Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Centrální správa linuxového firewallu

Roman Vohník

Diplomová práce

2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Roman Vohník**
Osobní číslo:	**I10426**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Centrální správa linuxového firewallu**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Teoretická část se zaměří na síťovou bezpečnost, firewally a jejich správu v operačním systému Linux. Bude rovněž vytvořen přehled dostupných aplikací pro správu firewallu.

V praktické části bude navržena a implementována aplikace pro centrální správu firewallu více počítačových stanic v lokální síti. Aplikace bude umožňovat vytvářet profily pro skupiny stanic a volitelné bloky pravidel, které se budou moci uplatnit dočasně na vyžádání. Aplikace bude mít pro ovládání webové rozhraní.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

SOSINSKY, Barrie. Mistrovství - počítačové sítě. 1. vydání. Brno: Computer Press, 2010, 840 s. ISBN 978-80-251-3363-7.
SUEHRING, Steve. Linux firewalls. 3rd ed. Indianapolis: Pearson Education, c2006, xvii, 533 s. ISBN 978-0-672-32771-1.
HARRIS, Shon, et al. Hacking – manuál hackera. 1. vyd. Praha: Grada Publishing, 2008. 400 s.
Kolektiv autorů. Linux - Dokumentační projekt. 4. aktualizované vydání. Brno: Computer Press, 2008. 1336 s. ISBN: 978-80-251-1525-1. URL textu: http://www.root.cz/knihy/linux-dokumentacni-projekt-4-vydani/stahnout/961/. URL obsahu: http://knihy.cpress.cz/p=actions&action=download/file&value=files&id=22019.
VRANÝ, Boleslav. Bezpečnost v digitálním věku [online]. 2004 [cit. 2009-10-21]. 2, 37 s. URL: http://www.bolekvrany.cz/downloads/security_cz.pdf.

Vedoucí diplomové práce: **Mgr. Tomáš Hudec**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2012

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury. Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

v Pardubicích dne 21. 8. 2013

Bc. Roman Vohník

Poděkování

Tímto bych chtěl poděkovat panu Mgr. Tomáši Hudcovi za podnětné rady a připomínky a za celkové vedení mé práce. Obzvláště si pak cením informací ohledně typografie a citací.

ANOTACE

Tato práce obsahuje popis zabezpečení sítí, firewallů a jejich správu v Linuxu. Hlavním tématem je pak návrh a vytvoření aplikace pro centrální správu linuxových firewallů.

KLÍČOVÁ SLOVA

GNU/Linux, síť, firewall, zabezpečení

TITLE

Central administration of Linux firewall

ANOTATION

This work describes the network security, firewalls and their management in Linux. Main part is about design and process of creation an application for central management of Linux firewalls.

KEY WORDS

GNU/Linux, network, firewall, security

Obsah

Úvod

Počítačové sítě vznikaly v době, kdy pronikalo využití výpočetní techniky do nejrůznějších odvětví a díky tomu bylo produkováno stále více digitálních dat. Tato data bylo čím dál častěji potřeba přemísťovat, nebo s někým sdílet a jedinou možností jak toho docílit, bylo pomocí datových nosičů (např. disket). Počítačové sítě tyto potřeby vyřešily a s příchodem internetu přinesly ještě daleko větší možnosti. S tímto rozšířením ale také vznikla nejen potřeba zabezpečit posílaná data (šifrování) ale také zabezpečit lokální sítě a samotné počítače proti cizímu přístupu. Pro prostředek, který takto chrání sítě a počítače se zažil výraz firewall[1].

Ochrana sítí tedy rozhodně není v dnešní době žádnou novinkou a tato práce si klade za cíl nejen popsat některé principy sítí, firewallů, jejich vývoj, možné útoky a ochranu proti nim ale také navrhnout a vytvořit software pro centrální správu linuxového firewallu. Ten by měl najít uplatnění například ve školních učebnách, nebo sítích, kde je vyžadována správa firewallů více stanic.

1 doslovný český překlad by byl „ohnivá zeď“, který i správně vystihuje podstatu ochrany (zdi) před něčím, se ale nepoužívá a je zažitý termín firewall

1 Počítačová síť

Jelikož jsou počítačové sítě nesmírně rozsáhlé téma, které by samo o sobě zaplnilo obsahem minimálně celou tuto práci, vysvětlím zde jen základní principy a technologie. Ty budou potřebné pro pochopení síťové bezpečnosti a firewallů. Pro tuto kapitolu jsem čerpal informace z knihy *Mistrovství – počítačové sítě* [1].

1.1 Síťové rozhraní a adresace

Slovo rozhraní obecně znamená propojení dvou prvků, látek, nebo médií. U síťového rozhraní tomu není jinak a znamená zde spojení mezi fyzickou přenosovou vrstvou (kabely, šíření vln, atd.) a vrstvou aplikační.

Někdo chápe síťové rozhraní pouze jako fyzickou síťovou kartu počítače, ale dá se chápat i jako celek spolu s ovladačem této karty a taktéž s virtuálním objektem v operačním systému.

Fyzické rozhraní

Fyzické síťové rozhraní je tedy přímo fyzická síťová karta (anglicky Network Interface Card, NIC), dnes již integrovaná do čipových sad počítačů a nejen tam. Síťová rozhraní obsahují dnes běžně televize, set-top boxy a spousty dalších zařízení.

Rozhraní tedy dokáže fyzicky komunikovat, neboli převádět data z/do formy kterou můžou cestovat sítí. Podle typu rozhraní se může jednat o nejběžnější drátovou technologii Ethernet, bezdrátovou WiFi, optické signály a další. Rozhraní pouze informace přenáší a maximálně manipuluje se záhlavím (obálkou), případně obsahuje vyrovnávací paměť (buffer), kam si postupně ukládá došlá data (pakety). O samotnou manipulaci s daty se pak stará operační systém a provádí se v procesoru[2].

Důležitou vlastností fyzického rozhraní je to, že je adresovatelné, aby bylo možné správně doručit data. Standardně se totiž data v síti rozesílají úplně všem. Pouze rozhraní, kterému jsou adresovaná, je přijme a ostatní rozhraní je zahodí. V sítích typu Ethernet se tato adresa nazývá MAC (Media Access Control) adresa a je to 48bitové číslo, které je výrobcem napevno zadané přímo do paměti rozhraní a lze pouze číst.

2 výjimkou jsou karty podporující zpracování již v zásobníku karty (TCP offload engine, TOE)

Logické rozhraní

Logickým síťovým rozhraním je myšlen propojovací bod mezi sítí a samotným systémem. Kromě případu rozhraní typu zpětné smyčky je každé logické rozhraní svázáno se svým fyzickým rozhraním. Rozhraní zpětné smyčky (dále pod anglickým názvem loopback) je virtuální rozhraní, které je emulováno systémem. Chová se stejně jako jakékoli jiné rozhraní a slouží pro komunikaci sama se sebou. Zní to zbytečně, ale je to způsob, jak například komunikovat se serverovou aplikací běžící lokálně.

Dalším případem jsou linuxová virtuální rozhraní, kdy nad jedním fyzickým existuje více logických rozhraní. Takže se pak takovéto zařízení s jednou síťovou kartou jeví jako by měla karty dvě a každé toto rozhraní může navíc patřit do jiné podsítě. Tato rozhraní jsou v linuxových systémech standardně identifikována dvojtečkou a číslem v názvu, viz v následujícím příkladu rozhraní **wlan0:1**.

Příklad

V následujícím příkladu je ilustrativní výpis příkazu ifconfig, kde je vidět rozhraní zpětné smyčky lo a logická rozhraní wlan0 a wlan0:1, které je virtuální. Hwadr znázorňuje MAC adresu zařízení.

```
lo        Link encap:Místní smyčka
          inet adr:127.0.0.1  Maska:255.0.0.0
          inet6-adr: ::1/128 Rozsah:Počítač
          AKTIVOVÁNO SMYČKA BĚŽÍ  MTU:16436  Metrika:1
          RX packets:11318 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11318 errors:0 dropped:0 overruns:0 carrier:0
          kolizí:0 délka odchozí fronty:0
          Přijato bajtů: 4121101 (4.1 MB) Odesláno bajtů: 4121101 (4.1 MB)

wlan0     Link encap:Ethernet  HWadr 00:21:6a:12:34:56
          inet adr:192.168.1.4  Všesměr:192.168.1.255  Maska:255.255.255.0
          inet6-adr: fe80::221:6aff:fe19:7912/64 Rozsah:Linka
          RX packets:2658775 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1768686 errors:0 dropped:0 overruns:0 carrier:0
          kolizí:0 délka odchozí fronty:1000
          Přijato bajtů: 3174080 (3.1 MB) Odesláno bajtů: 228914 (228.9 KB)

wlan0:1   Link encap:Ethernet  HWadr 00:21:6a:12:34:56
          inet adr:192.168.1.66  Všesměr:192.168.1.255  Maska:255.255.255.0
```

Adresace

Aby bylo možné rozhraní přesouvat mezi sítěmi (mobilní zařízení nebo prostá výměna rozhraní) tak kromě fyzické adresy MAC je každému rozhraní přiřazena tzv. síťová adresa. Přiřazovat je můžeme libovolně podle potřeb, pouze nesmí v daném segmentu sítě dojít ke

kolizi více stejných adres. Adresy můžeme rozhraním nastavit staticky, nebo dynamicky nechat přiřazení na protokolu DHCP[3] pokud máme v síti nastavený DHCP server.

1.2 Referenční modely

Referenčními síťovými modely popisujeme síťové transakce mezi dvěma systémy a slouží především ke standardizaci (zařízení a služeb), protože různé síťové komponenty mohou pracovat na různých úrovních a existuje mnoho výrobců síťových zařízení, které mezi sebou musí komunikovat. Modely neurčují implementaci, pouze definují principy.

ISO/OSI

Tento model vytvořila mezinárodní organizace ISO v roce 1984 a dodnes slouží jako nejlepší prostředek pro vysvětlení principů komunikace dvou síťových zařízení. Definuje sedm vrstev, které jsou na sebe navrstvené. Každá vrstva využívá služeb sousední nižší vrstvy a naopak poskytuje služby své sousední vyšší vrstvě, není tedy dovoleno přeskakovat vrstvy. Každá vrstva k datům předchozí vrstvy přidává obálku se svými daty a předává nižší vrstvě. Na opačné straně se zase analogicky provádí operace v opačném pořadí, data se rozbalují z obálek a posílají se vyšším vrstvám.

1. Fyzická vrstva – specifikuje fyzické vlastnosti přenosového média (kabely, optika, rádiové vlny, modulace, …),
2. linková vrstva – hlavně adresace hardwaru (MAC),
3. síťová vrstva – adresace systémů (IP),
4. transportní vrstva – přenos dat mezi uzly,
5. relační vrstva – relace mezi odesílatelem a příjemcem,
6. prezentační vrstva – transformace dat do podoby přijatelné aplikací (kódování, komprimace, struktura dat, …),
7. aplikační vrstva – spojení samotných aplikací pomocí sítě.

TCP/IP

Sítě založené na modelu OSI se prakticky téměř nepoužívají, ale stále najdou využití pro popis principů komunikace. Nejběžnějším modelem, který se dočkal praktického využití se

3 DHCP – Dynamic Host Configuration Protocol – protokol z rodiny TCP/IP pomocí kterého si zařízení zažádá síť o přidělení síťové adresy

stal model TCP/IP. Ten v porovnání s modelem OSI obsahuje vrstvy jen čtyři, ale velmi podobné.

1. Vrstva síťového rozhraní (hostitelská) – je v podstatě to samé jako fyzická a linková vrstva OSI modelu, neboli obstarává hardwarovou komunikaci a MAC adresaci (do hlavičky přidává zdrojovou a cílovou MAC adresu),
2. internetová – odpovídá síťové vrstvě OSI modelu, specifikuje protokol IP,
3. transportní vrstva – transportní a relační vrstva OSI modelu a specifikuje protokoly UDP a TCP,
4. aplikační vrstva – poslední vrstva, která odpovídá prezentační a aplikační vrstvě OSI modelu nad kterou jsou postaveny aplikační protokoly.

Jak je vidět, tak tento model vlastně zjednodušuje OSI model, za což je často kritizován, protože není dostatečně obecný a díky tomu muselo vzniknout mnoho různých protokolů. Navzdory tomu, jak již bylo řečeno, je prakticky používán a tvoří základ dnešního Internetu.

1.3 Internetové transportní protokoly

Jedná se vlastně o rodinu protokolů TCP/IP založených na stejnojmenném modelu. Ze složeného názvu TCP a IP by se zdálo, že se jedná pouze o tyto dva protokoly, ale opak je pravdou. Dříve do této rodiny patřily základní protokoly IP, UDP, TCP, (R)ARP, ICMP, IGMP a základní aplikační protokoly, například FTP, Telnet, HTTP, POP, DNS, atd. To je ale pouze základ a v dnešní době si pod pojmem TCP/IP musíme představit ještě spoustu dalších protokolů. Uvedeny a stručně popsány budou jen ty nejdůležitější.

Protokoly první vrstvy

Tato vrstva byla v souvislosti s TCP/IP často opomíjená, ale nachází se na ní několik různých protokolů. Nejznámějším je samozřejmě Ethernet, ale nesmíme zapomenout na protokoly PPP, WLAN, Frame Relay a další.

Protokoly internetové vrstvy

Internet Protocol

Hlavním a nejznámějším protokolem této vrstvy je Internet Protocol, ve zkratce IP. Tento protokol slouží ke komunikaci dvou počítačů napříč internetem, což může znamenat přenos dat přes celou řadu různých sítí. Označení IP je ale už nedostatečné, protože se tento protokol

nachází ve dvou verzích a je označován jako IP verze 4 nebo novější verze 6, ve zkratkách IPv4 a IPv6.

Rozdíl mezi těmito verzemi je markantní. IP verze 4 používá pro adresování takzvanou adresu IP o velikosti 32 bitů, kde se při zápisu jednotlivé bajty oddělují tečkou (např. 192.168.0.1). V době, kdy se rozmáhá síťové připojení ve všech možných zařízeních, musí nutně při této velikosti dojít k vyčerpání adres, jelikož adresy musí být celosvětově unikátní. Během doby bylo vymyšleno několik technik, které tento problém značně oddálily. Jednou z nich je zrušení fixního osmibitového rozdělení IP adresy na adresu sítě a na adresu konkrétního zařízení v síti. Tím je možné přidělovat menší a lépe se hodící rozsahy adres. Další technikou je překlad adres, označován zkratkou NAT z anglického názvu Network Address Translation. Podstatou je to, že se adresy z celé vnitřní sítě překládají (maskují se) na adresu hraničního zařízení. Vnitřní adresy tedy nemusí být vůči internetu unikátní, stačí když jsou unikátní uvnitř sítě. Nevýhodou tohoto řešení je, že tyto adresy nejsou z Internetu přímo dostupné, což způsobuje některým aplikacím problémy.

IP verze 6 je novější a řeší nedostatek adres rozšířením velikosti adresy na 128 bitů, což je mnohonásobně více než u verze 4. Při vývoji tohoto protokolu také došlo k přepracování obsahu hlavičky, odkud zmizela spousta příznaků, které se téměř nepoužívaly. V hlavičce jsou zde jen ty nejdůležitější informace, jako je verze protokolu, velikost dat, počet skoků, identifikace toku, třída dat a také příznak další hlavičky. Tento příznak specifikuje další záhlaví, kde se mohou nacházet rozšiřující informace protokolu IPv6 nebo už přímo záhlaví TCP.

ICMP protokol

Je to servisní protokol který je součástí protokolu IP. Jeho zkratka je složena z anglického názvu Internet Control Message Protocol. Je to velice důležitý protokol pro signalizaci situací a stavů zařízení v síti. Jeho pakety se balí do IP protokolu a přidává svou ICMP hlavičku, která má vždy 8 bajtů a jsou v ní obsaženy typ a kód ICMP paketu spolu s kontrolním součtem. Typ rozděluje ICMP zprávy do několika kategorií a kód slouží pro jemnější rozdělení. Tímto protokolem si například může kdokoli zažádat o signalizaci stavu aplikace (echo request, což je typ 8 kód 0) a aplikace odpoví takzvaným echem, typ 0. Další z možných zpráv jsou například zpráva o nedostupnosti nebo nenalezení sítě/uzlu/portu, zpráva o přetížení nebo zprávy pro práci se směrováním, synchronizační zprávy atd.

(R)ARP

Protokol ARP (Address Resolution Protocol) a reverzní RARP slouží k překladu hardwarové adresy MAC na adresu sítě IP a obráceně. Tohoto protokolu se využívá, když chceme poslat data počítači ve stejné sítí a známe pouze jeho IP adresu, proto operační systém vyšle ARP požadavek, na který odpoví cílový systém. Náš operační systém přijme tuto odpověď ve které je obsažena MAC odesílatele a uloží si ji do vyrovnávací paměti, aby nemusel toto vyhledávání provádět při každém požadavku.

Protokoly transportní vrstvy

Protokolem IP předchozí vrstvy zaručíme pomocí IP adresy doručení správnému systému, nijak ale není určeno jaké aplikaci v systému daná data patří. To nám zaručí protokoly transportní vrstvy, které obsahují číslo portu podle kterého systém pozná které aplikaci data náleží. Číslo portu je dvoubajtové a je tedy z rozsahu 0 až 65535. Nejběžnějšími protokoly jsou UDP a TCP a oba samozřejmě balí data se svou hlavičkou do protokolu IP.

UDP

UDP neboli User Data Protocol je nejjednodušší forma datagramové služby. Nevytváří totiž spojení a jeho záhlaví obsahuje pouze zdrojový port, cílový port, délku dat a kontrolní součet (jehož obsah je nepovinný). Tento protokol tedy nijak nezaručuje jisté doručení ani správnost dat, to vše je přenecháno vyšší vrstvě, která se o to musí postarat sama, pokud to vyžaduje.

TCP

TCP neboli Transmission Control Protocol má stejné poslání jako UDP dopravit data ke správné aplikaci. Ale jak název napovídá, tento protokol vytváří spojení mezi aplikacemi. Při tomto spojení dochází ke kontrole obsahu pomocí kontrolních součtů a také ke kontrole, zda dorazily všechny odeslané pakety. Pokud se kdekoli po cestě zjistí poškození, nebo dojde ke ztrátě některé části dat (pakety jsou číslovány), odesílatel se to dozví a pokusí se data odeslat znovu. Aplikace se o toto tedy starat vůbec nemusí.

Díky tomu musí být protokol trochu složitější, takže v záhlaví se nachází kromě položek, které jsou u UDP protokolu, ještě další položky. Musí zde být pořadové číslo odesílaného a také přijímaného segmentu, několik bitů pro signalizaci stavu spojení a další položky. Nejdůležitější příznaky se označují jako ACK, SYN, FIN a RST. Navazování spojení, kterému se také říká handshake (podání ruky), probíhá ve třech krocích. Klient, který chce navázat spo-

jení, odesílá první paket, kterému nastaví příznak SYN bez nastaveného příznaku ACK. Odpovědí mu je paket s příznakem ACK+SYN, na který ale ještě musí odpovědět paketem s příznakem ACK. Teprve nyní je na obou stranách spojení vedeno jako obousměrně plně funkční (stav established). Velice podobně se spojení také uzavírá, pouze se na to použijí pakety s příznaky FIN, FIN+ACK, ACK. Tedy i na ukončení se dohadují obě strany. Spojení ale může být tzv. polo-otevřené kdy jedna strana pošle FIN a už nemůže posílat, ale druhá strana může ještě data stále číst dokud sama spojení neukončí.

Protokoly aplikační vrstvy

Aplikačních protokolů existuje veliké množství, proto zde budou uvedeny pouze ty nejpoužívanější a budou popsány jen stručně (pro účely této práce není detailnější popis těchto protokolů potřeba). Tyto protokoly mají společnou minimálně jednu vlastnost a tou je, že využívají jednoho z protokolů nižší vrstvy – UDP nebo TCP. Pro běžné aplikace a jejich aplikační protokoly se zažila čísla portů na kterých obvykle běží. Tyto porty nejsou rezervované a jakákoli aplikace může naslouchat na jakémkoli portu v celém rozsahu, je to tedy pouze obecně doporučované nastavení a většina aplikací s těmito porty počítá. V unixových systémech se nachází soubor */etc/services*, který obsahuje seznam těchto běžně používaných aplikací vždy s portem a protokolem (TCP nebo UDP), který standardně využívají. Není neobvyklé, pokud nějaká aplikace využívá více portů, nebo dokonce pokud používá stejné číslo portu obou protokolů TCP i UDP.

DNS

Protokol Domain Name System (dále jen DNS) slouží k jednoduššímu adresování zařízení v Internetu. Využívá protokol UDP i TCP na portu 53. IP adresy jsou pro běžného člověka těžko zapamatovatelné, proto byl vyvinut doménový systém, kdy je zařízením přiřazeno doménové jméno. Tato jména jsou distribuována v celosvětové jmenné databázi a domény musí být samozřejmě unikátní, proto jsou spravovány nadnárodní organizací IANA (Internet Assigned Numbers Authority), respektive lokálními registrátory.

Doménové jméno je možné využít na všech místech, kde je očekávána IP adresa, a operační systém za nás, právě pomocí DNS protokolu, vyřídí překlad doménového jména na IP adresu (případně opačně). Operační systém samozřejmě udržuje vyrovnávací paměť překladů, aby se stále nemusel dotazovat. V systému tedy musíme kromě IP adresy a její masky nastavit ještě IP adresu nějakého DNS serveru. Těchto adres můžeme hlavně kvůli redundanci uvést

více, případně použít adresu vlastního DNS serveru lokální sítě. Tam můžeme udržovat doménová jména ve vnitřní síti, pro velmi malé sítě existuje ve většině operačních systémů soubor hostů (v unixových systémech soubor */etc/hosts*), kde lze manuálně IP adresám v síti doménová jména přiřadit.

Poštovní protokoly a web

Mezi poštovní protokoly patří protokoly SMTP, POP a IMAP.

Simple Mail Transfer Protocol (SMTP) využívá port 25/TCP a používá se k přenosu mailových zpráv. Ať už od uživatele, nebo mezi jednotlivými SMTP servery.

Pro příjem pošty je však potřeba jiného protokolu, proto vznikl Post Office Protocol (POP) a Internet Message Access Protocol (IMAP). POP je jednodušší variantou, protože pracuje pouze v takzvaném offline režimu, kdy se veškerá pošta stahuje ke klientovi. Využívá port 110/TCP. IMAP oproti tomu podporuje offline i online režim, dokáže tedy poštu synchronizovat a podporuje spoustu dalších pokročilejších možností. Využívá port 143/TCP.

