Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Testování bezpečnosti webových aplikací

Plzeň 2018 Jakub Löffelmann

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. května 2018

Jakub Löffelmann

Abstract

This thesis describes the issues of web application security and is focused on the most common types of attacks according to OWASP methodology. Theoretical part of the thesis explains individual types of attacks and illustrates vulnerable code fragments with examples of their attacks and their effect on the web application. Furthermore the fixed code for these vulnerabilities that prevents this type of attack is included. Practical part of the thesis designs, implements and tests a tool that is able to automatically analyse security of web applications for featured attacks. The functionality of this tool is verified by demo web applications, automatic and manual tests.

Abstrakt

Diplomová práce popisuje problematiku bezpečnosti webových aplikací a zaměřuje se na nejčastější typy útoků dle metodiky OWASP. V teoretické části práce jsou jednotlivé typy útoků vysvětleny a ilustrovány s využitím zranitelných kódů, ukázkou jejich napadení a výsledným efektem na webovou aplikaci. Následuje návrh opravy zranitelného kódu, který zabrání tomuto typu napadení. V praktické části práce je navržen, implementován a otestován nástroj, který je schopen automaticky testovat bezpečnost webové aplikace proti vybraným útokům. Funkčnost tohoto nástroje je ověřena s využitím testovacích webových aplikací, automatických i manuálních testů.

Poděkování

Rád bych touto cestou poděkoval Ing. Martinu Dostalovi, Ph.D. za odborné vedení a cenné rady v průběhu této práce. Dále patří mé poděkování inženýrům z firmy RTsoft za praktické rady a možnosti konzultace práce.

Obsah

1 Úvod

Cílem této diplomové práce je prostudovat problematiku bezpečnosti webových aplikací, vysvětlit principy základních bezpečnostních útoků a následně navrhnout a implementovat vlastní nástroj pro testovaní webových aplikací proti vybraným druhům útokům. Na závěr bude vlastní, implementovaný nástroj otestován a zhodnocen. Aplikace by měla sloužit jak k výukovým účelům, tak pro jednoduché experimenty.

V posledních letech své existence prodělaly webové aplikace bouřlivý vývoj. V době svého vzniku internet sloužil především armádním a akademickým účelům a v jeho prostředí se pohybovali vzdělaní lidé s čistými úmysly. Tak jako roste vývoj webových aplikací, narůstá také počet uživatelů, kteří používají internet [8]. Každý z těchto uživatelů již nemusí mít čisté úmysly a může být považován za potencionálního útočníka. Může jít například o běžného uživatele, konkurenční firmu či zpravodajce.

V dnešní době je otázka bezpečnosti velice důležitá. Při narušení chodu webové aplikace si můžeme z kamaráda udělat jen legraci [7], v tom lepším případě a v tom horším případě může firma přijít o velké peníze.

Samotná bezpečnost se dá rozdělit do více skupin. Například ji můžeme brát z pohledu zabezpečení mateřských webových serverů, nebo samostatných uživatelských účtů. V případě prolomení bezpečnosti brány může firma ztratit své nejlepší zákazníky, platit milionové pokuty, či se jen poučit ze své vlastní chyby.

Z historie moderních webových aplikací je známo, že některé firmy podceňují a ignorují otázku bezpečnosti [25, 26] a to i přesto, že jsou jim chyby reportovány. Důvodem ignorace takových chyb je investice financí do opravy. Taková investice je často vyšší, než samotné ztráty způsobené využitím chyby v aplikaci. V praxi tento způsob řešení bezpečnosti není nic neobvyklého.

Důsledkem výše popsáného je, že bezpečnost nelze brát na lehkou váhu a před samotným uvedením aplikace do provozu je nutné ji alespoň částečně otestovat.

V této práci jsem si kladl za cíl prostudovat problematiku bezpečnosti webových aplikací. Jednotlivé bezpečnostní problémy jsou teoreticky rozebrány a ilustrovány na konkrétních případech fragmentů zdrojových kódů, které jsou zpracovány v jazyce PHP[1] a JavaScript[2]. Databázové dotazy v ja-

[1] Hypertext Preprocessor, česky „PHP: Hypertextový preprocesor“, původně Personal Home Page - skriptovací jazyk k tvorbe dynamických webových aplikací

[2] Multiplatformní, objektově orientovaný skriptovací jazyk

zyce SQL[3] jsou připraveny pro SŘDB typu MySql nebo MariaDB. S drobnými modifikacemi by měly být funkční i na většině ostatních SŘDB.

Cílem diplomové práce bylo i mimo jiné připravit webovou aplikaci, která bude testovat vybrané druhy útoků. Návrhu a samotné implementaci aplikace byla věnována velká část této práce.

Práce se dělí na více částí. V kapitole 2. budou popsány nejčastější typy útoků nejen dle metodiky OWASP. Jednotlivé typy útoků budou vysvětleny a ilustrovány s využitím zranitelných kódu, ukázkou jejich napadení a následuje návrh opravy zranitelného kódu. V kapitole 3. budou popsány již existující aplikace a jejich funkcionalita. Kapitola 4. a 5. se bude již věnovat vlastnímu nástroji pro testování bezpečnosti webových aplikací.

[3]Structured Query Language – dotazovací jazyk k tvorbě dotazů v relačních databázích

2 Bezpečnost webových aplikací

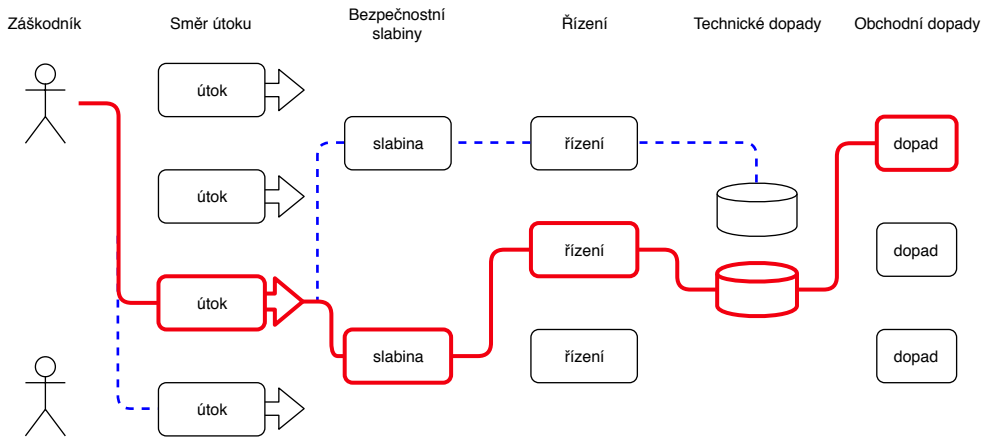
Tato část shrnuje základní problematiku bezpečnosti webových aplikací, popisuje základní bezpečností problémy a útoky nejen dle metodiky OWASP.

2.1 Motivace

Rozvojem nových technologií internetu se stále více aplikací přesouvá do online světa. Význam těchto aplikací roste a často jsou samotnou podstatu podnikání. World Wide Web je rozsáhlá a komplexní aplikační platforma, která je schopna poskytnout širokou škálu sofistikovaných aplikací. Valná část webových aplikací však prochází fázemi rychlého vývoje s extrémně krátkou dobou testování, což zvyšuje jejich zranitelnost. Čím více je náš software složitější a větší, tak se obtížnost zabezpečení exponenciálně zvyšuje. To si do jisté míry uvědomují i útočníci. V aktuální době zůstává bezpečnost hlavní překážkou různorodých druhů transakcí na internetu.

Základní otázkou může být, proč dodržovat bezpečnost webových aplikací, proč například šifrovat data a proč se bránit útokům? Kdo na nás může útočit?

Důvodů je mnoho. Jedním z nich může být například kamarád, který se nudí, či konkurenční obchodní prostředí. Každý chce uchovávat svá data v bezpečí. Nemusí jít pouze o osobní údaje, ale například i o další informace o tom, za co utrácí peníze, nebo s kým si píše. Existuje mnoho různých cest a způsobů (znázorňuje obrázek 2.1), jak mohou útočníci prostřednictvím naší aplikace poškodit firmu. Každá z těchto cest, která vede k poškození firmy, představuje riziko, které může mít různou úroveň závažnosti.



Obrázek 2.1: Možné způsoby a dopady útoků

Jedno je jisté, bezpečnost webových aplikací by se neměla v žádném případě podceňovat a brát na lehkou váhu, neboť různých samouků a profesionálních hackerů (označujme tak člověka co se snaží najít bezpečnostní díru ve webové aplikaci) je na světě nespočet.

Motivace takových útočníků může být různorodá a nemusí být vždy finanční. Může jít také o reputaci, zábavu, překonávání nejrůznějších výzev, nesouhlas se společností, zničení konkurence a dokonce i vojenský útok.

2.1.1 Obecné nařízení o ochraně osobních údajů

Z důvodu, že některé následující kapitoly se přímo dotýkají problematiky osobních údajů, bude GDPR popsané v této kapitole.

General Data Protection Regulation (GDPR) [3] je nové nařízení Evropské komise, které výrazně zvýší ochranu osobních dat občanů (fyzických osob).

GDPR představuje nový právní rámec ochrany osobních údajů v evropském prostoru s cílem hájit práva občanů Evropské unie, proti neoprávněnému zacházení s jejich daty včetně osobních údajů. Týká se všech firem a institucí, ale i jednotlivců a online služeb, které zpracovávají data uživatelů. Osobní údaje jsou v tomto případě definovány jako veškeré informace vztahující se k identifikované či identifikovatelné fyzické osobě (jméno, věk, email, IP adresa, atd. . .).

V minulosti se již několikrát stalo, že se velký webový server stal terčem útoku a uživatel se dozvěděl o uniku jeho dat až zpětně z médií. V rámci GDPR však platí, že v případě odcizení dat musí správce nejpozději do 72 hodin od okamžiku zjištění nahlásit příslušným orgánům tuto skutečnost.

GDPR zavádí celou řadu nových pravidel, která hojně podporují bezpečnost dat například, že daná aplikace zpracovává pouze ta data, která jsou ke konkrétním účelům nezbytná. Jde o základní pravidlo bezpečnosti "uchovávej jen ty data, která nutně potřebuješ".

GDPR tedy upravuje zákony tak, aby si webové servery nemohly dělat s daty o uživatelích, co se jim zlíbí. V případě nedodržení některých z podmínek čekají provozovatele webových aplikací obrovské pokuty.

2.2 Open Web Application Security Project

Open Web Application Security Project (OWASP) je nezisková organizace ve Spojených státech zabývající se zlepšením bezpečnosti softwaru v oblasti webových aplikací. Organizace OWASP vznikla 21.4.2004, ale první zprávy jsou na webových stránkách [19] datovány již k roku 2001. Jedná se o otevřenou komunitu, která publikuje dokumenty, nástroje, fóra zcela zdarma a to úplně všem, kteří se zajímají o zlepšení zabezpečení aplikací. Organizace dodržuje svoje základní hodnoty:

- **Otevřená** - všechny informace jsou zásadně otevřené, jak kód, tak finance.
- **Inovace** - povzbuzuje a podporuje inovace a experimenty v oblasti bezpečnosti softwaru.
- **Globální** - je pro každého, na celém světě.
- **Integrita** - čestná, pravdivá a neutrální.

Komunita se snaží dodržovat velký hlavní účel a to být prosperující globální komunitou, která řídí vývoj v oblasti bezpečnosti a zabezpečení světového softwaru zejména v oblasti webových aplikací. Vyžaduje dodržování etického kodexu, který čítá hned celou řadu pravidel. Mezi nejzajímavější pravidla patří:

- Provádět veškeré profesionální činnosti a povinnosti v souladu se všemi platnými zákony a nejvyššími etickými zásadami.
- Podporovat provádění a podporovat dodržování norem, postupů a kontrol bezpečnosti aplikací.
- Uchovávat přiměřenou důvěrnost vlastnických nebo jiných citlivých informací, s nimiž se členové setkávají v průběhu odborných činností.

- Zachovat nezávislost.
- Odmítnout nepřiměřený tlak průmyslu.
- Jednat s respektem a důstojností.

OWASP pravidelně vydává přehled deseti nejvíce nebezpečných bezpečnostních chyb webových aplikací, které budou vysvětleny v následujících kapitolách.

Komunita nabízí nejrůznější druhy volně dostupných nástrojů pro testování bezpečnosti, normy pro bezpečnost kódů, prezentace, videa, doporučené bezpečnostní prvky a knihovny, výzkum a v neposlední řadě pravidelné konference po celém světě.

2.3 Nejčastější typy útoků

V této kapitole budou představeny nejčastější typy útoků, se kterými by se měla rozumná webová aplikace vypořádat. Tyto útoky jsou často prováděny automaticky s využitím robotů a hledají nejméně zabezpečené aplikace na internetu. Tyto aplikace pak roboti zneužívají například k rozesílání spamu, sběru dat, sběru emailových adres, zobrazování nevyžádaných reklam a sběru osobních informacích o uživatelích webové aplikace.

Druhým typem útoků jsou cílené útoky na konkrétní aplikaci, které již provádějí útočníci ručně. Těmto typům útokům se často již ubránit nedá, takovým útokům předchází podrobná analýza webové aplikace, které se útočník snaží zmocnit. Nalezení některých chyb v aplikaci nemusí stát útočníka moc sil a za pomoci volně dostupných nástrojů může najít bezpečnostní slabinu, využít ji a na základě této chyby může získat obrovskou sumu peněz. Pro některé druhy webových aplikací jsou největší hrozbou samotní uživatelé, kteří podceňují rady a pravidla dané aplikace (například pro tvorbu hesla) a usnadňují útočníkům práci.

Pokud nebude uvedeno jinak, jako zdroj informací byla použita literatura OWASP Top 10 - 2017 [18]. V následujícím textu se bude vyskytovat často slovo útočník, tím se rozumí někdo, kdo se snaží prolomit bezpečnost aplikace.

2.3.1 Injection

Údaje, se kterými může uživatel manipulovat, se stávají potenciálním zdrojem vložení škodlivého kódu (code injection). Jedná se o nejvíce převládající možnost útoku, která se vyskytuje zejména ve starších aplikacích. Injekce kódu je známá převážně jako SQL Injekce (SQL injection) , ale již je méně známo, že existuje injekce například LDAP, XPath či NoSQL příkazů.

Důsledkem škodlivého kódu může být například ztráta dat či zveřejnění údajů, které nejsou běžně dostupné.

Aplikaci můžeme považovat za zranitelnou v případě, kdy vstupní data nejsou validována, filtrována a čištěna, nebo v případě, že dynamické dotazy jsou používané přímo v interpretu zdrojového kódu. Vstupní data by se neměla přímo řetězit s výkonným kódem a mělo by docházet k jejich kontrole a případnému escapování. Tento koncept je totožný pro většinu interpretů.

Nejlepší způsob, jak předejít injekcím kódu, je důkladné prozkoumání zdrojového kódu, následované automatizovaným testováním všech parametrů, vstupů, hlaviček atd. Injekci můžeme zabránit úplným oddělením vstupů od příkazů - dotazů, použitím speciálního přístupu, který s pomocí abstrakce odděluje výkonný kód od uživatelského vstupu. Příkladem může být PDO. Druhou a méně spolehlivou možností je komunitou ověřeného rozhraní, které filtruje nebezpečné znaky. Záchranou pojistkou je pak použití omezení výpisů, například v MySql klíčové slovo `LIMIT 1`, pro vrácení jednoho záznamu.

Praktickou ukázkou může být zobrazování článků na základě identifikátoru v adrese (kód 2.1).

```
http://nazev.foo/clanek?id=1
```

Kód 2.1: Url zobrazení článku na základě ID

SQL databáze pak vrací článek s identifikátorem 1. Databázový interpret pak můžeme zmást pomocí vždy splněné podmínky typu `1=1` (kód 2.2).

```
http://nazev.foo/clanek?id=1=' or '1'='1
```

Kód 2.2: SQL injekce v parametru ID

Záleží však na konkrétní implementaci. Typickým příkladem SQL injekce je pak databázový dotaz, který obsahuje tento vložený kód - viz kód 2.3.

```
SELECT * FROM clanek WHERE id = 1=' or '1'='1
```

Kód 2.3: SQL injekce v dotazu nad databází

Kdy se databáze snaží vyhledat články, které mají ID 1 a nebo je splněna podmínka `1=1`. Taková podmínka je však splněna vždy a databázový server vrátí všechen obsah z tabulky článek.

