity. Equally, with respect to Subfigure 5.12b and the importance of *Init_Fwd_Win_Byts* and *Init_Bwd_Win_Byts* for web attacks [303], both attack scenarios show strong feature importance for only one of those two features. Furthermore, for each attack, some features are more important for one attack, while they are completely irrelevant for others, such as *Active_Max/Min/Mean* for *brute_force_17*.



(**a**)



(**b**)

Figure 5.12: Comparison of sig_{attr} for (a) SQL-Injection-0 of CSE-CIC-IDS2018 (blue) with Web Attack - Sql Injection of CICIDS2017 (orange) and (b) Brute-Force-Web-1 of CSE-CIC-IDS2018 (blue) with Web Attack - Brute Force of CICIDS2017 (orange) for $\gamma = 30$.

The infiltration-attacks between CICIDS2017 and CSE-CIC-IDS2018 can hardly be compared since the infiltration scenarios in CSE-CIC-IDS2018 include the portscanning attack steps which are not considered within CICIDS2017. This is also confirmed by the significantly higher number of alerts associated with the 2018 infiltration scenarios reasoned by the portscan conducted after the attacker successfully gained access to the victim machine. Although similar CICIDS2017 and CSE-CIC-IDS2018 attack scenarios seem to have a weak similarity, the curvature characteristics from the γ-dependency-curves reveal better insights. Thus, with respect to Figure 5.10, we applied Gaussian Filtering to the curves in order to smooth them and better visualize their curvature. In Figure 5.13, a sample selection of smoothed γ-dependency-curves for similar and dissimilar attack scenarios from CICIDS2017 and CSE-CIC-IDS2018 is given. Taking into account measurement deviations, curves from similar attack scenarios, *sql_18_0* and *sql_18_1*, and even *sql_18_0/1* and *sql_17*, show monotonic decreasing behavior, while the curves of dissimilar attack scenarios, such as all Sql-Injection-ones in Figure 5.13 with *bot_17*, contain concave sections. Although not clustered together in the dendrogram, similarity for similar 2017 and 2018 attack scenarios can be derived from the curvature characteristic. Nevertheless, there might be various reasons for the poor result, such as poor quality of the data sets, e.g., by a slight difference in feature generation using an older version of CICFlowMeter for CICIDS2017 or due to slightly different attacks used. Those could mimic changes in similar attack campaigns as time passes, since, in real-world scenarios, adversaries change their strategy as well. However, further evaluation is necessary in future work.

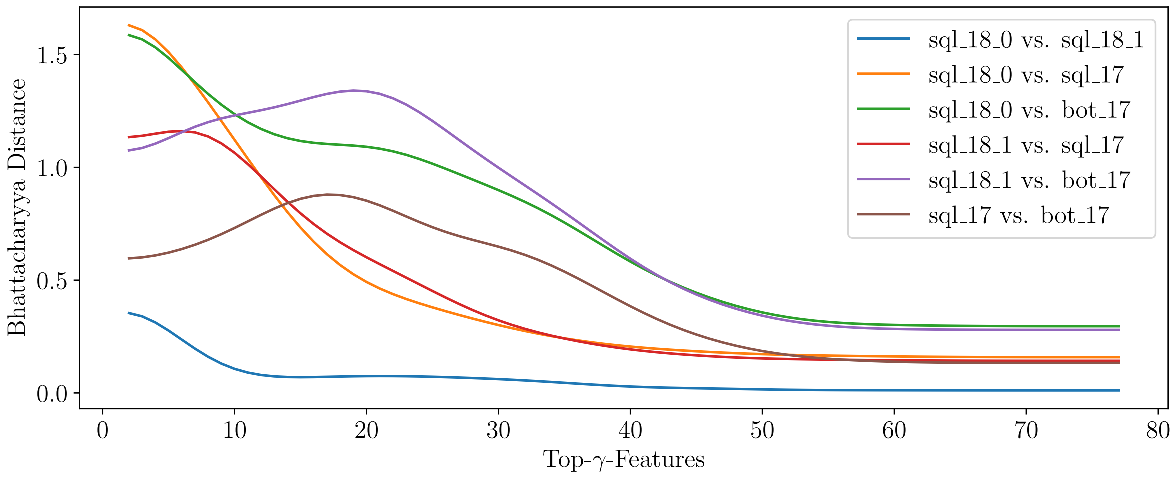


Figure 5.13: Smoothed γ-dependency-curves using Gaussian Filtering for a selection of similar and dissimilar attack scenarios.

The results of the average processing times to generate sig_{attr} for the evaluated attack scenarios is provided in Table 5.7. In contrast to the exponentially increasing processing time of sig_{com}, the generation time for sig_{attr} shows linear behavior with the number of alerts inside a cluster. The mean time per alert in our evaluation is approximately 7 µsec.

Table 5.7: Average processing time to generate sig_{attr} for a selection of attack scenarios of CICIDS2017 and CSE-CIC-IDS2018.

	brute_force_18_0	**brute_force_18_1**	**brute_force_17**	**sql_18_0**	**sql_18_1**	**sql_17**	**xss_18_0**	**xss_18_1**	**xss_17**	**bot_18_0**	**bot_18_1**	**bot_17**	**infiltration_18_0**	**infiltration_18_1**	**infiltration_18_2**	**infiltration_18_3**	**infiltration_17**	**heartbleed_17**
No. Alerts	250	363	1507	31	50	21	81	151	652	190240	95951	1966	42760	26111	38752	54311	36	308
Time (msec)	1.23	3.08	6.99	0.34	0.62	0.21	0.62	1.03	3.08	988.69	492.8	9.05	220.39	135.9	199.42	282.69	0.41	3.08

sig_{temp}
In order to find a histogram setting that satisfies all attack scenarios best, we computed the descriptive statistics maximum, minimum, mean, and median values of Δts for each attack scenario in CICIDS2017 and CSE-CIC-IDS2018. Either because the timestamping in CICIDS2017 is inaccurate, or because of the misleading labeling, whereby multiple attack scenarios of the same category, happening at different times, are combined together, e.g., for *Infiltration_17* and *PortScan_17*, we do not take into account the maximum Δt values to find a decent maximum bin parameter for the sig_{temp} histogram. Excluding the statistical strays in the maximum values for the CICIDS2017 attack scenarios, *Bot_17*, *DoS_slowloris_17*, *Infiltration_17* and *PortScan_17*, a decent maximum bin value for the histogram is 100 s (mean of residing values). Also *Heartbleed_17* is not a very representative attack scenario since it consists of 11 alarms likely reasoned in an attacker script that sends 11 Heartbeat messages at an interval of exactly 120 sec to exploit the vulnerability. In a different scenario, an attacker might modify this frequency resulting in another Δt statistic. Referring to minimum, mean and median values a decent fine-grained value for the bin width is 0.1 sec (median of Δts' median).

It is again noted that in real-world applications one might apply different histogram types as proposed in Section 5.2.4 such as *short*, *mid* or *long* to adjust the number and width of the bins depending on the order of magnitude of Δts. Thus, for instance, DoS-like attacks with lower mean and median Δt values than other attack types could be characterized as *short* attacks in advance with respect to the extremely low Δt values before generating signatures with significantly smaller values for the maximum bin value and bin width.

Figure 5.14 depicts the hierarchically clustered results of attack scenario similarity between CICIDS2017 and CSE-CIC-IDS2018 attacks based on sig_{temp} with the mentioned setting of maximum bin of 100 sec and a bin width of 0.1 sec. It can clearly be seen that similar attack scenarios such as DoS-like attacks, infiltration, brute force or web attacks are clustered together due to their high level of similarity. For instance, DoS-like attacks like *DoS_Hulk_18* and *DDOS_HOIC_18* as well as *DoS_Hulk_17* and *DDoS_17*, which are heavy DoS-attacks, are clustered together. Even the same DoS-attacks but from different data sets, *DoS GoldenEye_17* with *DoS-Goldeneye_18* and the less intensive *DoS slowloris_17* and *DoS-Slowloris_18*, show high similarity.

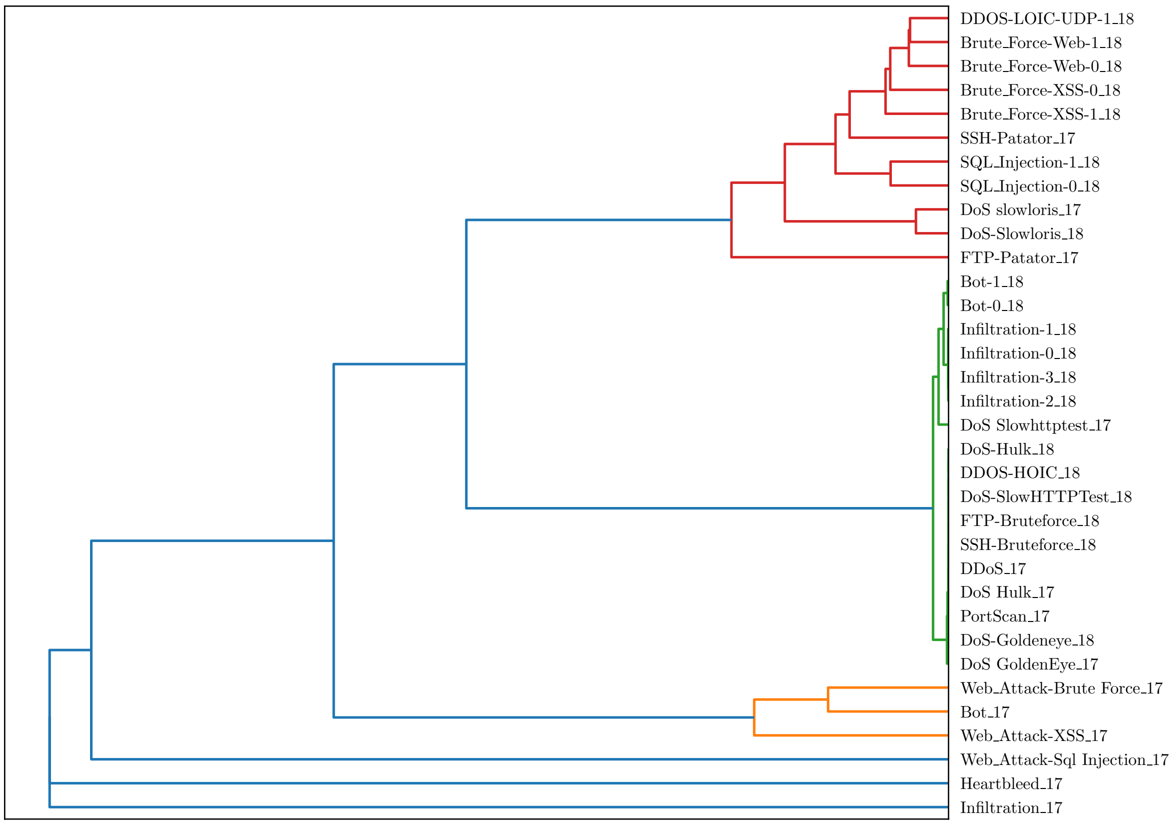


Figure 5.14: Hierarchical clustering of similarities between CICIDS2017 and CSE-CIC-IDS2018 attack scenarios based on sig_{temp}.

Notably, the same attack scenarios that were used multiple times in CSE-CIC-IDS2018 are in most cases clustered as significantly similar, such as all four Infiltration attacks, *Bot-0_18* and *Bot-1_18*, and the web attacks *SQL_Injection-0_18* and *SQL_Injection-1_18* or *Brute_Force-Web-0_18* and *Brute_Force-Web-1_18* as well as *Brute_Force-XSS-0_18* and *Brute_Force-XSS-1_18*.

Figure 5.15 demonstrates the high level of similarity leveraging sig_{temp} between the four timely different and unrelated infiltration attack scenarios taken from CSE-CIC-IDS2018. For the sake of visualization and with respect to the descriptive statistics maximum, minimum, mean, and median for those attack scenarios, we set the histogram values to a maximum bin value of 1 sec and the bin width to 0.01 sec.

Although not clustered in Figure 5.14, similarity can be seen between comparable attack scenarios taken from different data sets of CICIDS2017 and CSE-CIC-IDS2018 when adapting the histogram settings. This is shown with the *XSS* and *Brute Force* web attacks in Figure 5.16. The histogram is set to a maximum bin value of 30 sec and a bin width of 1 sec. However, the less fine-grained binning of the CICIDS2017 attacks can clearly be seen, which is reasoned in the way timestamps were generated as explained in Section 5.3.2.

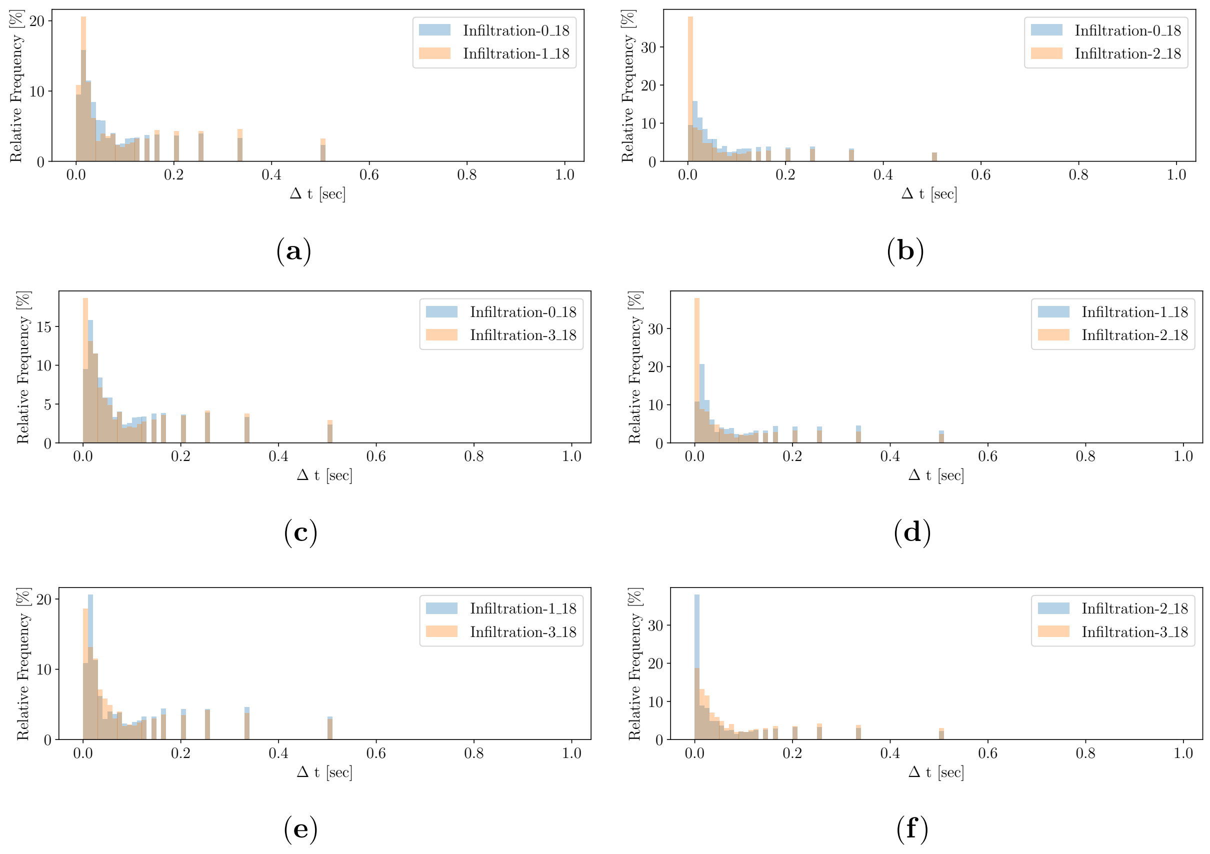


Figure 5.15: Comparison of Δt histograms of four distinct *Infiltration* attack scenarios (a)-(f) present in CSE-CIC-IDS2018.

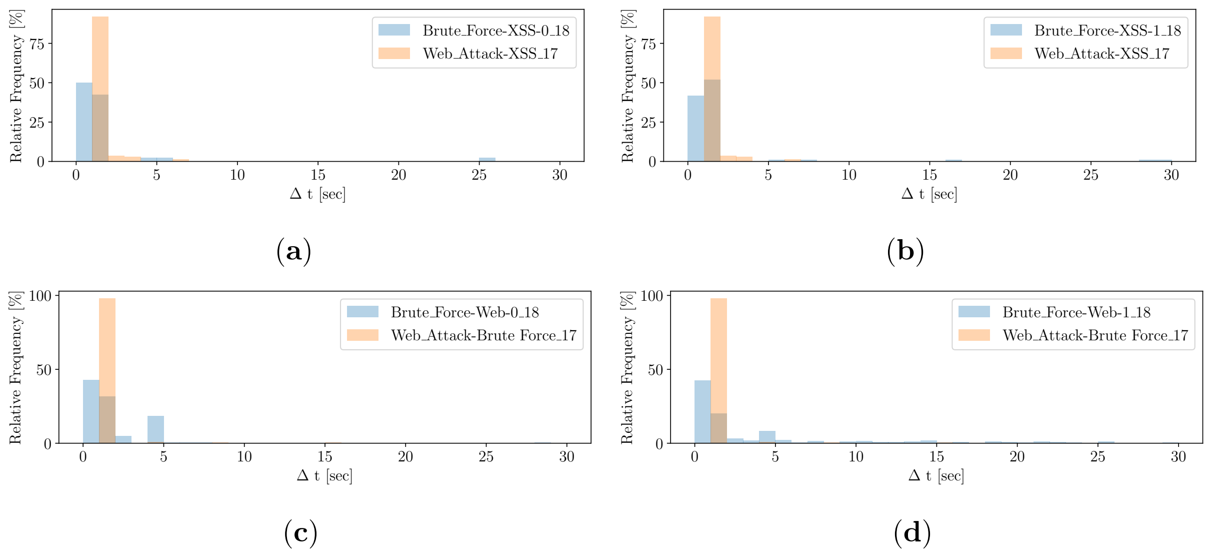


Figure 5.16: Comparison of Δt histograms of four distinct web attack scenarios - *XSS* (a)-(b) and *Brute Force* (c)-(d) - present in CICIDS2017 and CSE-CIC-IDS2018.

In contrast to the average processing times for sig_{attr}, the processing of sig_{temp} is split into two parts - the generation of Δts based on timestamps of the timely sorted alerts of a cluster and the generation of the histogram, the actual sig_{temp}. The result of the average processing times for the attack scenarios of CICIDS2017 is provided with the same histogram setting of a maximum bin value of 100 sec and bin width with 0.1 sec in Table 5.8 and for CSE-CIC-IDS2018 in Table 5.9. As with sig_{attr}, the processing times of sig_{temp} shows linear behavior with a mean time per alert for Δts generation of approximately 200 µsec and for histogram generation of approximately 4.5 µsec per alert. The latter depends on the histogram setting and increases to a value of approximately 138 µsec per alert for a maximum bin value of 550 sec and bin width with 0.01 sec.

Table 5.8: Average processing time to generate sig_{temp} for a selection of attack scenarios of CICIDS2017.

	brute_force_17	**xss_17**	**sql_17**	**infiltration_17**	**ddos_17**	**dos_hulk_17**	**dos_slowloris_17**	**dos_slowhttptest_17**	**dos_goldeneye_17**	**port_scan_17**	**ftp-patator_17**	**ssh-patator_17**	**bot_17**	**heartbleed_17**
No. Alerts	1,507	652	21	36	128,027	231,073	5,796	5,499	10,293	158,930	7,938	5,897	1,966	11
Generating Δts (sec)	0.1873	0.0810	0.0030	0.0053	15.6347	28.8301	0.7102	0.6742	1.2875	19.6687	0.9818	0.7314	2.4444	0.0018
Generating Histograms (msec)	0.69	0.58	0.51	0.54	13.78	24.49	1.14	1.10	1.57	17.06	1.38	1.18	0.76	0.57

Table 5.9: Average processing time to generate sig_{temp} for a selection of attack scenarios of CSE-CIC-IDS2018.

	brute_force_18_0	**brute_force_18_1**	**sql_18_0**	**sql_18_1**	**xss_18_0**	**xss_18_1**	**bot_18_0**	**bot_18_1**	**infiltration_18_0**	**infiltration_18_1**	**infiltration_18_2**	**infiltration_18_3**	**ddos-hoic_18**	**dos-hulk_18**	**dos-slowhttptest_18**	**dos-slowloris_18**	**ddos-loic-udp_18**	**dos-goldeneye_18**	**ftp-bruteforce_18**	**ssh-bruteforce_18**
No. Alerts	363	250	50	31	81	151	190,240	95,951	42,760	26,111	38,752	54,311	686,012	461,912	139,890	10,990	1,730	41,508	193,360	187,589
Generating Δts (sec)	0.0484	0.0701	0.0088	0.0142	0.0214	0.0400	49.1716	24.7964	10.4209	6.3606	9.4362	13.2450	170.4973	104.9426	23.1637	1.8217	0.3151	6.5640	29.9785	38.7183
Generating Histograms (msec)	0.58	0.61	0.56	0.56	0.57	0.59	21.56	11.06	5.52	3.54	4.91	6.77	75.33	51.48	15.60	1.14	0.76	5.27	21.35	20.94

6 A Resource-Preserving Self-Regulating Uncoupled MAC Algorithm to be Applied in Incident Detection

This chapter provides details on the so-called ***Uncoupled M**essage **A**uthentication **C**ode* (**Uncoupled MAC**) algorithm and is organized as follows - Section 6.1 provides an overview on incident detection mechanisms ranging from adaptive IDSs to the exploitation of the nature of cryptographic mechanisms in order to detect security-relevant incidents. Furthermore sampling systems are presented including Uncoupled MAC. Improvements and extensions of the original Uncoupled MAC algorithm are presented in Section 6.2. The novel self-regulating sampling approach based on the detection of Uncoupled MAC violations is described in Section 6.3. In Section 6.4 details on the implementation and evaluation environment are presented, attack scenarios described and evaluation metrics provided. Results of several measurements mainly showing the advantage of the self-regulation compared to the non-self-regulated version of Uncoupled MAC by also discussing the trade-off between detection capability and resource utilization are given in Section 6.5.

6.1 Incident Detection Mechanisms

6.1.1 Adaptive Intrusion Detection Systems

The term adaptive for IDS in the context of this work describes techniques of sampling and self-regulation. To cope with the increasing amount of traffic within networks while reducing large memory and CPU processing requirements, sampling turned out to be a promising scalable data aggregation technology for IDSs since the processing capacity of such systems are typically much smaller than the amount of data to be inspected. Because sampled traffic is an incomplete approximation of the actual one, multiple mechanisms for sampling data exist.

Sampling

In [371] a difference between packet- and flow-based sampling, crucial for the working of network-based IDS, together with deterministic and non-deterministic methods is made. Packet-based sampling is simple to implement with low CPU power and memory requirements but is inaccurate for inference of flow statistics like size distribution of the original flows. In contrast, flow-based sampling overcomes the limitations of packet-based sampling but suffers from prohibitive memory and CPU power requirements and is still too complex to implement [372]. The flow-size based sampling technique in [371] assumes that network attacks usually use small flows as traffic source. With the proposed selective sampling strategy such flows are sampled with a constant probability. Other related work evaluated that packet sampling has a negative impact on the efficiency of anomaly-based IDSs increasing the false positives but performs best when using a random flow sampling strategy. However, it is possible to maintain a high level of security while selectively inspecting packets with a minimal amount of processing overhead. An analytic and statistical model for the process of network intrusions has been introduced in [373] supporting the experimental results of [371] demonstrating that it is sufficient to inspect only a small number of sampled packets. In [374] a packet- and time-driven traffic sampling strategy for an IDS in a SDN is proposed that fully utilizes the inspection capability of malicious traffic, while maintaining the total aggregate volume of the sampled traffic below the inspection processing capacity of the IDS. The packet-driven approach inspects a packet every $1/x$ packets for a sampling rate x and the time-driven inspects all the packets within a time window of sampling rate x each sampling interval. The time-driven mechanism has the advantage of detecting stateful attacks because it captures all the packets for a certain time duration. However, if packets are mainly sent event-triggered, the time-driven approach is not feasible since there could be phases of sampling in which no packets are inspected. This could easily happen in networks with high fluctuations of the bandwidth. If an intruder is able to compromise the IDS or might know the sampling rate, he could exploit this knowledge by performing malicious activities outside the sampling interval. By increasing a sampled injection of malicious packets, he could also extract the sampling rate information by observing the reaction of the IDS in a trial and error fashion. Thus, a combination of a packet- and time-driven mechanism could mitigate such problems by applying a random chosen sampling interval within fixed boundaries.

Self-Regulation

Self-regulating sampling mechanisms have been presented in [375], for instance a method managing the processor usage in a network device through adaptive sampling in network security applications. The authors state that a wide range of common network anomalies only require a single sample in order to provide 100% accuracy of detection but there are also other network anomalies which cannot be detected with a single sample. An example is an anomaly which misuses a protocol for purposes which were not meant for it. This requires more advanced techniques than a simple signature check. Cryptographic mechanisms could be used to overcome such limitations. In [376] an adaptive packet-level sampling method on different traffic fluctuations and burst scales has been introduced. The method can dynamically adjust each packet sampling probability depending on the magnitude of traffic fluctuation. This approach achieves higher accuracy in contrast to random sampling methods. Another adaptive sampling method for anomaly detection algorithms has been presented in [377]. The adaptive sampling described is a promising general sampling technique that preserves well the traffic feature distributions and at the same time is able to improve the detection capabilities of the system. A hybrid sampling algorithm combining both flow statistics and feedback to intelligently choose the packets to sample is presented in [378] in order to achieve self-regulation. The sampling rate is determined by the current workload in the cloud, and thus minimizing the effects to normal workload. By the CIDS framework defined, an off-the-shelf IDS can be utilized in a cloud environment by reducing and balancing the data collection (packet capturing, filtering, sampling rate) and computation workload dynamically according to the resource utilizations in the cloud. Another example of adaptive sampling systems is the work presented in [379] that aims to effectively reduce the volume of traffic that Peer-to-Peer

(P2P) botnet detectors need to process while not degrading their detection accuracy. The system first identifies a small number of potential P2P in high-speed networks for botnet detection. In a 2-step approach first a suspicious host identification is performed by roughly sampling the traffic in order to detect potential P2P bots quickly. Second an in-depth analysis with more fine-grained detectors achieve an accurate detection on the identified hosts.