Nejznámějším webovým protokolem pro výměnu informací (dokumentů) je protokol HTML. Existuje v několika verzích a standardně využívá port 80/TCP, případně 8080/TCP. Původně to byl velice jednoduchý nástroj pro prezentaci informací (statických prezentací), ale postupem času se k němu přidaly další technologie a protokoly (MIME, XML, JSON, a další). Tím rozšířily možnosti například o přenos jakéhokoli typu souboru, nebo zpřístupnění webových služeb.

Vzdálené připojení

Pro vzdálené připojení k terminálu se dříve používaly známé protokoly telnet, rsh, rlogin a podobně. V dnešní době je nahradil zabezpečený protokol SSH, který standardně využívá port 22/TCP.

Výměna a sdílení souborů

Samozřejmou potřebou je výměna souborů po síti, jejich sdílení, synchronizování apod. Pro tuto specifickou potřebu bylo vytvořeno mnoho protokolů, mezi nejznámější patří FTP, NFS a SMB.

Protokolů existuje daleko větší množství, stejně tak jako existují zabezpečené protokoly HTTPS, IMAPS a podobně. Většina z nich bude popsána v některé z následujících kapitol věnujících se bezpečnosti.

2 Síťová bezpečnost

Při vzniku původních aplikačních protokolů nebyl téměř vůbec kladen důraz na zabezpečení přenosu dat, protože prostě počítačové sítě byly velmi malé. Internet vznikal před desítkami let jako americká armádní síť, dále se rozšířil do tamních univerzit a velkých korporací. Až potom se začal neskutečnou rychlostí rozrůstat do ostatních koutů světa a mezi obyčejné lidi. Proto tehdy nebylo tolik rizik, jednoduše nebylo tolik dostupných sítí, které by někdo mohl napadnout. Navíc nebylo ani tolik lidí, kteří by této problematice rozuměli a také ani nebylo z počátku tolik možností jak by někoho mohli ohrozit.

Pro lidi, kteří se snažili do detailu porozumět problematice a ze zvědavosti například pronikali do privátních sítí, se vžil název „hekři“ (anglicky hackers). Tito lidé dost často kradli pouze dokumentace k nějakým zařízením, softwaru, či protokolům samotným, aby jim lépe porozuměli.

Na sítích se ani nenacházelo tolik tajného a choulostivého materiálu jako dnes, nehrozilo zneužití identit a podobně. Právě v této době vznikaly protokoly, jež jsou kostrou dnešního Internetu a jsou popsány v předešlé kapitole. Byly navrženy pro aktuální potřeby. Jenže počítačové sítě a informační technologie obecně se postupně dostávaly mezi širokou veřejnost. A mezi lidmi se vždy najde někdo, kdo rád krade, škodí, nebo ničí. Ať už bez důvodu, nebo ještě hůře s úmyslem někoho poškodit záměrně. A čím více lidí mělo, má a bude mít přístup k Internetu a dostupné znalostí z informačních technologií, tím více se objeví i špatných lidí. Proto se pro útočníka, který nějakým způsobem narušuje počítačovou bezpečnost, ustálil odborný výraz „cracker“. Neodborná veřejnost ale bohužel často výrazy „hacker“ a „cracker“ nerozlišuje.

V ohrožení vždy byly a budou firmy a organizace připojené do veřejné sítě. Pokud nevěnují dostatečné finance do zabezpečení své sítě, tak se vystavují riziku nejvyššímu. A tím je ukradení firemního nápadu, neboli know-how, což bývá mnohdy několikanásobně dražší, než nějaký finální produkt, který z takového nápadu vychází.

Cílem útoků jsou ale také velice často samotní uživatelé internetu. Útok může směřovat například na internetové bankovnictví nebo se může zaměřit na využití uživatelova počítače jako takového pro další aktivity. Díky síti napadených počítačů po celém světě může pak útočník vyřadit různými metodami jiné služby nebo celé sítě z provozu.

2.1 Zranitelnosti sítí

Počítačové sítě a připojené zařízení jsou zranitelné mnoha různými způsoby. Nekladu si za cíl popsat všechny možnosti, ani je popisovat do největších detailů, jde jen o hrubý přehled pro nastínění problematiky. Celá tato kapitola vychází převážně z knihy Hacking bez tajemství [2].

Odposlech

Většina uživatelů dnešního Internetu a sítí obecně se domnívá, že jejich veškerá komunikace je tajná a že veškeré získané informace jsou důvěryhodné. Opak je bohužel pravdou. Sítě již od počátků disponují silnými mechanismy pro to, aby fungovaly, i když bude několik směrovacích zařízení po cestě mimo provoz. Těmi mechanismy je například směrování jako takové, ICMP protokol, nebo i protokol TCP. Sítě, do kterých jsme připojeni, jsou stále většinou sdíleným médiem, jak již bylo v dobách sítí typu Token Ring, nebo při použití zařízení typu Hub. Tehdy data v lokální síti jednoduše putovala mezi počítači a přijal je pouze ten, kterému patřila. K odposlechu cizích dat stačilo útočníkovi pouze nastavit jeho síťové rozhraní do režimu, kdy přijímalo data bez rozdílu. Od té doby se mnoho změnilo a dnes se již používají pokročilejší síťové prvky, jako jsou přepínače a směrovače, které jsou už součástí téměř každé domácnosti. Ty už data směrují inteligentněji, ale opět je lze určitými technikami odposlechnout.

Příkladem může být například technika směrování zdrojem, kdy je útočník schopen paketům přesně nastavit, kudy se mají směrovat až ke svému cíli. To samotné lze využít k testování sítí, ale také se to dá zneužít tím, že si útočník navíc zfalšuje IP adresu. Tu si nastaví takovou, jako má síť, ve které je připojena oběť. Pokud po cestě žádné zařízení takovýto paket neodfiltruje, tak jej cílový počítač bez problémů přijme. Bude se totiž opravdu podle zdrojové adresy domnívat, že pochází z jeho lokální sítě. Záleží pak na protokolu a na tom, jak je paket dobře zfalšován, jestli je ho možné detekovat na cílovém stroji, nebo ne. Takovéto pakety lze zjistit především na hraničním směrovači. Ale v případě využití více hraničních směrovačů (redundance) nemusí být detekce spolehlivá.

Další technikou může být přesměrování pomocí ICMP. To může nastat, pokud máme v síti více směrovačů. Pokud není výchozí směrovač ideální pro cestu k cíli, tak se tento domluví s druhým směrovačem a pomocí ICMP přesměrování nás na něj odkáže. Toho ale může využít útočník, který odposlouchává síť některou jinou technikou. Pokud zjistí, že se tážeme prvního

směrovače, ihned nám pošle paket s ICMP přesměrováním a zfalšovanou adresou směrovače. Tím donutí náš počítač posílat veškerá data přes útočníka, který má tak daleko více možností. Pokud tedy nepotřebujeme v síti využít více směrovačů, je bezpečné vypnout podporu takovýchto paketů. Pokud více směrovačů máme, je potřeba důkladně nakonfigurovat firewall.

Dalším zneužitelným protokolem může být i tak nízkoúrovňový protokol, jako je ARP. Tyto pakety lze určitým způsobem zaslat jak oběti, tak jeho cílovému stroji spolu s našimi zfalšovanými adresami. To donutí oba dva zasílat pakety nám, v domnění, že se jedná o cílový stroj. Před tímto útokem pomůže v podstatě jen pevné nastavení MAC adres počítačů s kterými chceme komunikovat. To je ale v dnešní době dosti nepraktické.

Podobným způsobem lze zneužít i protokol DNS pro překlad doménových jmen. Až dnes dochází k zabezpečování tohoto protokolu dodatečnými mechanismy pomocí certifikátů. Ale stále existuje spousta doménových serverů, které takto zabezpečené nejsou. Proto je možné vydávat se za daný DNS server a oběti zaslat odpovědi s podvrženými adresami, které budou směřovat opět na náš stroj.

Dnes tolik rozšířené bezdrátové technologie WiFi byly donedávna maximálně nebezpečné. Jelikož došlo k jejich masovému rozšíření a výrobci koncových zařízení pro domácnost usnadnili jejich používání, je tato technologie součástí téměř každé domácnosti. Z počátku ale výrobci zaspali a zařízení nastavovali v základním režimu bez jakéhokoli zabezpečení. Nezkušení uživatelé si zařízení pouze zprovoznili a o možnosti zabezpečení se nezajímali. Proto vznikalo tolik nezabezpečených sítí, které mohl odposlouchávat téměř kdokoli. Situace se ale zlepšila a většina dostupných sítí je již zabezpečena alespoň minimálním zabezpečením WEP. To je ale dávno překonáno a i se složitým heslem trvá jeho prolomení zkušenějšímu útočníkovi minimální čas. Proto je potřeba věnovat zabezpečení takových sítí chvíli času a nastavit nejlépe nejvyšší dostupné zabezpečení, jaké zařízení podporuje. Většinou to bývá šifrování WPA2. V rozsáhlejších sítích je navíc ještě standardem ověřování oproti serveru Radius apod.

Při odposlechu má tedy útočník přístup k datovému přenosu a veškerá nešifrovaná komunikace je velice dobře čitelná. Nešifrovaných protokolů se ještě dnes využívá veliké množství, například HTTP, FTP a další. Bezpečnostní riziko ale minimalizujeme, když použijeme místo těchto protokolů nějakou jejich zašifrovanou náhradu, viz kapitola 2.2. Tyto protokoly jsou samozřejmě bezpečné, ale stoprocentně bezpečné není nic. Pokud bude útočník odposlou-

chávat od samého počátku komunikace a dostatečně dlouho, může u některých protokolů odhadnout nebo vypočítat šifru, kterou zpětně data rozšifruje. Každopádně je to ale několikanásobně složitější než u běžných protokolů.

MITM

Některé útoky popsané u odposlechů by již bylo možné klasifikovat názvem man-in-the-middle, neboli muž uprostřed. Znamená to, že útočník některým dostupným způsobem docílil toho, že data od oběti putují přes útočníkův počítač, který je dále směruje k původnímu cíli. To umožňuje zmiňovaný odposlech, ale dá se toho využít k pokročilejším útokům. Lze totiž docílit toho, aby data i na zpáteční cestě putovala přes útočníka. Ten má pak příležitost dané spojení nejen odposlouchávat, ale i ukrást. Odposlechem mohl data pouze prohlížet, ale pokročilejšími technikami je možné data i modifikovat.

Nejjednodušším příkladem může být únos spojení. Pokud oběť používá například aplikace rlogin a rsh pro vzdálenou správu serveru, může útočník jednoduše takové připojení převzít a provádět zde jakékoli příkazy. Pro zmatení odpojené oběti lze navíc předstírat některé funkce terminálu. Proti tomuto se lze opět bránit za použití jiného protokolu. Pokud by oběť použila protokolu SSH-2, byla by komunikace šifrována. A pokud by do ní útočník zasáhl, protokol by to poznal a spojení ukončil.

Jak již bylo řečeno, nic není stoprocentně bezpečné. V případě protokolů SSH-1, nebo HTTPS mohlo dojít k útoku, při němž útočník popisovanými metodami přesměruje provoz přes sebe. Pokud se pak oběť pokusí připojit, útočník toto připojení přijme a na druhé straně vytvoří nové připojení k původnímu serveru s vlastním klíčem. Uprostřed má data rozšifrována.

Samozřejmě není protokol SSH tak jednoduché obelstít, proti takovým útokům disponuje databází známých otisků cizích počítačů, kterým důvěřuje. Pokud se mezi naši komunikaci vloží útočník, tak daný otisk sedět nebude a oběť bude varována. Bohužel jsou někteří lidé zvyklí jakékoli varování přeskakovat a odsouhlasí jej.

Podobné ověření obsahuje protokol SSL v případě zabezpečeného HTTPs. Tady nás upozorní přímo webový prohlížeč. V obou případech je obranou obezřetnost k varováním spojených s certifikáty. V případě SSH je navíc doporučováno používat verzi 2, která zavádí Diffie-Hellmanův algoritmus. Ten umožňuje to, aby se komunikující strany domluvily na šifrovacím klíči tak, že i když bude tato domluva odposlechnuta, není doposud možné takový klíč

zrekonstruovat[4]. Samozřejmě není vyloučené, že nebudou v budoucnu existovat techniky, které i toto překonají. Nebo, že neexistují již dnes, jen nejsou zveřejněny.

Fyzický přístup

Pokud má útočník fyzický přístup k počítačům, nebo zařízením ve vnitřní síti, nabízí se mu mnoho možností k útoku. Jednak se dnes dají velice snadno získat takzvané „keyloggery“, což jsou hardwarová zařízení, která se připojí nejčastěji mezi počítač a klávesnici. Tato zařízení pak nahrávají stisky kláves a tedy i zadávaná hesla. Ale lze napadnout i samotné počítače, buď k průniku do sítě, nebo k průniku do samotného počítače a nasazení nějakého škodlivého softwaru. Tyto možnosti ale nesouvisí s tématem této práce, proto nebudou do detailu vysvětlovány. Nelze se totiž proti nim bránit firewally.

Útok na služby

Službami jsou zde myšleny takové, které jsou poskytovány aplikacemi po síti. Využívá se pro ně také označení servery, které je ale možné chápat jako označení pro fyzický stroj poskytující takovéto služby. Hovoříme tedy o aplikaci, která běží obvykle jako nepřetržitý[5] systémový démon, kterému je operačním systémem přiřazen síťový port. Říkáme pak, že tento démon na tomto portu naslouchá. Takovéto služby neposkytují jen serverové operační systémy, ale i obyčejné osobní operační systémy. Jedná se o běžné služby vzdálených síťových disků, vzdáleného přihlášení, tisku a podobně. Tyto služby běží na pozadí většiny operačních systémů standardně, a i když je člověk nepoužívá a není si jejich existence vědom, tak jsou běžně dostupné v síti.

Takto dostupné aplikace musí být samozřejmě náležitě zabezpečeny. Čím kritičtější služba to je, tím by měla být lépe zabezpečena a ideálně by měly být zabezpečené všechny. Měly by data posílat v zašifrované formě a mely by podporovat autentizaci a autorizaci. Bohužel to často není pravda, zvláště proto, že si to služby musí řešit po svém. Jakákoli chyba v implementaci služby pak může vést k útoku na daný stroj. Takových chyb bývá objevováno poměrně velké množství a už záleží jen na druhu nalezené chyby a dovednostech útočníka. Kritických chyb je možné využít například i k obcházení autentizace a vzdálené spouštění cizího kódu, nebo eskalaci práv.

---

4 To ovšem platí v případě, že před útokem měl již klient klíč serveru, takže může ověřit pravost.

5 V případě využití super-démonu inetd démon neběží neustále, ale je spouštěn až po přijetí požadavku na daném portu.

Útokům takového typu se říká zero day, protože jej umožňují chyby, které nebyly doposud známe a opravené. Je proto nutné pravidelně aktualizovat bezpečností opravy veškerých poskytovaných služeb. Pokud navíc některé služby nevyužíváme, není důvod aby vůbec běžely. Proto je vhodné jejich běh v systému vypnout. Pokud je ale využíváme, je možné je zabezpečit firewallem.

DoS, DDoS a další

Pod zkratkou DoS se skrývá název Denial of Service, což v překladu znamená odmítnutí služby. To znamená, že útočník docílí toho, že nějaká služba přestane přijímat požadavky a bude nějakou dobu nedostupná. Tohoto stavu lze docílit různými technikami, všeobecně jde ale většinou o vyčerpání nějakého zdroje serveru, který danou službu poskytuje. Zdrojem může být jak síťová konektivita, počet spojení, velikost mailové fronty, tak i procesorový čas serveru.

Mezi nejjednodušší typy DoS útoku patří zahlcení linky oběti. K tomu lze využít možností v návrhu běžných protokolů. Například servisní protokol ICMP, který se běžně využívá a jeho zprávy jsou veliké typicky jen několik bajtů. Ve specifikaci protokolu je ale uvedena dostupná velikost až 65 KiB, a pokud zašleme takto velký dotaz, cíl nám odpoví stejně velkou odpovědí. Tím je možné zahltit linku oběti.

Další možností je využít SYN útoku, neboli zasílání TCP paketů s nastaveným příznakem SYN značícím počátek spojení. Server alokuje prostředky a odpoví paketem SYN+ACK, jenže odpověď mu nedorazí. Místo toho mu zašle útočník spoustu dalších SYN paketů. Serveru po určité době dojde zásobník polo-otevřených spojení a znemožní mu to přijímat další spojení.

Popsané metody jsou jen několika z mnoha dostupných a navíc stále přibývajících možností. Pokud se objeví nová metoda, bývá jedinou možností omezení specifického toku paketů fierwallem, ale na některé metody jsou i firewally krátké. Například proti útoku zahlcení SYN pakety existovalo jediné doporučení. Tím bylo zvětšit zásobník SYN paketů a zkrátit dobu zahazování neplatných. Až později byl vyvinut mechanismus SYN cookies, který zajišťuje, že SYN pakety jsou ze zásobníku odstraňovány ihned, a až přijde potvrzovací ACK paket, je z něj rekonstruován původní SYN.

Pokud byl útok veden z jedné nebo několika málo adres, bylo možné tyto adresy blokovat a útok zastavit. Vyvinul se ale druh DoS útoku, který je distribuovaný, proto název DDoS.

Tento útok je veden z obrovského množství adres, které pak není tak jednoduché blokovat. Útočníci samozřejmě nedisponují tisíci počítači po celém světě, ale našli si způsob jak takovéto počítače získat.

Existují takzvané „botnety“, což je síť počítačů napadených nějakým škodlivým softwarem. Majitelé těchto počítačů o tom většinou neví a byli nakažení díky bezpečnostním dírám v různých aplikacích. Útočník pak centrálně tyto počítače ovládá a může provádět distribuovaný útok. Tím, že má k dispozici tisíce počítačů po celém světě, nemusí být datový tok z jednoho počítače tak veliký, takže nebývá dlouhou dobu odhalen. Takovýto útok lze na cílovém serveru velmi špatně odfiltrovat.

Jak DoS útoků, tak předešlých útoků a slabin je obrovské množství a jistě existuje spousta dalších, které jsem zde nezmínil. Byly ale uvedeny pravděpodobně ty nejdůležitější spolu s nástinem toho, jak se proti danému druhu útoku dá chránit. Zdatný síťový administrátor by měl tyto útoky znát a rozumět jim, aby dokázal svou síť co nejlépe chránit všemi možnými prostředky, včetně firewallu.

Čerpal jsem ze zdroje [3], kde lze nalézt podrobnější informace a další typy útoků.

2.2 Zabezpečení protokolů

Některým uvedeným útokům a zranitelnostem se lze chránit prostým použitím zabezpečené verze protokolu. V ideálním světě by všechny používané protokoly měly šifrovat data, která přenáší a také ověřovat identitu protistrany. I když je spousta takových protokolů dostupných, tak stále existuje mnoho uživatelů, kteří dají přednost jednoduché konfiguraci před vyšším zabezpečením. Bohužel se takoví lidé najdou i ve firemní sféře. Příkladem může být protokol FTP pro přenos souborů. Přitom jeho zabezpečené alternativy existují již řadu let a jejich konfigurace není o mnoho složitější.

SSH

SSH, neboli Secure Shell je protokol založený na bázi TCP/IP protokolu. Tento název je poněkud zavádějící, protože tento protokol není pouze bezpečnou příkazovou řádkou, ale obsahuje další užitečné funkce, které je možné využít k úplně jiným potřebám. Historicky existovala první verze, nazývaná SSH-1, ale dnes je doporučována pouze novější verze SSH-2, která navíc přidává více možností zabezpečení a bezpečný přenos souborů pomocí sftp.

Základní podstata tohoto protokolu je specifikace toho, jak má probíhat zabezpečená komunikace po síti. Pro bezpečnou komunikaci po síti je potřeba několik vlastností. První je autentizace, neboli bezpečné určení vzdálené identity. Jednoduše řečeno, ověření toho, zda je vzdálená strana tím, za koho se vydává.

Další je šifrování přenášených dat, které zaručí to, že si tato zašifrovaná data dokáže rozšifrovat pouze platný příjemce. Nikoli někdo jiný, kdo by byl schopný se k daným datům dostat.

Poslední vlastností je zaručení integrity posílaných dat. Integritu dat jako takových zaručuje již vrstva TCP, ale touto integritou je myšlena kontrola toho, zda nedošlo na síti k záměrné úpravě dat třetí stranou. Tyto tři klíčové vlastnosti tedy zaručují spojení se s tím, s kým očekáváme, spojení je šifrováno vůči okolí a máme jistotu, že data po cestě nikdo nepozměnil.

Tyto vlastnosti pak implementují jednotlivá softwarová řešení, neboli aplikace. Obecně se nazývají stejným názvem ssh. Jsou to například řešení OpenSSH, F-Secure SSH, nebo SSH1. Ty pak nabízejí služby, jež můžeme použit. Těmi jsou zabezpečené vzdálené terminálové přihlášení a spouštění příkazů, zabezpečený přenos souborů a směrování (tunelování) portů.

Všeobecně je ssh vhodnou náhradou některých nezabezpečených aplikací. Jsou to aplikace rlogin pro vzdálené připojení, rsh pro vzdálené spouštění příkazů a rcp pro přenos souborů. Tyto tři aplikace lze tedy velice jednoduše nahradit pouze jednou aplikací, zvláště když má velice podobnou syntaxi.

Navíc přidává možnost vytvářet takzvané tunely. Přes takovýto tunel je možné tunelovat přenos jiných protokolů. Takovýto přenos se tedy na začátku zašifruje a na konci rozšifruje do původní podoby. Tímto způsobem je tedy možné vzdáleně si zpřístupnit jakékoli nezabezpečené aplikace, například telnet, nebo i grafický protokol x-window. [4]

SSL/TLS

Protokoly SSL a TLS jsou si velice příbuzné a v podstatě TLS vychází z protokolu SSL verze 3. Pro popis možností tedy není nutné mezi nimi rozlišovat.

Jedná se vlastně o dodatečnou vrstvu vloženou mezi TCP/IP a aplikační vrstvu. Při návrhu TCP/IP protokolu nebylo myšleno na šifrování, respektive byla tato vlastnost ponechána na starosti aplikační vrstvě. Bylo by ale zbytečné, aby si toto řešila každá aplikace úplně po svém, proto právě vznikly protokoly SSL/TLS. Tato vrstva nijak nezkoumá obsah od apli-

kační vrstvy, pouze jej převezme a paket po paketu jej zabezpečí. Proto je tuto vrstvu možné využít různými aplikacemi jako je zabezpečený webový protokol HTTPS, což je v podstatě označení pro spojení HTTP a SSL/TLS. Dalšími hlavními představiteli využití této vrstvy jsou poštovní protokoly POP3 a IMAP. Bohužel i dnes se ještě dají nalézt poštovní servery poskytující nezabezpečené verze.

