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

Diplomová práce

Systém pro řízení přístupu do kolejní sítě ZČU

Plzeň 2016 Jan Smitka

Prohlášení

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

V Plzni dne 20. června 2016

Jan Smitka

Poděkování

Na tomto místě bych rád poděkoval Ing. Michalu Petrovičovi za odborné vedení práce a cenné rady.

Dále bych rád poděkoval pracovníkům Centra informatizace a výpočetní techniky za konzultace a cenné podněty, zejména pak Ing. Jaromíru Staňkovi, Ing. Martinu Šimkovi, Ph.D., Ing. Michalu Švambergovi a Ing. Petru Žákovi.

V neposlední řadě bych pak chtěl poděkovat Ing. Vladimírovi Smitkovi, Mgr. Barboře Štětinové a Bc. Veronice Švecové za četnou pomoc při korekturách textu, rodičům za poskytnutí zázemí a podpory po celou dobu studia a přítelkyni za trpělivost.

Abstract

The goal of this thesis is to implement a new system for access control in dormitory network at University of West Bohemia. The theoretical part of the thesis deals with systems for network access control, compares selected access control solutions. The thesis also summarises systems and processes for incident resolution at the university. The analytical part evaluates the state of current access control solution, including the software of network services.

On the basis of these parts a new web portal for user registration was designed and implemented. The portal automatically queries connection information about the user acessing the portal. The software for network services that enables service configuration using dedicated scripts was selected. The scripts are controlled by a message queue stored in the database. The registration portal provides an API which is intended for integration into the processes for incident resolution. The API was verified and tested on a real-time web interface for a network probe.

Abstrakt

Tato diplomová práce se zabývá realizací nového systému pro řízení přístupu do kolejní počítačové sítě Západočeské Univerzity v Plzni. Teoretická část práce se zabývá systémy pro řízení přístupu v počítačových sítích a srovnání vybraných existujících řešení. Dále popisuje systémy a procesy pro řešení bezpečnostních incidentů na univerzitě. Analytická část práce pak zhodnocuje stávající systém pro řízení přístupu a jeho nedostatky, včetně softwarového vybavení síťových služeb.

Na základě těchto poznatků byl navržen a implementován nový webový registrační portál, který automaticky získává informace o připojení uživatele, který k němu přistupuje. Bylo vybráno nové softwarové vybavení síťových služeb, které jsou konfigurovány pomocí jednoúčelových konfiguračních skriptů. Konfiguračním skriptům je práce zadávána pomocí fronty zpráv uložené v databázi. Portál obsahuje rozhraní pro integraci do procesů řešení incidentů, jehož funkce byla ověřena na webovém rozhraní pro síťovou sondu. Toto rozhraní zobrazuje detekovaná data v reálném čase.

Obsah

1 Úvod

Na všech kolejích Západočeské univerzity v Plzni se ubytovaní připojují pomocí kabelového připojení, které odpovídá standardu Ethernet. V síti je potřeba řídit přidělování adres koncovým zařízením a evidovat, komu jaké zařízení patří. Při řešení bezpečnostních incidentů musí mít správci sítě možnost vlastníka koncového zařízení kontaktovat, případně mu v případě porušení definovaných pravidel sítě odepřít připojení.

Za tímto účelem v minulosti vznikl registrační portál Knet [38]. Ten umožňuje ubytovaným snadno registrovat své zařízení a správcům pak jednotlivé registrace zařízení procházet, mazat či blokovat. Tento systém však pochází z roku 2007 a v současnosti již není vyhovující – je postavený na již nepodporovaných technologiích a obsahuje celou řadu bezpečnostních chyb. Systém také není nijak integrován s dalšími informačními systémy na Západočeské univerzitě v Plzni.

Cílem této diplomové práce je seznámit se s problematikou řízení přístupu v počítačových sítích, analyzovat současný stav registračního systému Knet a seznámit se se systémy a procesy, které se používají při řešení bezpečnostních incidentů v síti WEBnet. Na základě těchto poznatků pak navrhnout řešení, které systém zmodernizuje a umožní jeho integraci se systémy a procesy řešení incidentů.

Následující kapitola popisuje základní protokoly a služby, které jsou v počítačových sítích používány v souvislosti s řízením přístupu. 3. kapitola popisuje principy pro NAC (Network Access Control), tedy řízení přístupu v počítačových sítích, a vybrané protokoly.

4. kapitola prezentuje analýzu několika vybraných dostupných řešení pro NAC. Vybrána byla zejména softwarová řešení, neboť je lze nasadit bez změn v síťové infrastruktuře. 5. kapitola se zabývá analýzou stávající kolejní sítě a systému, který aktuálně zajišťuje řízení přístupu. Tento systém je analyzován z hlediska jeho architektury, možností konfigurace či dalšího rozšíření a bezpečnosti. 6. kapitola popisuje procesy a systémy, které jsou používány při řešení incidentů v univerzitní síti WEBnet.

Kapitola 7 popisuje návrh a analýzu nového řešení, kapitola 8 návrh databáze registračního portálu. Kapitoly 9 a 10 popisují detaily implementace navrženého systému, který zajišťuje řízení přístupu v kolejní síti. Kapitola 11 se zabývá implementací webového rozhraní pro síťovou sondu, která byla integrována s registračním portálem. Kapitola 12 pak popisuje možnosti dalšího rozšíření systému.

2 Služby počítačových sítí

Pro usnadnění konfigurace koncových zařízení a zabezpečení přístupu k síti se používají některé síťové služby a s nimi související protokoly. V této kapitole budou popsány základní služby, které se k tomuto účelu používají v sítích LAN (Local Area Network). Všechny popsané protokoly pracují na aplikační vrstvě TCP/IP modelu.

2.1 Protokol DHCP

Protokol DHCP (Dynamic Host Configuration Protocol) se používá pro automatickou konfiguraci IPv4 (Internet Protocol verze 4) adresy koncových stanic. Ke své funkci vyžaduje správně nastavený DHCP server. [14] Tento protokol nahrazuje starší protokol BOOTP (Bootstrap Protocol). Pro komunikaci mezi klientem a serverem se používá UDP (User Datagram Protocol) port 67 (server) a 68 (klient).

Přiřazení IPv4 adresy je dočasné, server ji klientům pouze zapůjčuje (lease). Zapůjčení probíhá vždy na předem určenou dobu (lease time), pokud chce klient adresu používat dál, musí pravidelně žádat o prodloužení zapůjčení (lease renewal).

Protokol používá jednotný formát zpráv. Formát zprávy je znázorněn na obrázku 2.1. Obsahuje následující pole:

op: typ zprávy. hodnota 1 se používá pro požadavek, hodnota 2 pro odpověď DHCP serveru. Toto pole je uvedeno pro kompatibilitu s protokolem BOOTP, typ zprávy je pak dále určen v poli options.

<table>
<tr><td>op (1)</td><td>htype (1)</td><td>hlen (1)</td><td>hops (1)</td></tr>
<tr><td colspan="4">xid (4)</td></tr>
<tr><td colspan="2">secs (2)</td><td colspan="2">flags (2)</td></tr>
<tr><td colspan="4">ciaddr (4)</td></tr>
<tr><td colspan="4">yiaddr (4)</td></tr>
<tr><td colspan="4">siaddr (4)</td></tr>
<tr><td colspan="4">giaddr (4)</td></tr>
<tr><td colspan="4">chaddr (16)</td></tr>
<tr><td colspan="4">sname (64)</td></tr>
<tr><td colspan="4">file (128)</td></tr>
<tr><td colspan="4">options (312)</td></tr>
</table>

Obrázek 2.1: Formát DHCP zprávy. Čísla v závorkách určují délku pole v bajtech.

htype a hlen: typ přenosového protokolu a s ním související délka fyzické adresy v bajtech. Číslo protokolu je přiřazeno dle registru IANA (Internet Assigned Numbers Authority). Pro Ethernet bude délka adresy 6, neboť jím používaná MAC (Media Access Control) adresa má délku 6 bajtů.

hops: klient nastavuje na hodnotu 0, DHCP Relay Agent (bude vysvětleno dále) tuto hodnotu pak při každém přeposlání zprávy zvýší o jedna. Pokud hodnota dosáhne nakonfigurovaného limitu, zprávu zahodí.

xid: ID transakce. Klient vyplňuje náhodnou hodnotu, pomocí které se určují zprávy, které spolu souvisejí.

secs: klient uvádí počet sekund, které uplynuly od začátku procesu získání či obnovy IP adresy.

flags: příznaky zprávy.

ciaddr: adresa klienta. Klient uvádí v případě, že provádí prodloužení propůjčení adresy.

yiaddr: „Your IP Address“, v tomto poli server předává klientovi přidělenou IP adresu.

siaddr: IP adresa serveru, který klient může použít pro zavedení operačního systému ze sítě, obraz operačního systému je umístěn na tomto serveru.

giaddr: adresa DHCP Relay Agenta.

chaddr: fyzická adresa klienta.

sname: doménový název serveru.

file: název souboru, který má klient použít pro zavedení operačního systému ze sítě.

options: pole dalších nastavení s variabilní délkou. Každému nastavení je přidělen identifikátor o velikosti 1 byte. V poli options je uveden identifikátor a data daného nastavení. Formát dat je specifický pro každé nastavení. Vybraná nastavení jsou uvedena v tabulce 2.1, kompletní popis pak v [3].

Typy zpráv, které DHCP používá, jsou uvedeny v tabulce 2.2. Postup přidělení IP adresy probíhá následovně:

1. Stanice po připojení do sítě vyšle broadcast zprávu typu *DHCPDISCOVER*, ve kterém žádá o přidělení IPv4 adresy. Broadcast je zpráva určená všem stanicím v dané síti. Jako cílovou MAC adresu v Ethernet rámci stanice uvede FF-FF-FF-FF-FF-FF.
2. Zprávu může přijmout jeden nebo více DHCP serverů. Každý zkontroluje, zda může klientovi přidělit IP adresu. Pokud ano, pošle nabídku zprávou *DHCPOFFER*. Pokud klientovi nemůže IP adresu přidělit

Kód nastavení	**Popis**
1	Maska podsítě jako 4bajtová IP adresa.
3	IP adresa výchozí brány.
6	IP adresa DNS serveru.
50	Klientem požadovaná IP adresa.
51	Délka propůjčení IP adresy (lease time). Udává se v sekundách.
53	Typ DHCP zprávy.
54	IP adresa DHCP serveru, který posílá odpověď.
58	Tzv. renewal time (T1). Určuje čas v sekundách, po kterém má klient požádat o prodloužení zapůjčení IP adresy. Klient žádá o IP adresu stejný server, který mu IP adresu poskytl (uvedený v option 51).
59	Tzv. rebinding time (T2). Určuje čas v sekundách, po kterém může klient požádat o prodloužení IP adresy či přidělení nové adresy nebo adresy z jiného serveru.

Tabulka 2.1: Volby v poli DHCP Options.

(například žádá z podsítě, kterou DHCP server nespravuje), může odpovědět zprávou *DHCPNAK*, nebo neodpovídat vůbec.

3. Klient si z přijatých *DHCPOFFER* vybere konfiguraci (typicky vybere první odpověď, která mu byla doručena) a zkontroluje, zda daná IP adresa již není v síti použita, může například provést dotaz protokolem ARP (Address Resolution Protocol). Pokud použita není, požádá o přidělení IP adresy zprávou *DHCPREQUEST*. Tuto zprávu klient posílá také pomocí broadcast, avšak v option 54 uvede adresu serveru, jehož nabídku vybral. Pokud klient zjistí, že je daná IP adresa již použita, posílá zprávu *DHCPDECLINE*.
4. Server přijme *DHCPREQUEST* a zkontroluje, zda je jeho adresa uvedena v option 54. Pokud ano, uloží si informace o propůjčené IP adrese a klientovi zašle *DHCPACK*, kterým mu rezervaci potvrdí. Pokud z nějakého důvodu nemůže IP adresu přidělit (například již byla použita jiným klientem), posílá *DHCPNAK*.

Server IP adresu přiděluje buď dynamicky z předem definovaného rozsahu

Kód a název zprávy	**Posílá**	**Popis**
1 DHCPDISCOVER	Klient	Požadavek o přidělení IP adresy.
2 DHCPOFFER	Server	Nabídka IP adresy.
3 DHCPREQUEST	Klient	Požadavek o přidělení IP adresy, která již byla klientovi dříve nabídnuta zprávou DHCPOFFER.
4 DHCPDECLINE	Klient	Klient zjistil, že přidělená IP adresa je již v síti použita a informuje server. Server musí danou IP adresu označit jako použitou.
5 DHCPACK	Server	Potvrzení přidělení IP adresy.
6 DHCPNAK	Server	Odmítnutí přidělení IP adresy.
7 DHCPRELEASE	Klient	Klient ukončuje svojí činnost a žádá o uvolnění IP adresy, aby mohla být případně použita jiným klientem.
8 DHCPINFORM	Klient	Již nakonfigurovaný klient žádá o parametry sítě.

Tabulka 2.2: Typy DHCP zpráv.

adres, nebo může mít pro danou fyzickou adresu nastavenou rezervaci pro konkrétní IP adresu.

Server nemusí být přístupný přímo v dané síti, ale místo něj může být přítomný tzv. DHCP Relay Agent. Ten se chová jako proxy server a všechny požadavky klientů v síti přeposílá na nakonfigurovaný DHCP server. V poli `giaddr` pak uvádí svojí IP adresu v síti, ve které DHCP zprávu přijal.

Díky tomu je možné mít jeden centrální DHCP server pro více podsítí. O tom, z jaké podsítě má DHCP server přidělit adresu, se rozhoduje podle adresy uvedené v poli `giaddr`. Komunikace mezi DHCP serverem a DHCP Relay Agent probíhá pomocí unicast.

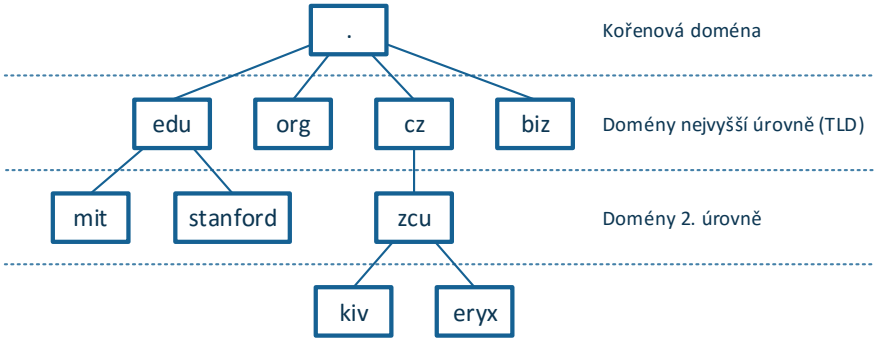
Nakonfigurovaný klient po uplynutí času T1 žádá o prodloužení IP adresy zprávou *DHCPREQUEST* pouze server, který mu adresu zapůjčil. Pokud server neposkytl čas T1, klient použije hodnotu 0.5× doba zapůjčení. Pokud nedostane odpověď, snaží se periodicky požadavek *DHCPREQUEST* opakovat. Mezi požadavky čeká vždy polovinu času, která mu zbývá do T2, nejméně však 60 sekund.

Pokud se mu nepodaří obnovit IP adresu v čase T2 (pokud nebyl uveden, použije se hodnota 0.875× doba zapůjčení), pošle požadavek *DHCP-REQUEST* pomocí broadcast všem serverům. I v tomto případě klient požadavek opakuje, a to vždy po uplynutí poloviny času, který mu zbývá do vypršení doby zapůjčení.

2.2 Protokol DNS

Protokol DNS (Domain Name System) se používá pro obousměrný překlad mezi doménovým jménem, což je pro člověka snadno zapamatovatelný textový řetězec, a IP adresou. Protokol používá UDP port 53. [22]

Prostor doménových jmen v systém DNS je rozdělen do stromové struktury. Každému uzlu stromu je přiřazeno doménové jméno. Kořenové doméně je přiřazeno prázdné doménové jméno, každému dalšímu uzlu pak řetězec o délce 1 až 63 znaků. Doménové jméno může obsahovat písmena, čísla a pomlčku, přičemž pomlčkou nesmí začínat ani končit. Příklad struktury je znázorněn na obrázku 2.2.



Obrázek 2.2: Příklad části stromu doménových jmen.

Doménová jména se zapisují oddělená tečkou od nejnižší úrovně, například „eryx.zcu.cz.“. Koncová tečka označující kořenovou domény je pro většinu DNS klientů nepovinná a doménu tedy lze zapsat pouze řetězcem „eryx.zcu.cz“. Každému doménovému jménu je přiřazen jeden nebo více doménových záznamů různých typů. Běžně používané typy záznamů jsou uvedeny v tabulce 2.3. Záznamy pro doménu se uchovávají v tzv. zónovém souboru.

Doménové jméno je spravováno jedním nebo více doménovými servery, ty jsou nazývány autoritativní servery. Z těchto serverů musí být jeden server hlavní, který uchovává vždy aktuální informace – master, někdy též

Typ záznamu	**Popis**
NS	Doménové jméno autoritativního serveru dané domény.
MX	Mail Exchange – doménové jméno e-mailového SMTP serveru.
A	IPv4 adresa uzlu.
AAAA	IPv6 adresa uzlu.
CNAME	Vytvoření aliasu pro jiné doménové jméno.
TXT	Záznam obsahující text.
PTR	Ukazatel na jiné doménové jméno. Používá se jako reverzní DNS záznam, pomocí kterého se provádí překlad IP adresy na doménové jméno.
SOA	Start Of Authority – záznam, který obsahuje název primárního DNS serveru domény, kontaktní informace na správce, verzi definice domény a časové intervaly, které slouží pro synchronizaci mezi master a slave servery: jak často má být zóna stahována, po jaké době může slave server pokus o stažení opakovat v případě chyby, po jakou dobu může slave server záznamy považovat za autoritativní a poskytovat je klientům. Kromě těchto hodnot také záznam obsahuje maximální dobu, po kterou si klientské aplikace mají uchovávat odpověď indikující neexistující doménu.

Tabulka 2.3: Typy DNS záznamů.

označovaný primary server. Od něj si ostatní servery spravující doménu pravidelně stahují informace o doménách. [12]

Servery, které spravují kořenovou doménu, se nazývají kořenové jmenné servery. Jejich seznam musí mít klient, který má překlad doménových jmen provádět, předdefinovaný.

Při překladu doménového jména „eryx.zcu.cz.“ na IPv4 adresu (A záznam) klient postupuje následovně:

1. Dotáže se některého z kořenových jmenných serverů na A záznam pro doménu „eryx.zcu.cz.“. Protože kořenové servery nemají pro tuto doménu záznam, vrátí klientovi NS záznamy pro servery spravující doménu „cz.“, spolu s adresami těchto serverů.

2. Klient se dotáže serverů domény „cz.“ na A záznam pro doménu „eryx.zcu.cz.“. Ani tyto servery neobsahují příslušný záznam, proto vrátí NS záznamy pro servery spravující doménu „zcu.cz.“ spolu s jejich adresami.
3. Klient se dotáže serverů domény „zcu.cz.“ na A záznam pro „eryx.zcu.-cz.“. Ty mu již vrátí příslušný záznam s IPv4 adresou.

Při obráceném překladu, tedy z IP adresy na doménové jméno, se klient dotazuje doménových serverů na doménové jméno „<ip>.in-addr.arpa.“, kde je IP zapsána obráceně v decimálním formátu. Například při zjišťování doménového jména adresy 147.228.53.17 by se klient dotazoval na jméno „17.53.228.147.in-addr.arpa.“. Dotazování probíhá stejným způsobem, je ale vyhledáván záznam typu PTR.

Koncové stanice typicky neprovádějí překlad DNS adres samy, avšak využívají nastaveného rekurzivního DNS serveru. Tento server se také nazývá cache-only DNS server, neboť si nalezené záznamy uchovává v paměti, aby je při příštím dotazu nemusel opětovně vyhledávat.

2.3 Zabezpečení a omezení přístupu: firewall

Dle informačního [15] je firewall síťový prvek, který se chová jako proxy zprostředkovávající přístup k určitému aplikačnímu serveru (například SMTP serveru), nebo který provádí filtrování paketů, případně kombinaci obojího.

Pokud je firewall v roli proxy serveru, tak implementuje určitou bezpečnou podmnožinu daného aplikačního protokolu, provádí rozšířené kontroly vstupních dat, běží v izolovaném prostředí, nebo kombinuje tyto metody dohromady.

Pokud firewall pracuje jako filtr paketů, tak analyzuje každý paket, který obdrží. Na základě inspekce paketu se pak rozhodne, zda paket předá dál v nezměněné podobě, zahodí jej, nebo zpracuje jiným způsobem. Toto rozhodování typicky probíhá na základě pravidel, které nadefinoval správce.

Podle informací, které jsou při filtrování použity, můžeme firewally rozdělit do následujících kategorií:

Paketový filtr: provádí rozhodování pouze na základě informací obsažených v paketu, a to ze síťové a transportní vrstvy. Nejčastěji jde o zdrojovou a cílovou adresu, zdrojový a cílový port a použitý protokol.

Stavový paketový filtr: uchovává si i stav jednotlivých spojení a používá jej i při rozhodování. Umožňuje například vytvořit pravidlo, které povolí průchod příchozích paketů, jestliže jsou odeslány v rámci spojení, které již bylo dříve navázáno některým ze strojů v chráněné podsíti.

Aplikační filtr: při rozhodování bere v potaz nejen stav a informace síťové a transportní úrovně, ale analyzuje i aplikační data. Typickým příkladem aplikačního firewallu je pravidlo, které analyzuje řídící příkazy FTP (File Transfer Protocol) spojení a povolí navázání pasivního spojení klienta na portu, na kterém se s ním dohodl FTP server. Kombinuje tak filtrování paketů a aplikační proxy.

Funkce firewallu může zajišťovat buď některý ze specializovaných prvků sítě, nebo jej může zajišťovat například linuxový server či směrovač.

2.4 Protokol SNMP

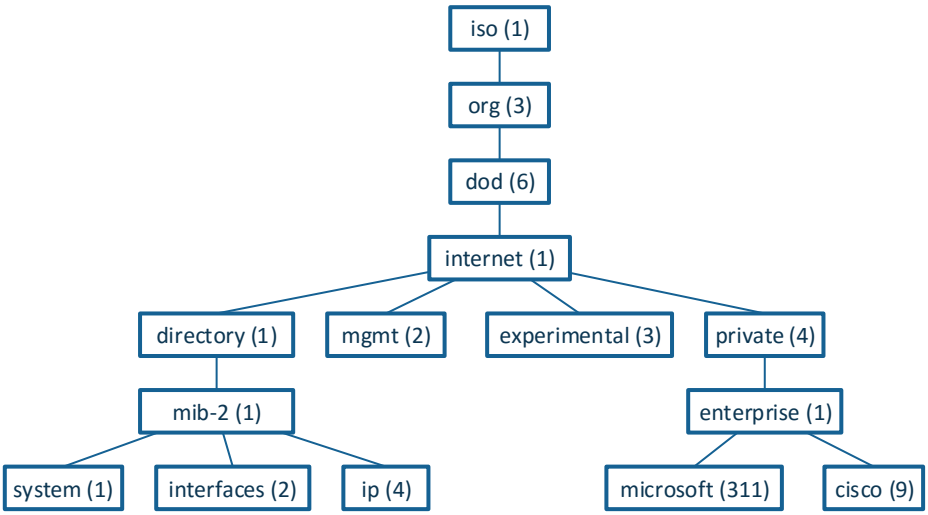
Protokol SNMP (Simple Network Management Protocol) slouží k zjišťování informací ze síťových prvků a koncových stanic (managed device – spravované zařízení) a k úpravě konfigurace těchto zařízení. Vychází z protokolu SGMP (Simple Gateway Monitoring Protocol). Používá UDP port 161 a 162.

Protokol prošel ve vývoji několika verzemi, které se od sebe liší podporovanými zprávami a některými vlastnostmi. V současnosti aktuální verze je SNMPv3. Jeho základní koncepty jsou definované v [17], formát zpráv pak v [27] a [8]. V praxi se také stále používá verze SNMPv2c, neboť mnoho koncových zařízení novější verzi stále nepodporuje z důvodu náročnější implementace.

SNMPv3 podporuje šifrovaný přenos, ověřování uživatele a řízení přístupu. V dřívějších verzích (SNMPv1, SNMPv2, SNMPv2c) probíhá ověřování uživatele pouze zadáním jména komunity, které se navíc přenáší nešifrovaně. V protokolu SNMPv3 se uživatel ověřuje uživatelským jménem a heslem, kterým podepisuje své zprávy algoritmem HMAC (Hash-based Message Authentication Code)[1]. Jako hashovací funkce algoritmu se používá buď MD5 (Hash-based Message Authentication Code), nebo SHA (Secure Hash Algorithm). Zpráva je dále šifrována algoritmem DES (Hash-based Message Authentication Code) nebo novějším AES (Advanced Encryption Standard) a uživatelovým tajným klíčem. [20]

Hodnoty, které mohou být pomocí SNMP čteny či zapisovány, jsou uspořádány do stromové struktury. Každá hodnota je přístupná pomocí jejího OID (Object Identifier) – identifikátor, který popisuje cestu k hodnotě ve stromu. Jedná se o sérii čísel, která jsou oddělena tečkami. Příklad stromu je znázorněn na obrázku 2.3.

[1] Algoritmus, jehož vstupem je zpráva a tajný klíč. Výstupem je kontrolní součet vypočítaný danou hashovací funkcí.



Obrázek 2.3: Strom OID hodnot.

Cestu k větvi stromu pojmenovanou „ip“ lze popsat identifikátorem 1.3.6.1.1.-1.4. Jednotlivé objekty jsou popsány jazykem SMI (Structure of Management Information) [21]. Jejich popis je uložen v tabulkách MIB (Management Information Base), ve kterých je i jednotlivým objektům přiřazen unikátní textový řetězec pro snazší zápis či čtení.

Protokol SNMP rozlišuje 3 základní role:

Spravované zařízení – zařízení, na kterém běží SNMP agent a jehož informace jsou čteny, případně jehož konfigurace je zapisována.

Agent – softwarové vybavení, které reaguje na požadavky protokolu SNMP a poskytuje data.

Network Management Station – zařízení, které čte data od agenta. Je schopno přijímat asynchronní notifikace od agenta – tzv. traps.

Protokol SNMPv3 definuje 8 typů zpráv. Jejich přehled je uveden v tabulce 2.4.

2.5 Adresářová služba LDAP

LDAP (Lightweight Directory Access Protocol) je protokol, který slouží pro přístup k adresářovému serveru. V adresářovém serveru jsou záznamy uspořádány do stromové struktury. Záznamem může být informace o uživatelích, uživatelských skupinách, koncových stanicích a dalších síťových zařízeních, službách v síti a další. Aktuální verze protokolu 3 je popsána v [30]. Protokol používá TCP (Transmission Control Protocol) a UDP porty 389 pro nezabezpečené a 636 pro zabezpečené spojení. [7]

Název zprávy	**Popis**
GetRequest	Požadavek na získání hodnoty s daným OID.
GetNextRequest	Požadavek na získání další hodnoty ve stromu.
GetBulkRequest	Požadavek na získání více hodnot. Přidáno v SNMPv2.
Response	Odpověď na některý z požadavků.
SetRequest	Požadavek na nastavení hodnoty.
Trap	Asynchronní notifikace, kterou agent posílá NMS v případě, že dojde k určité události. Příkladem takové události může být změna stavu portu na přepínači vyvolaná připojením klienta.
InformRequest	Podobné jako Trap, avšak tato zpráva vyžaduje potvrzení, že byla přijata. Protože SNMP používá UDP, není garantováno, že zpráva Trap bude úspěšně přijata a agent nebyl o ztracení paketu informován. Zpráva byla přidána v SNMPv2.
Report	Zpráva indikující interní chybu při zpracování požadavku. Přidána v SNMPv2.

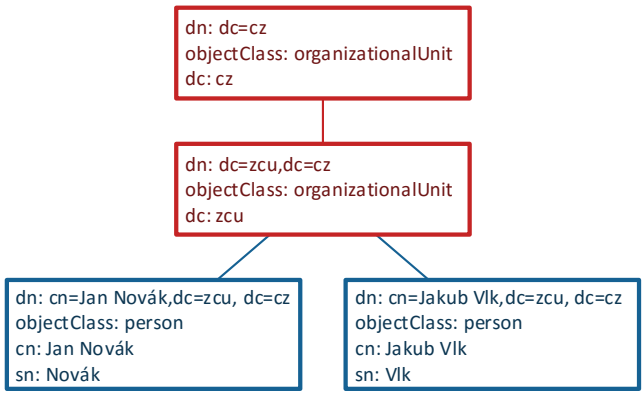
Tabulka 2.4: Typy zpráv v SNMPv3.

Každý záznam na adresářovém serveru se skládá ze sady záznamů. Každý záznam má název a jednu nebo více hodnot. Záznamu je přiřazen unikátní identifikátor – DN (Distinguished Name). Ten se vznikne spojením z RDN (Relative Distinguished Name) a DN rodičovského záznamu.

RDN vznikne spojením jednoho či více vybraných atributů, které jednoznačně identifikují daný záznam. Atributy jsou zapsány ve tvaru „<název atributu>=<hodnota>“. Pokud je atributů více, oddělují se znakem +. RDN a DN rodičovského záznamu se odděluje čárkou. Příklad stromu je uveden na obrázku 2.4.

U každého záznamu je v atributu s názvem objectClass určen jeho typ, tj. třída objektu. Třída definuje atributy, které mohou být záznamu přiřazeny, a které z těchto atributů jsou povinné. Definice atributů (včetně jejich datových typů) a tříd objektů jsou definovány v tzv. LDAP schema. Třídy objektů mohou být uspořádány do hierarchie (dědičnost) a mohou být některého z následujících typů:

STRUCTURAL – třída může být použita pro vytvoření nového objektu.



Obrázek 2.4: Příklad stromu v adresářovém serveru. Záznamy jsou uváděny v textovém formátu LDIF.

AUXILIARY – pomocná třída, která definuje doplňující atributy.
ABSTRACT – abstraktní koncept.

Při použití dědičnosti musí být nadřazená třída stejného typu. Výjimku tvoří abstraktní třída „top“, která může být použita k odvození libovolného typu a ukončuje hierarchii objektů. [7]

Atribut záznamu může mít více hodnot. Tato vlastnost platí i pro atribut objectClass, záznamu však musí být vždy přiřazena právě jedna třída typu STRUCTURAL. Tříd typu AUXILIARY a ABSTRACT může být záznamu přiřazeno libovolné množství. RDN je udáno při vytváření záznamu.

Protokol definuje operace popsané v tabulce 2.5.

Jednotlivé operace (s výjimkou operací *Bind*, *Unbind* a rozšiřující operace *StartTLS*) jsou vykonávány asynchronně – klient po zadání operace nemusí čekat na její dokončení a může zadat další. Server výsledky operací může zaslat v libovolném pořadí. Na operaci *Abandon* server neposílá žádnou odpověď, klient tedy nedostane informaci, zda operace byla skutečně přerušena.

Operace vyhledávání v adresářovém serveru má následující atributy (uvedeny nejdůležitější atributy, kompletní výpis je uveden v [30]):

baseObject – DN záznamu, ke kterému má být vyhledávání vztaženo.
scope – oblast vyhledávání. Může mít následující hodnoty:

- **baseObject** – prohledávání pouze záznamu definovaného v baseObject. Vyhledávání s tímto scope tedy vždy vrací nejvýše jeden záznam.
- **singleLevel** – prohledávání přímých následovníků záznamu baseObject, ale nikoliv záznamu samotného.
- **wholeSubtree** – prohledávání záznamu baseObject a všech jeho následovníků, přímých i nepřímých.

sizeLimit – maximální počet výsledků, které mají být vyhledáváním vráceny.

timeLimit – maximální doba v sekundách, po kterou má vyhledávání probíhat. Hodnota 0 znamená, že nemá být čas vyhledávání omezen. Server však může aplikovat vlastní limit.

filter – podmínky, kterým musejí hledané záznamy odpovídat. V protokolu je použita binární reprezentace filtru, v klientských aplikacích se však používá textová reprezentace popsaná v [18]. Gramatika textové reprezentace z tohoto dokumentu je pro ilustraci uvedena ve výpisu 2.1.

attributes – seznam atributů, které mají být u záznamů ve výsledku vyhledávání vráceny. Pokud nejsou uvedeny žádné atributy, jsou vráceny všechny.

```
<filter> ::= '(' <filtercomp> ')'
<filtercomp> ::= <and> | <or> | <not> | <item>
<and> ::= '&' <filterlist>
<or> ::= '|' <filterlist>
<not> ::= '!' <filter>
<filterlist> ::= <filter> | <filter> <filterlist>
<item> ::= <simple> | <present> | <substring>
<simple> ::= <attr> <filtertype> <value>
<filtertype> ::= <equal> | <approx> | <greater> | <less>
<equal> ::= '='
<approx> ::= '~='
<greater> ::= '>='
<less> ::= '<='
<present> ::= <attr> '=*'
<substring> ::= <attr> '=' <initial> <any> <final>
<initial> ::= NULL | <value>
<any> ::= '*' <starval>
<starval> ::= NULL | <value> '*' <starval>
<final> ::= NULL | <value>
```

Výpis 2.1: Gramatika textové reprezentace filtru protokolu LDAP. Převzato z [18].

Příklad filtru: `(&(objectClass=person)(|(sn=V*)(sn=Novák)))` – filtru odpovídají záznamy, jejichž atribut objectClass je roven „person“ a jejich atribut „sn“ (příjmení) začíná na „V“, nebo je roven „Novák“.

Název operace	**Popis**
Bind	Ověření uživatele.
Unbind	Ukončení LDAP sezení a uzavření spojení.
Search	Vyhledávání v adresáři. Server vrací jeden nebo více záznamů, které odpovídají zvoleným kritériím. Tato operace může být použita k získání atributů jednoho záznamu, záznamů přímo podřízených danému záznamu, nebo celému podstromu daného záznamu.
Modify	Úprava jednoho záznamu v adresáři.
Add	Přidání nového záznamu do adresáře. Nový záznam je umístěn do struktury podle zadaného DN. První komponenta DN je použita jako RDN.
Delete	Odstranění záznamu se zadaným DN.
Modify DN	Změna DN záznamu. Změnou DN záznamu lze změnit jeho RDN a/nebo přesunout záznam a celý jeho podstrom na jiné místo ve struktuře.
Compare	Umožňuje ověřit, zda atributy záznamu se zadaným DN odpovídají zadaným pravidlům.
Abandon	Zrušení předchozího požadavku.
Extended	Tato operace slouží k rozšíření protokolu. Server může nadefinovat doplňující operace, které poté mohou být využity klientem. Příkladem takové operace může být operace StartTLS, která slouží pro zahájení šifrované komunikace.

Tabulka 2.5: Operace definované protokolem LDAP.

3 Řízení přístupu v počítačových sítích: NAC

Podle [31] je NAC označení pro metody a technologie, jejichž hlavním cílem je zvýšit zabezpečení sítě pomocí ověřování identity, integrity a zabezpečení každé koncové stanice před tím, než je stanici umožněn přístup do sítě. [16] však uvádí, že přesný význam NAC a co všechno by řešení mělo zahrnovat, se liší dle jednotlivých dodavatelů a nikdy nebyl přesně vymezen.

V této kapitole budou popsány základní principy NAC a vymezeny často používané pojmy. Dále pak budou popsány dva standardizované protokoly, které jsou v používány v souvislosti s ověřováním koncových stanic v síti.

3.1 Základní principy NAC

Pravděpodobně nejkomplexnější pohled na NAC poskytuje společnosti Trusted Computing Group (TCG). Ta spolu s dalšími společnostmi standardizovala otevřenou architekturu a rodinu protokolů TNC (Trusted Networking Communications). V následujícím textu bude popsán právě tento standard. Řešení jiných dodavatelů používají stejné principy, jejich terminologie a konkrétní implementační detaily se však mohou lišit.

Aktuální verzí standardu je 1.5, revize 4, vydaná v květnu 2012 [34]. V tomto standardu jsou definovány dvě základní a povinné role, které musejí být v síti zastoupeny:

Access Requestor (AR): koncová stanice, která žádá o udělení přístupu do sítě.

Policy Decision Point (PDP): server či síťové zařízení, které rozhoduje o udělení přístupu koncové stanice do sítě a k jednotlivým prostředkům v ní. Rozhodování probíhá na základě pravidel definovaných správcem sítě.

Kromě těchto rolí pak standard definuje další 3 role nepovinné:

Policy Enforcement Point (PEP): prvek, ke kterému je připojena koncová stanice (AR). Zajišťuje uplatnění pravidel pro přístup, která uděluje PDP. Může jít například o přístupový přepínač, bránu pro Virtual Private Network (VPN), firewall či jiný síťový prvek.

Metadata Access Point (MAP): ukládá a poskytuje informace o stavu jednotlivých stanic (AR). Tyto informace mohou být použity pro rozhodování a vynucování přístupových pravidel. Hlavní úlohou MAP je umožnit komponentám v síti, které se neúčastní procesu udělení přístupu do sítě, přístup k informacím o připojených stanicích. Další prvky sítě, například sondy síťového provozu, mohou do MAP ukládat informace pro další rozhodování.

MAP Client (MAPC): komponenta, která přistupuje k informacím (čte a/nebo zapisuje) o stanicích v MAP.

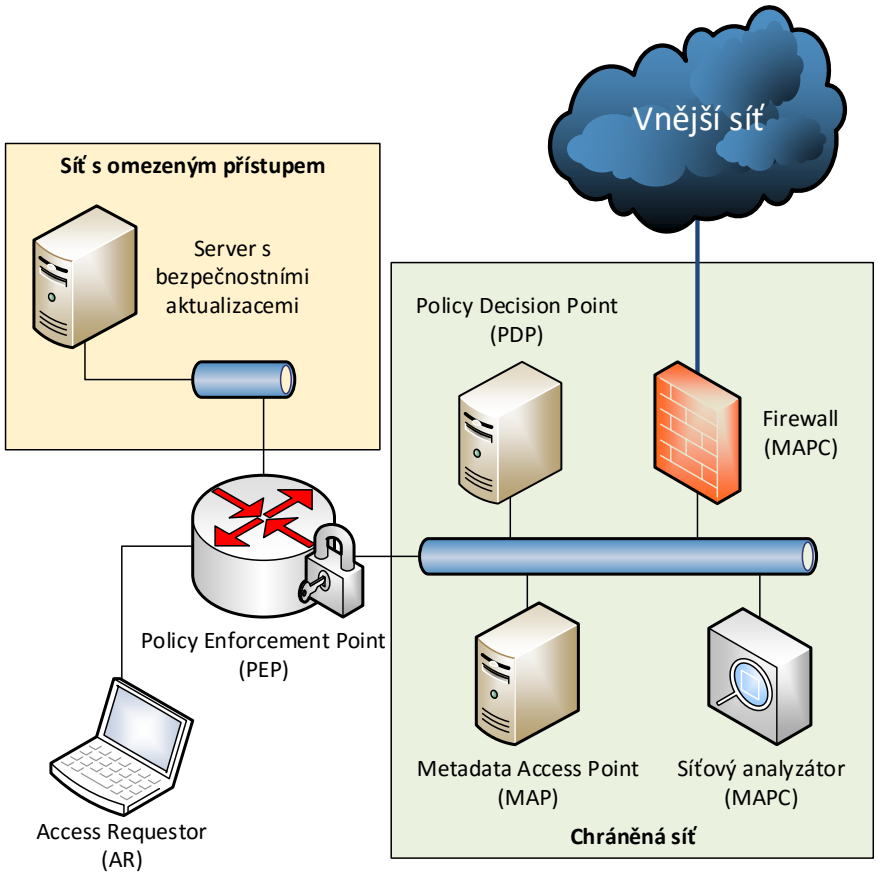
Konkrétní role, které budou v síti zastoupeny, se mohou lišit dle typu dané sítě a požadavků, které jsou kladeny na její zabezpečení. Rozhodování o udělení přístupu může probíhat například na základě následujících informací:

- identita koncové stanice a uživatele,
- zabezpečení stanice: aktuální verze antivirového programu, instalovány bezpečnostní aktualizace softwaru,
- lokalita, kde je stanice připojena, a aktuální čas.

Možná architektura sítě je znázorněna na obrázku 4.1. Postup udělení přístupu v takové síti je následovný:

1. Koncová stanice (AR) se připojí do sítě a proběhne její ověření proti PEP. Stanice také poskytne informace o jejím software a použitých verzích.
2. PEP informace o stanici přepošle PDP.
3. Pokud stanice vyhovuje definovaným pravidlům, je jí umožněn přístup do sítě a poskytnuté informace jsou uloženy do serveru MAP.
4. Na základě informací v MAP umožní firewall stanici přístup do vnější sítě, například do sítě internet.
5. Síťový analyzátor průběžně analyzuje chování stanice v síti. Pokud detekuje, že stanice je infikována škodlivým kódem, zapíše tuto informaci do MAP serveru a je jí odepřen přístup do sítě.