Applying sampling techniques in conjunction with a self-regulating IDS helps to reduce the measurement overhead for an IDS in terms of CPU, memory or bandwidth enabling the application of a partial IDS in future connected embedded systems. Similar to the concept of partial networking, the IDS components regulate their activeness such that in times of higher detection of malicious actions within the network more packets will be sampled leading to a higher resource consumption. On the other hand, in times of less or no detection, the IDS components lower their sampling or might even partially turn off completely. Furthermore, by using adaptive techniques that regulate, for instance, IDS relevant parameters or the sampling rate, the security level of a system can be adjusted by preserving a controllable overhead on resources.

6.1.2 Cryptographic Mechanisms for Incident Detection

In contrast to the classical IDS based on, e.g., statistical techniques, the following mechanisms are based on cryptographic schemes that can be exploited for alert generation and thus provide incident detection capability. Apart from the functionality of cryptographic protection, alerts can be raised if characteristics of the mechanisms fail, e.g., drifting sequence counters, incorrect decryption of messages or mismatching hash digests.

Encryption Schemes

To the best of the authors knowledge no work could be found during the writing of this article combining encryption schemes with IDS-functionality and sampling. However, one can distinguish between two types of secure data: full encryption and selective encryption. Full encryption means that all transmitted data is secured by cryptographic methods. These include the classic encryption protocols on Ethernet such as MACsec, IPsec, TLS or SSH. Similar to the sampling for IDSs, selective encryption mechanisms could be found mainly targeted to secure the data transfer over Ethernet on several communication layers.

MACsec, for instance, provides hop-by-hop security for layer-2 of Ethernet and is compatible with most of the Ethernet-based protocols. The data transmission is secured with GCM-AES. Within the MACsec frame format a Security Tag as an extension of the EtherType is defined that contains information about the association number within the channel, a packet number for replay protection and to provide a unique initialization vector for encryption and authentication algorithms which helps the receiver to identify the decryption key. The Integrity Check Value (ICV) generated by the GCM-AES following the payload guarantees that the packet was created by a node possessing the correct key. In the case of mismatching ICVs the frame gets dropped. This circumstance, failures in decryption or mismatching packet numbers could be exploited to trigger alerts. The same applies to other encryption protocols, for instance when Keyed-Hash Message Authentication Codes (HMAC) in IPsec do not match. However, since each device on a transfer route needs an implementation of the MACsec standard including special PHYs on all physical interfaces, MACsec is not very common and there are not many devices on the market. All components on a data path which have to be secured need certificates or mechanisms for pre-shared keys. Other protocols, according to [380], such as IPsec or TLS are very complex, suffer from context specific attacks, do not provide layer-2 security, or suffer from the lack of interoperability between their different versions.

In the context of selective encryption on the one side, the encryption of certain portions of a message is understood reducing the overhead spent on the encryption/decryption process. On the other side, selective encryption indicates that a sufficient number of messages is encrypted providing the necessary confidentiality for message transmission. This technique is mainly applied in resource-aware environments such as WSN or Mobile Ad Hoc Networks. Two methods are proposed in [381], alongside with a comprehensive literature survey, for the selection of message encryption. The first approach encrypts all messages and the second is based on the toss-a-coin approach in which approximately a half of the data get encrypted. The selective approach, however, leaves nearly a 50% chance for an adversary open to inject or manipulate packets which get not encrypted. The main application for selective encryption is in the field of image and video streaming and was not defined for the utilization on event-triggered messages. Further, the application in legacy-systems with legacy protocols is difficult since the encryption and decryption process retrofitted in these systems adds a significant delay to the original data transmission which makes it hardly applicable for real-time critical protocols. Beyond that, failing encryption or decryption could lead to a severe financial damage for instance in industrial control applications since packets are not guaranteed to be delivered in their deterministic time window. Since confidentiality in embedded environments is often of a minor priority, encryption/decryption schemes could be neglected [382]. Hence, authentication processes might be sufficient for the most of the cases.

Message Authentication Codes

Selective authentication is similar to selective encryption. However, to the best of the authors knowledge no comparable work could be found proposing selective authentication for network packet security. Unlike the full authentication, selective authentication only helps to identify whether or not data is still useful after being modified. A selective authentication approach presented in [383] is based on semi-fragile watermarks for vector geographical data.

Apart from the consideration of detection and protection methods separately, the authors of [384] and [385] combine the two classes for network security. In [384] a framework is proposed in which a multimodal biometrics or HMAC scheme is used for continuous authentication. Intrusion detection is modeled as sensors to detect the system's security state in order to protect high security mobile ad hoc networks. Intrusion detection in [384] is modeled as noisy sensors that can detect the system's security state (safe or compromised). Apart from Ethernet-based networks the approach of combining intrusion detection with a protection method is even proposed for the automotive CAN-bus. In [385] a simple intrusion detection method for detecting malicious frames is presented. The basis for this approach is that each electronic control unit is legitimately equipped with a hardware security module. By applying a certain type of MAC (HMAC, AES-CMAC, ...) and a counter within the proposed Domain_Activation and Domain_Violation frames, Denial of Service, replay and impersonation attacks can be detected and the driver notified about the risks.

Uncoupled MAC

Uncoupled MAC was originally described in [386] for securing network communication between legacy devices communicating with protocols which do not provide any security mechanisms. According to [387] many protocols in the TCP/IP stack do not provide mechanisms to authenticate the source or destination of a network traffic packet enabling to spoof the source address which is used in many attacks. By introducing dedicated embedded plug-in devices which are located interconnected to the legacy devices, authenticity and integrity of data communication can be provided by a MAC uncoupled from the original message.

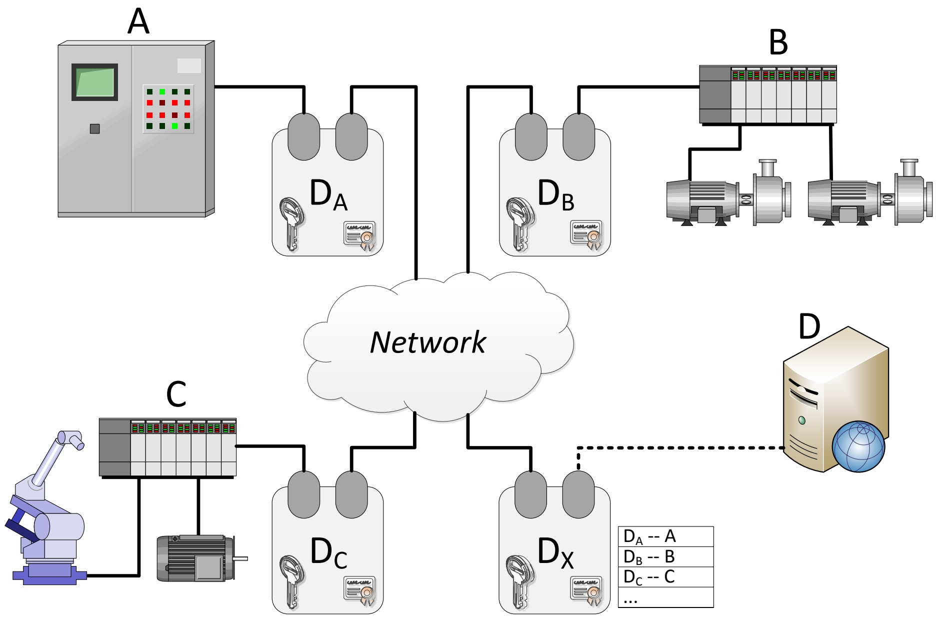


Figure 6.1: Network scenario of the proposed concept [388].

Thus, for instance with respect to Figure 6.1, if legacy device A is sending messages to B ($m_{\mathrm{A}\to\mathrm{B}}$), the embedded plug-in devices D_A and D_B are transparently forwarding the original message but decapsulated from the original message D_A and D_B are generating a MAC ($MAC(m_{\mathrm{A}\to\mathrm{B}})$). D_B is storing its generated MAC and waiting to verify it with the MAC generated by D_A sent over a secure communication channel. If the MAC are identical, data is accepted; otherwise, data is tampered. The overlay communication channel does not influence or delay the original traffic and, thus, prevents failures for instance through the MAC generation. Furthermore, in [386], the proposed security concepts (authenticated boot, direct attestation, key establishment) and the internal architecture for MAC generation and verification have been described for the embedded plug-in devices.

A more comprehensive approach of Uncoupled MAC was described in [388]. By introducing a further embedded surveillance device D_X (Figure 6.1) it is possible to securely bring new embedded plug-in devices into the network within the so called *Setup Phase* such that they can participate in the MAC generation and verification process. Within the *Runtime Phase*, D_X is used for aggregating info, warning and error messages produced by the embedded plug-in devices. A further significant characteristic of Uncoupled MAC is the separation of the Runtime Phase into a *MAC Phase* and an *Idle Phase* with a random duration such that an adversary is not able to exploit the algorithm. By only generating and verifying MAC within the MAC Phase, embedded plug-in devices only use a certain amount of bandwidth in order to have a low and deterministic impact on the original traffic (probe effect). By the predefined boundaries for the random chosen number of packets n, defining the MAC Phase and the duration of the Idle Phase α, a reasonable degree of security and additional overhead can be adjusted. A combination of a packet- and time-driven mechanism mitigates the problems of other systems described in the above sections in which n packets are sampled in each time-driven interval α by applying randomly chosen values within fixed boundaries. This combination allows the protection of event- and time-triggered communication.

Further, in [388] the extension of the MAC packets with a sequence counter and a timestamp was introduced guaranteeing the freshness and order of packets preventing replay attacks. Uncoupled MAC violations against the mentioned extensions are stored as anomaly logs within each individual embedded plug-in device and sent to the surveillance device. Besides a theoretical safety and security assessment, the concept of a complementary applied IDS within the *Control & Management* module is described. Thus, the combination of a protection- and detection-based technique was originally considered.

In [382] an assessment simulation model for a simple scenario including Uncoupled MAC is proposed. With the model, one is able to simulate bandwidth utilization and Uncoupled MAC overhead as well as detection rates by examining different values for n and α for a scenario in which an adversary is randomly injecting packets.

Compared to the approaches from the sections above, Uncoupled MAC has the following advantages and drawbacks.

Advantages

- Plug-and-play capability for legacy devices in form of embedded plug-in devices
- Integration in networked devices ensured by the concept
- Retrofitted cryptographic protection for protocols without security mechanisms
- No influence on underlying communication (e.g., real-time aspect will be preserved)
- Applicability to event- and time-triggered communication by the combination of a packet- and time-driven mechanism
- Functioning in communication intense systems by sampling (e.g., in switches or cloud platforms)
- Adjustable security level and additional overhead by adapting Uncoupled MAC parameters (e.g., in bandwidth-critical systems)
- Combination of a MAC algorithm with an IDS-functionality mitigates the lack of data integrity and data origin authenticity as with an IDS alone

Disadvantages

- Overhead on resources such as CPU, memory and bandwidth (but adjustable)
- 100% detection rate cannot be achieved due to sampling

For a holistic security eco-system, a hybrid system existing of the methods shown in Figure 6.2 is recommended. However, the main focus of this work is towards the resource conservation and protection methods. The detection by utilizing an IDS has already been proposed in [388]. More of interest is the interaction of the protection by Uncoupled MAC together with a self-regulating sampling method and the generation of alerts. On top of that a typical (more powerful) misuse- or anomaly-based IDS could be employed that is taking the sampled traffic as an input for in-depth intrusion detection. An alert analysis technique such as proposed in [165], on top of the resulting alerts from the protection and detection method, could be used to find a consensus and to initiate further reaction mechanisms, e.g., the reconfiguration or isolation of a network segment through SDN . The proper consensus finding might be beneficial in safety critical environments since it can be used to reduce for instance false positives. Uncoupled MAC allows to filter and sample for a certain protocol on a specific communication flow in parallel for multiple instances. Through the verification of n packets within a MAC Phase, not only packet- but also flow-based features for a downstream applied IDS can be derived.

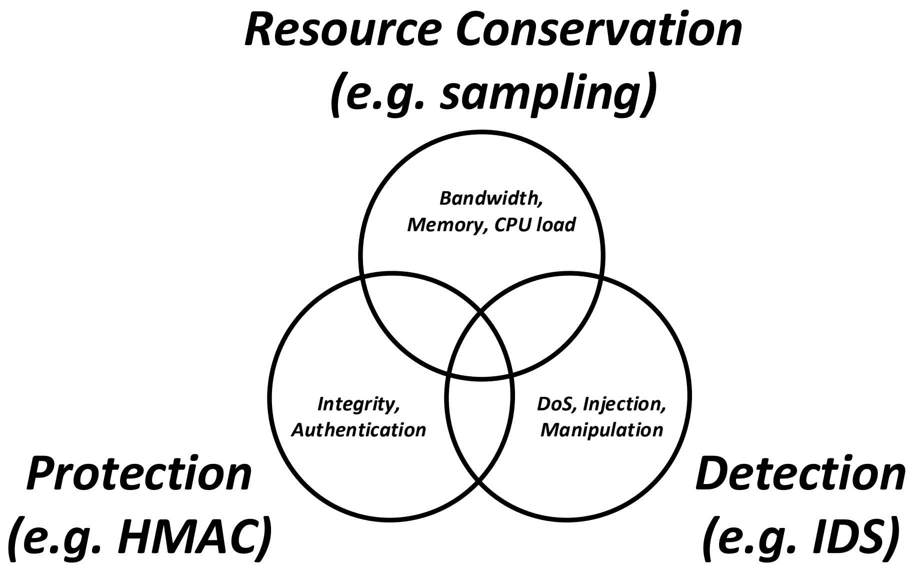

Resource Conservation
(e.g. sampling)
Bandwidth, Memory, CPU load
Integrity, Authentication
DoS, Injection, Manipulation
Protection
(e.g. HMAC)
Detection
(e.g. IDS)

Figure 6.2: Combination of methods.

6.2 Uncoupled MAC Algorithm Improvements

In this section improvements compared to the Uncoupled MAC algorithm proposed in [382, 386, 388] are provided. Some of them are needed to enable an IDS-functionality of the Uncoupled MAC algorithm and therefore facilitate the interaction with a typical IDS. Other improvements benefit the algorithm for the conservation of resources, with a specifiable level of security, or makes it applicable in a broader sphere and more effective.

6.2.1 Master/Slave Negotiation

One change of the algorithm compared to the original design, proposed in [388], is the negotiation between two communicating Uncoupled MAC partners devices to generate or verify MAC . Instead of a more complex negotiation between the securely communicating parties, the Uncoupled MAC partner that originally initiated their communication becomes the master and the other one becomes the slave. The direction of the data arriving first at one partner dictates which partner is set to master.

Rather than negotiating between master and slave which device starts with the generation/verification also defining the duration of the Idle Phase and how many packets will be examined during the MAC Phase, only the master is specifying this information and is communicating this to the slave. This significantly increases the performance of the algorithm since the overhead of the negotiation is prevented but the security aspect of randomly computing the parameters is still guaranteed.

6.2.2 Integration in Networked Devices

A major improvement compared to the originally intended use of Uncoupled MAC working on dedicated embedded plug-in devices is the integration in networked devices. In this mode Uncoupled MAC can also run on one interface such as *eth*0 as shown in Figure 6.3 enabling the application of the algorithm as a piece of software running on end devices or networking elements such as switches. This enables complete new fields of application for instance the utilization of Uncoupled MAC on ECUs or domain controllers in the automotive network domain.

In this mode of operation, a master and slave have to be defined at program start. The designated master device initiates the secure communication channel needed to pass status information and generation packets between the Uncoupled MAC partners. Instead of two hardware interfaces, only one interface is required and the internal ring-buffers for packet capturing are modified in a way that one captures only incoming traffic, while the other captures only outgoing traffic (Figure 6.3). The separation is necessary in order to fulfill the same functionality of the architecture of embedded plug-in devices shown in [386] and [388] that obviate the bridge interface *br*0.

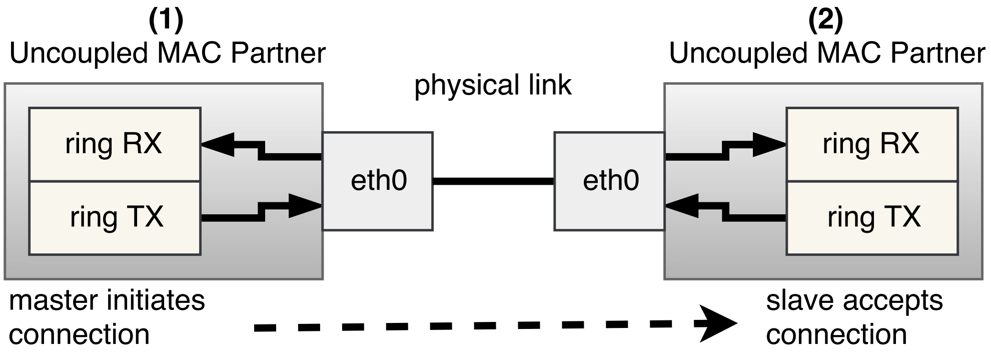


Figure 6.3: Integration of Uncoupled MAC operation in networked devices.

6.2.3 Synchronization of MAC Phase Start

A bottleneck of the decapsulation of MAC and message as with Uncoupled MAC is the start of the MAC Phases when making use of the proposed sampling technique (MAC Phase and Idle Phase). Similar to the so called *Two Generals Problem*, whose impossibility proof was first published in [389], the problem exists when two communicating Uncoupled MAC partners should start their MAC Phase. However, in the case of Uncoupled MAC different premises can be made by having a secure channel built by a certificate-based key establishment and an underlying time synchronization. By using a time synchronization protocol such as the Precision Time Protocol to synchronize the master and slave, it is possible to facilitate accurate timestamps that can be used to synchronize the start of the MAC Phase for both participants. This synchronicity is crucial for guaranteeing a stable operation of the algorithm and minimizes the chance of packets being missed by one partner. Therefore, after the Idle Phase with a random duration α, a timestamp t_0 is taken by the master at the beginning of the Runtime Phase, to which an offset value t_{off} is added. The value t_{off} is at least a half of the Round Trip Time (RTT) between the master and slave ($\frac{1}{2} \times RTT$) and should be as small as possible, leaving only a minimal time window for an attacker to take advantage of. The resulting time value t_1 ($t_0 + t_{off}$) marks the start of the MAC Phase for both participants and is added to the regulation packet that is sent to the slave, thus allowing both partners to start the MAC Phase at the same time. The timing diagram in Figure 6.4 depicts the timing synchronization mechanism.

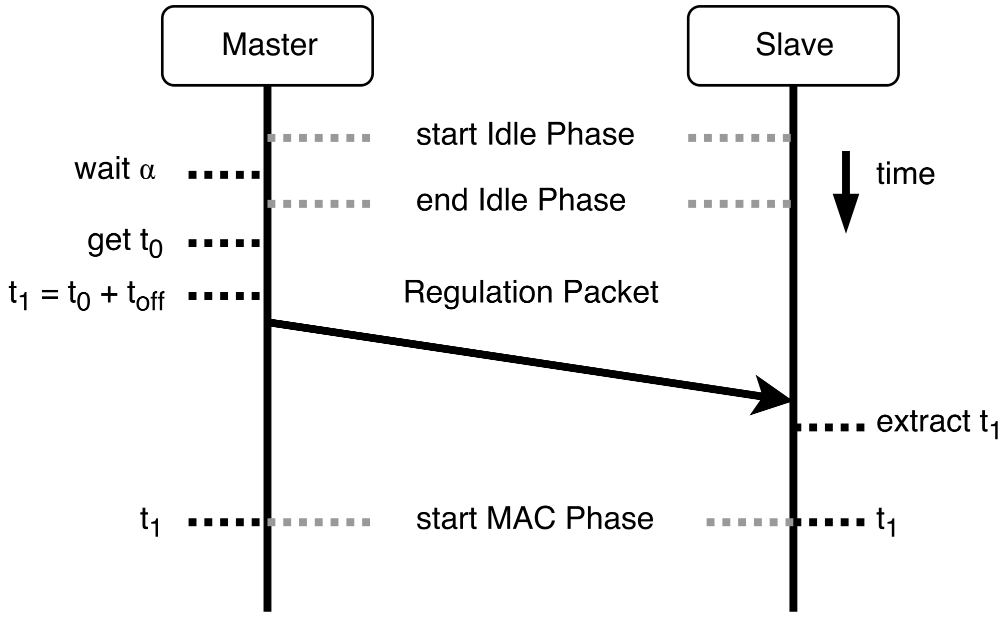


Figure 6.4: Timing diagram for a synchronized start of a MAC Phase.

6.2.4 Static Communication Mode

Depending on the network to secure, the embedded surveillance device presented in [388] may be omitted reducing the communication overhead towards each individual Uncoupled MAC partner. Especially in static networks, in which the number of Uncoupled MAC entities, communicating end-devices and thus the IP-addresses are defined, the static communication mode significantly speeds up the performance.

6.2.5 IDMEF Extension

The IDMEF, an XML-based language, was specified as a RFC document to standardize the interfaces between intrusion detection, response and management components. Over the last decade several public and proprietary system-level standardization initiatives arose but they all ended in criticism, yielded weak adoption or have even been withdrawn as stated in [77] pursuing a standardized format. IDMEF has the advantages that the messages specified are well structured, the message fields (i.e. tags and attributes) have rigorous syntactic and semantic definitions and support better message expressivity [77].

A large number of popular IDSs (OSSEC, Barnyard2, Samhain, Surricata, ...) or SIEM -systems are using IDMEF. To ensure compatibility with those systems, Uncoupled MAC algorithm is extended by an IDMEF module. The proprietary defined info/error/warning message is replaced with a standardized format allowing a central authority to handle and manage security exceptions detected by Uncoupled MAC violations. Such violations have been discussed together with possible attack vectors and errors around the application of Uncoupled MAC in [388] and are listed in the following.

- HMAC packet inconsistencies (Hash mismatch, HMACs are not received, no HMAC for original packet, no feedback HMAC packet, ...)
- Sequence counter issues (Expected counter value is not within the tolerance window or counter value occurs twice, ...)
- Timestamp issues (Uncoupled MAC partners do not share the same time base, timestamp in HMAC packets exceed tolerance windows, ...)
- Authentication issues (Authenticated boot fails, direct attestation not possible, verifying certificates fails, ...)
- Connection issues (Direct, secure communication between Uncoupled MAC partners or surveillance device is terminated unexpectedly, a device does not report its active state, ...)

The support for IDMEF not only equips Uncoupled MAC with an IDS capability but also allows the interoperability with other IDSs allowing to find a consensus on alerts from various incident detection sources on a standardized format. This benefits the usage of alert analysis techniques not only on the devices themselves but also on a central more intelligent platform orchestrating different detection and reaction components as shown in Figure 1.2 for systems that can, according to [73], be arranged distributed or decentralized. Apart from IDMEF, the more recent IDEA might be the integrated as part of future work since it is more lightweight compared.

6.3 Self-Regulation Algorithm

An Uncoupled MAC self-regulation algorithm is proposed in order to adjust the sampling parameters, the number of packets during a MAC Phase n and the duration of an Idle Phase α, on-the-fly. This allows for a given percentage q_0 of packets to be sampled/examined, which can be regarded as the adjustable security level, to increase or decrease the parameters n and α. This depends on the number of detected incidents z_D during a MAC Phase caused by Uncoupled MAC violations with respect to Section 6.2.5 and the number of unmeasured packets p_n during an Idle Phase.

Figure 6.5 shows a schematic control circuit illustrating the self-regulation. The proposed mechanism might not only be applied for Uncoupled MAC but also for an IDS capable of sampling. This self-regulation approach can be utilized for a reasonable overhead and in the case of IDSs for a processable amount of data while still adapting to the systems' security state. According to the schematic from Figure 6.5, formulas and algorithms for the Uncoupled MAC parameters n and α are derived for self-regulation.