Vrstva SSL/TLS poskytuje zašifrování, kontrolu integrity a autorizaci dat. Při navazování spojení se vždy ověřuje autentizace serveru na základě certifikátu a volitelně umožňuje provést i autentizaci klienta. Integrita je zajištěna pomocí kontrolních součtů, které jsou doplňovány o tajnou část, díky čemuž je složitá záměrná úprava přenášených dat útočníkem. Zajímavostí je, že je pro oba směry komunikace použit jiný symetrický šifrovací klíč a informace pro výpočet tajné části, která je doplňována kontrolnímu součtu. [5]

2.3 Firewall

Firewallem je často nazýván počítač, nebo zařízení, které od sebe oddělují nějaké sítě. To je samozřejmě správný název a firewally byly právě pro tento účel vytvořeny. Oddělení sítí probíhá většinou mezi vnitřní sítí a Internetem, popřípadě i mezi jednotlivými podsítěmi uvnitř. Děje se to z důvodu aby se vytvořil jeden uzel, na kterém je možné provádět kontrolu a filtraci provozu. Pokud by tomu tak nebylo, každý člověk v Internetu by měl přístup ke každému počítači ve všech podsítích. Cílem je tedy většinou propustit přístup z vnitřní sítě do Internetu a v opačném směru ho zakázat.

Pod pojmem firewall se pak skrývá spousta dostupných technologií, které se pro popsané potřeby využívají. Filtrování paketů se začala využívat jako první, když se zjistilo, že směrovače mají dostatek informací o jednotlivých paketech. Samotné filtrování je ale v některých případech nedostatečné a proto se vyvinuly další mechanismy, které se kombinují. Navíc i samotné filtrování prošlo vývojem od bezstavových po stavové. Ty již umí zaznamenávat stav a odstraňují tak největší slabiny bezstavového filtrování.

Jedním z mechanismů, se kterými se filtrování kombinuje je například NAT, který byl již zmíněný jako jedno z řešení nedostatku IP adres. Pomocí NAT totiž na firewallu zajistíme skrytí vnitřní sítě před Internetem. Útočník pak nemá šanci se přímo připojit k počítači ve vnitřní síti. Existují ale metody, jak se přes takovéto zabezpečení dostat, například díky přímému směrování.

Navíc má NAT ještě určité nevýhody a největší z nich je asi nefunkčnost některých protokolů. Jedná se o protokoly, které vyžadují vytvoření zpětného kanálu, například protokoly pro konferenční hovory. Jelikož se NAT stal tak rozšířeným i v domácích sítích, musely se aplikace využívající tyto protokoly naučit NAT také určitým způsobem obcházet.

Stejně tak nefunguje například vzdálené připojení ssh na počítač, který má přeloženou adresu. To se dá vyřešit přesměrováním jednoho portu firewallu na cílovou adresu. Problémem však je, že například domácnosti jsou připojeny přes poskytovatele a ten bohužel přesměrování většinou nepovolí.

Dalším mechanismem jsou takzvané proxy. Ty byly vyvinuty za účelem vyrovnávací paměti pro webové stránky. Ale s růstem počtu stránek začaly být proxy téměř zbytečné. Objevily se u nich ale vlastnosti, které lze využít pro jiné účely. Proxy totiž naslouchá ve vnitřní síti a při přijetí požadavku jej přijme a předá jej na výstupní rozhraní. Takto v podstatě funguje i NAT, ale proxy se odlišuje tím, že se vlastně chová zároveň jako cílové i vnitřní zařízení už na aplikační úrovni. Tím je možné, podobně jako u NAT, skrýt informace o vnitřní síti před Internetem. Navíc, jelikož jsou proxy na aplikační úrovni, mohou filtrovat dle specifických informací daného protokolu. Příkladem může být filtrování URL adres, nebo na základě částí obsahu webových stránek.

Internetem lze směrovat cokoli kamkoli, například propojení dvou podsítí. Ale pokud má být přenos dat bezpečný, nelze toho tradičními způsoby dosáhnout. Jelikož se stále využívá většina nezabezpečených protokolů, tak bude vždy existovat riziko odposlechu někde po cestě paketu. Předchozí mechanismy nám v tom nepomohou a spíše takové spojení u některých protokolů znesnadňují.

Proto byly vymyšleny takzvané VPN, neboli virtuální privátní sítě. Těmi lze bezpečně propojit vzdálené lokální sítě, nebo jen jednotlivé klientské počítače. Využívá se toho, že se veškerý datový provoz nehledě na protokoly zapouzdřuje do zašifrovaných IP paketů. Ty pak putují Internetem a na druhém konci se opět rozšifrují a doručí se do cílové sítě. VPN síť lze ustavovat právě na firewallech, byť to není podmínkou. Nutno podotknout, že se jedná o nástroj pro bezpečný přenos dat a VPN do původních paketů nezasahuje, ani je nefiltruje. Proto se opět kombinují s filtrováním a tak dále.

Firewally tedy mohou obsahovat spoustu různých mechanismů a určitými způsoby je kombinovat, aby bylo využito pozitivních vlastností těchto mechanismů. Sítě navíc prošly a stále procházejí vývojem, proto je nutné sledovat trendy a upravovat zabezpečení. V dnešní

době obsahuje firewall vlastně téměř každý počítač, protože nedílnou součástí operačních systémů se stal paketový filtr. Nejen serverových verzí operačních systémů ale i osobních.

Existují názory, že firewall na osobních počítačích je v dnešní době k ničemu. Tomu nahrává několik faktů. Domácí sítě jsou připojovány přes poskytovatele Internetu, kteří využívají překlad adres NAT z důvodu nedostatku veřejných IP adres verze 4. Díky překladu adres, často spolu s paketovým filtrem, tvoří poskytovatel Internetu domácnosti firewall. Tím odstíní většinu pokusů o útok, které by přišly v případě kdyby domácnosti disponovaly veřejnými adresami.

Další fakt je ten, že i útoky doznaly evoluce. Dnes není problém díky sociálnímu inženýrství „donutit“ uživatele, aby nevědomě poskytl přístup do svého počítače. A tím i do své sítě, v čemž by navíc nepomohl ani paketový filtr. [6]

Někteří lidé mohou také namítnout, že služby zpřístupňované do sítě by měly být zabezpečeny samy o sobě. Měly by tedy ideálně zabezpečit autentizaci, autorizaci a šifrovaný přenos dat. V takovém případě by byl opravdu firewall zbytečný, ale ne zcela. Pokud bychom měli konfigurovat více služeb, byla by to práce navíc, protože ve firewallu bychom tuto konfiguraci provedli jen jednou pro všechny služby.

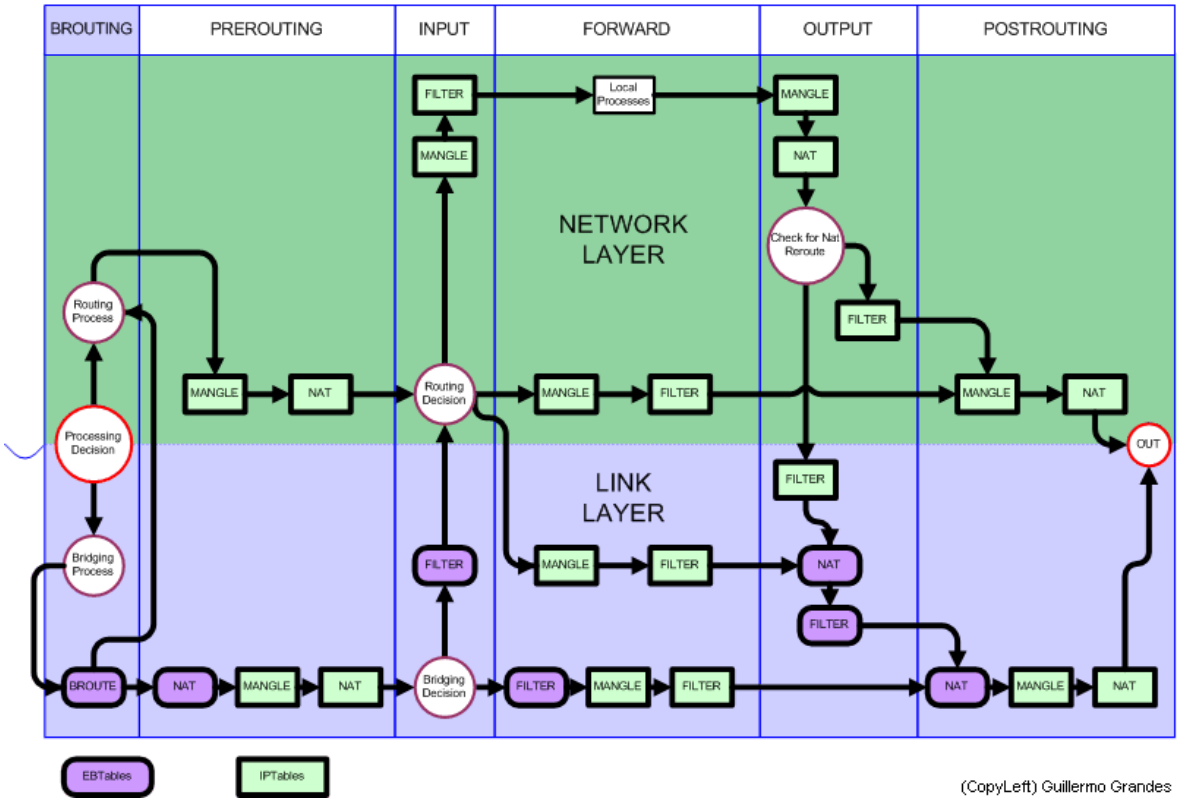
Služby navíc čas od času obsahují chyby v implementaci, které umožňují získání neoprávněného přístupu. Pokud jde o služby veřejně přístupné, tak jsou ve firewallu povoleny a ten nás tedy proti takovéto zranitelnosti neochrání. Maximálně může útočníkovi ztížit další postup. Pokud ale mají být služby dostupné pouze v lokální síti, nebo jen na lokálním počítači, je dle mého názoru firewall stále nejvhodnějším řešením zabezpečení. Navíc i v síti poskytovatele jsme připojeni k ostatním klientům v jedné síti a před nimi nás jeho firewall nechrání.

Firewall v Linuxu

Firewall lze v operačních systémech s linuxovým jádrem vytvořit poměrně snadno. Podpora paketového filtru je v linuxovém jádře zahrnuta již od verze 2, tedy zhruba od roku 1990. Od té doby samozřejmě vývoj obrovsky pokročil dopředu. Všechny verze ale mají již od počátku společný princip. Paketový filtr je reprezentován několika řetězci, anglicky chains, ve kterých se nachází pravidla. Paket pak postupně prochází tato pravidla, kterými může být povolen, nebo zakázán. Pokud paket projde až nakonec, aplikuje se na něj politika řetězce, která může být povolení, nebo zakázání. Samozřejmě zde také dochází ke směrování, kdy se zjišťuje, zda paket míří na daný stroj, či jen prochází.

V první verzi paketového filtru s názvem ipfwadm byly uživateli zpřístupněny pouze tři řetězce: in, forward a out. V další verzi, Linuxu 2.2, byl zaveden nový paketový filtr ip_fwchains ovládaný nástrojem ipchains. Ten obsahoval navíc možnost vytvářet vlastní řetězce, které mohly být i vnořené. Tři standardní řetězce zůstaly, jen byly přejmenovány na input, forward a output. [7]

Od verze Linuxu 2.4 až dodnes se v jádře nachází plně přepsaná a inovovaná verze paketového filtru. Skládá se z jednotlivých modulů jádra, které jsou součástí celku nazývaného netfilter. Ten se ovládá pomocí nástroje iptables. Většina lidí pak nazývá celý paketový filtr po tomto nástroji, nesprávně, iptables. Ač byl tento projekt celý přepsán, podstata zůstává stále stejná. Bylo ale přidáno několik nových standardních řetězců, takže je tok paketů daleko složitější. Jejich vypsání by bylo zbytečné, protože daleko lepší představu si lze udělat z následujícího obrázku.



Ilustrace 1: Netfilter - tok paketů [8]

Na diagramu je navíc modul modul ebtables, který slouží k filtrování v rámci síťových mostů. V typické instalaci bychom ho nenalezli a pakety by vstupovaly do řetězce PREROUTING. Z diagramu je patrné, že řetězce jsou ještě rozčleněny do jednotlivých tabulek: NAT,

MANGLE a FILTER. V řetězcích některých tabulek, lze totiž využívat specifické možnosti. Například v řetězcích tabulky MANGLE lze využívat modifikátory pro TTL, TOS, nebo MARK. Úpravou TTL na směrovači můžeme například skrýt přítomnost dalších směrovačů ve vnitřní síti, které TTL při každém průchodu snižují.

Důležitou vlastností oproti původním verzím je, že procházející pakety vůbec nevstoupí do řetězců INPUT a OUTPUT. Projde tedy jen řetězci PREROUTING, FORWARD a POSTROUTING.

Pomocí nástroje iptables jsme pak schopni spravovat veškerá pravidla v řetězcích. Není ale cílem vytvářet detailní manuál, který by zabral celou tuto práci. Návod k použití lze nalézt v manuálových stránkách přímo v linuxových systémech, případně jsou k dispozici online. Například na stránkách http://manpages.debian.net/cgi-bin/man.cgi?query=iptables.

Jak již bylo řečeno, firewall není jen o filtrování paketů. Linuxové jádro samo o sobě ale obsahuje jen zmíněný netfilter, možnost překladu adres a samozřejmě směrování. Pokud bychom ale potřebovali využít z nějakého důvodu proxy, nebo VPN, musíme sáhnout po dodatečném softwaru. Ten již není součástí jádra a běží většinou jako síťový démon, tedy v uživatelském režimu.

Na výběr jsou desítky různých možností, komerčních i volně dostupných. Proxy a VPN jsou důležité pro specifické použití, proto se zaměřuji jen na možnosti paketových filtrů. Pokud tedy dále v textu zmíním pojem firewall, budu tím myslet pouze možnosti linuxového jádra. Stejně tak aplikace pro centrální správu bude obsahovat pouze správu paketového filtru.

3 Požadavky na centrální správu

Hlavním tématem celé této práce je centrální správa linuxového firewallu. Tedy stanic s nespecifikovanou linuxovou distribucí. V této kapitole určím mé požadavky, jaké by měl tento systém splňovat a také provedu průzkum existujících řešení s jejich analýzou.

Požadavky

První požadavek je obsažen již v názvu, systém musí být centrální. Mělo by k němu mít přístup více uživatel (správců) pod svými přístupovými údaji.

Celý systém musí být co nejbezpečnější, zvláště když ho bude možné ovládat vzdáleně po nezabezpečené síti. Tím je myšlena nejen aplikace jako taková, ale i přenos dat mezi aplikací a administrátorem, stejně jako mezi aplikací a cílovými stanicemi. Na těch bude ovládán firewall, což vyžaduje administrátorská oprávnění. To musí být provedeno s rozvahou, aby nebyla vytvořena bezpečnostní díra ve všech ovládaných stanicích.

Mělo by být možné vytvářet skupiny spravovaných firewallů, například pro potřeby učeben. Těmto skupinám by mělo jít nastavovat společná pravidla.

Jeho používání by mělo být jednoduché a intuitivní, ale zároveň by nemělo zkušenému administrátorovi přidělávat potíže. Administrátor by neměl být díky tomuto systému nucen učit se novou syntaxi, a pokud systém novou syntaxi zavádí, měl by alespoň umožňovat klasickou správu.

3.1 Existující řešení

Vyhledáváním na internetu a v nabídce softwaru pro systém Ubuntu jsem nalezl následující možná řešení pro centrální správu. Musel jsem projít jejich dokumentaci a většinou je i nainstalovat a vyzkoušet, zda by vyhovovaly požadavkům. Jelikož se jedná o řešení s otevřeným zdrojovým kódem, nabízela by se i možnost nějaký z těchto systémů upravit.

Terminál a SSH

Provádění správy firewallu vzdáleně z terminálu pomocí SSH by bylo možné a efektivní pro malé množství spravovaných firewallů. Pokud by přihlašování probíhalo pomocí klíče a dané systémy by měly doménová jména, dalo by se toto provádět velice jednoduše a byla by zde i možnost přípravy hotových skriptů, které by se pouze aplikovaly. Z podstaty protokolu

SSH by tento způsob byl i velice bezpečný. Pro větší množství stanic by to ale byla varianta nepraktická a nevyhovující požadavku.

Firestarter a Ufw

Jsou minimalistické grafické aplikace pro správu linuxového firewallu. Jejich ovládání je velice intuitivní a jednoduché. Obě slouží ale jen pro správu jednoho lokálního stroje, ani jeden tedy nevyhovuje hlavnímu požadavku centrální správy.

Fwbuilder

Oproti tomu Fwbuilder je komplexní nástroj pro centrální správu firewallu. Podporuje kromě iptables také jiné platformy, jako je například Cisco, OpenBSD a FreeBSD, MacOS X a různé systémy WRT. Tím centralizuje nastavení všech možných systémů v síti pod jedno rozhraní.

Veškeré nastavení zde probíhá graficky pomocí takzvaných objektů, existují předpřipravené a uživatelské. Objektem je zde vše, IP adresa, jejich rozsah, port atd. Pravidla firewallu se pak skládají z objektů metodou drag-and-drop. Výsledek se pak překládá do formy pro danou platformu a nahrává se na cílové zařízení. To se děje pomocí bezpečných protokolů SSH a SCP.

Celkově je tato aplikace velice dobře použitelná a bezpečná. Mínusem je složitější (komplexnější) ovládání, které je odlišné od běžné správy iptables. Administrace více uživateli je možná tak, že každý bude mít vlastní instanci této aplikace a k disposici sdílený konfigurační soubor, který musí být patřičně zabezpečen.

Shorewall, APF a Pyroman

Tyto tři (a další podobné) aplikace slouží také pro administraci netfilterových firewallů. Jsou to ale pouze konzolové nadstavby nad iptables. Nesplňují tedy většinu požadavků. Centralizovaná vzdálená správa pomocí těchto aplikací by se musela taktéž odehrávat s pomocí protokolu SSH.

4 Vlastní řešení

Ani jedno z existujících řešení nevyhovuje požadavkům. Možností by byla úprava některého z existujících řešení, ale ani jedno z nich nemá potenciál pro jednoduchou úpravu. Mnohem jednodušší bude vytvořit nové řešení plně podle potřeb a požadavků.

4.1 Výběr řešení

Aplikaci tohoto rozsahu a těchto požadavků by bylo možno naprogramovat ve spoustě různých programovacích jazyků a pomocí různých technologií. Nejdříve je tedy třeba zhodnotit pro a proti jednotlivých řešení a nějaké vybrat.

Databáze

Aplikace musí v každém případě ukládat pravidla firewallů jednotlivých stanic. Bylo by možné je ukládat do textového nebo binárního souboru s vlastním formátem. Mimo pravidel by bylo nejspíše vhodné ukládat i další informace o stanicích a podobně. Navíc by u tohoto typu ukládání mohl časem vzniknout problém s rychlostí zpracování. Proto padl výběr na několik specializovaných databázových řešení.

Nejběžnějšími databázovými systémy jsou Oracle Database, Microsoft MS-SQL, PostgreSQL a MySQL. První dva jsou komerční a pro jejich používání je nutná určitá licence. Oproti tomu poslední dva systémy jsou dostupné pod nějakou volnou licencí (MIT, GPL, …). Vzhledem k tomu, že tato aplikace míří na školní učebny, menší domácí sítě apod., byla by licence za databázový systém neopodstatněná. Zvláště když tato aplikace bude z databázového systému využívat pouze základní vlastnosti, kterými disponují všechny tyto systémy.

Jelikož nebudou požadovány žádné složitější operace s databází, byl zvolen databázový systém MySQL. Ten je možné provozovat s různými „jádry“ a na výběr jsou nejznámější MyISAM a InnoDB. Použil jsem druhý jmenovaný, protože on jediný pro databázi MySQL umí cizí klíče a transakce, což jsou samozřejmě důležité vlastnosti pro bezpečné použití databáze.

Architektura

V dnešní době existuje mnoho rozličných mobilních zařízení různých architektur (ARM a podobně). Vytvoření aplikace pro centrální správu na každou takovouto architekturu by

vyžadovalo mnoho práce navíc a spoustu duplicitního kódu v různých programovacích jazycích.

Jelikož ale aplikace nebude primárně sloužit ke každodenní mobilní správě, bude daleko jednodušší řešení vytvořit aplikaci jako webovou. K té pak bude možné přistoupit pomocí jakéhokoli zařízení podporujícího webový prohlížeč se základními webovými standardy. Zabezpečení takovéhoto spojení bude zajištěno pomocí zabezpečené vrstvy SSL. Ta využívá certifikátů a šifruje celou komunikaci.

Dalším problémem je komunikace mezi serverem a cílovými stanicemi. Na cílových stanicích musí být zajištěno ovládání firewallu. Na těch běží operační systémy s linuxovým jádrem, které obsahuje netfilter moduly pro filtrování síťového provozu. Celkové řízení se provádí pomocí programu iptables, který pracuje v příkazové řádce. V zásadě tedy existují dvě možnosti vzdáleného řízení.

První z nich je vytvoření aplikace, která by naslouchala na určitém portu a centrální server by jí posílal příkazy. Ty by pak interpretovala do podoby příkazů iptables. Nevýhodou by byla nutnost na každou stanici instalovat další aplikaci. Tato aplikace by navíc musela do detailu řešit své vlastní zabezpečení (autorizaci i autentizaci) a nejlépe i šifrování přenosu dat mezi řídícím serverem.

Druhou možností je využití existujícího řešení pro vzdálené připojení a správu. V dnešní době se k tomuto využívá protokol a stejnojmenná aplikace SSH, neboli secure shell (bezpečná příkazová řádka). Princip tohoto protokolu je detailněji popsán v kapitole číslo 2.1 Zabezpečení protokolů.