Pokud by stanice při kontrole v kroku 3 nevyhovovala definovaným pravidlům (chybějící či zastaralý antivirový program, nebo neplatné uživatelské jméno a heslo), bude jí umožněn přístup pouze do sítě s omezeným přístupem. V této síti může být dostupný například server s bezpečnostními aktualizacemi či další konfigurační servery, které stanici umožní opravit případné nedostatky v její konfiguraci.



Obrázek 3.1: Příklad uspořádání sítě s NAC.

V síti s omezeným přístupem může být dostupný také tzv. captive portal. Jedná se zpravidla o webový server, který uživateli umožní registraci zařízení do sítě a jeho správnou konfiguraci. Na tento webový server je přesměrována komunikace koncové stanice, dokud neproběhne její ověření a přihlášení do sítě.

Některá řešení NAC ke své správné funkci vyžadují, aby byl na koncových stanicích nainstalován speciální software – agent. Ten poskytne PEP informace o konfiguraci stanice: verze softwaru, nainstalované bezpečnostní záplaty a další. Řešení, které nevyžaduje instalaci agenta na koncové stanice, se nazývá Agentless NAC.

3.2 Používané protokoly

V řešeních NAC se často používají protokoly 802.1X, pomocí kterého se koncové stanice ověřují k PEP, a RADIUS (Remote Authentication Dial In User Service), pomocí kterého může PEP komunikovat s PDP. Konkrétní

řešení mohou používat i jiné, často proprietární, protokoly.

3.2.1 Protokoly 802.1X a PP

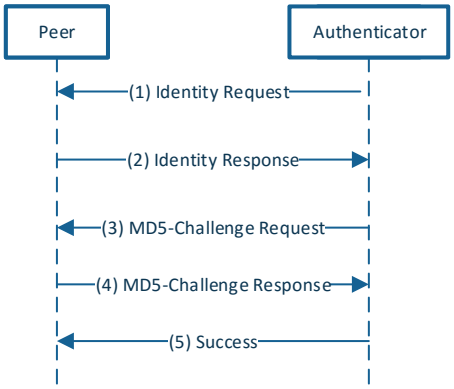
Protokol 802.1X umožňuje řízení přístupu na linkové úrovni. Je možné jej použít jak v drátových sítích, kde řídí přístup na úrovni fyzických přístupových portů, tak i v sítích bezdrátových. Protokol definuje způsob, jakým je možné do ethernetových rámců zapouzdřit zprávy protokolu EAP (Extensible Authentication Protocol). Toto zapouzdření se nazývá EAPOL (EAP over LAN). [6] Protokol 802.1X je standardizován organizací Institute of Electrical and Electronics Engineers (IEEE), protokol EAP je definován v [1]. Dále bude popsán protokol EAP, ze kterého protokol 802.1X vychází.

Protokol EAP rozlišuje dvě role:

Authenticator: strana, která iniciuje a vykonává ověřování. V případě sítí LAN jde například o přepínač v roli PEP. Authenticator vysílá vždy požadavky (Request) a čeká na jejich odpověď od koncové stanice.

Peer: koncová stanice, která je ověřována. Zpracovává požadavky od authenticator a posílá odpovědi (Response). V případě protokolu 802.1X zajišťuje zpracování požadavků software označovaný jako Supplicant.

Protokol definuje, že zařízení a aplikace musejí podporovat typy zpráv uvedené v tabulce 3.1.



Obrázek 3.2: Ověřování uživatele protokolem EAP.

Postup ověřování protokolem EAP je znázorněn na obrázku 3.2:

1. Authenticator vyžádá informace o identitě uživatele zprávou *Identity Request*.

Typ zprávy	**Popis**
Identity	Dotaz na identitu uživatele. Stanice odpovídá také zprávou typu identity.
Notification	Authenticator zasílá zprávu, která má být zobrazena uživateli. Zpráva je textový řetězec v kódování UTF-8. Stanice musí zprávu potvrdit zprávou Notification Response.
NAK	Stanice touto odpovědí indikuje, že nepodporuje požadovanou metodu ověřování.
MD5-Challenge	Authenticator vyzývá k ověření pomocí algoritmu MD5. V požadavku je uvedena náhodně vygenerovaná hodnota, stanice k hodnotě připojí své heslo, vypočítá MD5 součet výsledku a zašle jej zpět.
Expanded Types	Umožňuje rozšíření protokolu o další ověřovací algoritmy a protokoly. Ve zprávě je obsažen další identifikátor typu zprávy.
Success	Authenticator potvrzuje úspěšné ověření uživatele.
Failure	Authenticator indikuje, že ověření nebylo úspěšné.

Tabulka 3.1: Typy zpráv protokolu EAP.

2. Peer vyzve uživatele k zadání jména a hesla. Jméno zašle zpět ve zprávě *Identity Response*.
3. Authenticator vyžádá ověření uživatele zprávou *MD5-Challenge Request* s náhodně vygenerovanou hodnotou.
4. Peer zpracuje zprávu, k zaslané hodnotě připojí uživatelské heslo, výsledek transformuje algoritmem MD5 a zašle zpět ve zprávě *MD5-Challenge Response*.
5. Pokud je zadané heslo správné, potvrdí Authenticator úspěšné přihlášení zprávou *Success*. Pokud ne, zašle zprávu *Failure*.

Pokud je pro ověření uživatele použito více metod, mohou se kroky 3 a 4 opakovat do té doby, než bude ověření dokončeno. Zpráva *Success* je zaslána až po úplném ověření uživatele.

Protokol 802.1X přidává navíc další typy zpráv (rámců):

EAPOL-Start: Supplicant (Peer) zahajuje ověřování bez čekání na zprávu od Authenticatoru.

EAPOL-Logoff: Supplicant provádí odhlášení. Odhlášení se provede také v případě, že se koncová stanice fyzicky odpojí.

3.2.2 Ověřování uživatelů pomocí RADIUS (AAA)

Protokol RADIUS je protokol pro autentizaci, autorizaci a účtování (anglicky AAA: Authentication, Authorization and Accounting) vzdálených uživatelů. Pomocí tohoto protokolu mohou ostatní služby v síti (například DHCP server, telefonní ústředna, souborový server či přímo síťové prvky v roli PEP) provést ověření uživatele a určení jeho oprávnění proti centrální databázi uživatelů na vzdáleném serveru. [13]

Tento protokol také může být použit pro měření využití služby (počet přenesených dat, délka sezení a další) pro potřeby vyúčtování či určení dalších statistik. Protokol je definován v [29] a [28]. Používá UDP port 1812.

Klienti (typicky síťové služby, ke kterým se přihlašují uživatelé z klientských stanic), kteří mají využívat služeb RADIUS serveru, musejí mít na serveru nastavené sdílené heslo. Tímto heslem se serveru ověřují, aby nemohlo dojít ke zneužití služeb. Pokud se serverem komunikuje klient, který na serveru není nakonfigurován, nebo který má špatné heslo, jsou jeho požadavky ignorovány.

Protokol definuje typy zpráv uvedené v tabulce 3.2. Všechny zprávy mají stejný formát, od sebe jsou odlišeny kódem v hlavičce zprávy. Data zprávy tvoří sada atributů jejich hodnot.

Komunikace mezi klientem a serverem při ověřování uživatele probíhá následovně:

1. Klient posílá požadavek *Access-Request*, ve kterém uvede přístupové údaje klienta, například uživatelské jméno (atribut *User-Name*), heslo (*User-Password*), umístění (fyzický port, IP adresu, ...), ze kterého se přihlašuje, a jiné.
2. Server přijme požadavek a ověří správnost údajů. Server ověřuje heslo, ale může navíc ověřovat další údaje. Server může údaje přijmout a odpovědět zprávou *Access-Accept*, nebo vyžádat další ověření uživatele zprávou *Access-Challenge*. V této zprávě server uvede náhodně vygenerovanou hodnotu. V atributu *Challenge-Message* je navíc zpráva, kterou klient musí zaslat zpět. Pokud jsou údaje chybné, server odpovídá zprávou *Access-Reject*.
3. Klient přijme zprávu *Access-Challenge*, vyžádá od uživatele zašifrování náhodného čísla (například jeho soukromým klíčem) a zašle serveru novou zprávu *Access-Request*. Klient místo *User-Password* zašle zašifrovanou náhodnou hodnotu.

Typ zprávy	**Popis**
Access-Request	Požadavek na ověření klienta.
Access-Accept	Potvrzení správnosti přístupových údajů.
Access-Reject	Odmítnutí přístupových údajů. Mohou být špatné, nebo se klient může přihlašovat z nedovoleného umístění.
Accounting-Request	Klient posílá metriky o sezení k účtování. Konkrétní typ požadavku je uveden v atributu Acct-Status-Type, může jít o zahájení sezení, průběžné zasílání informací o sezení, nebo ukončení sezení a zaslání souhrnných metrik o sezení.
Accounting-Response	Potvrzení přijetí účtovacích metrik.
Access-Challenge	Server posílá náhodně vygenerovanou hodnotu, kterou klient musí zašifrovat svým tajným klíčem, aby se ověřil.
Status-Server	Zpráva, kterou klient může ověřit dostupnost daného serveru. Zpráva a její zpracování jsou definovány v [11].
Status-Client	Zpráva, jejíž význam nebyl definován, avšak její kód je uveden ve specifikaci.

Tabulka 3.2: Zprávy použité v protokolu RADIUS.

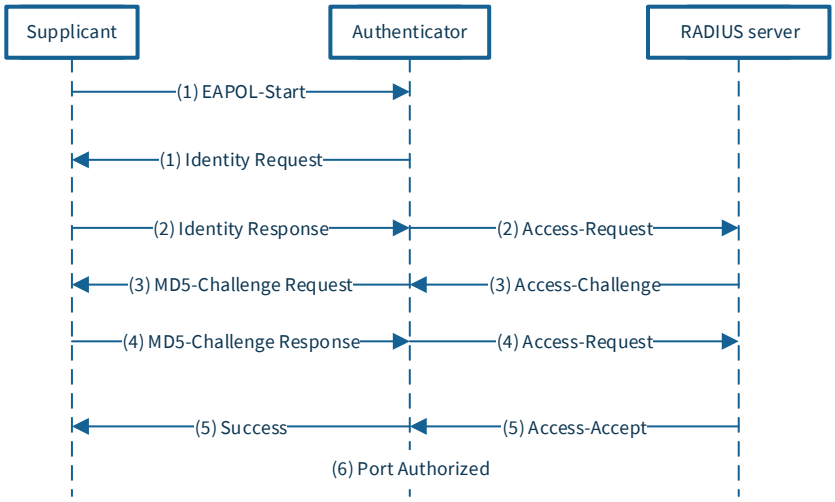
4. Server ověří platnost požadavku a klientovi jejich platnost potvrdí zprávou *Access-Accept*, odmítne zprávou *Access-Reject*, nebo vyžádá další ověření zprávou *Access-Challenge*.
5. Po přijetí zprávy *Access-Accept* klient umožní uživateli přístup.

Klientem RADIUS serveru typicky není koncová stanice, ale síťová služba, ke které se připojují uživatelé. Pokud má síťová služba podporovat rozšířené ověřování uživatele zprávou *Access-Challenge*, musí v jejím komunikačním protokolu být definováno, jakým způsobem informace o tomto ověřování vyměnit s uživatelem. V případě řízení přístupu do sítě na úrovni portů může být takovým protokolem 802.1X, který byl popsán v předchozím oddíle.

3.2.3 Vztah mezi protokoly EAP a RADIUS

Autorizační schéma protokolů RADIUS a EAP je vzájemně kompatibilní, takže PEP, u kterého se koncová stanice ověřuje, může zprávy mezi těmito protokoly překládat. S koncovou stanicí tak komunikuje protokolem EAP v Ethernetových rámcích (802.1X), s PDP pak protokolem RADIUS.

Kooperace těchto protokolů a ověření klienta protokolem 802.1X proti RADIUS serveru je znázorněno na obrázku 3.3. Roli Authenticator má přístupový přepínač, ke kterému je koncová stanice připojena.



Obrázek 3.3: Ověřování klienta protokolem 802.1X proti RADIUS serveru.

Po připojení neautorizovaného klienta je port v neověřeném stavu, ve kterém jím mohou projít pouze autorizační EAPOL rámce. Po úspěšném ověření klienta (6) je port uveden do autorizovaného stavu, ve kterém je povolena běžná komunikace.

4 Existující řešení pro NAC

Na trhu nalezneme celou řadu řešení, která poskytují NAC. Téměř každá společnost dodávající síťové prvky pro podnikové použití v minulosti představila vlastní řešení této problematiky. Jednotlivá řešení se liší podporovanými funkcemi, možnostmi ověřování uživatelů, administračním rozhraním a dalšími vlastnostmi.

Protože jsou v kolejní síti použity prvky společnosti Cisco, bude v následujícím textu popsáno právě řešení této společnosti. Dále bude popsáno řešení společnosti Microsoft, které je nezávislé na použitých síťových prvcích, a vybraná open-source řešení.

4.1 Cisco Network Admission Control

Network Admission Control (NAC) je komplexní řešení od společnosti Cisco. Ověřování uživatelů a uplatnění pravidel je integrováno v přepínačích této společnosti, konfiguraci a rozhodování o udělení přístupu zajišťuje specializovaný Cisco NAC Server. Na koncových stanicích musí být nainstalován Cisco NAC Agent, hosté mohou využívat webovou verzi agenta. Ta vyžaduje, aby klientská stanice podporovala technologii ActiveX, nebo Java 1.5 a vyšší. [10]

Ověřování probíhá proti již existujícím zdrojům pověření: RADIUS server, Windows Active Directory, Kerberos server a dalším. Řešení nabízí možnost vytvořit captive portal, který uživatelům umožní registraci zařízení do sítě.

V roce 2015 společnost oznámila plán ukončení prodeje a ukončení podpory tohoto řešení. [9]

4.2 Microsoft Network Access Protection

Network Access Protection (NAP) je řešení společnosti Microsoft, které je možné použít se síťovými prvky různých výrobců. Ověřování koncových stanic v sítích LAN provádějí přepínače podporující 802.1X, nebo DHCP server společnosti Microsoft. Rozhodování o udělení přístupu provádí Network Policy Server, který lze nakonfigurovat na serverech s operačním systémem Windows Server. [23]

Poskytování informací o použitém software klientské stanice zajišťuje NAP Client, který je součástí operačních systémů Windows XP SP3, Windows Vista, Windows 7, Windows 8 a Windows 8.1.

Operační systém Windows 10 již neobsahuje NAC Client a Windows Server 2016 odebírá podporu technologie NAP. [37]

4.3 PacketFence

PacketFence[1] je kompletní open-source řešení s Captive Portal. Podporuje prvky celé řady výrobců, včetně prvků společnosti Cisco, které jsou nasazeny v kolejní síti. Ověřování koncových stanic probíhá protokolem 802.1X, PacketFence v sobě integruje RADIUS server, který ukládá informace o registrovaných uživatelích a koncových stanicích.

Stav stanice je ověřován buď protokolem TNC SoH (Statement of Health), který musí koncová stanice podporovat. Je také možné použít síťové sondy, například Snort[2] či Suricata[3]. Řešení je tedy možné provozovat i v agentless režimu. Ověřování uživatelů může probíhat proti celé řadě externích zdrojů, například Active Directory, LDAP, Kerberos serveru, RADIUS serveru, či externím službám, například Google, Facebook či LinkedIn. Kompletní seznam podporovaných zdrojů a funkcí je uveden v oficiální dokumentaci [2].

Systém poskytuje JSON RPC (JSON Remote Procedure Call) API (Application Programming Interface), pomocí kterého je možné k datům přistupovat z jiných systémů. Podporuje pouze několik základních volání na zjišťování informací o zařízeních a pro registraci nových zařízení.

Systém vyvíjí převážně společnost Inverse inc., která poskytuje i komerční podporu. Dle oficiálního webu systém provozuje například University of Torronto nebo Seattle Pacific University. Dokumentace produktu je úplná a poskytuje kompletní přehled možností řešení a průvodce konfigurací samotného produktu i síťových zařízení.

4.4 pfSense

pfSense[4] je firewall s funkcemi NAC. Je založený na distribuci FreeBSD. Kromě firewallu poskytuje další volitelné moduly, především Captive Portal s integrací RADIUS serveru, load balancing, monitoring využití sítě a další. [26]

pfSense předpokládá, že bude v síti zajišťovat funkci gateway. Neposkytuje přímou podporu pro spolupráci se síťovým hardware, avšak je možné prvky

[1] http://packetfence.org/
[2] https://www.snort.org/
[3] https://suricata-ids.org/
[4] http://www.pfsense.org/

podporující 802.1X nastavit tak, aby využívaly integrovaný RADIUS server.

Systém vyvíjí společnost Electric Sheep Fencing LLC a poskytuje i komerční podporu. Společnost nabízí také již předinstalovaný specializovaný hardware.

4.5 OpenNAC

OpenNAC[5] je systém pro řízení přístupu do sítí drátových i bezdrátových sítí LAN. Skládá se z několika modulů: základního modulu pro NAC, modulu pro konfiguraci síťových zařízení či modulu pro monitoring sítě. Konfigurace je ukládána v centrální databázi, kde jsou uloženy i informace o registrovaných zařízeních. [25]

Ověřování koncových stanic probíhá pomocí protokolu 802.1X. Systém nabízí registrační portál, ve kterém uživatelé mohou zaregistrovat své zařízení. Uživatelské účty mohou být uloženy v lokální databázi, nebo v externích zdrojích – Active Directory či LDAP server. Řešení podporuje protokol TNC SoH.

Systém také poskytuje komplexní REST (Representational State Transfer) API, pomocí kterého je možné přistupovat k datům systému. Pomocí tohoto API je podle dokumentace možné přistupovat téměř ke všem objektům systému.

Systém vyvíjí společnost OPENNAC TECH a poskytuje i komerční podporu. Dokumentace je dostupná pouze ve formě wiki v systému Redmine. Některé pasáže dokumentace nejsou úplné, nebo úplně chybí.

4.6 Srovnání a vhodnost řešení

V předchozím textu byla popsána různá řešení, která by nevyžadovala hardwarové změny stávající infrastruktury kolejní sítě a s nimi spojené investice.

Řešení od společnosti Cisco a Microsoft jsou již ve stádiu ukončení podpory. Nejsou proto vhodné pro nová nasazení. Řešení pfSense je postavené na bázi FreeBSD, které však není na Západočeské univerzitě v Plzni podporováno, a řešení, která lze provozovat na systému Linux, jsou vhodnější. Navíc je nutné jej provozovat pouze v režimu gateway, což může způsobovat problémy s výkonem při přístupu velkého množství koncových stanic.

Zbývající řešení jsou si v mnoha ohledech podobná: PacketFence i OpenNAC podporují protokol 802.1X, mohou ověřovat uživatele proti externím

[5] http://www.opennac.org/opennac/en.html

zdrojům, podporují protokol TNC SoH. PacketFence má však lepší dokumentaci, širší podporu síťových prvků a externích ověřovacích metod a lze jej snadno integrovat se síťovými sondami. Nevýhodou PacketFence je omezené API, ke kterému navíc není k dispozici dokumentace.

Obě zbývající řešení používají protokol 802.1X. Ten musí podporovat jak síťové prvky, tak připojené stanice. Operační systém Windows, který je v kolejní síti nejrozšířenější, však i v jeho nejnovější verzi Windows 10 vyžaduje dodatečné povolení a konfiguraci protokolu 802.1X v drátových sítích. Tato konfigurace však může být pro některé uživatele příliš složitá.

Příkladem řešení je například poskytnutí přístupu neověřeným stanicím do hostovské sítě (Guest VLAN), ve které bude dostupný captive portál s automatickým konfigurátorem síťového připojení. Podobný automatický konfigurátor se již nyní používá pro automatickou konfiguraci bezdrátové sítě Eduroam.

Použité prvky (Cisco Catalyst 2960) podporují ověřování v následujících režimech:

Single-Host: na portu je povolena komunikace pouze jednomu zařízení, které se musí ověřit.

Multi-Host: první zařízení po ověření přepne port do autorizovaného režimu. Při použití dalšího přepínače je možné připojit libovolný počet dalších zařízení, která se nemusí ověřovat.

Multi-Auth: každé připojené zařízení se musí ověřit.

Protože k jednomu fyzickému portu koncového zařízení může být připojeno více zařízení různých uživatelů[6], je nutné, aby se každé zařízení při připojení do sítě ověřovalo. Přepínač by tedy měl fungovat v režimu Multi-Auth.

Na použitých prvcích však v tomto režimu nefunguje přiřazení nenakonfigurovaných stanic do hostovské sítě (tzv. Guest VLAN)[7] [32]. Uživatelé by tak neměli možnost automaticky nakonfigurovat své stanice. Jiné ověřovací režimy by ale buď znemožnily připojení některých uživatelů (Single-Host), nebo umožnily připojení neověřených zařízení a obcházení definovaných pravidel pro připojení v síti (Multi-Host).

Z těchto důvodů je protokol 802.1X, na kterém jsou zmíněná softwarová řešení postavena, se stávající infrastrukturou v kolejních sítích nevhodný pro nasazení.

[6]V akademickém roce 2015/2016 Správa kolejí a menz ZČU navýšila počet lůžek v některých pokojích bez úpravy strukturované kabeláže. V těchto pokojích tak uživatelé musejí sdílet fyzickou zásuvku a používat vlastní přepínače. Uživatel také může připojit vlastní přepínač, ke kterému se připojí další uživatelé.

[7]Novější prvky Cisco Catalyst 2960X tuto funkci v režimu Multi-Auth podporují.

5 Analýza stávajícího stavu

V této kapitole bude popsána kolejní síť a způsob, jakým je v současné době řešeno řízení přístupu do sítě, stávající registrační portál a proces, kterým musí uživatelé projít, aby zaregistrovali svá zařízení.

5.1 Architektura sítě a síťové služby

Architektura kolejní sítě vychází z hierarchické topologie společnosti Cisco. Neobsahuje však redundanci prvků – prvky v distribuční vrstvě nejsou zdvojené.

V přístupové vrstvě jsou použity L2 přepínače Cisco Catalyst 2960. Na nich je nastavena základní bezpečnostní politika: omezují počet ARP paketů, které může koncová stanice do sítě vyslat za jednotku času, omezují připojování dalších konfigurovatelných přepínaču funkcí BPDU (Bridge Protocol Data Unit) Guard a provádějí DHCP Snooping – kontrolu, zda připojená stanice získala adresu z některého z důvěryhodných DHCP serverů. V případě, že stanice má staticky nakonfigurovanou IP adresu, je jí odepřen přístup do sítě.

V distribuční vrstvě jsou použity multilayer přepínače Cisco Catalyst 3550 a 3750. Tyto prvky zajišťují směrování v kolejní síti a propojují jednotlivé segmenty sítě. Prvky také slouží jako DHCP Relay Agent – DHCP požadavky koncových stanic přeposílají na centrální DHCP server.

Všechny kolejní sítě jsou připojeny do centrálního přepínače Catalyst 6509. K němu jsou připojeny také servery, které zajišťují běh registračního portálu Knet a běh DHCP a DNS serveru pro koncové stanice. DHCP server přiděluje registrovaným stanicím jim rezervovanou IP adresu. Na základě IP adresy je tak možné jednoznačně stanici identifikovat v síti. Neregistrovaným stanicím přidělí dynamicky IP adresu z vyhrazeného rozsahu.

Síť je připojena do zbytku univerzitní sítě prostřednictvím linuxové gateway – stroj `gatek.zero.zcu.cz`. Ta provádí překlad adres (Network Address Translation, NAT) z privátních rozsahů z adresního prostoru 10.0.0.0/8 na adresy ve veřejném rozsahu univerzity. `gatek` také slouží jako firewall a provoz z neregistrovaných stanic směruje na registrační portál.

Na linuxové gateway je také řešena karanténa stanic – provoz ze zablokovaných stanic je omezen a je povolen přístup pouze na určité IP adresy – webové rozhraní pro e-mail a stránky uživatelské podpory na support.zcu.cz.

5.2 Registrační portál KNet

Portál, který uživatelé využívají k registraci svých zařízení, vznikl v roce 2007 v rámci bakalářské práce Michala Zbortka [38]. Jedná se o webovou aplikaci napsanou v jazyce PHP. Aplikace využívá framework Mojavi, který již v současnosti není udržován a jeho oficiální web již neexistuje.

S portálem pracují jak uživatelé, studenti, kteří jeho prostřednictvím registrují svá zařízení do sítě, tak lokální správci, kteří schvalují jednotlivé registrace, upravují registrace a mohou měnit konfiguraci sítě.

Neregistrovaná zařízení jsou při pokusu o přístup do internetu protokolem HTTP přesměrována na tento registrační portál. Po přihlášení pomocí jednotného přihlášení WebAuth si poté uživatel zaregistruje svůj počítač. Při registraci uživatelé zadávají MAC adresu počítače, zvolený hostname a své umístění: kolej, pokoj a zásuvku. MAC adresu systém ve formuláři předvyplňuje na základě informací získaných z ARP tabulky směrovače. Systém pak zajistí vytvoření záznamu pro daný počítač v DHCP a DNS serverech.

Lokální správce má možnost poté registraci schválit. Může také provést její úpravu, pokud uživatel například vybral špatný pokoj, nebo pokud se přestěhuje na jiný pokoj či kolej. Správci také mohou blokovat uživatele v případě, že poruší pravidla sítě, nebo na základě nahlášených bezpečnostních incidentů.

Správce může mít různé úrovně oprávnění:

reg_local: může schvalovat a upravovat registrace pouze na koleji, ke které je přiřazen. Má také možnost blokovat uživatele z této koleje a může zrušit blokaci pouze toho uživatele, kterého zablokoval.

reg_group: může schvalovat a upravovat registrace na všech kolejích ze stejné skupiny kolejí, ke které je přiřazen. Může blokovat uživatele z této skupiny a zrušit blokaci toho uživatele, kterého zablokoval.

reg_all: může schvalovat a upravovat registrace na všech kolejích, může blokovat všechny uživatele a zrušit všechny blokace.

admin: Může konfigurovat prvky kolejní sítě.

config: Oprávnění není dle zdrojových kódů použito.

Prostředí aplikace je vícejazyčné a obsahuje řetězce pro angličtinu a češtinu. Datový model nepodporuje lokalizaci, takže například při zablokování uživatele musí správce zadat poznámku v jazyce, kterému uživatel pravděpodobně bude rozumět, nebo poznámku zadat dvojjazyčně.

Aplikace neumožňuje plánování akcí. Pokud například správce zablokuje některého uživatele kvůli porušování pravidel sítě na týden, musí jej po týdnu manuálně odblokovat. Obdobně je pak nutné ručně spouštět mazání

existujících registrací na konci června při uzavření většiny kolejí a zahájení letního režimu.

5.3 Architektura

Webová aplikace je členěna do vrstev dle architektury MVC (Model-View-Controller). Datovou vrstvu tvoří třídy modelu, které zajišťují přístup do MySQL databáze. Aplikační vrstvu tvoří třídy jednotlivých akcí, které zpracovávají požadavky uživatele. Prezentační vrstvu tvoří šablony HTML kódu spolu s pomocnými třídami, které provádějí předzpracování dat pro šablony a určují, které bloky šablony budou vykresleny.

Aplikace je dělena do modulů, které obsahují jednotlivé části aplikace. Každý modul má vlastní konfiguraci, šablony, třídy modelu, akcí a prezentační vrstvy. Popis jednotlivých modulů je uveden v [38].

Kromě kódu v modulech obsahuje projekt také pomocné třídy, které zajišťují například načítání informací z adresáře LDAP, načítání informací z přepínačů protokolem SNMP či jejich konfiguraci pomocí protokolů SSH (Secure Shell) a Telnet.

Aby byl registrovaným stanicím umožněn přístup do sítě internet, musí systém zajistit aktualizaci záznamů v DHCP a firewallu. Systém tuto konfiguraci nemění přímo, ale v pravidelných intervalech generuje konfigurační soubory pro zmíněné služby. V současnosti je tento interval nastaven na 10 minut.

5.4 Konfigurace a rozšiřitelnost portálu

Webový portál používá konfigurační soubory pouze pro svou interní konfiguraci (konfigurace pro autoloading, konfigurace cest, názvů tříd pro zpracování výjimek apod.) a pro konfiguraci přístupu do databáze. Adresy síťových prvků jsou pak uloženy v databázi.

V konfiguraci není možné nastavit celou řadu voleb, například adresy a přístupové údaje do LDAP, hesla pro připojení do síťových prvků[1], seznam IP adres, které mohou stahovat konfiguraci, a další. Pokud dojde ke

[1]Hesla jsou také uvedena ve zdrojových kódech, které jsou součástí bakalářské práce [38] na přiloženém CD. Libovolný student či zaměstnanec ZČU si tak může ke studiu v knihovně vypůjčit CD obsahující heslo pro administrátorský SSH přístup do kolejních síťových prvků i heslo pro čtení protokolem SNMP. SSH přístup je naštěstí omezen pouze na určité IP adresy.

změně některého z uvedených údajů, musí se upravit přímo zdrojové soubory aplikace.

Portál lze poměrně snadno rozšířit o další moduly. Kód hotových modulů je vhodně dekomponován do tříd a metod, ale vazby mezi nimi jsou velmi těsné. To ztěžuje úpravy stávající funkcionality, existující moduly nelze snadno rozšířit a také komplikuje veškeré změny v infrastruktuře. Pokud dojde k výměně síťového prvku za jiný typ, který není s původním kompatibilní, musí se způsob komunikace s ním upravit změnou stávajících tříd.

5.5 Zabezpečení

Zabezpečení je nejslabší stránkou celé aplikace. Její uživatelské vstupy nejsou dostatečně ošetřeny. Nejzávažnějším nedostatkem je fakt, že se aplikace spoléhá na funkcionalitu `magic_quotes` při operacích s databází. Tato funkcionalita byla odstraněna v PHP 5.4.0. Při aktualizaci na tuto či novější verzi PHP by tedy aplikace obsahovala zranitelnost SQL injection a bylo by možné například smazat obsah databáze nebo modifikovat tabulku oprávnění.

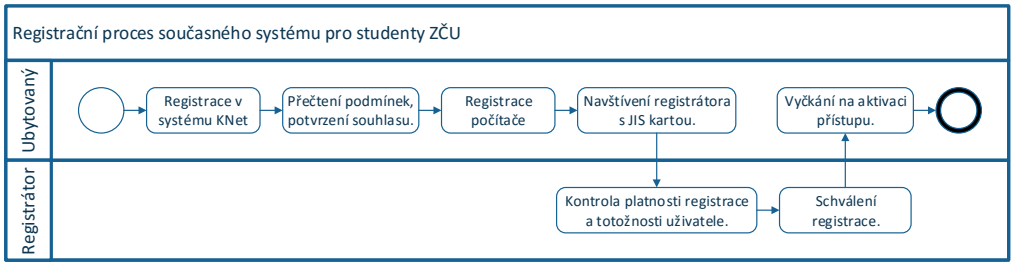
Aplikace dále obsahuje neošetřený vstup ve formuláři, pomocí kterého správce zadává své kontaktní údaje. Správce může vkládat HTML, včetně elementu `<script>`. Tím se otevírá zranitelnost Cross Site Scripting (XSS). Vstupy formulářů také nejsou ošetřeny proti Cross Site Request Forgery (CSRF), v kombinaci těchto dvou zranitelností může lokální správce poměrně snadno získat oprávnění hlavního správce[2].

5.6 Proces registrace uživatele

Současný proces registrace uživatele s Orion účtem v notaci BPMN (Business Process Model and Notation) je znázorněn na obrázku 5.1. V rámci procesu je vyžadováno ověření totožnosti uživatele, aby se snížilo riziko případů, kdy jedna osoba registruje pod svým uživatelským účtem počítače třetích osob. Toto ověření provádějí lokální správci a registrátoři kontrolou JIS karty, nebo jiného dokladu totožnosti.

Tento proces funguje pouze pro studenty ZČU s aktivním Orion účtem. Na kolejích ZČU však mohou být ubytováni i studenti jiných univerzit, zejména pak Lékařské fakulty Univerzity Karlovy v Plzni. Pro tyto studenty není v systému připraven žádný registrační proces a lokální registrátoři musejí jejich počítače registrovat manuálně.

[2]Formát vstupních dat formuláře pro úpravu oprávnění opět získá ze zdrojových kódů dostupných v univerzitní knihovně.



Obrázek 5.1: Současný proces registrace do kolejní sítě v notaci BPMN.

6 Řešení incidentů v síti WEBnet

Řešením bezpečnostních incidentů v síti WEBnet se zabývá tým WIRT (WEBnet Incident Response Team). Tým se zabývá nejen prošetřováním a řešením hlášených bezpečnostních incidentů, ale i aktivním vyhledáváním bezpečnostních incidentů v síti WEBnet a zaváděním preventivních opatření, aby k incidentům nedocházelo.

Tým má k dispozici nástroje pro sledování síťového provozu, které jim umožňují detekovat problémy v síti. Jedná se zejména o detekci:

- DoS (Denial of Service) či DDoS (Distributed DoS) útoků proti síti WEBnet, jejím součástem či uživatelům,
- šíření malware a škodlivého software v klientských stanicích,
- chyb v zabezpečení systémů a koncových stanic s přístupem do informačních systémů a
- porušení definovaných pravidel připojení, například další rozšiřování sítě a umožnění přístupu do sítě třetím osobám.

WIRT dále řeší stížnosti na protizákonnou aktivitu uživatelů sítě. Jedná se zejména o šíření nelegálního software a obsahu prostřednictvím P2P (Peer-to-Peer) sítí a používání nelicencovaného software.

V kontextu kolejní sítě se nejčastěji setkáváme s šířením škodlivého kódu, porušováním definovaných pravidel připojení (zejména pak vytváření bezdrátových sítí) a řešením stížností na šíření nelegálního obsahu prostřednictvím P2P sítí. Na úrovni jednotlivých kolejí tyto problémy pomáhá odhalovat a řešit také studentská samospráva.

Při řešení incidentů v síti WEBnet používá tým WIRT vlastní nástroj Mysphere2.

6.1 Portál Mysphere2

Mysphere2 je webová aplikace vytvořená v rámci projektu „Zkvalitnění procesu řešení bezpečnostních incidentů v síti WEBnet“ fondu rozvoje CESNET v průběhu roku 2011. Hlavním úkolem této aplikace je administrace a automatizace procesu řešení bezpečnostních incidentů. [5]

Tato aplikace usnadňuje a automatizuje celou řadu úkonů, které jsou spojeny s procesem řešení bezpečnostních incidentů, které dříve musely být prováděny manuálně. Jde zejména o:

- vyhledávání informací o identitě uživatele v síti WEBnet,
- automatické rozesílání notifikací uživatelům,
- řízení karantény v univerzitní síti,
- blokování identit v síti Eduroam a
- automatizace úkonů v interních agendách sítě WEBnet.

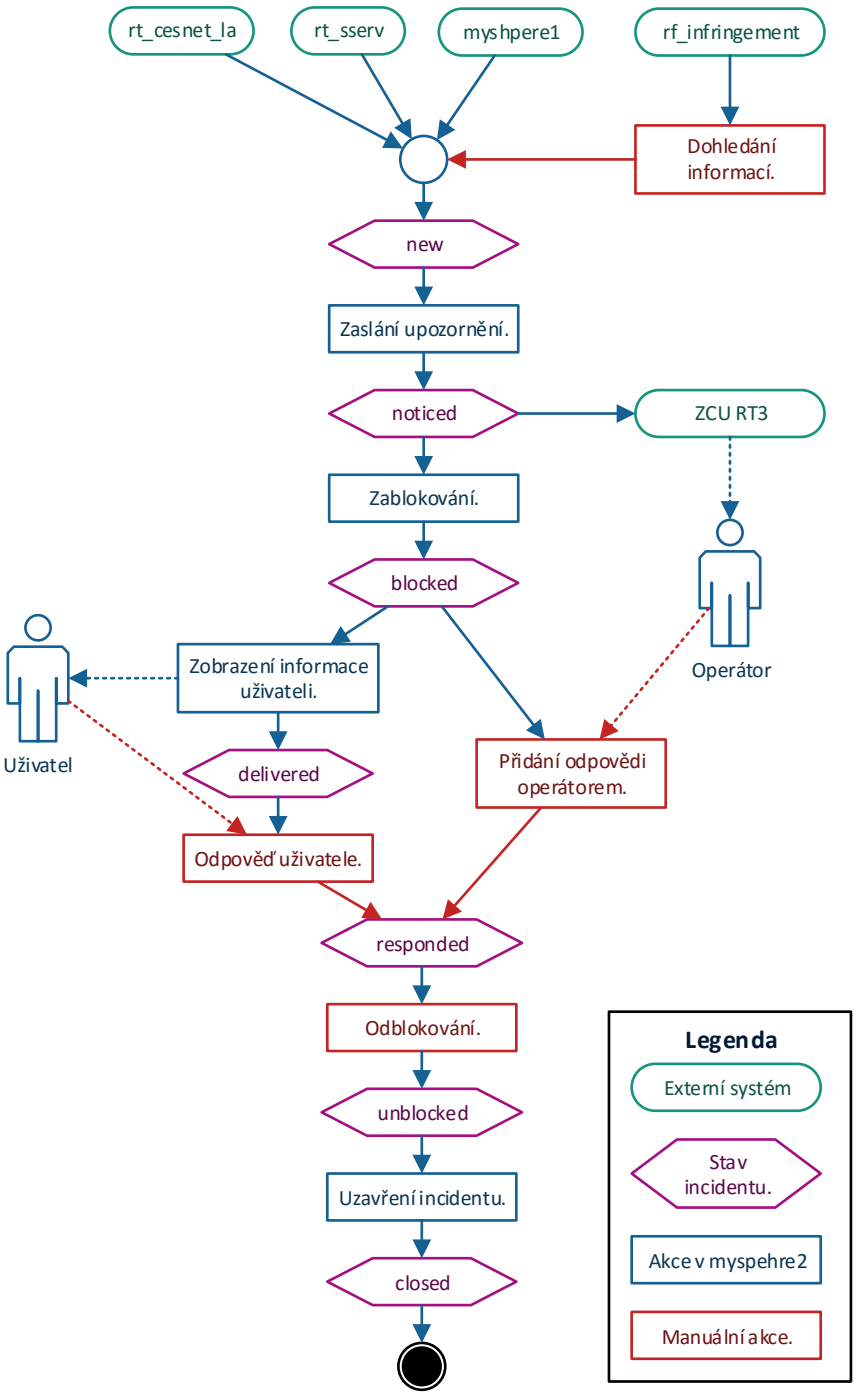
Aplikace obsahuje interní databázi incidentů a umožňuje administrátorovi řídit životní cyklus incidentu. Aplikace je schopná provádět potřebné úkony v závislostech na části sítě, ve které byl incident zaznamenán – například v síti Eduroam je nutné najít a zablokovat uživatele dle informací v RADIUS serveru.

Mysphere2 také zajišťuje interakci s uživatelem. Zablokovaným uživatelům je zobrazena informační stránka s důvodem jejich zablokování, uvedením dalšího postupu a kontaktním formulářem. Celý proces řešení incidentu je znázorněn na obrázku 6.1.

Celý průběh řešení incidentů, včetně reakcí zablokovaného uživatele, je zároveň evidován v systému RT. Mysphere2 je zároveň integrováno s interním nástrojem NetSpy, který automatizuje konfigurační zásahy v síti. S tímto nástrojem komunikuje pomocí HTTP REST API.

Aplikace je implementována v jazyce PHP s použitím MVC frameworku CakePHP.

V době psaní tohoto textu jsou připravovány zásadní úpravy software Mysphere2. S velkým rozvojem univerzitní sítě, zejména pak díky výstavbě nových výzkumných ústavů a velkému navýšení počtu prvků v síti WEBnet, rostou nároky na tento nástroj. Stávající řešení již není dostatečně flexibilní pro potřeby týmu WIRT. Předpokládá se, že realizace těchto změn začne v průběhu léta 2016.



Obrázek 6.1: Proces řešení incidentu v nástroji Mysphere2. Vytvořeno na základě [5].

7 Návrh řešení

Stávající řešení registračního portálu je již zastaralé. Používá knihovny, které již nejsou vyvíjeny a podporovány. Řešení také obsahuje bezpečnostní chyby a pokud by došlo k aktualizaci běhového prostředí na aktuálně podporovanou verzi[1], otevřela by se v systému závažná chyba SQL injection.

Nalezená open-source řešení pro řízení přístupu do sítě využívají ověřování uživatelů protokolem 802.1X. Ten však není z důvodu omezené podpory ve stávající infrastruktuře vhodný pro nasazení, neboť by buď bylo nutné použít ověřovací režim Multi-Host, ve kterém se na jednom portu přístupového přepínače ověřuje pouze první připojené zařízení, nebo režim Multi-Auth, ve kterém se sice ověřují všechna připojená zařízení, ale není dostupná funkcionalita Guest VLAN. Ta je potřebná pro poskytnutí automatické konfigurace klientům s vypnutým ověřováním protokolem 802.1X po drátových sítích.

