Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Problematika a optimalizace firewall nad OS Linux

Diplomová práce

Autor: Bc. Petr Halický
Studijní obor: AI2

Vedoucí práce: Mgr. Josef Horálek

Hradec Králové duben 2015

Prohlášeni:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne Jméno a příjmení

Poděkování:

Chtěl bych na tomto místě poděkovat svému vedoucímu práce, Mgr. Josefu Horálkovi, za jeho vedení, náměty a rady, které velkou měrou ovlivnily konečnou podobu diplomové práce.

Anotace

V rámci této diplomové práce jsou představeny principy ochrany počítačového systému pomocí firewallové aplikace. Následně je v praktické části navržen a vyvinut vlastní nástroj založený na automatické tvorbě pravidel dle výsledků analýzy síťové komunikace.

Annotation

Title: Problems and optimization of firewall Linux OS

As part of this Diploma Thesis presents the principles of protecting a computer system using firewall applications. Subsequently, the practical is designed and developed their own tool based on the automatic creation of rules according to the results of analysis of network communication.

Obsah

Seznam obrázků

Úvod

Bezpečnost. Soukromí. Ochrana dat. Stále častěji diskutovaná a probíraná témata. Jak lze v dnešní době uchovat jakékoliv údaje a data udržovaná v digitální podobě v bezpečí? Útoky zaměřené na samotné počítače či přenosové uzly v rámci síťové hierarchie mohou leckdy vyústit v obrovské škody i na samotných zařízeních, o vlastních datech nemluvě. I v případě, kdy jsou všechna důležitá data řádně zálohována, může dojít ke ztrátě čerstvě uložených informací a nebo zbytečné prodlevě kvůli nutnosti obnovit normální chod. Navíc je nutné podotknout, že ani zálohovaná data nemusí být v případě ztráty či odcizení řešením problému.

Většině problémů se však dá velice lehce předcházet. V dnešní době, kdy výkon jediného jádra procesoru dokáže mnohdy pokrýt potřeby celého systému, není přece problém obětovat jistou část výpočetní kapacity pro zajištění ochrany a bezpečnosti dat. Různá antivirová řešení už v mnohých situacích dokázala, že jejich samotná přítomnost může odhalit určitá nebezpečí a předejít i vážnějším dopadům těchto hrozeb. Navíc díky masivnímu rozšíření a přítomnosti internetového připojení není ani velkým problémem udržovat takové řešení v aktuálním stavu. Druhou možností, pokud skutečně nelze obětovat procenta výkonu lokálního stroje, je spolehnout se na externí virové skenery.

Ačkoli přínos antivirových aplikací nelze popřít, dokážou hrozbu zachytit až v momentě, kdy vstoupí do lokálního stroje. Navíc jen zlomek těchto programů dokáže reagovat i na jiné, než jen čistě virové hrozby. Nejen z těchto důvodů je tak stále vhodnější kombinovat jejich sílu s dalším ochranným opatřením a sice použitím firewallů.

Firewall představuje první linii ochrany počítače proti hrozbám přicházejících z vnějšího prostředí. Při umístění v síti potom lze ochránit hned několik lokálních strojů zároveň.

Správně fungující hráz dokáže aktivně filtrovat množství průchozí komunikace a tím snížit její množství ke kontrole samotným antivirovým řešením. V důsledku tak nejen, že dokáže zvýšit ochranu počítače, protože se může zaměřit i na hrozby, které by jinak prošly bez povšimnutí, ale dokáže v konečném důsledku ušetřit i čas vynaložený na nápravu případných komplikací.

Nelze se však spolehnout na jednorázově nastavená pravidla a chování firewallu. Hrozby se vyvíjejí každý den a dříve či později by se objevila taková, která by dokázala nastavení ochranné hráze jendoduše překonat. Vhodné není ani neustálé rozšiřování pravidel, které by vedlo k nadměrnému zatížení. Firewall tedy vyžaduje neustálé kontroly a úpravy, aby se zajistila maximální ochrana.

Vhodné je samozřejmě i využití více jednotlivých principů a návrhů, které by měly být umístěny v kaskádovité struktuře v pořadí, ve kterém bude docházet k největšímu omezení nežádoucí komunikace a tím i snížení nároků hlubších kontrol. Ani tento přístup by se však neobešel bez neustálé kontroly a úprav.

Určitým řešením by mohla být firewallová aplikace, která by se dokázala přizpůsobovat změnám v komunikaci, aby se zajistila ochrana i proti nově se objevivším hrozbám.

1 Rešerše dostupných řešení

Vzhledem k zaměření této práce byl proveden průzkum dostupných řešení, popřípadě různých studií, které popisují možná či vhodná opatření pro zajištění maximální bezpečnosti s využitím firewallové aplikace.

Firewall slouží především jako hráz proti neoprávněnému přístupu z vnější sítě do vnitřní. Podle jeho umístění a specifikací lze očekávat i určité problémy s jeho správou a údržbou. V dnešní době již jen zřídka stačí provést úvodní nastavení. Tento způsob dokáže totiž účinně fungovat pouze proti hrozbám, které byly v době nastavování známé a zmapované. Jenomže nové hrozby, či útoky dokážou vznikat prakticky každý den a firewall, který nebude řádně připraven, se stává velice zranitelným.