Nejběžnější implementací v linuxových systémech je aplikace OpenSSH, která je šířena pod open-source licencí GNU GPL. Použití této aplikace vyřeší zabezpečení přenosu dat po nedůvěryhodné síti, protože její implicitní funkcí je šifrování přenosu. Obsahuje také ověřování uživatele, takže nedovolí neoprávněný přístup. Mimo autentizaci pomocí uživatelského jména a hesla poskytuje i autentizaci pomocí klíčů. Po úspěšném přihlášení nám pak zpřístupní vzdálenou příkazovou řádku. Na té už je možné provádět jakékoli řádkové příkazy či spouštět skripty nebo posílat a přijímat soubory.

Problémem je, že je při připojování vyžadováno heslo. Je zde několik možností jak toto řešit. V případě malého počtu stanic si administrátor tato hesla bude pamatovat, proto by byla možnost při každém spojení zadávat heslo v nějakém vyskakovacím okně. Případně by mohl

u všech stanic vytvořit účty se stejným jménem a heslem. To jsou ale řešení, která nevyhovují jak bezpečností, tak možností dalšího rozšíření.

Další možností je ukládat přístupové jméno a heslo do databáze pro každou vzdálenou stanici. Tato hesla by ale bylo nutné předávat externím aplikacím, tedy v čitelné textové formě. Pokud bychom je takto ukládali do databáze, bylo by to obrovské riziko. Jednosměrnou šifrou je zašifrovat nemůžeme, protože bychom zpět textovou formu nezískali.

Hesla by šlo zašifrovat obousměrnou šifrou za použití jednoho nějakého zvoleného hesla. To by muselo být ale opět zadáváno z klávesnice, protože uložení hesla je vždy bezpečnostní riziko. Asi poslední možností je použití přihlašování pomocí veřejných klíčů. Ty mohou být taktéž ještě navíc zašifrovány, ale není to povinné a vyvstaly by stejné problémy s uchováním těchto hesel. Proto využiji klíče bez zadaného hesla. Tím se bezpečnost v podstatě přesune na úroveň souborového systému a zabezpečení soukromého klíče, který nesmí být čitelný pro nikoho jiného.

Programovací jazyk a webový server

Aplikace musí běžet na nějakém centrálním serveru, ať už v nějaké lokální (firemní, školní, domácí, …) síti nebo přímo v síti Internet. Serverová aplikace musí být zvolena dle technologie, kterou je vlastní aplikace tvořena. V případě programovacího jazyka JAVA a přidružených webových technologií by jako server sloužila některá podporovaná implementace webového serveru (Apache Tomcat a podobně). V tomto případě je na výběr velké množství implementací a některé jsou opět licencované. Tyto servery ale dokáží být náročné na operační paměť a další zdroje. Také jejich konfigurace bývá náročnější.

Protože tato aplikace bude směřována nejen na rozsáhlejší školní sítě, ale také na menší domácí sítě, vybral jsem jednodušší variantu. Tou je skriptovací jazyk PHP (což je zkratka pro PHP: hypertext preprocesor). Tento jazyk samotný je pro bezpečný návrh aplikací z mého hlediska vysloveně nedostatečný. Podporuje sice již v aktuálních verzích všechny prostředky pro kvalitní OOP (objektové orientované programování), ale má několik zásadních nevýhod, které programátora nevedou k čistému návrhu aplikace.

Z nich například dynamická typová kontrola (datové typy nejsou nikde vyžadovány a při operacích s nimi se provádí dynamická konverze pokud je potřeba). Není kontrolován počet parametrů funkcí, některé chyby při provádění kódu nejsou reportovány programátorovi

vůbec, jiné špatně. Celkově je výpis chyb špatně čitelný. PHP skripty je možné psát rovnou do HTML kódu, což velice znepřehlední celou aplikaci.

Přes to všechno je to pro menší aplikace neocenitelný jazyk, jehož skripty je možné velice rychle nasadit pomocí serverové aplikace Apache s modulem pro PHP jazyk. Jeho konfigurace je pro základní potřeby také poměrně jednoduchá. Pro komunikaci s cílovými hosty pomocí vybraného protokolu SSH má jazyk PHP přímo implementovány třídy. Takže je práce s tímto protokolem velice zjednodušena.

Framework

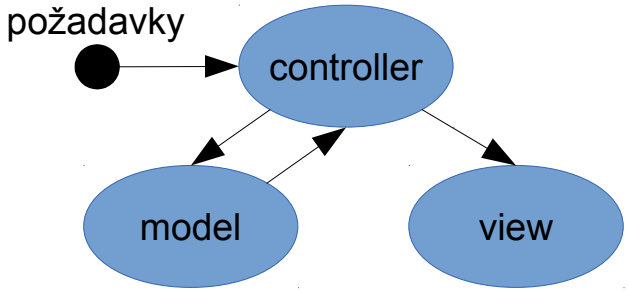
Nedostatky jazyka jako takového a spoustu dalších věcí je možné zdokonalit pomocí sady nástrojů, které se obecně říká framework. K dispozici je jich opravdu veliká spousta, namátkou například projekty Zend, Symfony a CakePHP. Více jich není třeba vypisovat, existují jich stovky a nejlepší se najít nedá. Každý má své pro a proti, bohužel se kolikrát na slabiny přijde až v průběhu, proto je lepší vybírat z prověřených projektů. Po prozkoumání dokumentací jsem se rozhodl použít český Nette framework [9], se kterým jsem již měl drobné zkušenosti.

Tento framework se nesnaží striktně opravovat nedostatky jazyka PHP, ale dává k dispozici nástroje, které jeho používání radikálně usnadní. Například základní třída Object, od které dědí veškeré třídy frameworku a my ji můžeme také využít. Pokud ji využijeme, tak nás framework upozorní na některé, již výše popsané, chyby. Mezi nimi je například i přístup k nedeklarované proměnné apod.

Jednou z nejzásadnějších věcí, která šetří čas a pomáhá, je takzvaná „Laděnka“. Je to ladicí nástroj, který v celé aplikaci odchytává veškeré chyby a oproti holému PHP je vypisuje ve srozumitelné formě. Navíc obsahuje zpětné trasování jednotlivými skripty až tam, kde se vyskytla chyba. To vše s výpisem částí kódů těchto skriptů.

Další věcí je, že programátora vede k čistšímu návrhu celé aplikace, ale není to nutností. Vede nás totiž k návrhu aplikace do vrstev MVC, což je zkratka slov model, view (pohled) a controller (řadič). Vrstvou modelu v architektuře MVC je myšlena datová vrstva aplikace, nikoli jediná třída modelu. Uchovává stav a navenek poskytuje rozhraní ke změnám tohoto stavu. Neví o existenci ostatních vrstev a naopak o existenci modelové vrstvy by měl vědět pouze controller.

Controller, v případě frameworku Nette reprezentovaný Presentery, reaguje na uživatelské vstupy a upravuje dle nich pomocí rozhraní stav modelu. Tento stav je pak většinou nutné zpět reprezentovat uživateli, což nedělá přímo controller, ale žádá o toto vrstvu view, zde reprezentovanou šablonami. Těm proto předá potřebná data a šablony je vykreslí.



Ilustrace 2: MVC architektura

Tato architektura je výhodná z několika hledisek. Jelikož logicky odděluje spolu související části aplikace, je výsledná aplikace daleko přehlednější. Tím pádem je také většinou méně chybová a bezpečnější. Přináší to také možnost zasahovat do logiky modelu, aniž bychom ovlivnili zbytek, nebo naopak můžeme změnit reprezentaci stavu pouhou úpravou šablony. Pro šablony má navíc tento framework vlastní šablonovací systém Latte, který se snaží syntaktický zápis vykreslovaných dat zpřehlednit a zjednodušit do nejvyšší možné míry.

Databázová vrstva

Databázovou vrstvou je myšlena vrstva, pomocí které komunikujeme se samotným databázovým serverem. Po výběru databáze MySQL, skriptovacího jazyka PHP, frameworku Nette máme na výběr velké množství dostupných produktů. Liší se jak složitostí, tak způsobem použití. Pokud bychom zvolili pro danou aplikaci nevhodnou databázovou vrstvu, byli bychom při pozdější nápravě nuceni přepsat veškeré modely. Proto je nutné podrobnější prozkoumání možností.

První možností je využít nativních knihovních funkcí PHP s prefixem „mysql_“. Jde především o funkce *mysql_connect()*, *mysql_query()*, *mysql_fetch_assoc()* a *mysql_close()*. Práce s touto knihovnou je velice nízkoúrovňová a pro čistý objektový návrh by vyžadovala spoustu programování navíc oproti dalším řešením. Navíc jsou tyto funkce již označeny jako zastaralé a nebudou v dalších verzích podporovány (cyklus použití nových verzí PHP je ale poměrně pomalý).

Nástupcem těchto jednoduchých funkcí je opět nativní knihovna se jménem PDO, což je zkratka pro PHP Data Objects. To značí, že už je to pokročilejší knihovna s objektovým návrhem. Je navíc abstraktní, takže obsahuje takzvané ovladače pro různé databázové servery, pro které poskytuje stejné funkce. Tím je pak možný případný přechod na jinou databázi. S knihovnou PDO je jednoduché využít například transakce, nebo takzvané prepared statements. To je funkcionalita, která velice usnadňuje bezpečné využití databáze. Transakce nám zajistí integritu dat při práci s více záznamy a prepared statements mimo jiné ošetří proměnné proti útoku SQL injekce. PDO knihovnu by již šlo velice dobře použít, ale existují ještě pokročilejší knihovny, které poskytují další cenné vlastnosti. Mnoho z nich staví právě na této knihovně PDO.

Za zmínku stojí český projekt Dibi dostupný na adrese http://dibiphp.com. Jedná se o velmi malou knihovnu, která velice usnadňuje zápis SQL dotazu tím, že zavádí jednoduchou syntaxi pro zápis parametrů a podobně. Výsledky dotazů dokáže vracet v požadované struktuře vnořených polí, což usnadňuje následné zpracování. Celkově je to výborná knihovna, kterou by také bylo možné použít, ale její použití se spíše blíží první možnosti.

K dispozici máme také vrstvu Nette Database, která staví nad PDO. Ta je, jak již název napovídá, součástí celého Nette frameworku, proto se nejsnáze do aplikace integruje. Její použití je poměrně odlišné. Máme sice možnost psát celé dotazy SQL, jak jsme zvyklí, ale tato knihovna přidává možnost takzvaného „fluent“ zápisu. To znamená, že místo jednoho dlouhého dotazu SQL voláme postupně funkce pojmenované stejně jako jsou jejich ekvivalenty v SQL syntaxi.

Výsledný objekt si ve výsledku uvnitř sebe daný dotaz složí a provede ho. Tímto se stává zápis dotazu a hlavně parametrů daleko rychlejší, jelikož nám vývojové prostředí dokáže napovídat a doplňovat jména funkcí. Výsledkem, který se nám dostane je pak jednotlivý řádek, nebo kolekce řádků. Výhodou skládání dotazu je také to, že si průběžný dotaz (objekt) můžeme posílat například mezi funkcemi, nebo objekty a podle potřeby mu přidávat další části dotazu. Parametry se samozřejmě vždy automaticky ošetřují na SQL injekci. To vše (a ještě více) usnadňuje a zrychluje vývoj aplikace.

Vybírat můžeme ale i ze skupiny databázových vrstev kterým se říká ORM (Object Relational Mapping). To už je vyšší stupeň, který nad nějakou abstraktní databázovou vytváří další vrstvu. Ta se stará o převod z relační databáze do objektů a naopak. V předchozích situacích

nám byl vždy vrácen řádek z databáze, případně kolekce řádků. Byť to většinou byly objekty, nebylo nijak zajištěno mapování na databázové údaje.

Jména tabulek a sloupců bylo nutné znát a psát ručně. V případě ORM je určitými metodami docíleno mapování na skutečné tabulky. To se často děje tak, že nám daný ORM framework vygeneruje na základě reálné databáze třídy s potřebnými metodami a vlastnostmi. Na programátorovi pak je, aby si doprogramoval dodatečnou funkčnost. Tyto třídy jsou vlastně modely a krásně zapadají do architektury MVC. Je už ale na daném ORM, jakým a jak přívětivým způsobem se pracuje s těmito třídami a jak tyto třídy samotné vypadají. Nejznámějšími zástupci ORM frameworků pro jazyk PHP jsou Doctrine, Propel a Symfony.

Z povahy tohoto projektu, který není tak databázově orientovaný a rozsáhlý, jsem usoudil, že by stačilo použít některé z prvně zmiňovaných řešení (PDO, dibi, Nette Database). Jakýkoli ORM framefork by byl pro tento projekt nejspíše zbytečně složitý. Je všeobecně známo, že výhody použití této složitější vrstvy stoupají s rozsáhlostí databázového modelu. Také doba naučení se některému z nabízených ORM řešení by byla vyšší než u předchozích.

Někde mezi ORM a holým PDO, nebo dibi, leží projekt Nette Database. Jeho výhody převyšují ostatní dvě řešení a jelikož je přímo součástí frameworku Nette, zvolil jsem Nette Database jako databázovou vrstvu kterou použiji v projektu.

4.2 Instalace a konfigurace prostředí

Jako server byl použit operační systém Ubuntu Desktop verze 12.04 LTS, proto bude instalace dodatečných aplikací probíhat pomocí balíčkovacího systému APT převzatého ze systému Debian. K instalaci je možné použít několik různých nástrojů, například dpkg, apt-get, aptitude, nebo některý z grafických nástrojů. Já jsem použil nástroj aptitude a proto budou uvedené instalační příkazy pomocí něj. Ostatní uvedené nástroje mají ale podobnou syntaxi, proto není složité aptitude nahradit za jiný.

Jednotlivé aplikace by mohly být získány jako zdrojové kódy nebo zkompilované binární soubory z webových stránek jednotlivých autorů, ale balíčkovací systém skýtá výhody jednoduché instalace, pravidelných aktualizací a odladěných verzí softwaru. Nevýhodou je naopak to, že se aplikace v základních repositářích Ubuntu nemusí nacházet v nejnovější verzi. To ale většinou není na škodu.

Databáze

Databázový systém MySQL nainstalujeme jednoduše tímto příkazem:

```
sudo aptitude install mysql-server mysql-common mysql-client
```

Při instalaci je pouze nutné zadat heslo hlavního administrátora databázového serveru (root), které bude potřeba pro další operace. Pokud je potřeba, upravíme konfiguraci v souboru /etc/mysql/my.cnf, kde se nachází veškeré nastavení. Základní nastavení je ale pro tento příklad dostačující, proto netřeba měnit.

Sice by bylo možné pracovat s databází pod uživatelem root, ale to je velice nedoporučované a nebezpečné řešení. Proto je vhodné vytvořit pro tuto aplikaci vlastní přístup s vlastním heslem. To je možné provést pomocí nějakých externích nástrojů jako je například Adminer, PhpMyAdmin a další. Nebo je možné použít přímo základního mysql klenta. Použijeme tedy příkaz z příkazové řádky a po dotazu na heslo zadáme heslo, které jsme nastavili při instalaci.

```
mysql -u root -p
```

Nyní už máme k dispozici veškeré příkazy SQL. Nového uživatele jménem 'diplomova_prace', který se bude moci připojit z počítače jménem localhost a se zadaným heslem vytvoříme následujícím příkazem. Jméno uživatele je možné zvolit samozřejmě jakékoli, zadané údaje pak stačí nastavit v konfiguračním souboru aplikace.

```
CREATE USER 'diplomova_prace'@'localhost' IDENTIFIED BY PASSWORD
'*EE617D63119CF212601409D6A76F67D8230E4DC9';
```

Heslo je možné zadávat jako čistý text s tím, že si ho mysql interně zašifruje. Existuje tu ale riziko toho, že se tento příkaz jako takový i s tímto heslem objeví v záznamech. Proto je doporučováno po tomto příkazu promazat záznamy, pokud je to možné. Nebo, jako v tomto příkazu, neuvádět heslo v textové formě, ale rovnou zašifrováno. [10]

Webový server

Zvolený webový server Apache sám o sobě nezajišťuje podporu jazyka PHP. Jednotlivé vlastnosti se mu dodávají pomocí modulů. Je tedy potřeba nainstalovat následující části:

- apache2 – meta-balíček, který automaticky zajistí základní komponenty potřebné pro běh webového serveru abychom nemuseli ručně instalovat desítky balíčků,
- php5 a php5-fpm – balíčky se samotnou implementací jazyka PHP,
- php5-mysql – podpora připojení k databázovému serveru MySQL z jazyka PHP,

- libapache2-mod-php5 – podpora jazyka PHP ve webovém serveru Apache,
- libssh2-php – podpora protokolu SSH pro jazyk PHP.

```
sudo aptitude install apache2 php5 php5-fpm php5-mysql
libapache2-mod-php5 php5-mysql libssh2-php
```

Konfigurace webového serveru Apache a PHP se nachází v adresářích /etc/apache2 a /etc/php5/apache2, ale pro základní použití není třeba nic měnit. Je potřeba pouze zkontrolovat, zda je v souboru /etc/apache2/ports.conf nastaven jak port 80, tak zabezpečený port 443. Pokud ne, je potřeba provést zhruba takovouto konfiguraci:

```
NameVirtualHost *:80
Listen 80
<IfModule mod_ssl.c>
    NameVirtualHost *:443
    Listen 443
</IfModule>
```

Dále je potřebné nainstalovat podporu bezpečného šifrování přenosu ke klientovi. Zde je na výběr řešení OpenSSL, nebo GNUTLS. Druhé řešení se mi dle dokumentace nepovedlo zprovoznit spolu s webovým serverem Apache v rozumném čase. Oproti tomu řešení OpenSSL fungovalo téměř okamžitě. K jeho použití je nejdříve potřeba nainstalovat samotný balík openssl. Jeho podpora je již v základní instalaci Apache, oproti GNUTLS, kde se musí ještě instalovat modul libapache2-mod-gnutls.

```
sudo aptitude install openssl
```

Webový server je nyní nainstalovaný, ale zatím pouze ve výchozím stavu. Nyní je nutné jej nakonfigurovat. Jednak je potřeba povolit dodatečné moduly, které standardně povolené nejsou. Jedná se o moduly rewrite, php5 pro podporu jazyka PHP a ssl pro podporu šifrování pomocí knihoven OpenSSL. Moduly se zapínají a vypínají pomocí aplikace a2enmod, respektive a2dismod. Po zapnutí modulů je nutné restartovat běžící server, případně ho analogickým způsobem nastartovat.

```
sudo a2enmod rewrite php5 ssl
sudo service apache2 restart
```

Webový server Apache dokáže v jedné své instanci poskytovat více než jedno umístění. Takže je například možné poskytovat více různých webových služeb, fyzicky umístěnými na jednom serveru, pod různými adresami, nebo doménami. Je tedy nějakým způsobem nutné definovat vztahy mezi adresáři ve kterých sídlí poskytované aplikace a adresami, které na

dané aplikace směřují. Základním nastavením je adresář */var/www* (případně */var/htdocs*), který je dostupný na všech adresách adresách dostupných na daném serveru.

Předem zmíněné vztahy se v terminologii apache nazývají virtual hosts. Jejich nastavení sídlí (v systémech Ubuntu a dalších) v adresářích */etc/apache2/sites-available* a */etc/apache2/sites-enabled*. V prvním z nich jsou dané konfigurace a do druhého adresáře se vytváří symbolické odkazy pro ty konfigurace, jež chceme aktuálně jako aktivní.

Aktivování a deaktivování se provádí příkazem a2ensite, respektive a2dissite. Po změně konfigurace není třeba server restartovat (což by odpojilo aktuálně připojené klienty), ale postačí provést takzvaný reload nastavení příkazem *sudo service apache2 reload.*

Nastavení virtuálních hostů se bude lišit podle toho, kde bude aplikace provozována. Navíc je detailní vysvětlování jednotlivých prvků konfigurace mimo téma této práce, proto jen stručně na funkčním příkladu:

```
1   <IfModule mod_ssl.c>
2   <VirtualHost *:443>
3        ServerAdmin webmaster@localhost
4        ServerName domena.cz
5        DocumentRoot /var/www/aplikace
6        <Directory />
7              Options FollowSymLinks Indexes
8              AllowOverride None
9              Order allow,deny
10             Allow from all
11       </Directory>
12       <Directory /var/www/aplikace >
13             Options Indexes FollowSymLinks MultiViews
14             AllowOverride all
15             Order allow,deny
16             allow from all
17       </Directory>
18
19       ErrorLog ${APACHE_LOG_DIR}/aplikace_error.log
20       LogLevel warn
21       CustomLog ${APACHE_LOG_DIR}/aplikace_ssl_access.log combined
22
23       SSLEngine on
24       SSLCertificateFile    /etc/apache2/ssl/ssl/domena.cz.crt
25       SSLCertificateKeyFile /etc/apache2/ssl/ssl/domena.cz.key
```

```
26  </VirtualHost>
27  </IfModule>
```

První řádek pouze ověřuje, zda je zaveden potřebný modul pro ssl. Druhý řádek je už daleko důležitější. Říká, že tento host bude dostupný pod všemi adresami na portu 443, což je standardní port pro zabezpečenou webovou komunikaci protokolem HTTPs.

Dále je zde nastavení názvu serveru, emailu administrátora a také kořenový adresář, kde budou vyhledávány jednotlivé skripty aplikace. Všem adresářům je pak ještě možné specifikovat přístupová práva a další možnosti. Následuje nastavení logování chyb do samostatných souborů, kvůli odlišní od ostatních virtuálních hostů.

Posledními položkami je nastavení šifrování pomocí SSL. To se dá nastavit daleko podrobněji, ale pro základ postačí uvedené nastavení certifikátu a klíče. Podrobnější konfiguraci lze nalézt v manuálových stránkách, odkud bylo čerpáno.

Certifikát OpenSSL

Výše zmíněný certifikát a klíč je nutné ale také někde získat. Pokud je již nemáme, musíme si je vygenerovat. [11] Nejprve je potřeba vygenerovat klíč – 2 048 bitů dlouhý klíč vygenerujeme například takovýmto příkazem:

```
openssl genrsa -out domena.cz.key 2048
```

Výsledkem bude soubor *domena.cz.key*, kde za doménu si zvolíme takovou, na které aplikace poběží. Dále musíme vytvořit žádost o certifikát pomocí tohoto klíče:

```
openssl req -new -key domena.cz.key -out domena.cz.csr
```

Během tohoto procesu vyplníme žádané informace o doméně, jejím umístění a podobně. Výsledný soubor *domena.cz.csr* bychom nyní zaslali nějaké důvěryhodné certifikační autoritě, která by nám zaslala na oplátku požadovaný certifikát. Za tuto službu se ale platí[6], proto jsem si pro testování vydal certifikát sám s platností jeden rok následujícím způsobem.

```
openssl req -x509 -key www.domena.cz.key -in www.domena.cz.csr -out
www.domena.cz.crt -days 365
```

Takovýto certifikát bude plně funkční, jen jej nepoznají prohlížeče, proto zahlásí, že certifikát není podepsán žádnou známou a důvěryhodnou certifikační autoritou. Certifikát lze do prohlížeče nainstalovat. Hrozí zde ale riziko, že pokud se budeme přihlašovat z prohlížeče, který nemá tento certifikát nainstalovaný, nemáme jistotu, že nám certifikát někdo po cestě

6 Až dodatečně, při psaní práce, jsem zjistil že existuje autorita poskytující základní certifikát zdarma, viz http://www.startssl.com/

nepodvrhl. Proto je toto řešení vhodné pouze na testování a na produkční server patří vždy certifikát podepsaný důvěryhodnou autoritou.