2.3.2 Broken Authentication - Narušená autentifikace

V tomto případě mají útočníci přístup k obrovskému seznamu platných i neplatných uživatelských kombinací jmen a hesel. Existují 2 zdroje uživatelských jmen a hesel:

1. Standardní slovník.

2. Seznam uživatelských jmen a hesel uniklých z jiného systému.

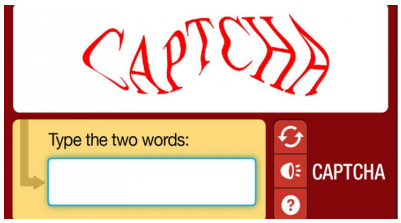
Nejčastěji se setkáváme s druhou variantou. Útočníci aplikují útok Credential stuffing (podmnožina kategorie brutální síly - útok hrubou silou), který zkouší uživatelské jméno a heslo. V případě zdroje ze standardního slovníku pak využívají čistě brutální síly a vytvářejí vlastní kombinace uživatelských jmen a hesel.

Některé seznamy jsou volně dostupné na internetu, některé se prodávají za velkou sumu peněz na černém trhu. Seznamy většinou vznikají prolomením databázového serveru [11], nebo umístěním .sql dumpu zálohy do složky přístupné pro webserver, který obsahuje nešifrované přístupové údaje - což je hrubá chyba.

V závislosti na aplikaci, útočníkům stačí získat přístup k jednomu uživatelskému účtu. V tom horším případě může jít o přístup k administrátorskému účtu.

Obranou proti napadení může být například určení bezpečnostní politiky hesel - uživatel nesmí používat slabá hesla, musí je pravidelně měnit, či nesmí používat stará, již použitá hesla. Určitou alternativou může být zablokování přístupu k přihlašovacímu formuláři po několika neplatných pokusech, nebo nutnost vyplnění většinou obrázkového hesla, takzvaný CAPTCHA kód.

CAPTCHA je Turingův test, který se na webu používá ve snaze automaticky odlišit skutečné uživatele od robotů (například po několika neúspěšných přihlašovacích pokusech). V prvních verzích měla CAPTCHA (2.2) zpravidla podobu obrázku s několika písmeny, které jsou zobrazeny takovým způsobem, že automatický systém bude mít problémy text rozluštit, zatímco člověk dokáže text normálně opsat.

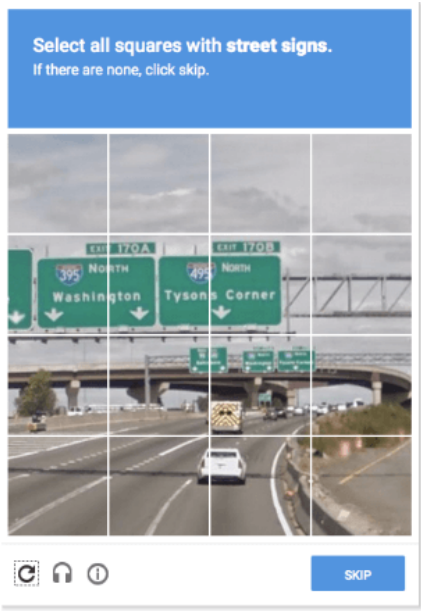


Obrázek 2.2: Textová CAPTCHA

ReCAPTCHA (2.3) v první verzi dokonce pomocí uživatelů pomáhá digitalizovat tištěné média (knihy a časopisy).

Obrázek 2.3: Textová reCAPTCHA

S prudkým vývojem umělé inteligence však přišly programy, které takové textové CAPTCHA kódy dokážou během pár sekund rozluštit. Společnost Google proto přišla s verzí Google Recaptcha v2 (2.4), která na základě pohybu kurzoru dokáže určit, zda-li je uživatel robot. V případě, že si není jistá, zobrazí uživateli kontrolní obrázek s výzvou, ať vybere například všechny části obrázku, kde se vyskytuje název ulice.



Obrázek 2.4: reCAPTCHA v2

Nadstavbou reCAPTCHA v2 je pak neviditelná reCAPTCHA, kterou uživatel vůbec nevidí a nemusí na nic klikat, funguje obdobně jako reCAPTCHA v2.

Dobrým přístupem při validaci uživatelského hesla také může být porovnání zadaného hesla s žebříčkem nejpoužívanějších hesel. Použitím dvoufázového ověřování [2, 12], lze také předejít prolomení uživatelským účtů a v současné době se jedná o velmi rozšířený druh zabezpečení.

Dvoufázové ověření (anglicky two factor authentication) je proces dvou nezávislých způsobů, jak ověřit totožnost uživatele při přihlašování nejen k webovým aplikacím. Jak bylo zmíněno výše v textu, většina služeb používá nejjednodušší jednofázové ověření pomocí uživatelského jména a hesla. V případě, že dojde k úniku hesla, toto ověření přestává být bezpečné.

Nejčastěji je dvoufázové ověření kombinací uživatelského jména, hesla a faktoru vlastnictví. Faktorem vlastnictví může být využití mobilního telefonu nebo speciálního hw klíče. V případě mobilního telefonu pak lze využít běžné sms zprávy nebo specializovanou aplikaci pro generování kódu. Hardwarové klíče se buď připojují do pc přes usb a požívají se k podpisu požadavku, nebo jde o externí zařízení, odkud buď opíšeme aktuální kód, nebo nám přetransformuje kód 1 na kód 2. Existují ale i jiné kombinace, jako jsou například biometrická ověření (otisky prst, oční duhovka, DNA) - faktory vnitřní.

Dvoufázové přihlášení pomocí faktorů vlastnictví podporují například následující webové aplikace:

1. Facebook,
2. GitHub,
3. Google,
4. Amazon Web Services (AWS),
5. Apple ID.

Důležitou roli hraje dvoufázové ověření také v internetovém bankovnictví, kdy vlastník autorizuje své platby pomocí SMS kódu. V dnešní době je to snad již standard, který využívají všechny větší banky (Airbank, Raiffeisenbank, Fio banka, Česká spořitelna, Equa bank).

2.3.3 Získání citlivých dat

Během těchto útoků útočníci často získají klíče a vykonávají útok Man in the middle a odposlouchávají citlivá data během přenosu, či přímo z klientského prohlížeče. Do této kategorie útoků se také řadí například krádež nešifrovaných hesel z databáze serveru. Nejčastější chybou je nedostatek šifrování nebo použití slabého klíče. Selhání často ohrožuje všechna data, která mají být chráněna. Mezi citlivé osobní údaje patří dle GDPR osobní informace, zdravotní záznamy, osobní údaje a kreditní karty. Tyto údaje vyžadují často ochranu dle zákonů dané země, nebo ochranu dle Obecného nařízení na ochranu osobních údajů od EU (GDPR - General Data Protection Regulation).

V první fázi je zapotřebí určit bezpečnostní potřeby v systémua zjistit, jaká data potřebujeme zpracovávat a jakou podmnožinu těchto dat musíme šifrovat. GDPR přímo vyžaduje minimalizaci zpracovávaných dat a neumožňuje zpracování informací, které nutně nepotřebujeme k zákonnému důvodu. Důležité je zajistit bezpečnost přednášení dat, vyhýbáme se nešifrovanému protokolu HTTP a využíváme zabezpečenou verzi v podobě HTTPS. Pokud není možné využít šifrovaný protokol, měli bychom použít vlastní šifrování dat pro účely jejich přenosu. Pokud šifrujeme data, je důležité šifrovat i jejich zálohy, používat nejmodernější a aktualizované šifrovací kryptografické algoritmy, obměňovat klíče. Nutit uživatele používat aktualizované webové prohlížeče, které vyžadují šifrování a platnost všech certifikátů.

V případě, že se útočník dostane do databáze, nevidí původní hesla v čisté podobě (pokud jsou šifrována). Může však použít několik metod, jak toto

opatření oslabit. Bezpečnost uložených, šifrovaných hesel se dá v tomto případě zvýšit tzv. solením. Solení je hashování hesla s přidaným řetězcem, obvykle veřejným v rámci databáze, nejčastěji jde o náhodně generovaný řetězec (tzv. sůl), který je často ukládán do speciálního sloupce v databázi. Generování soli může být ale zajištěno i vlastní šifrovacím algoritmem (password_hash a password_verify v PHP).

Je potřeba zakázat ukládání citlivých údajů do mezipaměti a hesla ukládat pomocí silného adaptivního a "soleného" hashování.

V případě, že se útočník dostane do databáze a odcizí seznam zašifrovaných hesel bez solení, může pomocí útoku hrubou silou útočit na tyto hesla. Útok spočívá v tom, že útočník vlastní seznam hashů a hesel a snaží se je porovnat s nějakým otiskem v databázi. V případě nesoleného hashování to má mnohem jednodušší. Seznamy nejčastějších otisků jsou volně dostupné na internetu. Pokud útok provádí na výpočetním stroji se silným procesorem, nebo na grafické jednotce, může se výsledku dočkat velice rychle.

2.3.4 XML externí entity (XXE)

Útok na externí XML entitu je typ útoku, který analyzuje vstup XML. Tento útok nastává v případě, když XML vstup obsahuje odkaz na externí entitu. Tento útok může vést k odtajnění důvěrných údajů a informací, odmítnutí služeb, nebo padělání požadavků na straně serveru, skenování portů a další systémové dopady.

XML standard verze 1.0 definuje strukturu dokumentu, který obsahuje koncept nazývaný entita, která slouží jako skladovací jednotka pro nějaký typ entity. Existuje několik různých typů entit. Analyzovaná entita se často zkracuje na externí entitu a má přístup k místnímu nebo vzdálenému obsahu prostřednictvím deklarovaných systémových identifikátorů. Identifikátor systému se považuje za URI, který může procesor XML zpřístupňovat při zpracování entit. Procesor XML pak nahrazuje výskyty pojmenované externí entity s obsahem dereferencovaným systémovým identifikátorem. Pokud identifikátor systému obsahuje falešná data a dereference procesoru XML tyto poznamenané údaje, procesor XML může zveřejnit důvěrné informace, které běžně aplikaci nejsou přístupná.

Útoky mohou zahrnovat odhalení lokálních souborů, které mohou obsahovat citlivé údaje, jako jsou hesla, nebo soukromá data uživatelů.

Příkladem útoku může být změna XML, ve kterém se útočník snaží získat data ze serveru (ENTITY xxe SYSTEM) - viz kód 2.4.

```
<?xml version="1.0"encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd»]>
<foo>&xxe;</foo>
```

Kód 2.4: Ukázka XEE

2.3.5 Využití řízení přístupu

Kontrola přístupu, často nazývaná jako ACL (access control list) je způsob, jakým webové aplikace udělují přístup k obsahu a funkcím různých uživatelů. Takové kontroly se provádějí po ověření uživatele a řídí, co mohou oprávnění uživatelé provádět za akce. Řízení přístupu může působit jako jednoduchý problém. Takové řízení přístupu webové aplikace je velice úzce spjato s obsahem a funkcemi, které poskytuje daná webová aplikace. Kromě toho mohou uživatelé spadat do několika skupin či rolí s různými oprávněními.

Vývojáři často podceňují spolehlivé mechanismy řízení kontroly přístupu. Mnoho z těchto mechanismů vzniká společně s vývojem webových stránek a nebylo navrženo a důkladně promyšleno předem. Takový přístup se nazývá "Adhoc". V tomto případě jsou pravidla řízení přístupu roztroušená po celém zdrojovém kódu a stávají se často nepřehlednou množinou pravidel, ve které někdo další snadno udělá chybu. Mnoho z těchto chybných schémat kontroly přístupu není obtížné objevit, či dokonce je zneužít. Následky takových chyb mohou být zničující. Kromě prohlížení neautorizovaného obsahu, může útočník změnit nebo smazat datový obsah, provádět neautorizované funkce, nebo dokonce převzít správu nad webovou aplikací.

Nejčastějším typem těchto problémů jsou kontroly řízení přístupu administrátorských rozhraní, která umožňují správcům webových aplikací spravovat web prostřednictvím Internetu. Takové administrátorské rozhraní slouží k tomu, aby umožnilo správcům efektivně spravovat jednotlivé uživatele, data a obsah ve webových aplikacích. Ve většině případů podporují webové stránky řadu administrátorských rolí, která vede k jemnější granularitě správy aplikací. Vzhledem k síle takových rozhraních jsou často hlavním cílem útoků.

Prakticky všechny stránky mají určité požadavky na kontrolu řízení přístupu. Taková politika kontroly přístupu by měla být jasně zdokumentovaná. Pokud taková dokumentace neexistuje, je větší pravděpodobnost, že webová aplikace je zranitelná.

Kód implementace bezpečnostní politiky by měl být dobře strukturovaný, modulární a pravděpodobně centralizovaný. Měla by být provedena

podrobná kontrola kódu, aby se ověřila správnost implementace kontroly řízení přístupu. Navíc může být penetrační testování[1] velmi užitečné při určovaní, zda jsou v systému řízení přístupu problémy.

Nejčastější implementací bývá tzv. matice řízení přístupu, která definuje jednotlivá pravidla řízení přístupu. Bez dokumentace bezpečnostní politiky však neexistuje žádná definice toho, co znamená být pro danou lokalitu bezpečné. Taková politika by měla dokumentovat, jaké typy uživatelů mohou přistupovat k systému a jaké funkce mohou vykonávat, nebo s jakým obsahem manipulovat, Takový mechanizmus řízení přístupu by měl být dostatečně otestován, aby bylo více než jasné, že neexistuje žádný způsob jak ho obejít.

2.3.6 Chybná konfigurace zabezpečení

Útočníci se často pokouší zneužít nezpracované chyby a výchozí přístupové údaje k účtům. Stejně tak se zaměřují na nepoužívané stránky, nechráněné soubory a adresáře. Nesprávná konfigurace zabezpečení může nastat na všech úrovních aplikace, ať už jde o síťové služby, webovým, aplikační server, databázový server, virtuální stroje atd. Existují automatické skenery, které hledají nesprávnou konfiguraci, výchozí účty či zbytečné služby atd. Tyto skenery ulehčují útočníkům práci. Při získání kontroly nad hlavními prvky webové aplikace, může útočník často převzít kontrolu nad celou aplikací.

Aplikace se považuje za nebezpečnou v případě, že není přiměřeně zabezpečená v jakékoliv části aplikace, nebo je nesprávně nakonfigurovaná, či jsou nainstalovány, nebo zapnuty nepotřebné a zbytečné funkce (porty, služby, účty, oprávnění). V případě, že jsou povoleny výchozí účty a jejich hesla zanechána beze změny, je aplikace také považována za nebezpečnou, či v případě, kdy jsou uživatelům zobrazovány podrobné chybové hlášení.

Je třeba zavést bezpečnostní procesy instalace včetně použití minimalistických platforem bez zbytečných funkcí a s odstranění zbytečný frameworků a komponent. Procesy kontrol aktualizací a konfigurací.

Příkladem může být špatně nastavená konfigurace serveru, při které je dovoleno procházet adresáře. Útočník zjistí, že si může jednoduše vypsat adresáře a najde v nich kompilované soubory jazyka Java. Tyto soubory stáhne a dekompiluje je pomocí reverzního inženýrství, tím pádem si může zobrazit zdrojový kód a může v něm nalézt závažnou chybu, pomocí které může proniknout do systému.

[1] Testování zabezpečení aplikace

2.3.7 Cross-Site Scripting (XSS) - vkládání skriptů

XSS (Cross-site scripting) může být do jisté míry odhalen s využitím automatických nástrojů. Nejčastěji důslednou kontrolou vstupů s využitím regulárních výrazů. Podle OWASP se jedná o nejčastější problém, který se nachází ve dvou třetinách všech aplikací. Problémy se nejčastěji nachází ve vyspělých technologiích jako je například J2EE a PHP.

XSS se dělí na tři kategorie:

- Odražené,
- Uložené,
- DOM.

V případě odraženého XSS se jedná o případ, kdy aplikace obsahuje neupravený a nezkontrolovaný vstup od uživatele, který je součástí HTML výstupu. Takový úspěšný útok může dovolit útočníkovi vykonat libovolný JavaScript kód v prohlížeči napadaného. Obvykle je potřeba takový vstup integrovat s nějakým odkazem. Pomocí takového útoku může být například zobrazena reklama či podvodný obsah.

Uložené XSS ukládá neupravený a nezkontrolovaný vstup od uživatele a je později zobrazen jiným uživatelem. Tento případ je označován jako kritický.

DOM XSS je druh útoku, který je vykonán v prohlížeči uživatele a je důsledkem pozměnění kódu stránky. Může se jednat o změnu URL adresy u některých odkazů v menu nebo o úpravu libovolné textové nebo grafické informace na webu.

Typické útoky pomocí XSS jsou zaměřeny často na krádež relací, účtů, nahrazení uzlů DOMu, podvodné přihlašovací formuláře, útoky na prohlížeč uživatele.

Zabránění XSS útokům v první řadě vyžaduje oddělení nedůvěryhodných dat z aktivního obsahu prohlížeče. Toho lze dosáhnout za pomoci použití moderních frameworků, použitím kontextu citlivého na kódování při úpravě dokumentu v prohlížeči, či povolením Content Security Policy (CSP).