6.3.1 Number of Packets n per MAC Phase

In Equation 6.1 the number of packets to be examined in the next MAC Phase $n[k+1]$ is calculated depending on the number of packets of the current MAC Phase $n[k]$, the scaled number of packets identified by Uncoupled MAC detection ($z_D[k]$) and an intermediate

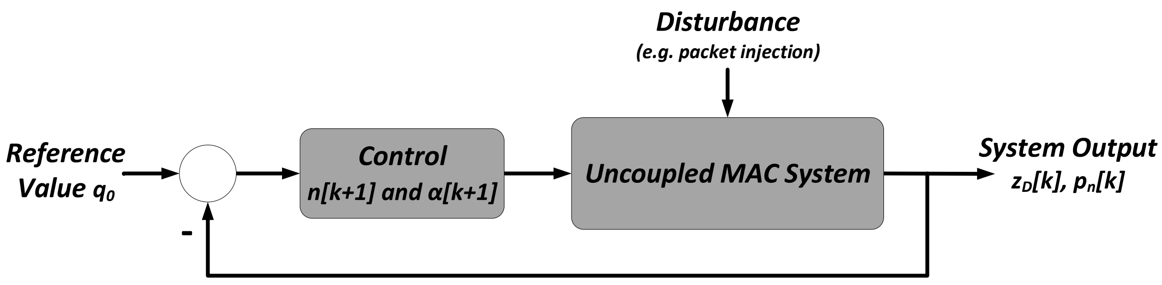


Figure 6.5: Self-regulating sampling approach for Uncoupled MAC.

value $q[k]$. This value $q[k]$ is responsible to keep the percentage of examined packets on a predefined value of q_0 depending on the number of unmeasured packets during the Idle Phase $p_n[k]$ (Equation 6.2). The more packets are deemed as faulty by the algorithm, the higher the number of packets to be controlled in the next phase is set. However, if no Uncoupled MAC violations occur, the percentage of packets to be examined retains its value of q_0.

$$n[k+1] = n[k] + n_s \cdot z_D[k] + q[k] \tag{6.1}$$

$$q[k] = \left(q_0 - \frac{n[k]}{n[k] + p_n[k]}\right) \cdot n[k] \tag{6.2}$$

Where:

$n[k+1]$: packets to check in the next MAC Phase

$n[k]$: packets checked in the present phase

n_s: scale factor to set the impact of erroneous packets on the next phase

$z_D[k]$: number of erroneous packets detected in the current phase

$q[k]$: percentage of packets to check in the next phase

q_0: desired percentage of packets to check

$p_n[k]$: packets not checked due to the Idle Phase

6.3.2 Waiting Duration α in the Idle Phase

In Equation 6.3 the waiting duration $\alpha[k+1]$ for the next Idle Phase is computed depending on a random value X_r within fixed boundaries $[\alpha_{MIN}, \alpha_{MAX}]$ and the scaled number of packets identified by Uncoupled MAC detection ($z_D[k]$) of the current MAC Phase.

$$\alpha[k+1] = X_r \in [\alpha_{MIN}, \alpha_{MAX}] - \alpha_s \cdot z_D[k] \tag{6.3}$$

Where:

$\alpha[k+1]$: duration of the next Idle Phase

X_r: random value in range $X_r = [\alpha_{MIN}, \alpha_{MAX}]$ calculated in each Idle Phase

α_s : scale factor to set the impact of erroneous packets on the next phase

$z_D[k]$: number of erroneous packets detected in the current phase

6.3.3 Formula Verification

A simple formula verification implemented in the *Python* programming language shows the desired behavior of $n[k+1]$ and $\alpha[k+1]$ depending on a predefined percentage of 60% of packets to be examined ($q_0 = 0.6$). The scaling factors in this example have been chosen to be $n_s = 0.3$ and $\alpha_s = 0.4s$. A total number of 1000 phases (MAC Phase + Idle Phase) have been examined. X_r is computed in each phase within the range of $[2s, 3s]$. Further, for each phase, a random number of unmeasured packets can be determined between the range $[3, 5]$. As shown in Figure 6.6, two scenarios for detected packets (security incidents) by Uncoupled MAC ($z_D[k]$) are considered at $\frac{1}{3}$ and $\frac{2}{3}$ of the total of 1000 phases. At $\frac{1}{3}$, a constant number of 4 packets was detected for 100 phases resulting in a clear increase of $n[k+1]$ and a significant drop of $\alpha[k+1]$. However, after no more packets were detected, the values for n and α leveled off such that 60% of packets were examined again. At $\frac{2}{3}$, a steady increase of $z_D[k]$ shows clearly that more packets can be examined expressed by a steady increase of n and a shorter Idle Phase through a smaller α value. Thus, in a network having an increase on malicious activities more packets can be examined. It must be noted that Equation 6.1 and 6.3 showing linear behavior regarding the term adjusted with z_D. However, in some cases, for instance when a faster adaption of n and α is desired, it might be necessary to consider non-linear behavior using a non-linear function $f(z_D)$.

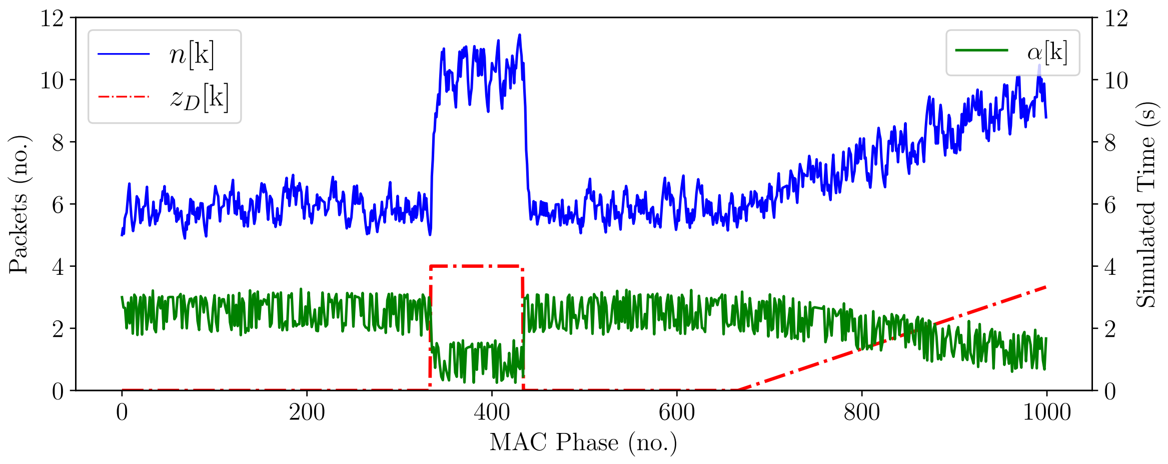


Figure 6.6: Verification of the self-regulation formulas.

6.3.4 Algorithm Notation

In order to implement Equation 6.1 and 6.2 that dictate the behavior of the self-regulation system for the number of packets to be examined during a MAC Phase, Algorithm 8 is proposed. In addition to the computation given by the equations above, additional mechanisms must be added to the algorithm. One mechanism is to prevent $n[k+1]$ from falling below a threshold value N_{MIN}, ensuring that a minimum of packets is checked in each phase. Complementary to this, an upper threshold N_{MAX} must be defined to prevent $n[k+1]$ from reaching a too high value, producing excessive overhead.

For the α value described in Equation 6.3, Algorithm 9 is proposed. In addition to the computation given by Equation 6.3, two thresholds are added. These thresholds prevent α from reaching a value less than α_{MIN} or greater than α_{MAX}. This ensures that α is always a positive integer, not exceeding the specified Idle Phase duration value.

Algorithm 8: Calculation of n for the next phase $k+1$.

```
Input: $z_D[k]$, $n[k]$, $p_n[k]$
Output: $n[k+1]$
    Initialization : $q[k]$
    Constants : $N_{MIN}$, $N_{MAX}$, $q_0$, $z_s$
 1: $q[k] \leftarrow q_0 - (n[k]/(n[k]+p_n[k])) \cdot n[k]$
 2: if $(q[k] < 0)$ then
 3:     $q[k] \leftarrow 0$
 4: end if
 5: $n[k+1] \leftarrow n[k] + z_s \cdot z_D[k] + q[k]$
 6: if $(n[k+1] < N_{MIN})$ then
 7:     $n[k+1] \leftarrow N_{MIN}$
 8: else if $(n[k+1] > N_{MAX})$ then
 9:     $n[k+1] \leftarrow N_{MAX}$
10: end if
11: return $n[k+1]$
```

Algorithm 9: Calculation of α for the next phase $k+1$.

```
Input: $X_r$, $z_D[k]$
Output: $\alpha$
    Constants : $\alpha_s$, $\alpha_{MAX}$, $\alpha_{MIN}$
1: $\alpha \leftarrow X_r - \alpha_s \cdot z_D[k]$
2: if $(\alpha < \alpha_{MIN})$ then
3:     $\alpha \leftarrow \alpha_{MIN}$
4: end if
5: if $(\alpha > \alpha_{MAX})$ then
6:     $\alpha \leftarrow \alpha_{MAX}$
7: end if
8: return $\alpha$
```

6.4 Evaluation

The proposed self-regulation algorithm and the Uncoupled MAC improvements have been added to the implementations presented in [386] and extended in [388]. The programming language is native *C* using the libraries *OpenSSL* for key establishment and HMAC-SHA-256 generation/verification in the MAC Phases, *PF_RING* in combination with *libpcap* for packet processing and *libprelude* for IDMEF support. The evaluation in this article deals with customized attack scenarios in an environment targeted to exploit the weaknesses of Uncoupled MAC's concept of Idle Phases to show their impact on the detection capability by also examining the trade-off of preserving resources. It is therefore no aim to compare the IDS-functionality with other attack detection mechanisms in this work since Uncoupled MAC is a cryptographic scheme by nature and provides as a side effect simple IDS-functionality benefiting environments characterized by resource constraints and static communication. Especially a large network data diversity, not provided with the evaluation environment, is a mandatory aspect to apply anomaly-based machine learning mechanisms in order to properly learn the network behavior.

6.4.1 Virtualized Environment

A virtualized environment is preferred compared to the Simulation Assessment Model presented in [382] since the simulation is an idealized model of the algorithm with many constraints such as the neglection of bidirectional communication. The virtualized environment behaves like an authentic evaluation with real (embedded) devices since among others the hardware and network resources and constraints can be evaluated as well. For more flexibility in carrying out exhausting measurements evaluating various scenarios for trend estimation of Uncoupled MAC behavior, the self-regulation algorithm might be implemented as an extension for the Simulation Assessment Model in future work.

The virtualized evaluation environment is implemented on a virtual *Proxmox* platform for scalability and flexibility. All machines as well as the intermediary devices are virtual entities on the same virtualization host as illustrated in Figure 6.7 including the associated hardware constraints.

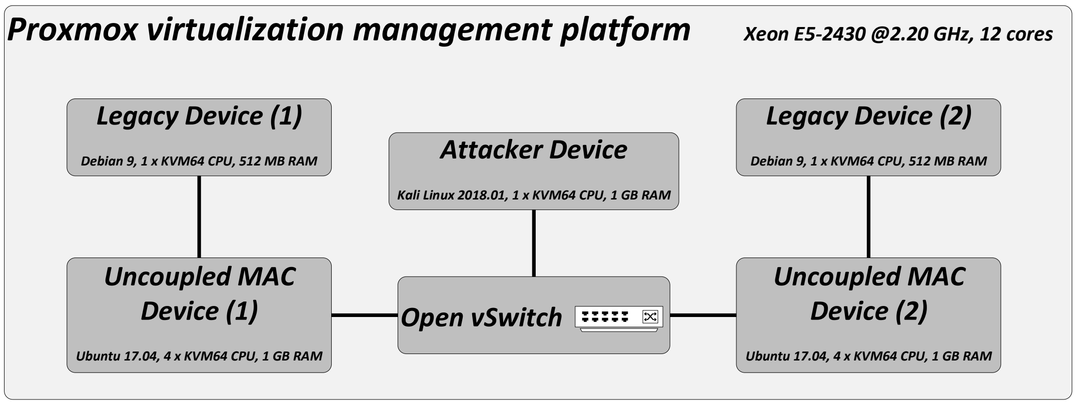

Proxmox virtualization management platform
Xeon E5-2430 @2.20 GHz, 12 cores
Legacy Device (1)
Debian 9, 1 x KVM64 CPU, 512 MB RAM
Attacker Device
Kali Linux 2018.01, 1 x KVM64 CPU, 1 GB RAM
Legacy Device (2)
Debian 9, 1 x KVM64 CPU, 512 MB RAM
Uncoupled MAC Device (1)
Ubuntu 17.04, 4 x KVM64 CPU, 1 GB RAM
Open vSwitch
Uncoupled MAC Device (2)
Ubuntu 17.04, 4 x KVM64 CPU, 1 GB RAM

Figure 6.7: Structure of the virtual evaluation environment.

In order to evaluate both, the plug-in and integrated mode, two Uncoupled MAC Devices and two Legacy Devices are used. Testing in the integrated device mode having only communication between the Uncoupled MAC Devices each evaluation ends up with the same results compared to the plug-in mode in which the Uncoupled MAC Devices secure the Legacy Device communication. To connect the Legacy Devices with the Uncoupled MAC Devices, a simple virtual Proxmox-internal bridge is sufficient. The Uncoupled MAC Devices are connected via an Open vSwitch (OVS) to allow port-mirroring and packet injection by the attacker, which is connected to the same switch.

In the following evaluations for plug-in and integrated mode, the traffic between the Legacy Device (1) and (2) is secured by the Uncoupled MAC Devices (1) and (2) and originates from Legacy Device (1). For evaluation purposes, the Internet Control Message Protocol (ICMP) is used in form of ICMP echo-requests and corresponding replies. An Attacker Device is impersonating Legacy Device (1) and is injecting packets to test the Uncoupled MAC detection capability in different scenarios. In addition to this, several conditions and test parameters are defined:

1. the interval between ICMP echo-requests is set to 500 ms simulating a periodic communication
2. the standard size for an ICMP echo-request and echo-reply is used (98 bytes total length, 48 bytes payload)

3. a maximum of 32 packets for n is checked per MAC Phase
4. a minimum of 8 packets for n is checked per MAC Phase
5. a maximum value of 50 (5 s) for α is determining the Idle Phase maximum for the attack scenarios continuous and stochastic, while α is set to 42 (4.2 s) as an Idle Phase maximum value for the single and burst injection attacks; the details on each attack scenario are described in Section 6.4.2
6. a minimum value of 3 (300 ms) for α is determining the Idle Phase minimum for all attack scenarios
7. Uncoupled MAC Device (1) is set as the master, Uncoupled MAC Device (2) as the slave
8. the master device is responsible for HMAC verification, the slave device is responsible for HMAC generation
9. the Uncoupled MAC Devices are synchronized via the Precision Time Protocol, enabling precise start times for the MAC Phases on both devices and accurate generation packet timestamps

6.4.2 Attack Scenarios

Focus of the following customized attack scenarios is to exploit the Uncoupled MAC algorithm's Idle Phase times in order to inject spoofed data. In those phases no packet authenticity and integrity checks are performed. Since any other attack type (modification, replay or man-in-the-middle) as discussed in [388] would also be detected due to Uncoupled MAC violations (Section 6.2.5), only variations of packet injection attacks are defined to simulate different adversary skill levels. In order to evaluate the reliability of detection and the benefits of the self-regulated version of Uncoupled MAC compared to the non-self-regulated one, the following scenarios are specified:

1. *Continuous Injection* - a single packet is injected in fixed intervals
2. *Stochastic Injection* - a random number of packets in a specific range is injected at random intervals with a fixed minimum and maximum time between injections
3. *Bandwidth Low Injection* - one (single) or multiple (burst) packets are injected after a bandwidth low is detected indicating an Idle Phase
4. *Weak Spot Injection* - a single packet is injected immediately after the Regulation Packet is sent out by the master and detected by the attacker

For single and burst bandwidth low injection an intelligent adversary might monitor the cycle of MAC Phases and Idle Phases and learns the average bandwidth overhead in order to inject packets after a bandwidth low is detected indicating an Idle Phase. One could say that an attacker who would be able to only monitor the actual network traffic produced by Uncoupled MAC, e.g., by monitoring Uncoupled MAC's port number, would be able to inject packets without measuring the bandwidth. He would though be able to see when no generation and verification packets are exchanged indicating an Idle Phase (if no heartbeat messages are applied) but could not determine an actual trigger value because of Uncoupled MAC's random parameters n and α. Thus, a learning phase by a bandwidth measurement over multiple phases is necessary in order to gain a dedicated trigger value for injection. It must be noted that these scenarios are rather theoretical since in a realistic network environment more than two parties communicate resulting in an even higher bandwidth utilization with fluctuations such that it might not be possible to determine the actual bandwidth overhead by Uncoupled MAC in order to detect the Idle Phases through bandwidth lows.

Weak spot injection assumes that an attacker is able to identify the regulation packet on the network data. However, as stated in [388], Uncoupled MAC communication is transferred over a secure encrypted channel with applied heartbeat messages. Thus, on the one side the communication overhead increases but leaves no possibility that an attacker is able to distinguish a regulation packet from a heartbeat message. Since these features are complementary depending on the desired security level and possible overhead, the evaluation deals with a lightweight version not considering applied heartbeat messages.

6.4.3 Attacker Implementation

The attack scenarios are implemented using Python-based scripts. For continuous injection, the script is used to inject packets at an interval of two seconds between injections. The stochastic injection sends a random count of attacker packets $a_{packets} = [a_{min}, a_{max}]$ at random times $\Delta t_{random} = [\Delta t_{min}, \Delta t_{max}]$. Both scenarios could also represent a non-malicious behavior for instance when a network participant acts as a so called “babbling idiot” in sending out continuous or stochastic packets as a result of a malfunction. The latency of the attacker script must also be considered in the evaluation of the continuous and stochastic attacks. If, for example, the script is started during a MAC Phase, the first injection may take place in the Idle Phase, resulting in a low detection rate for the first phase. On the other hand, if the script is started in an Idle Phase, a high detection rate might be achieved for the first phase. This non-deterministic behavior of the attacker program results from first constructing the attacker packet at the start of the script and the delay introduced by the sockets.

For the bandwidth low injection, an intelligent attacker program is developed. This intelligent attacker program first sniffs 120 packets passed between the two Uncoupled MAC partners in a learning phase. During this phase, bandwidth data are computed and stored, which is then assessed in the processing phase. This phase yields mean values for the upper, lower and middle bandwidth. The mean values can then be used in the attack phase, in which the bandwidth is continuously measured using a sampling rate of 10 ms. Then packets are injected either in single (a single packet) or burst mode (e.g., three packets) each time a bandwidth low is reached. The bandwidth low marks the start of an Idle Phase for the attacker, thus leaving a confined time window for an attack. An example bandwidth measurement of the attacker in an arbitrary attack phase with injected packets when a bandwidth low is detected is shown in Figure 6.8.

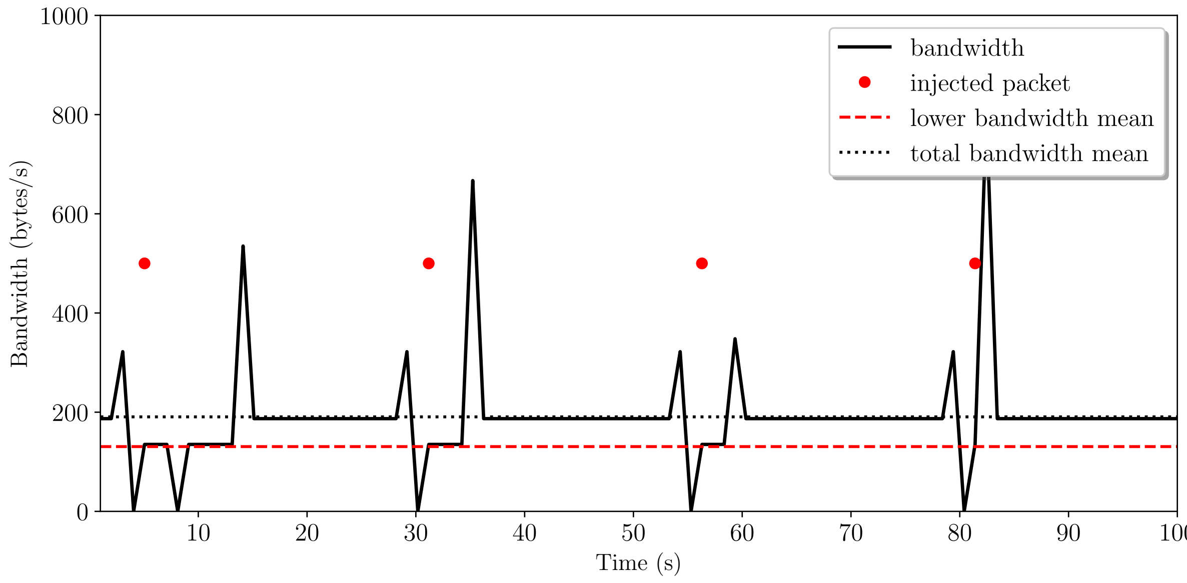


Figure 6.8: Attacker bandwidth measurement and injected packets in detected bandwidth lows.

6.4.4 Evaluation Metrics

For the evaluation of IDSs, especially for the problem of statistical classification, different characteristic values are used. One of the most important notations is to use the parameters derived from the Confusion Matrix (referring to Table 2.2 in Section 2.1.4) as stated among other literature in [390]. This specific table allows the representation of the performance of an algorithm, typically used for machine learning but in this context used to build a bridge towards Uncoupled MAC's incident detection functionality.

From the parameters of Table 2.2, formulas for the computation of many other characteristic values (sensitivity, specificity, accuracy, etc.) can be derived. For Uncoupled MAC, the following assumptions can be made: Due to the nature of MAC generation and verification in a MAC Phase, no false positives can be obtained (FP=0) since modified or injected packets can certainly be detected to be true positives. Unexamined packets within the Idle Phase will either yield true negatives in the case of non-malicious packets or false negatives in the other case. The possible conditions for Uncoupled MAC in an arbitrary phase are shown in the example of Figure 6.9.

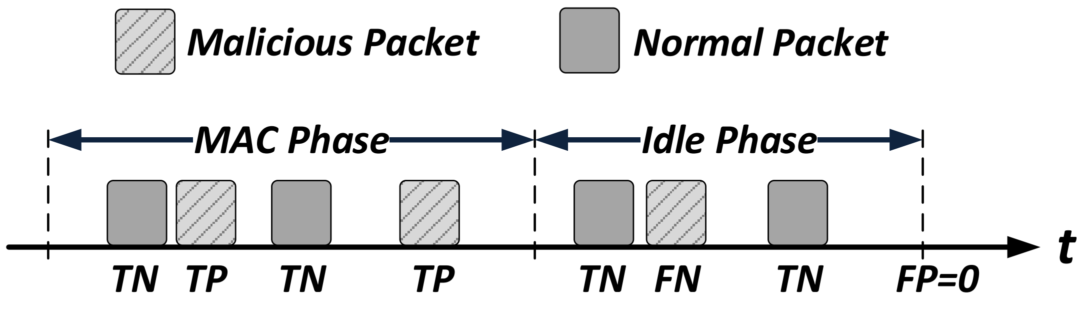


Figure 6.9: Confusion Matrix parameters for some Uncoupled MAC phase k.

The quantity used to measure the efficiency of an algorithm's self-regulation and non-self-regulation is the detection rate since it is according to [378] the most important metric for an IDS. For Uncoupled MAC evaluation it is defined as the percentual value of the number of injected attacker packets versus the number of these packets detected by the Uncoupled MAC algorithm. The detection rate (DR) described in percent corresponds to the true positive rate (TPR) or sensitivity derived from the Confusion Matrix which can be computed according to Equation 6.4. In the example from Figure 6.9, for some phase k, $TP = z_D[k]$ equals 2 and $FN = 1$ yields a detection rate $DR = TPR = \frac{2}{2+1} = 67\%$.

$$TPR = \frac{TP}{TP + FN} \tag{6.4}$$

6.5 Measurement Results

The results from measurements, taken within the evaluation environments while multiple attack scenarios are executed, are described in the following sections. The values for non-self-regulation are set such that comparable to self-regulation approximately $(q_0 \times 100)\%$ of packets are examined by default.

6.5.1 Continuous Injection

A continuous attack is carried out on both the self-regulation and the non-self-regulation algorithm. In this attack mode, a single attacker packet z_D is injected into the network in fixed intervals of two seconds. The scenario has been carried out ten times for each algorithm over a total of 30 MAC phases. The mean values for the detection rates are calculated for both the non-self-regulation algorithm and the self-regulation algorithm with the following set of parameters for self-regulation: $n_s = 1.30$, $\alpha_s = 2.0$, $q_0 = 0.75$.

While the non-self-regulation version of Uncoupled MAC performs poorly in this scenario, the self-regulation version produces a significantly higher mean detection rate (Figure 6.10). By adjusting the Uncoupled MAC parameters n and α, the detection rate increases over runtime due to the increase of n and average decrease of α following an exponential approximation curve for self-regulation. However, both algorithms would reliably detect an attacker or a malfunctioning device acting as a "babbling idiot".

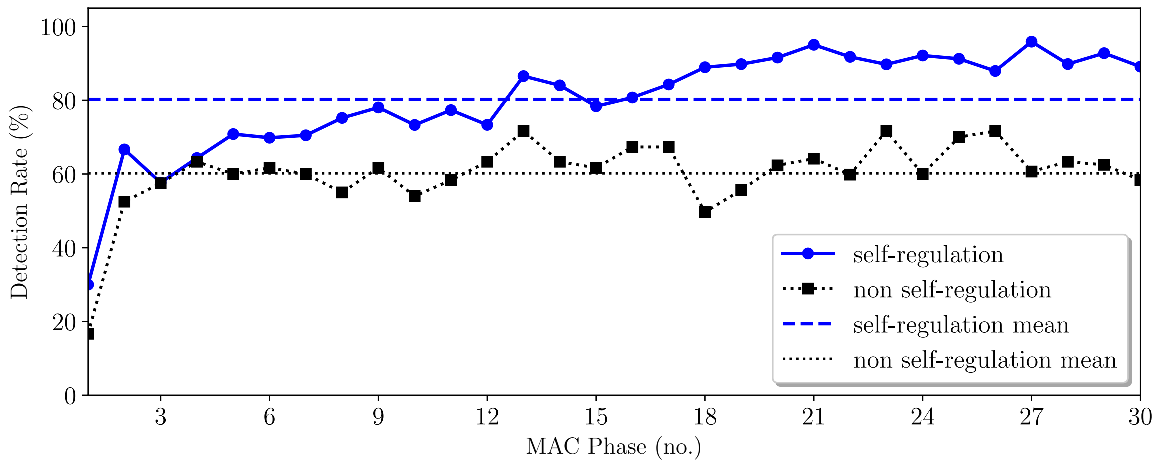


Figure 6.10: Detection rate - continuous packet injection.