OpenSSH

Aplikace bude tedy ovládat firewally vzdálených stanic pomocí protokolu SSH a jeho implementace OpenSSH. Ten podporuje přihlašování pomocí hesel i klíčů a přihlašuje se samozřejmě pod nějakým uživatelským jménem. Centrální server bude v ssh spojení hrát roli klienta a proto je na něj potřeba nainstalovat balíček OpenSSH klienta. Vzdálené stanice budou naopak v roli ssh serveru, jelikož se k nim budeme připojovat, proto potřebují balíček OpenSSH serveru. K tomu musíme navíc vytvořit uživatelský účet, pod kterým se bude centrální server připojovat.

Instalace OpenSSH serveru, jeho konfigurace a vytvoření uživatelského účtu jsou v podstatě jediné operace, které se musí provést na straně všech klientů. Navíc tyto věci již bývají standardně nastavené, aby byly stanice jednodušeji spravovatelné. Pokud nejsou, provedeme je následovně:

```
sudo adduser centralni_sprava
sudo aptitude install openssh-client
sudo aptitude install openssh-server
```

Konfigurace OpenSSH serveru se nachází v souboru */etc/ssh/sshd_config* a je zde několik zásadních voleb, které se týkají především zabezpečení. Především je zde adresa a port na kterých server naslouchá. Standardně je to port 22 a všechny dostupné adresy. V tomto případě se ale předpokládá, že tyto vzdálené stanice budou ležet ve vnitřní síti a bude k nim přistupovat řídící centrální server ve stejné síti. Proto není nutné ssh server chránit takovým způsobem jako bychom to udělali při připojení takového serveru do sítě internet. Samozřejmě je třeba nepodcenit rizika a chránit i služby ve vnitřní síti, zvláště u vzdáleného připojení přes ssh. Proto jsou tu další volby, které by měly být nastavené pokud možno co nejstriktněji. [4]

```
Protocol 2
PermitRootLogin no
StrictModes yes
RSAAuthentication yes
PubkeyAuthentication yes
PasswordAuthentication yes
PermitEmptyPasswords no
```

Tím zajistíme vynucení protokolu verze 2, který je již delší dobu považovaný za standard a oproti starší verzi je daleko bezpečnější. Dále zakážeme přihlášení přihlášení super administrátorského účtu root, vynutíme striktní režim, povolíme přihlašování heslem i pomocí veřejného klíče a zakážeme přihlašování prázdným heslem.

Konfigurační soubor klientské částí OpenSSH se nachází v souboru */etc/ssh/ssh_config* a obsahuje téměř ty samé direktivy jako u předešlé konfigurace serverové části. Zde ale není nutné nic měnit, protože standardní nastavení vyhovuje většině využití. Navíc jsme stejně při připojení odkázáni na to, co nám povolí cílový server.

Aby fungovalo připojení pomocí veřejného klíče, musíme nejdříve na centrálním serveru vygenerovat pár klíčů. Tedy klíč privátní a veřejný. Vygenerujeme je pomocí jednoduchého příkazu:

```
ssh-keygen -t dsa
```

Po zadání se nás zeptá nejdříve na umístění, kam se mají vygenerované klíče uložit. Nabídne nám standardní umístění, které se nachází adresáři .ssh aktuálně přihlášeného uživatele. Proto je vhodné se přihlásit jako uživatel, kterého jsme v jednom z minulých kroků vytvářeli.

Poté budeme ještě dotázáni na frázi, kterou se má daný privátní klíč zašifrovat. Při přihlašování pomocí klíče pak nezadáváme heslo k účtu, ale tuto frázi, která se použije pouze pro opětovné rozšifrování klíče. Sice je to v několika zdrojích nedoporučováno, ale lze použít i prázdnou frázi. Klíč pak tedy nebude zašifrován. Kdokoli by takový klíč získal, mohl by se přihlásit na naše vzdálené stanice bez hesla.

Pokud by se něco takového stalo a my se to dověděli včas, je možné ihned vygenerovat nový pár klíčů a ty rozdistribuovat do všech stanic. Já pro svoji aplikaci tuto metodu nezašifrovaného klíče použiji. Jednak kvůli možnosti přihlášení bez hesla, ale také protože si myslím, že takovýto přístup není méně bezpečný proti běžnému zadávání hesla.

Málokdo si hesla volí s dostatečnou složitostí a ještě navíc rozdílná ve všech účtech a službách. Situace s centrální správou dvaceti firewallů by znamenala dvacet různých a dostatečně silných hesel, který by bylo nutné vyplňovat ručně při každém připojení. Ani takto svědomitý přístup ale nezaručuje úplnou bezpečnost. Pokud se systém delší dobu nebude používat, hesla administrátor zapomene, nebo zůstanou někde zapsána (což je samozřejmě riziko).

Pokud bude navíc náš server cíleně kompromitován, není zas takový rozdíl mezi nalezením klíčů a odchycením hesel v čisté podobě. Zpět k vygenerovaným klíčům, standardně se tedy vygeneruje privátní klíč do souboru */home/centralni_sprava/.ssh/id_dsa* a veřejný klíč do souboru */home/centralni_sprava/.ssh/id_dsa.pub*. [12]

Privátní klíč je právě ten, který musíme chránit. V normálních situacích se takovému klíči nastaví souborová práva pouze pro čtení a pouze vlastníkem. Problém nastává při využití klíče třetí stranou, což je v mém případě webová aplikace, respektive SSH knihovna v PHP. Jelikož procesy webového serveru Apache běží standardně pod uživatelem a skupinou s názvem *www-data*[7], nebyl by k souboru s takovýmito právy umožněn přístup.

Možností by bylo například využít projektu suPHP[8], který sestává z modulu mod_suphp pro Apache a samotné aplikace suphp. Tento modul zajišťuje to, že Apache volá tuto aplikaci, která pak zabezpečí změnu uživatele pod kterým pracuje proces vykonávající PHP kód.

Další možností je nakonfigurování celého Apache serveru tak, aby jeho podprocesy běžely pod potřebným uživatelem, či skupinou. To se v systému Ubuntu a aktuálním balíku Apache provede nastavením v konfiguračním souboru */etc/apache2/envvars*, kde jsou následující dvě proměnné. Stačí tedy nastavit proměnnou APACHE_RUN_USER na uživatele *centralni_sprava*.

```
export APACHE_RUN_USER=centralni_sprava
export APACHE_RUN_GROUP=www-data
```

Pokud bychom chtěli, aby běžel pod specifickou skupinou, museli bychom nastavit druhou proměnnou. A navíc bychom ještě museli nastavit privátnímu klíči právo na čtení pro skupinu.

S tím přichází ještě třetí možnost a tou je pouhé přivlastnění privátního klíče přímo skupinou www-data a nastavení práva čtení pro skupinu.

Jednoznačně nejbezpečnější variantou je hned ta první. Tedy využití aplikace suPHP a vyhrazení uživatele s omezenými oprávněními. Veškeré potřebné PHP skripty jsou pak vlastněny tímto uživatelem, který má přístup i ke klíči.

---

7 server Apache startuje jako uživatel root aby mohl naslouchat na privilegovaném portu 80, potom vytvoří potomky pod uživatelem www-data, kterým předává veškeré požadavky a tím je zaručena bezpečnost

8 podrobnější informace na adrese http://www.suphp.org

4.3 Konfigurace klientů

Na jednotlivé stanice, které chceme ovládat naší aplikací nainstalujeme a nakonfigurujeme OpenSSH server stejným způsobem jako na server. Zde ale nepotřebujeme vytvářet sadu klíčů. Naopak server se bude pomocí klíčů přihlašovat na klienty. Pro to není třeba žádné nastavení, protože aplikace se poprvé přihlásí pomocí hesla a veškeré nastavení se postará sama. Jedná se vlastně jen o přenesení klíče a uložení do souboru s názvem *authorized_keys* v adresáři .ssh přihlášeného uživatele. Pokud bychom ještě chtěli zvýšit zabezpečení, existuje zde jednoduchá možnost. Do téhož souboru je možné před klíč vložit direktivu from v následujícím tvaru.

```
from="centralni_sprava.domain" ...klíč...
```

Tím docílíme toho, že OpenSSH server odmítne připojení z jiných serverů, než je ten, který jsme uvedli. Místo jména je možné uvést přímo i IP adresu, nebo za použití hvězdičky jako zástupného znaku celý rozsah adres.

Existují i další direktivy, které mohou zvýšit bezpečnost. Například direktivy no-port-forwarding a no-agent-forwarding, které zakazují směrování portů a agentů. Dokonce můžeme zakázat alokaci terminálu pomocí direktivy no-pty. To zde bohužel ale není možné, protože pro použití skriptu iptables-apply je potřeba interaktivní terminál. [4]

Práva

Nastává zde ale problém, na kterém závisí bezpečnost klientů. K čemu by byla správa firewallu, kdyby její samotná aplikace usnadňovala potencionální útok. Jde o to, že pokud se aplikace vzdáleně přihlásí je vše v pořádku. Ale aplikace potřebuje provádět příkazy, pro které je třeba administrátorské oprávnění (root). Příkladem je samotný ovládací příkaz iptables, který je nutné použít.

Samozřejmě je zde možnost přihlásit se vzdáleně jako uživatel root, což bychom museli povolit v OpenSSH konfiguraci. Ale toto řešení je nedostatečné a případný útočník, kterému by se podařilo ukrást soukromý klíč serveru, by měl rázem přístup k root účtu všech stanic. Proto je nutné, aby se aplikace přihlašovala pod účtem s klasickými právy.

Abychom mohli využívat zabezpečené aplikace, je tedy nutné požádat o zvýšení svých práv až po přihlášení a až tehdy, když je to zapotřebí. Ke zvýšení práv, slouží mimo jiné aplikace sudo, která se hojně využívá v systému Ubuntu. Ta ale potřebuje zadat heslo k účtu,

který má zvýšená práva. To je nechtěné chování, kvůli kterému by přišlo vniveč přihlašování bez hesla a opět bychom se dostali do situace s ukládáním hesel.

Aplikace sudo se ale řídí svým konfiguračním souborem */etc/sudoers*, případně soubory v adresáři */etc/sudoers.d/*, pomocí nichž lze aplikaci nakonfigurovat k potřebnému výsledku. Je zde možné nastavit jednotlivému uživateli nebo skupině zvýšená práva bez zadání hesla.

To samotné by samozřejmě nebylo o mnoho lepší, než přímé přihlášení pod uživatelem root. Konfigurace ale umožňuje u daného nastavení také uvést seznam aplikací, na které toto povolení platí. Tím povolíme pouze potřebné zabezpečené aplikace, k ostatním bude přístup klasicky odepřen. Konfigurace bude vypadat následovně. [4]

```
centralni_sprava     ALL = NOPASSWD: /sbin/iptables,
/sbin/iptables-save, /sbin/iptables-apply
```

Podobné řešení jsem nalezl ještě jedno. To spočívá v samotném souboru protokolu ssh – authorized_keys, ve kterém lze vynutit příkaz po přihlášení pomocí ssh. Nedojde tak vůbec k terminálovému přístupu. Ale v tomto případě, kdy potřebuji provádět vzdáleně více než jeden příkaz, by bylo nutné navíc ještě vytvořit jeden spustitelný skript. V tom by se muselo pomocí předávaného parametru rozhodovat, jaký příkaz následně zavolat.

Oproti předchozí by tato možnost byla nejspíše jen o něco málo bezpečnější, ale vyžadovala by dodatečné vytvoření skriptu. Tomuto skriptu by se navíc také muselo povolit (např. pomocí sudo) spouštění iptables pod právy superuživatele.

4.4 Konfigurace aplikace

Nyní je již možné přejít k návrhu samotné aplikace, databázové části, návrhu tříd a podobně, ale nejdříve ještě zbývá zprovoznit a nakonfigurovat základ aplikace. Ten bude stejný ať už bude návrh jakýkoli a zjistíme tím, jestli jsme dobře nakonfigurovali celé prostředí.

Framework Nette je ke stažení v sekci download z webu [9] aktuální stabilní verzi (v době psaní této práce verze 2.0.10). K dispozici je na výběr verze pro PHP 5.3/5.4, která bude použita v této práci a verze pro starší PHP 5.2 s prefixy nebo bez (v PHP 5.2 nejsou jmenné prostory proto je zde nutná obezřetnost aby nedošlo ke kolizi jmen). Stáhnutý archiv obsahuje tyto adresáře:

- *API-reference* – kompletní API frameworku vygenerované do formy HTML pro snadné a přehledné prohlížení ve webovém prohlížeči,

- *client-side* – adresář se skripty pro klienta, obsahuje například javascript pro validaci formulářů v Nette,
- *examples* – jednoduché vzorové příklady použití frameworku,
- *Nette* a *Nette-minified* – adresář se samotným frameworkem sestávajícím z mnoha souborů (pro každou třídu je jeden soubor), minified verze je potom minimalizovaná verze, kde jsou veškeré soubory spojeny do jednoho – to ulehčuje serveru, který tak nemusí načítat stovky souborů z disku ale jen tento jeden,
- *sandbox* – adresář se základní předpřipravenou adresářovou strukturou, která byla s malými úpravami použita,
- *tests* – jednotkové testy frameworku, kterými jej lze automaticky otestovat,
- *tool* – různé nástroje, jako je například Requirments-Checker který otestuje webový server na potřebné vlastnosti pro správný běh frameworku, a další.

Kontrola závislostí

Jako první věc je vhodné otestovat, zda jsou splněny alespoň ty nejzákladnější požadavky. To nám řekne právě výše zmíněný Requirments-Checker. Ten kontroluje existenci a správnou konfiguraci serveru. Všechny požadavky nemusí být striktně dodrženy, pouze pak nebude fungovat určitá funkcionalita. Například pokud nemáme nainstalován modul GD, nebudou možné určité operace s obrázky. Všechny základní požadavky by již měly být splněny předchozí instalací, ale pokud se nějaký vyskytne, je nutné doinstalovat požadovanou knihovnu. To provedeme klasicky přes preferovaný instalační nástroj (aptitude, synaptic). Nebo máme možnost použít nástrojů PECL, nebo PEAR. Oba dva dělají v podstatě to samé a tím je distribuce PHP knihoven a komponent.

Kostra

V případě, že máme v pořádku požadavky, můžeme začít vytvářet prostředí pro aplikaci. Nette framework nám násilně nic nenutí a lze použít jakákoli adresářová struktura. K dispozici ale dává adresář sandbox (někdy nazýván jako skeleton, neboli kostra), který obsahuje předpřipravenou a doporučenou adresářovou strukturu. Tento adresář jsem si tedy zkopíroval do kořenové složky webového serveru. Základní struktura tedy vypadá následovně.

- /www – zde je kořenový adresář webu, neboli document_root a sídlí zde jediný[9] vstupní bod aplikace, neboli skript index.php, dále jsou zde adresáře s javascripty a kaskádovými styly,
- /app – adresář, kde je umístěna celá aplikace a z webového prohlížeče je sem, stejně jako do dalších adresářů, zakázaný přístup,
- /temp a /log – adresář pro dočasné odkládací soubory a adresář pro logy – oba musí mít právo zápisu pro uživatele, pod kterým běží webový server Apache (většinou uživatel www-data),
- /libs – veškeré knihovny.

Do adresáře /libs je tedy nutné ještě nakopírovat framework jako takový, což je vlastně také knihovna. Během vývoje se používá vícesouborová verze, do produkčního režimu se pak používá jedno-souborová. Ta je takzvaně minifikovaná. Tento proces zahrnuje složení všech skriptů do jednoho a odstranění zbytečných prázdných znaků (tabulátory, odřádkování). Důvodem je menší velikost a snížení počtu diskových operací serveru. Během vývoje se ale nepoužívá, protože pak ladící nástroje nedokáží říci, na kterém řádku ve zdrojovém kódu došlo k chybě.

Konfigurace

První věcí, kterou je třeba nakonfigurovat je vstupní bod aplikace, neboli skript *index.php*. Ten ale v případě Nette obsahuje pouze nastavení konstant s absolutními cestami k nejzákladnějším adresářům. Zároveň to jsou téměř jediné globální konstanty celého frameworku. Ty pak na několika místech slouží k odkazování se přímo na dané umístění. To je ale potřeba jen v konfiguračním skriptu *bootstrap.php* a ojediněle přímo v aplikaci. V rámci aplikace se totiž odkazuje pomocí technik frameworku, které budou uvedeny dále.

```
<?php
define('WWW_DIR', __DIR__);
define('APP_DIR', WWW_DIR . '/../app');
define('LIBS_DIR', WWW_DIR . '/../libs');

$container = require APP_DIR . '/bootstrap.php';
$container->application->run();
```

9 Jediný vstupní bod je zajištěn modulem rewrite a konfiguračním souborem .htaccess, který říká, že jakýkoli požadavek se má přeložit na skript index.php a přesměrování si pak řídí framework sám

Jde vlastně o konfiguraci adresářové struktury popsané v předchozí kapitole. Pokud bychom se rozhodli z nějakého důvodu pozměnit ji, stačí upravit tyto konstanty.

Poslední dva řádky volají konfigurační skript *bootstrap.php*, který vytváří takzvaný Dependency Injection kontejner. Což je v podstatě třída, která obsahuje továrničky pro veškeré dostupné služby. Ty jsou pak díky tomuto kontejneru dostupné skrze téměř celou aplikaci. Dependency Injection jako takové je jedním z mnoha návrhových vzorů. Tento vzor říká, že o závislosti jednotlivých tříd si neshání tyto třídy samy, ale jsou jim dodávány z venku někým jiným. Jakým způsobem už tento vzor nespecifikuje, může to být pomocí konstruktoru, nebo metodou (setterem).

Uvažujme třídu, která potřebuje ke své práci databázi. Klasickým přístupem by bylo, kdyby si tato třída obstarala databázové připojení z nějakého statického kontejneru, nebo ještě hůře, přímo z globálního prostoru. Pokud použijeme návrh dle vzoru Dependency Injection, předáme této třídě databázové připojení konstruktorem, nebo metodou. Toto databázové připojení tedy dostaneme vždy od spodnější vrstvy. V nejspodnější vrstvě ale už nám nemá kdo služby dodávat a právě proto existuje Dependency Injection kontejner.

Tento popis by mohl zavádět k tomu, že nás framework vysloveně nutí používat tento návrhový vzor, opak je ale pravdou. Pokud nechceme, můžeme v kontejneru nakonfigurovat pouze základní nastavení a služby jako takové si můžeme obstarávat jak jsme zvyklí. Jelikož mi to ale framework usnadní a bude díky tomu návrh celé aplikace čistší, není důvod toho nevyužít.

Tento hlavní kontejner spolu s celou aplikací musíme tedy nejdříve nakonfigurovat. To se provádí ve skriptu *bootstrap.php* spolu s konfiguračním souborem *app/config/config.neon*. Skript *bootstrap.php* obsahuje pouze konfiguraci ladícího režimu, nastavení adresáře pro odkládací soubory a zapnutí RobotLoaderu. To je knihovna, která automaticky načítá potřebné skripty, které neustále nemusíme načítat manuálně pomocí klíčových slov require, či include.

Dále už jen načítá zmíněný konfigurační soubor, ve kterém se konfiguruje zbytek nastavení. Tento soubor je rozdělen na sekce common, production a development, neboli sekce pro společnou, produkční a testovací konfiguraci. Jako první je zde konfigurace parametrů připojení k databázi a umístění ssh klíčů. Při zprovoznění této aplikace na jiném serveru, při změně databáze, nebo úpravě umístění ssh klíčů, stačí pak upravit tuto sekci:

```
parameters:
        database:
              driver: mysql
              host: 127.0.0.1
              username: diplomova_prace
              password: diplomova_prace
              database: diplomova_prace
              charset: utf8
        ssh:
              keyAlgorithm: ssh-dss
              privateKey: /home/diplomova_prace/.ssh/id_dsa
              publicKey: /home/diplomova_prace/.ssh/id_dsa.pub
```

Dále je zde sekce nette, ve které se konfigurují části samotného frameworku jako jsou sessions, ladící nástroj, odchytávání výjimek s chybovým presenterem. Tyto konfigurace nejsou tak zajímavé pro podrobný popis, proto je zde nebudu uvádět.

Zajímavější je ale sekce s databázovým připojením jako takovým a se službami, neboli services. Zde jsem nakonfiguroval veškeré služby, které bude kontejner poskytovat. Služby jsou vlastně modelovou vrstvou aplikace a závisí na návrhu tříd a návrhu databáze, které budou detailněji popsané dále v textu.

Konfigurace služby se skládá z jejího názvu a po oddělení dvojtečkou následuje název php třídy která tuto službu bude reprezentovat. Většina služeb navíc ještě závisí na jiných službách, případně parametrech, které se uvádí do kulatých závorek. Jednotlivé závislosti se uvozují zavináčem a jedná se o předávání závislostí konstruktorem, a tedy o vzor Dependency Injection.

Při vytváření kontejneru se pak vlastně vygenerují továrničky, které po zavolání vytvoří instanci uvedené třídy a konstruktoru předá dané parametry a instanci nám vrátí. My pak už nejsme nuceni při každém vytváření služby shánět ony závislosti a službě je předávat. Udělá to za nás někdo jiný a tím je právě tento kontejner a jeho továrničky.