Stále častěji jsou k průniku či ochromení firewallových hrází používány kombinace několika rozličných metod, proti kterým jsou firewally nezřídka bezbranné. Navzdory tomuto faktu je leckdy i neúnosné používat současně více řešení, která by se navzájem doplňovala, ať už z finančních či výkonnostních důvodů. Současná řešení tedy dokážou účinně zpracovat pouze útoky, které znají a pouze takové, které jsou schopny odrazit. Stále častěji je využíváno útoků DoS (Denial of Service ~ odepření služby), jejichž hlavním cílem je přetížit a zahltit ochranný val, pro vytvoření trhliny, kterou se dále může šířit virus. [1]

Přestože jsou firewallová řešení stále častěji doplňována virovými monitory či systémy pro detekování průniků, jejich samotná přítomnost nestačí. Mnoho typů útoků i přesto vyžaduje pomoc člověka, který by rozhodl o dalším kroku. Určitou naději na změnu přinášejí pokusy o vývoj firewallu založeného na prvcích umělé inteligence, která by byla schopna přizpůsobovat se měnícím podmínkám a tím ochránit síť i proti budoucím typům hrozeb. [1]

Inteligentní firewall by mohl zvýšit bezpečnost vnitřní sítě nejen díky schopnosti se přizpůsobit, ale také pomocí technik pro analýzu celého paketu, hlaviček i datové části a detekčních přístupů pro odhalení možného zneužití komunikace. [1]

Je však důležité kromě ochrany vnitřní sítě, ochránit i samotný firewall. Mnoho typů útoků si za svůj primární cíl neberou chráněné prostředky, ale samotný ochranný mechanismus, který se pokouší vyřadit z činnosti. Již bylo zmíněno využití útoků typu DoS na propašování virové nákazy skrze možné trhliny. Existují však i takové formy útoků, které chtějí ochranný val nejen prorazit, ale i zničit.

Není důležité množství použitých firewallů, důležité je rozeznat dobře fungující a selhávající aplikace. Nestačí se však pouze podívat na současné vytížení zařízení nebo programu, protože se ještě nemusí jednat o selhání, ale o projev úspěšného odrážení útoku. Základem je tedy průzkum a pochopení pravidel, která firewall definují. Nejlepším místem pro začátek takového zkoumání je nejvíce zranitelný prvek v celé topologii. Nabízí se více technik, které umožňují měřit výkon jednotlivých řešení, je však důležité si výsledky i správně vyložit. Nezbytné totiž je soustředit se na správný firewall v ten nejdůležitější čas, aby se zajistila jeho stabilita, přičemž vůbec nezáleží na tom, jak sofistikovaný firewall je. [2]

Stále častěji jsou však vedeny i útoky proti samostatným aplikacím s veřejným webovým rozhraním než jenom proti celým sítím. Takové služby se většinou nemohou

spoléhat na běžná řešení, která jsou uzpůsobena pro komunikace s velkou variací protokolků a služeb. Nezřídka totiž mohou způsobit výrazný pokles výkonu a rychlosti spojení, obzvláště pokud je počítáno s vysokým počtem uživatelů. Navíc se stále častěji prokazuje náchylnost detekcí založených na porovnávání se vzorem k selhání při setkání s komplexním typem útoku nebo dokonce neznámým druhem. [3]

Možným způsobem, jak překlenout nedostatky bežných řešení, je využití nástroje založeného na monitorování webové komunikace. Většina již existujících programů využivá jednoduchého principu porovnávání se vzorem, kdy analyzuje celý požadavek zaslaný na server nebo jen vybrané prvky. Každopádně má však tento přístup svá negativa. Aby bylo možné útok odchytit, musí existovat záznam jeho vzoru a odpovídající pravidla pro jeho zastavení. Z toho plyne velká náchylnost při jednoduchém zastarávání pravidla nebo při setkání s útokem nultého dne (zero-day attacks), proti kterým účinná ochrana není zpravidla dostupná. [3]

Určitou možností může být vyvinutí obdobného mechanismu, který by však odhalování a léčení napadených požadavků prováděl na jiném principu než porovnávání se vzorem. Takový přístup podle [3] je založen na vytvoření a správě profilů známých požadavků přicházejících na server. Tyto jednotlivé profily jsou následně využity pro léčení podezřelé komunikace. Velkou výhodou je skutečnost, že nedochází k prostému zahození požadavku, ale opravě jeho obsahu, čímž se relativně snižuje míra chybného označení. [3]