6.5.2 Stochastic Injection

In this attack scenario the attacker sets a minimum and a maximum of packets z_i to inject. For this test, the interval $z_i \in [1; 4]$ was chosen. Further, a minimum and maximum for the time between injections t_i (in seconds) needs to be fixed, which in this case is set to the interval $t_i \in [0.5; 10]$. A number of packets to inject (z_i) and a wait time (t_i) are selected randomly from the respective interval. The attack is carried out 10 times over 30 phases with the following parameters chosen for the self-regulation algorithm: $n_s = 0.80$, $\alpha_s = 2.0$, $q_0 = 0.75$.

The mean values for the detection rates are calculated for both the non-self-regulation algorithm and the self-regulation algorithm and illustrated in Figure 6.11. Compared to the continuous injection, the average detection rate is smaller due to a greater difficulty to detect random injection. Further, the fluctuation expressed by the curve is greater because of the mean computation of multiple stochastic measurements. However, again the self-regulation performs better than non-self-regulation.

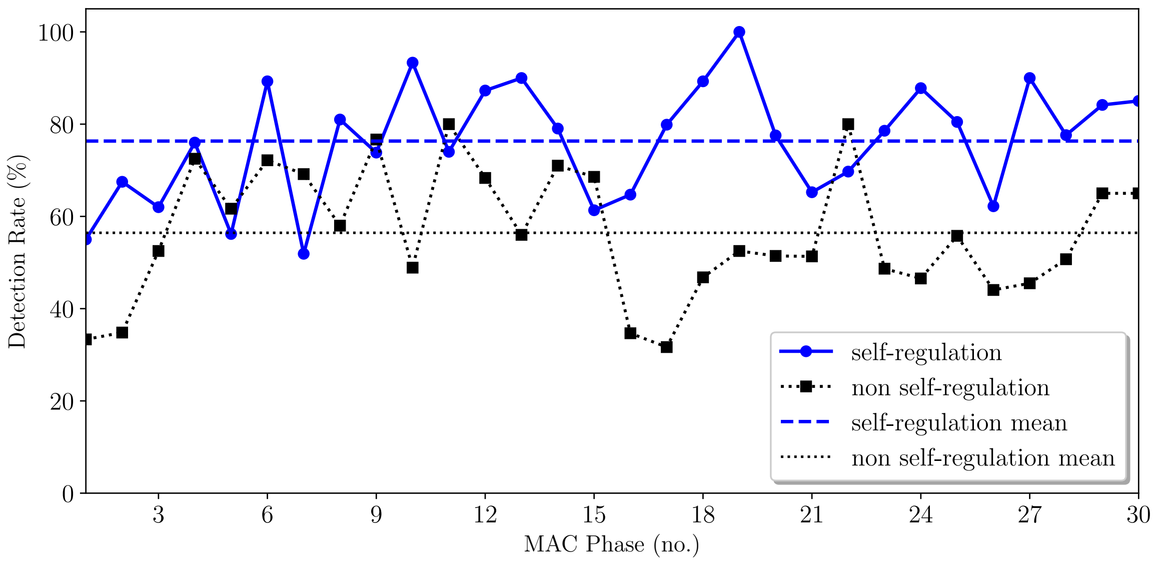


Figure 6.11: Detection rate - stochastic attack mode.

6.5.3 Bandwidth Low Injection

A more sophisticated attack scenario compared to the continuous or stochastic injection is the exploitation of the Idle Phase by measuring the typical MAC Phase and Idle Phase period and injecting packets when a bandwidth low representing an Idle Phase is detected. Two subscenarios are performed. One in which only a single packet is injected by an attacker and a second in which more information is transferred in form of multiple injected packets.

Single Injection

Figure 6.12 depicts the results of an attack carried out over a time period of 300 seconds for both variants of the Uncoupled MAC algorithm. The chart shows the duty-cycle of Uncoupled MAC, alternating between MAC and Idle Phases. The packets that are detected by Uncoupled MAC are shown in green, the packets that were not detected are shown in red. While the non-self-regulation algorithm does not exceed a detection rate of 20%, the self-regulation algorithm detects about 50% of the attacker packets on average. When an attacker packet is detected by the self-regulation algorithm, the duration of the Idle Phase (α) is reduced and the number of packets to be checked is increased (n) as described in Section 6.3. Due to this behavior, the margin for packet injection is also reduced, resulting in fewer possibilities for an attack, while the detection rate of Uncoupled MAC increases. For this scenario the following parameters where chosen for the self-regulation algorithm: $n_s = 18.0$, $\alpha_s = 25.0$, $q_0 = 0.75$. By choosing stricter parameters, the impact of a single attacker packet on the resulting behavior is greater than with the scenarios described in Section 6.5.1 and 6.5.2, resulting in a faster reaction compared to a detected injection in the continuous and stochastic attack scenarios.

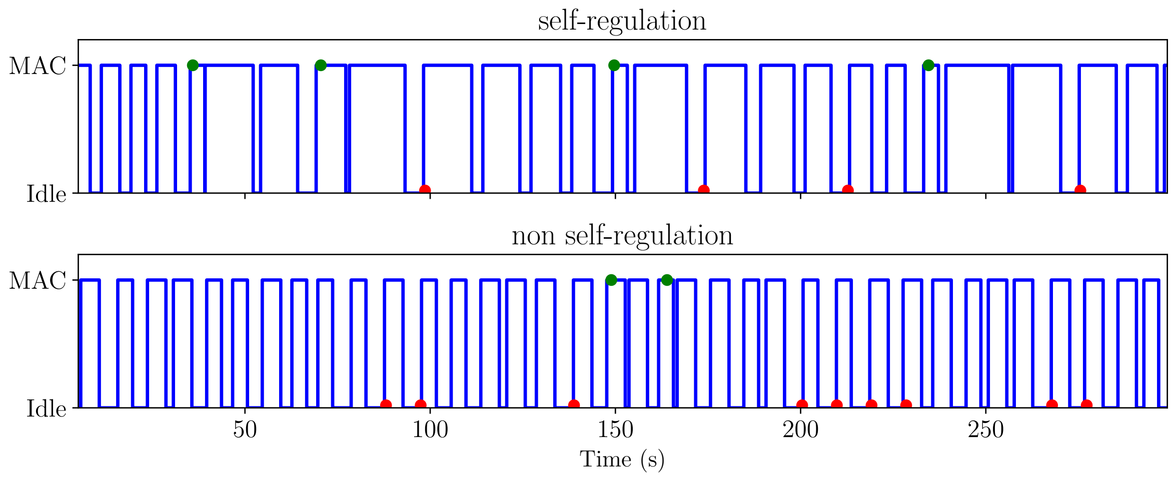


Figure 6.12: Packet detection per phase - single attack mode.

Burst Injection

Figure 6.13 depicts the results of an attack carried out over a time period of 300 seconds for both variants of the Uncoupled MAC algorithm. The chart shows again the duty-cycle of Uncoupled MAC, alternating between MAC and Idle Phases when three packets are injected per bandwidth low. The detected packets are shown in green, the undetected packets in red. Again the margin for the attacker program is reduced by the self-regulation algorithm. In this scenario the number of attacker packets introduced to the test network is greater than with the single injection scenario, making the limitation of the attacker scope more distinct. While 72 packets are injected with the non-self-regulation algorithm having a detection rate of approximately 47%, only 21 packets are injected for the self-regulation variant (detection rate of approximately 62%), due to the fact that the self-regulation variant dynamically adjusts its behavior with respect to the attacker packets detected. In this scenario, the same parameters as with the single injection were selected for the self-regulation algorithm, again with the aim to maximize the detection rate after an attack was registered as fast as possible.

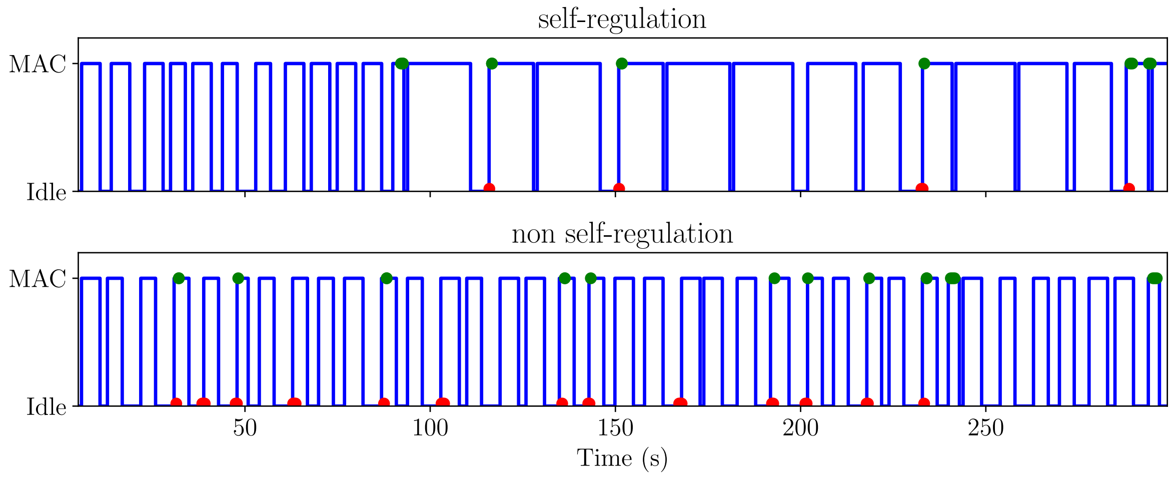


Figure 6.13: Packet detection per phase - burst attack mode.

6.5.4 Weak Spot Injection

As described in Section 6.2.3, the ideal offset t_{off} to start a new MAC Phase initiated by the master in sending a Regulation Packet to the slave is $\frac{1}{2} \times RTT$. However, in a practical implementation when referring to Figure 6.4, t_{off} must include deviations for instance due to network jitter, processing delays for transmitting and receiving the Regulation Packet or delays considering the timestamp and t_{off} generation. Therefore it must be typically larger than $\frac{1}{2} \times RTT$. Even if according to the concept the exchanged Uncoupled MAC information is encrypted and thus an attacker cannot identify the Regulation Packet directly, in this evaluation he might be able to exploit the small time window after an Idle Phase ends until a new synchronized MAC Phase starts (weak spot) to inject a malicious packet.

The injection of a single packet of the intelligent attacker described in Section 6.4.2 is triggered immediately after identifying a Regulation Packet for a total number of 15 phases. Due to the fact that the t_{off} value is implemented in both algorithms with and without self-regulation in an identical way, and independent of the calculation of n and α values, the stated facts concern both algorithms likewise. The detection rate of Uncoupled MAC for changing integer multiples of the RTT is illustrated in Figure 6.14 for Uncoupled MAC with applied self-regulation. The average RTT in this scenario derived utilizing the *ping* tool is 0.804 ms. The greater the value for t_{off}, the less malicious packets are detected. For a practical implementation of Uncoupled MAC, thus, values for $t_{off} \in [1; 5[\times RTT$ are recommended for this scenario in order to mitigate weak spot injection. However, even a large value for t_{off} of approximately $75 \times RTT$ would still detect attacker packets under the set conditions. The graph states further the estimated detection rate in network environments in which the Regulation Packet is delayed for instance due to overloaded network switches up to a maximum of $100 \times RTT$ when applying self-regulated Uncoupled MAC.

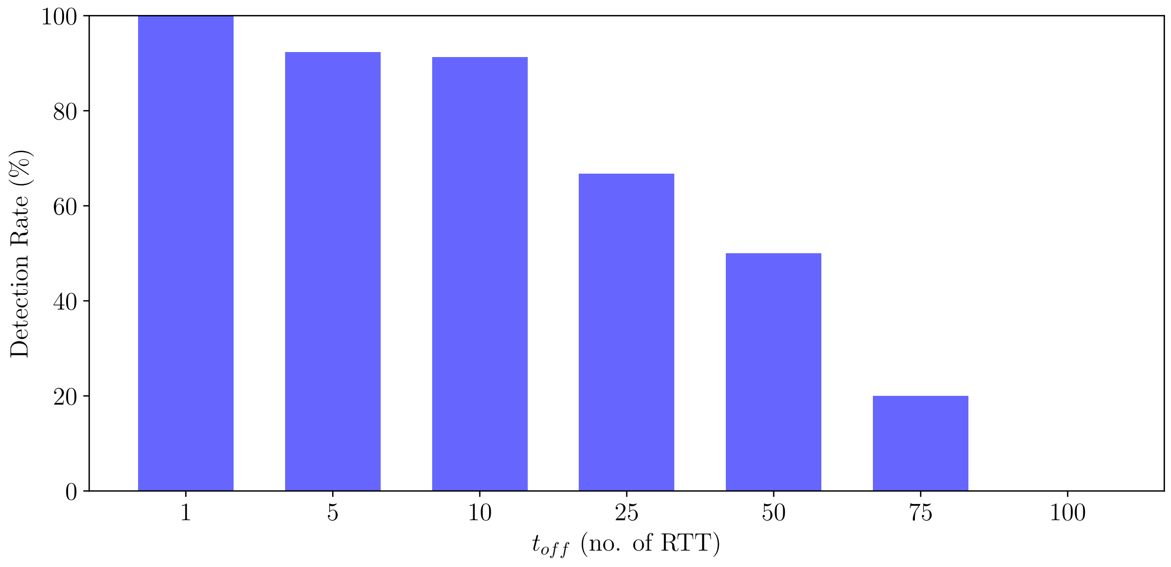


Figure 6.14: Detection rate - weak spot injection.

6.5.5 Uncoupled MAC Overhead

In this section two of the aforementioned disadvantages of Uncoupled MAC are addressed. On the one side the additional network utilization is measured including the presence of an attacker and on the other side the overhead on resources based on a CPU and memory measurement required to perform the Uncoupled MAC algorithm.

Network Utilization

Uncoupled MAC and especially self-regulation increases the overhead of network utilization in presence of an attacker since its parameters are dynamically adjusted due to the detection of the mechanisms' violations. In order to show the impact of different attacker behavior on the network overhead, the continuous injection was modified such that the interval between each single packet injection is decreased over runtime simulating an increasing attacker load. Figure 6.15 is illustrating this scenario for an Uncoupled MAC setting of $n_s = 1.30$, $\alpha_s = 2.0$, $q_0 = 0.75$, an underlying basic ICMP traffic with a total packet size of 1008 bytes and an attacker that injects packets after approximately 20 seconds. The range of injection is $[2, 0.25]$ in seconds having a decreasing interval of 0.05. The attack is carried out 10 times showing the averaged curves in Figure 6.15. Until approximately 20 seconds no attacker packets are injected resulting in an overhead of 8%. After approximately 20 seconds the injected attacker packets lead to a steady increase of the total ICMP bandwidth. However, even if Uncoupled MAC detects the injected attacker packets and adjust its n and α values, the overhead stays about the same with a slight increase to 9%. The reason for this are the random MAC and Idle Phases that interfere for the averaged 10 attack measurements. In a real network environment the overhead by Uncoupled MAC including self-regulation would therefore remain nearly constant when considering the mean bandwidth utilization.

It must be noted that the overhead of Uncoupled MAC, even when an attacker is present, depends on two factors. One impact is the security configuration of Uncoupled MAC for instance whether using additional feedback messages or not which would increase or decrease the overhead. For the network utilization measurement no feedback messages are considered. On the other side the packet size of the underlying protocol to be secured plays a major role. Independent of this size, the size of Uncoupled MAC generation packets is fixed, mainly characterized by the 256-bit HMAC. The percentual Uncoupled MAC overhead would therefore increase for a minor basic packet size.

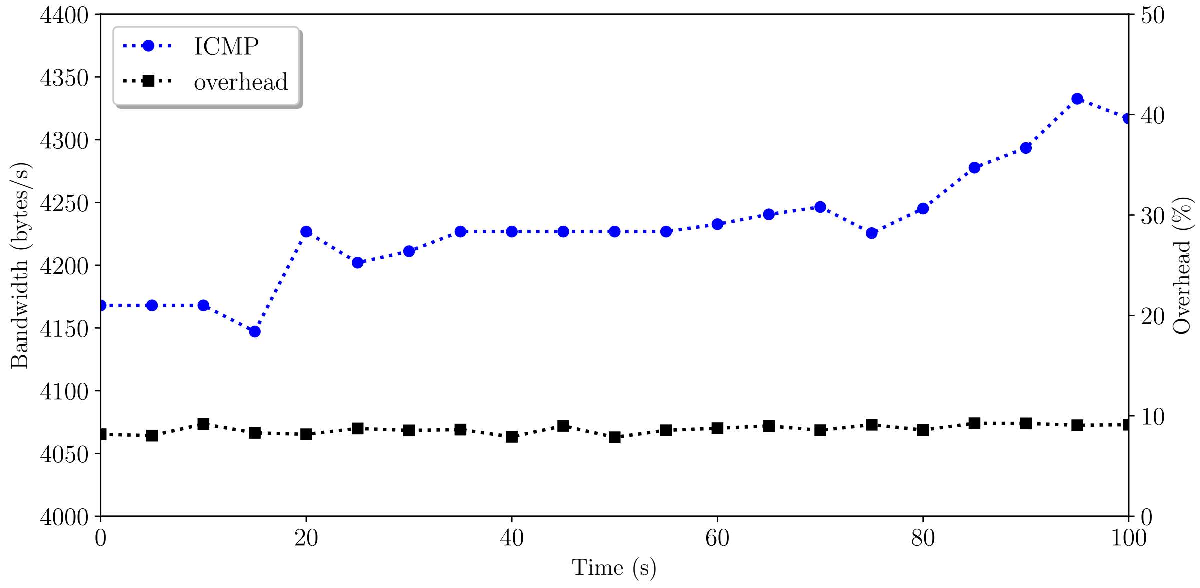


Figure 6.15: Bandwidth overhead in presence of an attacker.

CPU and Memory Utilization

IDSs, especially complex machine learning based anomaly detection algorithms, demand expensive resources [391]. Due to the nature of Uncoupled MAC, a certain resource consumption overhead is present when verifying the integrity and authenticity of a specified q_0 percent of packets. Figure 6.16 shows the CPU (left plot) and memory (right plot) utilization of the self-regulated Uncoupled MAC process with $q_0 = 0.75$ carried out on the Uncoupled MAC Device (1) applying the *psrecord* and the *Massif (Valgrind)* tool. In particular, the memory utilization is split into the heap and stack utilization.

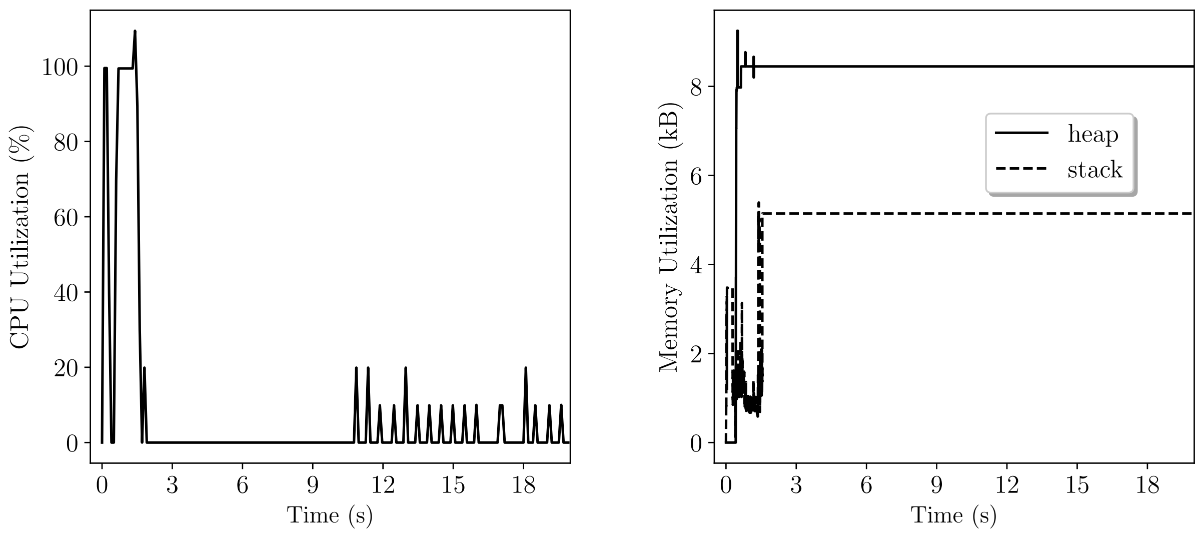


Figure 6.16: CPU (left) and memory (right) utilization of the Uncoupled MAC process.

The figure covers the Setup Phase with the key agreement, one Idle Phase and one MAC Phase in generation mode for the Uncoupled MAC master. The Setup Phase in the current implementation is quite CPU consuming but needs to be done only once for each Uncoupled MAC connection setup ($\approx$ 0-2 s). This phase also results in fluctations of the heap and stack memory. After the Setup Phase, a thread is created performing the HMAC generation and verification which results in a constant heap and stack memory utilization of approximately 8.45 kB (heap) and 5.14 kB (stack) for all subsequent Idle and MAC Phases. The Idle Phases do not demand any CPU load since the master's process suspends for a random time depending on the value of α ($\approx$ 2-10.5 s). In the experiment after approximately 10.5 seconds the MAC Phase starts showing minor peaks for each computed HMAC. The CPU overhead produced by Uncoupled MAC even during MAC Phases is, however, decent with only approximately 10% on average.

7 Conclusion

This chapter firstly provides a general conclusion based on the research presented in this thesis. Furthermore, the chapter revises the main findings with regard to the research questions and considers the strengths and limitations of this work. Finally, we suggest and sketch directions for future research.

7.1 Summary of the Work

Chapter 1, the introduction, illuminated the motivation of the present thesis. In recent times, trends and technologies, such as the Internet of Things, Software-Defined Everything and Artificial Intelligence (AI), have accelerated the increasing interconnection of networking devices. Future IT systems will be exposed to increasing risks due to newly emerging technologies and trends with blurred borders (keyword “IoT-ification”) associated with novel and highly advanced attack possibilities, e.g., malware driven by AI. These circumstances have led to a steady blurring of boundaries between diverse application domains and are characterized by high-volume, high-speed and high-dimensional Streaming Data (SD) posing enormous challenges for applied security mechanisms. Cryptography alone can not postulate protection for network communication in the future therefore advanced and intelligent mechanisms need to be developed for cyber defense. Incident handling is the process that involves the detection, analysis and response of and to security-related incidents. However, as provided with the problem statement, Intrusion Detection System (IDS)s have various limitations, artificial neural networks cannot be applied, e.g., due to inherent resource constraints and streaming data introduces challenges for contemporary security mechanisms. The generic framework called ***An**omaly-based Incident **D**etection and **R**esponse **S**ystem* (**ANDERS**) was introduced on which four major research questions for the thesis were defined. According to [24], the most dangerous attacks only occur rarely such that Machine Learning (ML) based Outlier Detection (OD) will seemingly play a crucial role in future network security. Thus, this thesis has aimed for the following overall research goal and defines four major contributions in Chapter 1.

Research Goal: *Improve Outlier Detection for Data Streams to Enhance Computer Network Security.*

Relevant background knowledge and state-of-the-art research on the topics (i) incident detection, (ii) incident analysis and (iii) incident response was provided for the reader in Chapter 2. For each of the four contributions, the dedicated chapters make up the main part of the thesis that covers the aforementioned topics. Thus, Chapter 3 introduced the ***U**nsupervised **F**eature **S**election for **S**treaming **O**utlier **D**etection* (**UFSSOD**) algorithm, an online capable Feature Selection (FS) algorithm for the purpose of OD, which improves the input quality of the detection mechanism. An improved OD algorithm was discussed in Chapter 4 with the ***P**erformance **C**ounter-**B**ased **iForest*** (**PCB-iForest**) algorithm. Chapter 5 introduced the ***S**treaming **O**utlier **A**nalysis and **A**ttack **P**attern*

***R**ecognition* (**SOAAPR**) framework that improves the output quality of OD mechanisms by performing streaming Alert Correlation (AC) and introduces three types of fingerprint-like signatures for attack scenario representation and comparison. Outlier detection can be improved by incorporating the feedback of their outcome, which was shown by the so-called ***Uncoupled M**essage **A**uthentication **C**ode* (**Uncoupled MAC**) algorithm, a self-regulating protection-based cryptographic technique equipped with IDS-functionality. A short summary of this thesis, the revision of the research questions and a glance at the future work of ongoing research finalizes the thesis in this conclusion - Chapter 7.

7.2 Revising the Research Questions

In this section, we are providing the answers for the four research questions (RQ) defined in the introduction - Chapter 1. After each answer, the corresponding research contribution (RC) is provided to reach the thesis' overall research goal. All four research contributions have been published or are submitted journal articles.