Na základě těchto informací bylo rozhodnuto o přepsání celé aplikace webového portálu do aktuálně podporovaných technologií. V rámci úprav také dojde k úpravám v realizaci registračního procesu, které celý proces pro uživatele zjednoduší a zpřehlední.

Hlavním podkladem pro návrhy na vylepšení je pětiletá zkušenost s fungováním kolejní sítě z pohledu lokálního správce. Při sestavování požadavků na nové řešení byl také lokálním správcům rozeslán dotazník, jehož cílem bylo zjistit jejich postřehy z fungování registračního systému a kolejní sítě obecně. Výsledky byly následně osobně konzultovány s ostatními správci.

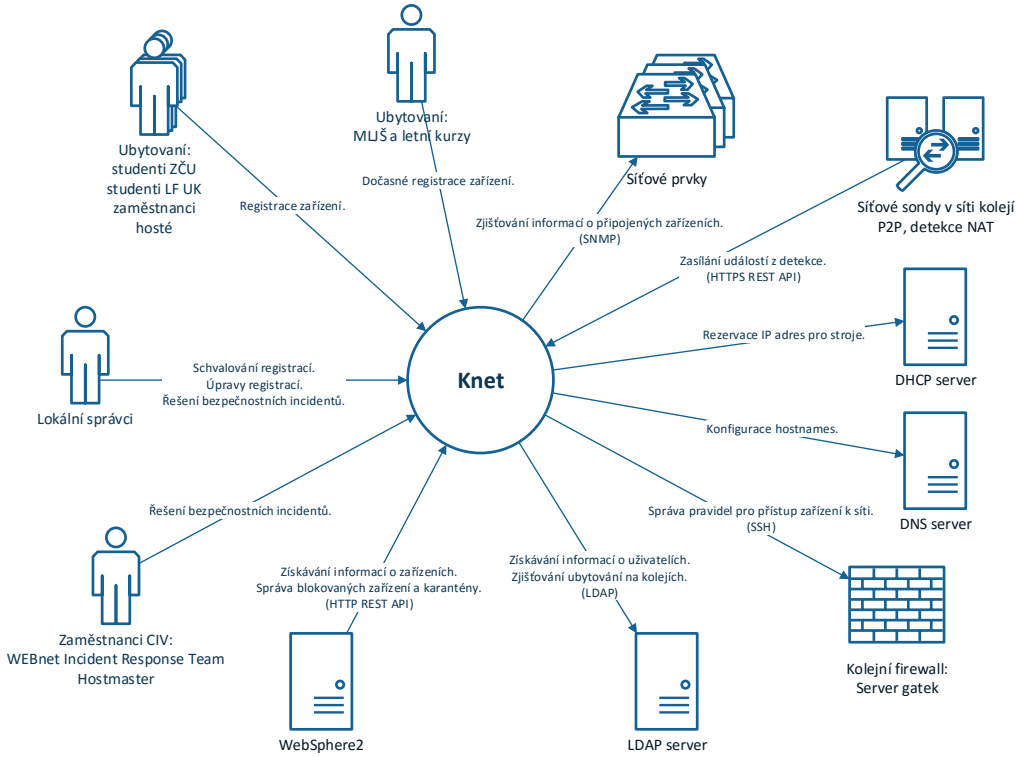
7.1 Kontext systému

Celé řešení pro řízení přístupu do kolejní sítě je komplexní systém. Se systémem pracují různé skupiny uživatelů: ubytovaní na kolejích (studenti ZČU, studenti ostatních univerzit a hosté ubytovaní v rámci letních akcí univerzity – Mezinárodní Letní Jazyková Škola (MLJŠ) a ArtCamp), lokální správci z řad studentů a zaměstnanci CIV (Centrum informatizace a výpočetní techniky): členové skupiny hostmaster a tým WIRT.

Systém zároveň musí interagovat s ostatními systémy: načítat informace uživateli ze síťových prvků a z LDAP, konfigurovat služby kolejní sítě a ko-

[1]Podpora pro PHP 5.3 skončila 14. srpna 2014. Bezpečnostní aktualizace již nejsou vydávány. Přehled verzí s ukončenou podporou je k dispozici na adrese `http://php.net/eol.php`

munikovat se systémy pro řešení bezpečnostních incidentů. Kontext celého systému je znázorněn na obrázku 7.1.



Obrázek 7.1: Kontext registračního systému Knet.

7.2 Časté problémy uživatelů při registraci

Při registraci se zejména méně technicky zdatní uživatelé potýkají s různými problémy.

V původní verzi registračního systému musel uživatel vyplnit umístění (kolej, pokoj a zásuvka) a hostname registrovaného počítače. Mnoho uživatelů nevědělo, co přesně mají vyplnit do položky hostname, proto bylo do systému doplněno předvyplnění hostname na základě Orion účtu.

Dalším problémem při vyplňování registračního formuláře je chyba při výběru zásuvky, neboť zásuvky jsou na pokojích umístěny za skříněmi či dalším nábytkem, kde jsou obtížně přístupné. Uživatel tak nevidí označení na zásuvce a mnohdy vybere zásuvku, která je připojena do jiné podsítě, než kam je uživatel skutečně připojen. K tomu dochází zejména na pokojích, kde jsou dostupné tři či více zásuvek.

Možným řešením tohoto problému je zavedení automatické detekce zásuvky, ke které je uživatel připojen. Ta by pak byla ve formuláři předvyplněna. Pokud by byla zavedena automatická detekce umístění uživatele, bylo by možné celý registrační formulář nahradit za pouhou výzvu ke kontrole detekovaných údajů a „jednotlačítkovou“ registraci.

Po potvrzení registrace je uživatel přesměrován na výpis počítačů. Nezobrazí se mu žádné potvrzení, že jeho registrace byla uložena, ani další kroky, které by měl provést. O tom, že po uložení registrace musí navštívit některého z lokálních registrátorů, se tak dozvídá pouze z instrukcí umístěných na kolejních nástěnkách a chodbách, případně od svých spolubydlících, co již mají s registrací zkušenosti.

Po úspěšném potvrzení registrace je vhodné uživateli nejen zobrazit potvrzení, že jeho registrace byla uložena, ale také mu poskytnout instrukce o dalších krocích, které má provést.

Současný systém umožňuje registraci pouze jednoho uživatele na zásuvce. Pokud se tedy uživatel do pokoje přistěhuje po začátku roku a předchozí obyvatel pokoje nezrušil svoji registraci, systém uživateli zásuvku nenabídne k výběru. To vede ke komplikaci celého procesu: uživatel v tomto případě musí navštívit správce, který předchozí registrace smaže, teprve poté uživatel může provést registraci svého počítače.

Toto opatření přináší zbytečnou komplikaci. Kontrolu, zda na dané zásuvce není registrovaný jiný uživatel, může provést správce při schvalování registrace. Systém může registrace na stejné zásuvce automaticky vyhledat a v případě, že budou nějaké registrace nalezeny, mohou být správci nabídnuty ke smazání či úpravě.

Další problém nastává při stěhování uživatelů. Pokud před stěhováním uživatel nepožádá správce o smazání či úpravu registrace, není mu po připojení zařízení do jiné podsítě přidělena adresa z DHCP serveru. Použitý DHCP server (Internet Standards Consortium (ISC) DHCP[2]) nepřidělí IP adresu, pokud koncová stanice žádá z jiné podsítě, než ve které má vytvořenou registraci. Uživatel tak opět musí vyhledat správce, aby upravil jeho starou registraci.

Řešením tohoto problému by bylo použít jiný DHCP server, který umožní registrovat jednu MAC adresu do více podsítí, a pokud záznam pro danou koncovou stanici nalezne v jiné podsíti, přidělí jí IP adresu dynamicky podobně jako v případě neregistrovaných stanic.

Protože po vytvoření a schválení je až 10 minut prodleva, než se obnoví konfigurace DHCP serveru a firewallu, mohou si někteří uživatelé myslet, že

[2]https://www.isc.org/downloads/dhcp/

registraci provedli chybně. Nové řešení by tedy mělo prodlevu mezi schválením a obnovením konfigurace služeb minimalizovat.

7.3 Integrace s dalšími systémy

7.3.1 Integrace s Mysphere2

Systém Mysphere2 řídí celý proces řešení bezpečnostního incidentu. V průběhu životního cyklu incidentu je potřeba vyhledat informace o uživateli, který je původcem daného incidentu, uživatele kontaktovat a následně mu zablokovat přístup do sítě, aby se předešlo dalšímu průběhu nebezpečné činnosti (například dalšímu šíření škodlivého software aj.).

V současném systému musí člen týmu WIRT ručně najít informace o uživateli a poté jej ručně zablokovat. V rámci integrace je potřeba připravit rozhraní, pomocí kterého může systém Mysphere2 najít identitu uživatele a následně jej zablokovat. Při vyhledávání informací o identitě je systému známá typicky IP adresa zařízení, které je původcem incidentu.

Hlavní linie řešení incidentu (bez vazeb na další externí systémy a komunikace s uživatelem) a navržená interakce mezi systémem Mysphere2 a Knet je znázorněna na obrázku 7.2.

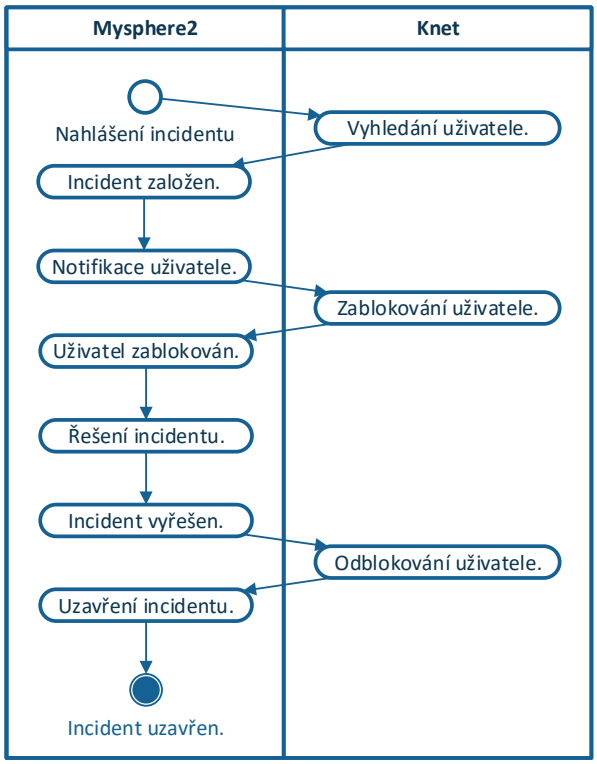
Pro implementaci API pro vyhledávání uživatelů a správu incidentů je vhodná technologie HTTP REST API. Systém Mysphere2 tuto technologii již používá pro komunikaci s nástrojem NetSpy, navíc je implementace klientské knihovny snadná ve všech vyšších programovacích jazycích.

Jako formát pro kódování dat se v REST API nejčastěji používá XML (eXtensible Markup Language) a JSON (JavaScript Object Notation). Pro implementaci komunikace byl zvolen JSON, který je „stručnější“: reprezentace dat ve formátu JSON je typicky menší, než reprezentace stejných dat ve formátu XML.

Pro ověřování vzdálených klientů můžeme použít několik metod:

Ověřování API klíčem v hlavičkách požadavku: klíč musí být přenášen v každém požadavku. Pokud dojde ke kompromitaci spojení či úniku tajného klíče, má potenciální útočník časově neomezený přístup k API.

Protokol OAuth2: aplikace si nejprve vyžádá přístupový klíč s časově omezenou platností, ten použije v dalších voláních. Útočník by musel zachytit data původního požadavku, přístupové klíče z ostatních volání API mu poskytnou jen časově omezený přístup.



Obrázek 7.2: Průběh řešení incidentu v systému Mysphere2 a interakce se systémem Knet.

Ověřování klienta pomocí SSL (Secure Sockets Layer) certifikátů: nepřenáší se žádné přístupové kódy, identitu klienta ověřuje přímo HTTP server na základě poskytnutého certifikátu. Nevýhodou tohoto přístupu je nutnost zavést dodatečné postupy a procesy pro vytváření, ukládání, správu a zneplatňování certifikátů. V tomto případě může být pro správce a registrátory obtížné použít API z prostředí webového prohlížeče.

Pro API registračního systému bylo zvoleno použití OAuth2. Při použití šifrovaných spojení je toto řešení bezpečné, API pak navíc bude možné snadno použít z prohlížeče a pro běžně používané vysokoúrovňové jazyky jsou dostupné knihovny, které provedou ověření klientské aplikace a získání přístupového klíče.

Protože aplikace Mysphere2 projde v létě 2016 zásadními úpravami, bylo po dohodě s týmem WIRT rozhodnuto o realizaci integrace mezi těmito systémy až v její nové verzi. Na straně systému Knet bude připraveno API a bude připravena klientská knihovna, která aplikaci Mysphere2 umožní komunikovat s novým registračním systémem.

Funkčnost API bude ověřena na integraci síťové sondy, která běží v prostředí kolejní sítě a jejíž požadavky na komunikaci se systémem jsou podobné jako u aplikace Mysphere2.

7.3.2 Integrace se síťovými sondami

Lokální správci provozují v kolejní síti vlastní sondy, které umožňují detekovat obcházení definovaných pravidel sítě a používání P2P protokolů. Na kolejích Máchova, Baarova a Klatovská jsou provozovány následující dvě sondy:

Natdet: detekce zařízení, které používají NAT. Používá se pro odhalování rozšiřování kolejní sítě pomocí bezdrátových směrovačů, vytváření Wi-Fi hotspotů a dalších metod připojování zařízení, která nejsou registrována v kolejní síti. K sondě aktuálně neexistuje žádné grafické rozhraní a její záznamy jsou analyzovány ručně.

Detekce P2P spojení: sonda využívá projekt ipp2p[3], který do linuxového netfilter přidává detekci P2P protokolů.

Pro integraci byla vybrána sonda pro detekci P2P spojení. Tato sonda je využívána všemi lokálními správci na kolejích a incidenty související se sdílením autorského obsahu prostřednictvím P2P sítí jsou častější.

Stávající webové rozhraní pouze zobrazuje posledních 500 záznamů ze souboru s logem ze sondy ipp2p. Tuto webovou stránku je potřeba neustále obnovovat. Rozhraní zobrazuje pouze informace o spojení a získává název počítače ze systému DNS. Správce tak musí ručně dohledávat další informace o uživateli.

Výsledky jsou navíc řazeny pouze chronologicky a nejsou žádným způsobem seskupovány, každý řádek je pouze podbarven podle lokální IP adresy v síti tak, aby spojení stejného počítače měla stejnou barvu. Pokud tedy probíhá zároveň více P2P komunikací, musí správce ručně procházet celý záznam a hledat řádky se stejnou barvou.

V rámci diplomové práce tedy bude vytvořeno nové rozhraní, které záznamy zpřehlední a usnadní lokálním správcům dohledávání informací o zařízeních.

Integrace s API registračního systému umožní sondě efektivně získávat informace o koncových stanicích. Umožní také v případě překročení nastavených limitů automaticky založit incident v registračním systému, upozornit uživatele a případně jej zablokovat.

[3] http://xtables-addons.sourceforge.net/

Pokud správce provede změnu automaticky založeného incidentu, bylo by vhodné změnu zobrazit i ve webovém rozhraní sondy. Data lze do webového rozhraní získávat dvěma hlavními způsoby:

- Periodická kontrola změn – sonda bude pomocí API periodicky kontrolovat změny a stahovat nové komentáře k incidentům.
- API, prostřednictvím kterého bude registrační systém zasílat změny incidentů do webového rozhraní.

Protože ke změnám v incidentech nebude docházet často, byla zvolena druhá varianta, kdy registrační systém Knet zasílá změny zpět do webového rozhraní sondy. Díky tomu správci také uvidí změny incidentů v reálném čase.

Pro zasílání změn je možné implementovat podobné API jako v případě hlavního registračního systému, nebo zvolit jednodušší variantu. Tou jsou například tzv. webhooks – při vyvolání určité události (změna incidentu, přidání komentáře) systém zašle na uvedenou adresu POST požadavek, v jehož těle budou uvedena aktualizovaná data. Pro ověření, že zprávu zaslal skutečně zdrojový systém, je možné použít autorizační klíč, nebo podepisování algoritmem HMAC.

7.4 Přístup k síťovým prvkům

Ze síťových prvků potřebuje registrační systém získávat následující informace:

- MAC adresa počítače, ze kterého uživatel přistupuje k registračnímu systému,
- port, ke kterému je uživatel připojen, a
- seznam aktivních počítačů v síti.

Uvedené informace je možné získat zasíláním vhodných příkazů pomocí protokolu SSH či Telnet, nebo protokolem SNMP. Protože jednotlivé prvky mohou mít různá rozhraní příkazové řádky, je vhodnější použít protokol SNMP, kde je přístup ke informacím standardizovaný v MIB tabulkách.

Pokud má přepínač či směrovač společnosti Cisco nakonfigurovaných více VLAN, poskytují některé tabulky informace vždy z jedné VLAN. Způsob, jakým v požadavku určit VLAN, jejíž tabulka má být prohledávána, je závislý na verzi protokolu.

Při použití protokolu SNMP v1 a v2 je nutné použít tzv. community indexing. Za název komunity, která se používá pro přístup k prvku, se připojí znak @ následovaný indexem VLAN. Pokud tedy chceme prohledávat tabulku

pro VLAN 75 a název komunity je *TestROAccess*, použije se pro dotazování komunita *TestROAccess@75*.

V případě protokolu SNMP v3 je nutné v konfiguraci prvku udělit uživateli oprávnění pro přístup ke kontextu ve tvaru `vlan-<id>`, kde `<id>` je číslo VLAN. Tento kontext je pak použit při dotazování.

Je možné také uživateli přidělit přístup ke kontextům všech VLAN příkazem[4]

```
snmp-server group <group> v3 auth context vlan- match prefix
```

Získání MAC adresy počítače

MAC adresu připojeného počítače je možné získat z ARP tabulky směrovače (či L3 přepínače) dané podsítě. Ta je dostupná přes SNMP celkem ve 3 standardních tabulkách. Nejdéle podporovanou možností je tabulka *atTable*[5], která je definována v MIB-1. V nové verzi MIB-2 je tato tabulka však již označena za zastaralou a nahradila jí tabulka *ipNetToMediaTable*[6]. Poslední možností je tabulka *ipNetToPhysicalTable*[7], jež obsahuje i nejvíce informací, včetně časového razítka, kdy byl záznam naposledy aktualizován. Tato tabulka však není podporována prvky, které jsou použity v kolejní síti. Nejvhodnější je tedy použít tabulku *ipNetToMediaTable*.

Tabulka *ipNetToMediaTable* obsahuje sloupce s ID rozhraní (na zařízeních Cisco i zároveň číslo VLAN), fyzickou adresou, IP adresou a typem záznamu. Klíčem do této tabulky je číslo VLAN a IP adresa koncové stanice. Pokud tedy chceme získat MAC adresu zařízení s IP adresou 10.10.75.17 ve VLAN 75, je potřeba ze SNMP přečíst záznam s OID 1.3.6.1.2.1.4.22.1.2.75.10.10.75.17.

Tabulka *ipNetToMediaTable* obsahuje informace ze všech VLAN.

Získání portu, ke kterému je uživatel připojen

Fyzický port, ke kterému je uživatel připojen, je možné získat z přístupových prvků, konkrétně z tabulky *dot1dTpFdbTable*[8]. Tato tabulka obsahuje sloupce s fyzickou adresou, identifikátorem fyzického portu a stavem záznamu. Stav identifikuje, zda jde o již neplatný záznam, platný záznam, který se zařízení naučilo, nebo záznam zadaný administrativně.

[4]Dostupnost tohoto příkazu je závislá na použitém prvku a verzi firmware. Na prvcích v kolejní síti s aktuální verzí firmware je tento příkaz podporován.

[5]OID 1.3.6.1.2.1.3.1

[6]OID 1.3.6.1.2.1.4.22

[7]OID 1.3.6.1.2.1.4.35

[8]OID 1.3.6.1.2.1.17.4.3

Pro nalezení portu, na kterém je dostupné zařízení s danou MAC adresou, je postačující se dotazovat do sloupce s identifikátorem portu. Klíčem do tabulky je MAC adresa zapsána po bajtech v desítkové soustavě. Pro získání ID portu, ke kterému je připojeno zařízení s MAC adresou 40:8D:5C:26:80:A3, je tedy potřeba se dotázat na objekt s OID:

1.3.6.1.2.1.17.4.3.1.2.**64.141.92.38.128.163**

Hodnoty v této tabulce jsou vázány na konkrétní VLAN, je tedy potřeba použít community indexing pro SNMP v1 a v2, nebo uvést kontext pro SNMP v3.

Po získání identifikátoru portu je potřeba zjistit jeho název. To můžeme provést dotazem do tabulky *dot1dBasePortTable*[9], ze které získáme index do tabulky *ifXTable*[10]. Poslední jmenovaná tabulka obsahuje informace o portu, včetně krátkého názvu portu v zařízení, například „Fa0/15“. Indexy v této tabulce se mohou měnit po restartu zařízení, pokud nebyla povolena vlastnost IfIndex Persistence[11].

Protože v kolejní síti je jedna podsíť (VLAN) nakonfigurována na více prvcích, poskytne nám jen jeden z nich informaci o tom, kde je uživatel připojen. Ostatní prvky zapojené do stejné podsítě buď nebudou mít pro dané zařízení žádný záznam, nebo budou mít záznam uvedený u portu, který je použit pro propojení jednotlivých prvků.

Pro identifikaci správného portu, na kterém je uživatel připojen, můžeme použít dvě varianty:

1. Stromově prohledávat síť. Jako první bude položen dotaz na hraniční směrovač, který oznámí číslo portu, ke kterému je připojený prvek s připojeným zařízením. Tohoto zařízení se pak můžeme zeptat dále, hledaná koncová stanice bude připojena buď přímo k němu, nebo opět na některém dalším prvku.
   Tento přístup však vyžaduje buď předchozí znalost sítě, kterou je nutno udržovat, nebo se na adresu připojených prvků dynamicky dotazovat do tabulek CDP (Cisco Discovery Protocol). Aby nebylo potřeba se na informace o spojích mezi prvky ptát při každém vyhledávání MAC adresy, je vhodné načtené spoje ukládat do paměťové cache.
2. Dotázat se všech přístupových prvků dané podsítě a vybrat z nich takový port, který je v databázi systému uveden jako port přístupový.

Druhé zmíněné řešení je univerzální a lze jej použít s prvky různých výrobců, které nepodporují protokol CDP. První zmíněné řešení však vyžaduje méně

[9] OID 1.3.6.1.2.1.17.1.4
[10] OID 1.3.6.1.2.1.31.1.1
[11] Příkaz `snmp-server ifindex persist`.

dotazů do síťových prvků, zvláště v případě rozlehlých sítí a použití paměťové cache.

Do výsledného řešení bude vhodné implementovat obě varianty. Pro současnou infrastrukturu postavenou na síťových prvcích od společnosti Cisco pak bude použita první varianta, kdyby v budoucnu byly přidány další prvky, které protokol CDP nepodporují, může být použita varianta druhá.

Získání aktivních MAC adres

Seznam aktivních MAC adres je možné získat obdobně jako při překladu IP adresy na MAC adresu. Systém získá data z tabulky *ipNetToMediaTable* hraničního směrovače sítě, avšak místo dotazu na konkrétní záznam podle jeho klíče provede operaci *Walk* a zpracuje celou tabulku.

7.5 Konfigurace síťových služeb

Pro povolení přístupu do vnější sítě nově registrované stanici je potřeba přidat záznamy do DHCP a povolit přístup na hraničním firewallu. Při registraci stanice je také vhodné přidat A a PTR záznam do DNS serveru, aby bylo možné se na stanici odkazovat jejím názvem v síti.

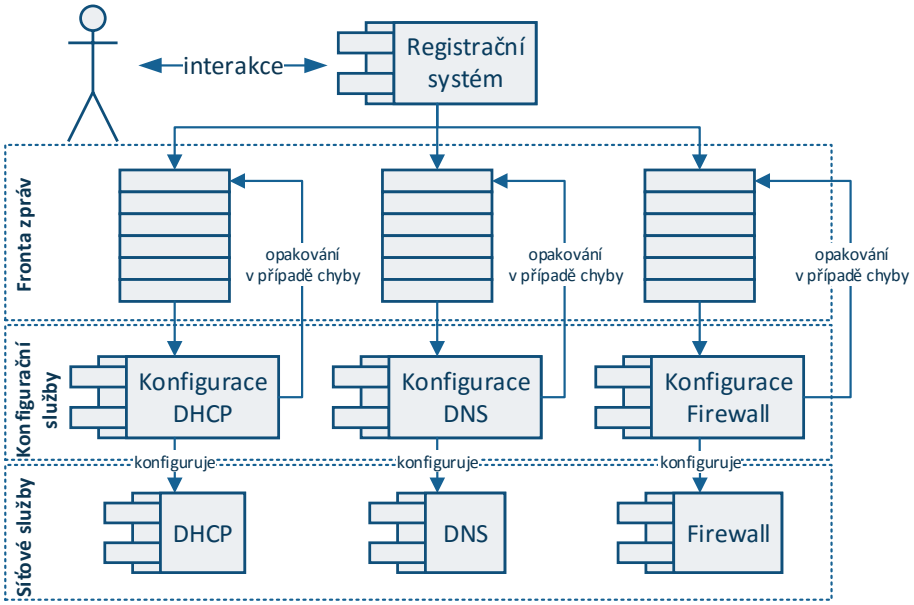
Při hledání nového řešení bylo hlavním cílem urychlení odezvy. Místo periodického obnovování konfigurace je vhodnější novou konfiguraci ihned aplikovat, abychom eliminovali prodlevu mezi schválením registrace a skutečným povolením přístupu.

Vhodným řešením je použít software, který své záznamy ukládá do databáze, nebo který poskytuje konfigurační API. Aktualizace nastavení pak může probíhat ihned poté, co dojde k příslušné události.

Konfigurační změny je možné provádět přímo při úpravách ve webové aplikaci. Nová registrace by se tak například uložila nejen do databáze registračního systému, ale i do databáze DHCP a DNS serveru. To však s sebou nese problémy spojené s koordinací transakcí mezi více databázovými servery. Navíc v případě, že bude například databáze některého serveru dočasně nedostupná (například kvůli její údržbě), akce uživatele selže a bude ji muset opakovat později. Řešením je neprovádět konfigurační změny přímo, ale vkládat požadavky na změnu konfigurace do fronty zpráv a implementovat tak architekturu producent – konzument. Pokud požadavek selže, je možné jej automaticky opakovat později bez interakce s uživatelem.

Výběr požadavků z databáze a samotné konfigurační změny pak budou provádět specializované skripty, které poběží jako služby. Každá konfigurační služba bude zajišťovat konfiguraci jedné síťové služby (DHCP, DNS, firewall),

selhání jednoho skriptu tak neovlivní konfiguraci ostatních služeb. Zároveň bude možné službu snadno upravit či nahradit za jinou. Interakce jednotlivých komponent je znázorněna na obrázku 7.3.



Obrázek 7.3: Interakce mezi komponentami pro konfiguraci síťových služeb.

7.5.1 Fronta zpráv

Pro implementaci fronty zpráv existuje celá řada hotových řešení, komerčních i volně dostupných. Software, který zajišťuje ukládání a distribuci zpráv, se nazývá message broker. V následujícím textu bude uveden stručný přehled vybraných open-source řešení.

U řešení bylo zkoumáno:

- pro jaké jazyky jsou dostupné knihovny pro práci s frontou z aplikací,
- jaké komunikační protokoly jsou podporovány,
- jaká jsou praktická nasazení v produkčním prostředí a
- zda message broker podporuje plánované doručení zpráv, které je potřebné pro opakování zprávy v případě chyby při konfiguraci cílové služby.

RabbitMQ

RabbitMQ[12] je message broker implementovaný ve funkcionálním jazyce Erlang. Komunikace probíhá pomocí TCP spojení univerzálním protokolem AMQP (Advanced Message Queue Protocol), klientské knihovny jsou dostupné pro jazyky Java, PHP, Python, C, C++, C# a další.

RabbitMQ lze rozšířit pomocí pluginů. Ve formě rozšíření je dostupné například webové rozhraní pro správu, podpora pro odložené doručování zpráv a rozšíření pro podporu doplňujících protokolů (JSON-RPC, SMTP a další). Pomocí odloženého doručování je možné naplánovat opakované doručení zprávy v případě chyby.

Vývoj zajišťuje společnost Pivotal ve spolupráci s aktivní komunitou na serveru GitHub. Mezi významné uživatele patří například společnosti VMware, Mozilla, Unique Identification Authority of India či AT&T. [4]

ZeroMQ

ZeroMQ[13] je message broker implementovaný v jazyce C++. Komunikace probíhá vlastním protokolem ZeroMQ. Klientské knihovny jsou dostupné pro jazyky Python, Java, PHP, C, C++, C# a další. Vývoj zajišťuje společnost iMatix.

ZeroMQ nepodporuje plánované doručení zprávy.

Na webu projektu je uveden seznam společností, které produkt využívají, avšak nejsou poskytnuty žádné podrobnosti o konkrétním nasazení, ani zda byla technologie použita v produkčním prostředí.

NSQ

NSQ[14] je message broker, jehož hlavním cílem je poskytnout prostředky pro vybudování distribuované a decentralizované topologie bez jediného bodu selhání (single point of failure). Fronta je implementována v jazyce Go, dostupné jsou oficiální klientské knihovny pro jazyky Python, Go, JavaScript a C. K dispozici jsou i knihovny pro jazyky Java, PHP, C#, Ruby. Komunitou projektu jsou vyvíjeny i knihovny pro další jazyky.

NSQ podporuje odložené opětovné doručení v případě chyby.

Vývoj původně zajišťovala společnost Bitly, nyní je projekt vyvíjen ve spolupráci s dalšími společnostmi. Web projektu uvádí, že je v produkci nasazen u celé řady společností, například MOZ, digg, stripe a další.

[12] https://www.rabbitmq.com/
[13] http://zeromq.org/
[14] http://nsq.io/

Návrh řešení

Uvedená řešení vyžadují, aby zprávy do fronty vkládala přímo webová aplikace. Pokud by změny byly prováděny přímo v databázi (typicky jde o hromadné operace s registracemi), nepromítly by se tyto změny do konfigurace služeb. Navíc jde o další technologii, která by byla nasazena na serveru s registračním systémem a kterou by bylo potřeba udržovat a aktualizovat.

Zprávy je možné ukládat v databázi podobně jako data aplikace. Využitím prostředků pro programování na straně databáze je tak možné zajistit, aby se při provedení operace vložily příslušné zprávy do fronty.

Konzument, tedy konfigurační služba, pak musí zprávu vyzvednout. Pro výběr zprávy je opět možné použít několik přístupů:

- Periodicky kontrolovat dostupnost zpráv ve frontě. Konzument by kontroloval, zda v jeho frontě čekají zprávy ke zpracování. Pokud ano, zpracuje je. Po ukončení kontroly se konzument uspí na určitou dobou a poté opakuje kontrolu.
- Použít asynchronní notifikace databázového systému. Tuto možnost podporují pouze databáze PostgreSQL a Microsoft SQL Server. Konzument čeká na přijetí notifikace na pojmenovaném kanálu. Producent vloží zprávu a vyšle do příslušného pojmenovaného kanálu notifikaci, že jsou ve frontě dostupné zprávy. Databázový server notifikaci pošle všem připojeným klientům, kteří na daném kanálu naslouchají. Pro doručení notifikace se používá stejné spojení, které klient používá pro příjem notifikací.

Řešení s použitím notifikací je efektivnější a umožňuje zprávy zpracovat rychleji. Pro použití v kolejní síti je vhodná databáze PostgreSQL, neboť je možné ji provozovat na systému OS Linux, který využívají všechny ostatní služby kolejní sítě.

PostgreSQL implementuje výměnu zpráv pomocí příkazů `LISTEN` a `NOTIFY`. Klient, který si přeje přijímat zprávy, zavolá příkaz `LISTEN <channel>`, kde `<channel>` je název kanálu, který má být použit pro interakci. Kanál není nutné explicitně vytvářet, je vytvořen automaticky příkazem `LISTEN`.

Příjem zpráv je asynchronní: po zavolání příkazu `LISTEN` není klient zablokován a nejsou kladena žádná omezení na práci s databázovým systémem, klient může serveru zaslat libovolné příkazy.

Klient však musí zajistit, aby zprávu ve vhodný okamžik vybral a zpracoval. Může buď periodicky kontrolovat, zda spojení s databázovým serverem obsahuje data ke čtení (polling), nebo je možné použít mechanismy operačního systému, které umožňují klientovi pasivně čekat na dostupnost dat ve

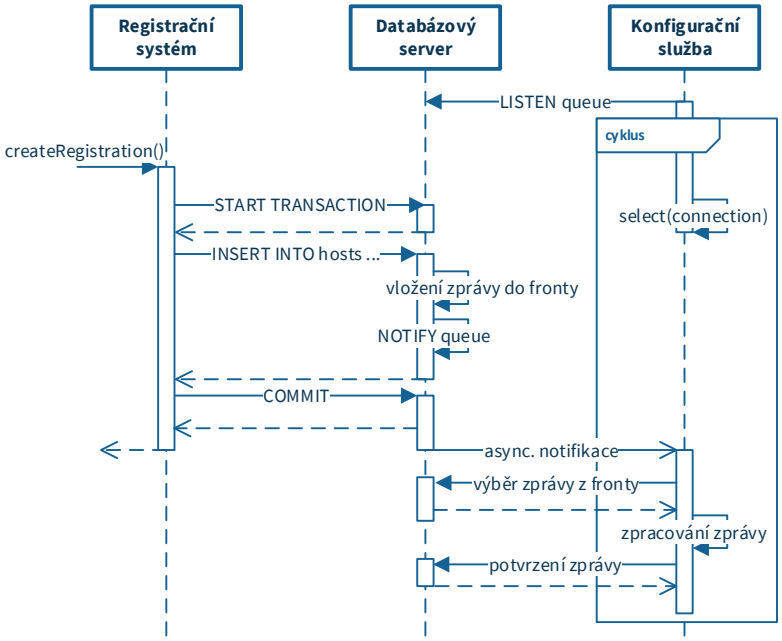
spojení. Takovým mechanismem je například systémové volání `select`, které klienta zablokuje do doby, než budou v komunikačním kanálu k dispozici data ke čtení.

Zprávy je možné do kanálu zaslat voláním `NOTIFY <channel>[, <payload>]`. `<payload>` je volitelný textový řetězec, který reprezentuje obsah zprávy. Zprávy nejsou ukládány – pokud na kanálu nenaslouchá žádný klient, zpráva bude zahozena.

Zasílání a příjem zpráv jsou omezeny transakcemi. Pokud některý z klientů zavolá `NOTIFY` a aktuálně se nachází v transakci, je zpráva doručena až po úspěšném potvrzení transakce. Pokud je transakce zrušena, je zrušena i zpráva zaslaná příkazem `NOTIFY`.

Pokud je některý z klientů, který dříve zavolal příkaz `LISTEN`, v době vyvolání zprávy v transakci, je mu zpráva doručena až po dokončení transakce. Je tedy nutné zajistit, aby klient čekající na zprávu nikdy nebyl uprostřed transakce. V databázi PostgreSQL je transakce implicitně zahájena libovolným příkazem, který čte nebo zapisuje data, včetně příkazu `SELECT`. Je proto nutné takto implicitně zahájenou transakci potvrzovat v případě, že bylo provedeno libovolné čtení dat.

Průběh zasílání zpráv a interakce mezi jednotlivými komponentami jsou znázorněny na obrázku 7.4.



Obrázek 7.4: Asynchronní zasílání zpráv pomocí databázového serveru PostgreSQL.

Doručení zprávy musí být spolehlivé – zpráva musí být vždy doručena příjemci a smazána může být až v případě, že byla úspěšně zpracována. Ze strany producenta garantuje vložení zprávy do fronty použití transakcí – pokud byla transakce úspěšně potvrzena, je zpráva uložena ve frontě.

Pokud konzument zprávu vybere z fronty, není garantováno, že ji úspěšně zpracuje. Kterákoliv část systému může selhat, proto musí být implementováno potvrzení, že byla zpráva zpracována. Pokud mezi zpracováním zpráv a úspěšným doručením potvrzení některá část systému selže, může být zpráva po obnovení funkce doručena ještě jednou.

Proto je nutné systém navrhnout tak, aby opětovné doručení zprávy bylo tolerováno a nedošlo například k duplicitnímu vložení záznamů, nebo k selhání zpracování kvůli duplicitě dat. Zprávy tedy poté nebudou mít význam příkazu, který má být proveden, ale budou nést stav, do kterého má být systém uveden. Konfigurační služba pak musí zajistit uvedení konfigurace do tohoto nového stavu. Zprávy tak budou idempotentní.

Při doručování může dojít k situaci, kdy by zprávy mohly být doručeny v jiném pořadí, než v jakém byly vloženy do fronty. Uvažujme následující situaci:

1. Uživatel zaregistruje zařízení, do konfigurační fronty se vloží zpráva pro založení záznamu v DHCP.
2. Konzument vybere zprávu, avšak kvůli dočasné nedostupnosti databáze DHCP její zpracování selže. Vloží ji zpět do fronty k pozdějšímu zpracování.
3. Během této doby uživatel zjistí, že registrace byla chybná a smaže ji. Do fronty se vloží příkaz na smazání záznamu v DHCP serveru.
4. Konzument zprávu vybere. Databáze je nyní již dostupná a zjistí, že zařízení v databázi neexistuje. To je požadovaný stav, proto zprávu potvrdí a odstraní ji tak z fronty.
5. Po uplynutí časového limitu opakuje pokus o vytvoření záznamu. Databáze je dostupná, a tak založí požadovaný záznam a zprávu potvrdí.

Z hlediska fronty zpráv jsou nyní všechny zprávy zpracovány, avšak systém není v očekávaném stavu. Je nutné zajistit konzistenci dat a pozastavit zpracování vkládaných zpráv v případě, že ve frontě čeká nezpracovaná zpráva, se kterou by vkládaná zpráva mohla kolidovat. Při potvrzení zprávy pak dojde k zadání další čekající zprávy v pořadí ke zpracování.

7.5.2 DHCP

Jak již bylo uvedeno v sekci 7.2 Časté problémy uživatelů při registraci, při výběru DHCP serveru bylo hledáno takové řešení, které bude mít možnost číst informace o registrovaných koncových stanicích z externí databáze bez nutnosti přenačítání celé konfigurace.

Zkoumána byla dvě řešení: v současnosti používaný server ISC DHCP a DHCP server Kea, který je vyvíjen stejnou společností.

ISC DHCP

ISC DHCP[15] server má možnost používat registrace zařízení načtená z konfiguračního souboru. Od verze 4.2.0 byl také do zdrojových kódů integrován patch, který umožňuje načítat registrace zařízení z LDAP serveru.

Tento DHCP server však má omezení, které neumožňuje přidělit IP adresu podle MAC adresy, která je nakonfigurovaná v jiné podsíti.

Kea DHCP

Kea DHCP[16] server je produkt vyvíjený společností ISC. Hlavní předností tohoto serveru je podpora IPv6, podpora ukládání rezervace zařízení v databázi a s ní související možnost změn bez nutnosti přenačítání konfigurace, API pro řízení serveru a sběr statistik.

Ve verzi 1.0 podporuje server ukládání rezervací IP adres pro zařízení pouze do MySQL databáze. Podpora databáze PostgreSQL je připravovaná ve verzi 1.1, jejíž vydání je plánováno na polovinu roku 2016.

Zvolené řešení

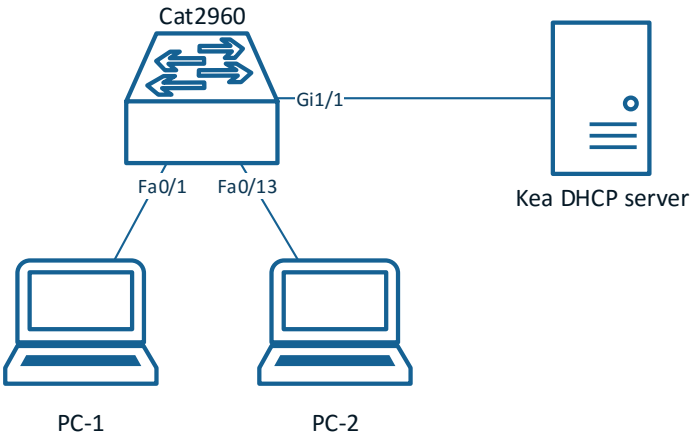
Pro nasazení do kolejní sítě byl vybrán DHCP server Kea. V jednoduchém laboratorním prostředí bylo otestováno, že server podporuje rezervaci jednoho zařízení do více podsítí. Pokud zařízení v podsítí nemá rezervaci, je dynamicky přidělena IP adresa z nakonfigurovaného rozsahu.