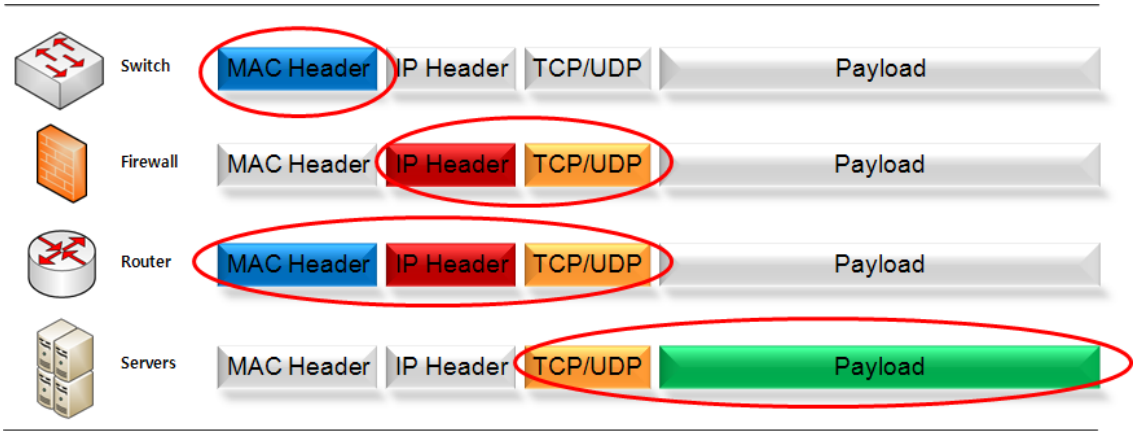
Zvláště u rozsáhlých sítí s komplexním nastavením firewallu je velice obtížné udržovat všechna pravidla aktuální. I jediné zapomenuté pravidlo může v konečném důsledku znamenat obrovskou hrozbu a potencionální průlom této ochranné hráze. S nárustem komunikace, která daným rozhraním protéká, a zvyšováním počtu kontrolních pravidel, může docházet i k neúnosnému zpomalení toku vinou probíhající analýzy. Není zajisté nutné dodávat, že každým rozšířením chráněné zóny pravděpodobně dojde i k nárustu pravidel, z nichž se velké množství může navzájem rušit či překrývat. [4]

Vhodným uspořádáním nebo vyfiltrováním aktivních pravidel pro firewallovou analýzu se tak může ušetřit nejen výpočetní čas a výkon, ale i předejít budoucím komplikacím s údržbou či aktualizací ochrany. Důležité je tedy pouze umět rozhodnout, která pravidla umístit do jaké pozice vzhledem k řetězci pro filtrování, a která je vhodnější zcela vypustit. K tomuto účelu lze využít statistická data průchozí komunikace, která poslouží nejen k přizpůsobení se konkrétní síti, ale umožní i provádět dynamické řazení existujcích pravidel. [4]

Velice důležitou vlastností je i zaměření firewallové ochrany. Mnoho bezpečnostních hrází pracuje pouze na omezeném prostoru a vytváří tak příležitost pro škodlivé útoky, které jednoduše uniknou aktivní kontrole. Této skutečnosti využívají stále častěji útoky založené na virech a počítačových červech. [5]

Moderní inteligentní systémy používané jako protiopatření či prevence útokům a průnikům do interní sítě jsou stále robustnější a obsahují sofistikovanější metody pro kontrolu aktivní komunikace. Jejich metody jsou ve většině případů spojené s hledáním vazeb mezi zachycenými nebo kontrolovanými daty a bezpečnostními událostmi, které vyvolají. Existující bezpečnostní metody se ukazují jako náchylné vůči úzce zaměřeným

kyber útokům, což jenom posiluje potřeby odolávat těmto pokusům pomocí hledaní vazeb a nikoli pouze hrubou blokací. Dalším rozšířením může být nahrazení již dnes ne zcela nepřekonatelného principu porovnávání vzorů metodou, která by byla schopna odhalit i neznámé druhy útoků či virů. [5]



Obrázek 1 - Využití paketu síťovým hardwarem [5]

Každý paket, který je zpracován ať už firewallovem nebo odpovídajícím síťovým zařízením, je vždy složen ze dvou hlavních částí a to oblasti dat a hlaviček, které představují popis a určení původu a cíle paketu. Různá zařízení pracují s danými oblastmi informací, která jsou v daném okamžiku důležitá, viz. Obrázek 1. Aby byla zajištěna maximální bezpečnost, je nezbytné nejen rozšířit firewallem zpracovávanou oblast, která je podrobena analýze a testování, ale zaručit i minimální latenci při této činnosti, aby nedocházelo k výrazným poklesům přenosové rychlosti síťového rozhraní. [5]

Pro zajištění bezpečnosti lze samozřejmě kromě různých forem aplikačního softwaru využít i specializovaný síťový hradware. Výhodou takového přístupu je obecně snížení režie na chráněných počítačích, protože analýza a kontrola probíhá na samostatném hardwaru. Jak však laboratoře NSS v rámci testování předních modelů odhalily, ani takové řešení nepředstavuje záruku bezpečnosti. Pět z testovaných zařízení se ukázalo náchylných k chybě v rámci jejich základní konfigurace. Přestože lze tento problém jednoduše odstranit, taková oprava může mít negativní dopad na celkovou výkonnost zařízení. [6]

2 Představení principů firewall na OS Linux

V původním významu označuje slovo "firewall" termín z architektury a sice "ohni odolnou stěnu", která má zabránit šíření destruktivního živlu a zvětšování již napáchaných škod.

V případě digitálních technologií je význam jiný, ale účel zůstává. Firewall představuje první linii ochrany počítače, popřípadě místní sítě, před případnými hrozbami. Nebudeme-li rozlišovat podle typů, představuje firewall sadu pravidel, podle kterých je posuzována příchozí, volitelně i odchozí, komunikace skrze síťová rozhraní počítače. Výsledkem je buď povolení nebo zamítnutí daného spojení. [7]

Základem dobře fungujícího firewallového řešení jsou tedy správně nastavená pravidla pro kontrolu probíhajících nebo navazovaných spojení. Existují různá doporučení a pravidla, jak správně postupovat. Ovšem i ten nejjednodušší návod vyžaduje určité znalosti pro jeho aplikování. Musíme však počítat s tím, že i tyto návody časem zastarají a bude potřeba jejich nahrazení.