RQ 1: *How can unsupervised feature selection be applied on streaming data for the purpose of outlier detection?*
RC 1: *Unsupervised Feature Selection for Outlier Detection on Streaming Data to Enhance Network Security.*

We illuminated the necessity of unsupervised FS for OD in SD for domains, e.g., intrusion detection, in network security that are ever-increasingly facing high-volume and high-dimensional data that need to be processed in almost real-time. With the extensive review on FS approaches for either (i) SD or (ii) OD, we pointed out that, to the best of our knowledge, the proposed Unsupervised Feature Selection for Streaming Outlier Detection, called UFSSOD, is the first method of its kind to fill the research gap by providing a solution bridging (i) and (ii). Two application scenarios of UFSSOD, together with online capable OD methods, are discussed along with UFSSOD's operation and functionality. This also includes its ability to provide the amount of top-performing features by clustering the score values which is often circumvented in the literature by setting a pre-defined number. Extensive measurements were conducted on 15 real-world data sets by applying multiple feature subsets to 6 widely accepted off-the-shelf online OD methods obtained from UFSSOD and state-of-the-art competitors FSDS as a representative of (i) and IBFS of (ii). The evaluation examines the alleviation of the negative effect brought by irrelevant features of the outlier detectors, considering both, the classification and computational performance. The discussion of the results pointed out the non-applicability of FSDS for OD because it doesn't achieve reliable and satisfactory results, independent of its cluster parameter k. In comparison to (offline) IBFS, UFSSOD achieves at least comparable results while operating in an online fashion. It was able to decrease the average runtime of individual classifiers by approximately 12% and improve the $F1$ by 14% on average. UFSSOD is also evaluated in a true online setting by providing a feature subset in line with xStream for each sample and to Loda $_{\text{Two Hist.}}$ using a windowed approach. In this setting, applying UFSSOD yields better results than the bare versions of the OD methods operating on full dimension. For instance, UFSSOD could improve the $F1$ up to 45% for Loda $_{\text{Two Hist.}}$ while reducing the average runtime by 22% for individual data sets.

RQ 2: *How can a flexible framework for unsupervised online outlier detection be designed to provide an online scoring functionality for feature importance?*
RC 2: *On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data.*

Over the past few years, the continuous increase of high-volume, high-speed and high-dimensional unlabeled streaming data has pushed the development of anomaly detection schemes across multiple domains. These require an efficient unsupervised and online processing ability capable of dealing with challenges such as concept drift. The most popular state-of-the-art OD methods for streaming data are discussed in this thesis, with a focus on algorithms based on the widely known Isolation Forest (iForest), and compared with thoroughly engineered requirements pointing out the lack of a flexible, robust and future-oriented solution.

Thus, this thesis introduces and discusses a novel framework called PCB-iForest that "wraps around", generically, any ensemble-based online OD method. However, due to its popularity, the main focus lies on the incorporation of iForest-based methods for which we present two variants. PCB-iForest$_{\mathrm{EIF}}$ is (to the best of our knowledge) the first application of the iForest improvement called Extended Isolation Forest on streaming data. PCB-iForest$_{\mathrm{IBFS}}$ applies a recently proposed feature importance scoring functionality designed for static data, which is adapted to function in a streaming fashion. We are providing details of PCB-iForest's core functionality based on performance counters assigned to each ensemble component in order to favor or penalize well or poorly performing components. The latter will be replaced if concept drifts are detected by newly built components based on samples within a sliding window. Since drift detection is crucial in regularly updating our model where required, we rely on a recently proposed method denoted as NDKSWIN but are open to any multi-dimensional data-centric method.

Our extensive evaluation firstly evaluates the drift detection functionality which shows that NDKSWIN is able to detect concept drifts and, even if afflicted by some additional detections, regularly updates PCB-iForest. Comprehensively comparing both PCB-iForest methods with state-of-the-art competitors points out the superiority of our method in most cases. In terms of the Area Under the ROC Curve, we achieve the best results on four data sets used by online iForest-based competitors. On the multi-disciplinary ODDS, PCB-iForest clearly outperforms 9 competitors in approximately 50% of the data sets while achieving comparable results of 80% with respect to the $F1$ metric and its tradeoff with the average runtime. Utilizing the four most efficient competitors and our PCB-iForest variants on four security-related UNSW-NB15 data sets again proves the superiority of our approach by achieving the highest $F1$ excluding the poor performance of all classifiers on one data set while still being comparable to the extremely fast processing Loda algorithm.

RQ 3: *How can the output of online outlier detection mechanisms be exploited to characterize and compare novel attack patterns?*
RC 3: *Exploiting the Outcome of Outlier Detection for Novel Attack Pattern Recognition on Streaming Data.*

With the advent of anomaly-based IDS to detect malicious activity in streaming network data, especially online-capable unsupervised OD algorithms, novel techniques for AC methods are increasingly needed. Those must be capable of mining information from the outcome of OD algorithms without knowledge information such as the intrusion type, which is typically provided by misuse-based IDS. Alerts are continuously generated from the high-volume, high-speed and high-dimensional streaming data in the form of an alert stream, which might be afflicted by a high amount of False Positive (FP) and False Negative (FN) detections. Human experts can no longer be expected to handle this massive amount of alerts and certain types of attacks are likely to be overlooked.

Thus, a novel framework called SOAAPR has been introduced that is able to deal with outcomes from online OD algorithms in multiple configuration settings to improve the input quality for the streaming alert correlation/clustering module in terms of reducing FPs and mitigating FNs. For this, alerts are equipped, apart from the typical intrinsic attributes such as IP, port or timestamp information, with feature importance scores and the respective outlier scores. The core component of SOAAPR clusters the streaming alerts according to their attributes' similarity. The resulting clusters can evolve over time and, if not discarded in the case of irrelevant clusters, can be saturated, meaning that these clusters potentially capture attack scenarios. In order to ensure a short response time for security analysts, the alarms of those clusters are promptly fed into a consecutive module that generates three types of signatures, denoted sig_{com}, sig_{attr} and sig_{temp}. These fingerprint-like characteristics represent the attack scenarios in terms of the attack's communication behavior, their cause in the data's features and the temporal sequence of associated alerts. The signatures can then be used to find similarities between attack scenarios or look for similar signatures that can be collected in a knowledge base or, e.g., shared with other companies or institutes.

The evaluation leveraging of the widely-known CICIDS2017 and CSE-CIC-IDS2018 datasets is split into two parts. First, the streaming clustering module of SOAAPR is compared against a graph-based competitor, the alert clustering component of GAC [337]. We rely on four different metrics, the completeness, the soundness, the Jaccard index and the elapsed time for alert processing the data sets as a representative for computational performance. Since GAC's complexity increases with the graph size, chunk processing with 5,000 alerts had to be applied. SOAAPR is configured with hyperparameters that are, for the sake of equal comparison, similar to GAC and are even set to capture attack scenarios with a low number of associated alerts. Second, the signaturing functionality of SOAAPR, generating and comparing sig_{com}, sig_{attr} and sig_{temp}, is the subject of our experimental investigations. Thus, we investigate the similarity between attack scenarios' signatures and their average computing time in our experiments.

The discussion of results for alert clustering reveals that SOAAPR reliably clusters attack scenarios as well as GAC while being significantly more efficient in terms of processing time. In the best case SOAAPR clusters DoS-attack scenarios faster by a factor of 190 compared to GAC. Although it may be that attack scenarios are split into a couple of clusters, even in the worst case for the web attacks, SOAAPR reduces the associated 2,180 alerts into only 24 clusters, which is a compression factor of magnitude 91. Adjusting SOAAPR's hyperparameters even aids to notably reduce so-called ghost-clusters, which are mainly caused by FPs. Given the multitude of different attacks and their characteristics, we propose to leverage multiple SOAAPR instances each parameterized with hyperparameters specifically for each attack type for real-world application. A holistic evaluation of the effects of FPs and FNs on the clustering result is part of further work.

The discussion of results with respect to SOAAPR's signaturing reveals that all three signature types generally can be used to characterize attacks and find similarities between attack categories. For sig_{com}, an attack scenario similarity of up to 95.05% could be obtained. However, its processing time shows exponential behavior over the number of alerts. In order to compute and compare sig_{attr}, the supervised Random Forests classifier has been utilized to generate feature importance scores in the operation modes *single system - multiple algorithms*, in which multiple classifiers work in parallel to improve alert quality. While yielding strong similarity between comparable attack scenarios of the same data set, similarity could even be shown with sig_{attr} between similar scenarios of different data sets by investigating the curves' curvature depending on the number of the top-performing features. The results comparing the timing behavior of a total number of 34 attack scenarios, captured with sig_{temp}, yields a strong similarity especially between similar attack scenarios from the same data sets. In contrast to sig_{com}, the processing time of sig_{attr} and sig_{temp} is significantly faster and shows linear behavior. Overall, some congruent attack scenarios from different data sets showed weak similarity for sig_{temp}, which is mainly reasoned in the poor quality of the available data sets.

RQ 4: *How can a cryptographic scheme be leveraged to function as a detection mechanism and have its feedback be incorporated in improving performance over runtime as part of response functionality?*
RC 4: *A Resource-Preserving Self-Regulating Uncoupled MAC Algorithm to be Applied in Incident Detection.*

In this thesis, Uncoupled MAC has been presented as a protection-based security technique improved and extended to also work as a detection-based method. By the sampling approach having MAC Phases and Idle Phases, a certain security level can be adjusted with a balanced overhead on resource consumption and network utilization. The proposed self-regulation mode examines a specifiable amount of packets in network environments with security incidents happening over the average runtime. However, the more malicious activities are detected, the more active Uncoupled MAC becomes, thereby showing behavior of a partially working IDS component that incorporates feedback. Applying self-regulation, malicious actions can be detected quite fast and reliably compared to classical Uncoupled MAC.

Securing layer-2 protocols is either complex to implement or popular IDS solutions do not cover data link layer detection. Having a layer-2 support and the functionality to work integrated in devices, Uncoupled MAC enables completely new fields of applications in a holistic approach. As a combination of a protection and detection-based method it can be applied as retrofitted security add-on in dedicated embedded plug-in devices or as a software application running transparently in the background of end-devices or network elements while utilizing tolerable resource consumption over the average runtime.

7.3 Future Work & Research Perspectives

In the final section of this thesis, an overview of possible directions for future research is provided as our research efforts are far from being accomplished. This overview is split into four main parts with respect to the four contributions related to UFSSOD, PCB-iForest, SOAAPR and Uncoupled MAC.

1. **UFSSOD - Chapter 3**

   - **Future Work 1.1:** As assumed, UFSSOD works better if the outliers in the data set tend to occur in the same features, which is mostly the case, as stated, in the network security domain. Even passing the stress test on data sets that do not meet this presumption, as part of future work, UFSSOD will be thoroughly examined on more domain-specific data sets, e.g., CSE-CIC-IDS2018.
   - **Future Work 1.2:** Since in the current configuration measurements only relied on the γ_{min} functionality with respect to Algorithm 3, a part of our future work will be comparing it with the *distance* functionality which tends to result in more features leading to higher computational costs, but might improve the classification task even further. Additionally, a more thorough testing of the outlier scaling and its combination across multiple online detection approaches will be performed.
   - **Future Work 1.3:** Inspired by approaches used in CINFO or ODEFS and the nature of UFSSOD to provide outlier candidates, in future work we will examine the possibility of applying supervised FS, similarly to sparse Lasso regression in CINFO, based on pseudo-labels obtained from UFSSOD's outlier candidates. Thus, not only will the pure nature of Loda 's functioning form the basis for FS but it may also be improved by shrinking the solution, resulting in a number of features that are not correlated to the outlier score.
   - **Future Work 1.4:** To even more alleviate the circumstance that FS is performed independently of the subsequent online OD method, possibly yielding a suboptimal and biased solution for OD, we might also include outlier candidates from the Unsupervised Online OD module. Instead of performing iterative validations to reduce loss in an offline setting having all data objects available, we will research possibilities to do so in a slimmed-down online fashion by exploiting the strength of the theoretically infinite running time of SD.

2. **PCB-iForest - Chapter 4**

   - **Future Work 2.1:** PCB-iForest's current implementation faces two limitations which will be addressed as part of future work taking advantage of the frameworks' flexible design to incorporate any ensemble-based OD algorithm and to replace the drift detection method. Thus, firstly, we will focus on the integration of Loda $_{\text{Two Hist.}}$ into PCB-iForest, since (i) its fast-processing runtime and (ii) its classification results could be further improved by replacing the pair of alternating windows with our approach. Therefore the set of histograms will only partially be replaced if a concept drift is detected and not completely be discarded after each window.
   - **Future Work 2.2:** Secondly, drift detection is a crucial component of our approach since it must reliably detect concept drifts and should not be prone to a high number of false detections because this will degrade both the classification and computational performance. Therefore, our future work will (i) focus on the improvement of NDKSWIN, since it can be prone to the multiple comparisons problem, and (ii) evaluate the application of other multi-dimensional data-centric algorithms within PCB-iForest.
   - **Future Work 2.3:** More research will also be part of PCB-iForest$_{\text{IBFS}}$'s feature scoring functionality, since, even if able to score and rank the best-performing features during runtime, determining the "optimum" number of features that should be used as a feature subset is still an open issue. Although the subsets with 25%-75% of

the whole feature set could achieve promising results, it is highly dependent on the data source and algorithm type. Thus, a method will be developed that determines the best feature subset based on the analysis of the feature scores. For this, we can refer to the methods developed in UFSSOD.

3. **SOAAPR - Chapter 5**

   - **Future Work 3.1:** With respect to sig_{attr}, up to now, we relied on the supervised RF - SHapley Additive exPlanations - feature importance scoring functionality for better result interpretability. However, in future evaluation, we want to replace RF with an online unsupervised OD algorithm equipped with this functionality, such as Loda [119] or PCB-iForest [364].
   - **Future Work 3.2:** A more holistic evaluation including more intensive measurements is also part of future work. This will include experiments of the effects of FPs and FNs on the clustering result and the evaluation of SOAAPR on other data sets that provide, for instance, better timestamping for sig_{temp} evaluation or contains IP and port information for sig_{com} evaluation. We showed that the ambitious aim to exploit the outcome of OD algorithms in order to generate attack patterns generally works. Nevertheless, in further work we also want to investigate the impact of introducing FPs and FNs on signature comparison.
   - **Future Work 3.3:** Since SOAAPR's streaming clustering is sensitive to the timing behavior of an attack and its associated alerts, attack scenarios might be split into multiple tiny clusters if SOAAPR's hyperparameters are not properly set. Thus, further research will be part of investigating the possibility of combining split clusters if their overall similarity is high, e.g., by measuring the cluster centroids distances or leveraging the comparison of signatures sig_{com}, sig_{attr} and sig_{temp}. Split clusters with highly similar signatures might likely be assigned to the same attack scenario. Thus, not only may the clustering result be improved but multi-stage attack detection might also be enabled for which SOAAPR initially was not designed. This would be similar to the Intrusion Session Rebuilding component of IACF [352] or the Attack Interconnection phase of GAC .
   - **Future Work 3.4:** An intelligent adversary might inject bogus alerts to camouflage its actual attack by deceiving SOAAPR's clustering. Further work should, on one hand, focus on the solutions' robustness against attacks. On the other hand, in order to lower the determinism for an attacker to not exploit SOAAPR's time boundary, a certain amount of jitter can be introduced, e.g., by leveraging a similar mechanism as proposed with Uncoupled MAC.

4. **Uncoupled MAC - Chapter 6**

   - **Future Work 4.1:** The concept of self-regulation used in Uncoupled MAC could be extended to other IDS systems performing sampling in future research work and a more intelligent interaction, for instance, by applying alert analysis techniques should be established, as proposed with SOAAPR. The basis for this is given by extending Uncoupled MAC with IDMEF and might easily be changed to IDEA, which is more lightweight.
   - **Future Work 4.2:** An even more intelligent sampling mechanism based on a combination of statistics, local, global and feedback from an IDS cluster, as proposed in [378], could be added to the self-regulation in order to further improve the interaction of detection and protection approaches with resource conservation.

- **Future Work 4.3:** Especially targeted towards future-orientated IoT-enabled applications, Uncoupled MAC shall be implemented in an even more lightweight variant applying for instance the lightweight MAC Chaskey [392] or LMAC [393] for MAC generation and verification together with, e.g., the MQTT protocol for lightweight Uncoupled MAC partner communication. By using a more lightweight MAC function, the additional overhead not only from a CPU but also from a memory point of view can be reduced. From the bandwidth utilization perspective, using for instance LMAC, can decrease the load by using the 64-bit digest instead of the 256-bit HMAC mainly determining the generation's message size.
- **Future Work 4.4:** Since Uncoupled MAC, as a cryptographic scheme by nature, is only a possibility for incident detection providing basic IDS-functionality, it might be of interest to compare it with other attack detection mechanisms by examining the trade-off between detection capability and resource consumption. Part of further work is therefore to set up an appropriate evaluation environment, e.g., [394, 395], with, among others, a larger protocol and network data diversity in order to properly train the models for anomaly-based ML algorithms, for instance Loda or PCB-iForest, and integrating known attack scenarios such that misuse-based IDS can also be compared.

In the introduction, the generic framework denoted ANDERS was proposed, which consists of components that leverage different capabilities in terms of incident detection, analysis and response. Thus, it may be able to constitute to a promising solution for an automated incident handling system in a holistic cyber defense life cycle. This thesis answered four significant research questions with regard to anomaly-based detection, in particular ML-based OD. We deem that the implementation of fully operational ANDERS prototypes integrated into a Software-Defined Networking permeated next-generation infrastructure, and in consideration of state-of-the-art solutions such as a misuse-based IDS, is the natural continuation of this research. Although this thesis has contributed to the enhancement of computer network security through improved OD for data streams, we end this dissertation by reminding Eugene Howard Spafford's famous quote.

The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards - and even then I have my doubts.

Bibliography

[1] T. R. Vittor, T. Sukumara, S. D. Sudarsan, and J. Starck, "Cyber security - security strategy for distribution management system and security architecture considerations," in *2017 70th Annual Conference for Protective Relay Engineers (CPRE)*, pp. 1–6, IEEE, 2017.

[2] G. Press, "60 cybersecurity predictions for 2019," *Forbes https://www.forbes.com/sites/gilpress/2018/12/03/60-cybersecurity-predictions-for-2019 (online, accessed 22 July 2019)*, 2018.

[3] Z. Inayat, A. Gani, N. B. Anuar, S. Anwar, and M. K. Khan, "Cloud-based intrusion detection and response system: Open research issues, and solutions," *Arab. J. Sci. Eng.*, vol. 42, no. 2, pp. 399–423, 2017.

[4] S. Anwar, J. Mohamad Zain, M. F. Zolkipli, Z. Inayat, S. Khan, B. Anthony, and V. Chang, "From intrusion detection to an intrusion response system: Fundamentals, requirements, and future directions," *Algorithms*, vol. 10, no. 2, p. 39, 2017.

[5] A. Shameli-Sendi, M. Cheriet, and A. Hamou-Lhadj, "Taxonomy of intrusion risk assessment and response system," *Comput. Secur.*, vol. 45, pp. 1–16, 2014.

[6] M. Heigl, L. Doerr, A. Almaini, D. Fiala, and M. Schram, "Incident reaction based on intrusion detections' alert analysis," in *2018 International Conference on Applied Electronics (AE)*, pp. 1–6, IEEE, 2018.

[7] P. Nespoli, D. Papamartzivanos, F. Gomez Marmol, and G. Kambourakis, "Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks," *IEEE Commun. Surv. Tutor.*, vol. 20, no. 2, pp. 1361–1396, 2018.

[8] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi, "A language-based intrusion detection approach for automotive embedded networks," *Int. J. Embed. Syst.*, vol. 10, pp. 1–12, 2018.

[9] B. Arrington, L. Barnett, R. Rufus, and A. Esterline, "Behavioral modeling intrusion detection system (bmids) using internet of things (iot) behavior-based anomaly detection via immunity-inspired algorithms," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–6, 2016.

[10] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *CoRR*, vol. abs/1901.03407, 2019.

[11] A. Drewek-Ossowicka, M. Pietrołaj, and J. Rumiński, "A survey of neural networks usage for intrusion detection systems," *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 1, pp. 497–514, 2021.

[12] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Appl. Sci. (Basel)*, vol. 9, no. 20, p. 4396, 2019.

[13] A. Pektaş and T. Acarman, "Deep learning to detect botnet via network flow summaries," *Neural Comput. Appl.*, vol. 31, no. 11, pp. 8021–8033, 2019.

[14] S. Potluri and C. Diedrich, "Accelerated deep neural networks for enhanced intrusion detection system," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, IEEE, 2016.

[15] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2018.

[16] E. Min, J. Long, Q. Liu, J. Cui, and W. Chen, "TR-IDS: Anomaly-based intrusion detection through text-convolutional neural network and random forest," *Secur. Commun. Netw.*, vol. 2018, pp. 1–9, 2018.

[17] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, (Red Hook, NY, USA), pp. 4768–4777, Curran Associates Inc., 2017.

[18] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you?: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, (New York, New York, USA), ACM Press, 2016.

[19] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE trans. emerg. top. comput. intell.*, vol. 2, no. 1, pp. 41–50, 2018.

[20] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018.

[21] I. M. Coelho, V. N. Coelho, E. J. d. S. Luz, L. S. Ochi, F. G. Guimarães, and E. Rios, "A GPU deep learning metaheuristic based model for time series forecasting," *Appl. Energy*, vol. 201, pp. 412–418, 2017.

[22] B. Subba, S. Biswas, and S. Karmakar, "A neural network based system for intrusion detection and attack classification," in *2016 Twenty Second National Conference on Communication (NCC)*, pp. 1–6, IEEE, 2016.

[23] K. Zhang, F. Zhao, S. Luo, Y. Xin, H. Zhu, and Y. Chen, "Online intrusion scenario discovery and prediction based on hierarchical temporal memory (HTM)," *Appl. Sci. (Basel)*, vol. 10, no. 7, p. 2596, 2020.

[24] H. Zhang, K. Nian, T. F. Coleman, and Y. Li, "Spectral ranking and unsupervised feature selection for point, collective, and contextual anomaly detection," *Int. J. Data Sci. Anal.*, vol. 9, no. 1, pp. 57–75, 2020.

[25] C. Mironeanu, A. Archip, C.-M. Amarandei, and M. Craus, "Experimental cyber attack detection framework," *Electronics (Basel)*, vol. 10, no. 14, p. 1682, 2021.

[26] H. Hindy, D. Brosset, E. Bayne, A. K. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens, "A taxonomy of network threats and the effect of current datasets on intrusion detection systems," *IEEE Access*, vol. 8, pp. 104650–104675, 2020.

[27] A. Sadighian, "Intrusion detection from heterogenous sensors," *Ph.D. thesis, Polytechnique Montreal*, 2015.

[28] A. A. Ramaki, A. Rasoolzadegan, and A. G. Bafghi, "A systematic mapping study on intrusion alert analysis in intrusion detection systems," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 1–41, 2018.

[29] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, *Network traffic anomaly detection and prevention: Concepts, techniques, and tools.* Cham, Switzerland: Springer International Publishing, 1 ed., 2017.

[30] A. Mahfouz, A. Abuhussein, D. Venugopal, and S. Shiva, "Ensemble classifiers for network intrusion detection using a novel network attack dataset," *Future internet*, vol. 12, no. 11, p. 180, 2020.

[31] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017.

[32] N. Reunanen, T. Räty, J. J. Jokinen, T. Hoyt, and D. Culler, "Unsupervised online detection and prediction of outliers in streams of sensor data," *Int. J. Data Sci. Anal.*, vol. 9, no. 3, pp. 285–314, 2020.

[33] J. Ma, Z.-T. Li, and W.-M. Li, "Real-time alert stream clustering and correlation for discovering attack strategies," in *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, pp. 379–384, IEEE, 2008.

[34] M. U. Togbe, M. Barry, A. Boly, Y. Chabchoub, R. Chiky, J. Montiel, and V.-T. Tran, "Anomaly detection for data streams based on isolation forest using scikit-multiflow," in *Computational Science and Its Applications – ICCSA 2020*, pp. 15–30, Cham: Springer International Publishing, 2020.

[35] I. Kovačević, S. Groš, and K. Slovenec, "Systematic review and quantitative comparison of cyberattack scenario detection and projection," *Electronics (Basel)*, vol. 9, no. 10, p. 1722, 2020.

[36] ISO/IEC 27000:2018, "Information technology – security techniques – information security management systems – overview and vocabulary," *International Organization for Standardization, `https://www.iso.org/standard/73906.html` (online, accessed 15 June 2019)*, 2018.

[37] A. F. Murillo Piedrahita, V. Gaur, J. Giraldo, A. A. Cardenas, and S. J. Rueda, "Leveraging software-defined networking for incident response in industrial control systems," *IEEE Softw.*, vol. 35, no. 1, pp. 44–50, 2018.

[38] M. Doering and M. Wagner, "Retrofitting SDN to classical in-vehicle networks: SDN4CAN," *Universität Tübingen, `http://hdl.handle.net/10900/78141`*, 2017.

[39] K. Halba and C. Mahmoudi, "In-vehicle software defined networking: An enabler for data interoperability," in *Proceedings of the 2nd International Conference on Information System and Data Mining*, (New York, NY, USA), ACM, 2018.

[40] Z. Khan, M. Chowdhury, M. Islam, C. Huang, and M. Rahman, "In-vehicle false information attack detection and mitigation framework using machine learning and software defined networking," *CoRR*, vol. abs/1906.10203, 2019.

[41] A. Alioua, S.-M. Senouci, and S. Moussaoui, "DSDiVN: A distributed software-defined networking architecture for infrastructure-less vehicular networks," in *Innovations for Community Services*, pp. 56–67, Cham: Springer International Publishing, 2017.