Bylo také otestováno, že server kontroluje údaje v databázi při každém požadavku klienta, pokud se tedy pro danou MAC adresu vytvoří či změní rezervace IP adresy, projeví se změna ihned při dalším pokusu o její prodloužení.

Laboratorní prostředí se skládalo z jednoho přepínače Cisco Catalyst 2960, který podporuje funkci DHCP Relay Agent. Na něm byly nakonfigurovány 3 VLAN: 1, 10 a 20. Porty 1- 12 byly zapojeny do VLAN 10, porty 13 - 24

[15] https://www.isc.org/downloads/dhcp/
[16] http://kea.isc.org/wiki

do VLAN 20. Zbytek portů do VLAN 1. Ve VLAN 10 a 20 byl nastaven DHCP Relay Agent, který DHCP požadavky předával na DHCP Kea server připojený do VLAN 1. Schéma prostředí je znázorněno na obr. 7.5.



Obrázek 7.5: Laboratorní prostředí pro otestování DHCP serveru Kea.

Nevýhodou tohoto serveru je konfigurace délky zapůjčení IP adresy pouze na úrovni podsítě, nelze ji nastavit na úrovni rezervace IP adresy či konfigurovat na základě dalších pravidel. Není tedy možné v základní verzi nastavit, aby dynamicky přidělené IP adresy neregistrovaným zařízením měly kratší platnost, než IP adresy přidělené na základě rezervací.

Server však obsahuje rozhraní pro jeho rozšiřování v jazyce C++. Je tak možné ovlivnit délku propůjčení IP adresy na základě vlastních pravidel, která budou implementována jako rozšíření serveru.

7.5.3 DNS

I v případě výběru DNS serveru bylo hledáno řešení, které umožní načítat záznamy z databáze bez nutnosti opětovného načítání konfigurace ze souborů. V tomto případě však není načítání z databáze nutností, neboť chybějící záznam stanice v DNS neovlivní její připojení do vnější sítě.

Záznamy je možné aktualizovat i pomocí protokolu DDNS (Dynamic DNS). Při jeho použití však nelze najít předchozí záznamy stanice na základě znalosti jeho identifikátoru, je tak nutné znát jeho předchozí doménové jméno a IP adresu. To zkomplikuje proces aktualizace záznamů.

ISC BIND

BIND[17] (Berkley Internet Name Domain) je nejrozšířenější DNS server. Může sloužit jako caching (rekurzivní) server i jako autoritativní server. V aktuální verzi nemá možnost načítat DNS záznamy z databáze. BIND podporuje protokol DDNS.

PowerDNS

PowerDNS[18] je rozdělen do dvou částí: autoritativní server a resolver – caching server. Tyto části jsou na sobě nezávislé.

Autoritativní server má možnost načítat DNS záznamy z různých zdrojů: databáze (podporovány jsou MySQL, PostgreSQL, Oracle a další), konfigurační soubory pro Bind, LDAP a další. Kompletní seznam podporovaných zdrojů záznamů je uveden v oficiální dokumentaci. U autoritativního serveru je také možné nastavit adresu serveru, na který bude zasílat dotazy pro rekurzivní překlad, které autoritativní server nedokáže sám zpracovat. Tím může být PowerDNS resolver, či jiný DNS server.

KnotDNS

KnotDNS[19] je vysoce výkonný server vyvíjený společností CZ.NIC. Jeho hlavním účelem je nasazení na autoritativní DNS uzly zpracovávající velké množství dotazů.

KnotDNS umí načítat záznamy ze zónových souborů BIND, nepodporuje však načítání záznamů z databáze. Server také neumí provádět rekurzivní překlad záznamů, avšak umí překlad delegovat na jiný server podobně jako PowerDNS.

Konfiguraci serveru je možné dynamicky měnit bez jeho restartu, avšak změna zónových souborů vyžaduje jejich kompletní přenačtení, nebo použití protokolu pro dynamické aktualizace.

Zvolené řešení

Pro implementaci byl vybrán server PowerDNS, který jako jediný ze zkoumaných DNS serverů podporuje ukládání záznamů do databáze.

Funkčnost serveru byla opět otestována v laboratorním prostředí. Aby PowerDNS omezil počet dotazů do databáze, používá cache dotazů. Ta je rozdělena na cache pro existující záznamy a pro neexistující. Pro každou

[17] https://www.isc.org/downloads/bind/
[18] https://www.powerdns.com/
[19] https://www.knot-dns.cz/

lze definovat dobu expirace zvlášť. Výchozí dobou expirace pro existující záznamy je 20 sekund, pro neexistující záznamy je výchozí 60 sekund.

Po vložení záznamu tedy může trvat až 60s, než server načte novou hodnotu. To je pro potřeby kolejní sítě dostačující a bylo by i možné tuto hodnotu zvýšit.

7.5.4 Firewall

Funkci firewallu v kolejní síti aktuálně zajišťuje linuxový stroj, kde probíhá filtrování paketů pomocí linuxového modulu netfilter. Ten se konfiguruje příkazem iptables. Pomocí něj lze přidávat, upravovat a mazat pravidla. Změny v pravidlech se projeví ihned, nejsou však ukládána: při restartu stroje se provedené změny odstraní. Je proto nutné obsah pravidel při změnách ukládat příkazem `iptables-save`.

V budoucnu by tento linuxový stroj mohl být nahrazen specializovaným hardware, například firewallem FortiGate od společnosti Fortinet, Inc. Ten poskytuje pro modifikaci pravidel vlastní API. Protože prozatím není výměna brány plánovaná, bude se následující text zabývat pouze variantou s modulem netfilter v OS Linux.

Pravidla jsou rozdělena do sad, které se nazývají chain. Nad paketem se spouští pravidla ze sady podle toho, o jaký typ paketu jde. Nad příchozími pakety se například spouští pravidla z chain `INPUT`, nad odchozími pravidla z chain `OUTPUT`. Lze definovat i vlastní sady (chains), na které se pak lze z ostatních pravidel odkazovat a zpracování paketu nechat provést podle pravidel v nich uvedených.

Registrační systém musí vzdáleně upravovat pravidla v netfilter. Při hledání řešení je hlavním kritériem bezpečnost: pravidla může měnit pouze superuživatel, zároveň pak je možné úpravou pravidel vytvořit potenciální bezpečnostní slabinu systému.

V rámci hledání řešení byly zkoumány dvě varianty: nástroj rfw, který umožňuje modifikovat pravidla iptables pomocí HTTP REST API, a vzdálený přístup pomocí SSH.

rfw: HTTP REST API pro iptables

rfw[20] je webová aplikace napsaná v jazyce Python. Tato aplikace poskytuje HTTP REST API, pomocí kterého je možné vkládat a mazat pravidla pro iptables. Ověřování klienta probíhá pomocí HTTPS certifikátů.

[20]https://github.com/securitykiss-com/rfw

Pomocí API lze vkládat a mazat pouze jednoduchá pravidla, například povolit přístup určitému stroji. rfw neumožňuje vložit pravidlo do vlastního chain.

V dokumentaci aplikace je uvedeno, že musí běžet s právy superuživatele. Nepodporuje spouštění příkazu iptables jako superuživatel pomocí příkazu sudo. To může být potenciální bezpečnostní slabinou, neboť chyba v kódu aplikace může potenciálně vést ke vzdálenému vykonávání kódu s právy superuživatele. Dokumentace neuvádí, zda jsou modifikace v pravidlech ukládány pomocí příkazu `iptables-save`.

Přístup pomocí SSH

Další možností, jak vzdáleně modifkovat pravidla, je pomocí vzdáleného přístupu do příkazové řádky pomocí protokolu SSH. Vzdálený systém nemusí mít plný přístup s právy superuživatele, je postačující, aby příkaz `iptables` spouštěl pomocí příkazu `sudo`. Ten také umožňuje omezit, jaké příkazy lze pomocí něj spouštět.

Můžeme tak omezit spouštění pouze příkazu `iptables`, případně na spouštění vlastního skriptu, který umožní provádět pouze předkonfigurované operace. Pokud ve skriptu řádně omezíme vstupy a omezíme jeho úpravy pouze na superuživatele, bude možné měnit pouze omezenou množinu pravidel a nebude možné jej zneužít k vytvoření bezpečnostní slabiny ve vzdáleném systému.

Zvolené řešení

Protože nástroj rfw nepodporuje operace s vlastními chains a musí být spouštěn s právy superuživatele, byla zvolena modifikace pravidel pomocí vlastních skriptů prostřednictvím SSH.

Konfigurační služba bude udržovat SSH spojení se vzdáleným strojem. Při požadavku na úpravu pravidel spustí pomocný skript, který provede požadované úpravy. Vstupem skriptu bude ID počítače, jeho IP adresa a požadovaný stav – povolení přístupu, karanténa či zablokování přístupu.

Firewall díky tomu bude zabezpečen i v případě, že dojde ke kompromitaci stroje s registračním systémem.

8 Návrh databáze

Pro uložení dat byla zvolena relační databáze PostgreSQL. Tuto databázi lze provozovat na operačním systému OS Linux, má podporu pro asynchronní notifikace zmíněné v oddíle 8.4.1 a poskytuje rozsáhlé prostředky pro programování na straně databáze. Výhodou je i vestavěná podpora pro datové typy reprezentující IP adresu, IP adresu se síťovou maskou (CIDR, Classless Inter-Domain Routing) a MAC adresu.

V tomto oddíle budou postupně uvedeny hlavní části navržené databáze. Uvedené diagramy jsou zjednodušené a neobsahují všechny tabulky a sloupce. Schéma databáze bylo navrhováno v software pgModeler. U jednotlivých sloupců jsou uvedeny datové typy a modifikátory. Význam jednotlivých modifikátorů:

pk: Primary Key,
fk: Foreign Key,
uq: Unique Key a
nn: Not Null

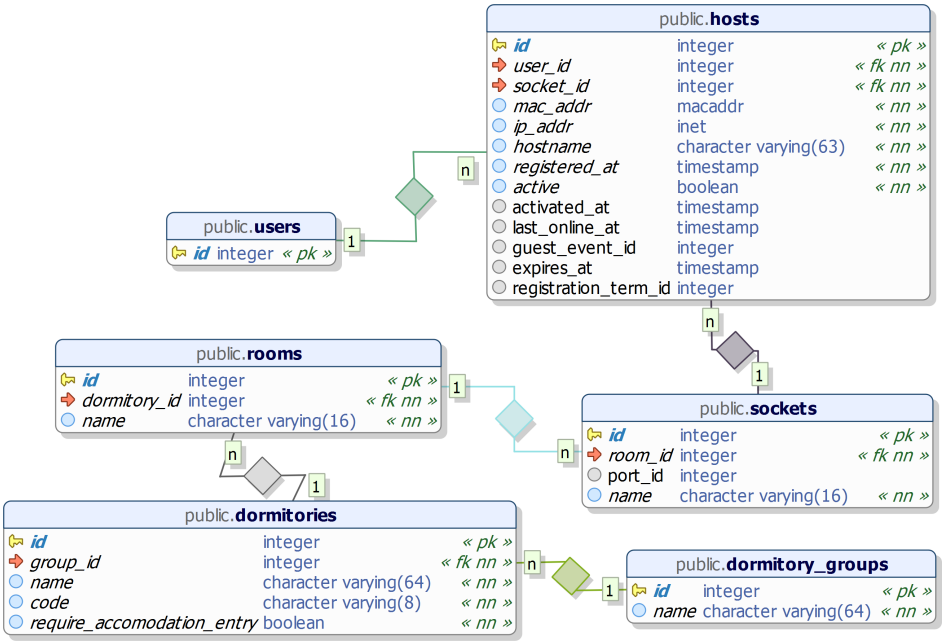
Při návrhu databáze byly některé volné vazby realizovány pomocí dvojice OID tabulky a ID záznamu. V databázi PostgreSQL má každý objekt databáze (a tedy i tabulka) unikátní číselný identifikátor. PostgreSQL obsahuje prostředky, které s tímto identifikátorem umožňují snadnou práci. Všechny operace jsou uvedeny v oficiální dokumentaci [24], zde budou uvedeny pouze dvě nejpoužívanější operace.

Výraz pro převod názvu tabulky na OID:
`'hosts'::REGCLASS::OID`
Příklad výstupu: `16974`

Převod OID na název tabulky:
`16947::REGCLASS::CHARACTER VARYING`
Příklad výstupu: `'hosts'`

8.1 Registrace zařízení

Schéma je znázorněno na obrázku 8.1. Registrace koncových stanic jsou uloženy v tabulce hosts. Každý uživatel (`users`) může mít více registrovaných stanic. Umístění stanice je určeno podle vazby stanice na zásuvku – uloženy v tabulce `sockets`. Zásuvky jsou dále přiřazeny k pokojům (tabulka `rooms`),



Obrázek 8.1: Schéma registrací koncových stanic.

pokoje jsou přiřazeny k budovám kolejí (`dormitories`) a koleje jsou rozděleny do skupin (`dormitory_groups`).

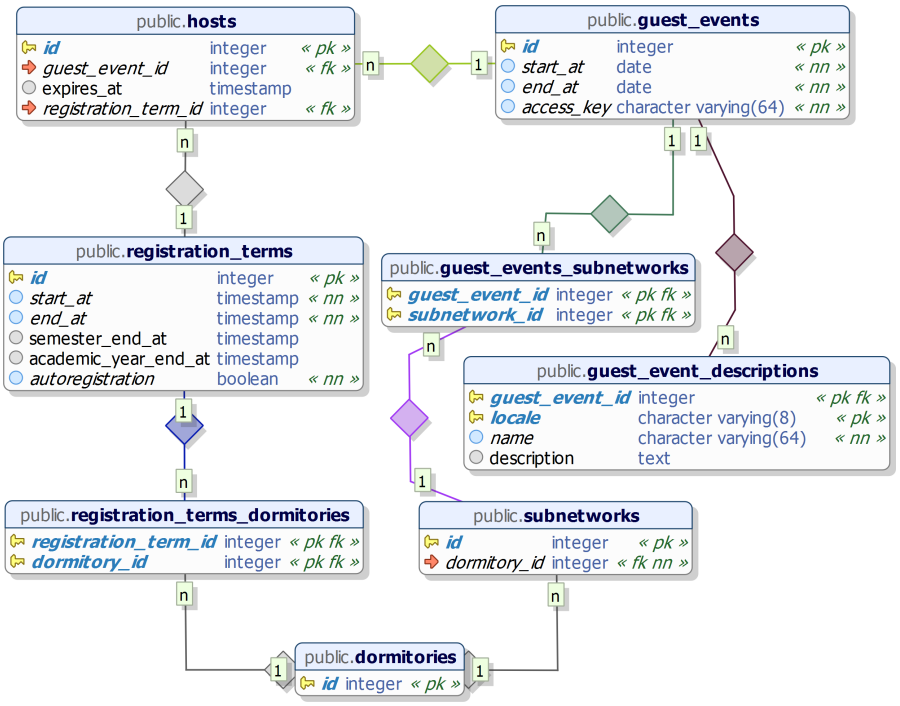
Při registraci studenta ubytovaného na kolejích je stanici přiřazeno registrační období (`registration_terms`), v rámci kterého byla stanice registrována. Podle tohoto registračního období se řídí expirace registrace, tj. datum, kdy má být smazána. Registrační období mohou být na každé budově kolejí jiná a měla by být definována tak, aby byl zajištěn plynulý přechod kolejí do letního režimu.

V případě, že probíhá letní akce univerzity (MLJŠ, ArtCamp či jiné), registrují se hosté pomocí jiného formuláře. Ten jejich registraci spáruje s konkrétní letní akcí (`guest_events`), v rámci které se registrovali. Dle data ukončení letní akce se určuje expirace registrace.

Databázový systém nezajišťuje mazání expirovaných záznamů, k tomuto účelu slouží obslužné rutiny v aplikaci. Vazby mezi stanicí a registračními obdobími jsou znázorněny na obrázku 8.2. Registrační období (`registration_terms`) mají vazbu N:M na seznam kolejí. Díky tomu je možné nastavit registrační období dle harmonogramu uzavření kolejí.

Letní akce (`guest_events`) jsou vázány na konkrétní podsítě. To umožňuje přesnější nastavení, kde má probíhat režim volné registrace. K registraci v režimu letních akcí není vyžadováno přihlášení, pouze přístupový klíč. Pokud by na jedné koleji byli ubytování studenti i účastníci letních akcí, je nežádoucí, aby se studenti mohli registrovat bez přihlášení v rámci letních

akcí. Tito studenti se musejí pro registraci prokázat svým Orion účtem.



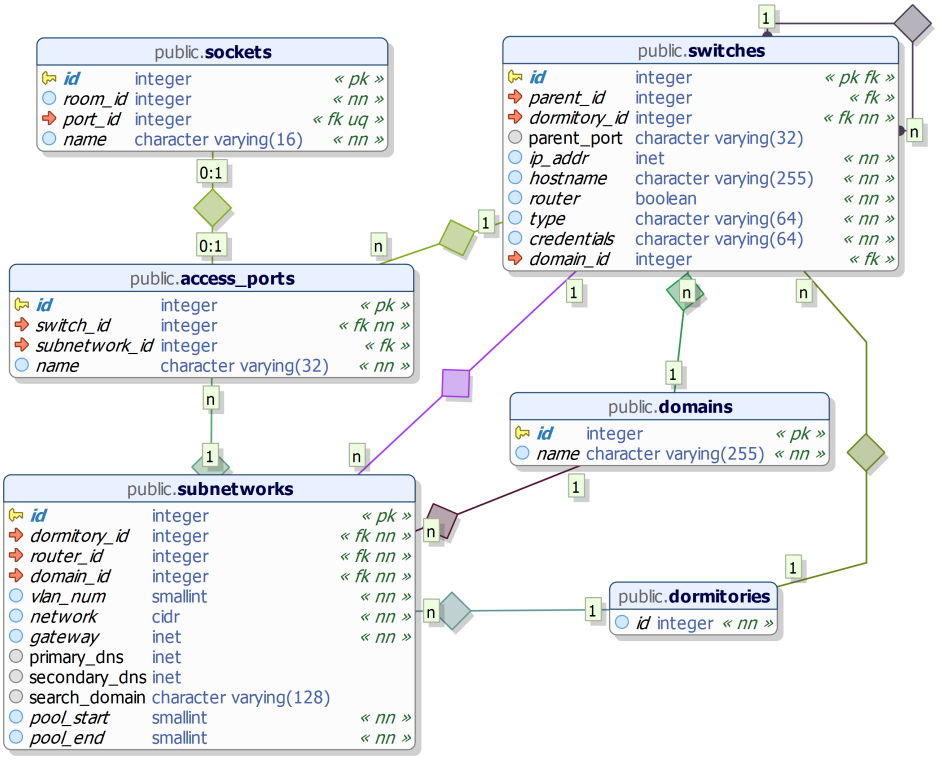
Obrázek 8.2: Vazby mezi registracemi zařízení a registračními obdobími.

8.2 Síťové prvky a podsítě

Schéma síťových prvků a podsítí je znázorněno na obrázku 8.3. Každá zásuvka (`sockets`) může být přiřazena k jednomu přístupovému portu přepínače (`access_ports`). Přístupový port má přiřazenou podsíť (tabulka `subnetworks`). Přepínač (tabulka `switches`) může mít libovolné množství portů. U každého přepínače jsou uvedeny základní informace (jeho IP adresa, hostname), příznak router udává, zda se jedná o L3 přepínač s možností směrování. Sloupec type udává identifikátor konektoru, který má být použit pro přístup k danému přepínači.

Konektory jsou definované v aplikaci a obsahují kód, který má být použit pro načítání informací z daného přepínače.

U podsítí je pak uvedeno číslo VLAN, které dané síti odpovídá, adresa sítě včetně síťové masky (sloupec `network`), výchozí brána, adresy DNS serverů a další. Tyto údaje mohou být použity k vygenerování síťové konfigurace pro DHCP server. Sloupce `pool_start` a `pool_end` pak udávají rozsah IP adres, které mají být použity při přidělování IP adres registrovaných stanic. Sloupce jsou číselné a hraniční IP adresy jsou určeny přičtením jejich hodnot k adrese



Obrázek 8.3: Schéma síťových prvků a podsítí.

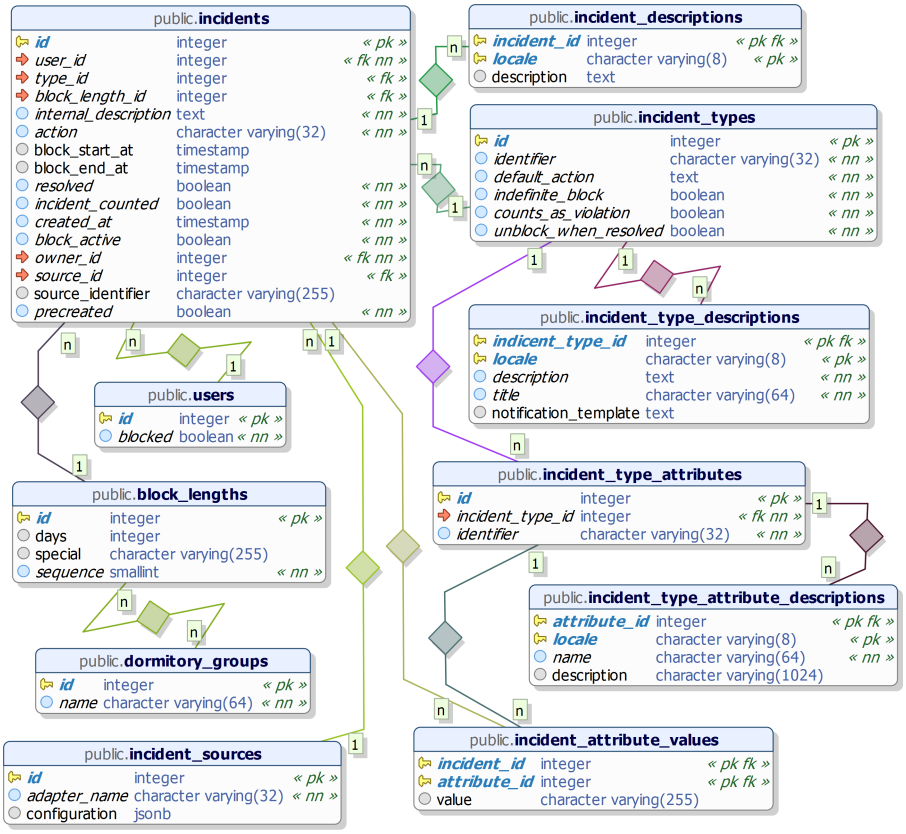
sítě. Pokud tedy adresa sítě bude 10.10.75.0, `pool_start` 11 a `pool_end` 230, budou rezervace přidělovány z rozsahu 10.10.75.11 - 10.10.75.230.

8.3 Incidenty

Schéma tabulek, které reprezentují incidenty, je znázorněno na obrázku 8.4. Incidenty jsou uloženy v tabulce `incidents`. U každého incidentu musí být uvedeno, jakému uživateli náleží, typ incidentu, akce a jaký uživatel incident založil. Akce incidentu může nabývat hodnot `none` (bez akce), `notify` (zaslání upozornění) a `block` (zablokování uživatele). K incidentu dále mohou být přiřazeny volitelné textové atributy, které poskytují doplňující informace (například typ detekovaného malware či počet detekovaných paketů).

Pokud je akcí incidentu blokace uživatele, obsahuje incident dále informaci o začátku a konci blokace. Ve sloupci `block_active` je uvedeno, zda je blokace uživatele stále aktivní.

Schéma incidentů podporuje lokalizace. K incidentu jsou ve vazbě 1:N přiřazeny popisky (`incident_descriptions`), které budou zobrazeny uživateli. Každý řádek v tabulce reprezentuje popisek v jedné lokalizaci.
Typy incidentů (`incident_types`) slouží jako šablona. Jsou k nim opět



Obrázek 8.4: Schéma incidentů a vazeb mezi tabulkami.

přiřazené lokalizované textové popisky, dále pak určují chování incidentu – výchozí akci (`action`) a příznaky:

`indefinite_block`: Pokud bude uživatel zablokován, bude ve výchozím nastavení délka blokace neurčena. Uživatele bude nutno odblokovat manuálně. Při zakládání incidentu je možnost délku změnit.

`counts_as_violation`: Započítat incident jako porušení pravidel. U uživatele je evidován počet porušení pravidel kolejní sítě. Na základě toho pak systém dokáže automaticky určit délku jeho blokace. Délky jsou definovány v tabulce block_lengths, která bude vysvětlena dále v textu.

`unblock_when_resolved`: Automaticky odblokovat uživatele v případě, že je incident označený jako vyřešený.

Typ incidentu také definuje atributy, které je možné u daného incidentu použít (`incident_type_attributes`). Názvy atributů je opět možné lokalizovat (`incident_type_attribute_descriptions`).

V tabulce `block_lengths` jsou definovány časové délky, po kterou má být uživatel zablokován. U každé délky je uveden buď počet dní, nebo

jedna ze speciálních hodnot: `end_semester` (blokace do konce semestru) či `end_academic_year` (blokace do konce akademického roku). Tyto speciální hodnoty se určují na základě aktuálního registračního období. Pokud nejsou definovány, dopočítají se automaticky[1].

Při určení délky blokace se vyberou všechny definované délky, které jsou přiřazeny skupině kolejí, ke které je blokovaný uživatel přiřazen. Délky jsou seřazeny podle hodnoty ve sloupci `sequence`. Poté je vybrána délka v pořadí podle počtu porušení pravidel uživatele. Pro 0 přestupků první záznam, pro 1 přestupek druhý atd.

Systém obsahuje i podporu pro zasílání změn v incidentech zpět do systémů, ze kterých byl incident založen pomocí API. Díky tomu je možná obousměrná synchronizace incidentů. Každému systému je přiřazeno unikátní ID v tabulce `incident_sources`. Ve sloupci `adapter_name` je vyplněn identifikátor třídy, která má synchronizaci zajišťovat, ve sloupci `configuration` pak volitelná konfigurace ve formátu JSON. Ta je předána třídě, která zajišťuje synchronizaci. Systém v současnosti podporuje pouze adaptér `webhook`.

Systém synchronizace incidentů a rozhraní třídy zajišťující synchronizaci jsou popsány dále v textu.

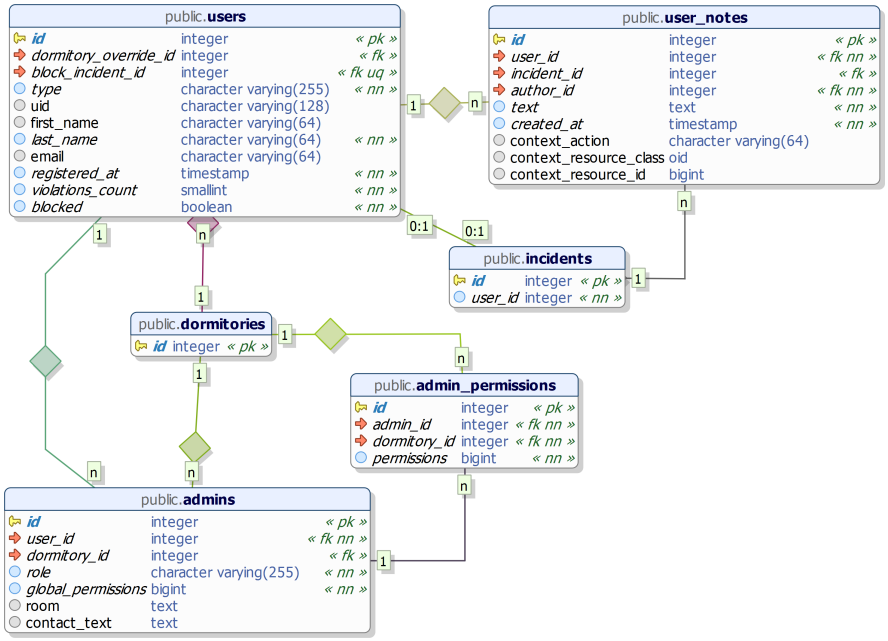
8.4 Uživatelé a systém oprávnění

Schéma tabulek uživatelů a jejich oprávnění je znázorněno na obrázku 8.5. Uživatelé systému jsou uloženi v tabulce `users`. V této tabulce jsou obsaženy informace o uživateli, informace, kdy byl registrován a zda je zablokován. V případě, že je uživatel zablokován, je ve sloupci `block_incident_id` uvedeno i ID incidentu, kterým je uživatel blokován.

Sloupec `dormitory_override_id` slouží k přepsání koleje, na které je uživatel ubytován. Registrace zařízení je dovolena jen uživatelům, kteří jsou na dané koleji ubytováni. Tato informace je načítána z adresáře LDAP. Pokud je informace v adresáři chybná, je možné ji v tomto sloupci přepsat.
Ve sloupci type je uveden typ uživatele. Může nabývat následujících hodnot:

system: Systémové konto. Používá se pro určení identity klienta API.

[1]Konec akademického roku lze určit přesně: 31. 8. Tímto datem končí i letní semestr. Výpočet konce zimního semestru je komplikovaný, závisí na začátku a délce zkouškového období, počtu svátků, které připadají na pracovní dny, a dalších faktorech. V programu je použita zjednodušená varianta, která předpokládá délku zkouškového období 6 týdnů a začátek zkouškového období 2. 1., nebo první pondělí, pokud 2. 1. připadá na pátek, sobotu či neděli. Předpokládá se, že se konec zimního semestru bude v systému definovat na základě platné vyhlášky rektora.



Obrázek 8.5: Schéma uživatelů a systému oprávnění.

ldap: Student či zaměstnanec ZČU, informace jsou načítány ze systému LDAP.

external: Student jiné univerzity. Typicky půjde o studenty Lékařské fakulty Univerzity Karlovy v Plzni.

guest: Host registrovaný v rámci letní akce.

Pro uživatele typu ldap je ve sloupci UID uvedeno jejich přihlašovací jméno.

Ke každému uživateli je možné přidat jednu nebo více poznámek. U poznámky je kromě textu, času a autora uveden i kontext, ve kterém byla poznámka přidána. Poznámka může být přidána k incidentu, nebo v rámci jiné akce – například schvalování registrace počítače. Ve sloupci `context_action` pak bude textový identifikátor akce, `context_resource_class` bude obsahovat OID tabulky, ke kterému se váže daná akce (při schvalování registrace bude uvedeno OID tabulky `hosts`) a `context_resource_id` ID záznamu v tabulce (při schvalování registrace zde bude uvedeno ID schvalovaného zařízení).

Správci systému budou mít dále uvedený záznam v tabulce admins. Každý záznam v této tabulce je vázaný na kolej, na které je daný správce ubytován. Uživatel může mít více záznamů, pokud vykonává funkci správce na více kolejích. Sloupec role může nabývat následujících hodnot:

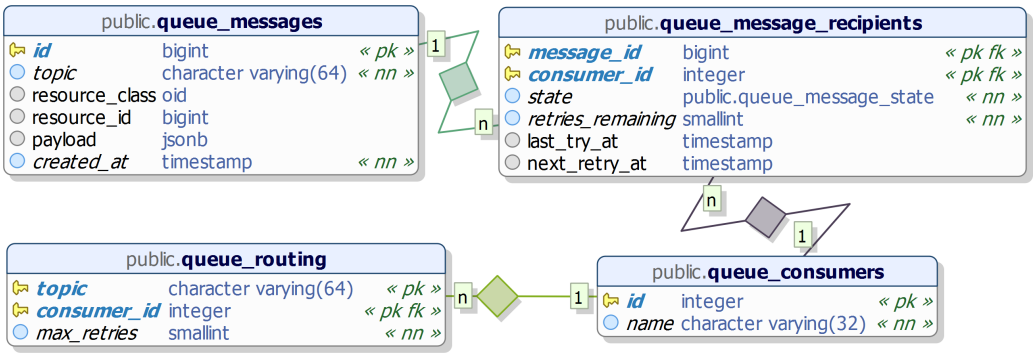
`local_admin`: Lokální správce. Oprávnění se určují dle sloupce `global_permissions` a tabulky admin_permissions.

super_admin: Administrátor. Má všechna oprávnění a tabulka `admin_permissions` je ignorována.

V tabulce `admin_permissions` jsou uvedena oprávnění správce na různých kolejích. Ve sloupci `permissions` je uložena bitová maska oprávnění, která byla správci na dané koleji přidělena.

8.5 Fronta zpráv

Schéma tabulek, ve kterých je uložena fronta zpráv, je znázorněno na obrázku 8.6.



Obrázek 8.6: Schéma fronty zpráv.

Zprávy jsou uloženy v tabulce `queue_messages`. U zprávy je uložen typ (`topic`), obsah (`payload`) a datum vytvoření. Ve sloupcích `resource_class` (OID) a `resource_id` je uložena reference záznamu, ke kterému se zpráva váže. Typicky zde bude OID tabulky `hosts` a ID zařízení, kterého se konfigurace týká.

Jedna zpráva může být určena více konzumentům (`queue_consumers`) – příjemci jsou uloženi v tabulce `queue_message_recipients`. Zpráva má pro každého příjemce vlastní stav:

wait: čeká na zpracování dalších zpráv,

active: aktivní zpráva, čeká na vyzvednutí a zpracování,

error_retry: došlo k chybě, čeká na opakování zprávy. V čase určeném next_retry_at by měla být zpráva znovu zařazena ke zpracování přepnutím do stavu **active**.

failed: při zpracování zprávy byl vyčerpán maximální počet pokusů. Zprávu je nutné ručně analyzovat, odstranit překážky v jejím zpracování a případně znovu zařadit do fronty.

Pomocí tabulky `queue_message_recipients` je pro každého konzumenta vytvořena jeho vlastní fronta. Konzument po zpracování zprávy odstraní svůj záznam o přijetí zprávy. Pokud je posledním příjemcem, je odstraněna i zpráva z tabulky `queue_messages`.

Příjemce zprávy je určen na základě jejího typu (`topic`) a záznamů v tabulce `queue_routing`, která mapuje typ zprávy na jednotlivé příjemce a určuje, o kolik pokusů o zpracování se má daný příjemce u zprávy pokusit. Zpráva je doručena všem příjemcům, pro které je nalezen odpovídající záznam v `queue_routing`. V této tabulce je možné používat také stejné zástupné znaky jako v PostgreSQL u operátoru `LIKE`. Pokud zde bude pro příjemce definován řádek, kde je jako typ zprávy definován řetězec `host.%`, budou příjemci doručeny všechny zprávy, které začínají řetězcem `host.`.

Předání zprávy příjemci voláním `NOTIFY` je řízeno automaticky pomocí databázových triggerů. Automaticky je také řízeno mazání zprávy z `queue_messages`, pokud jej zpracují všichni příjemci.

Správné zpracování zpráv vyžaduje zamykání obou tabulek následujícím příkazem:

```
LOCK TABLE queue_messages, queue_message_recipients
IN ACCESS EXCLUSIVE MODE;
```

Všechny obslužné rutiny používají toto pořadí zamykání, je proto nutné jej dodržet a nikdy nezamykat tabulku queue_message_recipients dříve než tabulku queue_messages. Je také nutné mít na paměti, že tabulku implicitně uzamkne libovolný příkaz, který k ní přistupuje, a to včetně příkazu SELECT.

Z těchto důvodů by s tabulkami nemělo být manipulováno přímo, ale prostřednictvím obslužných rutin. V databázi jsou definovány následující funkce, pomocí kterých je možné frontu obsluhovat.

`send_message(topic, resource_class, resource_id, payload)`
Přidá do fronty zprávu s daným obsahem. Pokud jsou ve frontě doposud nezpracované zprávy se stejným resource_class a resource_id, je zpráva vložena ve stavu wait.

`fetch_active_messages(consumer_id)`
Vybere z fronty všechny zprávy ve stavu active pro daného konzumenta.

`get_message(message_id, consumer_id)`
Získá informace o zprávě s daným ID pro uvedeného konzumenta. Vybírá informace z tabulek queue_messages a queue_message_recipients.

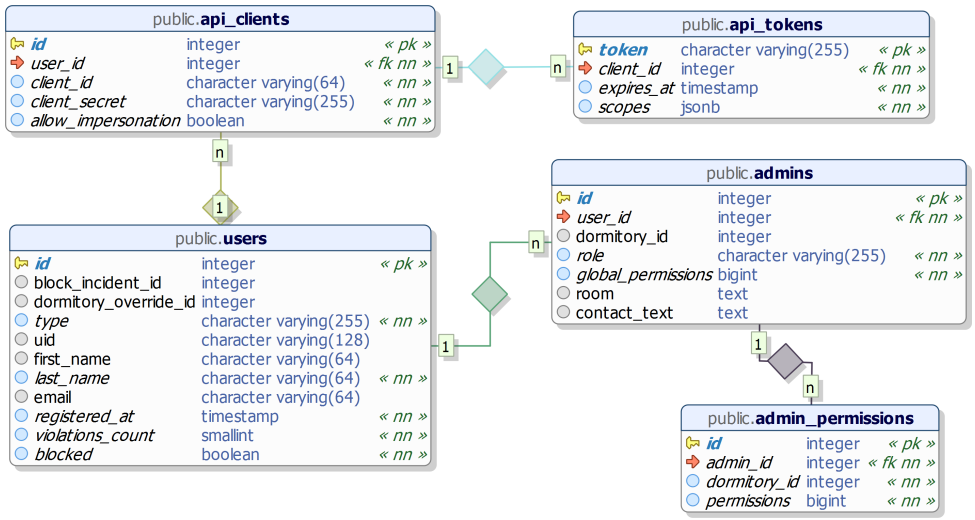
`confirm_message(message_id, consumer_id)`
Potvrdí zprávu s daným ID pro daného konzumenta. Pokud zpráva nemá dalšího příjemce, je smazána z fronty. Pokud po smazání ve frontě čeká zpráva ve stavu wait, která má stejné resource_class a resource_id, je přepnuta do stavu active.

`retry_message(message_id, consumer_id, retry_interval)`
Zařadí zprávu do fronty k pozdějšímu pokusu. Pokud již byl vyčerpán počet pokusů, je zpráva přepnuta do stavu failed.

8.6 Přístupy k REST API

Schéma klientů API je znázorněno na obrázku 8.11. V tabulce `api_clients` jsou uloženy definice klientů, kteří mají oprávnění přistupovat k API. U každého klienta je uloženo jeho *Client ID* a *Client Secret*. *Client Secret* není uloženo přímo, ale pouze jeho otisk vypočítaný metodou bcrypt[2]. Ke každému klientovi API se váže jeho identita v tabulce `users`. K této identitě je přiřazeno oprávnění stejně jako u libovolného jiného uživatele.

Přístupové klíče jsou uloženy v tabulce `api_tokens`. Každý klíč má expiraci, jejich mazání zajišťuje periodická úloha v aplikaci.

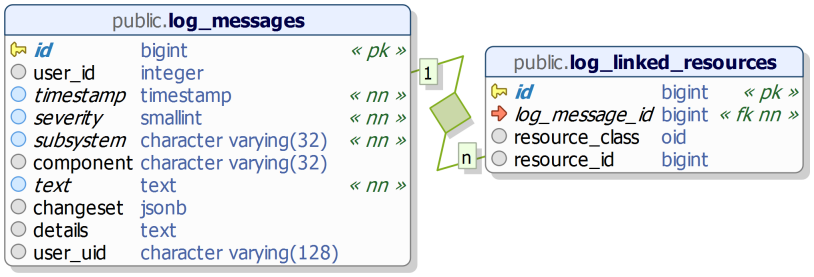


Obrázek 8.7: Schéma ukládající přístupy k API.

[2]http://php.net/manual/en/function.password-hash.php

8.7 Záznamy akcí – logování

Každá akce, která je v systému provedena, se zaznamenává do databáze. Do databáze také mohou konfigurační služby zapisovat záznamy o své činnosti. Tabulky, ve kterých se ukládají záznamy prováděných akcí, jsou znázorněny na obrázku 8.8.



Obrázek 8.8: Schéma tabulek se záznamy akcí.

Každá zpráva v tabulce `log_messages` má čas zaznamenání, závažnost, identifikaci části systému, které se zpráva týká (sloupce `subsystem` a `component`), a text. Se zprávou může být volitelně spojena sada změn (`changeset`), která byla provedena, doplňující detaily a ID a přihlašovací jméno uživatele, který změnu provedl.

Zprávu je také možné v tabulce `log_linked_resources` spojit se záznamy v ostatních tabulkách, kterých se změna týká. Pokud například správce upraví informace o registraci zařízení, bude mít zpráva o úpravě registrace v této tabulce záznam, kde `resource_class` bude nastavené na OID tabulky `hosts` a `resource_id` nastavené na ID registrace zařízení. Na základě této vazby je pak možné vyfiltrovat jen ty záznamy z logu, které se týkají určitého objektu v databázi.