Udržování bezpečného počítače není jednorázová činnost. Nové hrozby vznikají s každým novým programem, webovou službou či stránkou. I dobře nastavený firewall bude vyžadovat úpravy a doplnění, aby mohl hrozby včas zachytit a zabránit tak škodám.

Určitou variantou jsou různá automatická řešení, která se spoléhají na data sesbíraná z nejrůznějších systémů a určená pro tvorbu nových sad pravidel distribuovaných skrze aktualizace. Takovéto programy však často nedokážou zastavit nejnovější hrozby, dokud nebudou dostatečně zmapovány a nové postřehy implementovány do další várky aktualizací.

2.1 Typy firewallů

Ačkoli se může zdát zbytečné dělit firewallová řešení, mezi jednotlivými druhy implementací bývají podstatné rozdíly. Za jednu z implementačních kategorií lze považovat umístění firewallu vzhledem k chráněným objektům.

Jako základní rozdělení lze chápat firewall jako program, tedy softwarovou implementaci, která je povětšinou určena k ochraně konkrétního počítače na němž se vyskytuje. Výhodou je stejná úroveň zabezpečení proti hrozbám přicházejícím z vnější a vnitřní sítě. Samotný proces analýzy komunikace však může způsobit zpomalení počítače, což lze v některých situacích považovat za nežádoucí.

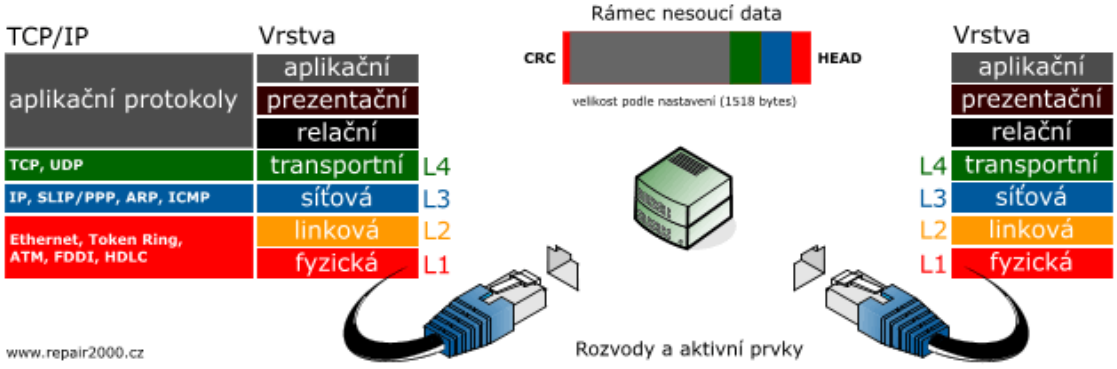
Protikladem je specializovaný hardware, který se zpravidla umisťuje v hierarchii počítačové sítě před chráněnou zónu. Takovéto řešení bývá zpravidla výkonnější, protože je schopno ochránit mnohem širší počet strojů před hrozbami vznikajícími mimo chráněnou síť a současně nedochází k čerpání prostředků takto chráněných zařízení. Nicméně už nedokáže efektivně chránit jednotlivé počítače, pokud se původce hrozby, například infikovaný počítač, dostane do oblasti za firewallem.

Dále lze firewally dělit podle principu jejich funkce nezávisle na pozici. Stejná funkcionalita může být prezentována jak na lokálním softwarovém řešení, tak na specializovaném hardwarovém vybavení.

Podle umístění a typu firewallové aplikace se odvíjí i způsob jeho kontroly a nastavení. Velké množství aplikací je založeno na konsolovém principu. Výhodou je určitá univerzálnost a samozřejmě nižší nároky samotné aplikace. Druhou možnost představuje grafické uživatelské rozhraní, které většinou nabízí ucelené ovládání celého programu. Samozřejmě je možné sehnat i různé nadstavby, které jednu či druhou variantu dokážou rozšířit či upravit směrem k zbývající možnosti.

2.1.1 Paket

Než budou popsány konkrétní modely firewallových aplikací, je vhodné představit prostředí, v kterém všechny tyto programy pracují. Řeč je samozřejmě o prostředí síťové komunikace, která je často zobrazována ve formě referenčního modelu ISO/OSI. Jedná se o příklad řešení komunikace v rámci počítačových sítí za využití sedmi jasně separovaných vrstev s definovanými odpovědnostmi. Tento přístup přináší možnost jednoduché výměny prvků v rámci jedné z těchto vrstev a současně ponechat ostatní beze změn. V praxi je však častěji využíván upravený model TCP/IP, který se skládá ze čtyř separovaných vrstev.