Tento návrh má ještě jednu dobrou vlastnost a tou je takzvaný Lazy Loading, neboli opožděné nahrávání. To znamená, že tyto služby (třídy) nejsou instancovány už při spouštění aplikace, ale až když je jich potřeba před jejich voláním. Některé nemusí být potřeba vůbec a tak se po celý běh aplikace nevytvoří a nezabírají tudíž paměť. Některé vybrané služby pak vypadají například takto:

```
db: @Nette\Database\Connection
```

```
authenticator: Common\Models\Security\Authenticator(@users,@loginHistory)

menu: FirewallModule\Models\Menu
users: FirewallModule\AclModule\Models\Users(@db::table(users))
loginHistory:
FirewallModule\Models\LoginHistory(@db::table(login_history))
roles: FirewallModule\AclModule\Models\Roles(@db::table(roles))
```

Ze zápisu je jasné, že definujeme službu s názvem db, která se vytvoří z třídy *Nette\Database\Connection* . Tato služba je ale speciální, protože je přímo součástí frameworku a zavináč značí, že už je konfigurovaná někde jinde v kontejneru. Tato třída má samozřejmě nějaké parametry, ale Nette disponuje funkcí auto-wiringu. To znamená, že než framework vygeneruje tělo továrničky, podívá se na konstruktor dané třídy. Tam zjistí parametry, které daná třída vyžaduje pro vytvoření a pak se je dle jejich názvů a datových typů snaží dohledat a předat.

Dále je zde například služba authenticator (slouží k přihlašování uživatel) a je reprezentována třídou Authenticator ze jmenného prostoru *Common\Models\Security.* Jako parametry ji předáme závislosti služby users a loginHistory, které mají samozřejmě také své závislosti. Tímto tedy velice jednoduše popisujeme hierarchii závislostí a framework za nás vygeneruje kontejner s továrničkami, jež by jsme si museli psát jinak ručně sami.

Výsledkem je tedy kontejner s vygenerovanými továrničkami služeb, přes který se spouští celá aplikace a tyto služby nám zpřístupňuje. K tomuto kontejneru se uvnitř aplikace, přesněji ve všech presenterech, dostaneme přes proměnnou s názvem *context*. Ze služeb (modelů) se dle principů MVC do jiných vrstev aplikace nedostaneme a bylo by tedy chybou návrhu kdybychom si do modelů předávali celý context. Proto se předávají vždy jen potřebné závislosti, což jsou většinou databázové tabulky, nebo další modely. Tento návrh by bylo složitější dodržovat v běžné PHP aplikaci, ale zde nám ho velice usnadňuje právě framework s předchozí konfigurací závislostí. Výsledkem je čistší a přehlednější kód, což je velice důležité pro odhalování chyb a další úpravy.

Soubor *bootstrap.php* kromě načítání tohoto konfiguračního souboru nastavuje aplikaci do jednoho ze dvou režimů. Pro vývoj se používá DEVELOPMENT režim a pro produkční analogicky PRODUCTION režim. V produkčním režimu jsou frameworkem potlačeny veškeré chybové výstupy, stejně jako jsou odchytávány výjimky, které by se dostaly až na chybový výstup. Místo toho jsou vyvolávány chybové Presentery a šablony a prohlížeči jsou posílány

standardní stavové kódy označující chybu, neboli stav 404, 500 a podobně. Ve vývojovém režimu je vypisování chyb vyžadovaná vlastnost a proto je zobrazován ladící nástroj Laděnka.

Navíc se v tomto souboru nachází ještě konfigurace takzvaného routování, neboli směrování. To v aplikaci znamená převod URL adresy na správný Presenter a správnou akci tohoto Presenteru. Taktéž musí být zajištěn převod v opačném směru, když framework generuje z Presenteru, nebo šablony nový odkaz s URL adresou, tak musí vědět jaký tvar vygenerovat.

Aplikace bez směrování fungují pouze se jmény php souborů přímo v URL adrese, takže taková adresa pak vypadá například takto:

www.domena.cz/akce.php?parametr=1&strana=2

Směrování nám dává možnost použít adresu ve tvaru například:

www.domena.cz/akce/2?parametr=1

A právě podle konfigurace směrování může takováto adresa vyvolat například třídu DefaultPresenter a její metodu actionAkce s parametrem 1. Volba je při použití směrování na nás a největší výhodou je, že se toto nastavení může změnit až po vytvoření celé aplikace, když se rozhodneme, že by nějaký Presenter měl být dostupný pod jiným tvarem adresy. Navíc nám umožňuje vytvořit vícejazyčné tvary adresy, kdy na jeden Presenter povede více různých adres a on dle parametru jazyka načte správný obsah. Tato možnost v aplikaci využita nebyla, ale díky ní by přeložení celé aplikace bylo do budoucna jednodušší.

V nejnovější verzi frameworku se tato konfigurace směrování přesunula do vlastní třídy RouterFactory.php se stejnojmennou třídou a metodou createRouter(). Jinak se použití nezměnilo a směrování této aplikace je v tuto chvíli pouze takto jednoduché:

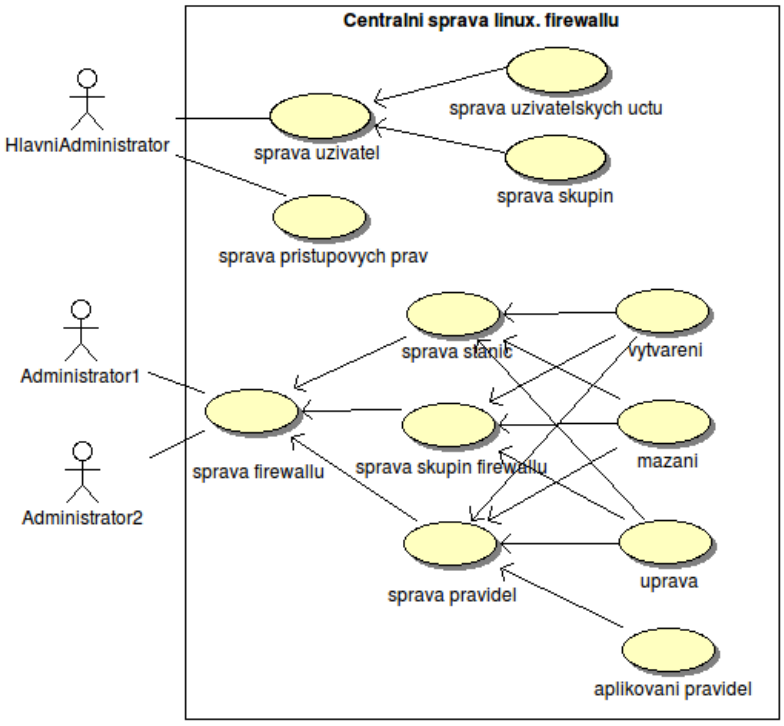
```
$router = new RouteList('Firewall');
$router[] = $userRouter = new RouteList('User');
$userRouter[] = new Route('user/<action>[/<id>]', 'User:profile');
$router[] = new Route('<presenter>/<action>[/<id>]',
'Homepage:default');
```

První řádek značí, že pro všechny podřazené routy se budou třídy hledat v modulu Firewall. Tento modul pak ještě obsahuje vnořený modul s názvem User, viz druhý řádek. Ten už má v sobě jednu routu, která definuje adresu ve tvaru prvního parametru. Takže adresa *www.domena.cz/user/profile/1* povede na třídu User a její metodu *actionProfile($id = 1)*. Poslední routa je standardní a bere v potaz všechny ostatní Presentery a jejich akce v celé aplikaci. Takže když vytvořím třídu presenteru s názvem FirewallPresenter a v ní metodu acti-

onList, tak bude tato metoda dostupná pod adresou www.domena.cz/firewall/list. Za povšimnutí stojí to, že jsou routy definovány relativně, je jedno na jaké běží doméně. [9]

4.5 Use Case diagram

Tento diagram[10] je v podstatě velice jednoduchý. Celý systém by měl být variabilní a toto je jen jedna z možných variant použití. V ideálním případě by zde měla být taková správa uživatel a oprávnění aby bylo možné vytvářet různé množství správců. Ti pak mohou mít různá oprávnění ke správě firewallů dle uživatelských rolí.



Ilustrace 3: Use Case diagram

4.6 Návrh databázového modelu

Návrh byl prováděn ve volně dostupné aplikaci MySQL Workbench, která umožňuje jednoduchou synchronizaci modelu se skutečnou databází. Výsledný soubor jsou samozřejmě spolu s vygenerovanými SQL skripty součástí aplikace, viz Příloha – Obsah CD.

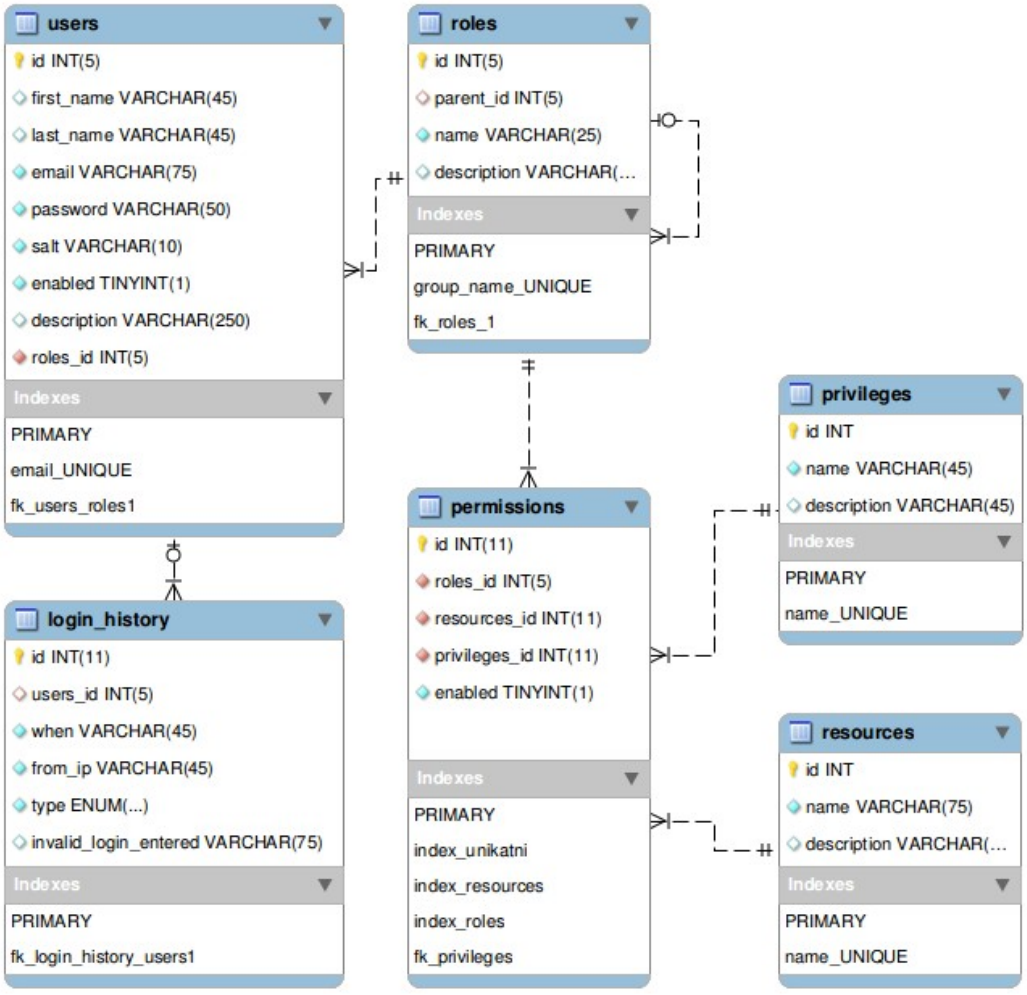
Z návrhu mi postupně vyšlo to, že bude vhodné rozdělit model na dva. Nejen při návrhu ale i přímo v aplikaci použít vlastnost frameworku Nette, který umožňuje spolu související části rozdělit do takzvaných modulů. V aplikaci bude totiž obsažená správa uživatelů, kteří

10 omluva autora za diagram bez diakritiky, použitý UML modelovací nástroj ji nepodporoval

budou moci tento systém využívat spolu se správou jejich oprávnění. Tento celek bude tedy jedním modulem a druhým bude vlastní správa firewallů. Tímto bude projekt přehlednější a navíc umožní případně časem přidávat další moduly.

Uživatelská část

Model uživatelské části sestává z klasických tabulek *users* a *roles* pro uživatele s jejich role. Při zpětném pohledu na model by asi obecně bylo vhodné, aby uživatel mohl mít více rolí, tedy mezi těmito tabulkami zavést vazbu M:N. Při počtu uživatel, kteří tento systém budou pravděpodobně používat je ale takovéto rozčlenění zbytečné a jedna skupina pro uživatele by měla stačit pro všechny možnosti použití.



Ilustrace 4: Návrh databáze – uživatelská část

K těmto tabulkám není moc co říci, všechny sloupce mají samovysvětlující názvy. Jen sloupec salt u uživatele nemusí být jasný. Měl by tu být uchováván náhodný řetězec, který by

měl být zašifrován spolu s heslem. Pokud by se někdo dostal k této databázi, a hesla by byla zašifrována bez tohoto řetězce, mohl by útočník hesla odvodit. Na internetu se totiž nachází databáze běžně používaných hesel a jejich zašifrovaných verzí. Toto opatření alespoň trochu zvýší zabezpečení celé aplikace.

Pro evidenci přihlášení je zde tabulka *login_history*. Ta uchovává informace o tom, kdo se kdy připojil z jaké IP adresy. Obsahuje také sloupec type, který udává, jestli šlo o platné přihlášení, nebo bylo špatně zadané přihlašovací jméno, či heslo. Tím je možné kontrolovat, zda někdo záměrně nezkouší hesla k nějakému účtu a případně zavést protiopatření.

Dále je zde trojice tabulek *permissions*, *privileges* a *resources*. Jsou to běžné názvy pro návrh takzvaného ACL, neboli Access Control Listu, což je v překladu systém řízení přístupu. Pomocí tohoto systému jsme pak schopni povolit někomu provedení určité operace nad nějakým zdrojem. Zdroje se ukládají do tabulky *resources* a může jimi být cokoli, například názvy jednotlivých Presenterů, nebo jiné logické zdroje, ke kterým budeme chtít řídit přístup.

Tabulka *privileges* pak obsahuje výčet jednotlivých akcí, které na všech daných zdrojích budeme chtít povolovat. Třetí jmenovaná tabulka je pak vlastně jen logickým spojením těchto dvou tabulek s tabulkou *roles*, jelikož nebudeme řídit přístup jednotlivým uživatelům ale celým skupinám, neboli rolím. Jeden řádek v tabulce privileges pak vlastně říká, že pro danou roli povoluje danou akci nad daným zdrojem.

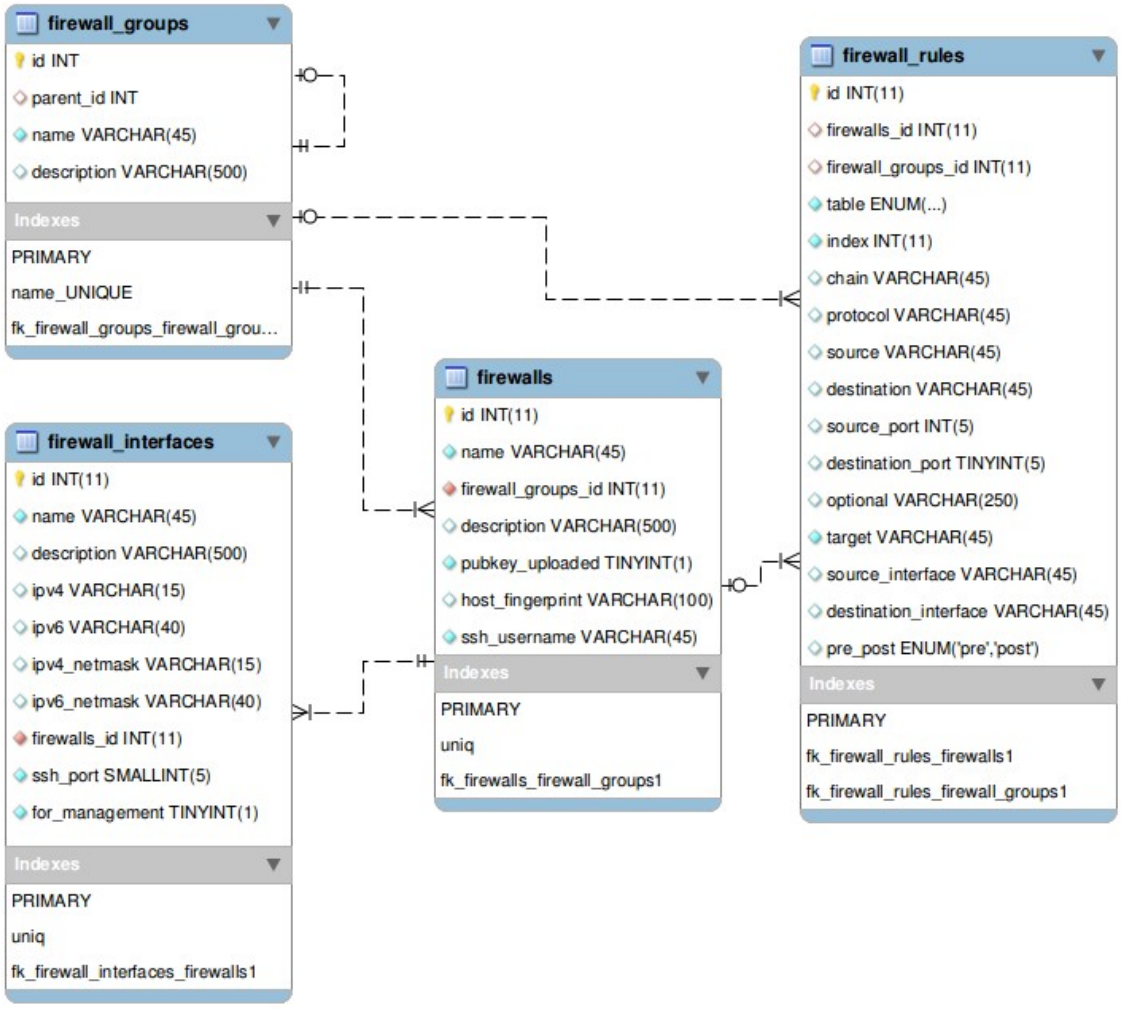
Tato architektura dovolí docela jemné řízení přístupu napříč aplikací. Samozřejmě, že se pak na příslušných místech v kódu aplikace musí tato nastavení zohlednit stejně, jako kdyby byla oprávnění nějak napevno nastavena v kódu. Takto ale bude mít možnost administrátor ovlivnit přístup podle aktuálních potřeb.

Firewally a pravidla

Tato část modelu obsahuje čtyři tabulky. První z nich je tabulka *firewalls*, která slouží pro základní informace o samotných vzdálených strojích[11], na kterých budeme řídit firewally. Je zde nutné uchovávat uživatelské jméno (sloupec *ssh_username*) pod kterým se bude aplikace k danému stroji přihlašovat, ale také například informaci o tom, zda je již na tento vzdálený stroj nahrán veřejný klíč (sloupec *pubkey_uploaded*). Sloupec *host_fingerprint* by měl sloužit pro bezpečné rozpoznání toho stroje, aby se zamezilo MITM útoku.

11 Pod pojmem firewall je tedy v celé aplikaci považován vzdálený stroj, jehož paketový filtr ovládáme

Aby aplikace věděla na jakou adresu a port se má připojit, musí mít informace o rozhraní daného firewallu a na to je zde tabulka *firewall_interfaces*. Nejspíše by se tyto informace daly ukládat do předchozí tabulky, ale při návrhu bylo myšleno na to, že by do budoucna byla vhodná možnost přiřadit jednomu firewallu více rozhraní.



Ilustrace 5: Návrh databáze – firewally a pravidla

Klíčová by měla být v některých situacích možnost seskupit více spravovaných firewallů do skupin. Proto je zde tabulka *firewall_groups*, která obsahuje v podstatě jen jméno, popis skupiny a ID nadřazené skupiny spolu s cizím klíčem na sebe sama. Skupiny bude tedy možné jednoduše hierarchicky uspořádávat.

Řešení stromové struktury v relačních databázích za pomocí jediného sloupce s odkazem na rodiče je nejjednodušší z mnoha dostupných řešení. Skýtá mnoho nevýhod při průchodu a hlavně při manipulacích se stromem, jako je odstranění nebo přesun nějakého prvku. Tyto

operace jsou pak v tomto případě složitější a ve výsledku také pomalejší. V tomto případě ale na rychlosti ani složitosti nezáleží z důvodu toho, že zde nebude tak obrovský strom, aby to způsobilo nějaké potíže. Naopak bude tento návrh jednoduché implementovat.

Nyní už zbývá poslední tabulka, která je v podstatě nejdůležitější. Je to tabulka *firewall_rules* a je jasné že obsahuje jednotlivá pravidla firewallů. Nese tedy veškeré informace důležité pro sestavení iptables příkazu. Jde především o název tabulky firewallu (mangle, nat, filter, …), řetězec do kterého patří (input, prerouting, …), protokol, zdrojový a cílový port, zdrojovou a cílovou adresu, zdrojové a cílové rozhraní a cíl příkazu (accept, drop, …).

Zbylé nadstandardní parametry iptables příkazu mají své místo ve sloupci *optional*, protože nejsou tak často využívány, aby musely mít každý svůj sloupec. Je zde také sloupec *index*, který je velice důležitý pro řazení pravidel správně po sobě. Tato tabulka je cizím klíčem provázána s tabulkou *firewall*, ale také s tabulkou *firewall_groups*. To je z důvodu, že budou moci existovat pravidla skupinová, která budou aplikována pro celou skupinu a k nim se přidají pravidla jednotlivých firewallů.

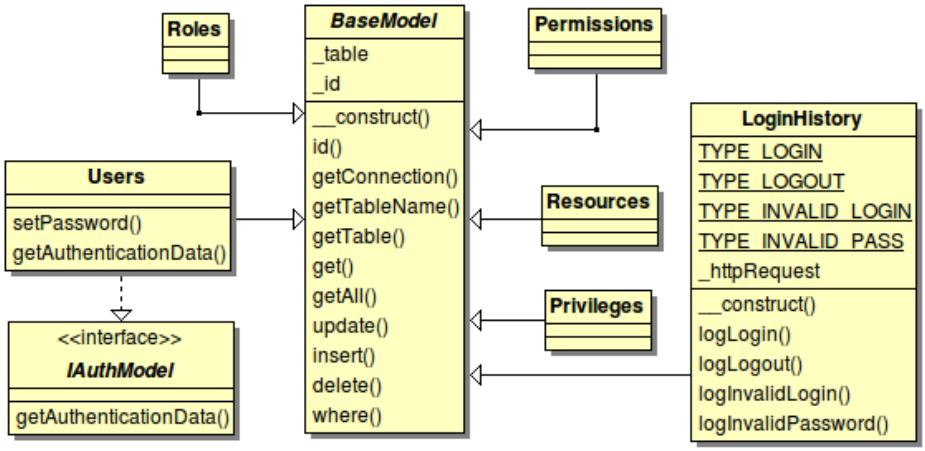
4.7 Návrh tříd

Návrh tříd je rozdělen také do již zmiňovaných modulů, ale navíc ještě do vrstev dle architektury MVC. Je to z důvodu rozsáhlosti návrhu tříd. Vrstva View bude ale vynechána, protože ji netvoří třídy, ale rovnou jednotlivé soubory šablon. Do návrhu tříd tedy nepatří.