[42] C. Jiacheng, Z. Haibo, Z. Ning, Y. Peng, G. Lin, and S. Xuemin, "Software defined internet of vehicles: architecture, challenges and solutions," *J. Commun. Inf. Netw.*, vol. 1, no. 1, pp. 14–26, 2016.

[43] W. B. Jaballah, M. Conti, and C. Lal, “A survey on software-defined vanets: Benefits, challenges, and future directions,” *CoRR*, vol. abs/1904.04577, 2019.

[44] A. Mahmood, W. Zhang, and Q. Sheng, “Software-defined heterogeneous vehicular networking: The architectural design and open challenges,” *Future internet*, vol. 11, no. 3, p. 70, 2019.

[45] T. Häckel, P. Meyer, F. Korf, and T. C. Schmidt, “Software-defined networks supporting time-sensitive in-vehicular communication,” in *89th IEEE Vehicular Technology Conference, VTC Spring 2019, Kuala Lumpur, Malaysia, April 28 - May 1, 2019*, pp. 1–5, IEEE, 2019.

[46] M. O. Kalinin, V. M. Krundyshev, and P. V. Semianov, “Architectures for building secure vehicular networks based on SDN technology,” *Autom. Contr. Comput. Sci.*, vol. 51, no. 8, pp. 907–914, 2017.

[47] R. Shrestha, R. Bajracharya, and S. Y. Nam, “Challenges of future VANET and cloud-based approaches,” *Wirel. Commun. Mob. Comput.*, vol. 2018, pp. 1–15, 2018.

[48] A. A. Khan, M. Abolhasan, and W. Ni, “5G next generation VANETs using SDN and fog computing framework,” in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–6, IEEE, 2018.

[49] A. Tsuchiya, F. Fraile, I. Koshijima, A. Ortiz, and R. Poler, “Software defined networking firewall for industry 4.0 manufacturing systems,” *J. Ind. Eng. Manag.*, vol. 11, no. 2, p. 318, 2018.

[50] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren, and C. Mahmoudi, “Fog computing conceptual model,” tech. rep., National Institute of Standards and Technology, Gaithersburg, MD, 2018.

[51] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, IEEE, 2008.

[52] M. U. Togbe, Y. Chabchoub, A. Boly, M. Barry, R. Chiky, and M. Bahri, “Anomalies detection using isolation in concept-drifting data streams,” *Computers*, vol. 10, no. 1, p. 13, 2021.

[53] CESNET, “IDEA - Intrusion Detection Extensible Alert,” *https://idea.cesnet.cz/en/index (online, accessed 22 July 2019)*, 2017.

[54] S. Haas, F. Wilkens, and M. Fischer, “Efficient attack correlation and identification of attack scenarios based on network-motifs,” in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–11, IEEE, 2019.

[55] S. C. Sundaramurthy, L. Zomlot, and X. Ou, “Practical ids alert correlation in the face of dynamic threats,” in *Proceedings of the International Conference on Security and Management (SAM)*, p. 1, Citeseer, 2011.

[56] M. Heigl, M. Schramm, and D. Fiala, “A lightweight quantum-safe security concept for wireless sensor network communication,” in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 906–911, IEEE, 2019.

[57] E. Çayirci and C. Rong, *Security in wireless ad hoc and sensor networks.* Chichester, UK: John Wiley & Sons, Ltd, 2009.

[58] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A network security monitor," in *Proceedings. 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 296–304, IEEE, 1990.

[59] F. Sabahi and A. Movaghar, "Intrusion detection: A survey," in *2008 Third International Conference on Systems and Networks Communications*, pp. 23–26, IEEE, 2008.

[60] K. A. Al-Utaibi and E.-S. M. El-Alfy, "Intrusion detection taxonomy and data preprocessing mechanisms," *J. Intell. Fuzzy Syst.*, vol. 34, no. 3, pp. 1369–1383, 2018.

[61] M. Bijone, "A survey on secure network: Intrusion detection & prevention approaches," *Am. J. Inf. Syst.*, vol. 4, no. 3, pp. 69–88, 2016.

[62] E. Hodo, X. J. A. Bellekens, A. W. Hamilton, C. Tachtatzis, and R. C. Atkinson, "Shallow and deep networks intrusion detection system: A taxonomy and survey," *CoRR*, vol. abs/1701.02145, 2017.

[63] K. Khan, A. Mehmood, S. Khan, M. A. Khan, Z. Iqbal, and W. K. Mashwani, "A survey on intrusion detection and prevention in wireless ad-hoc networks," *J. Syst. Arch.*, vol. 105, no. 101701, p. 101701, 2020.

[64] A. N. Jaber, M. F. Zolkipli, H. A. Shakir, and M. R. Jassim, "Host based intrusion detection and prevention model against DDoS attack in cloud computing," in *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 241–252, Cham: Springer International Publishing, 2018.

[65] D. Fallstrand and V. Lindstroem, "Applicability analysis of intrusion detection and prevention in automotive systems," *Master's thesis, Computer Systems and Networks, Chalmers University of Technology Goteborg*, 2015.

[66] Z. Inayat, A. Gani, N. B. Anuar, M. K. Khan, and S. Anwar, "Intrusion response systems: Foundations, design, and challenges," *J. Netw. Comput. Appl.*, vol. 62, pp. 53–74, 2016.

[67] M.-Y. Su, "Using clustering to improve the KNN-based classifiers for online anomaly network traffic identification," *J. Netw. Comput. Appl.*, vol. 34, no. 2, pp. 722–730, 2011.

[68] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in internet of things," *J. Netw. Comput. Appl.*, vol. 84, pp. 25–37, 2017.

[69] R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, and Y.-L. He, "Fuzziness based semi-supervised learning approach for intrusion detection system," *Inf. Sci. (Ny)*, vol. 378, pp. 484–497, 2017.

[70] A. Taylor, "Anomaly-based detection of malicious activity in in-vehicle networks," *Ph.D. thesis, University of Ottawa*, 2017.

[71] I. Butun, S. D. Morgera, and R. Sankar, "A survey of intrusion detection systems in wireless sensor networks," *IEEE Commun. Surv. Tutor.*, vol. 16, no. 1, pp. 266–282, 2014.

[72] A. M. Ahmed, "Online network intrusion detection system using temporal logic and stream data processing," *Ph.D. thesis, University of Liverpool*, 2013.

[73] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, and M. Fischer, "Taxonomy and survey of collaborative intrusion detection," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–33, 2015.

[74] Z. Zhang, J. Li, C. Manikopoulos, J. Jorgenson, and J. Ucles, "HIDE: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification," *In Proc. IEEE Workshop on Information Assurance and Security*, pp. 85–90, 2001.

[75] M. Jahnke, "An open and secure infrastructure for distributed intrusion detection sensors," *In Proceedings of the NATO Regional Conference on Communication and Information Systems (RCMCIS'02), Zegrze, Poland*, 2002.

[76] H. Debar, D. Curry, and B. Feinstein, "The intrusion detection message exchange format (idmef)," RFC 4765, RFC Editor, March 2007. `http://www.rfc-editor.org/rfc/rfc4765.txt`.

[77] R. Lupu, R. Badea, and I. C. Mihai, "Agent-based IDMEF alerting infrastructure for distributed intrusion detection and prevention systems: Design and validation," in *2016 International Conference on Communications (COMM)*, pp. 281–284, IEEE, 2016.

[78] W. Hu, J. Gao, Y. Wang, O. Wu, and S. Maybank, "Online adaboost-based parameterized methods for dynamic distributed network intrusion detection," *IEEE Trans. Cybern.*, vol. 44, no. 1, pp. 66–82, 2014.

[79] I. H. Sarker, A. S. M. Kayes, S. Badsha, H. Alqahtani, P. Watters, and A. Ng, "Cybersecurity data science: an overview from machine learning perspective," *J. Big Data*, vol. 7, no. 41, 2020.

[80] F. Iglesias and T. Zseby, "Analysis of network traffic features for anomaly detection," *Mach. Learn.*, vol. 101, no. 1-3, pp. 59–84, 2015.

[81] W. Khreich, E. Granger, A. Miri, and R. Sabourin, "Adaptive ROC-based ensembles of HMMs applied to anomaly detection," *Pattern Recognit.*, vol. 45, no. 1, pp. 208–230, 2012.

[82] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Comput. Commun.*, vol. 35, no. 7, pp. 772–783, 2012.

[83] J. Zhang, C. Chen, Y. Xiang, and W. Zhou, "Semi-supervised and compound classification of network traffic," *Int. J. Secur. Netw.*, vol. 7, no. 4, p. 252, 2012.

[84] J. Zhang and M. Zulkernine, "A hybrid network intrusion detection technique using random forests," in *First International Conference on Availability, Reliability and Security (ARES'06)*, pp. 8 pp.–269, IEEE, 2006.

[85] K. Kuźniar and M. Zając, "Some methods of pre-processing input data for neural networks," *Computer Assisted Methods in Engineering and Science*, vol. 22, pp. 141–151, 2015.

[86] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera, "A survey on data preprocessing for data stream mining: Current status and future directions," *Neurocomputing*, vol. 239, pp. 39–57, 2017.

[87] M.-A. Zöller and M. F. Huber, "Benchmark and survey of automated machine learning frameworks," *J. Artif. Intell. Res.*, vol. 70, pp. 409–472, 2021.

[88] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, (New York, NY, USA), p. 847–855, Association for Computing Machinery, 2013.

[89] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pp. 1–6, IEEE, 2009.

[90] A. Thomas, V. Feuillard, A. Gramfort, and S. Clémençon, "Learning hyperparameters for unsupervised anomaly detection," in *ICML 2016 Anomaly Detection Workshop*, 2016.

[91] J. Singh Malik, P. Goyal, and M. K. Sharma, "A comprehensive approach towards data preprocessing techniques, & association rules," *Proceedings of the 4th National Conference INDIACom*, pp. 12–21, 2010.

[92] D. Tomar and S. Agarwal, "A survey on pre-processing and post-processing techniques in data mining," *Int. j. database theory appl.*, vol. 7, no. 4, pp. 99–128, 2014.

[93] B. KumarSingh, K. Verma, and A. S. Thoke, "Investigations on impact of feature normalization techniques on classifier's performance in breast tumor classification," *Int. J. Comput. Appl.*, vol. 116, no. 19, pp. 11–15, 2015.

[94] H. Xie, J. Li, and H. Xue, "A survey of dimensionality reduction techniques based on random projection," *CoRR*, vol. abs/1706.04371, 2017.

[95] N. Lim and R. J. Durrant, "Linear dimensionality reduction in linear time: Johnson-lindenstrauss-type guarantees for random subspace," *arXiv: Machine Learning*, 2017.

[96] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," *ACM Comput. Surv.*, vol. 50, no. 4, pp. 1–36, 2017.

[97] M. E. Aminanto, R. Choi, H. C. Tanuwidjaja, P. D. Yoo, and K. Kim, "Deep abstraction and weighted feature selection for WI-fi impersonation detection," *IEEE trans. inf. forensics secur.*, vol. 13, no. 3, pp. 621–636, 2018.

[98] P. Moradi and M. Gholampour, "A hybrid particle swarm optimization for feature subset selection by integrating a novel local search strategy," *Appl. Soft Comput.*, vol. 43, pp. 117–130, 2016.

[99] H. T. Nguyen, S. Petrović, and K. Franke, "A comparison of feature-selection methods for intrusion detection," in *Lecture Notes in Computer Science*, pp. 242–255, Springer Berlin Heidelberg, 2010.

[100] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–45, 2018.

[101] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Proceedings of the 18th International Conference on Neural Information Processing Systems*, (Cambridge, MA, USA), pp. 507–514, MIT Press, 2005.

[102] R. Wieland, A. Kerkow, L. Früh, H. Kampen, and D. Walther, "Automated feature selection for a machine learning approach toward modeling a mosquito distribution," *Ecol. Modell.*, vol. 352, pp. 108–112, 2017.

[103] M. Luo, F. Nie, X. Chang, Y. Yang, A. G. Hauptmann, and Q. Zheng, "Adaptive unsupervised feature selection with structure regularization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 4, pp. 944–956, 2018.

[104] S. Aljawarneh, M. Aldwairi, and M. B. Yassein, "Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model," *J. Comput. Sci.*, vol. 25, pp. 152–160, 2018.

[105] O. Maimon and L. Rokach, eds., *Data mining and knowledge discovery handbook*. Boston, MA: Springer US, 2010.

[106] D. M. Hawkins, *Identification of Outliers*. Dordrecht: Springer Netherlands, 1980.

[107] R. A. Johnson and D. W. Wichern, *Applied multivariate statistical analysis*. Philadelphia, PA: Pearson Education, 3 ed., 1992.

[108] V. Barnett and T. Lewis, *Outliers in Statistical Data*. Chichester, England: John Wiley & Sons, 3 ed., 1994.

[109] G. Pang, L. Cao, L. Chen, and H. Liu, "Unsupervised feature selection for outlier detection by modelling hierarchical value-feature couplings," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 410–419, IEEE, 2016.

[110] P. Thakkar, J. Vala, and V. Prajapati, "Survey on outlier detection in data stream," *International Journal of Computer Applications*, vol. 136, no. 2, pp. 13–16, 2016.

[111] A. Muallem, S. Shetty, J. W. Pan, J. Zhao, and B. Biswal, "Hoeffding tree algorithms for anomaly detection in streaming datasets: A survey," *J. Inf. Secur.*, vol. 08, no. 04, pp. 339–361, 2017.

[112] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "On-line random forests," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pp. 1393–1400, IEEE, 2009.

[113] L. Liu, M. Hu, C. Kang, and X. Li, "Unsupervised anomaly detection for network data streams in industrial control systems," *Information (Basel)*, vol. 11, no. 2, p. 105, 2020.

[114] H. Yao, X. Fu, Y. Yang, and O. Postolache, "An incremental local outlier detection method in the data stream," *Appl. Sci. (Basel)*, vol. 8, no. 8, p. 1248, 2018.

[115] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *CoRR*, vol. abs/1802.09089, 2018.

[116] K. Yu, W. Shi, and N. Santoro, "Designing a streaming algorithm for outlier detection in data mining-an incrementa approach," *Sensors (Basel)*, vol. 20, no. 5, p. 1261, 2020.

[117] S. C. Tan, K. M. Ting, and T. F. Liu, "Fast anomaly detection for streaming data," in *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Two*, pp. 1511–1516, AAAI Press, 2011.

[118] S. Sathe and C. C. Aggarwal, "Subspace outlier detection in linear time with randomized hashing," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 459–468, IEEE, 2016.

[119] T. Pevný, "Loda: Lightweight on-line detector of anomalies," *Mach. Learn.*, vol. 102, no. 2, pp. 275–304, 2016.

[120] E. Manzoor, H. Lamba, and L. Akoglu, "XStream: Outlier detection in feature-evolving data streams," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, (New York, NY, USA), ACM, 2018.

[121] C. C. Aggarwal and S. Sathe, "Theoretical foundations and algorithms for outlier ensembles," *SIGKDD Explor.*, vol. 17, no. 1, pp. 24–47, 2015.

[122] Z. Ding and M. Fei, "An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window," *IFAC proc. vol.*, vol. 46, no. 20, pp. 12–17, 2013.

[123] H. Sun, Q. He, K. Liao, T. Sellis, L. Guo, X. Zhang, J. Shen, and F. Chen, "Fast anomaly detection in multiple multi-dimensional data streams," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 1218–1223, IEEE, 2019.

[124] H. Ma, B. Ghojogh, M. N. Samad, D. Zheng, and M. Crowley, "Isolation mondrian forest for batch and online anomaly detection," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3051–3058, IEEE, 2020.

[125] B. Pfahringer, G. Holmes, and R. Kirkby, "New options for hoeffding trees," in *AI 2007: Advances in Artificial Intelligence*, pp. 90–99, Springer Berlin Heidelberg, 2007.

[126] L. Sun, S. Versteeg, S. Boztas, and A. Rao, "Detecting anomalous user behavior using an extended isolation forest algorithm: An enterprise case study," *arXiv preprint arXiv:1609.06676*, vol. abs/1609.06676, 2016.

[127] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[128] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*, (New York, New York, USA), ACM Press, 2006.

[129] G. Wang, J. Hao, J. Ma, and H. Jiang, "A comparative assessment of ensemble learning for credit scoring," *Expert Syst. Appl.*, vol. 38, no. 1, pp. 223–230, 2011.

[130] B. D, A. Chakrabarti, and D. Midhunchakkaravarthy, "Smart devices threats, vulnerabilities and malware detection approaches: A survey," *Eur. j. eng. res. sci.*, vol. 3, no. 2, p. 7, 2018.

[131] P. Amudha, S. Karthik, and S. Sivakumari, "Classification techniques for intrusion detection - an overview," *International Journal of Computer Applications*, vol. 76, no. 16, pp. 33–40, 2013.

[132] W. Hu, W. Hu, and S. Maybank, "AdaBoost-based algorithm for network intrusion detection," *IEEE Trans. Syst. Man Cybern. B Cybern.*, vol. 38, no. 2, pp. 577–583, 2008.

[133] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas, "On combining classifiers," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 3, pp. 226–239, 1998.

[134] B. Parhami, "Voting algorithms," *IEEE trans. reliab.*, vol. 43, no. 4, pp. 617–629, 1994.

[135] V. J. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artif. Intell. Rev.*, vol. 22, no. 2, pp. 85–126, 2004.

[136] J. Gao, W. Fan, D. Turaga, O. Verscheure, X. Meng, L. Su, and J. Han, "Consensus extraction from heterogeneous detectors to improve performance over network traffic anomaly detection," in *2011 Proceedings IEEE INFOCOM*, pp. 181–185, IEEE, 2011.

[137] Y.-D. Lin, Y.-C. Lai, C.-Y. Ho, and W.-H. Tai, "Creditability-based weighted voting for reducing false positives and negatives in intrusion detection," *Comput. Secur.*, vol. 39, pp. 460–474, 2013.

[138] G. Giacinto, F. Roli, and L. Didaci, "Fusion of multiple classifiers for intrusion detection in computer networks," *Pattern Recognit. Lett.*, vol. 24, no. 12, pp. 1795–1803, 2003.

[139] A. A. Aburomman and M. B. I. Reaz, "A survey of intrusion detection systems based on ensemble and hybrid classifiers," *Comput. Secur.*, vol. 65, pp. 135–152, 2017.

[140] F. Cheng and X. Qiu, "Network anomaly detection based on frequent sub-graph mining approach and association analysis," in *2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, pp. 12–16, IEEE, 2016.

[141] Y. Liu, H. Xu, H. Yi, Z. Lin, J. Kang, W. Xia, Q. Shi, Y. Liao, and Y. Ying, "Network anomaly detection based on dynamic hierarchical clustering of cross domain data," in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 200–204, IEEE, 2017.

[142] X. Zhao, G. Wang, and Z. Li, "Unsupervised network anomaly detection based on abnormality weights and subspace clustering," in *2016 Sixth International Conference on Information Science and Technology (ICIST)*, pp. 482–486, IEEE, 2016.

[143] Y. Maleh, A. Ezzati, Y. Qasmaoui, and M. Mbida, "A global hybrid intrusion detection system for wireless sensor networks," *Procedia Comput. Sci.*, vol. 52, pp. 1047–1052, 2015.

[144] M. Yassine and A. Ezzati, "Lightweight intrusion detection scheme for wireless sensor networks," *IAENG International Journal of Computer Science*, vol. 42, pp. 347–354, 2015.

[145] C. Guo, Y. Ping, N. Liu, and S.-S. Luo, "A two-level hybrid approach for intrusion detection," *Neurocomputing*, vol. 214, pp. 391–400, 2016.

[146] M. Weber, S. Klug, E. Sax, and B. Zimmer, "Embedded hybrid anomaly detection for automotive CAN communication," in *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, 2018.

[147] L. A. Maglaras, J. Jiang, and T. J. Cruz, "Combining ensemble methods and social network metrics for improving accuracy of OCSVM on intrusion detection in SCADA systems," *J. Inf. Secur. Appl.*, vol. 30, pp. 15–26, 2016.

[148] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Skorić, "Measuring intrusion detection capability: An information-theoretic approach," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security - ASIACCS '06*, (New York, New York, USA), ACM Press, 2006.

[149] T. Holz, "Security measurements and metrics for networks," in *Dependability Metrics*, pp. 157–165, Springer Berlin Heidelberg, 2008.

[150] D. Ashok Kumar and S. R. Venugopalan, "A novel algorithm for network anomaly detection using adaptive machine learning," in *Advances in Intelligent Systems and Computing*, pp. 59–69, Singapore: Springer Singapore, 2018.

[151] S. Ossenbuhl, J. Steinberger, and H. Baier, "Towards automated incident handling: How to select an appropriate response against a network-based attack?," in *2015 Ninth International Conference on IT Security Incident Management & IT Forensics*, pp. 51–67, IEEE, 2015.

[152] N. S. Arunraj, R. Hable, M. Fernandes, K. Leidl, and M. Heigl, "Comparison of Supervised, Semi-supervised and Unsupervised Learning Methods in Network Intrusion Detection System (NIDS) Application," *Anwendungen und Konzepte der Wirtschaftsinformatik (AKWI)*, vol. 6, pp. 10–19, 2017.

[153] G. Suarez-Tangil, E. Palomar, J. M. de Fuentes, J. Blasco, and A. Ribagorda, "Automatic rule generation based on genetic programming for event correlation," in *Advances in Intelligent and Soft Computing*, pp. 127–134, Springer Berlin Heidelberg, 2009.

[154] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *Proceedings of the 25th USENIX Conference on Security Symposium*, (USA), pp. 911–927, USENIX Association, 2016.

[155] J. Steinberger, A. Sperotto, M. Golling, and H. Baier, "How to exchange security events? overview and evaluation of formats and protocols," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 261–269, IEEE, 2015.

[156] M. Kerrisk, "The linux programming interface," *Muenchen: No StarchPress*, 2010.

[157] J. Postel, "Internet protocol," STD 5, RFC Editor, September 1981. `http://www.rfc-editor.org/rfc/rfc791.txt`.

[158] R. Koch, M. Golling, and G. Dreo, "Evaluation of state of the art ids message exchange protocols," *International Journal of Computer and Systems Engineering*, vol. 7, pp. 1017–1026, 2013.

[159] CESNET, "Warden - a system for efficient sharing information about detected events (threats)," *https://warden.cesnet.cz/en/index (online, accessed 22 July 2019)*, 2017.

[160] MISP-Project, "MISP - Malware Information Sharing Platform," *https://www.misp-project.org/ (online, accessed 05 September 2021)*, 2021.

[161] A. AlEroud and G. Karabatis, "Beyond data: Contextual information fusion for cyber security analytics," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, (New York, NY, USA), ACM, 2016.

[162] J. Haines, D. Kewley Ryder, L. Tinnel, and S. Taylor, "Validation of sensor alert correlators," *IEEE Secur. Priv.*, vol. 1, no. 1, pp. 46–56, 2003.

[163] G. Gu, A. A. Cárdenas, and W. Lee, "Principled reasoning and practical applications of alert fusion in intrusion detection systems," in *Proceedings of the 2008 ACM symposium on Information, computer and communications security - ASIACCS '08*, (New York, New York, USA), ACM Press, 2008.

[164] W. Meng, Y. Wang, W. Li, Z. Liu, J. Li, and C. W. Probst, "Enhancing intelligent alarm reduction for distributed intrusion detection systems via edge computing," in *Information Security and Privacy*, pp. 759–767, Cham: Springer International Publishing, 2018.

[165] S. Salah, G. Maciá-Fernández, and J. E. Díaz-Verdejo, "A model-based survey of alert correlation techniques," *Comput. netw.*, vol. 57, no. 5, pp. 1289–1317, 2013.

[166] N. Hubballi and V. Suryanarayanan, "False alarm minimization techniques in signature-based intrusion detection systems: A survey," *Comput. Commun.*, vol. 49, pp. 1–17, 2014.

[167] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 3, pp. 146–169, 2004.

[168] A. Siraj, "A unified alert fusion model for intelligent analysis of sensor data in an intrusion detection environment," *Ph.D. thesis, Mississippi State University*, 2006.

[169] H. T. Elshoush and I. M. Osman, "An improved framework for intrusion alert correlation," *Proceedings of the World Congress on Engineering*, vol. 1, pp. 1–6, 2012.

[170] P. Ning, Y. Cui, and D. S. Reeves, "Analyzing intensive intrusion alerts via correlation," in *Lecture Notes in Computer Science*, pp. 74–94, Springer Berlin Heidelberg, 2002.

[171] F. Maggi and S. Zanero, "On the use of different statistical tests for alert correlation – short paper," in *Lecture Notes in Computer Science*, pp. 167–177, Springer Berlin Heidelberg, 2007.

[172] M. GhasemiGol, A. Ghaemi-Bafghi, and H. Takabi, "A comprehensive approach for network attack forecasting," *Comput. Secur.*, vol. 58, pp. 83–105, 2016.

[173] S. A. Mirheidari, S. Arshad, and R. Jalili, "Alert correlation algorithms: A survey and taxonomy," in *Cyberspace Safety and Security*, pp. 183–197, Cham: Springer International Publishing, 2013.

[174] H. T. Elshoush and I. M. Osman, "Alert correlation in collaborative intelligent intrusion detection systems—a survey," *Appl. Soft Comput.*, vol. 11, no. 7, pp. 4349–4365, 2011.

[175] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *Lecture Notes in Computer Science*, pp. 54–68, Springer Berlin Heidelberg, 2001.

[176] X. Zhuang, D. Xiao, X. Liu, and Y. Zhang, "Applying data fusion in collaborative alerts correlation," in *2008 International Symposium on Computer Science and Computational Technology*, pp. 124–127, IEEE, 2008.