Obrázek 2 - ISO/OSI model [8]

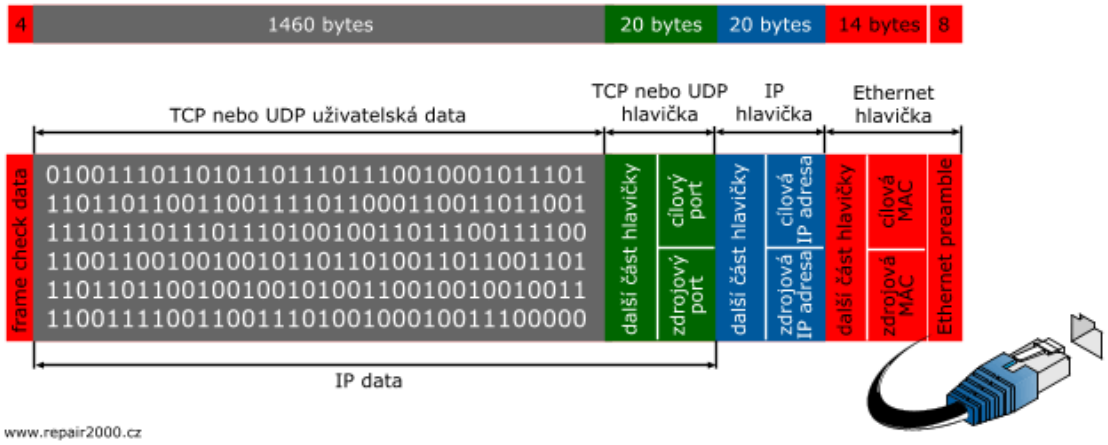
Každá jednotlivá vrstva slouží specifickému účelu a je představována i vlastními protokoly či požadavky. Jednotlivé úrovně jsou tvořeny přístupovými body, které zajišťují komunikaci s okolními vrstvami a předávání dat. Data určená k odeslání vždy postupují od nejvyšší úrovně a jsou postupně zabalovány, označovány a řazeny v rámci dané vrstvy a následně předány do nižší. Jakmile dojde k dosažení poslední úrovně modelu, jsou data přepisována na fyzické médium, reprezentované formou kabeláže popřípadě bezdrátovým standardem.

Kromě prostředí je důležité i s jakým elementem firewally dokážou pracovat. Ve většině případů jsou zaměřeny na jednotlivé pakety, které nesou samotná uživatelská data. Základní struktura každého paketu je identická.

Paket je podobně jako síťový model tvořen několika vrstvami, které odpovídají úrovním referenčního modelu a reprezentují zpracování dat při průchodu od vytvoření po odeslání na nejnižší hladině.

Každý paket je na základní úrovni tvořen dvojicí vrstev, řídícími a uživatelskými daty. Druhá jmenovaná představuje důvod existence komunikačního paketu, zatímco první umožňuje doručení do cílové destinace. Poskytuje totiž všem zařízením v síti, kterými v průběhu přenosu prochází, údaje pro další směrování a samozřejmě i kontrolu

samotného paketu. V závislosti na použitém přenosovém protokolu je potom možné informovat vysílací stranu o chybě či porušení struktury paketu. Pozice řídích dat je obvykle uvnitř hlavičky dané úrovně a na jejím konci. Uprostřed se nachází přenášená data, popřípadě data obalená hlavičkou vyšší vrstvy modelu. V případě přijetí a zpracování paketu dochází k postupnému odebírání jednotlivých hlaviček až jsou zpětně získána původní data.



Obrázek 3 - Struktura paketu [8]

Přenos jednotlivých paketů může probíhat pomocí jednoho ze dvou hlavních přenosových protokolů. Hlavní rozdíl mezi oběma protokoly je přístup k chybě při přenosu. Spolehlivý přenos dokáže oznámit chybu nebo selhání přenosu konkrétního paketu, popřípadě i jeho vlastní poškození, zatímco v případě nespolehlivého přenosu se tak neděje. Přestože se v obou případech dá hovořit o paketech, při použití nespolehlivého přenosu jsou označovány jako datagramy. [8] [9] [10] [11]

2.1.2 Paketové filtry

Ze své podstaty představují základní myšlenku firewallu, také proto, že se jedná o nejdéle existující typ řešení. Průchozí pakety představující komunikaci se vzdáleným serverem nebo počítačem jsou filtrovány na základě pravidel definovaných na adrese a portu původu a cíle. Jak bylo v předchozí kapitole popsáno, tyto informace nese každý paket v řídící části v jedné z hlaviček.

Mezi hlavní přednosti tohoto typu řešení patří vysoká rychlost zpracování průchozí komunikace. Je to dáno skutečností, že se pouze nahlédne do hlavičky paketu, vezmou se obsažená data a porovnají s řetězci pravidel pro nalezení shody, která by určila další chování. Tento princip však představuje i určitý problém neboť nenabízí možnost kontroly celých spojení a tím nalezení souvislostí mezi jednotlivými průchozími objekty.

Paketové filtry pracují na třetí a čtvrté vrstvě referenčního síťového modelu ISO/OSI, který byl představen v předchozí kapitole. [12] [13]

2.1.3 Aplikační brány

Nedlouho po prvních paketových filtrech se objevil další typ firewallového řešení a sice aplikační brány, někdy jsou též vzhledem k svému pricipu označovány jako proxy

firewally. Na rozdíl od předchozí varianty, která pouze kontroluje komunikace, tento typ se jí přímo účastní.