Přikladem může být odcizení mezipaměti prohlížeče - viz kód 2.5.

```
(String) page += "<input name='creditcard' type='TEXT' value='" +
request.getParameter("CC") + "'>";
```

Kód 2.5: Odcizení mezipaměti prohlížeče

kdy útočník pozmění v prohlížeči parametr CC (kód 2.6).

```
'><script>document.location='http://www.attacker.com/cgi-
bin/cookie.cgi?foo='+document.cookie</script>'.
```

Kód 2.6: Změněný parametr cc

2.3.8 Použití komponenty s chybou

Programy se skládají často z knihoven a pro mnohé je jednodušší použít hotové řešení než si psát vlastní. Pomocí tohoto řešení si můžeme do vlastní aplikace zanést nebezpečný kód, který může naší aplikaci ohrožovat. V závislosti na kontextu aplikace by mělo být riziko tohoto problému bráno na vědomí.

Riziko napadení díky těmto chybám vzniká za předpokladu, že nejsme informování o verzích knihoven, které používáme, nebo pokud je další software (OS, web/aplikační server, databáze, ...) v aplikaci nepodporovaný, zastaralý. Netestování kompatibility aktualizovaných knihoven zvyšuje také riziko napadení aplikace.

Předejít napadení se dá zabránit zavedením procesu spravování komponent, ve kterém by docházelo k odstranění nepoužívaných a nepotřebných závislostí, komponent, souborů a dokumentů. Nejlepší řešením je použití specializovaných nástrojů jako je například Retire.js[2], sledování zdrojů CVE[3] (Common Vulnerabilities and Exposures) a NVD[4] (National Vulnerability Database). Samozřejmostí je stahovat komplenty pouze z oficiálních úložišť ideálně s využitím balíčkovacích systémů typu Composer nebo npm.

2.3.9 Nedostatek záznamů a monitorování

Základem každého programu je dostatek záznamů o jeho chování a monitorování jeho běhu. Útočníci se často spoléhají na nedostatek těchto informací, včasné reakce administrátorů a spoléhají na to, že dosáhnou svých cílů bez toho aniž by byli detekováni.

Příkladem je chybějící monitoring nesprávného přihlášení nebo velkého množství dostazů na aplikaci nebo databázovou vrstvu v krátkém čase. Je důležité zajistit záznamy o veškerých přihlášeních, řízení přístupu, vstupu, chyby ověření a identifikovat podezřelé účty.

[2]https://retirejs.github.io/retire.js/
[3]https://cve.mitre.org/
[4]https://nvd.nist.gov/

Protokolování by nemělo být vidět uživatelem, jinak může dojít opět k ohrožení na základě těchto informací. Zároveň musí být jasné a srozumitelné, aby se v něm vyznali i správci webové aplikace. Jednotlivé záznamy je nutné generovat v jasném a srozumitelné formátů, který lze snadno získat z databáze. Velmi důležité je také stanovit plán reakcí na nehody a obnovy.

Názornou ukázkou může být případ, kdy útočník skenuje uživatele, kteří používají stejné heslo. Útočník je schopen se dostat pomocí jednoho hesla do všech účtů, které toto heslo používají a u všech ostatních zanechá pouze jedno chybné přihlášení. V tomto případě by se podle záznamů o přístupu k účtům a záznamů o chybném přihlášení by se tato nekalá aktivita dala odhalit, klidně i automaticky.

2.4 Další základní útoky

V této kapitole budou představeny další méně významné základní útoky. Většina uvedených útoků může být filtrována pomocí základního nastavení moderních frameworků, pomocí kterých se většina webových aplikací v dnešní době vytváří.

2.4.1 Path Traversal Attack

Často známý jako cesta k adresáři [15] v originálním názvu Path Traversal Attack má za cíl přistoupit k souborům a adresářům, které jsou uloženy mimo kořenovou složku webového adresáře. Manipulaci s parametry adresy, které odkazují na soubory, může být možné přistupovat k libovolným souborům a adresářům uloženým na souborovém systému včetně zdrojového kódu aplikace nebo konfigurace a kritických systémových souborů. Je třeba poznamenat, že přístup k souborům je omezen řízením přístupu operačním systémem. Tento útok je také znám (v originálním anglickém znění) jako dot-dot-slash, directory traversal, directory climbing and backtracking.

Takovému útoku se dá předejít tak, že máme seznam povolených souborů k zobrazení v případě, že je pak zadán jiný soubor, nebude zobrazen a cestu k adresáři si neposíláme v URL, ale je pevně daná v aplikaci.

Chyba je demonstrována na následujícím příkladě. Mějme adresu http:localhost/?page=home.html.

Útočník se pak může pokusit nahradit parametr page za cokoliv - viz kód 2.7.

```
http://localhost/?page=/../../../../etc/shadow

http://localhost/?page=/etc/passwd
```

Kód 2.7: Změněný parametr page

V případě že je webový server špatné nastaven a stránka nedostatečně kontroluje vstupní parametry, může dojít k zobrazení obsahu adresářů na základě parametru v adrese, což je je kritická chyba.

2.4.2 Nezabezpečený přímý odkaz na objekt

Velká část webových aplikací využívá pro svou navigaci jména souborů, nebo primární klíče z databáze, které jsou součástí adresy jako parametry. To dává možnost útočníkům pozměnit parametry a doufat v to, že se jim místo obrázku, na který mají uživatelské práva zobrazí obrázek jiného uživatele.

Tento útok úzce souvisí s útokem v předchozí kapitole 2.4.2. Útok změny cesty. Rozdíl je však v tom, že soubory jsou podle jména čteny přímo z databáze. Nejčastějším řešením pak bývá zavedení přístupových práv pro každý objekt, který je uživateli zobrazován, či použitím nepřímé reference na objekt.

Příkladem může být zobrazení obrázku na základě identifikátoru uživatele (kód 2.8.)

```
<img src="obrazek.php?id=1"alt="obrazek1»
```

Kód 2.8: Možný nezabezpečený odkaz

Útočník v tomto případě může libovolně přistupovat ke kterémukoliv obrázku, ke kterému zná identifikátor. Obrana proti tomuto druhu útoku je taková, že před přístupem k obrázku musí být provedena kontrola oprávnění, zda si daný uživatel může zobrazit vybraný obrázek viz kód 2.9.

```
if(!hasPermissionToReadUser($currentUser, $id)) {
    die("Nemáte oprávnění pro zobrazení obrázku");
} else {
    echo $obrazek;
}
```

Kód 2.9: Ukázka ošetření práv

2.4.3 Únos akce formuláře

Jinak řečeno Form action hijacking dovoluje útočníkovi zadat adresu URL akce formuláře pomocí parametru adresy. Útočník je schopen vytvořit adresu URL, která změní adresu akce formuláře tak, aby odkázala na server útočníka a to včetně Cross-Site Request Forgery (CSRF) tokenů.

Předejít takovému útoku se dá nepoužíváním dynamických hodnot v adrese a natvrdo je zapisovat do zdrojového kódu. V případě, že z nějakého důvodu potřebuje dynamicky měnit adresu akce formuláře v URL, použijeme tzv. whitelist. Whitelist je seznam povolných adres. Pokud tedy útočníka adresa ve whitelistu není, akce nebude serverem provedena.

Následující adresa vygeneruje akci formuláře pomocí parametru url. Po odeslání formuláře bude id a heslo odesláno na adresu útočníka. Příkladem může být adresa znázorněná v následujícím kódu 2.10.

```
http://localhost/?url=https://attackersite.com
```

Kód 2.10: Ukázka nebezpečné adresy

V případě použití takové adresy má pak zdrojový kód formuláře následující tvar:

```
<form name="form1" id="form1" method="post"
  action="https://attackersite.com">
  <input type="text" name="id" value="user name">
  <input type="password" name="pass" value="password">
  <input type="submit" name="submit" value="Submit">
</form>
```

Kód 2.11: Kód nebezpečného formuláře

2.4.4 Kontrola dostupnosti dat verzovacích systémů (.git)

Občas se stává, že společně se zdrojovým souborem aplikace je na sever nahrán celý adresář, který obsahuje různé konfigurační soubory pro vývoj a to včetně souborů o verzovacím systému, ve většině případu jsou tyto soubory v tomto adresáři pak dostupné přes webový prohlížeč a dají se stáhnout.

Takové soubory v rukou někoho, kdo má nekalé úmysly, mohou být nebezpečné. Dá se z nich totiž zjistit poslední stav zdrojových kódů.

Obranou proti takovému bezpečnostnímu pochybení může být nějaký automatický nástroj, který testuje přítomnost konfiguračních souborů. Další obranou může být například manuální kontrola člověkem nebo zakázání cesty v .htaccess.

2.4.5 Skenování otevřených portů

Skenování otevřených portů nemusí být v každým případě považováno za útok, spíše jen za získávání informací před útokem (nemusí se vždy týkat pouze webové aplikace).

Jedná se o metodu, pomocí níž útočník získává seznam otevřených síťových portů na vzdáleném počítači v počítačové síti. Na základě těchto dat lze zjistit, jaké služby na daném počítači běží. Běžně může jít například o službu webového serveru, sdílené disky, atd... Pro skenování síťových portů se používají speciální programy. Mezi nejznámější patří program s názvem nmap [13].

2.5 Crawler (robot pro stahování obsahu webu)

V následující kapitole bude popsáno, co je to webový crawler a jakým způsobem funguje. Tato kapitola byla zařazena do textu, z důvodu seznámení čtenáře se základním principem funkčnosti takového robota, jelikož je v praktické části hojně používán.

Crawler je robot [16], který prochází web za účelem vytvoření indexu. Tento index poté slouží k různorodým činnostem, nejčastěji pro vyhledávání. Crawler automaticky prochází webové stránky a ukládá si slova, která kde nalezl.

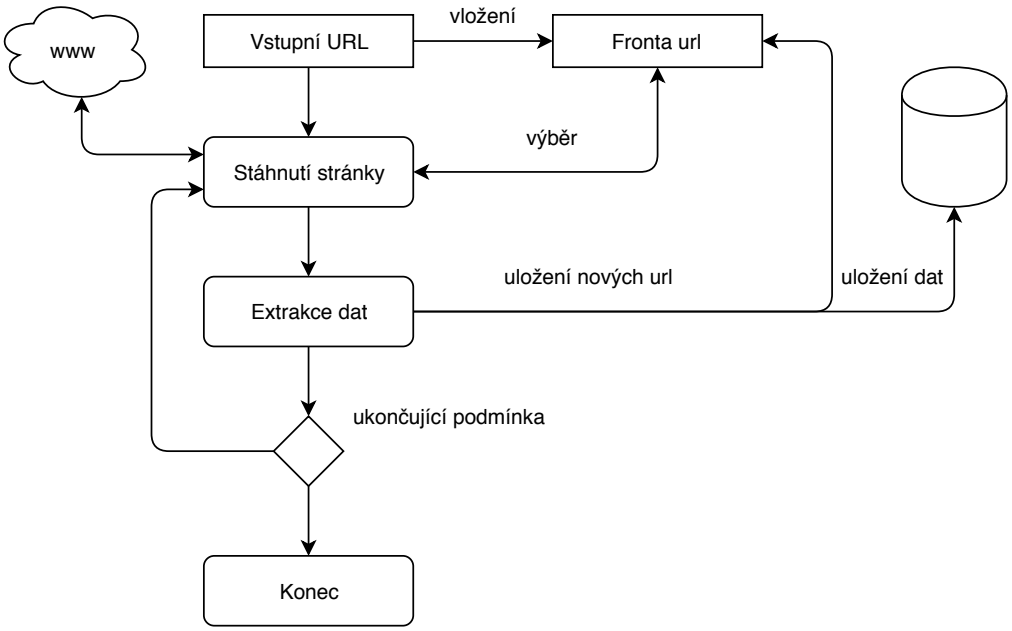
Život takového robota začíná seznamem adres, které má navštívit. Na každé adrese se podívá, kam vedou další odkazy a ty si poznamená k dalšímu procházení. Obvykle crawlery, které indexují obsah webových stránek, si ukládají nalezený obsah na pevný disk.

Teoreticky je na celém Internetu nekonečné množství webových stránek. Roboti se řídí svojí politikou, která se liší v závislosti na použitém crawleru, která stanoví způsob procházení a indexace stránek.

Crawler se potýká s celou řadou problémů a překážek, které musí překonat. Musí například normalizovat nalezené adresy, či být odolný vůči pastím (tzv. spider trap), které vedou k zacyklení.

Schéma funkčnosti robota je vidět na obrázku 2.5.

1. Vložení vstupní URL, která se uloží do fronty.
2. Robot vyzvedne URL z fronty a zpracuje.
3. V rámci zpracování proběhne stáhnutí obsahu URL a extrakce dat (dle typu robota).
4. Extrahovaná data jsou uložena do databáze.
5. Robot pokračuje výběrem nové adresy a pokud neexistuje tak je ukončen.



Obrázek 2.5: Schéma crawleru

3 Existující aplikace pro testování bezpečnosti

Bezpečnost webových aplikací není nic nového a celá řada nástrojů pro automatické testování již existuje. V této kapitole budou popsány některé zajímavé a populární nástroje, včetně jejich funkcionality.

3.1 Kali Linux

Kali Linux[1] je linuxová distribuce odvozená od Debianu, která je speciálně navržená pro penetrační testy [1]. Tato linuxová distribuce je určena pro potřebu bezpečnostních testů a chová se stejně jako každý jiný plnohodnotný operační systém. Velice často jsou na tento operační systém navázány různé komerční aktivity typu tréninkový kurz bezpečnosti. Tento operační systém již prošel řadou změn, aktuálně (9.4.2018) se nachází ve své 20 verzi Kali 2018.1.

Systém obsahuje celou řadu nástrojů [21], které jsou určeny nejen pro testování bezpečnosti webových aplikací. V aktuální době je součástí systému Kali 371 nástrojů pro testování bezpečnosti.

V následující části textu uvádím některé příklady nástrojů a balíčků, které jsou součástí distribuce Kali Linux a lze pomocí nich testovat zapezpečení webových aplikací. Mimo operační systém Kali Linux je většina balíčků distribuovaná jako veřejný gitový repozitář s volně dostupným kódem.
Sql injekce:

- sqlmap,
- Sqlninja,
- sqlsus,
- jSQL Injection.

Skenování portů:

- dnmap,
- Nmap.

[1] www.kali.org

Slovníkový útok:

- acccheck,
- THC-Hydra,
- John the Ripper.

XSS:

- XSSer.

Path Traversal Attack:

- DotDotPwn.

Využití ostatních chyb:

- Metasploit Framework.

3.2 OWASP Zed Attack Proxy Project (ZAP)

Jedná se o jeden z nejvíce populárních bezplatných bezpečnostních nástrojů na světě, který je aktivně podporován stovkami mezinárodních dobrovolníků. Software je napsaný v programovacím jazyce Java a čítá mnoho funkcí pro testování bezpečnosti internetových aplikací. ZAP je součástí standardní distribuce Kali Linuxu. Mezi moduly testovacího programu patří:

- Intercepting Proxy který poskytuje analýzu, modifikaci a v některých případech injekci do spojení mezi klientem a serverem.
- Automatizovaný skener pro testování základních útoků na webové aplikace (například XSS a Sql injekce).
- Pasivní skener pro skenování všech HTTP zpráv (požadavků i odpovědí) odeslaných na server.
- Skener hrubé síly pro testování kombinací přihlašovacího jména a hesla.
- Fuzzer pro hledání základních implementačních chyb pomocí datové injekce.
- Skener portů pro skenování portů.
- Spider pro nalezení všech adres v dané aplikaci.
- Testování bezpečnosti Web Socketu a REST API.

3.3 Burp Suite

Aplikace od firmy PortSwigger distribuovaná ve verzi Professional (placená) a Community Edition (zdarma, omezená funkcionalita) je relativně dobře hodnocený nástroj, který dokáže otestovat celou řadu bezpečnostních útoků [14].

Mezi jeho klíčové funkce patří automatický skener, který dokáže otestovat více než 100 obecných zranitelností, jako je například Sql injekce, XSS, Path Traversal Attack, XML externí entity, atd. Součástí programu pak jsou další skenery, jako jsou aktivní skenery a pasivní skenery. Program se dále může pochlubit testováním webového provozu pomocí proxy "man-in-the-middle".

Oproti předchozím uvedeným programům je Burp Suite placený program, jeho pořizovací náklady jsou vysoké a program je navíc licencován přímo pro uživatele. Pokud by si ho chtěla koupit firma pro více uživatelů, musí si pořídit více licencí, což může být nevýhoda.

4 Analýza a návrh aplikace pro testování bezpečnosti

V této kapitole bude popsána analýza aplikace implementované v rámci diplomové práce. V rámci analýzy byla navržena architektura a funkcionalita aplikace pro testování bezpečnosti webových aplikací.

Problém rozdělíme na obecnou charakteristiku aplikace, která popisuje hlavní požadavky na nástroj pro testování bezpečnosti a na požadované funkce a také způsob jakým bude aplikace pro testování bezpečnosti fungovat či bude zabezpečená.