ACL část – vrstva modelů

Z diagramu, viz následující obrázek, je patrné, že vrstva modelů je velice jednoduchá. Tvoří ji základní třída BaseModel, která je předkem všech modelů v aplikaci. Je to z důvodu toho, že většina modelů má podobný charakter. Jejich stav je většinou ukládán do databáze, proto má základní model jako vlastnost ID záznamu a název tabulky. Dále jsou zde definováno společné rozhraní (metody) pro práci s těmito daty. Pro základní operace vytvoření, úpravy a odstranění se vžila zkratka CRUD z anglických názvů těchto tří slov (create, update, delete). Dále jsou zde metody pro získání jednoho záznamu *get()* a všech záznamů pomocí *getAll()*. Pro potřeby omezení výsledků je zde meotda *where()*. Pak už tato třída obsahuje jen metody pro získání názvu tabulky, instance této tabulky a instance celého databázového připojení.

Ostatní třídy, pokud to potřebují, přidávají své vlastní metody. Tak jako je to v případě třídy *Users*, která deklaruje ještě metodu pro nastavení hesla. Navíc ještě implementuje metodu pro získání přihlašovacích údajů z důvodu toho, že implementuje rozhraní *IAuthModel*. To znamená, že tuto třídu je možné použít na místě, kde je očekáván model pro autentizaci. Zde je to logické, protože v této aplikaci bude docházet k autentizaci pomocí informací z databázové tabulky, kde je uloženo uživatelské jméno a zašifrované heslo.



Ilustrace 6: Model tříd, ACL modul, vrstva modelů

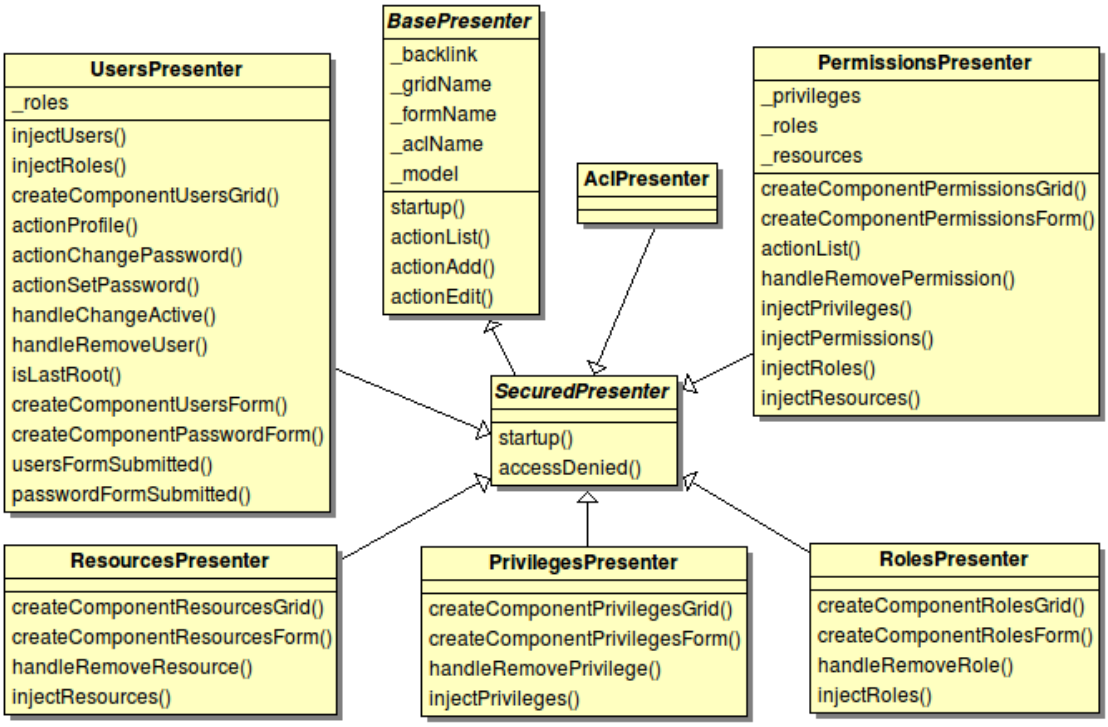
Třída modelu LoginHistory obsahuje především konstanty typů přihlášení a vlastnost typu HttpRequest, která je potřeba pro získání IP adresy. Pak jsou zde metody pro zaznamenání jednotlivých stavů přihlášení.

ACL část – vrstva presenterů

Tato část je oproti předchozí poměrně rozsáhlejší. Je to hlavně díky tomu, že jsou formuláře a tabulky vloženy do vlastních tříd. Mohly by být přímo součástí třídy, ve které se používají, jako je to v případě třídy *UserPresenter*. Pokud jsou ale mimo, je vše přehlednější a navíc nám to dává možnost využít daný formulář, nebo tabulky ve více presenterech, zvyšuje se tedy znovu-použitelnost kódu. Model se všemi těmito třídami by byl rozsáhlý, proto jsem jej ještě rozdělil na dva. První ukazuje hirearchii presenterů a druhý závislosti na formulářích, tabulkách a modelech.

Hierarchie tříd je zde taková, že všechny presentery, které mají být dostupné pouze přihlášeným uživatelům, dědí od abstraktní třídy *SecuredPresenter*. Tento presenter zajišťuje zabezpečení a před každým načtením některého z potomka hlídá v metodě *startup()* jestli je uživatel přihlášen. Až tato třída dědí od základního abstraktního presenteru *BasePresenter*, kde jsou

společné prvky. Nejdůležitější jsou metody s prefixem action. Jsou to metody pro CRUD operace a zde se nachází ověření, zda má přihlášený uživatel oprávnění k dané operaci.

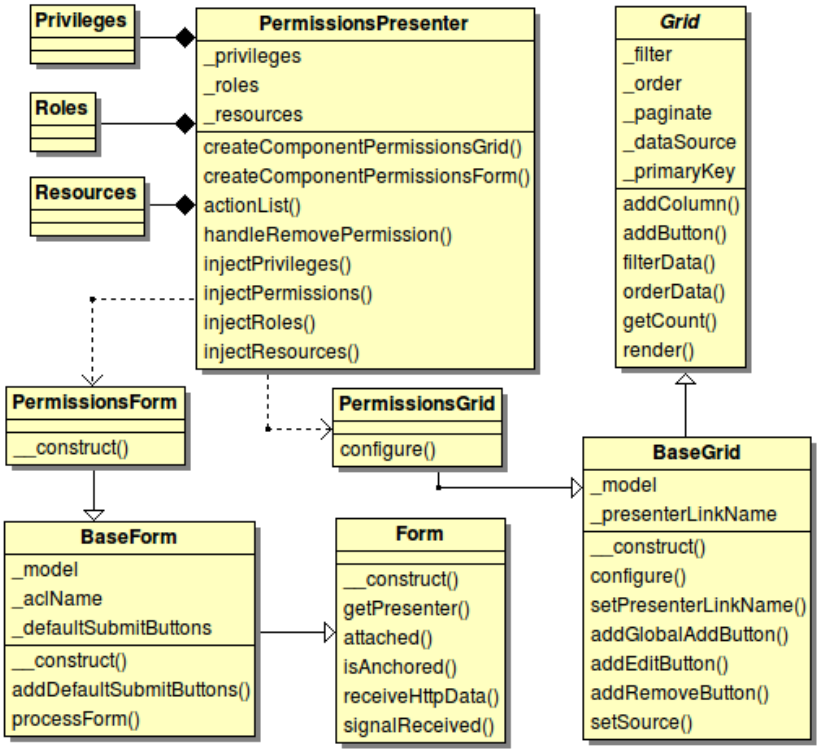


Ilustrace 7: Model tříd, ACL modul, vrstva presenterů

Dostupné třídy (ty co nejsou abstraktní) obsahují většinou metodu, nebo metody s prefixem inject. Jedná se o metody které nastavují danému presenteru nějakou závislost definovanou v konfiguračním souboru. Framework takovéto metody volá automaticky. Dále jsou zde továrničky pro zmíněné formuláře a tabulky. To jsou metody s prefixem *createComponent* a framework je opět volá automaticky když jsou potřeba. V poslední řadě jsou zde akce a handlery, což jsou metody s analogickými prefixy action a handle. Handlery jsou vlastně specifickými akcemi, které se většinou využívají při AJAX/AJAJ požadavcích. Obecně jsou to požadavky, které probíhají na pozadí a nevynucují překreslení celé webové stránky.

Formuláře jsou pojmenované pro přehlednost s postfixem Form a obsahují většinou pouze konstruktor, ve kterém se nastavují prvky daného formuláře. Je zde opět využito dědičnosti a společné vlastnosti a operace jsou v předkovi všech formulářů, neboli třídě *BaseForm*. Zde je metoda *processForm()* pro základní zpracování formuláře a metoda pro přidání standardních odesílacích tlačítek (potvrzení a storno). I tato třída má však ještě předka a tím je tří-

da Form, která je součástí frameworku a zajišťuje veškeré zpracování a zabezpečení formulářů.



Ilustrace 8: Model tříd, ACL modul, vrstva presenterů – závislosti

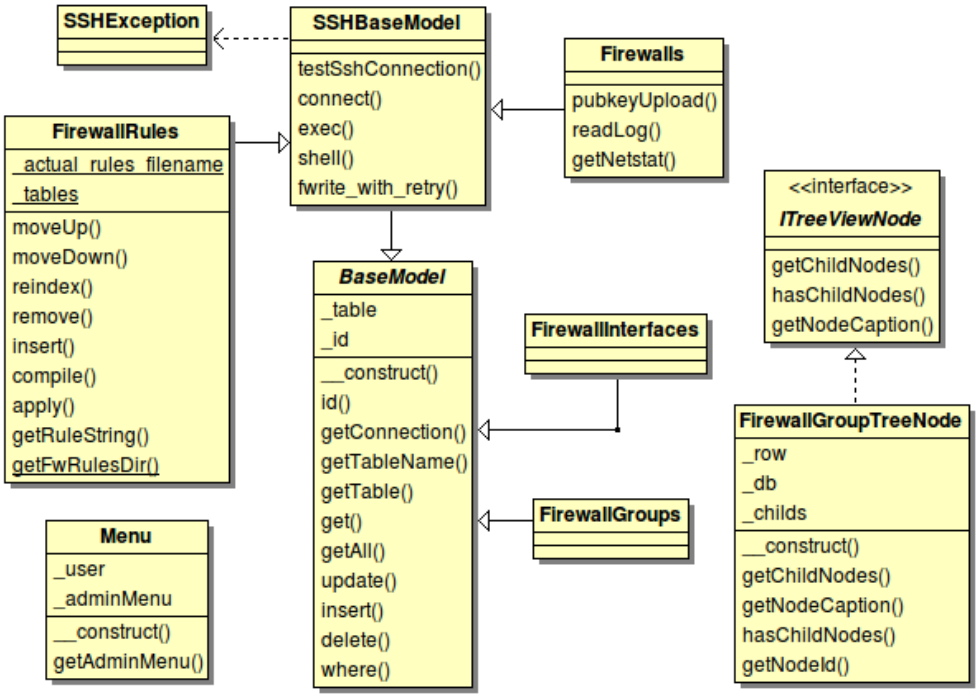
Podobný scénář je u tabulek, které mají postfix Grid. Taktéž mají předka *BaseGrid*, který dědí od třídy *Grid*. Ten však není součástí frameworku, ale v tomto případě se jedná o součást volně dostupné knihovny NiftyGrid.

Modul Firewall – vrstva modelů

Zde už je modelová vrstva o něco složitější. Předně je tu osamocena třída Menu, která slouží pouze pro získání prvků menu. Pak zde opět figuruje stejný *BaseModel* jako je v předchozí části, protože tuto třídu sdílejí oba dva moduly. Od něj dědí třída *SSHBaseModel*, která je zase společná pro modely, které využívají metod pro práci s ssh spojením. Proto zde lze nalézt například metody *connect()*, *exec()* a *shell()*. Ty jsou v podstatě jen nadstavbou nad nativními PHP funkcemi pro práci s protokolem SSH. Přidávají ale snazší použití a pro chybové stavy využívají vlastní výjimku *SSHException*.

Na tomto modelu jsou založeny dvě třídy, jedna reprezentující samotné firewally a druhá pro práci s jejich pravidly. U firewallů je potřeba metoda pro nahrávání veřejného klíče, aby

bylo možné další přihlašování bez hesla. Dále metoda pro čtení záznamů z daného firewallu a také doplňková metoda pro získání stavu aplikací netstat, která vypisuje aktuální stav připojení daného vzdáleného stroje. Třída *FirewallRules* musí mít oproti základnímu modelu ještě metody pro přesun pravidla nahoru a dolu, přeindexování, zkompilování a aplikování pravidel.

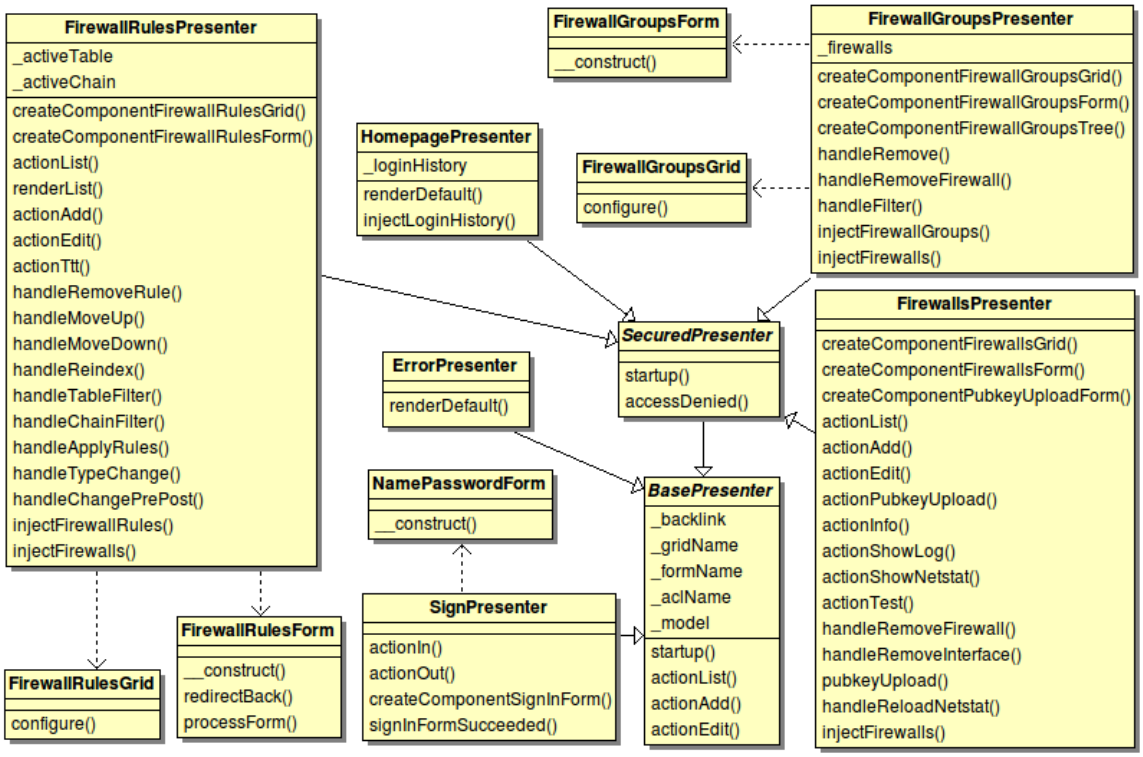


Ilustrace 9: Model tříd, Firewall modul, vrstva modelů

Třída *FirewallGroups* představuje skupiny firewallů. Ty ale mají být ale stromově uspořádány, proto jsem zde využil volně dostupnou knihovnu Tree pro Nette framework. Proto jsem vytvořil třídu *FirewallGroupTreeNode*, která implementuje rozhraní této knihovny a její povinné metody. Tato třída tedy reprezentuje jeden prvek ve stromové struktuře a je využitelná touto knihovnou pro zobrazení stromových struktur.

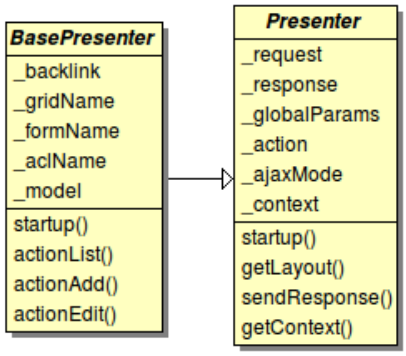
Modul Firewall – vrstva presenterů

Tato vrstva má vlastně stejnou architekturu jako vrstva presenterů u ACL modulu. Je zde navíc *SignPresenter*, který slouží pro přihlášení do aplikace *a ErrorPresenter*, který je využí-



Ilustrace 10: Model tříd, Firewall modul, vrstva presenterů

ván pro chybové stavy, především pro neoprávněný přístup. Pak je zde třída *HomepagePresenter*, což je vlastně rozcestník pro všechny další presentery. Jinak jsou zde ve třídách pouze akce a handlery pro práci s firewally a jejich pravidly. Ty je potřeba filtrovat a podobně, proto je těchto metod potřeba tolik.

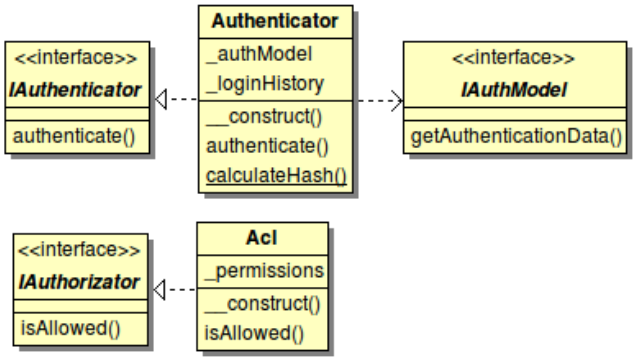


Ilustrace 11: Model tříd, Presenter

Ještě nutno podotknout, že ve vrstvách presenterů není uvedena vazba na základní třídu frameworku. Je to třída Presenter a v diagramech by byla zobrazena tak jako je na předchozím diagramu [Ilustrace 11].

Zabezpečení

Zbývající třídy se nehodí ani do jednoho modulu, proto je uvedu zvlášť. Jedná se o modely, které se využívají pro zabezpečení, přesněji pro autorizaci a autentizaci. Framework poskytuje rozhraní (*IAuthenticator*, *IAthorizator*), která stačí implementovat a naše třídy pak v konfiguraci předat. Je zde také vidět využití rozhraní IAuthModel, ze kterého si bere přihlašovací informace. V mé aplikaci jej implementuje pouze model Users, ale pokud bychom potřebovali přidat jiný typ připojení, nebyl by problém aplikaci rozšířit.



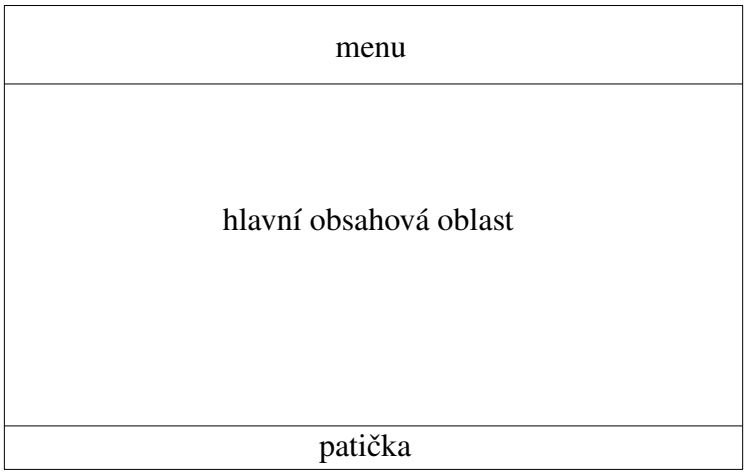
Ilustrace 12: Model tříd, zabezpečení

4.8 Grafický návrh

Grafické rozvržení aplikace by mělo být co nejjednodušší na používání. Proto jsem zvolil návrh jednoduchého rozvržení se třemi sekcemi, horní oblast pro menu, prostřední pro obsah a spodní oblast pro patičku (viz Ilustrace 13).

Hlavní obsahová oblast by měla být taktéž jednoduchá, ale také konzistentní napříč aplikací. Jelikož jsou všechna data v aplikaci tabulkového charakteru, je vhodné je zobrazit tabulkově za pomocí nějaké dostupné knihovny. K dispozici je několik knihoven pro framework Nette, ale pouze knihovna NiftyGrid splňovala mé požadavky. Těmi byla podpora databázové vrstvy Nette Database, napojení na nějaký grafický framework a jednoduché použití.

Grafickým frameworkem je myšlena například použitá knihovna jQuery UI, díky které se dá jednodušeji docílit jednotného vzhledu celé aplikace od tlačítek, přes formuláře až po tabulky. Za pomocí této knihovny pak není pro člověka, který není zběhlý v kódování a designu, problém vytvořit jednoduchý a přehledný vzhled.



Ilustrace 13: Návrh grafického rozložení

5 Výsledná aplikace

Výsledná aplikace je vzhledově minimalistická, jednotná a tedy dostatečně přehledná. Je toho docíleno především použitím zmiňované knihovny jQuery UI[12], ale také rozdělením do sekcí a vytvořením jednoduchého rozcestníku.

5.1 Vzhled

Díky tomu, že je aplikace jednotná, není potřeba podrobně popisovat vzhled jednotlivých podstránek, ale postačí zobrazení a popis jedné z tabulek a jednoho formuláře.

Tabulka

Vybral jsem tabulku v podstatě nejdůležitější a obsahově nejodlišnější. Jedná se o tabulku pravidel nějakého vybraného firewallu. Pravidla jsou samozřejmě smyšlená, ale ukazují možnosti zobrazení. Pravidla jsou samozřejmě seřazena podle indexu v prvním sloupci. Dle něj se pak budou na daném firewallu aplikovat.