9 Implementace webového portálu

Pro implementaci webového portálu byl zvolen jazyk PHP 7. Tento jazyk je velice populární pro tvorbu webových aplikací (podle serveru W3Tech, který provozuje společnost Q-Success, jej používá více než 80% webů na internetu [36]) a je podporován silnou komunitou.

Oproti dynamickým jazykům Python a Ruby, které jsou často pro tvorbu webových aplikací používány, obsahuje přímou podporu pro rozhraní a abstraktní třídy – lze je využít pro popis konektorů k externím službám a síťovým prvkům. Výhodou je i to, že v něm jsou implementovány také další systémy provozované na ZČU, například dříve zmíněná Mysphere2.

Dále byl použit framework Symfony 3.0[1], který má kvalitní dokumentaci a je podporován celosvětovou komunitou v čele se společností SensioLabs. Framework používá architekturu Model-View-Controller (MVC) a je rozdělen do několika komponent (konfigurace aplikace, formuláře, zabezpečení, validace dat a další). Komponenty jsou na sobě nezávislé a v případě potřeby lze použít jinou implementaci. Komponenty jsou také navrženy tak, aby bylo možné je snadno rozšířit, případně nahradit jednotlivé komponenty či jejich části jinou implementací.

Při realizaci aplikace však byla snaha využít co nejlépe prostředky, které jsou dostupné přímo v Symfony, neboť jsou kvalitně zdokumentovány a otestovány.

Pro přístup k databázi je použita ORM knihovna Doctrine 2[2], která je podle dokumentace frameworku Symfony doporučenou knihovnou pro přístup k databázi a jsou k dispozici prostředky pro její snadnou integraci do aplikace. Knihovna implementuje základní operace Create, Retrieve, Update, Delete (CRUD) a usnadňuje tak vývoj. Nevýhodou je zvýšená režie na zpracování, tu však lze snížit použitím cache.

Všechny závislosti aplikace jsou spravovány nástrojem composer[3]. Ten umožňuje snadno nainstalovat a případně aktualizovat všechny závislosti aplikace.

[1] http://symfony.com/
[2] http://www.doctrine-project.org/
[3] https://getcomposer.org/

9.1 Adresářová struktura

Webová aplikace má následující adresářovou strukturu:

app: Konfigurace a kód potřebný pro inicializaci aplikace.

- **config:** konfigurace celé aplikace. Přístupové údaje jsou uloženy v souboru parameters.yml, který se vygeneruje při instalaci aplikace. Ostatní soubory by neměly být při nasazení aplikace měněny.
- **DoctrineMigrations:** migrace databáze.

bin: Pomocné skripty pro správu aplikace z příkazové řádky. Skript console je použit pro spouštění obslužných rutin aplikace, nápovědu a dostupné příkazy je možné získat spuštěním php bin/console.

src: Zdrojové kódy aplikace. Aplikace je rozdělena do tzv. bundles. Každý bundle je sada souborů, které dohromady implementují určitou část systému. Bundle obsahuje jak PHP kód, tak šablony, konfigurační soubory, lokalizační řetězce a další zdroje.

- **ApiBundle:** bundle s REST API pro přístup k datům v systému. Popis bundle, včetně jeho adresářové struktury, bude uveden v sekci 10.4.
- **KNetBundle:** hlavní bundle projektu.

var: Soubory generované při běhu aplikace.

- **cache:** Předzpracovaná konfigurace aplikace, vygenerované proxy soubory a další aplikační cache.
- **logs:** Logy z běhu aplikace. Pokud v aplikaci dojde k chybě, bude zde zaznamenána.
- **sessions:** Soubory s daty uživatelských relací.

vendor: Závislosti aplikace: PHP knihovny třetích stran. Tato složka je kompletně spravována nástrojem composer.

web: Veřejně dostupná část projektu. Obsahuje soubor app.php, jež spouští inicializaci aplikace a zpracování požadavku.

- **css:** Soubory s kaskádovými styly uživatelského rozhraní.
- **images:** Obrázky použité v rozhraní.
- **js:** JavaScript uživatelského rozhraní.
  - **libs:** Knihovny třetích stran.

9.2 Architektura aplikace

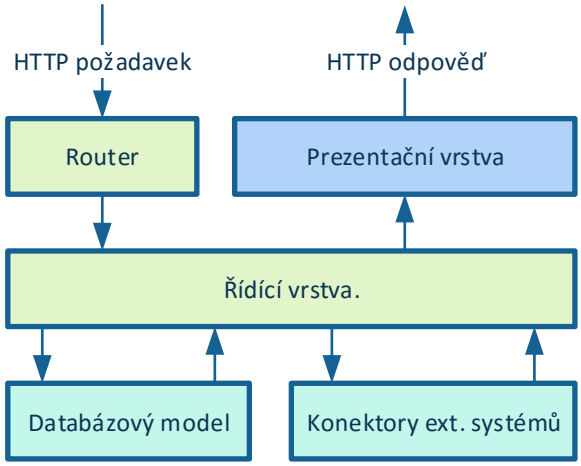
Jak již bylo uvedeno na začátku kapitoly, Symfony používá architekturu MVC. Aplikace je tak rozdělena do 3 vrstev:

Datová (Model): zajišťuje ukládání a získávání dat. V aplikaci je tvořena převážně Doctrine 2 entitami, třídami, které s nimi pracují, a dále pak třídami, které zajišťují načítání dat ze síťových prvků a z adresářové služby LDAP.

Prezentační (View): šablony, které generují HTML. Použit je šablonovací systém Twig, který je integrován v Symfony.

Řídící (Controller): zajišťuje zpracování požadavku od uživatele. Typicky se zpracování bude skládat ze 3 fází: validace požadavku od uživatele, načtení dat z modelu a vykreslení šablony.

Kromě těchto vrstev je v aplikaci použit tzv. router, který provádí mapování HTTP požadavků na kontrolér a jeho parametry. Router v Symfony také podporuje generování URL adresy podle cílového kontroléru. Vrstvy aplikace a jejich interakce při zpracování HTTP požadavku jsou znázorněny na obrázku 9.1.



Obrázek 9.1: Vrstvy aplikace a jejich vzájemná interakce.

V následujícím textu budou stručně popsány jednotlivé vrstvy aplikace. Důraz bude kladen na nejdůležitější principy, které usnadní pochopení aplikace, a případná rozšíření standardních komponent Symfony.

9.3 Datová vrstva

Ačkoliv je datová vrstva společná pro všechny části aplikace (`KNetBundle` i `ApiBundle`), jsou všechny její třídy umístěny v `KNetBundle`. Datovou vrstvu můžeme rozdělit na:

- práci s databází,
- načítání dat ze služby LDAP a
- načítání dat ze síťových prvků.

9.3.1 Práce s databází

Pro práci s databází je použita knihovna Doctrine. Ta umožňuje definovat datové entity, které je možné ukládat a načítat z databáze. Entita je třída v PHP, jejíž strukturu lze namapovat na databázové tabulky. K mapování lze použít konfiguraci v datových souborech (XML, YAML (YAML Ain't Markup Language)), nebo přímo anotace v dokumentačních komentářích v kódu dané třídy. V aplikaci je použito mapování pomocí anotací, aby byla entita i její mapování uloženo v jednom souboru.

Entity jsou uloženy ve složce `KNetBundle/Entity`. Každá entita obsahuje vlastnosti, které odpovídají databázovým sloupcům. Jejich struktura a vazby mezi nimi odpovídají datovému modelu, který je uveden v kapitole 8. Z tohoto důvodu zde nebudou entity dále popisovány.

Nejdůležitější třídou v Doctrine je `EntityManager`. Tato třída řídí načítání metadat (mapování) jednotlivých entit, řízení transakcí, vyhodnocování a ukládání změn a vytváření repositářů pro načítání entit. Repositář je instance třídy `EntityRepository`. Ta umožňuje vyhledat entitu podle jejího identifikátoru, nebo podle více kritérií zadaných jako asociativní pole.

`EntityRepository` má podobnou úlohu jako DAO (Data Access Object), avšak poskytuje pouze operace pro načítání entit. Příkladem může být operace `find($id)`, která najde entitu dle hodnoty primárního klíče.

Ukládání nových instancí a modifikace existujících objektů probíhá dávkově prostřednictvím třídy `EntityManager`. Voláním `persist($object)` se nejprve do dávky přidají všechny objekty, které mají být uloženy, voláním `flush()` pak Doctrine vypočítá všechny provedené změny a provede minimální množství dotazů do databáze, které dávku uloží.

Dotaz do databáze lze také zadat jazykem DQL (Doctrine Query Language). Tento jazyk vychází z jazyka SQL, nepracuje se v něm však s tabulkami, ale přímo s entitami, které jsou identifikovány názvem své třídy. Ten je pak překládán do jazyka SQL pro konkrétní platformu. Pokud bychom chtěli zís-

kat seznam všech počítačů (entita `KNetBundle\Entity\Host`), které vlastní uživatel s ID 42, použijeme následující DQL dotaz:

```
SELECT host FROM KNetBundle\Entity\Host host
WHERE host.user = 42
```

Doctrine tento dotaz přeloží na ekvivalent následujícího dotazu[4]:

```
SELECT hosts.* FROM hosts WHERE hosts.user_id = 42
```

S třídami z Doctrine nepracuje kontrolér přímo, ale používá servisní vrstvu. Každá entita má vlastní servisní třídu, jež poskytuje abstrakci nad Doctrine a obsahuje operace, které jsou pro danou entitu specifické.

Příkladem je třída `HostManager`, která poskytuje operace s entitami typu `KNetBundle\Entity\Host`. UML diagram znázorňující třídu `HostManager` a třídy související, je uveden na obrázku 9.2. Pro přehlednost jsou uvedeny pouze nejdůležitější vlastnosti a metody.

Tyto třídy často pracují s více entitami najednou. Například již dříve zmíněná třída `HostManager` ukládá nejen registraci zařízení, ale i kolekci jí přiřazených aliasů. Při vytvoření registrace také automaticky přidá záznam do tabulky logů. Kontroléry uživatelského rozhraní a API tak mohou používat stejné operace bez toho, aby musely implementovat stejnou logiku zpracování dat.

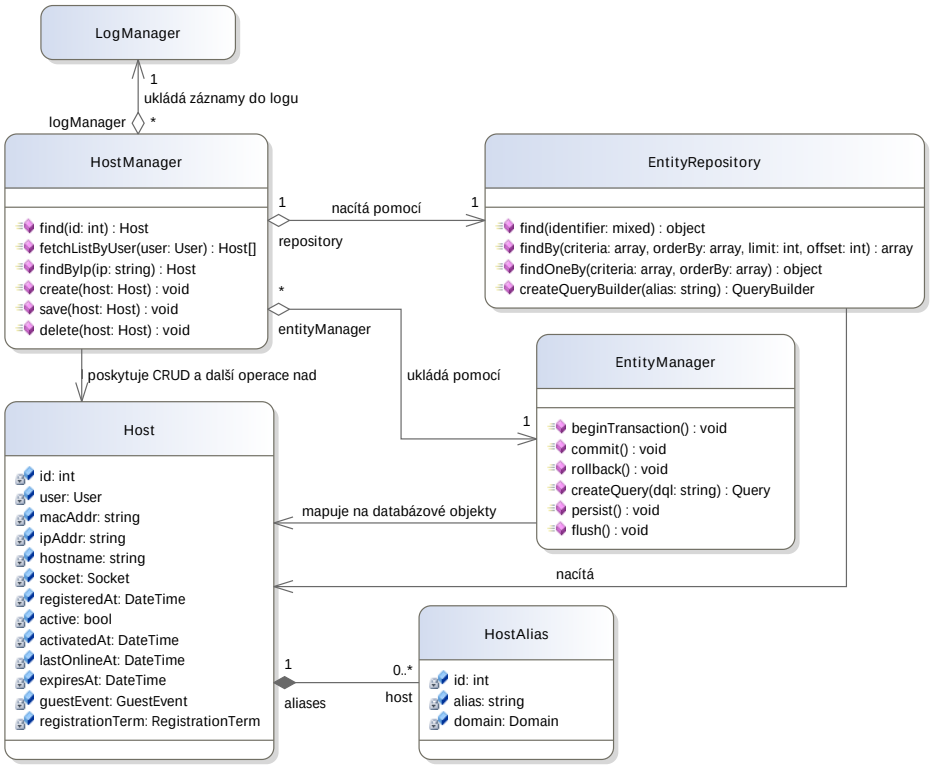
9.3.2 Načítání dat z LDAP

Načítání dat z LDAP zajišťuje třída `KNetBundle\Ldap\LdapConnector`, která je znázorněná na obrázku 9.3. Třída používá funkce z PHP modulu *ldap*. Pro získání informací o uživateli podle jeho přihlašovacího jména používá funkci `ldap_read()`, která provádí operaci *Search* se `scope=baseObject`. Hledá tedy konkrétní záznam dle jeho DN. Servery ZČU používají pro informace o účtech DN ve formátu `uid=<login>,ou=Users,ou=rfc2307,o=zcu,c=cz`.

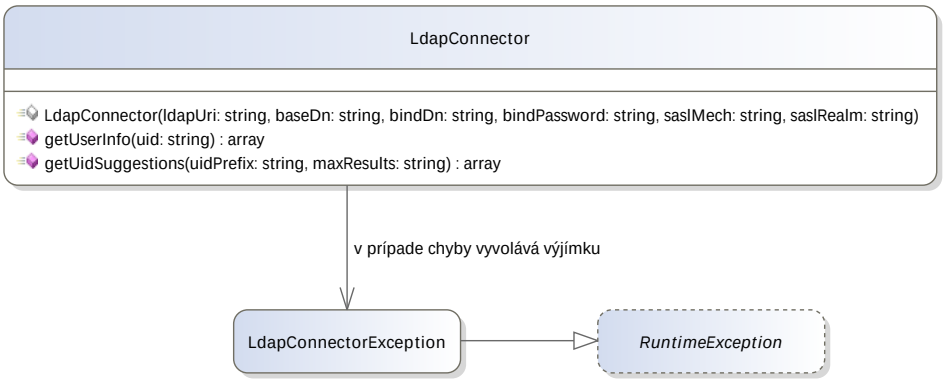
Hledání návrhů na přihlašovací jména používá funkci `ldap_search()`, která prohledává celý podstrom `ou=Users,ou=rfc2307,o=zcu,c=cz`.

Kořenové DN stromu, ve kterém jsou umístěny informace o uživatelských účtech, je konfigurovatelné. Třídě je možné jej zadat jako parametr konstruktoru `$baseDn`.

[4]Dotaz, který Doctrine vygeneruje, obsahuje výčet všech známých sloupců. Tabulky jsou navíc aliasovány a odkazovány vygenerovaným identifikátorem.



Obrázek 9.2: UML diagram třídy `HostManager`.



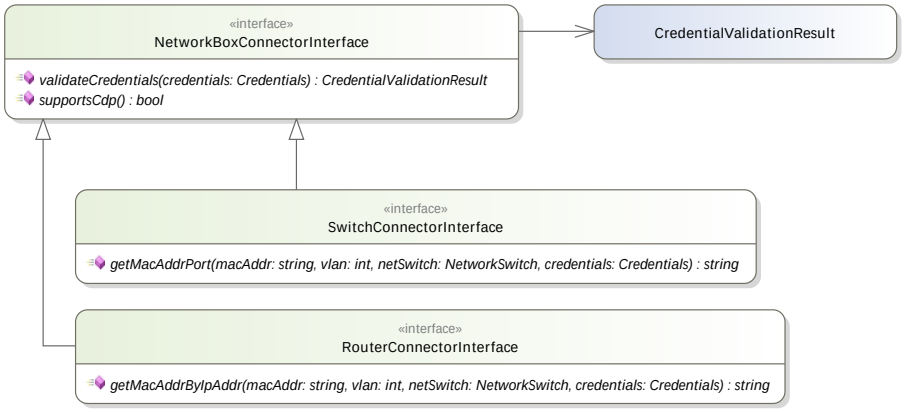
Obrázek 9.3: UML diagram třídy `LdapConnector`.

9.3.3 Načítání informací ze síťových prvků

Všechny třídy pro načítání informací ze síťových prvků jsou umístěny v namespace `KNetBundle\Connector`. V dalším textu již tento namespace nebude uveden. Načítání zajišťují konektory, které implementují rozhraní `SwitchConnectorInterface` a `RouterConnectorInterface`.

Všechny operace těchto rozhraní přijímají, kromě jiných argumentů, dva objekty: entitu `NetworkSwitch`, která reprezentuje přepínač v síti a obsahuje mimo jiné jeho IP adresu, a instanci třídy `Credentials`. Ta reprezentuje kolekci přístupových údajů do daného přepínače. V kolekci může být pro každý protokol (SNMP, SSH, atd.) jedna sada přístupových údajů.

Obě rozhraní rozšiřují rozhraní `NetworkBoxConnectorInterface`, které definuje dvě operace: `validate(Credentials $credentials)` a supportsCdp(). `validate` ověřuje, že zadaná kolekce obsahuje všechny přístupové údaje, které jsou potřebné pro správnou funkci daného konektoru. Druhá operace, `supportsCdp`, vrací logickou hodnotu `true`, pokud dané síťové zařízení podporuje protokol CDP. Rozhraní jsou znázorněna na obrázku 9.4.



Obrázek 9.4: UML diagram rozhraní pro konektory k síťovým prvkům.

V aplikaci jsou implementovány konektory pro přepínače Catalyst série 2900 a 3700. Stejný konektor pravděpodobně bude možné použít i pro ostatní přepínače stejného výrobce, avšak testovány byly pouze proti stávající infrastruktuře kolejních sítí.

Jednotlivé konektory jsou zaregistrovány jako služby v Depencency Injection Containeru, konkrétně v konfiguračním souboru `src/KNetBundle/Resources/config/services.yml`. Každá služba konektoru musí být označena tagem `knet.connector`, který má atribut `connector_name`:

```
services:
  knet.connector.cat2900:
```

```
    class: KNetBundle\Connector\Cisco\Catalyst2900
    arguments: ['@default_cache']
    tags:
      - { name: 'knet.connector', connector_name: 'cat2900' }
  knet.connector.cat3700:
    class: KNetBundle\Connector\Cisco\Catalyst3700
    arguments: ['@default_cache']
    tags:
      - { name: 'knet.connector', connector_name: 'cat3700' }
```

Více o službách naleznete v sekci 9.6 Konfigurace a předávání závislostí a v The Symfony Book, kapitole Service Container [33].

Snadný přístup k těmto konektorům v aplikaci pak zajišťuje třída `ConnectorProvider`. Této třídě je potřeba předat název, pod kterým byl konektor zaregistrován (např. cat2900), a třída vrátí příslušnou službu z DI Containeru. Tento název je také uveden v databázi jako typ síťového prvku v tabulce `switches`.

Přístupové údaje jsou uloženy v konfiguračním souboru, konkrétně v sekci `knet.credentials` souboru `parameters.yml`. Příklad příslušné sekce:

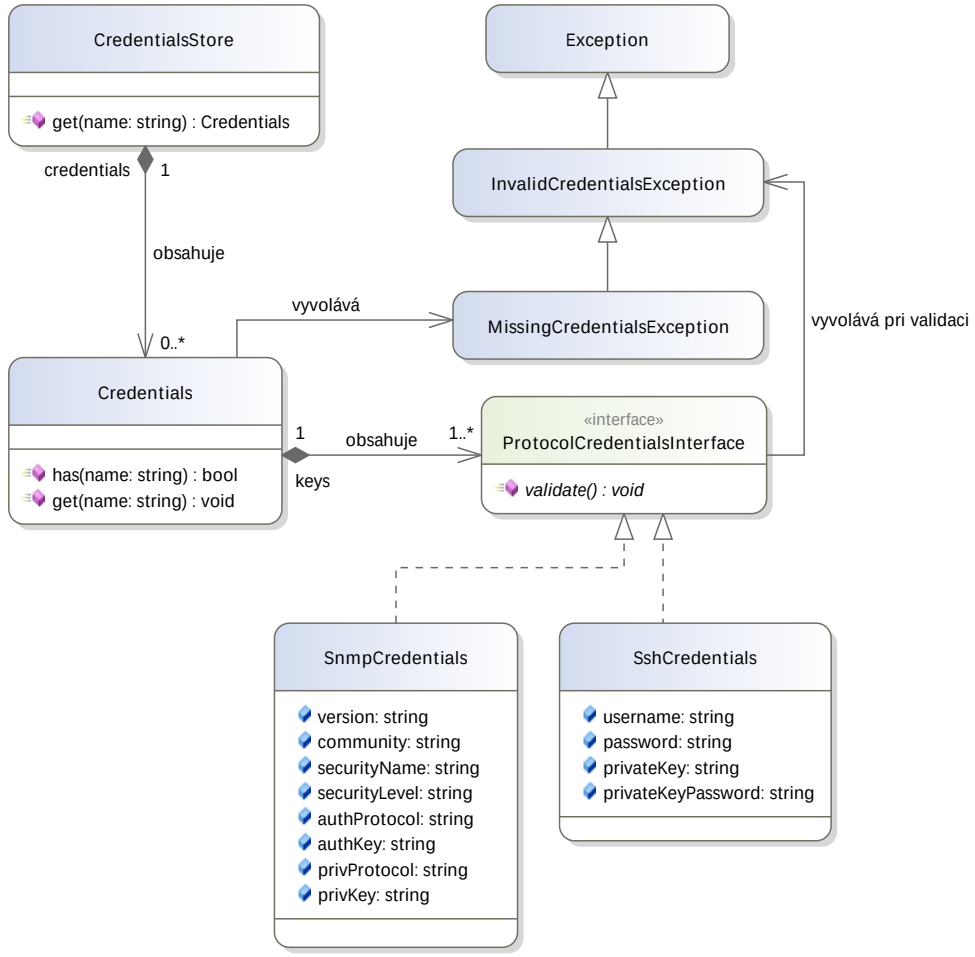
```
knet:
  credentials:
    mksw:
      snmp:
        community: SNMPCommunityExample
      ssh:
        username: knet
        password: SecretPassword
    bksw:
      snmp:
        version: 3
        securityName: snmpv3user
        securityLevel: authNoPriv
        authProtocol: SHA
        authKey: VerySecretKey
```

Tato sekce definuje dvě pojmenované sady parametrů: `mksw` a `bksw`. Sada `mksw` obsahuje přístupové údaje pro protokoly SNMP a SSH, sada `bksw` pouze údaje pro SNMP. U každého přepínače je pak v tabulce `switches` uveden název sady, který má být použit pro připojení (podrobnosti jsou uvedeny v sekci 8.2). Třída `CredentialsStore` tyto údaje mapuje na objekty v PHP.

Pro každou sadu parametrů vytváří třída `CredentialsStore` instanci třídy `Credentials`. Do této instance pak vkládá instance tříd, které implementují rozhraní `Credentials\ProtocolCredentialsInterface`. Použitá třída se vybírá na základě uvedeného protokolu – pro protokol identifikovaný řetězcem `snmp` se použije třída `Credentials\SnmpCredentials`, pro `ssh` třída `Credentials\SshCredentials`. Název třídy je sestavován dynamicky, je tedy možné přidat další protokoly pouhým vytvořením třídy bez nutnosti její registrace v aplikaci.

Každá třída implementuje validaci, zda jsou uvedeny všechny údaje potřebné pro připojení. Jednotlivé podklíče jsou pak přímo mapovány na veřejné vlastnosti těchto tříd.

Uvedené třídy jsou znázorněny v UML diagramu na obrázku 9.5.



Obrázek 9.5: UML diagram tříd přístupových údajů k síťovým prvkům.

Pro načítání dat pomocí SNMP používají konektory třídu `SnmpWrapper`. Tato třída poskytuje obálku nad třídou SNMP, jež poskytuje PHP rozšíření *snmp*.

`SnmpWrapper` dokáže načíst přístupové údaje z objektu `SnmpCredentials` a správně nakonfigurovat spojení. Implementuje také Community Indexing a VLAN kontext uvedený v sekci 7.4.

9.4 Řídící vrstva

Řídící vrstva je tvořena kontroléry, které jsou v každém modulu umístěny v namespace `Controller`. Každý kontrolér se skládá z několika akcí – metod. Metoda akce je označena anotací `@Route`:

```
/**
 * @Route(path="/hosts/{id}/edit", name="host_edit")
 */
public function editAction(Host $host, Request $request)
{
        // ...
}
```

Parametr `path` určuje URL adresu dané akce. Adresa může obsahovat pojmenované parametry, které se uvádí ve složených závorkách. V anotaci je možné uvést v parametru `name` také název, pod kterým je možné pro danou akci vygenerovat URL adresu ze šablony.

Symfony ve spojitosti s Doctrine podporuje dohledávání entit na základě parametrů v URL adrese. Pokud je například v argumentech akce uveden parametr typu `Host` (entita `KNetBundle\Entity\Host`) a v URL adrese parametr `id`, který je zároveň i identifikátorem dané entity, dokáže při mapování parametrů najít příslušnou instanci v databázi na základě příslušného ID.

Pokud entita s daným ID neexistuje, je vyvolána chyba 404. Tohoto chování je v aplikaci často využíváno, neboť zpřehledňuje kód kontroléru a odstraňuje z něj repetitivní kód pro vyhledávání entity v databázi.

Metoda akce vrací instanci třídy `Symfony\Component\HttpFoundation\Response`. Tou je typicky objekt s HTML kódem vyrenderované šablony. Více informací o kontrolérech je k dispozici v The Symfony Book, kapitola Controller [33].

9.5 Prezentační vrstva a lokalizace

Poslední akcí kontroléru je typicky volání metody `render()`:

```
return $this->render('@KNet/portal/hosts/edit.html.twig', [
        'host' => $host,
        'form' => $form->createView(),
]);
```

Toto volání zajistí načtení a vyrenderování zadané šablony. Šablony jsou uloženy v adresáři `Resources/views`. Popis šablonovacího systému Twig je k dispozici v oficiální dokumentaci [35].

Šablonovací jazyk Twig je možné rozšířit o vlastní funkce, filtry proměnných, vlastní makra a dokonce i o vlastní konstrukce jazyka. Ve třídě `KNetBundle\Twig\Extension\KNetExtension` je definováno několik vlastních funkcí a filtrů. Jednotlivé filtry jsou zdokumentovány v API dokumentaci.

Aplikace je vícejazyčná, řetězce pro uživatelské rozhraní jsou uloženy ve složce `KNetBundle/Resource/translations`. Řetězce jsou ve strukturovaném souboru YAML. Na jednotlivé klíče se lze odkazovat uvedením celé cesty v souboru. Více informací o překladech je k dispozici v The Symfony Book, kapitole Translation [33].

9.6 Konfigurace a předávání závislostí

Jednou ze základních komponent Symfony je *DepencencyInjection*. Tato komponenta umožňuje vytvořit tzv. Service Container. Jde o objekt, který vytváří a poskytuje nakonfigurované instance Services – služeb. Service je objekt, který poskytuje operace pro zajištění určité úlohy, například připojení k databázi, zasílání e-mailů či získávání informací ze síťových prvků.

Ostatní části aplikace tak mohou snadno získat příslušnou instanci bez toho, aby ji samy musely vytvářet a zajišťovat její konfiguraci. Service Container při prvním použití zajistí její vytvoření na základě konfiguračního souboru. Pokud má služba závislosti na dalších službách (například třídy servisní vrstvy modelu mají závislost na třídě `EntityManager`), zajistí Service Container vytvoření i těchto závislostí a předá je dané službě.

V aplikaci má každý bundle svůj konfigurační soubor, který obsahuje definice jednotlivých služeb. Ten je umístěn v `Resources/config/services.yml`. Definice služby může mít následující podobu:

```
knet.ldap_connector:
  class: KNetBundle\Connector\Ldap\LdapConnector
  arguments: [%ldap_uri%, %ldap_base%, %ldap_binddn%,
              %ldap_password%, %ldap_sasl_mech%,
              %ldap_sasl_realm%]
```

Tímto je definována služba s názvem `knet.ldap_connector`. Služba je instancí třídy `KNetBundle\Connector\Ldap\LdapConnector` a jejímu konstrukturu budou předány argumenty uvedené v poli arguments. Argumenty ve tvaru `%<parametr>%` jsou zástupné symboly pro parametry uvedené v konfiguračních souborech v sekci parameters, konektoru se tak předají nakonfigurované přístupové údaje.

Jako parametr lze zadat i název služby, a to zápisem `@<nazev>`. Výše uvedenou službu by bylo možné předat zápisem `@knet.ldap_connector`. Pokud má služba závislost na ostatních službách, není nutné je deklarovat v konfiguračním souboru. U služby je možné uvést příznak `autowire`:

```
knet.ldap_user_provider:
  class: KNetBundle\Security\LdapUserProvider
  autowire: true
```

Při hledání závislostí se použije signatura konstruktoru. V případě třídy `LdapUserProvider` má konstruktor následující parametry:

```
public function __construct(LdapConnector $ldapConnector,
                            UserManager $userManager,
                            AdminManager $adminManager,
                            DormitoryManager $dormitoryManager)
```

Service Container při instancování této služby najde a vytvoří všechny služby uvedených typů a předá je jako parametry konstruktoru. Vyhodnocování probíhá rekurzivně, Container provádí detekci smyček, takže pokus o vytvoření cyklické závislosti skončí chybou.

V kontroléru je možné ke službám přistupovat pomocí metody `get($id)`, kde `$id` je název příslušné služby.

Služby a závislosti mezi nimi se nesestavují při každém požadavku. Při první inicializaci aplikace se sestaví třída, která obsahuje kód pro vytvoření všech deklarovaných služeb. Tato třída se uloží do složky `var/cache` a aplikace ji využívá při zpracování všech dalších požadavků.

Více informací o předávání služeb je k dispozici v The Symfony Book, kapitole Service Container [33].

9.7 Uživatelské rozhraní

Uživatelské rozhraní a všechny jeho kontroléry jsou obsaženy v `KNetBundle`. Bundle také obsahuje všechny třídy datové vrstvy. Adresářová struktura tohoto bundle je následující:

Command: třídy příkazů pro rozhraní příkazové řádky, které je možné spustit pomocí skriptu bin/console.

Connector: připojení k externím systémům.

- **Cisco:** konektory pro síťové prvky společnosti Cisco.
- **Credentials:** třídy reprezentující přístupové údaje.
- **Ldap:** připojení k adresářové službě LDAP.

Controller:

- **Guest:** uživatelské rozhraní pro nepřihlášené uživatele a pro registrace bez Orion konta.
- **Portal:** uživatelské rozhraní pro přihlášené uživatele – studenty s aktivním Orion kontem, lokální správce a zaměstnance CIV.

DependencyInjection: třídy pro konfiguraci aplikace.

Doctrine: pomocné třídy pro Doctrine 2. Jedná se zejména o třídy, které rozšiřují jazyk DQL o funkce specifické pro databázi PostgreSQL.

- **DqlFunction:** vlastní funkce jazyka DQL.
- **Type:** definice datových typů v PostgreSQL. Třídy jsou potřebné, aby tyto typy bylo možné zpracovat a namapovat na typy v jazyce PHP.

Entity: datové entity.

EventListener: třídy s kódem, který má být vyvolán při určitých událostech v aplikaci. Typicky jde o zpracování událostí, které jsou vyvolány na začátku zpracování požadavku.

Form: definice formulářů.

HostList, Incident, Log, User: třídy pro zpracování entitních tříd, typicky pro sestavení seznamu.

Model: třídy servisní vrstvy modelu.

Registration: třídy automatických kontrol, které jsou prováděny při schvalování registrací.

Resources:

- **config:** konfigurace služeb.
- **translation:** soubory s lokalizačními řetězci.
- **views:** šablony.

Security: třídy pro zabezpečení aplikace – zpracování informací ze systému WebAuth, načítání informací o uživatelích, kontrola oprávnění.

Twig: rozšíření šablonovacího systému.

Validation: rozšiřující validační pravidla.

9.7.1 Kaskádové styly uživatelského rozhraní

Kaskádové styly (CSS, Cascading Style Sheets) aplikace jsou uloženy v adresáři `web/css`. Jsou psány v jazyce SCSS (Sassy CSS), což je rozšíření standardního CSS. SCSS do kaskádových stylů přidává proměnné, možnost vnořených selektorů a další vlastnosti, které snižují repetici stejného kódu a usnadňují organizaci jednotlivých pravidel.

Před odesláním stylů do prohlížeče je potřeba soubory zkompilovat do platného CSS. To je zajištěno knihovnami Assetic a scssphp.

Knihovna Assetic automatizuje zpracování CSS, JavaScriptu a dalších zdrojů používaných ve webových stránkách. Podporuje slučování více souborů a jejich komprimaci, což minimalizuje objem přenášených dat a počet HTTP požadavků potřebných pro načtení stránky. Umožňuje také aplikaci filtrů na slučované soubory, jedním z těchto filtrů je i filtr SCSS využívající knihovnu scssphp. Knihovna scssphp je implementací jazyka SCSS v PHP.

Styly jsou organizovány do souborů podle vizuálních komponent. Přímo v kořenové složce `web/css` se nachází soubor `style.scss`, který vkládá všechny ostatní soubory.

V kořenové složce se nacházejí následující soubory s globálními definicemi:

`_colors.scss`: proměnné s barvami uživatelského rozhraní.

`_mixnis.scss`: definice tzv. mixins. Jedná se o parametrické šablony, které lze vkládat do ostatních stylů.

`_layout.scss`: základní rozložení stránky a definice písem.

`_elements.scss`: styly pro základní HTML elementy: nadpisy, odstavce, odkazy, seznamy a další.

`_forms.scss`: styly formulářů.

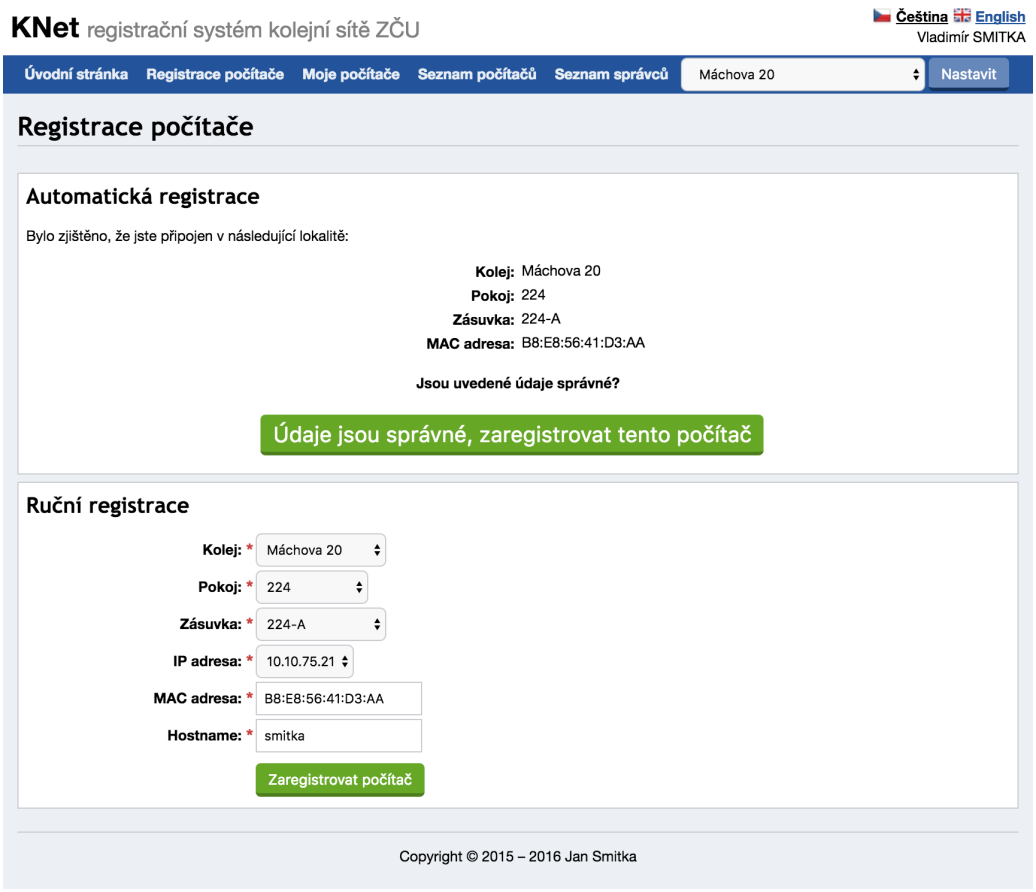
`_tables.scss`: styly tabulek.

V podsložce `components` jsou pak styly pro jednotlivé vizuální komponenty. Každá komponenta (hlavička, stránkování, registrační formulář apod.) má v této složce vlastní soubor se styly.

Navržené uživatelské rozhraní vychází z jednotného stylu Západočeské univerzity v Plzni. Ukázka registračního formuláře je uvedena na obrázku 9.6.

9.7.2 JavaScript pro uživatelské rozhraní

Pro vytvoření dynamického uživatelského rozhraní byl použit JavaScript. V jeho zpracování, zvláště pak v manipulaci s DOM (Document Object Model), jsou mezi prohlížeči rozdíly. Abychom zajistili správné zpracování



Obrázek 9.6: Ukázka rozhraní rozhraní registračního systému.

JavaScriptu napříč prohlížeči, je možné použít pomocnou knihovnu, která poskytne abstrakci nad DOM.

V uživatelském rozhraní je použita knihovna jQuery[5]. Jedná se o velice populární knihovnu, která poskytuje funkce pro manipulaci s dokumentem a také jednotné zpracování událostí, zasílání a zpracování asynchronních požadavků pomocí technologie AJAX (Asynchronous JavaScript and XML).

Pro knihovnu je také dostupná celá řada hotových uživatelských prvků, které je možné snadno integrovat do aplikace. V aplikaci je použita knihovna Ajax Autocomplete for jQuery[6], která poskytuje automatické našeptávání možností při zadávání textu.

Všechny skripty jsou umístěny ve složce `www/js`.

[5]https://jquery.com/

[6]https://www.devbridge.com/sourcery/components/jquery-autocomplete/

9.8 Přímá správa v databázi

Při provozu portálu může vzniknout potřeba provést některé operace hromadně, například odstranit registrace většího počtu uživatelů dle určitých kritérií. Protože se jedná o specifické situace, které navíc bude provádět pouze hlavní správce portálu, byly vytvořeny v databázi pomocné objekty. Jejich popis je uveden v příloze D.

9.9 API a integrace

V této sekci budou uvedeny prostředky pro integraci v obou směrech: jak z externích systémů do systémů Knet pomocí API, tak zpětná synchronizace ze systému Knet do externích systémů.

9.9.1 API pro přístup do systému Knet

Jak již bylo uvedeno v oddíle 7.3, pro integraci s ostatními systémy poskytuje systém HTTP REST API. Toto API bylo navrženo hlavně pro potřeby aplikace Mysphere2 a síťových sond, mohou jej však využívat i další aplikace.

HTTP REST API je možné formálně popsat. Pro popis existuje několik jazyků. Z nejrozšířenějších jmenujme například WADL (Web Application Description Language)[7], RAML (RESTful API Modeling Language)[8] a OpenAPI[9] (dříve Swagger).

Kolem každého z uvedených jazyků je vybudován ekosystém nástrojů, které z formální specifikace umožňují vygenerovat dokumentaci, klientské knihovny a v některých případech i kostru serverové implementace.

Po srovnání možností jednotlivých řešení a dostupných nástrojů byl vybrán jazyk OpenAPI. Popis API v tomto jazyce využívá syntaxi YAML, která je snadno čitelná a srozumitelná, oproti WADL, které používá poměrně složitou syntaxi XML. Jazyk RAML též využívá YAML.

Pro OpenAPI je dostupný oficiálně podporovaný nástroj Swagger Code Generator[10], který umožňuje generovat kód pro celou řadu jazyků. Oproti tomu je ekosystém nástrojů pro zpracování jazyka RAML poměrně roztříštěný. Například pro generování klientské knihovny pro jazyk PHP je použit jiný nástroj[11] než pro vygenerování knihovny pro Ruby[12], oba nástroje jsou navíc

[7] https://wadl.java.net/
[8] http://raml.org/
[9] https://openapis.org/
[10] https://github.com/swagger-api/swagger-codegen
[11] https://github.com/jayS-de/raml-php-generator
[12] https://github.com/zlx/raml-ruby-client-generator

vyvíjeny komunitně nezávisle na sobě. Hlavní stránka projektu WADL uvádí pouze generátor wadl2java[13], který generuje klientské třídy pouze v jazyce Java.

Ověřování pro přístup k API probíhá pomocí OAuth2 [, RFC6749] konkrétně pomocí schématu *Client Credentials*. Klient musí mít nejprve vygenerované své *Client ID* a *Client Secret*. Pomocí těchto údajů může získat přístupový klíč (Access Token) k API, kterým se ověří pro další volání.