Na závěr této kapitoly bude popsáno jakým způsobem probíhala volba technologií, ve kterých bude aplikace pro testování bezpečnosti webových aplikací implementována.

Tato kapitola pohlíží na program jako na softwarový produkt.

4.1 Obecná charakteristika

4.1.1 Rozsah

Vyvíjený software bude rozdělen na dvě části, každá z těchto částí může i nemusí běžet na jiném výpočetním stroji, avšak budou spolu vzájemně komunikovat.

Protože existuje celá řada bezpečnostních problémů a útoků na webové aplikace, implementuji pouze vybrané základní typy útoků. V souladu se zadáním bude třeba implementovat:

- uživatelské rozhraní,
- backendovou testovací službu a jádro,
- Slovníkový útok,
- Cross-site scripting (XSS),
- SQL injection,
- Path Traversal,
- Form action hijacking,

- Inline JavaScript,
- Test http vs https,
- Cross-Origin Resource Sharing (CORS),
- Kontrola dostupnosti dat verzovacích systémů (.git),
- Scan otevřených portů.

Aplikace bude sloužit k testování vybraných druhů útoků v rámci celé webové aplikace dle zadaného url. Vstupem aplikace bude odkaz na domovskou stránku a nastavení testovacího systému. Uživatel bude moci vytvářet jednoduché úlohy, které budou řazeny do fronty a postupně vykonávány na serverové části.

V dnešní době je většina aplikací dostupná nepřetržitě a online. Proto jsem se rozhodl program implementovat s využitím webových technologií jako tenkého klienta, který bude komunikovat se serverem, na kterém poběží testovací jádro. UI (uživatelské prostředí - user interface) bude jen prostředník pro zadávání jednotlivých úloh. Dále bude poskytovat přehled o jejich stavu a částečnou konfiguraci. Samotné testovaní bude probíhat na serveru.

4.1.2 Softwarový produkt

V dnešním softwarovém světě, nese každá aplikace hrdě své jméno a logo, které ji symbolizuje. Aplikace byla pojmenována “RedFace”.

Toto pojmenování (RedFace) jsem vybral na základě toho, že se aplikace snaží zabránit agresivnímu chování uživatelů. Další její úlohou je zajistit bezpečnost. Slovo - red v angličtině znamená červená a zde symbolizuje agresivitu. Níže můžete vidět, jaké bylo vytvořeno logo pro mojí aplikaci (obrázek 4.1).

Obrázek 4.1: Logo aplikace RedFace

4.2 Funkce aplikace

V následující kapitole budou detailně popsány jednotlivé funkce nástroje pro testování webové bezpečnosti. Dále bude popsáno jakým způsobem bude aplikace fungovat a jak budou jednotlivé operace rozděleny.

4.2.1 Systém úloh

Celá aplikace bude formou úloh. Každá úloha bude reprezentovat testovací scénář, který bude definován vstupním nastavením. Nastavením se v tomto případě rozumí to, jaké druhy útoků budou puštěny. Každý útok bude možno podrobně nastavit.

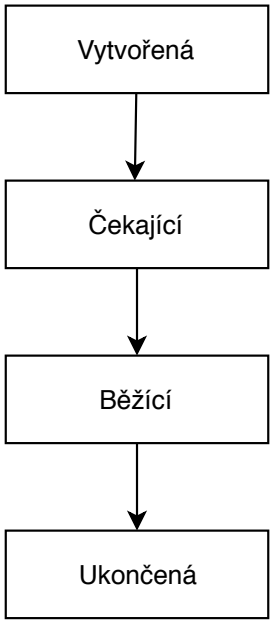
Úlohy budou systémem řazeny ve frontě typu FIFO[1] a budou spuštěny dle nastavení celé aplikace. Pod tímto si lze představit to, kolik administrátor dovolí pustit úloh paralelně, aby počítač, na kterém budou úlohy spuštěny nebyl extremně vytížen.

Jednotlivé úlohy budou seřazeny v přehledné tabulce, úlohy bude možno filtrovat dle stavu:

- Vytvořená,
- Čekající,
- Běžící,
- Ukončená.

[1]First In, First Out, což se do češtiny zpravidla překládá jako první dovnitř, první ven

Graf přechodů jednotlivých typů úloh je vidět na obrázku 4.2.



Obrázek 4.2: Graf přechodu úloh

Jak bylo zmíněno výše, úloha může nabývat čtyř stavů. V případě, že uživatel vytvoří úlohu, bude úloha ve stavu vytvořená a bude přesunuta plánovačem do stavu čekající, kdy bude čekat na volného exekutora. Následně, jakmile bude exekutor volný, přejde úloha do stavu běžící a bude provádět samostatné testování

Úloha ve stavu běžící půjde zastavit a úloha ve stavu ukončená, půjde opakovat se stejnou konfigurací. Ve stavech běžící a ukončená půjde zobrazit detail úlohy.

4.2.2 Testování

Každá úloha bude představovat testovací scénář. Uživatel si bude moci vybrat více druhů testování (viz kapitola 4.1.1) a spustit úlohu. Uživatel bude neustále informován o stavu své úlohy. V případě, že bude úloha spuštěna, bude si uživatel moc zobrazit průběžný log z testovaní a po dokončení zobrazit finální log, nebo úlohu bude moci znovu spustit. Ve stavu kdy bude úloha probíhat, ji bude možné zastavit a po dokončení smazat ze systému.

Součástí testovací aplikace bude modul, který bude zobrazovat informace o aktuálním stavu programu a serveru. V přehledu bude graficky zobrazeno aktuální zatížení serverového procesoru, podíl zastoupení jednotlivých typů

úloh. Dále si bude možné prohlédnout počet úloh ve frontě, počet právě aktivních úloh a také změnit velikost fronty a to z toho důvodu, aby server mohl pustit paralelně více úloh současně.

4.3 Výběr technologií

Tato kapitola shrnuje důvody výběru technologií pro implementaci aplikace pro testování webové bezpečnosti. Dále zmiňované technologie přicházely v úvahu pro vývoj aplikace. Technologie nebyla daná zadáním a je tedy možné jednotlvé technologie zvolit. Výběr proběhl čistě na základě mých preferencí a zájmu.

4.3.1 Uživatelské rozhraní

V dnešní době je trendem mít aplikaci online na internetu, aby byla dostupná odkudkoliv a kdykoliv. S moderními technologiemi je to možné a není to příliš obtížné. Mezi nejznámější knihovny, které by bylo možné použít, patří Angular a React. Každá z těchto knihoven má svojí podporu a komunitu. Po domluvě s vedoucím byl vybrán React.

Jelikož komunita obsahuje velké množství uživatelů, kteří pravidelně vyvíjí nové knihovny a tím dávají možnost ostatním je většinou bezplatně (na základě různých typů OpenSource) využívat, existuje celá řada rozšíření pro React. Jedním z nich je designový framework Semantic UI, který slouží pro tvorbu uživatelského prostředí. Semantic přináší celou řadu komponent, uživatelských a ovládacích prvků včetně jejich stylování. Podpora a komunita pro tento framework nabývá obřích rozměrů, což vedlo k jeho výběru.

4.3.2 Backend

Jak již bylo řečeno, React je knihovna pro tvorbu uživatelského rozhraní, je tedy zřejmé, že uživatelské rozhraní webové aplikace musí komunikovat s nějakým webovým serverem. V úvahu přichází hned několik cest, například PHP, JAVA, NodeJS, atd...

Jak jsem se bylo zmíněno v předchozí kapitole (4.3.1), testuji webové aplikace psané v jazyce JavaScript. JavaScript patří mezi populární technologii a jako nejvhodnější se zdá NodeJS.

4.3.3 Testovací jádro

Asi neznámějším nástrojem pro automatické testování nejen webových aplikací je Selenium [10], které poskytuje podporu v řadě populárních programovacích jazyků[2]. Jedná se o software s otevřeným zdrojovým kódem, který je vydán pod licencí Apache 2.0, to znamená, že ho weboví vývojáři mohou stáhnout a používat bez poplatků.

Se Seleniem pracuji již druhým rokem na profesionální úrovni, a proto jsem se ho rozhodl využít ve své práci, nehledě na další již existující nástroje

Vhodným prohlížečem pro testování webových aplikací v kombinaci se Seleniem je internetový prohlížeč Chrome v režimu Headless, pod tím si lze představit to, že prohlížeč může běžet skrytě bez GUI v podstatě na jakýmkoliv OS, jelikož podpora prohlížeče Chrome je vysoká.

4.3.4 Databáze

V dnešní době existuje již velká řada databází, avšak stále populární databází je MySQL s formátem uložení dat InnoDB. Aplikace neklade velké nároky na databázi, v úvahu však přichází více druhů databází.

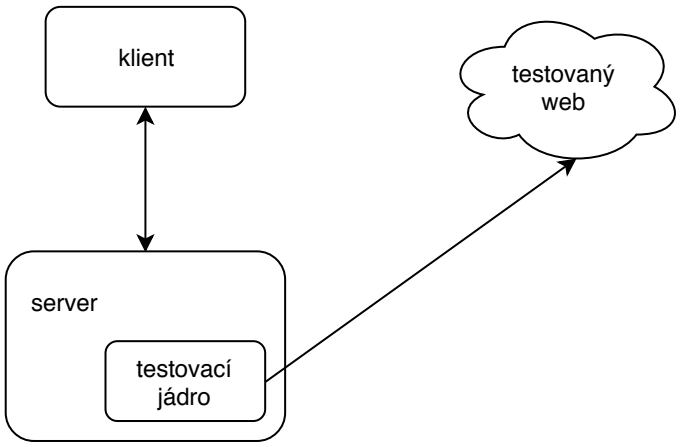
Jelikož bude aplikace pracovat s velkým množstvím záznamů z testů, bylo by možné použít například Logstash[3], který za pomoci Easticsearche [4] umožňuje zpracovat velké množství logovacích záznamů. Logstash je primárně určen práci s velkým objemem logů, ve kterých se bude vyhledávat. V aplikaci však takové techniky nebudou potřeba, a proto jsem se rozhodl využít MySQL, se kterou mám bohaté zkušenosti, které jsem během svého studia pochytil ve škole i v praxi.

[2] C#, PHP, Python, Java, JavaScript, ...
[3] https://www.elastic.co/products/logstash
[4] https://www.elastic.co/

5 Aplikace pro testování bezpečnosti

Následující kapitola popisuje, jakým způsobem je systém pro testování bezpečnosti webových aplikací realizován. Celá aplikace byla rozdělena do tří hlavních částí (viz. obrázek 5.1), které budou v následujících kapitolách popsány. Konkrétně se jedná o:

- **klient** - komunikuje se serverem, umožňuje vkládání, nastavení a zobrazení vytvořených úloh,
- **server** - poskytuje API a stará se o komunikaci, spravuje testovací jádro,
- **testovací jádro** - provádí samotné testování, je součástí serverové části.



Obrázek 5.1: Znázornění rozdělení aplikace

5.1 Klient

Klientská část je implementováná v JavaScriptové knihovně React, která poskytuje modularitu programu, to znamená, že se aplikace skládá z více částí, které spolu vzájemně komunikují a posílají si zprávy pomocí tzv. “Props”. Každá část uživatelského rozhraní je obvykle většinou jedna komponenta. V případě, že komponenta potřebuje komunikovat se serverem, má svůj vlastní kontroler, který je její součástí.

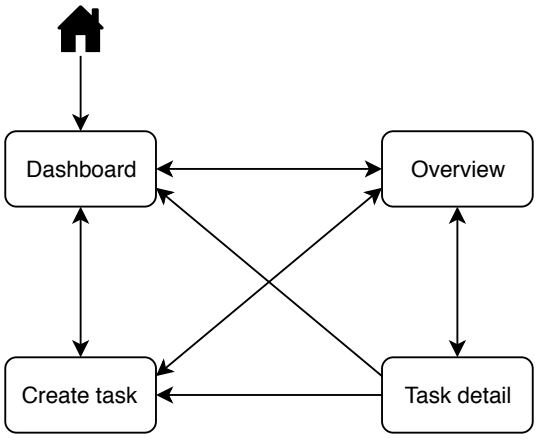
V této kapitole nebudu popisovat jednotlivé moduly, ale vyberu jen některé nejzajímavější a obecně popíši jakým způsobem funguje klientská část.

5.1.1 Struktura

Klienta tvoří čtyři části, na které je aplikace (obrázek 5.2) rozdělena, jmenovitě to jsou:

- Dashboard,
- Overview,
- Create task,
- Task detail.

Každá z těchto částí je jednotlivá stránka, po které se může uživatel volně pohybovat. Zároveň je každá tato sekce komponenta, která tvoří aplikaci.



Obrázek 5.2: Struktura aplikace

5.1.2 Komponenty a jejich obsluha

Všechny kompenenty aplikace dědí od hlavní komponenty Reactu, která je zahrnuta v balíku React. Tento předek přináší našim potomkům funkce, které jsou pro React typické a klíčové.

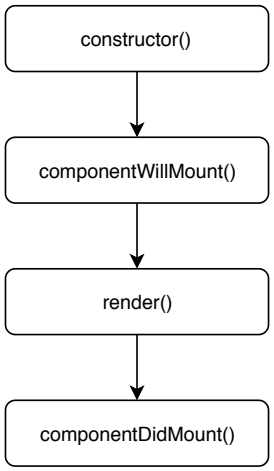
Jako zdroj informací byla použita dokumentace Reactu [9] ve verzi 15. Text se nemusí shodovat s Reactem ve verzi 16, ve které jsou plánované změny.

Životní cyklus (obrázek 5.3) začíná voláním konstruktoru (`constructor`), který je zavolán před tím, než je samotná komponenta nasazena. Při implementaci konstruktoru se běžně volá předek pomocí volání `super`, kterému jsou jako parametry předány tzv. `props` vlastnosti. V konstruktoru se běžně inicializují proměnné třídy a stav komponenty (`state`).

Po inicializaci konstruktoru je volána funkce `componentWillMount`, která je spuštěna těsně před kódem těla obsluhy vykreslení. V této funkci je doporučeno například stahovat data ze serveru (pokud je potřeba), či jiné pomocné operace, které jsou nutné udělat před vykreslením komponenty.

Následuje samotné vykreslení, které vrací elementy pro vykreslení. Je doporučeno, aby funkce byla čistá, to znamená, že přímo neupravuje stav komponenty a vrací stejný výsledek vždy, když je volána. Udržování "čistoty" ve funkci `render` ji činní více přehlednou.

Funkce `componentDidMount` je volána ihned po (prvním) vykreslení, stejně tak jako `componentWillMount` je tato funkce doporučena k načítaní dat ze serveru, dále je vhodná pro nastavení všech dalších funkcí (například `setInterval`).

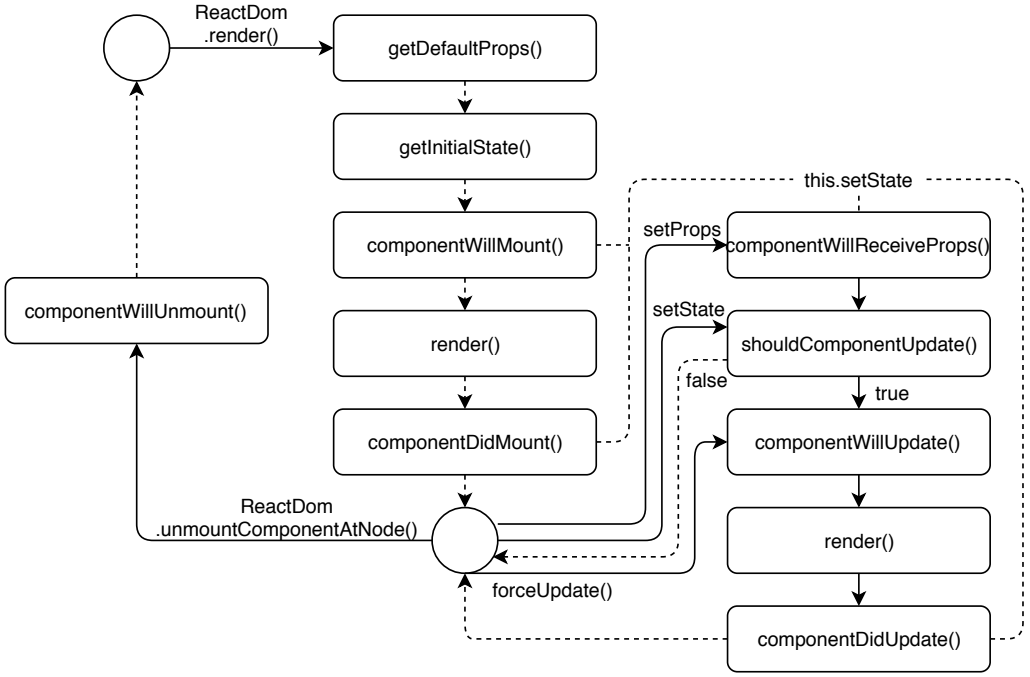


Obrázek 5.3: Životní cyklus komponenty

Reactová komponenta obsahuje další funkce, které jsou stejně důležité jako výše jmenované (označované zaváděcí). Další funkce se dělí do skupin aktualizační, odstraňovací a chybové.