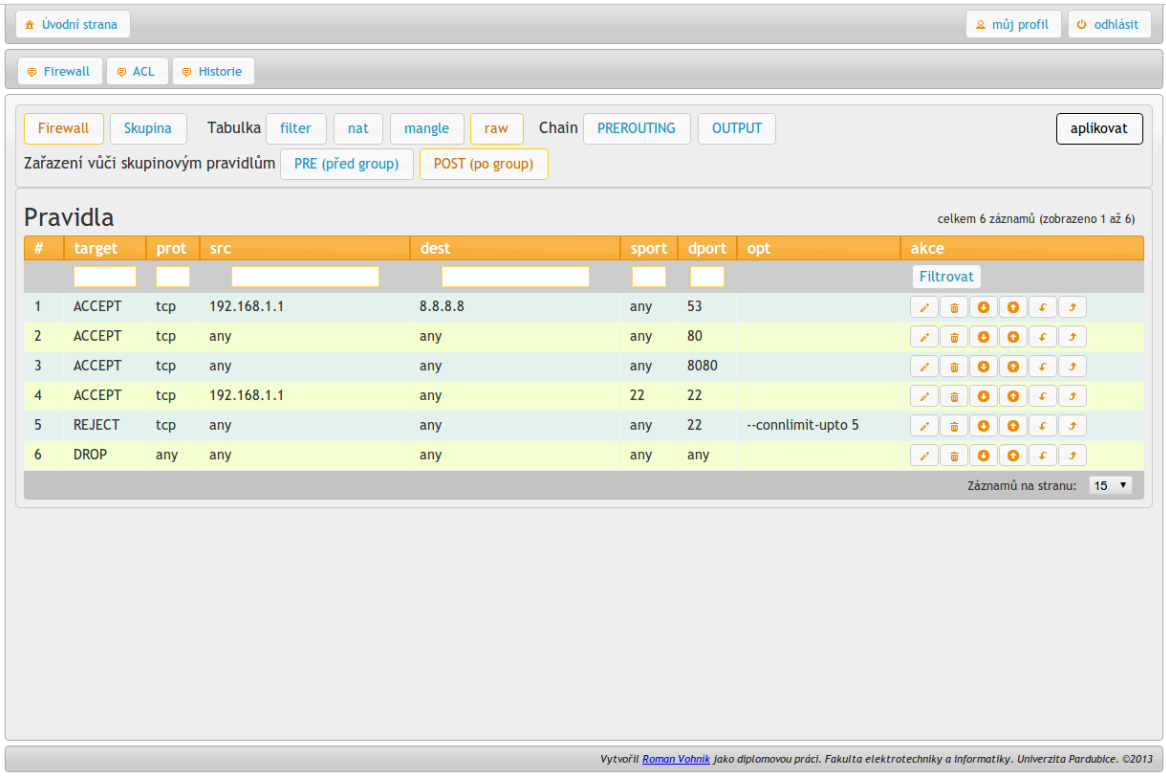
Další sloupce jsou obsahové a zobrazují to nejdůležitější z pravidel. Tím je protokol, zdrojový a cílový port, zdrojová a cílová adresa. Poslední obsahový sloupec s názvem opt obsahuje část volitelnou a pokud by byla delší než šířka sloupce, zobrazila by se zkráceně. To je z důvodu, že tato část pravidla může být ojediněle velice dlouhá, naopak často bude prázdná.

Pak už následuje sloupec akce, kde jsou tlačítka pro provádění akcí s daným řádkovým záznamem. V tomto případě je možno jej editovat, odstranit, posunout nahoru a dolu, případně přidat další pravidlo pod, nebo nad něj. Jelikož jsem předpokládal, že prohazování pravidel bude časté, implementoval jsem ajaxové posouvání myší stylem drag-and-drop, neboli chyť a pusť. To velice zjednoduší tuto operaci.

V této tabulce to není patrné, ale data u jiných tabulek je potřeba třídit a filtrovat dle jiných sloupců než je pořadí. Proto jsou pak tyto sloupce opatřeny polem pro zadání filtru a ikonkami šipek pro řazení. V tomto případě taková funkcionalita chybí, protože je zde zásadní pozice pravidla mezi ostatními a řazení, či filtrování by mohlo způsobovat chyby v úsudku administrátora.

12 Knihovna je volně dostupná z adresy http://jqueryui.com/

Každá tabulka obsahuje své zápatí, kde se zobrazuje nastavení počtu záznamů na jednu stranu. V případě, že by se záznamy na jednu stranu nevešly, objeví se tlačítka pro stránkování mezi jednotlivými stránkami.



Ilustrace 14: Výsledná aplikace – tabulka pravidel

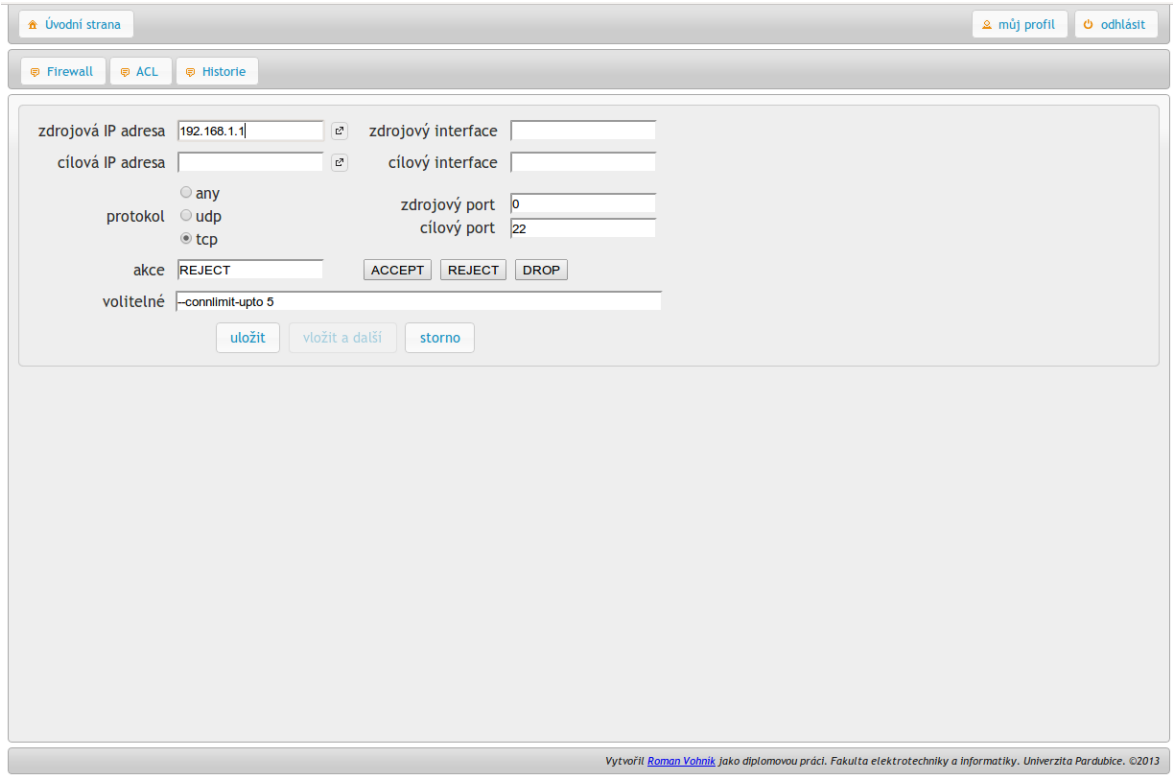
Doposud popsané vlastnosti se nachází u všech tabulek, ale záhlaví tabulky na obrázku je obsaženo pouze u tabulky pravidel. V tomto záhlaví se nachází několik tlačítek, které na první pohled vypadají chaoticky. Jejich funkce bude popsána mezi klíčovými funkcemi.

Formulář

Na dalším obrázku je vidět formulář pro pravidla firewallu. Jedná se v podstatě o informace zobrazené v předešlé tabulce. Jsou zde ale pomocné prvky, které umožňují využít při vyplňování automatického doplnění po vybrání myší. Jedná se především o tlačítka akce a o vyskakovací okénko s dostupnými rozhraními daného firewallu.

Tento stejný formulář je využíván jak pro vytváření tak pro úpravu pravidel, stejně tomu tak je u ostatních formulářů. Navíc je zde tlačítko pro vložení aktuálního záznamu a pokračování novým záznamem. Toto tlačítko je na obrázku zašedlé z důvodu toho, že obrázek pochází ze stránky pro úpravu pravidla, kde tato funkcionalita není možná.

Pro zvýšení komfortu při práci s formuláři jsou některé formulářové prvky opatřeny validačními pravidly. Například prvky vedle kterých se nachází hvězdička, jsou povinné. Klasické zpracování formuláře by to odhalilo až při odeslání, proto by byl formulář navrácen



Ilustrace 15: Výsledná aplikace – formulář pro pravidla

s chybou. Oproti tomu je zde využito takzvané živé validace, která kontroluje prvky dle nastavených pravidel neustále během vyplňování. Proto vůbec nedovolí formulář odeslat, pokud je nějaký prvek špatně vyplněn.

5.2 Použití a klíčové funkce

Základ použití byl nastíněn v předešlé kapitole. V podstatě všechny podstránky obsahují svou tabulku pro výpis dat v přehledné formě a formulář pro vkládání nového záznamu a editaci stávajících. Jednotlivé tabulky jsou pak provázány buďto přes základní menu, nebo přímo přes řádkové akce v tabulce. Tím se jednoduše dostaneme k požadovaným datům nějakým způsobem závislých na daném řádku. Příkladem může být tabulka firewallů, která má jako řádkovou akci odkaz na pravidla.

Ve vrchním prostoru se nachází tlačítko pro návrat na úvodní stranu a tlačítko s přihlášením, nebo odhlášením. Pokud jsme přihlášeni, máme ještě možnost vejít do podstránky s profilem, kde je možno jej upravit a změnit si heslo. To je důležité, protože aplikace nedisponuje registračním formulářem, proto uživatelské účty vytváří administrátor. Ten nově vytvořenému účtu nastaví nějaké heslo a účet předá novému uživateli, který si heslo změní. Aplikace samozřejmě obsahuje ošetření, aby nedošlo k odstranění posledního účtu, který má právo vytvářet další účty. Tím by se uzavřel jakýkoli legitimní přístup do aplikace. Jediným řešením by byl přímý zásah do databáze a ruční vložení zašifrovaného hesla, které známe. To je ale nepřípustné, proto je toto ošetřeno.

Pokud nejsme přihlášeni, zobrazí se nám pouze přihlašovací formulář se jménem a heslem. Ve formuláři je implementována ochrana proti hádání hesel. Počítá se totiž počet špatných přihlášení za posledních 5 minut. Pokud je tento počet větší než pět, skript se na jednu minutu uspí.

Pokud se úspěšně přihlásíme, zobrazí se již panel s rozcestníkem, který obsahuje pouze tři položky. Ty se ale rozvinou při najetí myši o odkazy do dané sekce. Například poslední položkou se dostaneme do sekce historie přihlášení, kde je možné sledovat jak platná, tak chybná přihlášení.

Přístup do této sekce, stejně jako do ostatních je možné nastavit v sekci ACL, která je dostupná uprostřed rozcestníku. Tato sekce obsahuje jednotlivé podstránky pro správu uživatel, skupin uživatel, zdrojů a oprávnění k těmto zdrojům. Tuto sekci může v základu spravovat pouze hlavní administrátor, ale díky samotnému ACL není problém nastavit přístup dalším skupinám. Dnes mají běžně webové aplikace formulář pro obnovu zapomenutého hesla, ten tu ale nenalezneme. Zapomenuté heslo může přepsat opět jen hlavní administrátor na podstránce s uživatelskými účty pomocí řádkové akce.

Zbývá už jen třetí položka menu a tou je samotný modul pro správu firewallů. Předně je zde podstránka se správou skupin firewallů, kde mohou být vyplněny například školní učebny apod. Tato tabulka má v horní liště navíc záložku Strom, která slouží pro stromové zobrazení skupin. Můžeme se tak jednoduše přepínat mezi tabulkovým a stromovým zobrazením stejných dat. Ze skupiny se pomocí řádkové akce můžeme dostat na tabulku s přiřazenými firewally. Tabulka skupin má navíc ještě možnost dynamicky ajaxově rozevřít danou skupinu a přímo uvnitř řádku celkem přehledně zobrazit vnořenou tabulku s firewally, které do této skupiny náleží.

Tu samou vlastnost má také druhá podstránka s firewally, kde si takto můžeme zobrazit tabulku přiřazených rozhraní. Zde můžeme firewally klasicky editovat, vkládat nové, odstraňovat, případně přejít na podstránku s přiřazenými pravidly. Pokud je firewall nově vložený, je u něj tlačítko pro nahrání veřejného klíče. To uživatele zavede na stránku s jednoduchým formulářem, který očekává vyplnění přístupového jména a hesla k ssh účtu daného firewallu.

Po zadání se pokusí spojit pomocí ssh protokolu a zadaných údajů k cílovému stroji. Pokud se spojení podaří, tak si aplikace uloží do databáze otisk daného hosta. Při každém dalším spojení je tento otisk znovu vyžádán a zkontrolován s uloženým. Pokud nesouhlasí, je spojení odmítnuto s chybovou hláškou. To se může stát jednak rozdílnou konfigurací hosta, ale pokud si toho není správce vědom, může se jednat o MITM útok.

V dalším kroku je pak přenesen veřejný klíč serveru na firewall. Každé další připojení už nebude potřebovat zadání uživatelského jména, ani hesla. Pro přenos klíče má PHP nativní knihovna funkce *ssh2_publickey_init()* a *ssh2_publickey_add()*. Po veškerých snahách tyto funkce s cílovými systémy nefungovaly, proto jsem vyřešil přenos klíče za pomocí sftp.

Nyní se už můžeme dostat ke správě pravidel z minulé kapitoly. Základní princip byl vysvětlen, ale zbývá vysvětlit horní panel. Ten obsahuje přepínání mezi pravidly daného firewallu a mezi pravidly skupiny, ve které se firewall nachází. Pravidla mohou být totiž přiřazována jak firewallu, tak společná pravidla skupině. Někdy ale může být užitečné v pravidlech firewallu přepsat pravidlo skupiny. Jelikož se pravidla zpracovávají postupně, musí tedy existovat možnost jak některé pravidlo firewallu předřadit všem pravidlům skupiny. Na to existují ve stejném panelu přepínací tlačítka PRE a POST. Pokud by tedy ve skupinových pravidlech bylo takové, které zakazuje určitý provoz. A my bychom chtěli u jednoho firewallu výjimku právě pro tento provoz. Proto bychom pravidlo povolující provoz zařadili do skupiny PRE. PRE tedy znamená před a POST znamená zařazení za skupinová pravidla. Pro zobrazení všech pravidel v řetězci tak, jak budou aplikována slouží tlačítko VŠE. Tím dostaneme pravidla seřazená, nejdříve PRE pravidla, pravidla skupiny a nakonec POST pravidla. Pak jsou zde tlačítka na přepínání skupin tabulky a řetězce. Jsou to všechny možnosti netfilter firewallu a lze tedy nastavit jakoukoli přípustnou kombinaci.

Zbývá už jen tlačítko s názvem aplikovat. Po stisku se nejdříve pravidla sestaví a následně se aplikují na vzdáleném stroji. Sestavení pravidel probíhá tak, že se postupně prochází celá databáze pravidel nastavených pro firewall i pro skupinu a vypisují se do souboru v dočasném adresáři. Tento soubor má svou specifickou syntaxi, kterou je nutné dodržet a samozřejmě

musí být pravidla ve správném pořadí. Soubor je na konci sestavení ve formátu, v jakém by byl při použití příkazu *iptables-save* na běžícím nastaveném firewallu.

Samotné aplikování pravidel pak probíhá ve třech krocích. Nejprve je potřeba daný soubor přenést na cílový stroj. To se provede přesně jako přenos veřejného klíče při prvním spojení. Nyní aplikace vzdáleně provede zálohu stávajících pravidel pomocí příkazu *iptables-save* do vlastního souboru s názvem obsahujícím aktuální datum. Ve třetím kroku už může dojít k aplikaci nových pravidel. To by bylo možné provést příkazem *iptables-restore*.

Pokud by ale došlo k zablokování přístupu našimi novými pravidly, nebyli bychom schopni vzdáleně pravidla opravit. Proto existuje skript *iptables-apply*, kterému předáme náš soubor s pravidly a číslo reprezentující počet sekund. Tento skript pravidla aplikuje a potom na textovém vstupu očekává potvrzení, zda jsou pravidla v pořádku. Pokud v námi stanoveném limitu odpovíme, je jasné, že jsme si přístup nezablokovali. V opačném případě se po limitu opět provede navrácen původních pravidel a my můžeme ta nová opravit.

Dodatečnou funkčností, která může být správci využita, je možnost zobrazení logů vzdálených stanic, kam vypisuje svůj stav i samotný netfilter firewall. Stejně jako výpisu aplikace netstat, která zobrazí aktuální síťová spojení a mnoho dalších informací. To jsou dvě vlastnosti, které se při správě firewallu hodí, proto jsem je zde zahrnul.

Závěr

V teoretické části se povedlo dle zadání shrnout základy síťové bezpečnosti a firewallů. Základy proto, že je toto téma nesmírně obsáhlé a vydalo by na nejednu celou práci. Praktickým cílem však bylo vytvoření nástroje pro centrální správu linuxových firewallů. Nejdříve jsem tedy provedl průzkum a popis existujících řešení, které bohužel nevyhovovaly požadavkům. Taktéž jejich úprava by byla složitější než vytvoření nové aplikace plně dle představ a požadavků.

Proběhl tedy postupný návrh architektury, výběr jednotlivých technologií a návrh celé aplikace od Use Case diagramu, přes diagramy tříd a databáze až po design. Jelikož byla aplikace vytvářena jedním člověkem, mohlo by se zdát, že byl takový návrh časově stejně náročný jako samotné programování. Opak je ale pravdou a z vlastní zkušenosti bylo poznat, že kvalitní návrh ušetří ve výsledku čas a nejspíše i zvýší výslednou kvalitu aplikace.

Výsledkem je tedy webová aplikace pro centrální správu linuxových paketových filtrů, která je dle mého názoru dobře použitelná díky jednoduchosti a přehlednosti. Může sloužit všude, kde vyvstává potřeba spravovat firewall na více vzdálených strojích. Hlavním benefitem je pak možnost zařazování stanic do skupin a nastavování skupinových pravidel. Tato výhoda se projeví, pokud potřebujeme skupině počítačů nastavit stejná pravidla a jen u některých provést drobné změny. Aplikace by mohla najít uplatnění například v linuxových učebnách, domácnostech s větším počtem počítačů, nebo i menších firmách a podobně.

Základní pojmy a zkratky

autentizace – jednoznačná identifikace uživatele

autorizace – ověření oprávnění uživatel

VPN – virtuální privátní síť, neboli šifrovaný tunel mezi sítěmi

GNU GPL – nejběžnější licence pro svobodný software

MITM – man-in-the-middle útok – jde o útok, kdy je útočník mezi námi a nějakým cílovým zdrojem, ke kterému se připojujeme

AJAX/AJAJ – asynchronní javascriptové vyvolání požadavku k webovému serveru, které vrací data ve formátu XML/JSON a nevynutí překreslení celé stránky.

JSON – formát dat, který má oproti XML daleko jednodušší syntaxi, proto je jeho použití vhodné pro struktury posílané prohlížeči

ACL – seznam pro řízení přístupu

Seznam ilustrací

Použitá literatura a cizí zdroje

[1] SOSINSKY, Barrie. *Mistrovství – počítačové sítě*. Vyd. 1. Brno: Computer Press, 2010, 840 s. Mistrovství (Computer Press). ISBN 978-80-251-3363-7.

[2] HATCH, Brian. *Hacking bez tajemství LINUX*. Vyd. 1. Brno: Computer Press, 2003, 644 s. ISBN 80-722-6869-4.

[3] Denial-of-service attack. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-08-07]. Dostupné z: http://en.wikipedia.org/wiki/Denial-of-service_attack

[4] BARRETT, Daniel J. a Richard E. SILVERMAN. *SSH: Kompletní průvodce*. 1. vyd. Brno: Computer Press, 2003, 556 s. ISBN 80-722-6852-X.

[5] DOSTÁLEK, Libor. *Velký průvodce protokoly TCP/IP: bezpečnost*. 2. aktualiz. vyd. Praha: Computer Press, 2003, xvi, 571 s. ISBN 80-722-6849-X.

[6] STREBE, Matthew a Charles PERKINS. *Firewally a proxy-servery*. Vyd. 1. Brno: Computer Press, 2003, xxi, 450 s. ISBN 80-722-6983-6.

[7] BROWN, Martin A. History of Linux Packet Filter Support. In: *Guide to IP Layer Network Administration with Linux* [online]. 2007 [cit. 2013-08-07]. Dostupné z: http://linux-ip.net/html/pf-intro.html

[8] GRANDES, Guillermo. Diagrama Netfilter Linux. 2004 [cit. 2013-08-07]. In: *Wikipedia: the free encyclopedia* [online]. Dostupné z: http://commons.wikimedia.org/wiki/File:Diagrama_linux_netfilter_iptables.gif

[9] NETTE FOUNDATION. *Nette Framework* [online]. © 2008-2013 [cit. 2013-08-07]. Dostupné z: http://nette.org/

[10] MySQL 5.5 Reference Manual. ORACLE CORPORATION AND/OR ITS AFFILIATES. MySQL: The world's most popular open source database [online]. 2013 [cit. 2013-08-07]. Dostupné z: http://dev.mysql.com/doc/refman/5.5/en/index.html

[11] Jak vytvořit žádost o SSL certifikát. ACTIVE 24 , s.r.o. *Active 24: Certifikační autorita* [online]. 2012 [cit. 2013-08-07]. Dostupné z: https://www.ca.cz/jak-to-funguje/navod/

[12] SSH. Wiki Ubuntu [online]. 2012 [cit. 2013-08-07]. Dostupné z: http://wiki.ubuntu.cz/ssh

Příloha – obsah CD

- tato práce ve formátu PDF
- adresář s názvem *bouml* obsahující UML diagramy ve formátu projektu pro aplikaci BOUML a vygenerované obrázky diagramů ve formátu PNG
- zdrojové kódy aplikace spolu se závislými knihovnami, které je dle licencí možné dále bezplatně šířit, to vše v adresáři *centralni_sprava_aplikace*
- SQL model pro aplikaci MySQL Workbench a vyexportovaný SQL skript v adresáři *mysq_model*