Firewall vstupuje do komunikačního řetězce jako další zařízení v síti. Veškerá komunikace je v rámci zabezpečení zprostředkována pomocí dvou samostatných linek. První představuje spojení mezi bránou a klientským počítačem. Druhá je potom tvořena přeposláním původního požadavku na cílové umístění, ale v roli klienta vystupuje samotná brána. Zjednodušeně řečeno jedná se o překladatele pro obě strany v rámci dané komunikace.

Použitím aplikační brány se otevírají možnosti pro hlubší kontrolu komunikace. Jelikož brána musí jakýkoli průchozí paket zcela zpracovat a pozměnit pro své potřeby, nabízí se i prostor pro využití dalších ochranných řešení jako jsou například antivirové programy, které tak nemusí zatěžovat klientský stroj, obzvláště pokud by byl firewall umístěn na dedikovaném hardwaru v rámci sítě. Vedlejším efektem překládání komunikace, je úplné skrytí adresy klienta, neboť pro vnější síť vždy jako klient vystupuje samotná brána.

Nevýhodou může být problém s komunikací, kdy je vyžadována po každé aplikaci schopnost vést komunikaci přes proxy rozhraní. Pokud by tato vlastnost nebyla funkční nebo podporována, stává se aplikační brána pevnou zdí. Dalším faktorem je poněkud vyšší náročnost na hardwarové vybavení, s čímž přichází i relativně nižší rychlost zpracování než u předchozího typu, paketového filtru. [12] [13]

2.1.4 Stavové paketové filtry

Jedná se o evoluci původního návrhu paketového filtru. Rozdíl představuje schopnost zapamatovat si stav povolených spojení, která firewallem prochází. Tento fakt umožňuje mnohem rychlejší zpracování paketů, které náleží do již jednou povoleného rámce komunikace, aniž by bylo nutné je znovu kontrolovat. Druhým zjednodušením díky vylepšení je možnost psát pravidla pouze pro odchozí spojení. Filtr pak sám dokáže určit, jestli příchozí pakety patří do odpovědi na odeslanou žádost a podle potřeby je může povolit i když neexistuje explicitní pravidlo, které by toto chování definovalo.

Stavové paketové filtry vychází z paketového filtru a nesou si s sebou tedy i obrovskou rychlost zpracování a díky svým vylepšením i zajištění vyšší bezpečnosti než u jejich předchůdce, aplikační brány jsou však stále považovány za bezpečnější. Zároveň díky možnosti vypustit určitá pravidla, se snížila i pravděpodobnost chybné konfigurace, která by tak mohla otevřít systém a vystavit ho možnému riziku.

Dalším stupněm vývoje tohoto typu firewallu je rozšíření schématu o schopnost hloubkové analýzy či inspekce průchozích paketů. Tento přístup umožňuje detekovat i maskovanou komunikaci či pokus o tunelování povoleného protokolu. Další možností je integrace systémů pro detekci útoků na základě porovnávání s databází signatur nebo otisků, které jsou pro podobné útoky typické.

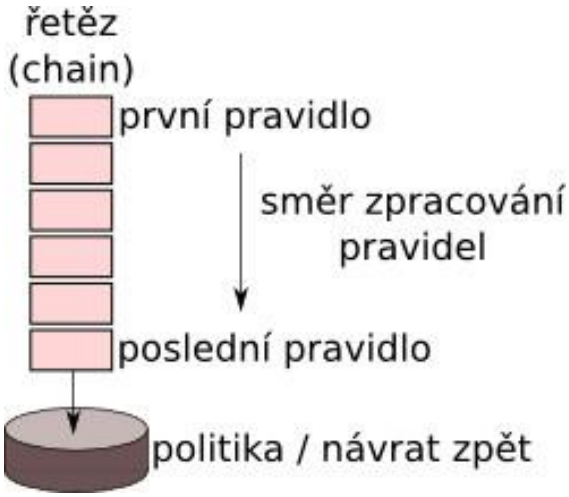
Oproti prostým stavovým paketovým filtrům je s těmito vylepšeními zvýšena bezpečnost, ale většinou za cenu snížení celkové rychlosti oproti prostým stavovým paketovým filtrům.

Takto vylepšený systém však s sebou nese i velké riziko. Jelikož je tvořen mnoha součástmi či moduly, zvyšuje se riziko selhání nebo výskytu kritické chyby, která by systém otevřela útoku. [12] [13]

2.2 Firewall na OS Linux

Prakticky již všechny běžné distribuce operačního systému GNU/Linux přicházejí vybavené firewallovým řešením na základě projektu Netfilter, který lze zcela jednoduše upravovat skrze rozhraní iptables [14].

Dřívější verze systému (do verze jádra 2.4) používaly starší verzi rozhraní iptables zvanou ipchains. Jednotlivá pravidla jsou uspořádána do řetězců, kterými prochází každý paket, dokud není nalezeno odpovídající popisné pravidlo nebo nedosáhne nastavené politiky na konci řetězce.



Obrázek 4 - Schéma funkce řetězu v linuxovém paketovém filtru [15]

Rozhraní iptables a projekt Netfilter je představován celkem pěti samostatnými řetězci, které umožňují, jak filtrování paketů na základě pravidel, tak i operace pro směrování či překlad adres. Pokud má však místní rozhraní fungovat podobně jako router, je třeba takové chování povolit v jádře systému.