Mezi aktualizační funkce patří `componentWillReceiveProps`, která je volána pokaždé, kdy komponenta přijímá nové vlastnosti, pokud potřebujeme aktualizovat stav na základě změn vlastností komponenty, může k tomu být použita například tato funkce. Funkce `shouldComponentUpdate` je volána před každým vykreslením komponenty a může ovlivnit vykreslení. V případě, že tato funkce vrací `false`, není vykonáno vykreslení pomocí funkce `render`.

Ve skutečnosti je celý životní cyklus (obrázek 5.4) komponenty poměrně složitý a jeho popsání by mohlo být návrhem zadání pro jinou samostatnou práci.



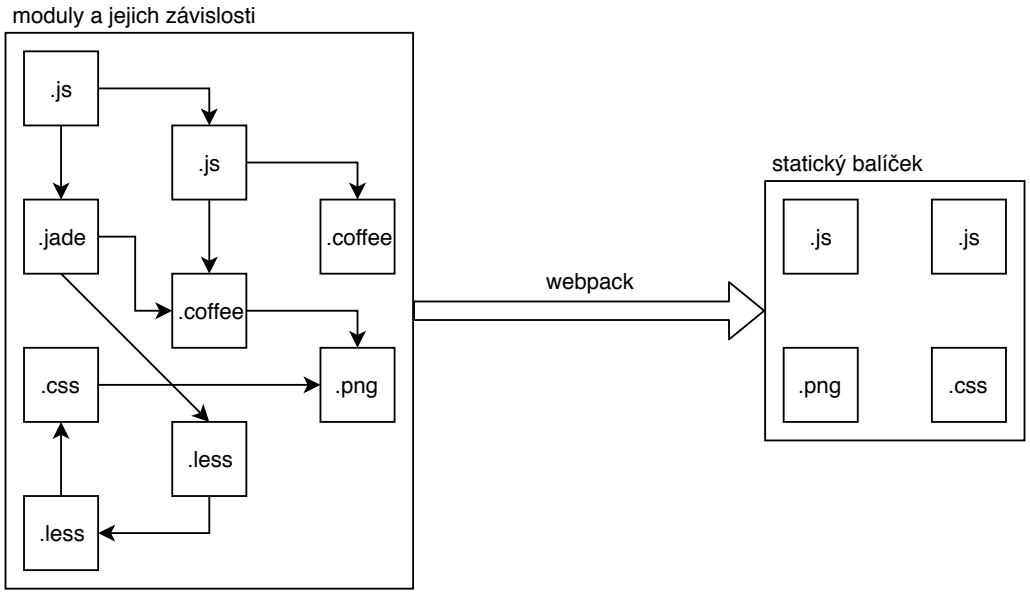
Obrázek 5.4: Kompletní životní cyklus komponenty

Použití Webpack a Babelu vede k optimalizaci zdrojových souborů. Webpack se dá použít ve více režimech. Jedním z nich je použití dle konfigurace ze vstupního souboru, tak je použit v práci. Nastavení je uloženo v `webpack.config.js`.

5.1.3 Sestavení

Takzvaný build, jinak řečeno sestavení se používá pro vytvoření balíčku, který je prohlížeč schopen interpretovat. K tomuto účelu jsou použity knihovny Webpack a Babel. Fyzicky se v Javascriptu nic nekompiluje, ale i přesto budeme v následující kapitole používat slovo build (sestavení), jedná se totiž o běžně používaný termín, součástí takového buildu je stažení a instalace závislostí. Navíc dochází k minimalizaci a spojení zdrojových souborů, což vede ke zvýšení rychlosti a zmenšení velikosti kódu.

Webpack [24] přebírá moduly se závislostmi a generuje (obrázek 5.5) z nich statické prostředky představující tyto moduly. Tyhle statické prostředky vytváří na základě grafu závislosti. Často se používá společně s modulem Babel, který transformuje zdrojový kód do podoby, které rozumí prohlížeč, provádí také minifikaci.



Obrázek 5.5: Princip Webpacku

5.2 Server

Pro implementaci serverové části byl použit softwarový systém Node.js, který umožňuje psaní vysoce škálovatelných internetových aplikací.

Mezi základní funkce serveru patří:

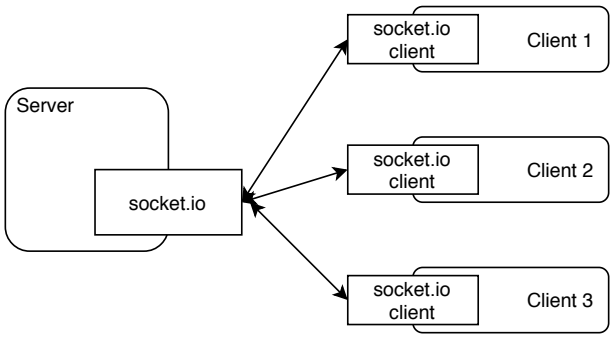
- klientské API - zpracování a poskytnutí dat klientům,
- komunikace s databází - modul mysql,
- správa testovacích úloh - spuštění testovacích úloh,
- poskytování statických souborů - webový server.

5.2.1 Klientské API

Komunikace klienta a serveru probíhá přes WebSocket, který má definovaný seznam zpráv. Každá zpráva je reprezentována JSONem (JavaScript object notation). Jednotlivé JSON zprávy budou detailně popsány v následujících kapitolách.

Knihoven zajišťující websocketovou podporu je celá řada, jednou z nejznámější je knihovna Socket.io[1] [23], která umožňuje celou řadu operací. V programu si vystačím pouze se základní funkcionalitou, jako je například funkce emit(název, data) pro odeslání dat nebo socket.on(název, handler). Knihovna Socket.io byla použita jak pro serverovou i klientskou část.

Socket.io tvoří mezi každým klientem kanál, přes který komunikují (viz obrázek 5.6). V pokročilejší fázi použití mohou klienti komunikovat i mezi sebou bez nutnosti přeposílát svou zprávu přes webový server.



Obrázek 5.6: Schéma komunikace Socket.io

[1] https://socket.io/

5.2.2 Databáze

Komunikace s databází je zajištěna pomocí modulu mysql. V případě požadavku o komunikaci s databází se zavolá příslušná metoda serveru, která sestaví příslušný dotaz.

Při spuštění serveru je provedeno připojení k databázi, toto připojení ilustruje ukázka zdrojového kódu 5.1.

```
const connection = mysql.createConnection(
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'red-face',
    // debug: true,
)
```

Kód 5.1: Inicializace modulu MySQL

Jednotlivé požadavky se pak provádí nad proměnnou connection, pomocí volání query, jejíž parametry jsou: SQL dotaz, případné parametry, zpětné volání (obsluha výsledku dotazu). Jedná se o asynchronní metodu. Ukázku použití lze vidět ve zdrojovém kódu 5.2.

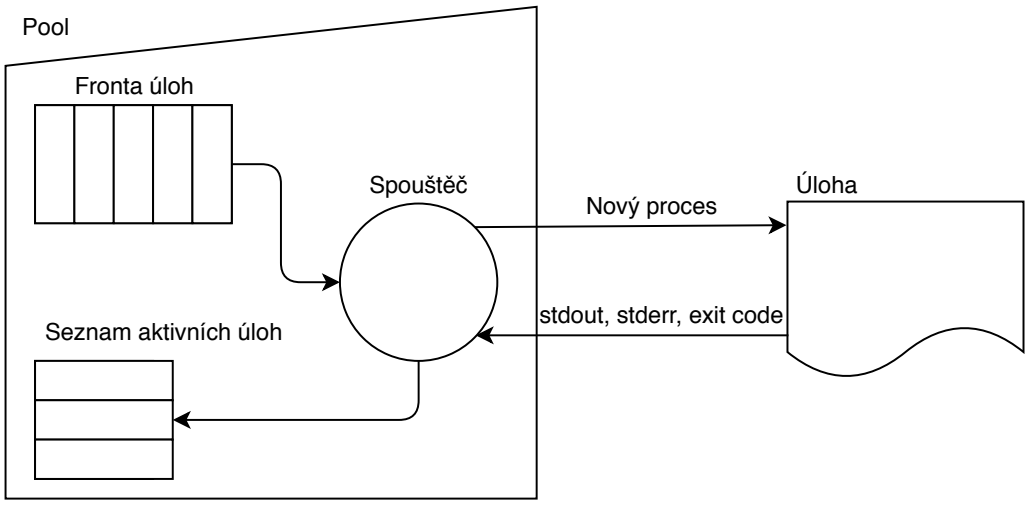
```
let params = [
  taskHome.TaskState.killed,
  library.getMySQLTime(),
  taskHome.TaskState.running,
  taskHome.TaskState.created]
connection.query('UPDATE task SET state = ?, endTime = ? WHERE
(state = ? OR state = ?)', params, (err) => {
    if (err) {
        logger.log('error', err)
        return
    }
})
```

Kód 5.2: Ukázka použití dotazu nad databází

5.2.3 Správa testovacích úloh

Jádro obsahuje tzv. Pool (obrázek 5.7), který drží informace o úlohách, které čekají na zpracování, nebo jsou právě zpracovávány. Pool je tvořen neprioritní frontou úloh, ve které čekají úlohy na zpracování. Jelikož je počet spuštěných úloh omezen (aby nedošlo k neúměrnému zatížení procesoru), může se stát, že ve frontě je více úloh.

Samotné spuštění úloh probíhá pomocí modulu child_process, tento modul pomocí asynchronního volání funkce spawn zajišťuje spuštění nového procesu. Nový proces pak běží zcela izolován od serveru, separace zajišťuje, že v případě neočekávané chyby testovací úlohy nedojde k pádu serveru. Rodičovský proces může z potomka číst standardní výstup, chybový výstup a návratový kód. Čtením standardního výstupu a následným zápisem do souboru je tvořen log úlohy.



Obrázek 5.7: Schéma systému úloh

5.2.4 Poskytování statických souborů

Takzvaný webový server, poskytuje statické html stránky klientům. V případě aplikace vytvořené v knihovně React pouze index.html a Javascriptové soubory vendor tvořící jádro, které vznikly sestavením a minifikací[2].

API je vytvořeno pomocí modulu express, což je populární framework pro tvorbu aplikací v Node.js. Poskytuje vše, co od webového serveru požadujeme. Jak je vidět v ukázce kódu 5.3. ve skutečnosti je web server tvořen jedním "middleware", který pokrývá všechny požadavky.

[2]Vytváření js balíčků z modulárního js kódu pro použití v prohlížeči

```
webServerApplication.get('*', (req, res) => {
    res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'))
})
```

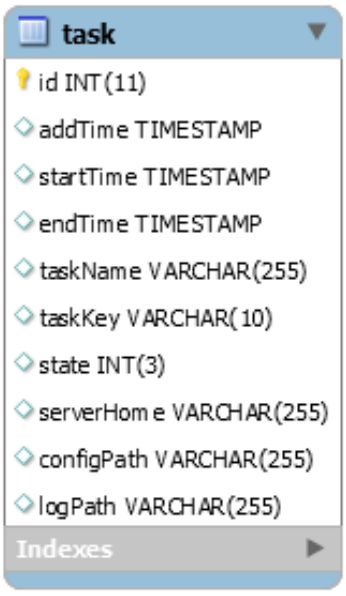
Kód 5.3: Middleware webového serveru

Tento endpoint vrací vždy kořenový soubor index.html k němuž zajišťuje cestu modul path, který zařizuje správné vytvoření systémové cesty do složky client/build/ ve které se nachází sestavená aplikace.

Celý tento web server naslouchá standardně na portu 80, ale může být změněn přes proměnné prostředí, které jsou dostupné v `process.env.PORT`.

5.2.5 Struktura databáze a konfigurace úloh

Databázové schéma je tvořeno pouze jednou tabulkou (viz obrázek 5.8), která obsahuje všechny potřebné informace pro samotný běh úlohy. Konfigurace a log úlohy není uložen v databázi, ale v pracovním adresáři spuštěné aplikace.



Obrázek 5.8: Schéma databáze

Jednotlivé sloupce jsou vysvětleny v následující tabulce 5.1.

Název	Datový typ	Význam
id	Integer	Id záznamu
addTime	Timestamp	Čas vytvoření úlohy
startTime	Timestamp	Čas startu úlohy
endTime	Timestamp	Čas dokončení úlohy
taskName	Varchar	Název úloh
taskKey	Varchar	Klíč úlohy - slouží k zobrazení detailu o úloze
state	Integer	Stav ve kterém se úloha nachází
serverHome	Varchar	Domovská URL, která bude testována
configPath	Varchar	Cesta ke konfiguraci
logPath	Varchar	Cesta k logu

Tabulka 5.1: Význam sloupců v tabulce

Doplnění tabulky 5.1:

- configPath - Cesta k soboru na disku, který obsahuje konfiguraci úlohy v datovém formátu JSON.
- logPath - Obsahuje cestu k fyzickému textovému souboru na disku, do kterého je zapisován log testu, v průběhu testování se jeho obsah mění.

5.3 Testovací jádro

5.3.1 Crawler

Většina testů předem vyžaduje prohledat zadaný web za účelem zjištění množiny adres, které je možné otestovat. K tomuto účelu slouží modul s názvem crawler-task.js, který je napojen na externí knihovnu local-web-crawler[3] jejímž jsem autorem a který vznikl výhradně za účelem této diplomové práce.

Úkolem crawleru je vytvořit množinu všech dostupných url a množinu stránek, který obsahuji elementy dle předem definované XPath, například je crawler schopen vrátit množinu všech url, které obsahuji element typu `//input`.

Z důvodu ochrany před zacyklením má robot definovanou hloubku zanoření. To znamená, kolik může maximálně projít stránek, než skončí, pro neomezený počet je hodnota rovna -1.

[3]https://www.npmjs.com/package/local-web-crawler

Jádro této knihovny tvoří Selenium s prohlížečem Chrome v režimu headless. V první fázi si robot inicializuje tzv. seed, url adresu na které bude prohledávání začínat. Robot pošle request na url, zjistí jaký je content-type v odpovědi hlavičky a v případě že se jedná o text/html přejde browser na aktuální url (Selenium volá get), v případě že ne, pokračuje dál.

Když se robot nachází na nějaké url, zjistí, zda-li aktuální url obsahuje nějaké další url voláním funkce `executeAsyncScript` nad Seleniovým driverem, v těle této funkce lze pak vykonat jakýkoliv JavaScriptový kód - `document.getElementsByTagName('a')` získá všechny linky a vrátí je. Po následné normalizaci odkazů a filtraci linků, které nevedou na stejnou doménu, ukládá crawler nalezené linky do pole. Při extrakci všech atributů se jménem a jsou extrahovány a kontrolovány zadané XPathy, které od crawlera požadujeme. Robot si pak pouze ukládá true v případě že element našel, nebo false v opačném případě.

Výsledky crawleru jsou dále předány další části aplikace, která s nimi pracuji při testování.

5.4 Implementace testů

5.4.1 Slovníkový útok

Vstupním nastavením slovníkového útoku jsou tři XPath řetězce, které definují elementy v DOMu:

1. Formulář - kód 5.4.

```
//form//input[@type='text' or @type='email']//ancestor::form
    //input[@type='password']//ancestor::form
```

Kód 5.4: Formulář

2. Input pro přihlašovací jméno - kód 5.5.

```
(//form//input[@type='text' or @type='email']//ancestor::form
    //input[@type='password']//ancestor::form//input)[1]
```

Kód 5.5: Input pro přihlašovací jméno

3. Input typu password pro heslo - kód 5.6.

```
(//form//input[@type='text' or @type='email']//ancestor::form
//input[@type='password']//ancestor::form//input)[2]
```

Kód 5.6: Input typu password pro heslo

Ve většině případů jsou tyto cesty stejné, většina přihlašovacích formulářů webových stránek vždy obsahují výše popsané.

Dalším vstupem algoritmu je seznam přihlašovacích jmen a hesel, ze kterých jsou vytvořeny všechny možné kombinace dvojic, které jsou postupně zkoušeny v přihlašovacím formuláři.

Aby byl celý proces rychlejší, vybírá se počet paralelních procesů, které zkouší uhádnout heslo.

Procentuální rozdíl stránek určuje, jak moc se má lišit stránka úspěšného přihlášení od stránky s chybovým přihlášením.

Slovníkový útok je možné pustit ve dvou režimech:

- automatický,
- manuální.

V případě automatického výběru přihlašovacího formuláře je použit crawler, který hledá tři výše popsané XPathy. Při manuálním výběru adresy stránky je stránka zadána uživatelem.