Client ID a *Client Secret* je pro každého klienta API unikátní. Ke každému klientovi je také vytvořena jeho systémová identita – záznam v tabulce `users` s typem nastaveným na `system`. Pod touto identitou budou vkládány záznamy do logů a vkládány nové komentáře. K identitě jsou také přiřazena přístupová oprávnění stejně jako v případě lokálních správců.

Požadavek na vytvoření Access Token zašle klient metodou `POST` na adresu `/api/oauth2/token`. Svoji identitu udává stejně jako v případě ověřování pomocí metody HTTP Basic authenticaton. Jako své jméno klient uvádí *Client ID* jako heslo *Client Secret*. V požadavku je uvedena hlavička Authorization, jejíž hodnota obsahuje řetězec „Basic “ následovaný řetězcem `<ClientID>:<ClientSecret>` zakódovaným pomocí base64. Pro *Client ID* `SomeClient` a *Client Secret* `SomeSecret` bude mít hlavička následující hodnotu:

```
Authorization: Basic U29tZUNsaWVudDpTb211U2VjcmV0
```

Požadavek musí být ve formátu `application/x-www-form-urlencoded`, v jeho těle musí být uvedený parametr `grant_type` s hodnotou `client_credentials`.

Volitelně může klient zadat parametr `scope`, kterým může určit rozsah požadovaného přístupu. Hodnotou parametru je seznam řetězců oddělených mezerami. Jednotlivé řetězce identifikují úrovně přístupu. API podporuje následující úrovně:

`read`: přístup pouze pro čtení.

`write`: přístup pro zápis. Uvedení tohoto parametru neimplikuje možnost čtení!

`impersonate`: možnost provádět úkony s identitou jiného uživatele – bude vysvětleno dále v textu.

Příklad HTTP požadavku na získání přístupového klíče:

[13] https://wadl.java.net/wadl2java.html

```
POST /api/oauth2/token HTTP/1.1
Authorization: Basic WnNmV0xEV3FzVTpLTlFicFpRdmRM
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&scope=read write
```

Odpověď serveru je ve formátu JSON. Obsahuje vygenerovaný Access Token, jeho typ a délku jeho platnosti:

```
{
        "access_token": "cNDgMn6PvpxFGTXZIm8o9Mt34...",
        "token_type": "Bearer",
        "expires_in": 3600,
        "scope": "read"
}
```

Přístupový klíč typu `Bearer` klient pouze vkládá do dalších požadavků v hlavičce `Authorization`. Hodnota obsahuje řetězec „Bearer “ následovaný hodnotou klíče. Příklad požadavku je následující:

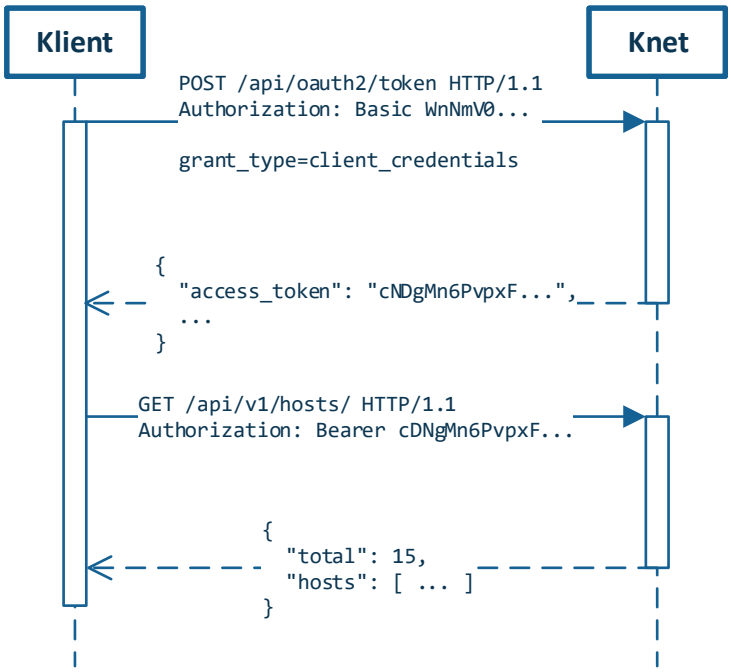
```
GET /api/v1/hosts/ HTTP/1.1
Authorization: Bearer cNDgMn6PvpxFGTXZIm8o9Mt34...
```

Specifikace OAuth2 uvádí, že všechny požadavky musí být provedeny prostřednictvím zabezpečeného kanálu, aby nemohlo dojít ke kompromitaci tajných klíčů. Celý postup získání a použití Access Token je znázorněn na obrázku 9.7.
Pokud je klíč vytvořen se scope `impersonate`, má klient možnost provádět volání s identitou jiného uživatele. Oprávnění pro přístup se nezmění, všechny akce však budou do logu zaznamenány stejně, jako by je provedl přímo uživatel. Obdobně pak všechny přidané komentáře budou uloženy pod identitou jiného uživatele.

K využití této funkce musí klient přidat do HTTP požadavku hlavičku `X-User-Uid`. Jako hodnotu uvede přihlašovací jméno uživatele, pod jehož identitou má být operace vykonána.

Přístupový klíč má omezenou platnost. Po vypršení platnosti musí klient znovu požádat o vytvoření nového klíče.

Samotné API je k dispozici na adrese `/api/v1/`. Jeho kompletní dokumentace, včetně formátu všech požadavků a odpovědí, je součástí dokumentace projektu a je také umístěna na přiloženém CD. Zde uvedeme pouze stručný přehled nejdůležitějších volání. Volání jsou uvedena v tabulce 9.2.



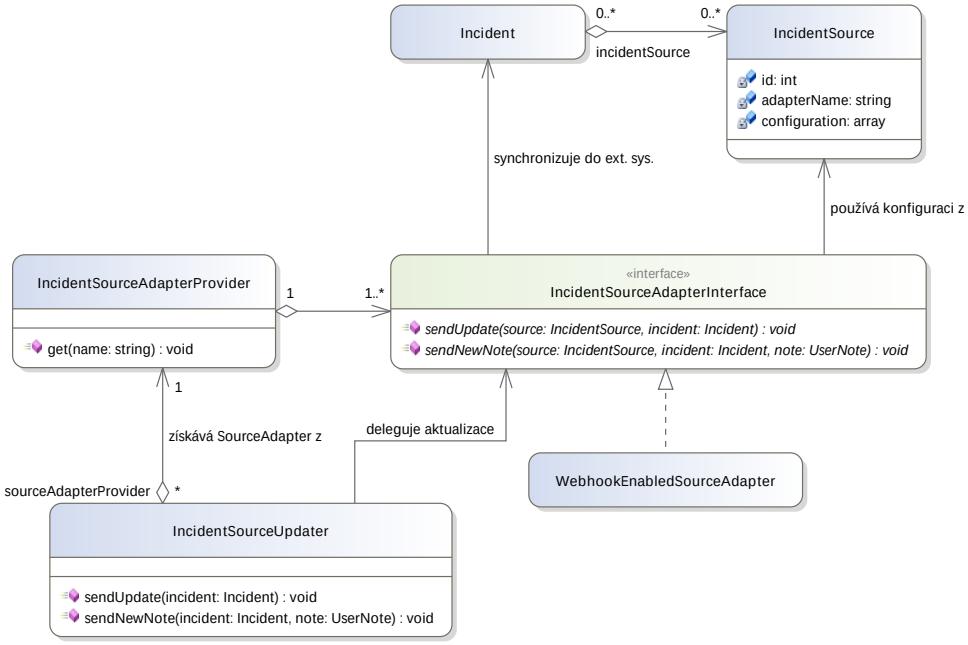
Obrázek 9.7: Postup získání přístupového klíče k REST API a jeho použití.

Pro ověření funkčnosti API byla připravena sada automatických testů s využitím nástroje PHPUnit. Celkem bylo vytvořeno 83 testovacích scénářů, které testují všechna volání API. Volání jsou simulována nástrojem BrowserKit, který umožňuje aplikaci předat stejná vstupní data, jež by byla vytvořena ze skutečného požadavku, a nevyžadují tedy nakonfigurovaný webový server. Postup spuštění testů je popsán v příloze A.1.2 Spuštění testů.

9.9.2 Synchronizace údajů zpět ze systému Knet

Externí systémy mohou mít různá API, pomocí kterých do nich lze zaslat aktualizovaná data. Pro synchronizaci dat v opačném směru bylo proto potřeba navrhnout řešení, které umožní snadno přidat konektor pro další systémy. Protože jediná data, která mohou být přes API vytvořena, jsou incidenty, týká se tato synchronizace pouze nich. Pokud externí systém potřebuje udržovat jiná data, musí použít jejich periodické stahování prostřednictvím API.

Při návrhu řešení byl stanoven předpoklad, že jediný systém, který potřebuje dostávat informace o změnách v incidentu, je ten systém, který daný incident založil. Pokud tedy lokální P2P sonda na kolejích Máchova založí incident, neměl by tento incident být propsán do systému Mysphere2 nebo



Obrázek 9.8: UML diagram rozhraní pro aktualizaci incidentů v externích systémech.

do ostatních síťových sond.

Proto je u každého incidentu uvedena reference na zdroj, který jej založil. U každého zdroje je uveden jeho typ, tj. třída, která bude zajišťovat synchronizaci, a volitelná konfigurace ve formátu JSON. Tato konfigurace je specifická pro daný zdroj. Synchronizační třída musí implementovat rozhraní `IncidentSourceAdapterInterface`. Toto rozhraní je znázorněno v UML diagramu na obrázku 9.8 spolu se souvisejícími třídami.

V systému je aktuálně implementován systém pro zasílání aktualizací pomocí tzv. webhooks. Pokud v systému dojde k určité události, v případě registračního portálu jde o úpravu incidentu či o přidání nového komentáře, je zasláno HTTP volání na předem nakonfigurovanou adresu.

Synchronizaci tímto způsobem je možné nakonfigurovat vložením řádku do tabulky `incident_sources`, kde jako `adapter_name` bude uveden řetězec `webhook`. V poli `configuration` je možné použít parametry uvedené v tabulce 9.1. Aplikace také obsahuje webové rozhraní pro konfiguraci zdrojů, které umožňuje parametry zadat prostřednictvím jednoduchého formuláře.

Zasílaný požadavek je typu `POST`. V hlavičce `X-Knet-Event` je uveden typ události: retězec `incident_update` pro aktualizaci incidentu, `incident_new_note` pro přidání nového komentáře. Pokud bylo nakonfigurováno sdílené heslo, je v hlavičce `X-Knet-Signature` uveden HMAC podpis zprávy.

Tělo zprávy obsahuje objekt ve formátu JSON. Pole objektu se liší dle typu zprávy. Pro událost `incident_update` obsahuje následující pole:

`id`: ID incidentu
`type`: Objekt reprezentující typ incidentu.
- **`id`:** ID typu.
- **`identifier`:** Identifikátor typu.
- **`title`:** Název daného typu.

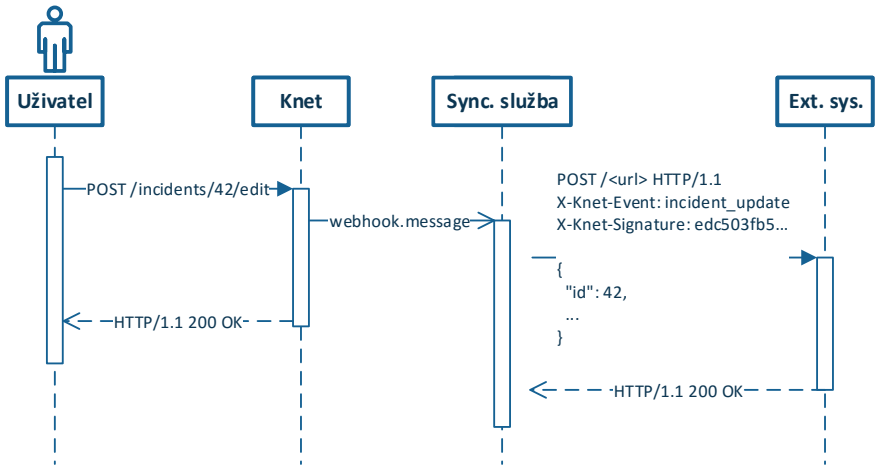
`internalDescription`: Interní popis incidentu.
`action`: Akce incidentu.
`blockActive`: Příznak, zda je uživatel zablokován.
`blockStartAt`: Datum a čas začátku blokace (ISO 8601).
`blockEndAt`: Datum a čas konce blokace (ISO 8601).
`resolved`: Příznak, zda byl incident vyřešen.

V případě zprávy `incident_new_note`:

`incidentId`: ID incidentu.
`id`: ID komentáře.
`author`: Objekt reprezentující autora komentáře:
- **`id`:** ID uživatele.
- **`uid`:** Přihlašovací jméno.
- **`firstName`:** Jméno.
- **`lastName`:** Příjmení.
- **`email`:** E-mail.

`text`: Text komentáře.
`createdAt`: Datum a čas vytvoření incidentu (ISO 8601).

Webové rozhraní nezasílá požadavek přímo, ale data zpráv vkládá do fronty podobně jako v případě konfigurace síťových služeb. Zaslání požadavku zajišťuje samostatná služba. Průběh synchronizace je znázorněn na obrázku 9.9.



Obrázek 9.9: Zasílání aktualizací do externího systému pomocí webhooks.

Název parametru	**Popis**	**Výchozí hodnota**
`url`	URL adresa, na kterou má být provedeno volání.	povinný parametr
`secret`	Sdílené heslo, pomocí kterého má být podepsáno tělo požadavku. Podpis je vytvářen metodou HMAC. Pokud není uvedeno, zpráva nebude podepsána.	žádná, nepovinné
`algo`	Hashovací algoritmus, pomocí kterého má být vytvořen podpis těla zprávy. Možné hodnoty: md5, sha1, sha224, sha256, sha384, sha512	sha512
`event_header`	Název HTTP hlavičky s typem události.	X-Knet-Event
`signature_header`	Název HTTP hlavičky s podpisem zprávy.	X-Knet-Signature

Tabulka 9.1: Konfigurační parametry pro webhooks.

Volání	**Popis**
`GET /hosts/`	Získá seznam registrací zařízení.
`GET /hosts/<id>`	Získá informaci o registraci s uvedeným ID.
`GET /hosts/find-by-ip`	Najde registraci zařízení dle IP adresy.
`GET /users/`	Seznam registrovaných uživatelů.
`GET /users/<id>`	Získá informace o uživateli se zadaným ID.
`GET /users/find-by-uid`	Najde uživatele dle přihlašovacího jména.
`GET /incidents/`	Získá seznam incidentů.
`POST /incidents/`	Vytvoří nový incident.
`GET /incidents/<id>`	Získá informace o incidentu se zadaným ID.
`PUT /incidents/<id>`	Aktualizuje incident se zadaným ID.
`GET /incidents/<id>/notes/`	Získá seznam poznámek incidentu se zadnaým ID.
`GET /incident-types/`	Získá seznam typů incidentů.
`GET /dormitories/`	Získá seznam kolejí.
`GET /dormitories/<id>/rooms/`	Získá seznam pokojů na koleji se zadaným ID.
`GET /rooms/<id>/sockets`	Získá seznam zásuvek v pokoji s uvedeným ID.

Tabulka 9.2: Přehled dostupných volání API.

10 Síťové služby a jejich konfigurace

Pro konfiguraci síťových služeb byly implementovány služby, které jejich konfiguraci provádějí na pozadí, nezávisle na zpracování požadavků od uživatelů. Pro jejich implementaci byl zvolen jazyk Python, neboť tento jazyk obsahuje celou řadu vestavěných knihoven, pomocí kterých je možné interagovat s operačním systémem. Lze tak snadno pracovat se sockety, čekat na dostupnost dat voláním `select()` a řadu dalších funkcí. K dispozici jsou i knihovny třetích stran pro práci s dalšími protokoly, například protokolem SSH.

Z důvodu udržení stávajícího prostředí byla zvolena verze Python 2.7, která je výchozí verzí v aktuálně používaných systémech Debian Wheezy, Debian Jessie i v připravovaném Debian Stretch. Skripty jsou kompatibilní i s verzemi Python 3.x.

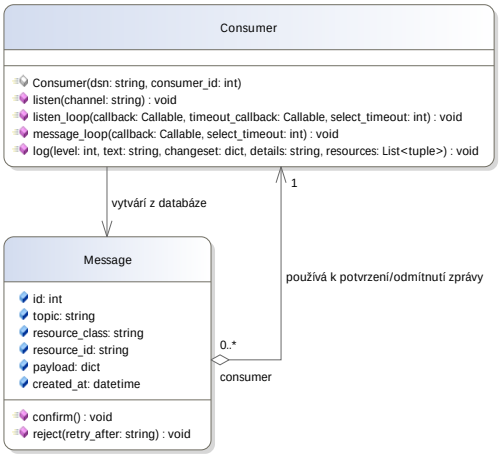
Pro všechny konfigurační služby byla implementována společná knihovna, která zajišťuje výběr zpráv z fronty. Knihovně je předána funkce, která má být spuštěna pro příchozí zprávu. Knihovna zajistí navázání spojení, zavolání `LISTEN` a spuštění smyčky čekající na příchod zpráv.

10.1 Knihovna pro práci s frontou zpráv

Knihovna je dodávána jako modul `knet_svc` v balíčku, kde jsou všechny ostatní konfigurační služby. Hlavním prvkem knihovny je třída `Consumer`, která je znázorněna v UML diagramu na obrázku 10.1. Třídě se při vytváření předají parametry pro připojení k databázi a ID konzumenta v databázi. Zpracování zpráv je zahájeno voláním `message_loop(callback)`. Tato metoda zablokuje vlákno aplikace až do přijetí zprávy.

Funkce `callback` je vyvolána při obdržení zprávy. Tato funkce musí přijímat jediný argument: objekt typu `Message`, který obsahuje všechny informace o zprávě. Služba zpracuje zprávu a pokud je zpracování úspěšně dokončeno, potvrdí zprávu zavoláním `confirm()` nad objektem zprávy. Pokud ne, zprávu odmítne zavoláním `retry()`. Metodě je možné volitelně zadat interval, za který má být zpráva znovu zpracována. Interval je zadán jako řetězec ve formátu, který používá v PostgreSQL datový typ INTERVAL[1]. Pokud není interval zadán, použije se výchozí hodnota 120 sekund.

[1] http://www.postgresql.org/docs/9.5/static/datatype-datetime.html



Obrázek 10.1: UML diagram třídy Consumer z knihovny pro práci s frontou zpráv.

Metoda `log()` třídy Consumer zapíše danou zprávu do logu systému. Tato zpráva bude viditelná z webového rozhraní. Kromě tohoto logování by měla služba používat vlastní systém logování, který bude podrobnější a bude možné jej použít pro podrobnou diagnostiku chyb.

10.2 Opakování zpráv při neúspěchu

V případě neúspěchu je nutné zajistit, aby byla zpráva po uplynutí uvedeného časového intervalu znovu předána ke zpracování. PostgreSQL nepodporuje žádný systém plánování událostí, proto bylo prozkoumáno několik možností pro plánování.

Od verze 9.3 PostgreSQL obsahuje podporu pro tzv. background workers. Jedná se o rozšiřující knihovnu implementovanou v jazyce C, která obsahuje obslužnou rutinu spouštěnou spolu s databázovým serverem jako nový proces. Z této rutiny je možné navázat spojení s databází a pracovat s ní podobně jako libovolný jiný klient. Výhodou je, že správu tohoto procesu zajišťuje samotný databázový systém. Nevýhodou je složitá konfigurace, komplikovaný monitoring a nutnost background worker překompilovat v případě vydání nové verze.

Aby nebylo nutné rozšiřující knihovnu překompilovávat při aktualizacích a bylo možné její běh snadno monitorovat, bylo zvoleno podobné řešení, jako je používáno pro konfigurační služby.

Plánovač se spustí jako samostatná služba, která je také implementována v jazyce Python a používá stejnou knihovnu. Tato služba si udržuje frontu zpráv, které mají být naplánovány. Fronta je prioritní a zprávy jsou seřazeny

podle času, kdy mají být zpracovány. Pokud dojde k zařazení některé zpráv k opakování pokusu, je služba notifikována pomocí asynchronních notifikací (`LISTEN/NOTIFY`).

Při přijetí notifikace služba zařadí novou zprávu do prioritní fronty a uspí se do doby, než přijde další zpráva, nebo má být zpráva na vrcholu fronty zařazena ke zpracování.

10.3 Služba DHCP

Pro konfiguraci DHCP bylo zvoleno přímé připojení do MySQL databáze. V databázi DHCP serveru Kea má každá rezervace zařízení vlastní ID. Jako ID rezervace je možné použít ID registrace z tabulky hosts. Díky tomu bude možné záznam snadno aktualizovat a mazat.

Protože DHCP server Kea neumožňuje konfigurovat délku zapůjčení IP adresy pro jednotlivé rezervace, ani neumožňuje použít jinou délku zapůjčení pro adresy dynamicky přidělené z vyhrazeného rozsahu adres a pro IP adresy na základě rezervací, bylo implementováno rozšíření v jazyce C++, které umožní pro dynamicky přidělené adresy použít kratší dobu zapůjčení.

Kea umožňuje zaregistrovat vlastní funkce, které budou spuštěny v různých fázích zpracování DHCP paketů – body rozšíření. Každý bod rozšíření je identifikován vlastním řetězcem. Rozšíření využívá dva body rozšíření:

`lease4_select`: Vyvoláno při určování délky zapůjčení pro nově přidělované IP adresy.

`lease4_renew`: Vyvoláno při prodlužování zapůjčení již přidělené IP adresy.

V obou těchto rozšiřujících bodech je možné upravit zvolenou délku zapůjčení. Rozšíření kontroluje, zda je přidělená adresy z dynamicky alokovaného rozsahu. Pokud ano, nastaví délku zapůjčení na krátkou hodnotu (300s). Všechny rezervace IP adres musí proto být mimo tento dynamicky alokovaný rozsah. Kód obslužné funkce je pro oba body rozšíření stejný a je velice jednoduchý:

```
int update_lease(CalloutHandle& handle) {
        Lease4Ptr lease;
        Subnet4Ptr subnet;

        handle.getArgument("lease4", lease);
        handle.getArgument("subnet4", subnet);
```

```
        // Check if the leased address belongs to a
        // dynamic pool declared for subnet.
        PoolPtr pool = subnet->getPool(Lease::Type::TYPE_V4,
                                       lease->addr_, false);
        if (pool != nullptr) {
                // If a pool was found, shorten the lease time.
                lease->valid_lft_ = short_lease_time;
        }

        return 0;
}
```

Všechny obslužné rutiny přijímají objekt typu `CalloutHandle`. Ten obsahuje všechny parametry dané rutiny. Více informací o rozšíření DHCP serveru Kea je k dispozici v oficiální dokumentaci pro vývojáře [19].

10.4 Služba DNS

Server PowerDNS používá pro konfiguraci databázi PostgreSQL. Obsahuje tabulku domén a k nim náležících záznamů. Jednotlivé záznamy mají ID, avšak není možné jej použít k identifikaci, ke kterému zařízení náleží, protože jeden počítač může mít v databázi více záznamů – jeden A záznam, reverzní PTR záznam a pak CNAME záznamy pro všechny své aliasy. Dále jsou v databázi typicky záznamy i pro síťové prvky.

Konfigurační skript proto při svém spuštění rozšiřuje tabulku záznamů o dva sloupce:

`knet_object_type`: Typ objektu, ke kterému se záznam váže. Výčtový typ, jenž může nabývat hodnoty `host`, `host_alias`, `switch`, nebo `switch_alias`.

`knet_object_id`: ID objektu, například ID rezervace počítače.

Tyto sloupce pak používá pro aktualizaci záznamů, které náleží příslušnému objektu.

10.5 Firewall

Konfigurační služba pro firewall pomocí protokolu SSH modifikuje tabulky modulu *netfilter* pomocí příkazu `iptables`. Pro navázání SSH spojení bylo posuzováno několik knihoven pro jazyk Python, které jsou součástí balíčků operačního systému Debian:

pylibssh2[2]**:** bindings pro knihovnu libssh2. Poslední verze vyšla v roce 2011, balíček proto pravděpodobně není udržovaný.

paramiko[3]**:** implementace protokolu SSH2 v Pythonu.

pexpect[4]**:** knihovna pro spouštění procesů a zpracování jejich výstupů. Součástí je modul `pxssh`, který umožňuje navázat spojení pomocí příkazu `ssh`.

Knihovna pylibssh2 nebyla testována, neboť by v budoucnu mohly být problémy v kompatibilitě s novějšími verzemi jazyka. Zbývající knihovny byly otestovány, hlavním aspektem testování bylo chování v případě síťové chyby.

Knihovna pexpect používá utilitu `ssh` z OpenSSH. Ta je určena pro interaktivní sezení, proto se v případě výpadku sítě snaží o zotavení z chyby a spojení se snaží obnovit. To způsobí zablokování celé aplikace, a to i na několik minut.

Knihovna paramiko umožňuje definovat u všech operací s relací maximální časový limit na vykonání. Pokud tedy vzdálená strana neodpoví do tohoto časového limitu, je vyvolána chyba, kterou je možné v aplikaci snadno zpracovat a zprávu ve frontě zadat k opakování.

Zvolena tedy byla knihovna paramiko, která poskytuje větší kontrolu nad spojením a umožňuje zotavení aplikace z chyby spojení. Aplikace tak může může udržovat SSH spojení, které v případě chyby uzavře a pokusí se jej opětovně navázat. Přes toto spojení pak provádí změny v pravidlech modulu *netfilter* příkazem `iptables`.

Pravidla modulu *netfilter* nejsou při úpravách ukládána, případné změny by se při restartu serveru smazaly. Je proto nutné zajistit jejich uložení příkazem `iptables-save`. Příkaz je možné volat po každé změně a teprve poté potvrdit doručení zprávy, to by však představovalo velkou režii, zvláště pak při hromadných operacích.

Proto bylo navrženo řešení, které změny nejprve zapíše do vlastního žurnálu a teprve poté provede změny v konfiguraci modulu *netfilter*. Po restartu serveru je možné pravidla ze žurnálu obnovit. Zároveň je možné periodicky pravidla ukládat pomocí `iptables-save` a žurnál mazat, aby jeho velikost nerostla.

Řešení používá pomocné skripty, které musejí být nainstalovány na serveru s bránou. Skripty jsou implementovány také v jazyce Python a nemají žádné závislosti. Jedná se o tři skripty:

`update_host.py`: Aktualizuje žurnál a pravidla v *netfilter*. Tento skript je spouštěn vzdáleně konfigurační službou.

`restore_from_journal.py`: Obnoví pravidla ze žurnálu. Tento skript by měl být spuštěn po startu serveru.

`persist_rules.py`: Provede uložení aktuálních pravidel v *netfilter* a smazání žurnálu. Skript by měl být spouštěn periodicky, například nástrojem *cron*.

Všechny skripty mají společný konfigurační soubor `config.cfg`, ve kterém je možné konfigurovat pravidla vkládaná do *netfilter*. Vytvářený žurnál je textový soubor, kde každá řádka odpovídá jedné změně pravidel. Řádka má následující formát:

```
<id registrace>:<ip adresa registrace>:<stav registrace>
```

Stav může mít jednu z následujících hodnot:

`active`: aktivní registrace, plný přístup do vnější sítě.
`quarantine`: zablokování počítače.
`deleted`: odstranění registrace.

10.6 Synchronizační služba pro webhooks

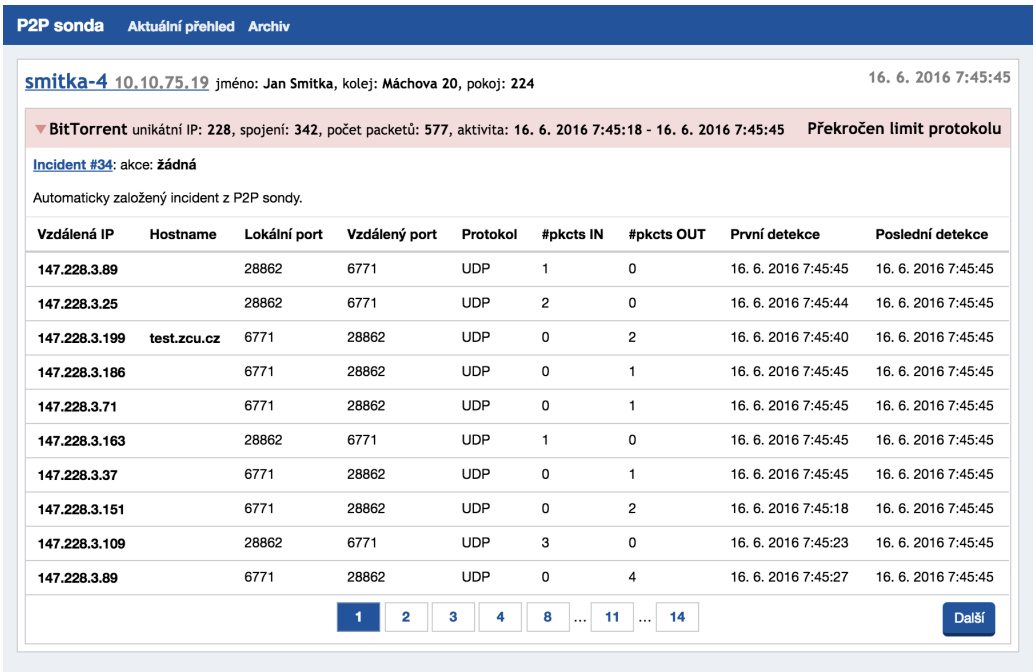
Součástí služeb je i synchronizační služba, která zajišťuje zasílání zpráv pomocí webhooks. Tato služba načítá konfiguraci externích systémů z databáze. Konfigurace je načítána při každém dotazu, takže je možné ji měnit bez nutnosti službu restartovat.

11 Rozhraní pro sondu detekce P2P

V rámci diplomové práce bylo vytvořeno webové rozhraní pro síťovou sondu detekující spojení, jež využívají P2P protokolů pro sdílení dat. Předchozí implementace rozhraní představovala jednoduchou webovou stránku, která zobrazovala posledních 500 řádek ze souboru se záznamy ze sondy ipp2p.

Nové rozhraní zaznamenává probíhající komunikaci v reálném čase. Díky integraci s registračním portálem jsou přímo v rozhraní k dispozici informace o komunikujících zařízeních a uživatelích, systém navíc zobrazuje i případné informace o probíhajícím řešení incidentů. Ukázka rozhraní je na obrázku 11.1.

V této kapitole bude popsána architektura webového rozhraní a jeho implementace.

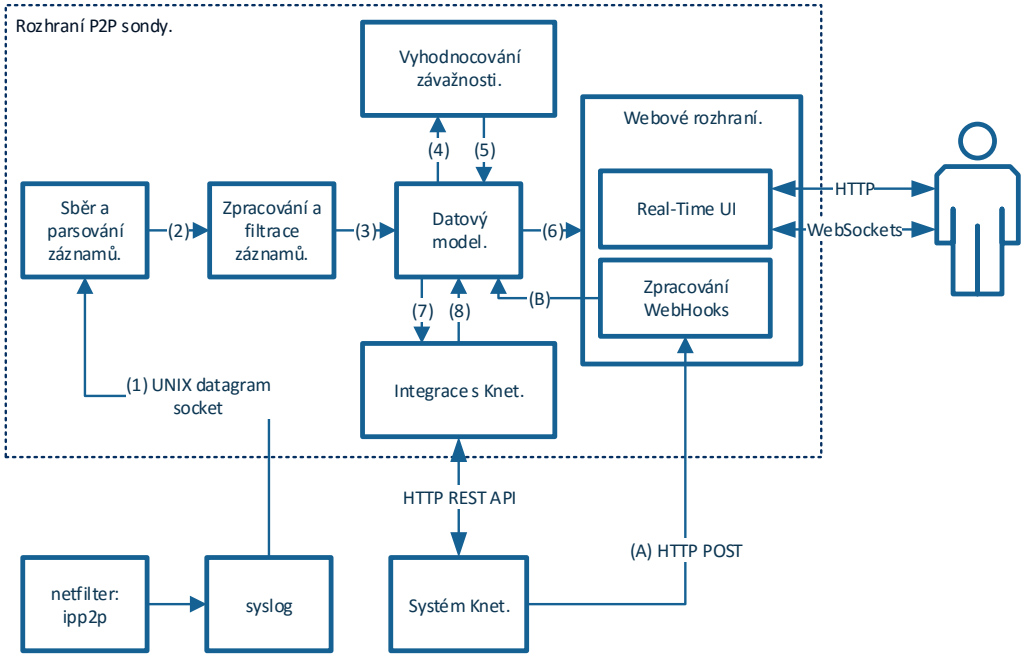


Obrázek 11.1: Webové rozhraní pro P2P sondu.

11.1 Architektura

Pro implementaci byl zvolen jazyk Python 3.4. Jazyk v této verzi obsahuje vestavěnou podporu pro asynchronní síťové operace (knihovna *asyncio*[1]), která umožňuje snadnou implementaci síťových serverů. Byla použita knihovna *aiohttp*[2], která implementuje protokol HTTP a podporu pro WebSockets.

Aplikace musí sbírat data ze sondy ipp2p. To je možné například nastavením syslog, aby záznamy z jádra zasílal na UNIX datagram socket. Architektura a průběh zpracování dat jsou znázorněny na obrázku 12.2.



Obrázek 11.2: Architektura a průběh zpracování dat v rozhraní P2P sondy.

Aplikace naslouchá na UNIX datagramovém socketu, kde přijímá záznamy z P2P. Aplikace provádí:

1. Přijmutí a parsování záznamu, které informace z řetězce zpracuje do interní reprezentace paketu.
2. Zpracování a filtrace: odstranění záznamů, které nepocházejí ze strojů v interní síti, a určení směru toku dat: zda je paket odchozí či příchozí.
3. Paket je zaznamenán do datového modelu. Jedná se o stromovou strukturu, v rámci které je datový tok seskupován podle koncové stanice,

[1] https://docs.python.org/3/library/asyncio.html
[2] http://aiohttp.readthedocs.org/en/stable/

detekovaného P2P protokolu (např. BitTorrent, eDokney) a síťových spojení. Na úrovni protokolů a spojení jsou udržovány čítače o počtech detekovaných paketů.

4. Aktualizovaný záznam o přenosu dat pomocí daného protokolu je předán k vyhodnocení, zda jsou spojení škodlivá.
5. Vyhodnocování probíhá na základě definovaných limitů. Pokud počet unikátních IP adres, počet spojení a počet detekovaných paketů přesáhne definovaný limit, jsou spojení daného zařízení vyhodnocena jako potenciálně škodlivá a stanice je označena v datovém modelu.
6. Aktualizované informace jsou předány do webového rozhraní. Záznamy jsou zobrazovány uživatelům v reálném čase.
7. Pokud jde o první detekci dané koncové stanici, je do Knetu asynchronně vyslán dotaz na vyhledání informací o daném zařízení. Pokud jsou spojení vyhodnocena jako škodlivá, je zároveň založen incident.
8. Pokud asynchronní požadavek na informace o zařízení vrátí výsledky, jsou získané informace zaznamenány do datového modelu. Nové informace se zobrazí ve webovém rozhraní.

Pokud sonda založí incident a ten je v systému Knet aktualizován, tak:

1. Systém Knet vyšle HTTP `POST` požadavek s novými informacemi.
2. Webové rozhraní požadavek zpracuje a aktualizuje informace o incidentu v datovém modelu.

Pro ukládání historických záznamů byla zvolena databáze MySQL. Ta je na strojích, které jsou ve správě lokálních správců, již nainstalována a používána pro další aplikace. V případě potřeby by bylo možné databázi snadno nahradit za PostgreSQL, neboť knihovny pro přístup k oběma databázím (aiomysql[3] a aiopg[4]) mají API odpovídající PEP (Python Enhancement Proposal) 249[5].

11.2 Implementace

Implementace využívá asynchronních Input/Output (IO) operací z knihovny *asyncio*. Při použití této knihovny je využito kooperativního multitaskingu pomocí korutin. Korutina je zvláštní funkce, která může svoji činnost v určitých místech svého kódu pozastavit a předat řízení jiné korutině.

[3]https://github.com/aio-libs/aiomysql
[4]https://github.com/aio-libs/aiopg
[5]https://www.python.org/dev/peps/pep-0249/

V případě korutin v knihovně asyncio je korutina pozastavena v okamžicích, kdy vykonává IO operace. Místo toho, aby se korutina zablokovala do dokončení dané operace, je operace naplánována k asynchronnímu vykonání (buď s využitím prostředků operačního systému, nebo v samostatném vlákně), korutina se pozastaví a je naplánována další korutina. Korutina je také pozastavena v případě, že vyvolá další korutinu.

Tímto chováním je simulováno paralelní zpracování, samotná aplikace však běží sériově. Výhodou je, že kód korutiny je velmi podobný jako u plně sériové verze programu, a není potřeba ošetřovat souběžný přístup k paměti. Výhodou také je, že odpadá režie s plánováním a přepínáním vláken.

Plánování korutin zajišťuje tzv. event loop. Jedná se o cyklus, který přijímá události z různých zdrojů (typicky dostupnost dat v komunikačním kanálu, uplynutí časového intervalu, atd.) a na základě těchto událostí plánuje spouštění korutin.

Tento přístup je velmi výhodný pro síťové aplikace, které vykonávají převážně IO operace a jejich zátěž na CPU (a tedy i množství vykonávaných instrukcí) je malá. Tímto případem je i webové rozhraní pro P2P sondu.

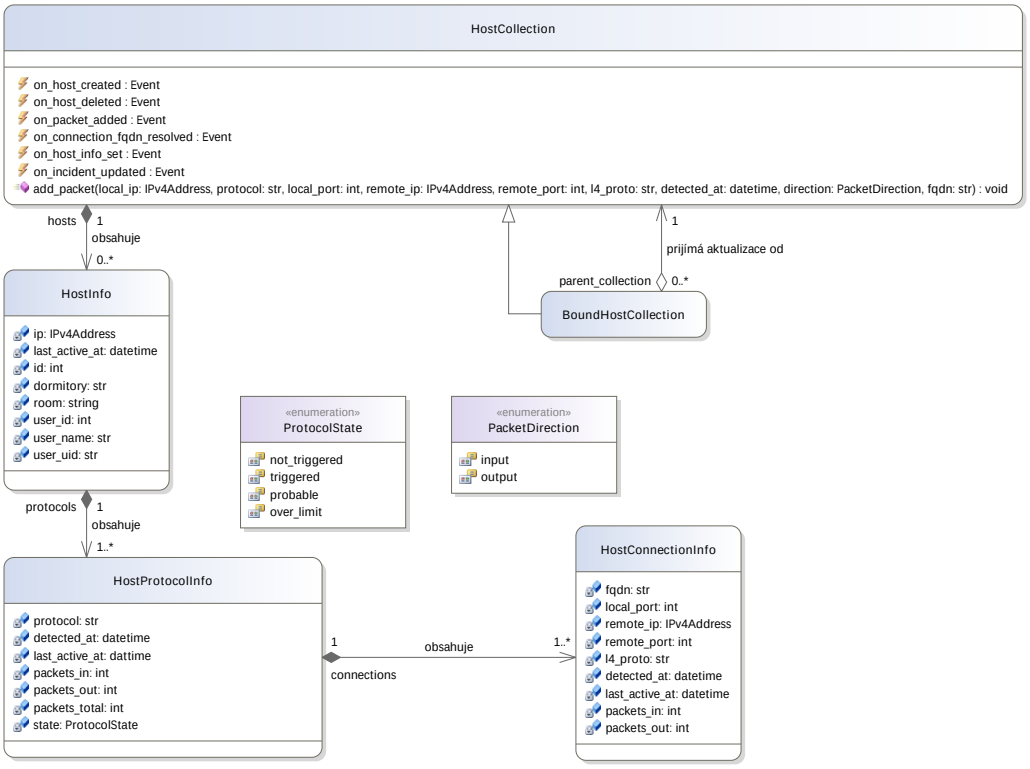
Hlavní částí aplikace je datový model, který uchovává informace o detekované komunikaci. V případě změn v tomto modelu jsou pomocí systému událostí notifikovány ostatní komponenty aplikace. V tomto oddíle popíšeme stručně datový model aplikace a poté následující moduly.

Hlavní komponentou aplikace je třída `P2PProbeApp`. Ta zajišťuje načtení konfigurace a inicializaci všech komponent aplikace.