[177] W. N. Thurman, M. E. Fisher, *et al.*, "Chickens, eggs, and causality, or which came first," *American journal of agricultural economics*, vol. 70, no. 2, pp. 237–238, 1988.

[178] P. Ning, Y. Cui, D. S. Reeves, and D. Xu, "Towards automating intrusion alert analysis," *2003 Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, pp. 1–19, 2003.

[179] B. Zhu and A. A. Ghorbani, "Alert correlation for extracting attack strategies," *IJ Network Security*, vol. 3, no. 3, pp. 244–258, 2006.

[180] S. Saad and I. Traore, "Extracting attack scenarios using intrusion semantics," in *Foundations and Practice of Security*, pp. 278–292, Springer Berlin Heidelberg, 2013.

[181] J. Maestre Vidal, A. L. Sandoval Orozco, and L. J. García Villalba, "Alert correlation framework for malware detection by anomaly-based packet payload analysis," *J. Netw. Comput. Appl.*, vol. 97, pp. 11–22, 2017.

[182] F. Cuppens and R. Ortalo, "LAMBDA: A language to model a database for detection of attacks," in *Lecture Notes in Computer Science*, pp. 197–216, Springer Berlin Heidelberg, 2000.

[183] S. T. Eckmann, G. Vigna, and R. A. Kemmerer, "STATL: An attack language for state-based intrusion detection," *J. Comput. Secur.*, vol. 10, no. 1-2, pp. 71–103, 2002.

[184] S. Cheung, U. Lindqvist, and M. W. Fong, "Modeling multistep cyber attacks for scenario recognition," in *Proceedings DARPA Information Survivability Conference and Exposition*, vol. 1, IEEE Comput. Soc, 2003.

[185] S. H. Ahmadinejad, S. Jalili, and M. Abadi, "A hybrid model for correlating alerts of known and unknown attack scenarios and updating attack graphs," *Comput. netw.*, vol. 55, no. 9, pp. 2221–2240, 2011.

[186] C. T. Kawakani, S. B. Junior, and R. S. Miani, "Intrusion alert correlation to support security management," in *Proceedings of the XII Brazilian Symposium on Information Systems on Brazilian Symposium on Information Systems: Information Systems in the Cloud Computing Era - Volume 1*, (Porto Alegre, BRA), pp. 313–320, Brazilian Computer Society, 2016.

[187] G. Suarez-Tangil, E. Palomar, A. Ribagorda, and I. Sanz, "Providing SIEM systems with self-adaptation," *Inf. Fusion*, vol. 21, pp. 145–158, 2015.

[188] M. Soleimani and A. A. Ghorbani, "Multi-layer episode filtering for the multi-step attack detection," *Comput. Commun.*, vol. 35, no. 11, pp. 1368–1379, 2012.

[189] A. Ahmadian Ramaki and A. Rasoolzadegan, "Causal knowledge analysis for detecting and modeling multi-step attacks: Causal knowledge analysis for detecting and modeling multi-step attacks," *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 6042–6065, 2016.

[190] X. Sun, J. Dai, P. Liu, A. Singhal, and J. Yen, "Using bayesian networks for probabilistic identification of zero-day attack paths," *IEEE trans. inf. forensics secur.*, vol. 13, no. 10, pp. 2506–2521, 2018.

[191] P. Ning and D. Xu, "Hypothesizing and reasoning about attacks missed by intrusion detection systems," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 4, pp. 591–627, 2004.

[192] G. Tedesco and U. Aickelin, "Real-time alert correlation with type graphs," *arXiv preprint arXiv:1004.4089*, vol. abs/1004.4089, 2010.

[193] G. P. Spathoulas and S. K. Katsikas, "Enhancing IDS performance through comprehensive alert post-processing," *Comput. Secur.*, vol. 37, pp. 176–196, 2013.

[194] H. Fatma and M. Limam, "A two-stage process based on data mining and optimization to identify false positives and false negatives generated by intrusion detection systems," in *2015 11th International Conference on Computational Intelligence and Security (CIS)*, pp. 308–311, IEEE, 2015.

[195] Q. Hui and W. Kun, "Real-time network attack intention recognition algorithm," *Int. j. secur. appl.*, vol. 10, no. 4, pp. 51–62, 2016.

[196] B. D. Bryant and H. Saiedian, "A novel kill-chain framework for remote security log analysis with SIEM software," *Comput. Secur.*, vol. 67, pp. 198–210, 2017.

[197] M. Abdlhamed, K. Kifayat, Q. Shi, and W. Hurst, "A system for intrusion prediction in cloud computing," in *Proceedings of the International Conference on Internet of things and Cloud Computing*, (New York, NY, USA), ACM, 2016.

[198] E. T. Anumol, "Use of machine learning algorithms with SIEM for attack prediction," in *Advances in Intelligent Systems and Computing*, pp. 231–235, New Delhi: Springer India, 2015.

[199] A. A. Ramaki and R. E. Atani, "A survey of IT early warning systems: architectures, challenges, and solutions: A survey of IT early warning systems: architectures,

challenges, and solutions," *Secur. Commun. Netw.*, vol. 9, no. 17, pp. 4751–4776, 2016.

[200] M. Apel, J. Biskup, U. Flegel, and M. Meier, "Towards early warning systems – challenges, technologies and architecture," in *Critical Information Infrastructures Security*, pp. 151–164, Springer Berlin Heidelberg, 2010.

[201] A. A. Ramaki, M. Khosravi-Farmad, and A. G. Bafghi, "Real time alert correlation and prediction using bayesian networks," in *2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC)*, pp. 98–103, IEEE, 2015.

[202] P. Holgado, V. A. Villagra, and L. Vazquez, "Real-time multistep attack prediction based on hidden markov models," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 1, pp. 134–147, 2020.

[203] S. N. Narayanan, S. Mittal, and A. Joshi, "Using data analytics to detect anomalous states in vehicles," *arXiv preprint arXiv:1512.08048*, vol. abs/1512.08048, 2015.

[204] A. M. del Rey, "Mathematical modeling of the propagation of malware: a review," *Secur. Commun. Netw.*, vol. 8, no. 15, pp. 2561–2579, 2015.

[205] M. Trawicki, "Deterministic seirs epidemic model for modeling vital dynamics, vaccinations, and temporary immunity," *Mathematics*, vol. 5, no. 1, p. 7, 2017.

[206] E. Magkos, M. Avlonitis, P. Kotzanikolaou, and M. Stefanidakis, "Towards early warning against internet worms based on critical-sized networks," *Secur. Commun. Netw.*, vol. 6, no. 1, pp. 78–88, 2013.

[207] S. Peng, S. Yu, and A. Yang, "Smartphone malware and its propagation modeling: A survey," *IEEE Commun. Surv. Tutor.*, vol. 16, no. 2, pp. 925–941, 2014.

[208] S. H. White, A. M. Del Rey, and G. R. Sánchez, "Modeling epidemics using cellular automata," *Appl. Math. Comput.*, vol. 186, no. 1, pp. 193–202, 2007.

[209] J. Pan and C. C. Fung, "An agent-based model to simulate coordinated response to malware outbreak within an organisation," *International Journal of Information and Computer Security*, vol. 5, no. 2, pp. 115–131, 2012.

[210] Y. El Ansari, A. El Myr, and L. Omari, "Deterministic and stochastic study for an infected computer network model powered by a system of antivirus programs," *Discrete Dyn. Nat. Soc.*, vol. 2017, pp. 1–13, 2017.

[211] J. Jiang, S. Wen, B. Liu, S. Yu, Y. Xiang, and W. Zhou, *Malicious attack propagation and source identification.* Basel, Switzerland: Springer International Publishing, 1 ed., 2018.

[212] Y. Wang, S. Wen, Y. Xiang, and W. Zhou, "Modeling the propagation of worms in networks: A survey," *IEEE Commun. Surv. Tutor.*, vol. 16, no. 2, pp. 942–960, 2014.

[213] L. Xu, "Markovian and stochastic differential equation based approaches to computer virus propagation dynamics and some models for survival distributions," *Ph.D. thesis, Department of Mathematics and Computer Science, Faculty of New Jersey Institute of Technology and Rutgers*, 2011.

[214] R.-R. Xi, X.-C. Yun, Z.-Y. Hao, and Y.-Z. Zhang, "Quantitative threat situation assessment based on alert verification: Threat situation assessment," *Secur. Commun. Netw.*, vol. 9, no. 13, pp. 2135–2142, 2016.

[215] G. Apruzzese, M. Marchetti, M. Colajanni, G. G. Zoccoli, and A. Guido, "Identifying malicious hosts involved in periodic communications," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pp. 1–8, IEEE, 2017.

[216] P. A. Porras, M. W. Fong, and A. Valdes, "A mission-impact-based approach to INFOSEC alarm correlation," in *Lecture Notes in Computer Science*, pp. 95–114, Springer Berlin Heidelberg, 2002.

[217] K. Alsubhi, E. Al-Shaer, and R. Boutaba, "Alert prioritization in intrusion detection systems," in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, pp. 33–40, IEEE, 2008.

[218] S. McElwee, J. Heaton, J. Fraley, and J. Cannady, "Deep learning for prioritizing and responding to intrusion detection alerts," in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, pp. 1–5, IEEE, 2017.

[219] R. Shittu, A. Healing, R. Ghanea-Hercock, R. Bloomfield, and M. Rajarajan, "Intrusion alert prioritisation and attack detection using post-correlation analysis," *Comput. Secur.*, vol. 50, pp. 1–15, 2015.

[220] R. Kumar and B. B. Gupta, "Stepping stone detection techniques: Classification and state-of-the-art," in *Proceedings of the International Conference on Recent Cognizance in Wireless Communication & Image Processing*, pp. 523–533, New Delhi: Springer India, 2016.

[221] D. Shah and T. Zaman, "Detecting sources of computer viruses in networks: Theory and experiment," in *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS '10*, (New York, New York, USA), ACM Press, 2010.

[222] V. Fioriti, M. Chinnici, and J. Palomo, "Predicting the sources of an outbreak with a spectral technique," *Appl. Math. Sci.*, vol. 8, pp. 6775–6782, 2014.

[223] D. Shah and T. Zaman, "Rumors in a network: Who's the culprit?," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5163–5181, 2011.

[224] D. Shah and T. Zaman, "Rumor centrality: A universal source detector," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems - SIGMETRICS '12*, (New York, New York, USA), ACM Press, 2012.

[225] K. Zhu and L. Ying, "A robust information source estimator with sparse observations," *Comput. Soc. Netw.*, vol. 1, no. 1, 2014.

[226] F. Altarelli, A. Braunstein, L. Dall'Asta, A. Lage-Castellanos, and R. Zecchina, "Bayesian inference of epidemics on networks via belief propagation," *Phys. Rev. Lett.*, vol. 112, no. 11, p. 118701, 2014.

[227] P. C. Pinto, P. Thiran, and M. Vetterli, "Locating the source of diffusion in large-scale networks," *arXiv preprint arXiv:1208.2534*, vol. abs/1208.2534, 2012.

[228] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang, "Worm origin identification using random moonwalks," in *2005 IEEE Symposium on Security and Privacy (S&P'05)*, pp. 242–256, IEEE, 2005.

[229] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 4, pp. 443–471, 2003.

[230] J. Kim, G. Lee, J.-T. Seo, E.-K. Park, C.-S. Park, and D.-K. Kim, "Y-AOI: Y-means based attribute oriented induction identifying root cause for IDSs," in *Fuzzy Systems and Knowledge Discovery*, pp. 205–214, Springer Berlin Heidelberg, 2005.

[231] Y. Guan, A. Ghorbani, and N. Belacel, "Y-means: a clustering method for intrusion detection," in *CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No.03CH37436)*, vol. 2, pp. 1083–1086, IEEE, 2003.

[232] S. O. Al-Mamory and H. Zhang, "Intrusion detection alarms reduction using root cause analysis and clustering," *Comput. Commun.*, vol. 32, no. 2, pp. 419–430, 2009.

[233] M. T. Kechadi, J. H. Bellec, and A. Tari, "Behavioural proximity discovery: an adaptive approach for root cause analysis," *Int. j. bus. intell. data min.*, vol. 6, no. 3, p. 259, 2011.

[234] D. Cotroneo, A. Paudice, and A. Pecchia, "Automated root cause identification of security alerts: Evaluation in a SaaS cloud," *Future Gener. Comput. Syst.*, vol. 56, pp. 375–387, 2016.

[235] I. Sanchez-Rola, I. Santos, and D. Balzarotti, "Clock around the clock: Time-based device fingerprinting," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, (New York, NY, USA), ACM, 2018.

[236] K.-T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, (New York, NY, USA), ACM, 2017.

[237] R. Shirey, "Internet security glossary," RFC 2828, RFC Editor, May 2000.

[238] N. Herold, "Incident handling systems with automated intrusion response," *Ph.D. thesis, Technical University Munich, Germany, https://dblp.org/rec/bib/phd/dnb/Herold17*, 2017.

[239] N. Stakhanova, S. Basu, and J. Wong, "A taxonomy of intrusion response systems," *Int. j. inf. comput. secur.*, vol. 1, no. 1/2, p. 169, 2007.

[240] C. Strasburg, N. Stakhanova, S. Basu, and J. S. Wong, "Intrusion response cost assessment methodology," in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security - ASIACCS '09*, (New York, New York, USA), ACM Press, 2009.

[241] A. Shameli-Sendi and M. Dagenais, "ORCEF: Online response cost evaluation framework for intrusion response system," *J. Netw. Comput. Appl.*, vol. 55, pp. 89–107, 2015.

[242] G. Gonzalez Granadillo, H. Débar, G. Jacob, C. Gaber, and M. Achemlal, "Individual countermeasure selection based on the return on response investment index," in *Lecture Notes in Computer Science*, pp. 156–170, Springer Berlin Heidelberg, 2012.

[243] A. Fawaz, R. Berthier, and W. H. Sanders, "Cost modeling of response actions for automated response and recovery in AMI," in *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, pp. 348–353, IEEE, 2012.

[244] V. Mateos, V. A. Villagrá, F. Romero, and J. Berrocal, "Definition of response metrics for an ontology-based automated intrusion response systems," *Comput. Electr. Eng.*, vol. 38, no. 5, pp. 1102–1114, 2012.

[245] N. Stakhanova, S. Basu, and J. Wong, “A cost-sensitive model for preemptive intrusion response systems,” in *21st International Conference on Advanced Networking and Applications (AINA ’07)*, pp. 428–435, IEEE, 2007.

[246] W. Kanoun, N. Cuppens-Boulahia, F. Cuppens, S. Dubus, and A. Martin, “Intelligent response system to mitigate the success likelihood of ongoing attacks,” in *IAS 2010 : 6th IEEE International Conference on Information Assurance and Security*, (Atlanta, United States), Aug. 2010.

[247] J. Baayer and B. Regragui, “New cost-sensitive model for intrusion response systems minimizing false positive,” *IJMER - International Journal of Modern Engineering Research 2*, 2012.

[248] Z. Zhang, P.-H. Ho, and L. He, “Measuring IDS-estimated attack impacts for rational incident response: A decision theoretic approach,” *Comput. Secur.*, vol. 28, no. 7, pp. 605–614, 2009.

[249] W. T. Yue and M. Çakanyıldırım, “A cost-based analysis of intrusion detection system configuration under active or passive response,” *Decis. Support Syst.*, vol. 50, no. 1, pp. 21–31, 2010.

[250] B. A. Fessi, S. Benabdallah, N. Boudriga, and M. Hamdi, “A multi-attribute decision model for intrusion response system,” *Inf. Sci. (Ny)*, vol. 270, pp. 237–254, 2014.

[251] J. Wang, K. Fan, W. Mo, and D. Xu, “A method for information security risk assessment based on the dynamic bayesian network,” in *2016 International Conference on Networking and Network Applications (NaNA)*, pp. 279–283, IEEE, 2016.

[252] D. Schnackenberg, K. Djahandari, and D. Sterne, “Infrastructure for intrusion detection and response,” in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX’00*, vol. 2, pp. 3–11, 2000.

[253] D. Schnackengerg, H. Holliday, R. Smith, K. Djahandari, and D. Sterne, “Cooperative intrusion traceback and response architecture (citra),” in *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX’01*, vol. 1, pp. 56–68, 2001.

[254] S. A. Zonouz, H. Khurana, W. H. Sanders, and T. M. Yardley, “RRE: A game-theoretic intrusion response and recovery engine,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 395–406, 2014.

[255] R. Sharma, H. Kalita, and B. Issac, “Plant based biologically inspired intrusion response mechanism : An insight into the proposed model PIRIDS,” *Journal of Information Assurance and Security*, vol. 11, no. 6, pp. 340–347, 2016.

[256] R. K. Sharma, B. Issac, and H. K. Kalita, “Intrusion detection and response system inspired by the defense mechanism of plants,” *IEEE Access*, vol. 7, pp. 52427–52439, 2019.

[257] H. A. Kholidy, A. Erradi, S. Abdelwahed, and F. Baiardi, “A risk mitigation approach for autonomous cloud intrusion response system,” *Computing*, vol. 98, no. 11, pp. 1111–1135, 2016.

[258] N. Herold, M. Wachs, S.-A. Posselt, and G. Carle, “An optimal metric-aware response selection strategy for intrusion response systems,” in *Foundations and Practice of Security*, pp. 68–84, Cham: Springer International Publishing, 2017.

[259] M. GhasemiGol, H. Takabi, and A. Ghaemi-Bafghi, “A foresight model for intrusion response management,” *Comput. Secur.*, vol. 62, pp. 73–94, 2016.

[260] A. Shameli-Sendi, H. Louafi, W. He, and M. Cheriet, "Dynamic optimal countermeasure selection for intrusion response system," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 5, pp. 755–770, 2018.

[261] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network configuration protocol (netconf)," RFC 6241, RFC Editor, June 2011. `http://www.rfc-editor.org/rfc/rfc6241.txt`.

[262] K. Djahandari and D. Schnackenberg, "Intruder detection and isolation protocol (IDIP) application layer protocol definition," *Active Networks Intrusion Detection and Response Program Technical Information Report*, vol. Prepared Under Contract N66001-00-C-8602 for SPAWARSYSCEN San Diego, 2002.

[263] G. Klein, H. Rogge, F. Schneider, J. Toelle, M. Jahnke, and S. Karsch, "Response initiation in distributed intrusion response systems for tactical MANETs," in *2010 European Conference on Computer Network Defense*, pp. 55–62, IEEE, 2010.

[264] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, 2004.

[265] A. Carlin, M. Hammoudeh, and O. Aldabbas, "Intrusion detection and countermeasure of virtual cloud systems - state of the art and current challenges," *Int. J. Adv. Comput. Sci. Appl.*, vol. 6, no. 6, 2015.

[266] T. Xing, Z. Xiong, D. Huang, and D. Medhi, "SDNIPS: Enabling Software-Defined networking based intrusion prevention system in clouds," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, pp. 308–311, IEEE, 2014.

[267] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing network security through software defined networking (SDN)," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, IEEE, 2016.

[268] K. Giotis, G. Androulidakis, and V. Maglaris, "Leveraging SDN for efficient anomaly detection and mitigation on legacy networks," in *2014 Third European Workshop on Software Defined Networks*, pp. 85–90, IEEE, 2014.

[269] M. B. Lehocine and M. Batouche, "Flexibility of managing VLAN filtering and segmentation in SDN networks," in *2017 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, IEEE, 2017.

[270] B. R. Granby, B. Askwith, and A. K. Marnerides, "SDN-PANDA: Software-defined network platform for ANomaly detection applications," in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pp. 463–466, IEEE, 2015.

[271] M. G. Perez, A. H. Celdran, F. Ippoliti, P. G. Giardina, G. Bernini, R. M. Alaez, E. Chirivella-Perez, F. J. G. Clemente, G. M. Perez, E. Kraja, G. Carrozzo, J. M. A. Calero, and Q. Wang, "Dynamic reconfiguration in 5G mobile networks to proactively detect and mitigate botnets," *IEEE Internet Comput.*, vol. 21, no. 5, pp. 28–36, 2017.

[272] B. Krawczyk and A. Cano, "Online ensemble learning with abstaining classifiers for drifting and noisy data streams," *Appl. Soft Comput.*, vol. 68, pp. 677–692, 2018.

[273] A. Zheng, *Feature Engineering for Machine Learning.* Sebastopol, CA: O'Reilly Media, 2018.

[274] M. Rahmaninia and P. Moradi, "OSFSMI: Online stream feature selection method based on mutual information," *Appl. Soft Comput.*, vol. 68, pp. 733–746, 2018.

[275] N. Almusallam, Z. Tari, J. Chan, and A. AlHarthi, "UFSSF - an efficient unsupervised feature selection for streaming features," in *Advances in Knowledge Discovery and Data Mining*, pp. 495–507, Cham: Springer International Publishing, 2018.

[276] C. Siu and R. Y. D. Xu, "Diverse online feature selection," *CoRR*, vol. abs/1806.04308, 2018.

[277] D. Panday, R. Cordeiro de Amorim, and P. Lane, "Feature weighting as a tool for unsupervised feature selection," *Inf. Process. Lett.*, vol. 129, pp. 44–52, 2018.

[278] C. Fahy and S. Yang, "Dynamic feature selection for clustering high dimensional data streams," *IEEE Access*, vol. 7, pp. 127128–127140, 2019.

[279] R. Ma, Y. Wang, and L. Cheng, "Feature selection on data stream via multi-cluster structure preservation," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, (New York, NY, USA), ACM, 2020.

[280] J. Wang, P. Zhao, S. C. H. Hoi, and R. Jin, "Online feature selection and its applications," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 698–710, 2014.

[281] M. S. Hammoodi, F. Stahl, and A. Badii, "Real-time feature selection technique with concept drift detection using adaptive micro-clusters for data stream mining," *Knowl. Based Syst.*, vol. 161, pp. 205–239, 2018.

[282] J. P. Barddal, H. Murilo Gomes, F. Enembreck, B. Pfahringer, and A. Bifet, "On dynamic feature weighting for feature drifting data streams," in *Machine Learning and Knowledge Discovery in Databases*, pp. 129–144, Cham: Springer International Publishing, 2016.

[283] S. Solorio-Fernández, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, "A review of unsupervised feature selection methods," *Artif. Intell. Rev.*, vol. 53, no. 2, pp. 907–948, 2020.

[284] W. Shao, L. He, C.-T. Lu, X. Wei, and P. S. Yu, "Online unsupervised multi-view feature selection," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 1203–1208, IEEE, 2016.

[285] H. Huang, S. Yoo, and S. P. Kasiviswanathan, "Unsupervised feature selection on data streams," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management - CIKM '15*, (New York, New York, USA), ACM Press, 2015.

[286] E. Liberty, "Simple and deterministic matrix sketching," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), ACM, 2013.

[287] T. A. Alamiedy, M. Anbar, A. K. Al-Ani, B. N. Al-Tamimi, and N. Faleh, "Review on feature selection algorithms for anomaly-based intrusion detection system," in *Advances in Intelligent Systems and Computing*, pp. 605–619, Cham: Springer International Publishing, 2019.

[288] J. S. Park, K. M. Shazzad, and D. S. Kim, "Toward modeling lightweight intrusion detection system through correlation-based hybrid feature selection," in *Information Security and Cryptology*, pp. 279–289, Springer Berlin Heidelberg, 2005.

[289] O. Y. Al-Jarrah, A. Siddiqui, M. Elsalamouny, P. D. Yoo, S. Muhaidat, and K. Kim, "Machine-learning-based feature selection techniques for large-scale network intrusion detection," in *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops*, pp. 177–181, IEEE, 2014.

[290] S. Chen, Z. Huang, Z. Zuo, and X. Guo, "A feature selection method for anomaly detection based on improved genetic algorithm," in *Proceedings of the 2016 4th International Conference on Mechanical Materials and Manufacturing Engineering*, (Paris, France), Atlantis Press, 2016.

[291] F. Gottwalt, E. Chang, and T. Dillon, "CorrCorr: A feature selection method for multivariate correlation network anomaly detection techniques," *Comput. Secur.*, vol. 83, pp. 234–245, 2019.

[292] J. Ren, J. Guo, W. Qian, H. Yuan, X. Hao, and H. Jingjing, "Building an effective intrusion detection system by using hybrid data optimization based on machine learning algorithms," *Secur. Commun. Netw.*, vol. 2019, pp. 1–11, 2019.

[293] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "IoT malicious traffic identification using wrapper-based feature selection mechanisms," *Comput. Secur.*, vol. 94, no. 101863, p. 101863, 2020.

[294] Y. Zhou, G. Cheng, S. Jiang, and M. Dai, "Building an efficient intrusion detection system based on feature selection and ensemble classifier," *Comput. netw.*, vol. 174, no. 107247, p. 107247, 2020.

[295] A. Nazir and R. A. Khan, "A novel combinatorial optimization based feature selection method for network intrusion detection," *Comput. Secur.*, vol. 102, no. 102164, p. 102164, 2021.

[296] T. Naidoo, J. R. Tapamo, and A. McDonald, "Feature selection for anomaly–based network intrusion detection using cluster validity indices," *SATNAC: Africa – The Future Communications Galaxy*, pp. 6–9, Sep 2015.

[297] N. N. R. R. Suri, M. N. Murty, and G. Athithan, "Unsupervised feature selection for outlier detection in categorical data using mutual information," in *2012 12th International Conference on Hybrid Intelligent Systems (HIS)*, pp. 253–258, IEEE, 2012.