Postup zpracování je zobrazen v následujícím kódu 5.7.

```
1. Odešli přihlašovací formulář s náhodným jménem a heslem
2. Ulož chybovou stránku
3. Odešli formulář se jménem a heslem
4. Ulož stránku
5. Porovnej stránku z kroku 2 a 4
6. Pokud je rozdílnost menší než zadaná - konec
7. Krok 3, pokud nejsou jména a heslo - konec
```

Kód 5.7: Pseudokód Slovníkového útoku

Slovníkový útok je prováděn pomocí Selenia. V první fázi je otevřena stránka s přihlašovacím formulářem a je odeslána s náhodnými hodnotami. Obsah stránky s chybovým přihlášením je uložen do paměti. Samotné odeslání formuláře probíhá nalezením elementu pomocí XPathy (to podporuje

Selenium) a zavoláním funkce `submit()`, která odešle formulář. Následně probíhá samotný pokus přihlášení kombinací jméno - heslo, po každém odeslání formuláře je uložen obsah stránky a porovnám s obsahem z kroku dva. Pokud je procentuální rozdílnost menší než zadaná hodnota, jméno a heslo bylo uhodnuto.

5.4.2 Cross-site scripting (XSS)

XSS (kód 5.10) test se dělí na dvě části:

1. test formulářů,
2. test parametrů.

V obou případech test vychází z výsledku crawler.

XSS chyby jsou testovány pomocí kódu, který může vyvolat JavaScriptový alert box. V případě, že je XSS útok úspěšný a zobrazí se javascriptové upozornění, může být za pomoci Selenia odhaleno. Okénko upozornění může být v testovacím kódu umístěno jakýmkoliv způsobem. Příkladem mohou být dva následující zápisy.

Například čistě JavaScriptovým kódem(kód 5.8).

```
<script>prompt(1)</script>
```

Kód 5.8: Ukázka testovacího kódu

Nebo chybovou funkcí při zobrazení obrázku(kód 5.9).

```
<img src='neexistuji' onerror=confirm(1)>
```

Kód 5.9: Ukázka chybové funkce obrázku

Testovací funkce jsou uživatelským vstupem.

V případě testu formulářů známe stránky na kterých se nachází vstupní elementy. Vstupním elementem se rozumí input a form.

Program si udržuje v paměti jednotlivé stránky a označení, zda-li obsahují jeden z výše zmíněných elementů, v případě že ano, přejde na danou adresu a vloží do cílového elementu text. Vložení textu zajišťuje Selenium, které je schopné dohledat element pomocí XPathy a vložit do něj testovací

kód. Po vložení testovacího kódu, je kontrolováno, zda-li existuje upozornění (v případě, že element obsahuje funkce `onChange`, nebo jinou), v případě že ne, pokusí se Selenium dohledat formulář elementu a odeslat ho (funkcí `elemet.form.submit`), po odeslání je opět kontrolována přítomnost upozornění.

```
1. Iteruj přes seznam adres, které obsahují input/form
2. Vlož text do elementu a kontroluj přítomnost upozornění
3. Pokud neexistuje upozornění, odešli formulář
```

Kód 5.10: Pseudokód XSS útoku

V případě, že se jedná o test parametrů, program postupně prochází množinu adres, které obsahují parametry, následně jsou parametry zaměněny za testovací kód a pomocí Selenia je adresa zobrazena. Pokud je přítomné upozornění, je adresa s parametrem označena jako nebezpečná.

5.4.3 SQL injection

Základem injekce SQL kódu je množina adres, kterou testovacímu algoritmu předá webový crawler. Algoritmus iteruje přes jednotlivé adresy a v případě, že adresa obsahuje parametry, jsou tyto parametry postupně zaměňovány za parametry nové.

Přikladem může být změna parametru ID, který obsahuje klíč příspěvku, který má být zobrazen. Například ID=1, v případě injekce SQL je změněn na ID=1 OR 1=1 a tím je vytvořená nová adresa, která je pomocí Selenia zobrazena.

Zobrazená stránka je uložena do paměti a porovnána s chybovou stránkou, na základě porovnání je určena procentuální rozdílnost a v případě, že je vyšší než předem definované X, adresa je určená jako zranitelná.

5.4.4 Path Traversal

Kontrola cest v adrese funguje na základě výstupu z webového robota. V první kroku algoritmus vybere množinu adres, která obsahuje parametry a je tak kandidátem na test.

Jednotliví kandidáti na test jsou postupně procházeni a v každém kroku iterace jsou všechny parametry postupně zaměněny za cestu k testovanému souboru. Následně je adresa otevřena ve webovém prohlížeči pomocí Selenia. Každý takový testovací soubor má přiřazenou klíčovou větu, která se pak na stránce nachází.

Příkladem může být soubor hosts, který je standardně součástí operačního systému Windows a je často umístěn na adrese C:/Windows/System32/drivers/etc/hosts. Ve většině případů obsahuje tento soubor komentář ve tvaru `# This is a sample HOSTS file used by Microsoft TCP/IP for Windows`. V okamžiku, kdy už je stránka zobrazena, nastává fáze, ve které se kontroluje přítomnost (pomocí funkce `contains`) komentáře v DOMu. Pokud je nalezen, můžeme s vysokou pravděpodobností říci, že jsme zobrazili soubor. Na webových serverech s operačním systémem Linux můžeme kontrolovat například soubor /etc/passwd s obsahem `root:x:0:0:root:/root:/bin/bash`.

5.4.5 Form action hijacking

Testování útoku vychází z výsledků, které poskytne webový crawler. Test iteruje přes množinu adres, která obsahuje formulář. V případě že adresa v sobě obsahuje parametry, je dále zpracovávána. Následně jsou procházeny parametry v adrese, pokud jich je víc, test vyzkouší všechny. Test nahradí parametr za testovací řetězec a přejde na novou adresu s nahrazeným řetězcem.

Pomocí Selenia je pak nová adresa s nahrazeným parametrem zobrazena a je kontrolováno, zda-li formulář obsahuje akci, která byla zadána v parametru. Kontrola akce formuláře probíhá pomocí vykonání JavaScriptového kódu, který čte akci formuláře: `$0.action` (kdy `$0` je proměnná s objektem formuláře)

Lepší demonstraci funkce testu ilustruji na následujícím příkladu:

Adresa s formulářem obsahuje v parametru url odkaz na akci formuláře(kód 5.11).

```
http://localhost/?page=/view/login.php&url=/scripts/login.php
```

Kód 5.11: Ukázka nebezpečné adresy

Formulář přímo čte parametr bez další kontroly(kód 5.12).

```
<form action='<?php echo $_GET['url'];?>' method='post'>
```

Kód 5.12: Ukázka čtení parametru URL

5.4.6 Inline JavaScript

Kontrola přítomnosti JavaScriptového kódu přímo ve stránce (jinak řečeno Inline JavaScript) probíhá na základě výsledků z crawleru. Pomocí Selenia test prochází jednotlivé adresy a kontroluje na nich přítomnost kódu.

Selenium vykonává JavaScriptový kód, který je spuštěn přímo v prohlížeči nad aktuálně zobrazenou stránkou.

Pomocí `document.scripts` jsou vráceny všechny kódy na aktuální stránce, následně je nad každým kódem provedena kontrola přítomnosti zdrojového kódu src. V případě, že src neexistuje, je stránka označena jako nebezpečná.

5.4.7 Test http vs https

Testování protokolu zajišťuje knihovna Puppeteer, která slouží jako komunikační rozhraní mezi aplikací v NodeJS a webovým prohlížečem Chrome.

Puppeteer přejde na domovskou adresu testovaného serveru a vykoná nad ním pomocí funkce `evaluate` (kód 5.13) JavaScriptový kód, který vrátí protokol.

JavaScriptový kód `location.protocol` vrací protokol, přes který jsou zobrazeny aktuální stránky. Výsledek testu je pak určen porovnáním získaného protokolu s klíčovým slovem “https”.

```
let protocol = await page.evaluate(() =>
    return location.protocol
)
```

Kód 5.13: Kód získání protokolu

5.4.8 Cross-Origin Resource Sharing (CORS)

Test CORS probíhá kontrolou hlavičky odpovědi ze serveru. Na domovský server se odešle žádost o data a z odpovědi je pak patrné, zda-li je CORS povolené. V případě že `response.headers['access-control-allow-origin']` se rovná `*`, v opačném případě ne. Během tohoto testu je důležité brát na vědomí, že CORS může být povolený pouze pro některé endpointy a aplikace může komunikovat na pozadí s jinou adresou než ze které ve skutečnosti je vracen obsah stránky.

5.4.9 Kontrola dostupnosti dat verzovacích systémů (.git)

V systému je definovaný výčet položek, které obsahuje konfigurace verzovacího systému GIT. Samotná kontrola pak probíhá tak, že se testovací software dotazuje webového serveru na existenci jednotlivých souborů v kořenovém adresáři.

Standardní výčet prvků standardního verzovacího systému GIT je následující:

- .git/COMMIT_EDITMSG,
- .git/FETCH_HEAD,
- .git/HEAD,
- .git/ORIG_HEAD,
- .git/branches,
- .git/config,
- .git/description,
- .git/hooks/,
- .git/hooks/applypatch-msg,
- .git/hooks/commit-msg,
- .git/hooks/post-commit,
- .git/hooks/post-receive,
- .git/hooks/post-update,
- .git/hooks/pre-applypatch,
- .git/hooks/pre-commit,
- .git/hooks/pre-rebase,
- .git/hooks/prepare-commit-msg,
- .git/index,
- .git/info/exclude,

- .git/logs/HEAD,
- .git/logs/refs/,
- .git/objects,
- .git/refs/heads/,
- .git/refs/remotes/,
- .git/refs/stash/,
- .git/refs/tags/.

Tento algoritmus (kód 5.14) je implementován pomocí Selenia. V první fázi si algoritmus uloží chybovou 404 stránku, následně algoritmus pokračuje procházením pole s konfigurací a zkouší přejít na jednotlivé adresy ve tvaru host/prvekKonfigurace. Po každém přechodu si uloží vrácenou stránku a porovná ji s chybovou z kroku jedna. Pokud se liší o zadané X, je určena jako nalezená část konfigurace verzovacího systému GIT.

```
1. Jdi na domovskou stránku /nahodnyText
2. Ulož si chybovou stránku 404
3. Čti z pole z konfigurací a přejdi na /itemZPole
4. Ulož si stránku
5. Porovnej procentuální odlišnost
6. Pokud je menší než zadaná - našel jsi itemZPole
7. Krok 3
```

Kód 5.14: Pseudokód kontroly dostupnosti dat verzovacích sytémů (.git)

5.4.10 Scan otevřených portů

Test otevřených portů zajišťuje knihovna, která nese název net-scan a je součástí komunity npm. Modul zajišťuje postupné procházení portů nad zadaným IP. V případě, že je port aktivní, je vráceno číslo portu, které je následně posláno do modulu portu-numbers, který drží seznam portů a jmen služeb, které nad daným portem standardně běž. V tomto případě je nutné brát na vědomí, že pojmenování služby běžící na otevřeném portu serveru je pouze orientační a ve skutečnosti tam může běžet úplně jiný protokol.

5.5 Komunikace

V následující kapitole bude detailně popsána komunikace a typy zpráv mezi klientem a serverem. V aplikaci jsou následující typy zpráv:

- get-system-stats,
- system-stats,
- set-system-settings,
- task-create,
- give-me-tasks,
- give-me-task-detail,
- there-is-task-detail,
- remove-task,
- repeat-task,
- stop-task,
- remove-all-tasks ,
- detail-*,
- task-start,
- task-done.

Na úvod této kapitoly budou popsány významné zprávy, které jsou doplněny kapitolou 5.5.3 další zprávy, ve které bude popsán zbytek zpráv.

5.5.1 Zpráva system-stats

Zpráva je odpověď serveru na get-system-stats, obsahem zprávy jsou systémové statistiky (tabulka 5.2).

Vlastnost	Datový typ	Význam
cpu	Float	Procentuální zatížení CPU
activeProcess	Integer	Počet aktuálně běžících procesů
maxProcess	Integer	Velikost fronty
queueStatus	Integer	Počet procesů ve frontě
stats	Array	Počet úloh v jednotlivých stavech

Tabulka 5.2: Obsah zprávy system-stats

5.5.2 Zpráva give-me-task-detail

Reprezentuje žádost klienta o detaily o úloze (5.3).

Vlastnost	Datový typ	Význam
id	String	Id úlohy
addTime	Timestamp	Čas přidání úlohy
startTime	Timestamp	Čas startu úlohy
endTime	Timestamp	Čas dokončení úlohy
taskName	String	Jméno úlohy
taskKey	String	Tajný klíč úlohy
state	Integer	Aktuální stav úlohy
serverHome	String	Domovská adresa serveru
log	String	Aktuální log úlohy

Tabulka 5.3: Obsah zprávy give-me-task-detail

5.5.3 Další zprávy

Aplikace obsahuje další celou řadu zpráv. Zbytek zpráv a jejich význam je popsán v této kapitole.

Zpráva get-system-stats reprezentuje požadavek klienta o systémové statistiky serveru. Zpráva nemá žádné tělo.

Zpráva give-me-tasks je pouze požadavek klienta na o informace o všech úlohách. Tělo zprávy není žádné.

Zpráva task-done (tabulka 5.4) je zpráva, která je poslaná všem klientům a informuje je o změně stavu z některé z úloh.

Vlastnost	Datový typ	Význam
running	Integer	Počet aktuálně běžících úloh
pending	Integer	Počet úloh ve frontě
taskdone	Integer	Id právě dokončené úlohy
endTime	Integer	Časová značka dokončení úlohy

Tabulka 5.4: Obsah zprávy task-done

Zpráva there-is-task-detail (tabulka 5.5) slouží jako odpověď serveru na požadavek klienta s informacemi o úloze.

Vlastnost	Datový typ	Význam
key	String	Klíč úlohy

Tabulka 5.5: Obsah zprávy there-is-task-detail

Pro smazání úlohy slouží zpráva remove-task (tabulka 5.6).

Vlastnost	Datový typ	Význam
id	Integer	Id úlohy, která má být odstraněna

Tabulka 5.6: Obsah zprávy remove-task

Zpráva repeat-task (tabulka 5.7 je pouze požadavek klienta serveru o opakování úlohy.

Vlastnost	Datový typ	Význam
id	Integer	Id úlohy, která má opakovat

Tabulka 5.7: Obsah zprávy repeat-task

Zpráva detail-* (tabulka 5.8) slouží jako komunikační prostředník pro příjem nové řádky v logovacím souboru. V případě, že úloha právě probíhá, do logu přibývají nové zprávy, jsou posílány na tento kanál. Označení zprávy je ve tvaru detail-ID úlohy.

Vlastnost	Datový typ	Význam
data	String	Nová řádka v logu

Tabulka 5.8: Obsah zprávy detail-*

Broadcastová zpráva task-start (tabulka 5.9) slouží jako informace pro všechny klienty, že úloha se specifickým ID byla právě zahájena.

Vlastnost	Datový typ	Význam
id	Integer	Id aktuálně spuštěné úlohy

Tabulka 5.9: Obsah task-create

Zpráva task-create reprezentuje žádost klienta na vytvoření nové úlohy. Tělo zprávy obsahuje kompletní konfiguraci úlohy.

Zpráva set-system-settings (tabulka 5.10) je zpráva klienta s nastavením maximálního počtu aktuálně zpracovávaných úloh.

Vlastnost	Datový typ	Význam
maxActiveTasks	Integer	Maximální počet paralelně běžících úloh

Tabulka 5.10: Obsah set-system-settings

6 Testování

Jedním z úkolů mé diplomové práce je provést testování implementované aplikace. V následujících kapitolách se budu zabývat způsoby, které byly pro testování použity.

Jelikož byl systém implementován jako webová aplikace, lze na testování nahlížet z celé řady stran [4]. Je možno testovat bezpečnost, zatížení serverů/klientů, kompatibilitu s prohlížeči, či požadovanou funkcionalitu. V této kapitole se budu věnovat podrobně testování požadované funkcionality a ostatních metody budou představeny jen stručně.

6.1 Kompatibilita

6.1.1 Kompatibilita uživatelského rozhraní

Jelikož je uživatelské rozhraní implementováno pomocí známého frameworku Semantic UI, který by měl být již dostatečně otestován, můžeme vycházet z dokumentace [6], ve které je výčet podporovaných prohlížečů.

- Nejnovější 2 verze FF, Chrome, Safari Mac,
- IE 11+,
- Android 4.4+, Chrome pro Android 44+,
- iOS Safari 7+,
- Microsoft Edge 12+.

Namátkou byly otestovány některé prohlížeče, zda-li v nich aplikace funguje, chová se korektně a layout není rozbitý. Nebylo však provedeno kompletní testování, protože to není obsahem této práce.

6.1.2 Server

S pohledy kompatibility serverové části je již o něco složitější, v rámci testování musíme brát v úvahu hned několik částí a to:

- podpora NodeJS a NPM,
- podpora knihoven NPM,
- podpora ChromeDriveru a Chrome v režimu headless.