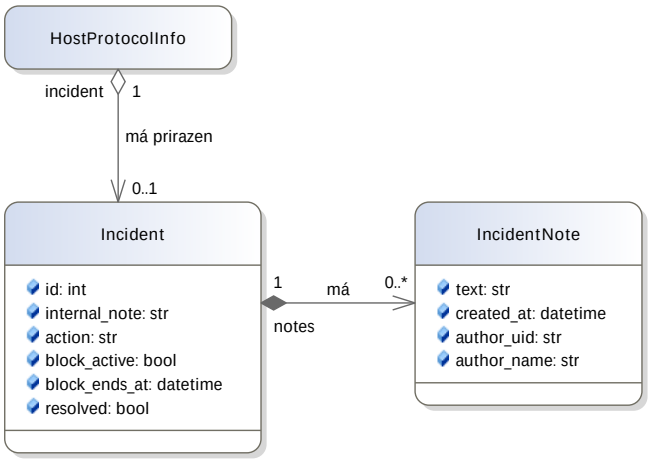
11.2.1 Datový model

Třídy datového modelu jsou znázorněny na obrázku 11.3. Zaznamenaná komunikace je seskupena podle koncových stanic (`HostInfo`), P2P protokolu (`HostProtocolInfo`) a spojení (`HostConnectionInfo`). Protože velká část komunikace v P2P sítích probíhá prostřednictvím protokolu UDP, jsou spojení lokální koncové stanice sdružována podle čtveřice (transportní protokol; lokální port; vzdálená adresa; vzdálený port).

Kolekce `HostCollection` obsahuje informace o všech koncových stanicích, u kterých byla detekována P2P komunikace. Při vkládání nového paketu je kontrolováno, zda již daná koncová stanice v seznamu existuje. Pokud ne, je jí vytvořen záznam. Poté jsou postupně inkrementovány čítače a aktualizována časová razítka u protokolu a u spojení. Doposud neexistující záznamy jsou opět vytvořeny.



Obrázek 11.3: UML diagram datového modelu P2P sondy.



Obrázek 11.4: UML diagram tříd popisujících incidenty v rozhraní P2P sondy.

K informacím o protokolu může být vázána instance třídy `Incident`, která obsahuje detaily incidentu. Tato třída je znázorněna na obrázku 11.4.

Změny v kolekci vyvolávají události, na které je možné reagovat. Toho využívá kolekce `BoundHostCollection`, která replikuje změny v hlavní kolekci. Tuto kolekci využívá webové rozhraní k dávkovému zasílání aktualizací klientům. Těm nejsou zasílány jednotlivé detekované pakety, ale souhrnné informace o změnách za určitý časový úsek (ve výchozím nastavení za 0.5 sekundy). Po odeslání změn je kolekce vymazána a systém tak začne sestavovat novou dávku.

11.2.2 Moduly pro zpracování záznamů

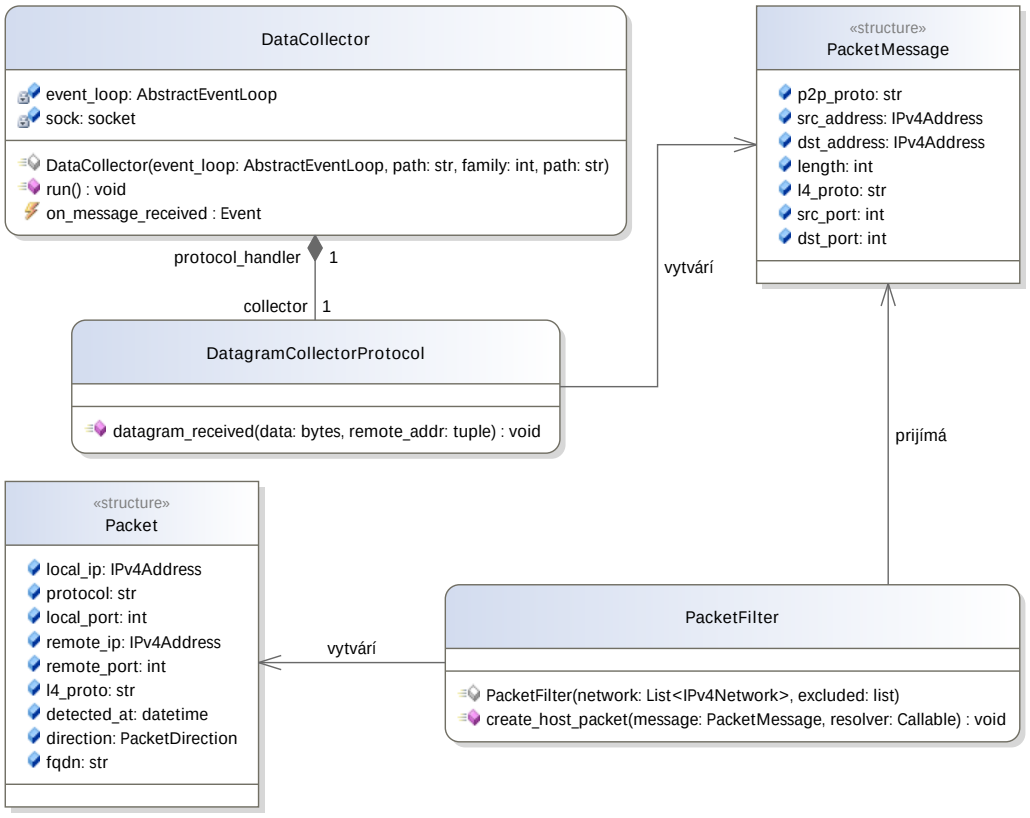
Příjem záznamů zajišťuje třída `DataCollector`. Ta vytváří UNIX socket (lze nakonfiurovat i UDP socket) na zadané cestě. Zpracování zpráv pak zajišťuje třída `DatagramCollectorProtocol`: pokud na socket přijde zpráva, třída ji zpracuje pomocí regulárního výrazu. Pokud výrazu odpovídá, vyextrahuje z ní příslušné pole, sestaví objekt typu `PacketMessage` a vyvolá událost `on_message_received`.
Na událost reaguje hlavní třída aplikace (`P2PProbeApp`) a zprávu předá třídě `PacketFilter`. Ta zkontroluje, že alespoň jedna z komunikujících stran je umístěna v nakonfigurovaných sítích. Pokud ano, sestaví objekt `Packet` tak, aby v poli `local_ip` byla vždy adresa uzlu ve vnitřní síti. Tato zpráva je předána hlavní aplikaci, která objekt `Packet` vloží do seznamu koncových stanic. Třídy jsou znázorněny na obrázku 11.5.

Vložení nového paketu do kolekce spustí vyhodnocení pravidel, která zkontrolují, zda komunikace koncové stanice nepřesáhla definované limity. To zajišťuje třída `StateRuleEvaluator`. Kontroluje se počet unikátních IP adres, se kterými koncová stanice komunikovala, počet spojení a celkový počet detekovaných paketů. Tyto limity jsou definovány pro každý protokol zvlášť.

Pokud jsou tyto limity překročeny, označí komunikaci v objektu `Host-ProtocolInfo` stavem `ProtocolState.over_limit`. Ve webovém rozhraní je tento stav zřetelně označen. Při dosažení tohoto stavu je také založen incident v systému Knet.

11.2.3 Webové rozhraní

Webové rozhraní je koncipováno jako SPA (Single Page Application). Uživateli se načte jediná stránka, obsahující JavaScript a styly uživatelského rozhraní. Komunikace se serverem pak probíhá pomocí protokolu WebSockets.



Obrázek 11.5: UML diagram tříd pro zpracování příchozích zpráv z ipp2p.

Zaslání příslušné stránky a komunikaci pomocí WebSockets zajišťuje třída `WebUI`. Ta zajišťuje zpracování požadavků na následující adresy:

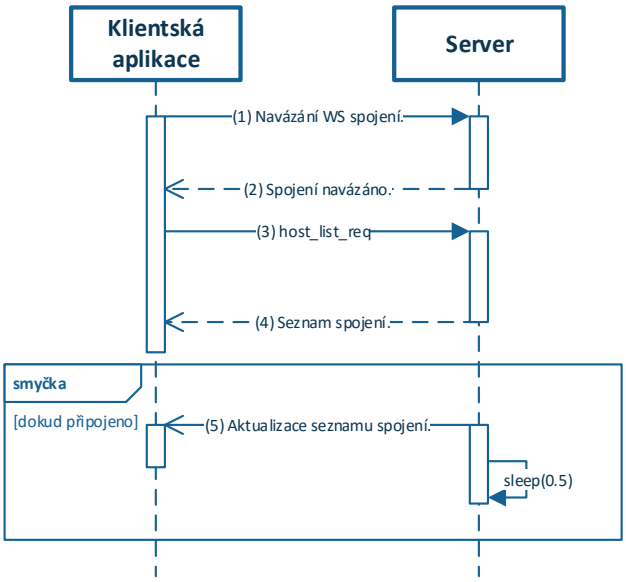
GET /: hlavní a jediná HTML stránka aplikace.
GET /ws: WebSocket spojení.
POST /webhook: příjem aktualizací od systému Knet.

Komunikace pomocí WebSockets využívá zprávy ve formátu JSON. V poli `type` je uveden typ zprávy. Použité zprávy jsou stručně popsány v tabulce 11.1. Schéma komunikace je znázorněno na obrázku 11.6.

Na adrese `/webhook` jsou přijímány zprávy od aplikace Knet. Aplikace provádí kontrolu jejich platnosti na základě nakonfigurovaného sdíleného klíče. Formát zpráv je popsán v sekci 9.9.2 Synchronizace údajů zpět ze systému Knet. Přijaté zprávy jsou předány třídě `KnetConnector`, která bude popsána v další sekci.

Zpráva	Zasílá	Popis
`host_list_req`	Klient	Požadavek na seznam aktuálních spojení.
`host_list`	Server	Seznam aktuálních spojení. Server zároveň klienta zařadí do seznamu pro příjem aktualizací. V poli „list“ je uvedena serializovaná verze aktuálního seznamu spojení.
`host_list_update`	Server	Aktualizace seznamu spojení. V poli „updates“ jsou uvedeny aktualizované záznamy, v poli „deletions“ seznam IP adres, které byly ze seznamu vyřazeny.

Tabulka 11.1: Typy zpráv při komunikaci webového rozhraní P2P sondy se serverem pomocí WebSockets.



Obrázek 11.6: Komunikace klientské aplikace P2P rozhraní se serverem pomocí WebSockets.

11.3 Integrace se systémem Knet

Pro integraci se systémem Knet je použita klientská knihovna vygenerovaná z OpenAPI definice rozhraní. Vygenerovaná knihovna obsahuje objektový model volání API, nepodporuje však získávání OAuth2 Access Token a všechna její volání jsou synchronní, takže je nelze použít v kombinaci s modulem *asyncio*.

Proto byla vytvořena dvě rozšíření knihovny:

`knet_api_oauth`
Přidává podporu pro získávání a ukládání OAuth2 tokenu v paměti. Token je získán před prvním voláním, poté je obnovován v případě, že se blíží konec času jeho platnosti.

`knet_api_aiohttp`
Rozšíření využívá knihovnu aiohttp pro integraci s modulem asyncio. Volání klientské knihovny jsou realizována pomocí korutin a je tedy možné knihovnu integrovat do rozhraní P2P sondy.

Obě rozšíření obsahují třídu `ApiClient`, kterou je možné použít k vykonání volání. Původní vygenerovaná knihovna obsahuje třídy `DormitoryApi`, `HostApi`, `IncidentApi` a `NoteApi`. Každá tato třída reprezentuje jednu část API a jejich operace vykonávají jednotlivá volání. Příklad použití:

```
client = knet_api_aiohttp.ApiClient(
        event_loop, client_id, client_secret, host=endpoint_url,
        token_url=token_url, scopes=['read', 'write']
)
host_api = HostApi(api_client=self.client)
```

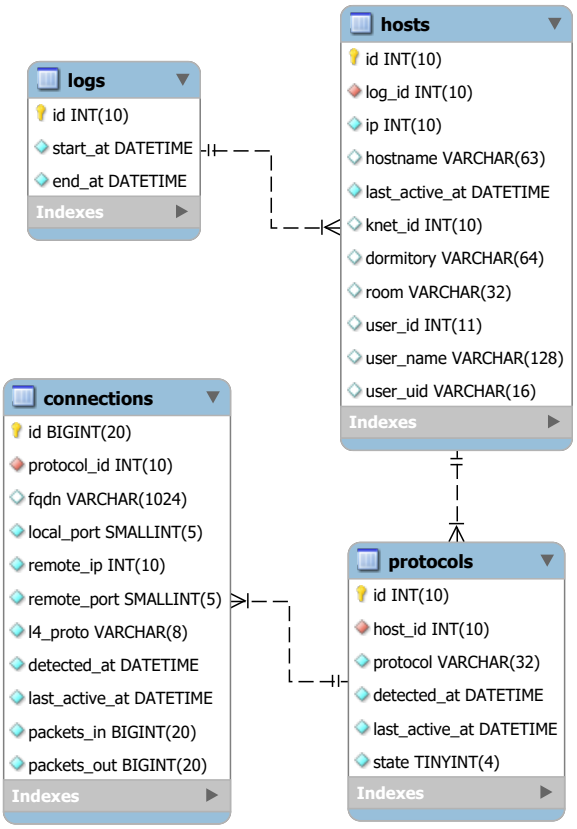
Integraci ve webovém rozhraní zajišťuje třída `KnetConnector`. Ta reaguje na události v modelu aplikace. Pokud dojde k detekci komunikace nové koncové stanice, vykoná korutina `query_host_info` této třídy asynchronní požadavek na informace o registraci stanice. Informace poté ukládá do modelu.

Pokud komunikace stanice některého protokolu přesáhla definované limity, založí tato třída incident v systému Knet. Založené incidenty průběžně aktualizuje v intervalu 60s. Aktualizaci zajišťuje korutina `sync_incidents`.

Třída také reaguje na webhook požadavky, které přijdou prostřednictvím webového rozhraní.

11.3.1 Ukládání záznamů

Záznamy v rozhraní jsou pravidelně ukládány do MySQL databáze. Tyto záznamy pak lze v aplikaci procházet a případně vyhledávat dle uživatelského jména či hostname koncové stanice. Struktura databáze je znázorněna na obrázku 11.7.



Obrázek 11.7: Databáze pro archivaci dat o P2P spojeních.

11.4 Klientská část webového rozhraní

Aplikace na straně klienta je implementována s využitím knihovny React[6]. Jedná se o JavaScriptovou knihovnu pro tvorbu uživatelských rozhraní. Rozhraní se skládá ze stromu komponent, které mají svůj vlastní stav a vlastní parametry. Při změně stavů nebo parametrů komponenty dojde k jejímu překreslení: volání metody `render()`, která vrací novou podobu komponenty.

Knihovna využívá virtuální DOM: překreslení neprovádí přímo v dokumentu, ale nejprve sestaví novou reprezentaci elementů v paměti a poté

[6]https://facebook.github.io/react/

provede minimální množství operací, které upraví stávající DOM webové stránky. Modifikace DOM jsou náročné, proto tento přístup přináší výkonnostní benefity.

Hlavním přínosem pro vývojáře je fakt, že nemusí v kódu provádět modifikace DOM, pouze sestaví novou podobu komponenty na základě jejího aktuálního stavu. Metodu `render()` tak lze chápat jako šablonu komponenty.

Knihovna React je používána v kombinaci s jazykem JSX[7]. Jedná se o rozšíření jazyka JavaScript, které umožňuje v kódu zapisovat XML elementy. V metodě `render()` tak lze zapsat přímo výsledný HTML kód. Komponenta pak může mít například následující podobu:

```
var HostInfo = React.createClass({
    render: function() {
        return <div>Hostname: {this.props.hostname}</div>;
    }
});

ReactDOM.render(
    <HostInfo hostname="knet.zero.zcu.cz" />,
    document.getElementById('app')
);
```

Tento jazyk je nutné před odesláním do prohlížeče přeložit do jazyka JavaScript. K tomu je použit nástroj webpack[8], který umožňuje slučování a aplikaci filtrů na JavaScriptové zdroje. Jedním z dostupných filtrů je i překladač jazyka JSX. Spolu s ním je použit i překladač Babel[9], který překládá nejnovější specifikaci jazyka ECMAScript 6 tak, aby bylo možné skripty zpracovat i ve starších prohlížečích. Tato verze jazyka také přidává podporu pro třídy, které jsou v aplikaci využity.

Všechny skripty webového rozhraní jsou uloženy ve složce `public/js`. V kořenovém adresáři jsou uloženy komponenty, které reprezentují hlavní obrazovky aplikace. Struktura podadresářů je následující:

`backend`: obsahuje třídy WebSocketBackend, která zajišťuje navázání spojení a komunikaci se serverem, a Notifications, která uživateli zasílá upozornění v případě, že dojde k překročení definovaných limitů.

[7] https://facebook.github.io/jsx/, https://facebook.github.io/react/docs/jsx-in-depth.html

[8] https://webpack.github.io/

[9] https://babeljs.io/

`components`: Komponenty uživatelského rozhraní: seznam detekované komunikace, komponenta pro vykreslení chybových zpráv a komponenta pro stránkování.

`model`: datový model klientské části aplikace. Strukturou odpovídá datovému modelu serveru.

12 Zhodnocení výsledků

Všechny části diplomové práce byly otestovány. V této kapitole budou shrnuty poznatky z testování a stručné srovnání nového systému s původním řešením. Pro snadné nasazení všech komponent byly vytvořeny balíčky pro operační systém Debian Stretch.

12.1 Webová aplikace

V rámci testování byla webová aplikace portálu nasazena na virtuální server knet-new.zero.zcu.cz[1] a nakonfigurována, tak aby získávala informace ze stávající síťové infrastruktury.

Byla prověřena správnost získávání informací ze systému LDAP i z kolejních přepínačů. Testy byly provedeny na blocích kolejí Máchova a Lochotín. Na kolejích Máchova pak byl umožněn přístup do systému stávajícím lokálním správcům a registrátorům. Ve spolupráci s nimi byly provedeny testy funkčnosti registrace, zejména pak rychlé registrace, která získává umístění uživatele.

Lokální správci také v průběhu testování odhalili několik chyb, většina z nich se týkala neočekávaných vstupů a chybějících závislostí na virtuálním serveru, např. rozšíření do PHP a doplňující funkce do PostgreSQL.

REST API systému bylo otestováno pomocí automatizovaných testů. Hlášení z průběhu testů ve formátu HTML je umístěno na přiloženém CD. Možnosti integrace navíc byly ověřeny vytvořením webového rozhraní pro P2P sondu, které automaticky zakládá a aktualizuje bezpečnostní incidenty.

Oproti předchozímu registračnímu systému se podařilo zjednodušit registrační proces pro uživatele. Systém navíc poskytuje instrukce o tom, co má uživatel provést. Pro správce a registrátory byl zabudován systém automatických kontrol registrací před schválením a vytvořena správa incidentů. Ta umožňuje naplánovat odblokování uživatele. Dále je možné v systému definovat registrační období, která umožní uživatelům hladký přechod do letního režimu, kdy se typicky stěhují mezi kolejemi. Ze systému byly odstraněny některé již nepotřebné funkce, například přiřazování zásuvek k portům na přepínačích.

[1] Server je dostupný pouze z kolejní sítě.

12.2 Konfigurační služby

Funkčnost konfiguračních služeb byla ověřena v izolovaném virtuálním serveru, neboť nebylo žádoucí, aby byla upravována konfigurace některého z existujících serverů. V něm proběhly i testy DHCP serveru Kea, které jsou popsané v sekci 7.5.2.

Všechny konfigurační služby správně prováděly očekávané konfigurační zásahy. Pouze služba pro konfiguraci DHCP serveru se po čase odpojila od jeho databáze, neboť vypršel časový limit neaktivity v MySQL spojení. Z této chyby se však služba správně zotavila a spojení navázala znovu.

12.3 Webové rozhraní P2P sondy

Webové rozhraní P2P sondy bylo opět otestováno v izolovaném prostředí virtuálního serveru. V něm byl otestován sběr dat (záznamy z ipp2p byly generovány pomocí automatického skriptu) a integrace s hlavním webovým portálem. Webové rozhraní správně automaticky načítalo informace o registracích, zakládalo incidenty a aktualizovalo existující informace.

Při změnách v incidentech ze strany systému Knet byla změna zaslána do webového rozhraní sondy a při obnovení informací (probíhá každou polovinu sekundy) zobrazena uživateli.

Rozhraní dále správně průběžně archivovalo své záznamy do databáze a umožnilo jejich procházení i vyhledávání.

Rozhraní bylo poté nasazeno na server, který v současnosti zajišťuje detekci P2P spojení. Zde bylo otestováno, že je aplikace schopná zpracovávat záznamy z ipp2p na reálném provozu.

Oproti původnímu řešení je nové rozhraní přehlednější a zobrazuje potřebné informace z registračního systému. Aktualizace probíhá v reálném čase a uživateli zobrazuje systémové notifikace v případě, že některý z uživatelů překročí definované limity připojení. Navíc obsahuje funkci archivace s vyhledáváním, což umožní zpětné dohledávání informací.

13 Možnosti rozšíření

Registrační systém i webové rozhraní P2P sondy byly navrženy s ohledem na možnost jejich budoucího rozšíření. V této kapitole bude uvedeno několik návrhů na novou funkcionalitu, která by mohla být doplněna.

13.1 Registrační portál

Webovou aplikaci registračního portálu by bylo možné rozšířit o další kontroly, které se provádějí při schvalování registrace. Lokálním správcům by pomohly následující kontroly:

- **Kontrola identifikátoru organizace, která přidělila MAC adresu (OUI).** Pomocí této kontroly by bylo možné odhalit zařízení, která byla registrována na falešnou MAC adresu. Někteří výrobci také dodávají převážně síťové prvky, například bezdrátové směrovače, bylo by tak možné odhalovat zařízení, která nejsou koncovou stanicí. Seznam přidělených identifikátorů poskytuje společnost IEEE v textovém souboru[1], bylo by možné automaticky stahovat a aktualizovat seznam přidělených identifikátorů.
- **Kontrola, zda na stejném či sousedním pokoji nebydlí uživatel, který byl zablokován.** V některých případech se stává, že zablokovaný uživatel požádá některého ze sousedů o vytvoření nové registrace jeho počítače. Tato kontrola by byla podnětem pro správce, aby zkusil zjistit více informací o daném incidentu a prověřil, že se jedná o legitimní registraci.

Dále by bylo možné implementovat další automatické kontroly mimo registrační proces, například kontrolu, zda jsou počítače připojené na stejných zásuvkách, kde jsou registrovány. Tato kontrola by nejen pomohla odhalit registrace, které byly provedeny za účelem obcházení pravidel připojení, ale i stěhování uživatelů a byla by podnětem pro aktualizaci záznamů v registračním systému.

Systém incidentů by bylo možné rozšířit o plnohodnotný šablonovací systém pro sestavování e-mailů a dalších zpráv pro uživatele. Správci by tak mohli plně ovlivnit podobu těchto zpráv a uživatelům tak poskytovat přesnější informace o jejich incidentech.

[1] http://standards-oui.ieee.org/oui.txt

Nabízí se také vytvoření mobilní verze systému, případně podpůrné mobilní aplikace, která by sloužila správcům. Pomocí této aplikace by správce mohl spravovat registrace i incidenty i v případě, že zrovna není na koleji, nebo není u svého počítače (například při řešení síťového problému u některého z ubytovaných).

Konfigurační služby pro webový portál by bylo možné rozšířit o podporu konfigurace specializovaného firewallu, například firewallu FortiGate od společnosti Fortinet, Inc. Dále by bylo možné integrovat jednotné přihlášení pomocí eduID.cz[2], což by umožnilo přihlašování studentům ostatních univerzit.

13.2 Webové rozhraní P2P sondy

Webové rozhraní sondy pro detekci P2P by bylo možné rozšířit o detekci aktualizačních služeb. Zejména různé on-line hry, které se aktualizují prostřednictvím P2P sítí, komunikují na určité předem známé adresy, případně používají specifické porty. Tyto znaky komunikace by bylo možné detekovat a zobrazit správcům, kteří na základě těchto informací mohou přistoupit k individuálnímu řešení incidentu.

Do webového rozhraní by dále bylo možné integrovat samotnou detekci. Pravidla, která kontroluje software IPP2P, by bylo možné integrovat přímo do aplikace v jazyce Python, nebo vyvinout nativní doplňující modul. Tím by se odstranila závislost na dalším software a kontrola by mohla probíhat v uživatelském režimu.

[2]https://www.eduid.cz/

14 Závěr

V rámci diplomové práce byl realizován nový registrační systém pro kolejní sítě Západočeské univerzity v Plzni. Před realizací byla provedena důkladná analýza stávajícího řešení, existujících řešení pro řízení přístupu v počítačových sítích a možností, které nabízí stávající infrastruktura kolejní sítě. V rámci práce také proběhlo seznámení s procesy řešení incidentů v univerzitní síti WEBnet a byla analyzována možnost integrace systému do těchto procesů.

Při návrhu bylo hledáno takové řešení, které nebude vyžadovat investice do nové infrastruktury. Na základě získaných poznatků a vlastních zkušeností z pozice lokálního správce kolejní sítě byl navržen nový registrační systém.

Tento systém pro uživatele přináší zejména jednodušší registraci, kterou je možné v běžných případech provést pouze stisknutím jediného tlačítka. Systém navíc v průběhu registrace uživateli poskytuje kontextové informace o tom, jaké kroky jsou od něj očekávány. Pro správce pak systém přináší lepší správu incidentů a automatizaci celé řady úkonů.

Z technického hlediska byla urychlena konfigurace síťových služeb. Tato změna obnáší i výběr jiného softwarového vybavení, které je možné konfigurovat prostřednictvím databáze. Změny v konfiguraci probíhají pomocí fronty zpráv, kterou zpracovávají konfigurační služby. Fronta zpráv je implementována v databázi PostgreSQL, garantuje doručení zpráv a používá asynchronní notifikace pro okamžitou notifikaci konzumentů fronty o dostupnosti nových zpráv.

Pro zadávání zpráv konfiguračním službám jsou využity prostředky aktivních databází. Změny je tedy možné provádět nejen ve webové aplikaci, ale i přímo v databázi pomocí software pro její správu. Složitější úkony tedy může správce systému provést přímo z příkazové řádky serveru.

V systému bylo připraveno REST API, které umožní integraci tohoto systému do procesů řešení incidentů v síti WEBnet. Možnost integrace byla prověřena na úrovni lokálních správců vytvořením moderního webového rozhraní pro sondu, která detekuje P2P spojení v kolejní síti. Pro toto API dále byla vytvořena sada automatických testů. Rozhraní bylo popsáno pomocí jazyka OpenAPI. Tento popis slouží jako jeho dokumentace a lze z něj vygenerovat klientské knihovny pro různé programovací jazyky.

Toto rozhraní zobrazuje informace o probíhající komunikaci v reálném čase. Data čerpá pomocí datagramového socketu z modulu ipp2p pro modul netfilter v OS Linux. Rozhraní také automaticky zakládá incidenty v registračním

portále a čerpá data o koncových stanicích. Rozhraní dále získává a zobrazuje informace o průběhu řešení jednotlivých incidentů, opět v reálném čase. Data o spojeních jsou periodicky ukládána do databáze a je možné v nich vyhledávat, nebo je procházet po časových intervalech.

Funkčnost řešení byla ověřena nasazením na připravený virtuální server s připravovaným OS Debian Stretch. Pro nasazení byly vytvořeny instalační balíčky, které automatizují proces instalace všech komponent.

Všechny body zadání byly splněny. Výsledný registrační systém by bylo možné rozšířit o další automatické kontroly registrací, případně vytvořit alternativní mobilní rozhraní. Rozhraní P2P sondy by bylo možné rozšířit o detekci aktualizací softwaru na koncových stanicích pomocí P2P sítí, případně do něj začlenit samotnou detekci P2P spojení.

Seznam použitých zkratek

AAA	Authentication, Authorization and Accounting
AES	Advanced Encryption Standard
AJAX	Asynchronous JavaScript and XML
AMQP	Advanced Message Queue Protocol
API	Application Programming Interface
AR	Access Requestor
ARP	Address Resolution Protocol
BOOTP	Bootstrap Protocol
BPDU	Bridge Protocol Data Unit
BPMN	Business Process Model and Notation
CDP	Cisco Discovery Protocol
CIDR	Classless Inter-Domain Routing
CSS	Cascading Style Sheets
DAO	Data Access Object
DDNS	Dynamic DNS
DDoS	Distributed DoS
DES	Hash-based Message Authentication Code
DHCP	Dynamic Host Configuration Protocol
DN	Distinguished Name
DNS	Domain Name System
DOM	Document Object Model
DoS	Denial of Service
DQL	Doctrine Query Language
EAP	Extensible Authentication Protocol
EAPOL	EAP over LAN
FTP	File Transfer Protocol
HMAC	Hash-based Message Authentication Code
IANA	Internet Assigned Numbers Authority
IPv4	Internet Protocol verze 4
JSON	JavaScript Object Notation
JSON RPC	JSON Remote Procedure Call
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LDIF	LDAP Data Interchange Format
MAC	Media Access Control

MAP	Metadata Access Point
MAPC	MAP Client
MD5	Hash-based Message Authentication Code
MIB	Management Information Base
MVC	Model-View-Controller
NAC	Network Access Control
OID	Object Identifier
P2P	Peer-to-Peer
PDP	Policy Decision Point
PEP	Policy Enforcement Point
RADIUS	Remote Authentication Dial In User Service
RAML	RESTful API Modeling Language
RDN	Relative Distinguished Name
REST	Representational State Transfer
SCSS	Sassy CSS
SGMP	Simple Gateway Monitoring Protocol
SHA	Secure Hash Algorithm
SMI	Structure of Management Information
SMTP	Simple Mail Transport Protocol
SNMP	Simple Network Management Protocol
SoH	Statement of Health
SPA	Single Page Application
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TNC	Trusted Networking Communications
UDP	User Datagram Protocol
VPN	Virtual Private Network
WADL	Web Application Description Language
WIRT	WEBnet Incident Response Team
XML	eXtensible Markup Language
YAML	YAML Ain't Markup Language

Literatura

[1] Aboba, B. et al. Extensible Authentication Protocol (EAP). RFC 3748, RFC Editor, Červen 2004. Dostupné z: http://www.rfc-editor.org/rfc/rfc3748.txt.

[2] *Administration Guide for PacketFence version 5.7.0* [online]. Inverse Inc., 2016. Dostupné z: https://packetfence.org/downloads/PacketFence/doc/PacketFence_Administration_Guide-5.7.0.pdf.

[3] Alexander, S. – Droms, R. DHCP Options and BOOTP Vendor Extensions. RFC 2132, RFC Editor, Březen 1997. Dostupné z: http://www.rfc-editor.org/rfc/rfc2132.txt.

[4] *AMQP Products and Success Stories* [online]. OASIS. [cit. 18. 4. 2016]. Dostupné z: https://www.amqp.org/about/examples.

[5] Bodó, R. – Kostěnec, M. Zkvalitnění procesu řešení bezpečnostních incidentů v síti WEBnet. Závěrečná zpráva projektu Fondu rozvoje CESNET 369/2010, Západočeská Univerzita v Plzni, 2011.

[6] Brown, E. L. *802.1X Port-Based Authentication*. Velká Británie: Auerbach Publications, 2006. ISBN 978-1420044652.

[7] Carter, G. *LDAP System Administration*, 1st edition. Beijing: O'Reilly Media, 2003. ISBN 978-0596551919. Kapitola *2 LDAPv3 Overview*.

[8] Case, J. et al. Message Processing and Dispatching for the Simple Network Management Protocol (SNMP). STD 62, RFC Editor, Prosinec 2002. Dostupné z: http://www.rfc-editor.org/rfc/rfc3412.txt.

[9] *Cisco NAC Appliance (Clean Access) End-of-Life and End-of-Sale Notices* [online]. Cisco Systems Inc., 2015. [cit. 15. 4. 2016]. Dostupné z: http://www.cisco.com/c/en/us/products/security/nac-appliance-clean-access/eos-eol-notice-listing.html.

[10] *Cisco Network Admission Control (NAC) Solution Data Sheet* [online]. Cisco Systems Inc., 2014. [cit. 15. 4. 2016]. Dostupné z: http://www.cisco.com/c/en/us/products/collateral/security/nac-appliance-clean-access/product_data_sheet0900aecd802da1b5.html.

[11] DeKok, A. Use of Status-Server Packets in the Remote Authentication Dial In User Service (RADIUS) Protocol. RFC 5997, RFC Editor, Srpen 2010. Dostupné z: http://www.rfc-editor.org/rfc/rfc5997.txt.

[12] Dostálek, L. – Kabelová, A. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5., aktualizované, vyd. Praha: Computer Press, 2008. ISBN 978-8025122365. Kapitola *11 DNS*.

[13] Dostálek, L. et al. *Velký průvodce protokoly TCP/IP*. 2., aktualizované, vyd. Praha: Computer Press, 2003. ISBN 978-8072268498. Kapitola *4.7 Protokoly RADIUS a TACACS+*.

[14] Droms, R. Dynamic Host Configuration Protocol. RFC 2131, RFC Editor, Březen 1997. Dostupné z: `http://www.rfc-editor.org/rfc/rfc2131.txt`.

[15] Freed, N. Behavior of and Requirements for Internet Firewalls. RFC 2979, RFC Editor, Říjen 2000. Dostupné z: `http://www.rfc-editor.org/rfc/rfc2979.txt`.

[16] Geer, D. Whatever Happened to Network-Access-Control Technology? *Computer*. září 2010, 43, 9, s. 13–16. ISSN 0018-9162. doi: 10.1109/MC.2010.269.

[17] Harrington, D. – Presuhn, R. – Wijnen, B. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. STD 62, RFC Editor, Prosinec 2002. Dostupné z: `http://www.rfc-editor.org/rfc/rfc3411.txt`.

[18] Howes, T. A String Representation of LDAP Search Filters. RFC 1960, RFC Editor, Červen 1996. Dostupné z: `http://www.rfc-editor.org/rfc/rfc1960.txt`.

[19] *Kea Developer's Guide* [online]. Internet Systems Consortium. [cit. 26. 4. 2016]. Dostupné z: `http://git.kea.isc.org/~tester/kea/doxygen/`.

[20] Mauro, D. – Schmidt, K. *Essential SNMP*, 2nd edition. Sebastopol: O'Reilly Media, 2005. ISBN 978-0596008406. Kapitoly *2 SNPMv1 and SNMPv2* a *3 SNMPv3*, s. 19-84.

[21] McCloghrie, K. – Perkins, D. – Schoenwaelder, J. Structure of Management Information Version 2 (SMIv2). STD 58, RFC Editor, Duben 1999. Dostupné z: `http://www.rfc-editor.org/rfc/rfc2578.txt`.

[22] Mockapetris, P. Domain names - implementation and specification. STD 13, RFC Editor, Listopad 1987. Dostupné z: `http://www.rfc-editor.org/rfc/rfc1035.txt`.

[23] *Network Access Protection Deployment Guide* [online]. Microsoft Corporation, 2012. [cit. 15. 4. 2016]. Dostupné z: `https://technet.microsoft.com/cs-cz/library/dd314175(v=ws.10).aspx`.

[24] *Object Identifier Types* [online]. PostgreSQL Global Development Group. [cit. 20. 4. 2016]. PostgreSQL 9.5 Documentation. Dostupné z: `http://www.postgresql.org/docs/9.5/static/datatype-oid.html`.

[25] *openNAC Wiki* [online]. OPENNAC TECH, 2016. [cit. 16. 4. 2016]. Dostupné z: `https://redmine.opennac.org/projects/opennac/wiki`.

[26] *pfSense Documentation site* [online]. Electric Sheep Fencing LLC, 2016. [cit. 16. 4. 2016]. Dostupné z: `https://doc.pfsense.org/index.php/Main_Page`.

[27] Presuhn, R. Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). STD 62, RFC Editor, Prosinec 2002. Dostupné z: `http://www.rfc-editor.org/rfc/rfc3416.txt`.

[28] Rigney, C. RADIUS Accounting. RFC 2866, RFC Editor, Červen 2000. Dostupné z: `http://www.rfc-editor.org/rfc/rfc2866.txt`.

[29] Rigney, C. et al. Remote Authentication Dial In User Service (RADIUS). RFC 2865, RFC Editor, Červen 2000. Dostupné z: `http://www.rfc-editor.org/rfc/rfc2865.txt`.

[30] Sermersheim, J. Lightweight Directory Access Protocol (LDAP): The Protocol. RFC 4511, RFC Editor, Červen 2006. Dostupné z: `http://www.rfc-editor.org/rfc/rfc4511.txt`.

[31] Serrao, G. J. Network access control (NAC): An open source analysis of architectures and requirements. In *Security Technology (ICCST), 2010 IEEE International Carnahan Conference on*, s. 94–102, říjen 2010. doi: 10.1109/CCST.2010.5678694.

[32] *Software Configuration Guide, Cisco IOS Release 15.2(2)E (Catalyst 2960, 2960-S, 2960-SF and 2960-Plus Switches)* [online]. Cisco Systems Inc., 2016. [cit. 16. 4. 2016]. Kapitola Configuring IEEE 802.1x Port-Based Authentication. Dostupné z: `http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst2960/software/release/15-2_2_e/configuration/guide/b_1522e_2960_2960c_2960s_2960sf_2960p_cg/b_1522e_2960_2960c_2960s_2960sf_2960p_cg_chapter_0110000.html#ID398`.

[33] *The Smfony Book* [online]. SensioLabs. [cit. 22. 4. 2016]. Dostupné z: `http://symfony.com/doc/current/book/index.html`.

[34] Trusted Computing Group. *TNC Architecture for Interoperability 1.5, revision 4*, 7. května 2012.

[35] *Twig Documentation* [online]. SensioLabs. [cit. 22. 4. 2016]. Dostupné z: `http://twig.sensiolabs.org/documentation`.

[36] *Usage statistics and market share of PHP for websites* [online]. W3Techs. [cit. 21. 4. 2016]. Dostupné z: `https://w3techs.com/technologies/details/pl-php/all/all`.

[37] *What's New in DHCP* [online]. Microsoft Corporation, 2015. [cit. 15. 4. 2016]. Dostupné z: `https://technet.microsoft.com/en-us/library/dn765482.aspx`.

[38] Zbortek, M. Rozšíření registračního systému pro kolejní sítě. Bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2008. Vedoucí práce Michal Petrovič.

Přílohy

A Postup nasazení

V této kapitole bude popsán postup nasazení jednotlivých komponent. Pro všechny komponenty byly vytvořeny balíčky pro operační systém Debian. Tyto balíčky instalují nejen soubory potřebné pro běh jednotlivých komponent, ale i vzorové konfigurační soubory.

Popis bude předpokládat nasazení do produkčního prostředí. Při testování ve vývojovém prostředí mohou být některé části nastavení vynechány, nebo mohou být jiné parametry.

A.1 Registrační portál

Pro nasazení registračního portálu byl vytvořen balíček knet_1.0-1_all.deb. Balíček obsahuje závislosti na všem potřebném software, včetně lokální databáze: Apache 2 (včetně modulů *mod_webauth* a *mod_webauthldap*), PHP 7.0 či vyšší včetně všech potřebných rozšířeních (včetně *opcache*[1] a *apcu*[2]), PostgreSQL 9.5 či vyšší.

Balíček je možné nainstalovat včetně všech závislostí například příkazem gdebi[3]:

```
$ sudo gdebi knet_1.0-1_all.deb
```

Postinstalační skript zajistí aktivaci potřebných modulů Apache, vytvoření uživatele knet (s náhodným heslem) a databáze knet v PostgreSQL, vytvoření konfiguračního souboru a vytvoření struktury databáze. Balíček také přidává do `/etc/cron.d` soubor s konfigurací pro cron.

Soubory webového portálu se instalují do složky `/var/www/knet`. Konfigurační soubor s přístupovými údaji do databáze je umístěn v podsložce `app/config/parameters.yml`. Pokud tento soubor neexistuje, v rámci instalace se spustí průvodce, pomocí kterého je uživatel vyzván k doplnění chybějících parametrů. Parametry pro připojení do databáze jsou doplněny automaticky. Příklad zadávání parametrů:

```
> Incenteev\ParameterHandler\ScriptHandler::buildParameters
Creating the "app/config/parameters.yml" file
```

[1] http://php.net/manual/en/book.opcache.php
[2] http://php.net/manual/en/book.apcu.php
[3] https://packages.debian.org/stretch/gdebi

```
Some parameters are missing. Please provide them.
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):
ldap_uri ('ldaps://psyche.zcu.cz:636'):
ldap_base ('ou=Users,ou=rfc2307,o=zcu,c=cz'):
ldap_binddn (null):
ldap_password (null):
ldap_sasl_mech (null): GSSAPI
ldap_sasl_realm (null):
```

Do konfiguračního souboru je potřeba přidat přístupové údaje pro přepínače:

```
knet:
  credentials:
    knetsw:
      snmp:
        community: SomeKnetCommunity
```

Všechny konfigurační soubory se slučují, kompilují a ukládají do cache. Po jeho změně je potřeba cache smazat, a to buď přímo smazáním složky `/var/www/knet/var/cache/prod`, nebo příkazem

```
$ SYMFONY_ENV=prod sudo -E -u www-data php bin/console
  cache:clear
```

Po instalaci je potřeba upravit konfiguraci pro Apache v souboru `/etc/apache2/sites-available/knet.conf`. Soubor obsahuje konfigurace pro HTTP (port 80) i HTTPS (port 443). Je nastaveno přesměrování z HTTP na HTTPS.