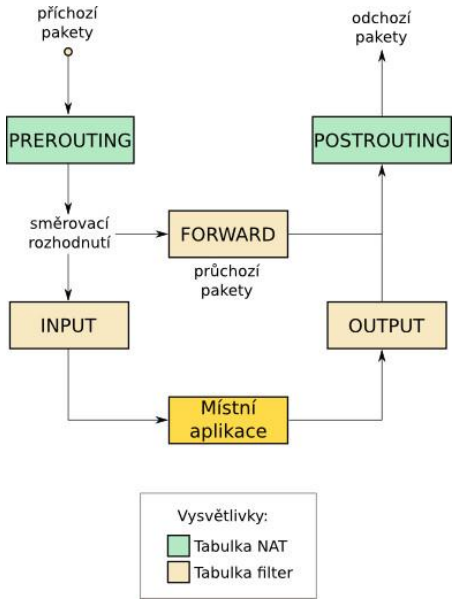
Jednotlivé řetězce lze rozdělit do dvou skupin podle jejich určení. První skupinu představují řetězce typu NAT ("Network Address Translation"), které jsou schopny zajistit úpravy cílové či zdrojové adresy v rámci paketu. Jedná se o dvojice řetězců PREROUTING a POSTROUTING, přičemž v každém lze provádět pouze změnu jedné informace v rámci překladu adres, jak již jejich názvy navozují. PREROUTING se používá pro příchozí pakety a je možné v rámci jeho struktury měnit cílovou adresu. V druhém zmíněném řetězci se potom upravuje zdrojová adresa odchozích paketů.

Další skupinu představuje tabulka pro filtrování příchozí, odchozí a předávané komunikace v případě povoleného směrovacího chování pomocí řetězců INPUT, OUTPUT a FORWARD.

Každý filtrovací řetězec je složen z jednotlivých pravidel, která pro kontrolovaný paket definují určité podmínky pro splnění a případnou akci při shodě. Pokud by bylo zaneseno více pravidel pro zkoumaný paket, je vždy aplikováno to, které se v rámci řetězce nachází výše. Je tedy nanejvýš vhodné věnovat pozornost i pozici jednotlivých pravidel a ne jen jejich obsahu.

Mezi základní akce, které mohou být provedeny v rámci pravidla nebo představovat politiku řetězce, patří ACCEPT (přijetí) a DROP (zahození). Třetí možnost není podporována jako politika řetězce, ale stále představuje volitelnou akci zamítnutí paketu, REJECT.

Pomocí jednoduchých pravidel lze nastavit chování na úrovni nestavového paketového filtru, který byl popsán v kapitole **2.1.2 Paketové filtry**. Kromě konkrétních pravidel je však vhodné definovat i základní politiku řetězce, která se aplikuje na pakety nevyhovující žádnému z obsažených pravidel.



Obrázek 5 - Průchod linuxovým paketovým filtrem [16]

Tímto však možnosti nastavení ani zdaleka nekončí. Využitím dalších možností pro pravidla lze definovat i chování na úrovni stavového paketového filtru, který dokáže rozlišit čtyři různé stavy spojení podle zadaného pravidla. Pokud paket vytváří nové spojení nebo je ve vztahu k již existujícímu, ve kterém však dosud neproběhla obousměrná komunikace, lze stav spojení označit jako NEW. Stavy ESTABLISHED a RELATED popisují situaci, kdy paket náleží do již probíhající komunikace nebo vytváří nové spojení s vazbou k existujícímu. Jako příklad může sloužit otevření datové linky protokolu FTP po navázání řídícího spojení mezi klientem a serverem. Poslední stav popisující pakety bez jakékoli příslušnosti k existujícím spojením se označuje INVALID.

Jelikož se stále rychleji přibližuje doba nutného přechodu ze současného standardu IPv4 na IPv6, je vhodné uvést, že projekt Netfilter je na tento okamžik vcelku dobře připraven. Filtry pro oba standardy jsou však oddělené. Pokud se tedy nastavuje firewall, je nezbytné nastavit chování pro oba samostatné filtry skrze odpovědné rozhraní.

Tato kapitola byla inspirována [15] [16] [17] [18].

3 Návrh řešení - logický model

Navržené řešení spočívá ve vytvoření firewallové aplikace schopné analyzovat průchozí komunikaci a na základě výsledku rozhodnout o tvorbě nových omezujících pravidel. Přestože aplikace by měla být schopna zcela automatického chodu, zůstává potřeba uživatelského rozhraní skrze které by šlo upravovat parametry běhu programu.

Vzhledem k velkému množství úloh, které musí být vykonávány v co nejkratším čase a s minimálními prodlevami, byl zvolen model založený na více vláknovém přístupu. Kvůli zvolenému programovacímu jazyku (objektový jazyk JAVA) je synchronizace vláken nejen poněkud obtížnější, ale ukázala se i nadbytečnou. Většina činností vykonávaných aplikací je tak řízena asynchronně pouze pomocí předávání signálů v podobě datových objektů pro zpracování. Žádné z paralelních vláken není schopno přímého zásahu do objektů, které jsou určeny pro ostatní vlákna.