Jak uvádí dokumentace NodeJS [17] jehož je NPM obvykle součástí, prostředí je podporováno operačními systémy Windows (x86/x64), macOS, Linux (x86/x64), Linux ARM.

NPM knihovny tvoří uživatelé, není proto garantována kompatibilita s výše uvedenými systémy. Všechny knihovny jsou vybrány tak, aby je bylo možné spustit na operačním systému Windows a Linux.

ChromeDriver je dostupný [5] pro všechny výše uvedené operační systémy stejně tak jako Chrome v režimu headless.

Tímto lze kompatibilitu považovat formálně za splněnou, k tomu jsem server otestoval na dvou rozdílných systémech, jmenovitě Windows 10 (64 bit) s UI, Debian GNU/Linux 8 (jessie) (64 bit) bez UI, Ubuntu 17.10.1 (64 bit) s UI.

6.2 Manuální a automatické testování

Valná většina uživatelských rozhraní webových aplikací se testuje pomocí Selenia (které je v práci použito). Vývoj automatický testů je v podstatě souběžně běžící projekt a zabere velké množství času. Často je potřeba testy přizpůsobit rychle se měnící aplikaci. Z tohoto důvodu nebyly automatické testy implementovány a vše bylo testováno manuálním způsobem.

“Ad hoc testing”, jinak řečeno manuální testování bez plánování, dokumentace a scénářů probíhalo v průběhu vývoje celé aplikace mnou a ještě jedním testerem. Během testování byl brán ohled na správnou funkcionalitu všech prvků UI, na korektní chování aplikace a na správně zobrazený layout.

6.3 Funkční testy a Proof of concept

Jinak řečeno ověření funkčnosti daného nástroje. K tomuto účelu byly vytvořeny dvě minimalistické aplikace stejné funkcinolity s rozdílnou implementací.

6.3.1 Popis testovacích aplikací

Testovací aplikace nesou název "TODO". Úkolem aplikací je spravovat seznam úkolů, které má uživatel udělat. Vlastnosti aplikace jsou následující:

- Všechny úlohy jsou dostupné všem uživatelům.
- Lze si zobrazit všechny úlohy.
- Lze si zobrazit detail úlohy.
- Lze se registrovat.
- Úlohy lze mazat.

Aplikace komunikují se stejným RestApi.

Testovací aplikace obsahují záměrně implementované zranitelnosti, aby bylo možné na nich ověřit funkcionalitu vytvořeného nástroje.

6.3.2 Zranitelnosti testovacích aplikací

V následující kapitole budou popsány zranitelnosti uživatelských rozhraní dvou testovacích aplikací a serverové části tzv. RestAPI.

Zranitelnosti uživatelského rozhraní popisují tabulky 6.1 a 6.2, kdy každá tabulka obsahuje adresu, název a jednoduchý popis útoku.

Adresa	**Útok**	**Popis**
/?page=	Path Traversal	Možno vložení cesty k soboru ?page=C:\hello.txt
/?page=/view/list.php	XSS	Ve formuláři pro vytvoření "TODO" záznamu
/?page=/view/detail.php&id=1	SQL Injection	V query id, id=1 OR 1=1
/?page=/view/login.php&url=/scripts/login.php	Form action hijacking	V query url
/?page=/view/detail.php&id=1	InlineJS	Ve zdrojovém souboru detail.php
/?page=/view/login.php&url=/scripts/login.php	Brute force	Prolomení hesla

Tabulka 6.1: Popis zranitelností v PHP testovací aplikaci

Adresa	**Útok**	**Popis**
/list	XSS	Ve formuláři pro vytvoření "TODO" záznamu
/?page=/view/detail.php&id=1	SQL Injection	V query id, id=1 OR 1=1
/?page=/view/detail.php&id=1	XSS	V query id, id=XSS
/?page=/view/login.php&url=/scripts/login.php	Brute force	Prolomení hesla

Tabulka 6.2: Popis zranitelností v React testovací aplikaci

Rest API, se kterým komunikují obě testovací aplikace obsahuje SQL injection chybu v endpointu "/detail/:id/", ve kterém není vstupní řetězec escapován a je vložen do dotazu tak, jak přijde.

6.3.3 Porovnání Seleniových prohlížečů

V rámci své diplomové práce se také zaměřím na porovnání dvou Seleniových prohlížečů. Selenium je pouze prostředek pro ovládání některého z internetových prohlížečů. Mezi nejznámější patří:

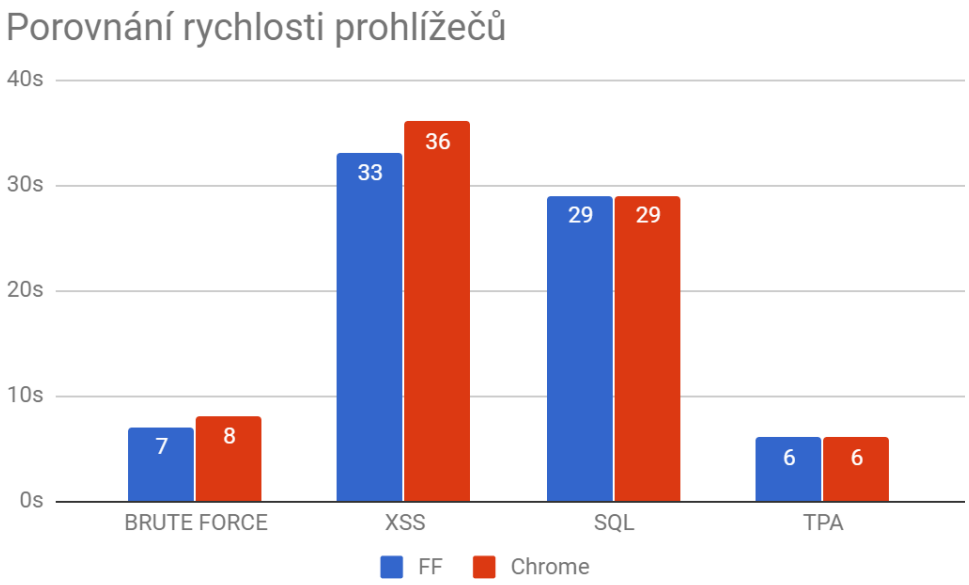
- Mozilla Firefox,
- Google Chrome.

Porovnání proběhlo opakovaným puštěním jednotlivých testů. Výsledný čas je určený jako průměr. Testování probíhalo na testovací aplikaci React. Je důležité poznamenat, že rychlost testování je přímo závislá na velikosti testovaného webu (velikost výstupní množiny stránek z webového crawleru).

Výsledky měření jsou zobrazeny v grafu 6.1, testování bylo prováděno na úlohách:

- Slovníkový útok,
- Cross-site scripting,
- SQL injekce,
- Path Traversal.

Výsledky měření ukazují, že rozdíl v prohlížečích Mozilla Firefox a Google Chrome je opravdu minimální. Tyto rozdíly jsou často způsobeny rychlostí jádra. Google chrome je založený na jádře Blink [22], zatímco Mozzila Firefox operuje s jádrem Gecko.



Obrázek 6.1: Porovnání rychlosti prohlížečů

Nehledě na různé druhy a výsledky testů porovnání prohlížečů je v práci použitý internetový prohlížeč Google Chrome ve spolupráci se Seleniovým ovladačem.

6.4 Zhodnocení dosažených výsledků

Aplikaci pro testování jsem pouštěl se základním nastavením na testovacích aplikacích opakovaně. Přehled odhalených chyb je zobrazen v tabulce 6.3 pro aplikaci React a v tabulce 6.4 pro aplikaci PHP.

Adresa	**Útok**	**Odhaleno**
/?page=	Path Traversal	Ano
/?page=/view/list.php	XSS	Ano
/?page=/view/detail.php&id=1	SQL Injection	Ano
/?page=/view/login.php&url=/scripts/login.php	Form action hijacking	Ano
/?page=/view/detail.php&id=1	InlineJS	Ano
/?page=/view/login.php&url=/scripts/login.php	Brute force	Ano - v manuálním režimu

Tabulka 6.3: Odhalené zranitelnosti v PHP aplikaci

Adresa	**Útok**	**Odhaleno**
/list	XSS	Ano
/?page=/view/detail.php&id=1	SQL Injection	Ano
/?page=/view/detail.php&id=1	XSS	Ano
/?page=/view/login.php&url=/scripts/login.php	Brute force	Ano - v manuálním režimu

Tabulka 6.4: Odhalené zranitolnosti v React aplikaci

Testování bylo prováděno pouze na testovacích aplikacích, které jsem sám vytvořil. Bohužel, z legislativních důvodů naší země není možné pouštět beztrestně nástroje jako je ten, co jsem vytvořil na webových aplikacích jiných stran. Takové testování by mohlo být považováno jako hackerský útok a následně bych mohl být někým zažalován. Proto jsem aplikaci nezkoušel na jiných webových aplikacích.

Vytvoření testovacích aplikací na míru pro program, který má tyto aplikace testovat může nepatrně zkreslovat výsledky měření, jelikož jsem autorem testovacího softwaru, vím přesně, jakým způsobem pracuje.

Aplikace je schopná otestovat velkou část základních bezpečnostních útoků v určité formě scénáře. Hojná část slabých míst má však různé podoby, například injekce škodlivého SQL, XSS kódu může mít více podob. Může být například ve vyhledávacích formulářích, je zmocněna povolovat pouze určité formy útočných dotazů a tvarů, atd.

Na vědomí je důležité brát, že testování bezpečnosti webových aplikací předchází důkladná analýza aplikace a tu dle mého osobního názoru robot nedokáže udělat tak dobře jako člověk.

Své dosažené výsledky hodnotím kladně, avšak v praxi bych se pouze na tento nástroj nespoléhal a automatické otestování bych doplnil dalším automatickým či manuálním nástrojem.

Mezi výhody aplikace může patřit možnost rychlého automatického otestování na velkém množství částí webové aplikace. Nevýhodou může být testování jednotlivých parametrů, protože je aplikace schopná otestovat pouze parametry typu ?par=test a s parametry složenými (par/test/test2/) si neví v aktuální verzi prozatím rady.

7 Závěr

Cílem této práce bylo prostudovat problematiku bezpečnosti webových aplikací, následně vysvětlit principy základních bezpečnostních útoků. Druhá část práce byla zaměřena více prakticky, v podobě návrhu a implementace nástroje, který umožňuje testování webové aplikace proti vybraným druhům útokům a to na nejvíce místech aplikace. V neposlední řadě bylo výsledný nástroj potřeba otestovat a zhodnotit jeho klady a zápory.

Na základě poznatků z teoretické části byl vytvořen nástroj, který je schopný otestovat celkem 10 bezpečnostních problémů do jisté míry. Většině testům předchází nastavení, na kterém při testu velice záleží.

Z důvodu testování nástroje pro testování bezpečnosti webových aplikací jsem vytvořil dvě aplikace, na kterých je možné výsledný nástroj testovat a demonstrovat jeho funkčnost. Výsledky testů dokazují, že funkčnost nástroje je vysoká. Je však důležité brát v úvahu fakt, že nástroj nebyl otestován na jiných aplikacích z legislativních důvodů. Ohlasy odborné veřejnosti na tento nástroj jsou pozitivní a lze tedy prohlásit, že funguje dle očekávání.

V rámci diplomové práce byla vytvořená javascriptová knihovna s názvem local-web-crawler, která je v práci použita (viz kapitola 5.3.1), tato knihovna byla publikována na NPM[1] kde ji může odborná veřejnost stahovat a používat bez jakéhokoliv omezení. Tato knihovna se stala poměrně populární, k dnešnímu dni 13.3.2018 má již přes 2283 stažení, což už je považováno za relativně velké číslo číslo.

V rámci diplomové práce byly vytvořeny následující aplikace:

- vlastní testovací nástroj,
- testovací aplikace v PHP,
- testovací aplikace v React,
- rest API k testovacím aplikacím,
- knihovna local-web-crawler,
- webový server pro provoz React aplikací.

Všechny tyto programy jsou přílohou této práce.

[1] https://www.npmjs.com/package/local-web-crawler

Na závěr je dobré poznamenat, že testování bezpečnosti webové aplikace předchází důkladná analýza webu. Tu bohužel mnou vytvořený nástroj udělat nedokáže, a tak se na něj nelze stoprocentně spoléhat. Podrobné zhodnocení výsledků lze nalézt v předchozí kapitole 6.4.

Vývoj aplikace RedFace bude i nadále pokračovat. V první řadě bude vývoj pokračovat tvorbou nových testů a následně autentifikací a autorizací celé aplikaci, pravděpodobně za použití protokolu OAuth 2.0 [20].

Literatura

[1] Allen, L. – Heriyanto, T. – Ali, S. *Kali Linux – Assuring Security by Penetration Testing*. Community experience distilled. Packt Publishing, 2014. Dostupné z: https://books.google.cz/books?id=QcBGAwAAQBAJ. ISBN 9781849519496.

[2] Aloul, F. – Zahidi, S. – El-Hajj, W. Two factor authentication using mobile phones. 2009. ISSN 2161-5322.

[3] Beacham, J. Is your practice GDPR ready? *In Practice*. 2018, 40, 3, s. 124–125. ISSN 0263-841X.

[4] Bertolino, A. Software Testing Research: Achievements, Challenges, Dreams. In *2007 Future of Software Engineering*, Washington, DC, USA, 2007. IEEE Computer Society. Dostupné z: https://doi.org/10.1109/FOSE.2007.25. ISBN 0-7695-2829-5.

[5] Chromium – teams, W. *ChromeDriver - WebDriver for Chrome* [online]. [cit.2018-04-17]. Dostupné z: https://sites.google.com/a/chromium.org/chromedriver/.

[6] community, S. U. *Getting up and running with Semantic UI* [online]. [cit.2018-04-17]. Dostupné z: https://semantic-ui.com/introduction/getting-started.html.

[7] Čížek, J. *Na stránkách Okamurovy strany se zjevil Hitler. Může za to vtípek s XSS* [online]. [cit.2018-04-17]. Dostupné z: https://www.zive.cz/bleskovky/na-strankach-okamurovy-strany-se-zjevil-hitler-muze-za-to-vtipek-s-xss/sc-4-a-189615/default.aspx.

[8] Český statistický úřad. *Využívání informačních a komunikačních technologií v domácnostech a mezi jednotlivci za období 2017* [online]. [cit.2018-04-17]. Dostupné z: https://www.czso.cz/documents/10180/46014700/06200417.pdf/a0bd4497-d2b6-450b-95f0-2f70c50786d5?version=1.1.

[9] Facebook Inc. *Hello World - React* [online]. [cit.2018-04-17]. Dostupné z: https://reactjs.org/docs/hello-world.html.

[10] Holmes, A. – Kellogg, M. Automating functional tests using Selenium. In *AGILE 2006 (AGILE'06)*, s. 6 pp.–275, 2006.

[11] Khandelwal, S. *Russia's Largest Portal HACKED; Nearly 100 Million Plaintext Passwords Leaked* [online]. [cit.2018-04-17]. Dostupné z: https://thehackernews.com/2016/09/russias-largest-portal-hacked-nearly.html.

[12] Lakshmiraghavan, B. Two-Factor Authentication. 2013, s. 319–343.

[13] Lyon, G. F. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure, 2009. ISBN 0979958717, 9780979958717.

[14] Mahajan, A. *Burp Suite Essentials.* Packt Publishing - ebooks Account, 2014. ISBN 1783550112.

[15] Myerson, J. M. Identifying enterprise network vulnerabilities. *International Journal of Network Management.* 12, 3, s. 135–144. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.433.

[16] Najork, A. H. Mercator: A scalable, extensible Web crawler. 1999. ISSN 1573-1413.

[17] Node.js Foundation. *Documentation* [online]. [cit.2018-04-17]. Dostupné z: https://nodejs.org/en/docs/.

[18] OWASP. *OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risk* [online]. [cit.2018-04-17]. Dostupné z: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.

[19] OWASP Foundation. *OWASP* [online]. [cit.2018-04-17]. Dostupné z: https://www.owasp.org.

[20] Parecki, A. *OAuth 2.0 Simplified.* lulu.com, 2017. ISBN 1387130102.

[21] Raphael Hertzog, . J. O. *Kali Linux Revealed: Mastering the Penetration Testing Distribution.* Offsec Press, 2017. ISBN 0997615605.

[22] Saloni ManhasEmail, S. T. A Comparative Analysis of Various Vulnerabilities Occur in Google Chrome. 2017. ISSN 978-981-10-5687-1.

[23] Socket.io community. *Documentation* [online]. [cit.2018-04-17]. Dostupné z: https://socket.io/docs/.

[24] The Webpack Organization. *Concepts* [online]. [cit.2018-04-17]. Dostupné z: https://webpack.js.org/concepts/.

[25] Warren, T. *Facebook ignored security bug, researcher used it to post details on Zuckerberg's wall* [online]. [cit.2018-04-17]. Dostupné z: https://www.theverge.com/2013/8/18/4633046/facebook-security-bug-let-anyone-post-on-walls.