V konfiguračním souboru je potřeba:

1. Nastavit jméno serveru (direktiva ServerName).
   Výchozí hodnota je knet.zero.zcu.cz.
2. Nastavit cestu k SSL certifikátu a privátnímu klíči – konfigurační direktivy `SSLCertificateFile` a `SSLCertificateKeyFile`.
3. Povolit či zakázat WebAuth. Pro účely testování může být vhodné webauth zakázat, v takovém případě je potřeba do konfigurace přidat nastavení proměnné `REMOTE_USER`, která uchovává jméno přihlášeného uživatele:
   `SetEnv REMOTE_USER "<uid>"`

Po úpravě konfigurace je nutné site povolit příkazem:

```
$ sudo a2ensite knet
```

Po nastavení potřebných parametrů a restartu Apache je již registrační portál dostupný na nastavené adrese a portu. Pro přidělení administračních oprávnění pro uživatele je možné použít následující příkazy:

```
$ cd /var/www/knet
$ SYMFONY_ENV=prod sudo -E -u www-data php bin/console
  user:create-superadmin <uid>
```

Příkazy aplikace je potřeba spouštět jako uživatel `www-data` a s proměnnou prostředí `SYMFONY_ENV`, nebo s parametrem `-env=prod`. Pokud by byl příkaz spuštěn jako jiný uživatel, nebude mít webový server oprávnění pro zápis vytvářených souborů či složek v cache.

A.1.1 Načtení dat

Po instalaci je potřeba naplnit systém daty. Informace o doménách (`domains`) skupinách kolejí (`dormitory_groups`), kolejích (`dormitories`) a texty pravidel (`rules_texts`) je nutné zadat ručně do příslušných tabulek. Alternativně je možné naimportovat do databáze soubor `sql/initial-data.sql`, který obsahuje definice převzaté z původního systému.

Po přihlášení do systému je nutné nadefinovat podsítě (sekce Sítě) a přepínače (sekce Sítě > Switche). Po jejich nadefinování je možné načíst místnosti a zásuvky z popisků portů jednotlivých přepínačů, a to příkazem:

```
$ SYMFONY_ENV=prod sudo -E -u www-data php bin/console
  switch:load-sockets <id>
```

`<id>` nahraďte za seznam ID všech přepínačů, ze kterých mají být načteny zásuvky. Je možné zadat seznam oddělený čárkami, nebo rozsah ve tvaru `<min> <max>`. Přepínače se zadaným ID nemusejí v systému existovat, je tedy možné zadat velký rozsah a načíst tak informace ze všech přepínačů.

Předpokládány jsou popisky ve formátu „<místnost> (<zásuvka>)“. Porty, jejichž popisek je prázdný nebo neodpovídá tomuto formátu, budou ignorovány. Formát popisku lze určit regulárním výrazem – do souboru `app/config/parameters.yml` je možné přidat dodatečný parametr `port_description_pattern`. Výchozí hodnotou je:

```
#^(?P<room>.+)\s+\((?P<socket>.+)\)#
```

A.1.2 Spuštění testů

Pro spuštění testů je nutné nainstalovat závislosti pro vývoj pomocí nástroje composer s příznakem `-dev`:

```
$ composer install --dev
```

Před spuštěním testů je nutné vytvořit v PostgreSQL databázi, která bude pro testy použita. Do databáze je potřeba přidat rozšíření `unaccent` příkazem:

```
CREATE EXTENSION unaccent;
```

V OS Debian je rozšíření součástí balíčku postgresql-contrib. Rozšíření může přidávat pouze uživatel s administrátorskými právy k databázovému serveru.

Po vytvoření databáze nakopírujte či přejmenujte soubor `app/config/parameters_test.yml.dist` na `parameters_test.yml`. V něm pak nastavte název databáze (výchozí je `knet_test`).

Strukturu databáze je poté nutné vytvořit příkazem:

```
$ SYMFONY_ENV=test php bin/console doctrine:migrations:migrate
```

Proměnnou prostředí lze nahradit parametrem `-env=test`. Testy lze z kořenového adresáře projektu spustit příkazem:

```
$ vendor/bin/phpunit -d memory_limit=256M
```

Testy vyžadují alespoň 256MB paměti, neboť v nich dochází k opakované inicializaci aplikace v rámci jednoho procesu.

A.2 Konfigurační služby

Konfigurační služby jsou v balíčku knet-conf-services_1.0-1_all.deb. Balíček instaluje všechny potřebné závislosti: Python 2.x (včetně všech použitých knihoven) a supervisord. Balíček také přidává systémového uživatele knet-svc, pod kterým konfigurační skripty běží.

Před spuštěním služeb je potřeba pro ně vytvořit fronty zpráv v databázi. To je možné příkazem `message-queue:create-consumer`:

```
$ SYMFONY_ENV=prod sudo -E -u www-data php bin/console
  message-queue:create-consumer dhcp dns firewall webhook
```

Supervisord slouží ke spouštění skriptů na pozadí. Běží jako služba systém a pokud skript neočekávaně skončí, spustí jej znovu.

Po instalaci je potřeba upravit konfigurační soubor služeb, který je umístěn na `/opt/knet-conf-services/config.cfg`. V sekci `[global]` je nutné nastavit přístupové údaje k databázi registračního portálu, v konfiguračních sekcích jednotlivých služeb přístupové údaje k databázím DHCP serveru, DNS serveru a SSH na stroji s firewallem.

Po úpravě konfiguračních souborů je nutné restartovat supervisord:

```
$ sudo /etc/init.d/supervisor start
```

Konfigurační skripty se spustí automaticky. V případě potřeby je možné skripty ovládat utilitou supervisorctl:

```
$ sudo supervisorctl
supervisor> restart knet:*
supervisor> stop knet:*
supervisor> start knet:*
```

A.3 DHCP server

Pro DHCP server Kea bylo vytvořeno rozšíření, které nastavuje kratší dobu zapůjčení pro dynamicky přidělované adresy. Rozšíření je k dispozici v balíčku knet-kea-ext_1.0-1_amd64.deb. Po jeho instalaci je potřeba do konfiguračního souboru DHCP serveru přidat jeho načtení:

```
{
        "Dhcp4":
        {
                # ...
                "hooks-libraries": [{
                        "library": "/usr/lib/libkea-knet.so"
                }],
                # ...
        }
}
```

Konfiguraci sítí je možné ze systému vygenerovat příkazem:

```
$ SYMFONY_ENV=prod sudo -E -u www-data php bin/console
  config:generate:dhcp > kea-subnets.json
```

Vygenerovaný seznam podsítí je možné začlenit do konfiguračního souboru například pomocí nástroje jq[4]:

```
$ jq -s ".[0] * .[1]" /etc/kea/kea-dhcp4.conf kea-subnets.json
```

A.4 Firewall

Na stroji s firewallem je nutné nainstalovat balíček knet-fw-helpers_1.0-1_all.deb, který obsahuje pomocné skripty pro konfiguraci iptables. Šablona pravidel se nastavuje v souboru `/opt/knet-fw-helpers/config.cfg`. Obsahuje 3 sekce:

[**rules_active**] : pravidla přidaná při povolení přístupu koncové stanice do vnější sítě.

[**rules_quarantine**] : pravidla přidávaná při zablokování koncové stanice.

[**rules_deleted**] : pravidla přidávaná při smazání koncové stanice. Sekce by měla být prázdná, tj. pro stanici nebudou přidána žádná pravidla.

Při změně pravidel se vždy odstraní všechna pravidla pro danou koncovou stanici a poté se přidají pravidla nová. Každá sekce může obsahovat několik pravidel:

```
[rules_active]
r1 = -I INPUT -s %(IP)s -j ACCEPT
r2 = -I FORWARD -s %(IP)s -j ACCEPT
```

Názvy klíčů je možné zvolit libovolně, skripty je ignorují. Hodnotou jsou parametry příkazu iptables, který se má spustit. V parametrech probíhá náhrada následujících řetězců:

`%(ID)s`: ID koncové stanice.

`%(IP)s`: celá IP adresa koncové stanice.

`%(IP_1)s`, `%(IP_2)s`, `%(IP_3)s`, `%(IP_4)s`: první až čtvrtý oktet IP adresy jako číslo v desítkové soustavě.

A.5 Webové rozhraní pro P2P sondu

Webové rozhraní pro P2P sondu je instalováno balíčkem knet-p2p-probe_1.0-1_all.deb. Soubory sondy se nainstalují do složky `/opt/knet-p2p-probe`.

[4] https://stedolan.github.io/jq/

Součástí je opět konfigurační soubor pro supervisor, jeho použití je popsáno v sekci Konfigurační služby.

Konfigurační soubor je umístěn v `/opt/knet-p2p-probe/config.yml`. V něm je potřeba nastavit následující sekce:

collector: cesta, na které má být vytvořen socket pro naslouchání na zprávy z ipp2p. `socket_family` nastavit na `AF_UNIX`, `socket_path` na cestu k socketu (možno ponechat výchozí).

packet_filter: Maska sítě, ve které jsou připojeni uživatelé. Pro koleje Máchova je maska 10.10.0.0/16, pro koleje Borská pole 10.20.0.0/16, pro koleje Bolevecká 10.30.0.0/16. Volitelně je možné vyloučit určité části sítě, např. na kolejích Máchova je vhodné vyloučit rozsah, ve kterém jsou registrovány studentské servery: 10.10.3.0/24.

knet_connector: Client ID a Client Secret vygenerované z registračního portálu (po přihlášení do rozhraní v sekci Přístup k API), dále pak URL adresu registračního portálu.

resolver: Adresy DNS serverů, které budou používány pro překlad adres. Pokud nebudou vyplněny, použitá knihovna bude překlad provádět sama.

web_ui: IP adresa a port, na kterém bude dostupné webové rozhraní, dále pak klíč použitý při podepisování websocket požadavků ze systému knet.

Po konfiguraci je možné rozhraní spustit restartováním supervisor, případně jen přenačtením jeho konfigurace příkazem reload a poté pomocí supervisorctl:

```
$ sudo /etc/init.d/supervisor reload
$ sudo supervisorctl restart knet_p2p_probe
```

Dále je potřeba nakonfigurovat pravidla firewallu pro ipp2p a syslog, aby zprávy zasílal na příslušní Unix socket. Příklad pravidel pro iptables:

```
/sbin/iptables -t mangle -A PREROUTING -m ipp2p --bit -i eth0
        -j LOG --log-prefix 'P2P:BitTorrent '
/sbin/iptables -t mangle -A PREROUTING -m ipp2p --dc -i eth0
        -j LOG --log-prefix 'P2P:DirectConnect '
/sbin/iptables -t mangle -A PREROUTING -m ipp2p --ares -i eth0
        -j LOG --log-prefix 'P2P:Ares '
/sbin/iptables -t mangle -A PREROUTING -m ipp2p --kazaa -i eth0
        -j LOG --log-prefix 'P2P:Kazaa '
/sbin/iptables -t mangle -A PREROUTING -m ipp2p --gnu -i eth0
```

```
        -j LOG --log-prefix 'P2P:Gnutella '
/sbin/iptables -t mangle -A PREROUTING -m ipp2p --edk -i eth0
        -j LOG --log-prefix 'P2P:eDonkey '
/sbin/iptables -t mangle -A PREROUTING -m ipp2p --soul -i eth0
        -j LOG --log-prefix 'P2P:SoulSeek '
```

Příklad konfigurace rsyslog pomocí modulu omuxsock[5]:

```
$ModLoad omuxsock
$OMUxSockSocket /var/run/p2p-probe/p2p-ui.sock

if $msg contains "] P2P:" then {
        :omuxsock:
        stop
}
```

Modul omuxsock umožňuje zasílat zprávy pouze do jednoho globálně nakonfigurovaného socketu. Pokud již v systému je nějaký socket nakonfigurovaný, je nutné použít zasílání pomocí UDP modulem omfwd[6] a nakonfigurovat rozhraní, aby místo na unixovém socketu (AF_UNIX) naslouchalo na IPv4 (AF_INET).

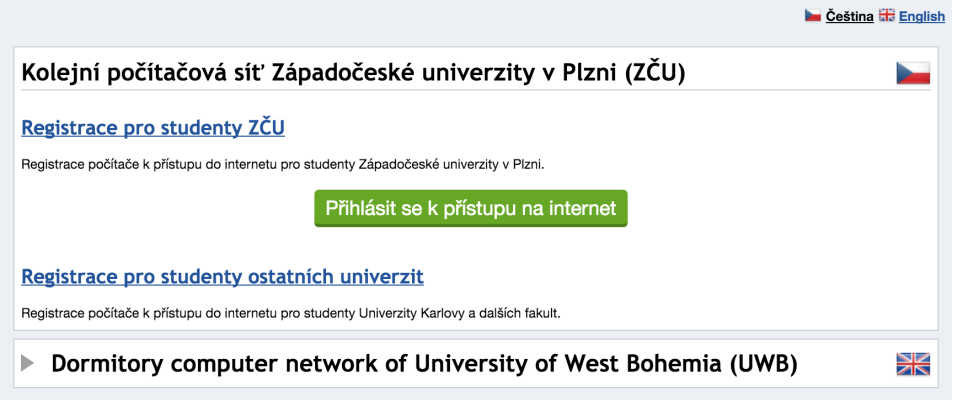
[5]http://www.rsyslog.com/doc/v8-stable/configuration/modules/omuxsock.html
[6]http://www.rsyslog.com/doc/v8-stable/configuration/modules/omfwd.html

B Postup registrace počítače

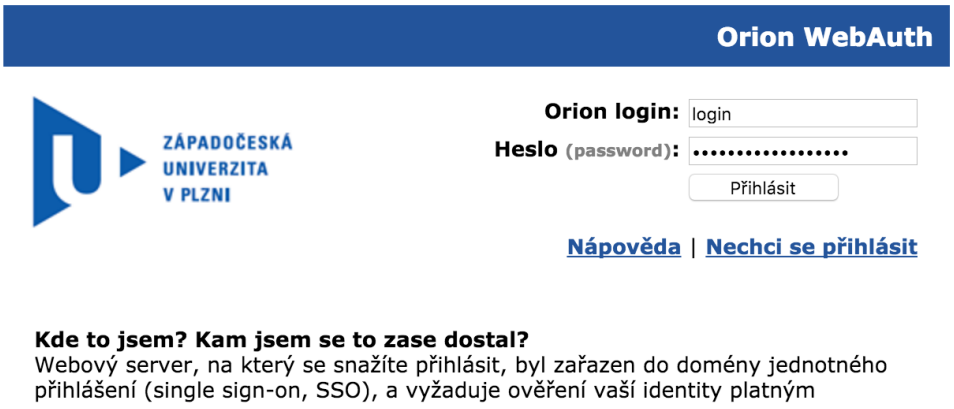
Pro připojení internetu na koleji je potřeba zaregistrovat svůj počítač.

1. Připojte počítač do síťové zásuvky pomocí síťového kabelu. Zásuvka je typicky umístěna v blízkosti některého rohu pokoje.

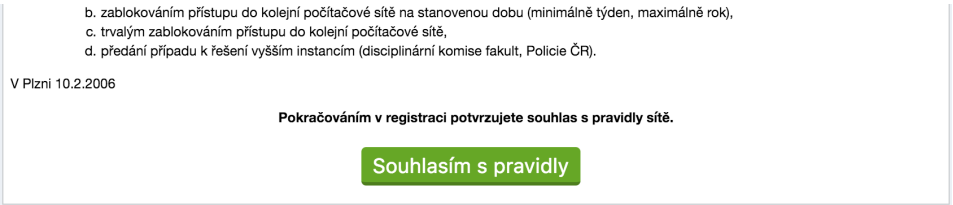
2. Otevřete internetový prohlížeč a zadejte adresu knet.zero.zcu.cz. Na stránce stiskněte tlačítko Přihlásit se k přístupu na internet.



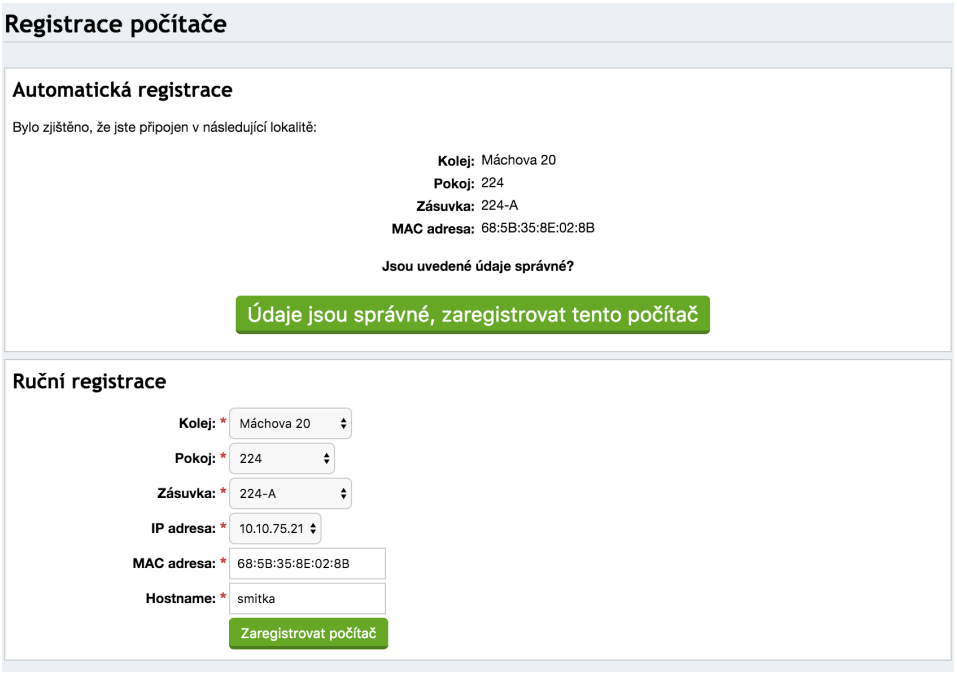
3. Přihlaste se pomocí svého účtu Orion. Pokud jej ještě nemáte, co nejdříve si jej zařiďte, bez něj se nelze na kolejích připojit k internetu!



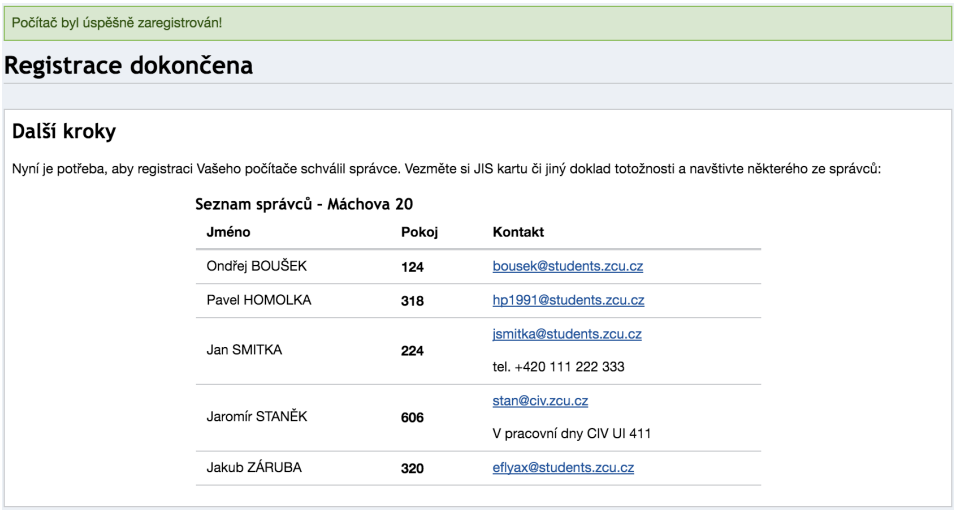
4. Po přihlášení si pozorně přečtěte pravidla sítě a potvrďte souhlas na konci stránky. Neznalost pravidel neomlouvá!



5. Systém zjistí, na kterém pokoji je Váš počítač připojen, a nabídne rychlou registraci. Zkontrolujte správnost údajů a registraci potvrďte.



6. Po uložení registrace navštivte některého ze správců sítě. S sebou budete potřebovat JIS kartu či jiný doklad totožnosti, na kterém je Vaše jméno a fotografie. Správce potvrdí Vaši registraci a případně poskytne další informace.



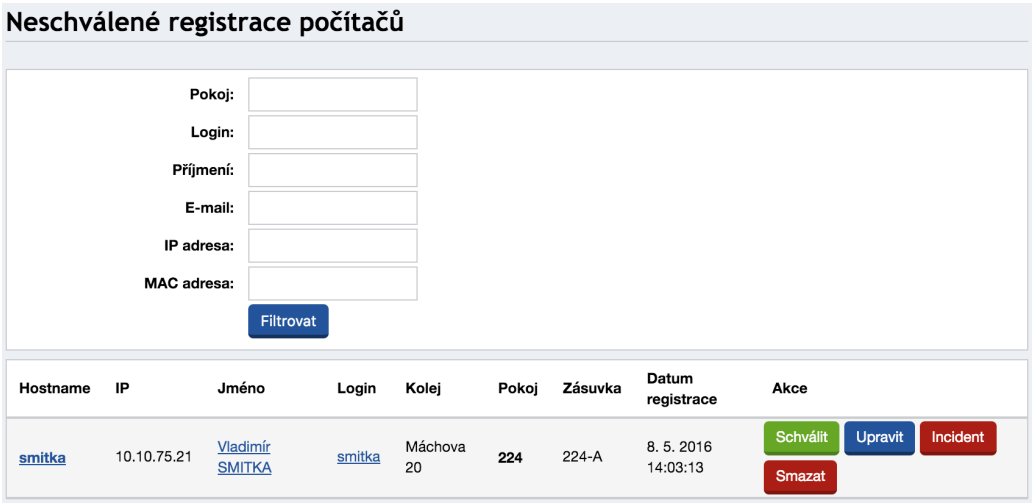
7. Po potvrzení registrace by mělo být připojení do několika minut aktivní. Pokud se tak nestane, zkuste restartovat počítač. Pokud ani po restartu počítače připojení nefunguje, navštivte znovu správce.

C Manuál správce

Nejdůležitějším úkonem, který správce provádí, je schvalování a úprava registrací, případně řešení incidentů. V tomto manuálu proto bude detailně popsán právě postup práce s registracemi a incidenty, ostatním funkcím portálu bude věnován stručný popis v závěru textu.

C.1 Schvalování registrací

V sekci Neschválené registrace je možné schvalovat registrace. V přehledu se zobrazují všechny neschválené registrace počítačů z vybrané koleje.

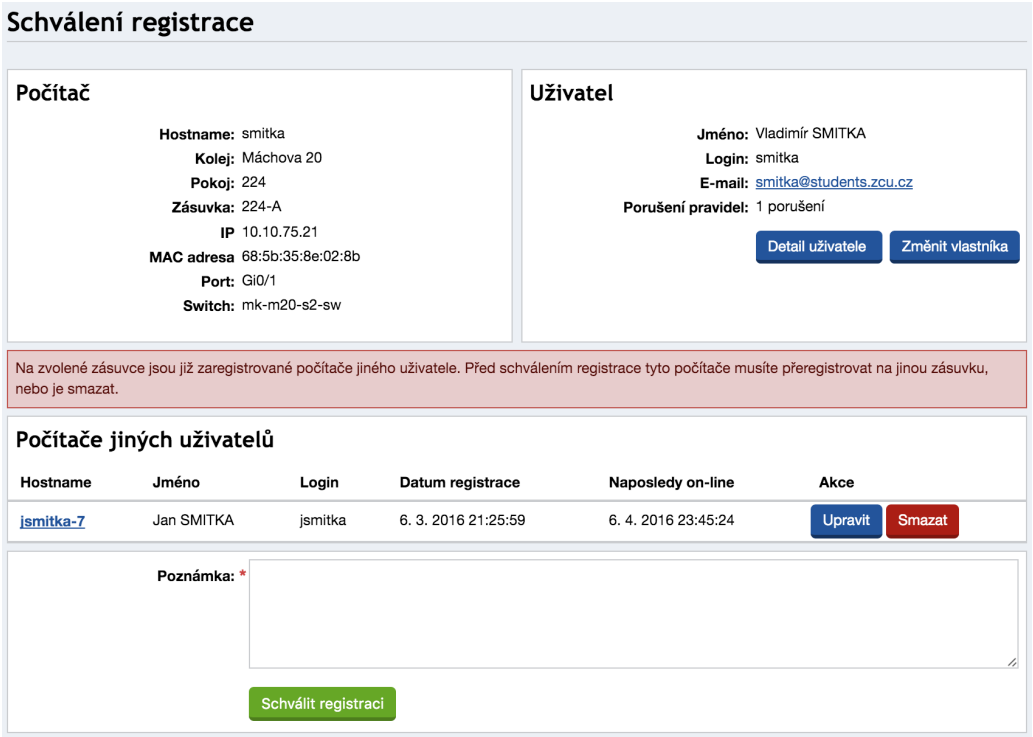


V pravém horním rohu je možné změnit kolej, která má být zobrazena. Toto nastavení ovlivňuje všechny přehledy v systému.

Při schvalování registrace systém provádí několik automatických kontrol. Jde zejména o:

- Kontrolu, zda je registrovaná MAC adresa platná – systém nedovoluje registrovat multicastové MAC adresy. V případě locally-administered adres je potřeba vyplnit komentář, proč byla daná registrace schválena.
- Kontrolu, zda na stejné zásuvce nejsou registrováni jiní uživatelé. Pokud ano, je nutné registrace jiných uživatelů smazat (pokud se již odstěhovali), nebo přesunout na jiný pokoj (pokud došlo ke stěhování). Při

úpravách registrací se snažte od člověka, jehož registraci schvalujete, zjistit co nejvíce informací. Může Vám pomoci také kontrola časů, kdy byl počítač naposledy spatřen on-line.



Problémy, které jsou zobrazeny žlutě, slouží pouze jako upozornění. Registraci je možné schválit, ale je nutné vyplnit komentář, proč registrace byla schválena. Problémy, které jsou zobrazeny červeně, je nutné nejprve odstranit.

C.2 Přehled počítačů

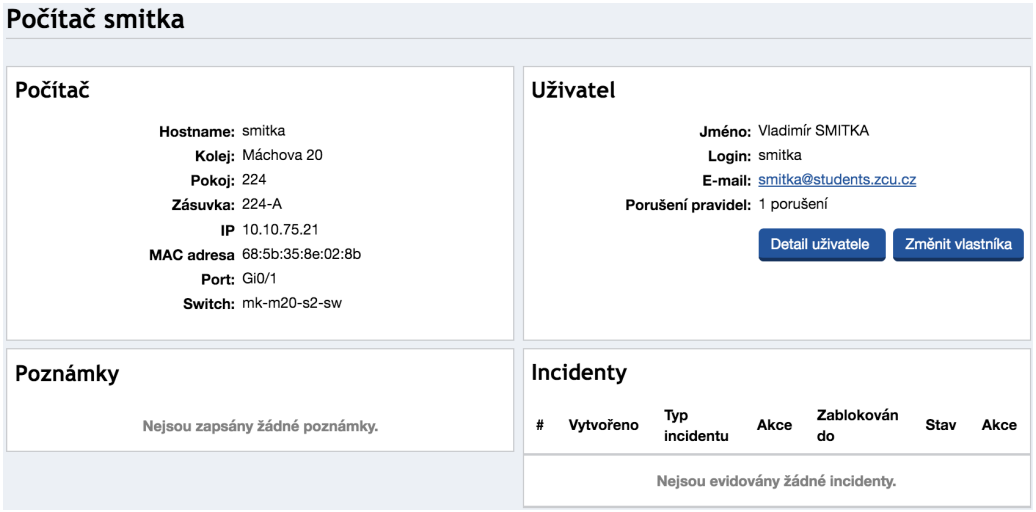
Registrace je možné upravovat z pohledu Seznam počítačů. Tento přehled je možné filtrovat podle pokoje, uživatelského jména, příjmení, e-mailu, IP adresy a MAC adresy. S výjimkou IP a MAC adresy provádí všechna pole textové vyhledávání. Je možné použít % jako zástupný znak: řetězec „sm%“ tedy bude hledat všechny hodnoty, které začínají na písmena „sm“. Filtrace ignoruje velikosti písmen a diakritické znaky.

V případě IP adresy je možné zadat buď konkrétní IP adresu, nebo IP adresu s maskou sítě: například 10.10.75.0/24. Zobrazeny budou všechny počítače, jejichž adresa odpovídá této masce. MAC adresu je nutné zadat přesně a není možné používat zástupné znaky.

Tlačítkem Upravit je možné upravit registraci. Správce může upravovat pouze registrace na kolejích, u kterých má oprávnění registrace schvalovat.

Může však registraci přesunout na některou z kolejí, které spravuje, nezávisle na tom, kde je registrace vytvořena.

Kliknutím na hostname počítače se zobrazí detaily uživatele. V seznamu jsou zobrazeny všechny poznámky zapsané k uživateli, seznam incidentů a záznamy v logu pro daného uživatele.



V pravé horní části je možné zobrazit detaily uživatele, případně změnit uživatele, který počítač vlastní. Změnu vlastníka mohou provádět jen správci se zvláštním oprávněním.

V detailu uživatele je navíc k dispozici přehled všech počítačů, které má daný uživatel registrované.

C.3 Incidenty

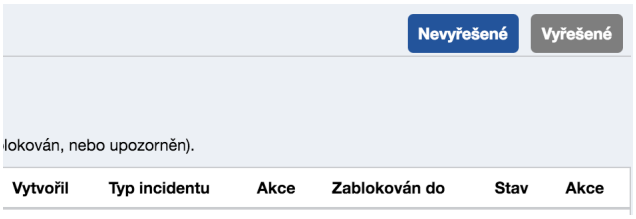
Přehled „Incidenty“ je rozdělen do dvou kategorií:

Nevyřešené incidenty: incidenty, které jsou buď označeny jako nevyřešené, nebo zablokovaly uživatele a tato blokace je stále aktivní. Tento přehled je rozdělen do dvou podsekcí:

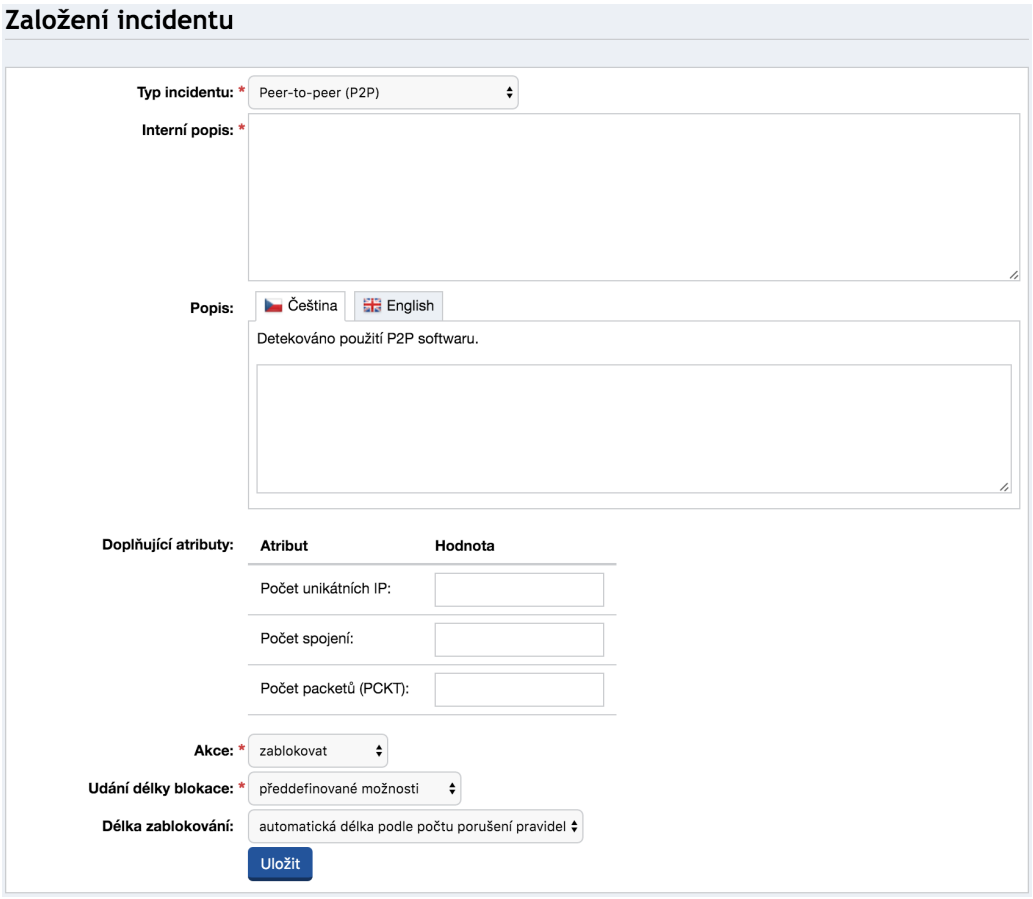
- aktivní incidenty a
- incidenty bez akce – incidenty, kde nebyla zvolena žádná akce. V této sekci jsou také zobrazeny automaticky zakládané incidenty z P2P sondy.

Vyřešené incidenty: zbylé incidenty, tj. buď vyřešené, nebo ty, kde byl uživatel již odblokován.

Přepínání incidentů probíhá pomocí tlačítek v pravém horním rohu přehledu.



Incident je možné založit z libovolného přehledu počítačů. Při zakládání incidentu je nutné zvolit jeho typ a zadat interní popis – ten se uživateli nezobrazí. Na základě typu incidentu se určuje výchozí akce, šablona popisku a to, zda se incident počítá jako porušení pravidel



Popis pro uživatele je možné zadat ve více jazycích. Je možné také doplnit dodatečné atributy – například typ detekovaného protokolu, počet detekovaných paketů a další. Tyto atributy jsou informativní a jsou definované pro každý typ incidentu zvlášť.

V položce akce je možné vybrat 3 hodnoty:

Žádná: incident bude pouze pro interní evidenci.

Zaslat upozornění: zašle uživateli informační e-mail. Pod políčkem se zobrazí náhled zprávy, která bude uživateli zaslána. Jako odesílatel zprávy bude vyplněn Váš e-mail.

Zablokovat: zablokuje uživatele. Při určení délky je možné vybrat jednu z předdefinovaných možností, zadat vlastní počet dní, případně zablokovat na neurčito. Pokud byla délka zadána, bude uživatel automaticky odblokován, v opačném případě musíte odblokování provést ručně.

K vytvořeným incidentům je v jejich detailu možné přidávat komentáře.

C.4 Ostatní funkce

V závislosti na úrovni oprávnění jsou správcům dostupné následující přehledy:

Seznam správců
Všichni správci zde mohou upravit své kontaktní údaje. Administrátoři systému navíc mohou upravovat oprávnění, přidělovat oprávnění novým uživatelům a oprávnění odebírat.

Uživatelé
Přehled uživatelů, kteří se do systému přihlásili. Je možné zobrazit detaily uživatele, případně zadat informaci, na které koleji bydlí. Pokud přijde ubytovaný s problémem, že mu systém nedovoluje registraci, pravděpodobně se do všech systémů ZČU ještě nezapsala informace o jeho ubytování. Ověřte, že na koleji skutečně bydlí (nejlépe platnou smlouvou o ubytování) a informaci do systému doplňte.

Fronta zpráv
Systém předává síťovým službám informace o registracích počítačů prostřednictvím konfiguračních zpráv. Pokud dojde k chybě, je pokus o konfiguraci několikrát opakován. Pokud některému uživateli nefunguje připojení k internetu, ačkoliv je již schváleno, zkontrolujte frontu zpráv.

Pokud je pro danou registraci ve frontě zpráva ve stavu `fail` (selhalo), prohlédněte si záznamy, které se ke zprávě váží, a případně ji zkuste opakovat. Pokud se ani po opakování zprávy problém nevyřeší, kontaktujte hlavního správce.

Letní akce
Umožňuje spravovat plánované letní akce (například MLJŠ či ArtCamp). Naplánováním akce se vybrané podsítě přepnou do zvláštního režimu, ve kterém

se mohou ubytovaní registrovat i bez Orion účtu. K registraci potřebují jen přístupové heslo, které jim bude poskytnuto při ubytování. Jejich registrace bude automaticky schválena a připojení bude ihned aktivní.

Registrační období
Umožňuje nadefinovat období, ve kterém se registrují počítače. Registrace všech počítačů vytvořené v některém z definovaných období se na konci tohoto období odstraní. Jednotlivá období se mohou překrývat. Pokud období nelze určit při registraci jednoznačně, vybere se to, které bylo definováno později.

V rámci období je možné na vybraných kolejích povolit letní (autoregistrační) režim – registrace jsou schvalovány automaticky a není potřeba zásah správce. V letních měsících je vhodné nakonfigurovat registrační období tak, aby respektovala harmonogram uzavření kolejí, který na konci června vydává SKM.

Typy incidentů
Správa typů incidentů. V této sekci je také umístěna správa délek trestů a možnost přiřazení délek ke skupinám kolejí.

Logy
Prohlížení záznamů. V přehledu záznamů lze filtrovat pomocí základních kritérií.

Přístup k API
Správa klientů, kteří mají oprávnění přistupovat k API. Pro každého klienta se vygeneruje jeho unikátní Client ID a Client Secret. Client Secret je v databázi uloženo pouze v zahashované podobě, proto si jej po vytvoření ihned uložte! Všichni klienti mají oprávnění pro čtení. Pokud mají mít přístup pro zápis, je potřeba jim přidělit oprávnění.

Z této sekce lze také spravovat tzv. „zdroje incidentů“. Lze nadefinovat externí systémy, do kterých se mají posílat informace o aktualizacích jimi vytvořených incidentů. Zdroj je poté nutné spárovat s klientem API.

Přístup k API
Správa podsítí a přepínačů. Přepínače již musejí být překonfigurované, pro načítání nových portů slouží konzolový příkaz `switch:load-sockets`.

D Rozhraní pro správu v databázi

Pro správu v databázi byl vytvořen pohled `hosts_view`, funkce `changelog_field_like` a `fetch_reference`.

Pohled `hosts_view`

Pohled obsahuje sloupce z následujících tabulek:

`hosts`: všechny sloupce, bez prefixu.
`users`: všechny sloupce, prefixovány řetězcem `user_`.
`sockets`: sloupec s názvem zásuvky: `socket_name`.
`rooms`: sloupce `id` a `name`, prefixovány `room_`.
`dormitories`: všechny sloupce, prefixovány `dormitory_`.
`dormitory_groups`: sloupec s názvem skupiny: `dormitory_group_name`.
`ports`: sloupce `id` a `name`, prefixovány `port_`.
`switches`: sloupce `id` a `hostname`, prefixovány `switch_`.
`subnetworks`: sloupce `id`, `network` a `vlan_num`, prefixovány `subnetwork_`.

Z pohledu je možné mazat řádky. Smazáním řádku se smaže registrace zařízení z tabulky `hosts`. Sloupce, které se váží k tabulce `hosts`, lze také upravovat. Sloupce vázané na tabulku `users` lze upravovat jen tehdy, jestliže se nemění vlastník registrace (tj. sloupec `user_id`).

Funkce `changelog_field_like(changelog, filed, value)`

Funkce slouží k ověření, zda pole zadaného seznamu změn (sloupec `changeset` tabulky `log_messages`) odpovídá určitému řetězci. Řetězec je vyhledáván jak v původní i v nové hodnotě a pro porovnání je použit operátor `LIKE`. Funkce vrací logickou hodnotu a má následující parametry:

`changelog JSONB`: Seznam změn z tabulky `log_messages`.
`field CHARACTER VARYING`: Pole, jež má být kontrolováno na shodu se zadanou hodnotou, např. „hostname“.
`value TEXT`: Hodnota, která má být vyhledávána, např. „vomacka%“.

V praxi lze funkci použít pro vyhledávání v seznamu zpráv:

```
SELECT * FROM log_messages
WHERE changelog_field_like(changeset, 'hostname', 'vomacka%');
```

Funkce `fetch_reference(table_oid, row_id)`
Funkce vrací záznam se zadaným ID z tabulky, která je specifikována pomocí OID. Záznam je ve formátu JSON. Funkci lze použít k získávání záznamů z nepřímých referencí, jak bylo uvedeno v kapitole 8 Návrh databáze.

Funkce může být použita například pro získání záznamu, ke kterému se váží komentáře napsané k určitému uživateli:

```
SELECT
  *,
  fetch_reference(context_resource_class, context_resource_id)
FROM user_notes
WHERE user_id = 42;
```

Tato nepřímá vazba je použita také v tabulce `log_linked_resources`, která uvádí vazby záznamů v logu k objektům v databázi, kterých se záznam týká.