[298] G. Pang, L. Cao, and L. Chen, "Outlier detection in complex categorical data by modelling the feature value couplings," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pp. 1902–1908, AAAI Press, 2016.

[299] G. Pang, L. Cao, L. Chen, D. Lian, and H. Liu, "Sparse modeling-based sequential ensemble learning for effective outlier detection in high-dimensional numeric data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32 (1), 2018.

[300] M. Prasad, S. Tripathi, and K. Dahal, "Unsupervised feature selection and cluster center initialization based arbitrary shaped clusters for intrusion detection," *Computers & Security*, vol. 99, p. 102062, 2020.

[301] L. Cheng, Y. Wang, X. Liu, and B. Li, "Outlier detection ensemble with embedded feature selection," *Proc. Conf. AAAI Artif. Intell.*, vol. 34, no. 04, pp. 3503–3512, 2020.

[302] Q. Yang, J. Singh, and J. Lee, "Isolation-based feature selection for unsupervised outlier detection," *Proc. Annu. Conf. Progn. Health Manag. Soc.*, vol. 11, no. 1, 2019.

[303] I. Sharafaldin., A. Habibi Lashkari., and A. A. Ghorbani., "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP,*, pp. 108–116, INSTICC, SciTePress, 2018.

[304] H.-P. Kriegel, P. Kroger, E. Schubert, and A. Zimek, "Interpreting and unifying outlier scores," in *Proceedings of the 2011 SIAM International Conference on Data Mining*, (Philadelphia, PA), pp. 13–24, Society for Industrial and Applied Mathematics, 2011.

[305] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[306] S. Inka, "Adaptive real-time anomaly detection for multi-dimensional streaming data," *Master's thesis, Aalto University (Aalto 8370)*, 2017.

[307] H. Wang and M. Song, "Ckmeans.1d.Dp: Optimal k-means clustering in one dimension by dynamic programming," *R J.*, vol. 3, no. 2, pp. 29–33, 2011.

[308] M. Song and H. Zhong, "Efficient weighted univariate clustering maps outstanding dysregulated genomic zones in human cancers," *Bioinformatics*, vol. 36, no. 20, pp. 5027–5036, 2020.

[309] S. F. Yilmaz and S. S. Kozat, "Pysad: A streaming anomaly detection framework in python," 2020.

[310] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *J. Mach. Learn. Res.*, vol. 20, pp. 96:1–96:7, 2019.

[311] S. Rayana, "ODDS library," *Stony Brook University, Department of Computer Sciences*, 2016.

[312] Q. Zhou and D. Pezaros, "Evaluation of machine learning classifiers for zero-day intrusion detection - an analysis on CIC-AWS-2018 dataset," *CoRR*, vol. abs/1905.03685, 2019. Withdrawn.

[313] A. Kenyon, L. Deka, and D. Elizondo, "Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets," *Comput. Secur.*, vol. 99, no. 102022, p. 102022, 2020.

[314] A. Kumar, M. Shridhar, S. Swaminathan, and T. J. Lim, "Machine learning-based early detection of iot botnets using network-edge traffic," *CoRR*, vol. abs/2010.11453, 2020.

[315] C. C. Aggarwal, *Outlier Analysis.* Cham: Springer International Publishing, 2017.

[316] Y. Ben-Haim and E. Tom-Tov, "A streaming parallel decision tree algorithm," *Journal of Machine Learning Research*, vol. 11, no. 2, 2010.

[317] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 3133–3181, 2014.

[318] S. Hariri, M. C. Kind, and R. J. Brunner, "Extended isolation forest," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 4, pp. 1479–1489, 2021.

[319] G. Staerman, P. Mozharovskyi, S. Clémençon, and F. d'Alché Buc, "Functional isolation forest," in *Asian Conference on Machine Learning*, pp. 332–347, PMLR, 2019.

[320] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Advances in Intelligent Data Analysis VIII*, pp. 249–260, Springer Berlin Heidelberg, 2009.

[321] C. Raab, M. Heusinger, and F.-M. Schleif, "Reactive soft prototype computing for concept drift streams," *Neurocomputing*, vol. 416, pp. 340–351, 2020.

[322] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," *Commun. ACM*, vol. 61, no. 7, pp. 56–66, 2018.

[323] M. Kianpour and S.-F. Wen, "Timing attacks on machine learning: State of the art," in *Advances in Intelligent Systems and Computing*, pp. 111–125, Cham: Springer International Publishing, 2020.

[324] L. Liao and B. Luo, "Entropy isolation forest based on dimension entropy for anomaly detection," in *Communications in Computer and Information Science*, pp. 365–376, Singapore: Springer Singapore, 2019.

[325] X. Zhang, W. Dou, Q. He, R. Zhou, C. Leckie, R. Kotagiri, and Z. Salcic, "LSHiForest: A generic framework for fast tree isolation based ensemble anomaly analysis," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 983–994, IEEE, 2017.

[326] D. Xu, Y. Wang, Y. Meng, and Z. Zhang, "An improved data anomaly detection method based on isolation forest," in *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, pp. 287–291, IEEE, 2017.

[327] D. Cortes, "Distance approximation using isolation forests," *CoRR*, vol. abs/1910.12362, 2019.

[328] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence – SBIA 2004*, pp. 286–295, Springer Berlin Heidelberg, 2004.

[329] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM International Conference on Data Mining*, pp. 443–448, Philadelphia, PA: Society for Industrial and Applied Mathematics, 2007.

[330] T. Dasu, S. Krishnan, D. Lin, S. Venkatasubramanian, and K. Yi, "Change (detection) you can believe in: Finding distributional shifts in data streams," in *Advances in Intelligent Data Analysis VIII*, pp. 21–34, Springer Berlin Heidelberg, 2009.

[331] L. I. Kuncheva, "Change detection in streaming multivariate data using likelihood detectors," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 5, pp. 1175–1180, 2013.

[332] D. Renuka Devi and S. Sasikala, "Online feature selection (OFS) with accelerated bat algorithm (ABA) and ensemble incremental deep multiple layer perceptron (EI-DMLP) for big data streams," *J. Big Data*, vol. 6, no. 1, 2019.

[333] J. López Lobo, "Synthetic datasets for concept drift detection purposes," *Harvard Dataverse*, 2020.

[334] D. Dua and C. Graff, "UCI machine learning repository," *University of California, Irvine, School of Information and Computer Sciences*, 2017.

[335] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, IEEE, 2015.

[336] A. Thakkar and R. Lohiya, "A review of the advancement in intrusion detection datasets," *Procedia Comput. Sci.*, vol. 167, pp. 636–645, 2020.

[337] S. Haas and M. Fischer, "On the alert correlation process for the detection of multi-step attacks and a graph-based realization," *ACM SIGAPP Appl. Comput. Rev.*, vol. 19, no. 1, pp. 5–19, 2019.

[338] E. Shao, "Encoding IP address as a feature for network intrusion detection," *Purdue University Graduate School*, Dec. 2019.

[339] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches," *IEEE Trans. Syst. Man Cybern. C Appl. Rev.*, vol. 42, no. 4, pp. 463–484, 2012.

[340] C. K. I. Williams, "The effect of class imbalance on precision-recall curves," *Neural Comput.*, vol. 33, no. 4, pp. 853–857, 2021.

[341] D. Bolzoni, S. Etalle, and P. H. Hartel, "Panacea: Automating attack classification for anomaly-based network intrusion detection systems," in *Lecture Notes in Computer Science*, pp. 1–20, Springer Berlin Heidelberg, 2009.

[342] Q. S. Qassim, A. M. Zin, and M. J. A. Aziz, "Anomaly-based network IDS false alarm filter using cluster-based alarm classification approach," *Int. J. Secur. Netw.*, vol. 12, no. 1, p. 13, 2017.

[343] J. Shin, S.-H. Choi, P. Liu, and Y.-H. Choi, "Unsupervised multi-stage attack detection framework without details on single-stage attacks," *Future Gener. Comput. Syst.*, vol. 100, pp. 811–825, 2019.

[344] K. L. K. Sudheera, D. M. Divakaran, R. P. Singh, and M. Gurusamy, "ADEPT: Detection and identification of correlated attack stages in IoT networks," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6591–6607, 2021.

[345] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[346] L. Wang, A. Liu, and S. Jajodia, "Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts," *Comput. Commun.*, vol. 29, no. 15, pp. 2917–2933, 2006.

[347] C. C. Aggarwal, P. S. Yu, J. Han, and J. Wang, "A framework for clustering evolving data streams," in *Proceedings 2003 VLDB Conference*, pp. 81–92, Elsevier, 2003.

[348] R. Sadoddin and A. A. Ghorbani, "Real-time alert correlation using stream data mining techniques," in *Proceedings of the 20th national conference on Innovative applications of artificial intelligence - Volume 3*, pp. 1731–1737, AAAI Press, 2008.

[349] H. Ren, N. Stakhanova, and A. A. Ghorbani, "An online adaptive approach to alert correlation," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 153–172, Springer Berlin Heidelberg, 2010.

[350] A. A. Ramaki, M. Amini, and R. Ebrahimi Atani, "RTECA: Real time episode correlation algorithm for multi-step attack scenarios detection," *Comput. Secur.*, vol. 49, pp. 206–219, 2015.

[351] F. Faraji Daneshgar and M. Abbaspour, "Extracting fuzzy attack patterns using an online fuzzy adaptive alert correlation framework: Fuzzy adaptive alert correlation," *Secur. Commun. Netw.*, vol. 9, no. 14, pp. 2245–2260, 2016.

[352] K. Zhang, F. Zhao, S. Luo, Y. Xin, and H. Zhu, "An intrusion action-based IDS alert correlation analysis and prediction framework," *IEEE Access*, vol. 7, pp. 150540–150551, 2019.

[353] Z. Zohrevand and U. Glässer, "Should I raise the red flag? A comprehensive survey of anomaly scoring methods toward mitigating false alarms," *CoRR*, vol. abs/1904.06646, 2019.

[354] D. Ourston, S. Matzner, W. Stump, and B. Hopkins, "Applications of hidden markov models to detecting multi-stage network attacks," in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pp. 10 pp.–, IEEE, 2003.

[355] H. Zhang, X. Jin, Y. Li, Z. Jiang, Y. Liang, Z. Jin, and Q. Wen, "A multi-step attack detection model based on alerts of smart grid monitoring system," *IEEE Access*, vol. 8, pp. 1031–1047, 2020.

[356] K. Xylogiannopoulos, P. Karampelas, and R. Alhajj, "Early DDoS detection based on data mining techniques," in *Information Security Theory and Practice. Securing the Internet of Things*, pp. 190–199, Springer Berlin Heidelberg, 2014.

[357] A. Prakash, M. Satish, T. S. S. Bhargav, and N. Bhalaji, "Detection and mitigation of denial of service attacks using stratified architecture," *Procedia Comput. Sci.*, vol. 87, pp. 275–280, 2016.

[358] J. Galeano-Brajones, J. Carmona-Murillo, J. F. Valenzuela-Valdés, and F. Luna-Valero, "Detection and mitigation of DoS and DDoS attacks in IoT-based stateful SDN : An experimental approach," *Sensors (Basel)*, vol. 20, no. 3, p. 816, 2020.

[359] M. Heigl, L. Doerr, N. Tiefnig, D. Fiala, and M. Schramm, "A resource-preserving self-regulating uncoupled MAC algorithm to be applied in incident detection," *Comput. Secur.*, vol. 85, pp. 270–287, 2019.

[360] A. Zubaroğlu and V. Atalay, "Data stream clustering: a review," *Artif. Intell. Rev.*, vol. 54, no. 2, pp. 1201–1236, 2021.

[361] Kurniabudi, D. Stiawan, Darmawijoyo, M. Y. Bin Idris, A. M. Bamhdi, and R. Budiarto, "CICIDS-2017 dataset feature analysis with information gain for anomaly detection," *IEEE Access*, vol. 8, pp. 132911–132921, 2020.

[362] M. Jeng, "Error in statistical tests of error in statistical tests," *BMC Med. Res. Methodol.*, vol. 6, no. 1, p. 45, 2006.

[363] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc.*, vol. 35, pp. 99–109, 1943.

[364] M. Heigl, K. A. Anand, A. Urmann, D. Fiala, M. Schramm, and R. Hable, "On the improvement of the isolation forest algorithm for outlier detection with streaming data," *Electronics (Basel)*, vol. 10, no. 13, p. 1534, 2021.

[365] G. Ollmann, "Why stix/taxii/cybox sharing is incompatible with ai threat detection systems," *https://www.linkedin.com/pulse/why-stixtaxiicybox-sharing-incompatible-ai-threat-systems-ollmann (online, accessed 19 July 2019)*, 2017.

[366] F. Sadique, S. Cheung, I. Vakilinia, S. Badsha, and S. Sengupta, "Automated structured threat information expression (STIX) document generation with privacy preservation," in *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pp. 847–853, IEEE, 2018.

[367] Q. Wang, J. Jiang, Z. Shi, W. Wang, B. Lv, B. Qi, and Q. Yin, "A novel multi-source fusion model for known and unknown attack scenarios," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 727–736, IEEE, 2018.

[368] X. Wang, L. Yu, H. He, and X. Gong, "MAAC: novel alert correlation method to detect multi-step attack," *CoRR*, vol. abs/2011.07793, 2020.

[369] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, 2021.

[370] P. Ning, Y. Cui, D. S. Reeves, and D. Xu, "Techniques and tools for analyzing intrusion alerts," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 2, pp. 274–318, 2004.

[371] G. Androulidakis and S. Papavassiliou, "Improving network anomaly detection via selective flow-based sampling," *IET Commun.*, vol. 2, no. 3, p. 399, 2008.

[372] J. Mai, A. Sridharan, C.-N. Chuah, H. Zang, and T. Ye, "Impact of packet sampling on portscan detection," *IEEE j. sel. areas commun.*, vol. 24, no. 12, pp. 2285–2298, 2006.

[373] E. G. Bakhoum, "Intrusion detection model based on selective packet sampling," *EURASIP J. Multimed. Inf. Secur.*, vol. 2011, no. 1, 2011.

[374] T. Ha, S. Kim, N. An, J. Narantuya, C. Jeong, J. Kim, and H. Lim, "Suspicious traffic sampling for intrusion detection in software-defined networks," *Comput. netw.*, vol. 109, pp. 172–182, 2016.

[375] R. E. Jurga and M. M. Hulboj, "Technical report - packet sampling for network monitoring," *CERN openlab report, `https://openlab.cern/` (online, accessed 05 September 2021)*, 2007.

[376] L.-B. Xu, G.-X. Wu, and J.-F. Li, "Packet-level adaptive sampling on multi-fluctuation scale traffic," in *Proceedings. 2005 International Conference on Communications, Circuits and Systems, 2005*, vol. 1, pp. 604–608, IEEE, 2005.

[377] K. Bartos, M. Rehak, and V. Krmicek, "Optimizing flow sampling for network anomaly detection," in *2011 7th International Wireless Communications and Mobile Computing Conference*, pp. 1304–1309, IEEE, 2011.

[378] Q. Xia, T. Chen, and W. Xu, "CIDS: Adapting legacy intrusion detection systems to the cloud with hybrid sampling," in *2016 IEEE International Conference on Computer and Information Technology (CIT)*, pp. 508–515, IEEE, 2016.

[379] J. He, Y. Yang, X. Wang, and Z. Tan, "Adaptive traffic sampling for P2P botnet detection: Adaptive traffic sampling for P2P botnet detection," *Int. J. Netw. Manage.*, vol. 27, no. 5, p. e1992, 2017.

[380] B. S. Shankar and A. T. Murthy, "The detailed study and verities of macsec in wired ethernet," in *International Journal of Emerging Trends in Engineering and Development*, vol. 4, pp. 508–512, June 2015.

[381] A. Kushwaha, H. R. Sharma, and A. Ambhaikar, "A novel selective encryption method for securing text over mobile ad hoc network," *Procedia Comput. Sci.*, vol. 79, pp. 16–23, 2016.

[382] L. Doerr, D. Fiala, M. Heigl, and M. Schramm, "Assessment simulation model for uncoupled message authentication," in *2017 International Conference on Applied Electronics (AE)*, pp. 1–4, IEEE, 2017.

[383] N. Ren, Q.-S. Wang, and C.-Q. Zhu, "Selective authentication algorithm based on semi-fragile watermarking for vector geographical data," in *2014 22nd International Conference on Geoinformatics*, pp. 1–6, IEEE, 2014.

[384] J. Liu, F. R. Yu, C.-H. Lung, and H. Tang, "Optimal combined intrusion detection and biometric-based continuous authentication in high security mobile ad hoc networks," *IEEE Trans. Wirel. Commun.*, vol. 8, no. 2, pp. 806–815, 2009.

[385] A. Boudguiga, W. Klaudel, A. Boulanger, and P. Chiron, "A simple intrusion detection method for controller area network," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2016.

[386] M. Heigl, M. Schramm, L. Doerr, and A. Grzemba, "Embedded plug-in devices to secure industrial network communications," in *2016 International Conference on Applied Electronics (AE)*, pp. 85–88, IEEE, 2016.

[387] A. Bremler-Barr and H. Levy, "Spoofing prevention method," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 1, pp. 536–547, IEEE, 2005.

[388] M. Heigl, M. Aman, A. Fuchs, and A. Grzemba, "Securing industrial legacy system communication through interconnected embedded plug-in devices," in *Applied Research Conference (ARC)*, pp. 501–508, 2016.

[389] E. A. Akkoyunlu, K. Ekanadham, and R. V. Huber, "Some constraints and tradeoffs in the design of network communications," in *Proceedings of the fifth symposium on Operating systems principles - SOSP '75*, (New York, New York, USA), ACM Press, 1975.

[390] G. Kumar, "Evaluation metrics for intrusion detection systems - a study," *International Journal of Computer Science and Mobile Applications (IJCSMA)*, vol. 2, no. 11, pp. 11 – 17, 2014.

[391] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, no. 1-2, pp. 18–28, 2009.

[392] N. Mouha, B. Mennink, A. Van Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede, "Chaskey: An efficient MAC algorithm for 32-bit microcontrollers," in *Selected Areas in Cryptography – SAC 2014*, pp. 306–323, Cham: Springer International Publishing, 2014.

[393] A. R. Chowdhury and S. DasBit, "LMAC: A lightweight message authentication code for wireless sensor network," in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2015.

[394] M. Rezvani, "Assessment methodology for anomaly-based intrusion detection in cloud computing," *Journal of AI and Data Mining*, vol. 6, no. 2, pp. 387–397, 2018.

[395] V. Zieglmeier, S. Kacianka, T. Hutzelmann, and A. Pretschner, "A real-time remote IDS testbed for connected vehicles," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019* (C. Hung and G. A. Papadopoulos, eds.), pp. 1898–1905, ACM, 2019.

Appendices

A Author's Publications

Journal Articles

Nari S. Arunraj, Robert Hable, Michael Fernandes, Karl Leidl, and Michael Heigl, *Comparison of Supervised, Semi-supervised and Unsupervised Learning Methods in Network Intrusion Detection System (NIDS) Application*, Anwendungen und Konzepte der Wirtschaftsinformatik (AKWI), [S.l.], n. 6, pp. 10-19, ISSN 2296-4592, November 2017. `https://ojs-hslu.ch/ojs302/index.php/AKWI/article/view/89`.

Martin Schramm, Reiner Dojen, and Michael Heigl, *A Vendor-Neutral Unified Core for Cryptographic Operations in GF(p) and GF(2^m) Based on Montgomery Arithmetic*, Security and Communication Networks, vol. 2018, Article ID 4983404, June 2018, doi: 10.1155/2018/4983404.

Michael Heigl, Laurin Doerr, Nicolas Tiefnig, Dalibor Fiala, and Martin Schramm, *A resource-preserving self-regulating Uncoupled MAC algorithm to be applied in incident detection*, Computers & Security, vol. 85, pp. 270-287, ISSN 0167-4048, May 2019, doi: 10.1016/j.cose.2019.05.010.

Michael Heigl, Kumar Anand, Andreas Urmann, Dalibor Fiala, Martin Schramm, and Robert Hable, *On the Improvement of the Isolation Forest Algorithm for Outlier Detection with Streaming Data*, Electronics (Basel), 10(13), 1534, June 2021, doi:10.3390/electronics10131534

Michael Heigl, Enrico Weigelt, Andreas Urmann, Dalibor Fiala, and Martin Schramm, *Exploiting the Outcome of Outlier Detection for Novel Attack Pattern Recognition on Streaming Data*, Electronics (Basel), 10(17), 2160, September 2021, doi:10.3390/electronics10172160

Michael Heigl, Enrico Weigelt, Dalibor Fiala, and Martin Schramm, *Unsupervised Feature Selection for Outlier Detection on Streaming Data to Enhance Network Security*, Computers & Security, *revised version submitted 02 September 2021*

Conference Papers

Michael Heigl, Martin Aman, Andreas Fuchs, Andreas Grzemba, *Securing Industrial Legacy System Communication Through Interconnected Embedded Plug-In Devices*, Applied Research Conference, pp. 501-508, ISBN 978-3-86460-494-2, Pro BUSINESS GmbH, Augsburg, Germany, June 2016

Michael Heigl, Martin Schramm, Laurin Doerr, Andreas Grzemba, *Embedded Plug-In Devices to Secure Industrial Network Communications*, 2016 International Conference on Applied Electronics (AE), pp. 85-88, Pilsen, Sept. 2016, doi: 10.1109/AE.2016.7577247

Martin Schramm, Reiner Dojen, Michael Heigl, *Experimental Assessment of FIRO- and GARO-based Noise Sources for Digital TRNG Designs on FPGAs*, 2017 International Conference on Applied Electronics (AE), pp. 221-226, Pilsen, Sept. 2017, doi:10.23919/AE.2017.8053618

Laurin Doerr, Michael Heigl, Dalibor Fiala, Martin Schramm, *Assessment Simulation Model for Uncoupled Message Authentication*, 2017 International Conference on Applied Electronics (AE), pp. 45-48, Pilsen, Sept. 2017, doi: 10.23919/AE.2017.8053580

Michael Heigl, Laurin Doerr, Amar Almaini, Dalibor Fiala, Martin Schramm, *Incident Reaction Based on Intrusion Detections' Alert Analysis*, 2018 International Conference on Applied Electronics (AE), pp. 45-50, Pilsen, Sept. 2018, doi:10.23919/AE.2018.8501419

Michael Heigl, Martin Schramm, Dalibor Fiala, *A Lightweight Quantum-Safe Security Concept for Wireless Sensor Network Communication*, 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp. 906-911, Kyoto, Japan, March 2019, doi:10.1109/PERCOMW.2019.8730749

Michael Heigl, Laurin Doerr, Martin Schramm, Dalibor Fiala, *On the Energy Consumption of Quantum-Resistant Cryptographic Implementations Suitable for Wireless Sensor Networks*, Proceedings of the 16th International Joint Conference on e-Business and Telecommunications - Volume 2: SECRYPT, pp. 72-83, Prague, Czech Republic, July 2019, doi:10.5220/0007835600720083

Laurin Doerr, Michael Heigl, Dalibor Fiala, Martin Schramm, *Comparison of Energy-Efficient Key Management Protocols for Wireless Sensor Networks*, In Proceedings of the 2019 International Electronics Communication Conference (IECC '19). Association for Computing Machinery, New York, NY, USA, 21–26, Okinawa, Japan, July 2019, doi:10.1145/3343147.3343156

Talks

Karl Leidl, Martin Aman, Michael Heigl, Andreas Grzemba, *Intrusion Detection Sensoren für industrielle Netzwerke*, CYBICS - Cyber Security for Industrial Control Systems, Würzburg, Germany, June 2016

Laurin Doerr, Michael Heigl, Andreas Grzemba, Christian Boiger, *IT-Security-Architektur für Next-Generation Kommunikationssysteme im Automobil*, VDI/VW-Gemeinschaftstagung: Fahrerassistenzsysteme und automatisiertes Fahren, Wolfsburg, Germany, November 2016

Michael Heigl, Karl Leidl, *An Approach to an Embedded Anomaly-Based IDS on the Example of SOME/IP*, 3rd Vector Testing Symposium, Stuttgart, Germany, Mai 2017

Michael Heigl, *Distributed Embedded Incident Detection and Response Mechanisms*, ProtectIT Security Convention 2017 (ProSecCon'17), Technische Hochschule Deggendorf, Deggendorf, Germany, July 2017

Michael Heigl, *Decentralized Anomaly Detection*, Tag der Forschung 2018, Technische Hochschule Deggendorf, Deggendorf, Germany, March 2018

B Contents of the Enclosed SD-Card

The enclosed SD-card contains the LaTeX source files and the final PDF versions of this thesis as well as the four publications with regard to the four major contributions. Furthermore, the SD-card contains the programming source code of all contributions together with the resulting data. To overcome closed source as well as system dependency issues, all programs, data and results are also available upon request from the authors. The structure of the enclosed SD-card is given beside.

```
SD-Card of the Thesis
├─ Manuscripts (LaTeX)
│  ├─ 01_Thesis
│  ├─ 02_UFSSOD
│  ├─ 03_PCB-iForest
│  ├─ 04_SOAAPR
│  ├─ 05_Uncoupled_MAC
│  └─ 06_Figure_Sources
├─ Chapter_3_UFSSOD
│  ├─ Source_Code
│  └─ Results
├─ Chapter_4_PCB-iForest
│  ├─ Source_Code
│  └─ Results
├─ Chapter_5_SOAAPR
│  ├─ Source_Code
│  └─ Results
└─ Chapter_6_Uncoupled_MAC
   ├─ Source_Code
   └─ Results
```