[26] Warren, T. *LinkedIn ignored security flaw from researcher who hacked Zuckerberg's Facebook wall* [online]. [cit.2018-04-17]. Dostupné z: https://www.theverge.com/2017/12/15/16776176/linkedin-security-flaw-security-khalil-shreateh.

Příloha A: CD

Součástí práce jsou níže uvedené přílohy, které jsou umístěny na přiloženém CD.

- Elektronická verze tohoto dokumentu a jeho zdrojové soubory (adresář **doc**).
- Elektronická verze uživatelské příručky (soubor **RedFace - manual.pdf**).
- Zdrojové soubory testovacích aplikací (adresář **test app**).
- Zdrojové soubory aplikace RedFace (adresář **red face**).
- Poster (adresář **Poster**).

Příloha B: Uživatelská příručka

Uživatelská příručka

RedFace

Západočeská univerzita
Fakulta aplikovaných věd
Jakub Löffelmann

Obsah

1. Úvod

Tento uživatelský manuál je součástí diplomové práce, jejímž autorem je Jakub Löffelmann. Dokument popisuje, jakým způsobem nainstalovat a používat aplikaci RedFace.

1.1. RedFace

RedFace je aplikace na testování webových aplikací. Uživatelské prostředí je implementováno pomocí knihovny React a backendové jádro je napsán v prostředí NodeJS.

2. Konfigurace programu

Aplikace se skládá z frontendové a backendové části. Tyto dvě části budou popsány v následující části dokumentu.

Jak bylo řečeno výše, aplikace se skládá ze dvou částí, které jsou separované. Složka backend obsahuje zdrojové soubory pro backend aplikaci, frontend analogicky pro frontend aplikaci.

2.2. Minimální požadavky

2.2.1. Frontend

Operační systém Windows, Linux, nebo jiný.

Podporované prohlížeče jsou:

- Nejnovější 2 verze FF, Chrome, Safari Mac
- IE 11+
- Android 4.4+, Chrome pro Android 44+
- iOS Safari 7+
- Microsoft Edge 12+

2.2.2. Backend

Pro běh backendové služby je potřeba program NodeJS, tento program stahuje balíčky definované v aplikaci (některé balíčky mají specifické systémové požadavky), všechno však splňuje požadavky kladené na dnešní standardní aplikace (OS: Windows, Linux).
Dalším požadavkem je MySQL databáze (podporované jsou poslední 2 nejnovější verze).

2.3. Instalace

V následující sekci (a ve všech dalších) je předpokládáno, že NodeJS je správně nainstalovaný v systému a systémové cesty jsou nastaveny tak, aby jednotlivé příkazy šly použít v příkazové řádce/terminálu. Jednotlivé kroky je potřeba vykonat postupně po sobě tak, jak je uvedeno v této dokumentaci.

1. Frontend - **npm install** ve zdrojové složce Frontend projektu a následně **npm run-scrip build** pro vytvoření build adresáře. Přeložený (přesněji minifikovaný a sloučený) výstupní soubory budou uloženy ve složce dist. Tato složka obsahuje všechny potřebné soubory pro běh (index.hmt, javascriptové a další soubory).
2. Backend **npm install** ve zdrojové složce Backend projektu.

3. Spuštění

3.0.1. Frontend

Provádí se přesunutím obsahu složky dist do složky webového souboru v adresáři **backend/scripts/client/build**.

3.0.1.1. Vývojové spuštění

V případě vývoje, je možné spustit projekt ve vývojovém prostředí příkazem **npm start**.

Webová aplikace pak bude dostupná na adrese http://localhost:3000 . Při úpravě zdrojových souborů se sama obnoví.

3.0.2. Backend

Před samotným spuštěním je potřeba nastavit připojení do MySQL databáze (soubor **scripts/utils/database.js**) a následně vygenerovat tabulku v databázi přiloženým SQL skriptem **script.sql**.

Samotné spuštění aplikace se provede příkazem **node api.js**.

Backendová aplikace pro svůj chod využívá prohlížeč Chrome, je tedy třeba ho mít nainstalovaný.

V případě operačního systému Windows může být Google Chrome nainstalovaný standardně pomocí instalačního balíčku. Na strojích s operačním systémem Linux může Google Chrome nainstalován přes terminál pomocí programu **WGET**.

4. Popis ovládání programu

Tato část popisuje jednotlivé sekce programu a jejich ovládání.

4.1. Dashboard

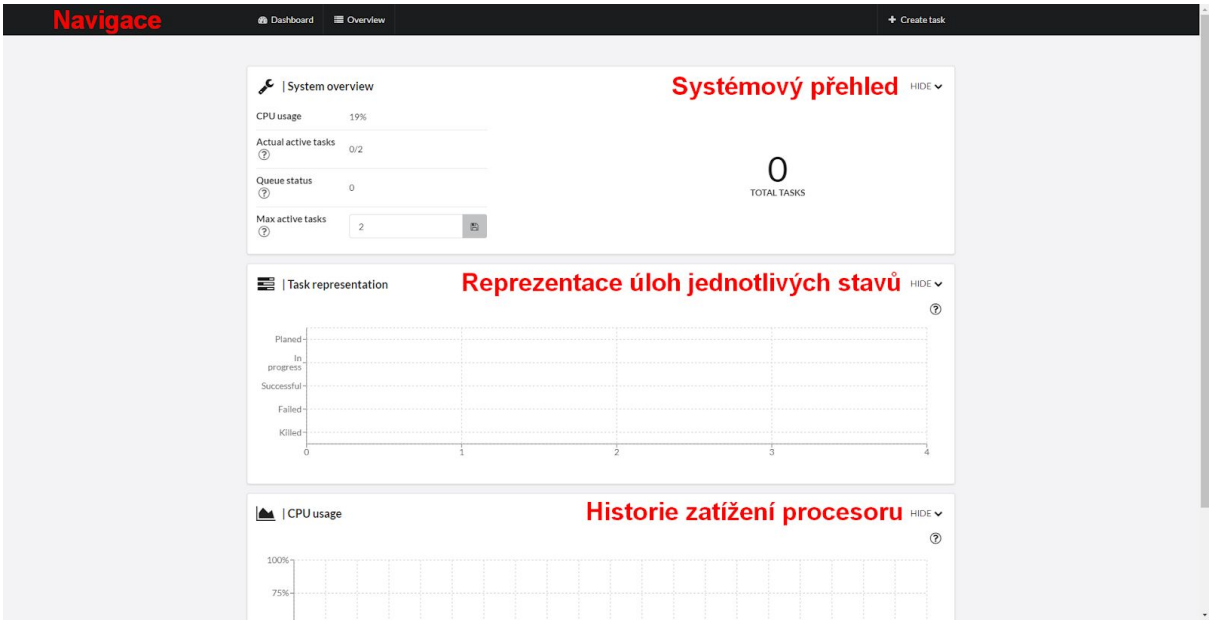
Dashboard (Obrázek 1: Sekce Dasboard) slouží jako přehled o celkovém stavu programu.

Navigace slouží k přechodu mezi jednotlivými sekcemi.

V sekci Systémový přehled jsou zobrazeny hodnoty testovacího jádra:

- Aktuální zatížení procesoru serveru.
- Aktuální počet aktivních úloh (kolik jich právě běží a kolik jich může běžet maximálně.
- Počet úloh ve frontě čekajících na spuštění.
- Možnost změny počtu aktivních úloh (kolik jich může být spuštěno vzájemně).

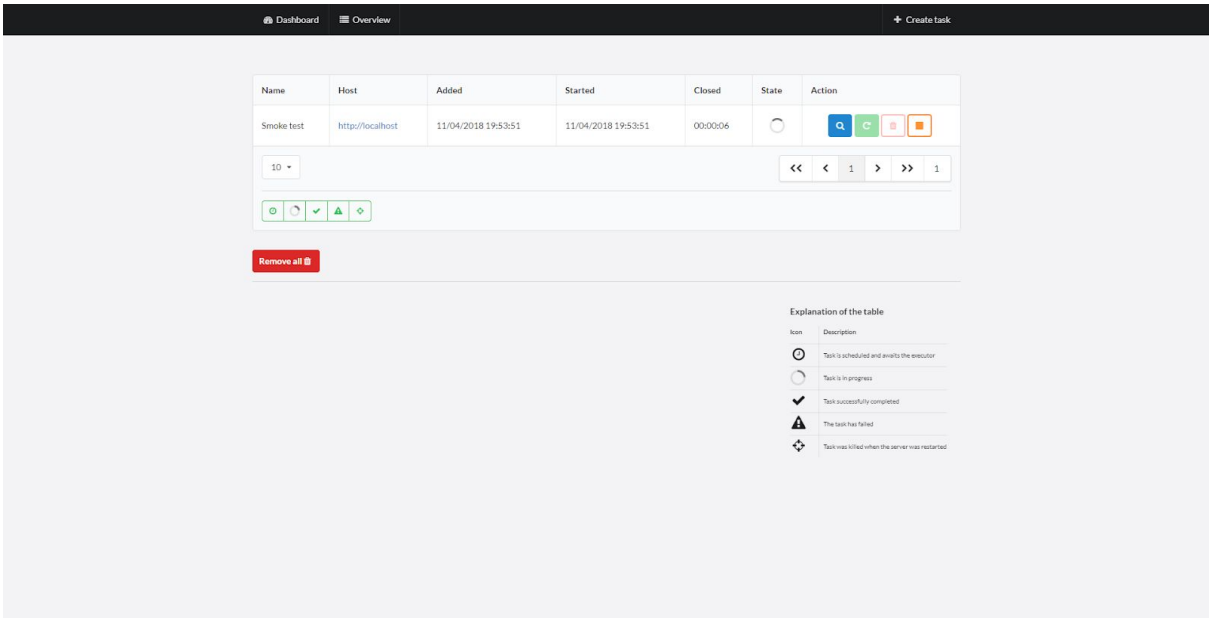
Graf Reprezentace úloh jednotlivých stavů zobrazuje podíl zastoupení stavu úloh všech úloh, které jsou zaznamenané v databázi. Graf Historie zatížení procesoru zobrazuje procentuální zatížení procesoru serveru.



Obrázek 1: Sekce Dasboard

4.2. Overview

Sekce overview (Obrázek 2: Sekce Overview) slouží jako celkový přehled o stavu všech úloh, v tabulce jsou zobrazeny všechny úlohy ve všech stavech.



Obrázek 2: Sekce Overview

V tabulce lze listovat pomocí stránek ve spodní části tabulky, úlohy lze filtrovat dle stavu ve kterých se nachází, přehled filtrů je zobrazen v následující tabulce

Filtr	Význam
	Úlohy čekající na spuštění ve frontě
	Právě probíhající úlohy
	Dokončené úlohy
	Dokončené úlohy s chybou procesu
	Přerušené úlohy serverem

Tabulka 1: Filtry tabulky přehled

Tlačítkem “Remove all” se odstraní všechny úlohy.

4.2.1. Akční tlačítka

Akční tlačítka v tabulce přehledu jsou součástí každé úlohy. Každému tlačítku jsou přiřazeny příslušné akce. Jednotlivá akční tlačítka jsou popsána v tabulce Tabulka 2: Akční tlačítka v přehledu úloh.

Tlačítko	Akce
	Otevře detail úlohy
	Zahájí stejnou úlohu znovu
	Smaže úlohu
	Přeruší úlohu

Tabulka 2: Akční tlačítka v přehledu úloh

4.3. Create task

Sekce pro vytvoření úlohy slouží k vytvoření nové testovací úlohy, v záhlaví této stránky lze přepínat mezi jednotlivým nastavením úloh. Každou úlohy je nutné nejprve aktivovat přepínačem “Enable”. V případě, že je úloha povolená, lze nastavit. V zápatí stránky je globální nastavení úlohy.

Pole **URLs** slouží k zadání adres testovacích serverů ve tvaru http://server.com, či s protokolem https. Do vstupního formuláře lze zadat i více adres, oddělené novým řádkem, například v následujícím tvaru:

http://localhost:8080 **http://localhost:8181**

Crawler max deep udává maximální hloubku crawleru, v případě že je pole zašedlé a vypnuté k editaci, není crawler potřeba pro testování a jeho nastavení je programem ignorováno.

Task name je jméno úlohy, které je zobrazeno v celkovém přehledu a detailu úlohy.

Nastavení jednotlivých úloh bude popsáno v následujících kapitolách.

4.3.1. Úloha Brute force

Login form XPath expression je validní XPath výraz, který lokalizuje přihlašovací formulář, který obsahuje vstup pro přihlašovací jméno (**Input name XPath expression** - XPath) a pro přihlašovací heslo (**Input password XPath expression** - XPath). V případě že je aktivní **Automatically** jsou výše zmíněné atributy vyplněny automaticky, opačné jsou k vyplnění poskytnuty uživateli.

Vyplnění testovacích hesel probíhá pomocí polí **Login names** a **Passwords** (oddělené novým řádkem). V případě že je aktivní **Automatically** jsou přihlašovací jména a hesla brána automaticky ze souboru, který je uložen na serveru v souboru **backend\scripts\task_settings\defaulbruteforce**.

V souboru se nacházejí přihlašovací jména a hesla na každé nové řádce oddělené prázdnou řádkou:

login1
login2
login3

psw1
psw2
psw3

URL location určuje adresu přihlašovacího formuláře, například **/login**, v případě že je aktivní **Automatically find login form**, bude použit pro vyhledání formuláře webový crawler.

Parallel nodes určuje počet paralelně běžících procesů pro testování přihlášení.

Page percentage difference určuje procentuální rozdíl přihlášené a nepřihlášené stránky, například, bude-li se přihlášená stránka od nepřihlášené lišit o 50% a vstupní hodnota Page percentage difference bude 60%, bude přihlášení považováno jako úspěšné.

4.3.2. Úloha XSS

Útok XSS je prováděn pomocí kontroly přitomnosti alert oken. V případě že je zobrazeno alert okno, je detekován XSS útok.

Input XSS attack (is checked by alert) udává útočící skripty XSS, v případě že jich je zadáno víc, je každý na novým řádku. Use default znamená, že je použit původní XSS kód pro útok, v případě že Use default není aktivní, může uživatel vložit vlastní XSS.

V případě že je aktivní **Test url params** bude testování prováděno na parametrech url, parametrem se rozumí objekt za otazníkem (?parametr=xss). Podobně u **Test page form**, které testuje vstupní formuláře na stránkách.

Všechny testovací vstupy jsou hledány pomocí webového crawleru.

4.3.3. Úloha Other

- **Form action hijacking** povoluje spuštění testu Form action hijacking.
- **Test inline JS** povoluje spuštění testu Test inline JS.
- **Test http and https** povoluje spuštění testu Test http and https.
- **Cross-Origin request** povoluje spuštění testu Cross-Origin request.
- **Test GIT config** povoluje spuštění testu Test GIT config, tento test vyžaduje zadání procentuální rozdílnosti chybové stránky a stránky zobrazené konfigurace programu (obdobně jako Page percentage difference v kapitole 4.3.1. Úloha Brute force). Soubory, které aplikace testuje lze upravit v konfiguračním souboru: **backend\scripts\task_settings\configuration\git_config**.
- **Port scanner** povoluje spuštění testu Port scanner, vstupem je interval skenu portů.

4.3.4. Úloha SQL

Input XSS attack (is checked by alert) určuje vstupní Sql dotazy, jednotlivé dotazy jsou na více řádkách. Každý z těchto dotazů je přímo vložen za původní parametr. V případě že je **Use default** aktivní, program používá základní nastavení. **Test url params** určuje, že se budou testovat pouze url parametry za otazníkem (?parametr=sql). Číslo udává procentuální rozdíl zobrazené stránky od chybové.

4.3.5. Úloha PTA

Configuration je dvojice proměnných, které určují jednu testovací množinu. Na první řádku je soubor, který se snaží program zobrazit a na druhém řádku je řetězec, který pomocí funkce **contains** vyhledává v zobrazeném souboru.

Příklad:

C:\Windows\System32\drivers\etc\hosts **# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.**

Program se snaží zobrazit soubor **hosts** a následně v zobrazené stránce hledá řetězec **# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.** Testovacích dvojic může být víc, dalším řádkem začínají nové dvojice.

Use default určuje, zda-li mají být použity základní testovací řetězce, nebo vstup od uživatele.

5. Testovací režim aplikace

Aplikaci lze pustit v testovacím režimu úpravou konfiguračního souboru (**backend\scripts\utils\web-driver.js)**, který řídí Selenium.

Pro zobrazení Google Chrome prohlížeče, ovládaného Seleniem je potřeba změnit proměnnou **CHROME_OPTIONS** tak, aby argumenty (**args**) byly:

"--test-type", "--start-maximized", "--profile-directory='red-face'"

Vypnutí testovacího režimu se provede změnou na defaultní parametry:

'--headless', '--test-type', 'disable-web-security', '--log-level=3', '--silent', **'--no-sandbox', '--disable-gpu', '--log-path=NUL'**