Aplikace podporuje uživatelské profily, které umožňují proměnlivé přizpůsobení se chování programu. Jednoduchým přepnutím lze bez nutnosti velkého zásahu změnit citlivost na průchozí komunikaci, stejně jako politiku aplikovaných pravidel. Vzhledem k očekávanému množství pravidel spadající pod jediný uživatelský profil, je pro každý účet vytvořen a spravován samostatný soubor v rámci datového prostoru. Současně je možné propojit jednotlivé profily pomocí sdílené analýzy, kdy všechna nová pravidla zachycená pod daným účtem, budou automaticky duplikována i pro takto propojené profily.

V rámci uživatelského rozhraní bude uživatel schopen měnit jednotlivá pravidla, která byla automaticky vygenerována, stejně jako přidávat vlastní. Změny v pravidlech se v takovém případě nebudou promítat do propojených uživatelských profilů. Současně bude možné i editovat seznam portů a přípustných služeb sloužící pro filtrování komunikace.

Aplikace je schopna kontrolovat obsah paketů na všech vrstvách ISO/OSI modelu. Vzhledem k absenci algoritmů pro rozpoznávání závadného obsahu nebo virového skeneru je kontrola samotného obsahu přenášeného paketem limitována na čtení aktualizačních zpráv, které jsou tímto způsobem předávány v rámci místní sítě. Každý průchozí paket je však kontrolován na zdrojovou a cílovou hardwarovou adresu, pro případ podvrženého paketu v případě odchytávané komunikace, síťové adresy a čísla portů ve spojení s transportním protokolem a přidělenou službou.

3.1 Více vláknová aplikace

Moderní počítače a operační systémy jsou navrženy, aby dokázaly spravovat a provádět více různých činností najednou. Se stále rostoucím počtem vícejaderných procesorů a nárustem průměrné velikosti operační paměti, je tato činnost navíc mnohem plynulejší a užitečnější. Není výjimkou použití několika různých aplikací na jednom stroji, které mohou přehrávat hudbu, upravovat dabázi a načítat webové stránky a všechno zdánlivě najednou. Ve skutečnosti jsou jednotlivé aplikace pouštěny a odebírány z procesoru v milisekundových intervalech a za běžných okolnotí je prakticky nemožné všimnout si přerušení její aktivity. Tento pricnip umožňuje

zefektivnit správu dostupných zdrojů a smysluplně vytížit i stále výkonnější procesorové jednotky.

Mluvíme-li o jednotlivých aplikacích, je každá v rámci operačního systému reprezentována ve formě procesu. Každý spustitelný program po své aktivaci vytvoří proces, který provádí jeho činnost. Proces je vždy tvořen nejméně jedním vláknem představujícím menší pracovní jednotku s konkrétním úkolem. Podobně jako v případě více běžících aplikací, je možné řídit i více vláken v prostoru jediného procesu. Opět tak dochází ke zvýšení výkonu a zlepšení vytížení dostupných prostředků, které byly procesu přiděleny. Na rozdíl od přepínání jednotlivých procesů obstarávaného operačním systémem, jsou však vlákna řízena v rámci aplikace jejím vlastním kódem. Využití vláken při programování a návrhu aplikace přináší mnohé výhody. Všechna vlákna vytvořená v rámci procesu sdílejí stejný adresný prostor, jejich přepínání se normálně považuje za beznákladné stejně jako komunikace mezi jednotlivými vlákny.

Při návrhu více vláknové aplikce je však potřeba brát ohled i na jisté požadavky a problémy, které mohou nastat. Jedním ze základních problémů je synchronizace přístupu ke sdílenému objektu nebo sadě informací. Je nezbytné zajistit, aby se dvě různá vlákna nepokoušela simultánně přistoupit ke stejným datům a provádět jejich změny. Dalším typickým problémem je zajištění předávání dat mezi vlákny s pomocí vyrovnávací paměti či objektu s pevnou velikostí. Vlákno generující data se nesmí pokoušet vložit nové hodnoty pokud je vyrovnávací pamět plná a naopak, jejich konzument nesmí číst, pokud je prázdná. Oba tyto problémy lze vyřešit mechanizmy synchronizace vláken a použití zpráv pro vzájemnou komunikaci.

Stejně jako se procesy v rámci operačního systému nacházejí vždy v nějakém stavu, i jednotlivá vlákna mohou nabývat různých hodnot v rámci svého životního cyklu. V rámci programovacího jazyka JAVA jsou stavy vláken plně svázány s běhovým virtulním prostředím a nijak nezávisí na stavech jednotlivých procesů v prostředí operačního systému.

Hlavním důvodem pro používání vláken je možnost paralelních výpočtů či přenesení náročné činnosti mimo hlavní vlákno aplikace, které tak nebude zpomalováno například čekáním při zápisu nebo čtení souboru. Vlákna je možné dynamicky vytvářet a rušit podle toho, zda jsou v daném okamžiku vyžadována. [19]

3.2 Činnosti vláken

Firewallová aplikace využívá hlavní vlákno pro uživatelské rozhraní a dále pět pomocných vláken, kdy každé odpovídá za určitou aktivitu nezbytnou pro celkový chod, které by neměly být přerušeny ostatními činnostmi.

Hlavní vlákno je vyčleněno na obsluhu uživatelského vstupu, vykreslování uživatelského rozhraní a správu souborů aplikace i její komunikaci s prostředím shellu operačního systému.

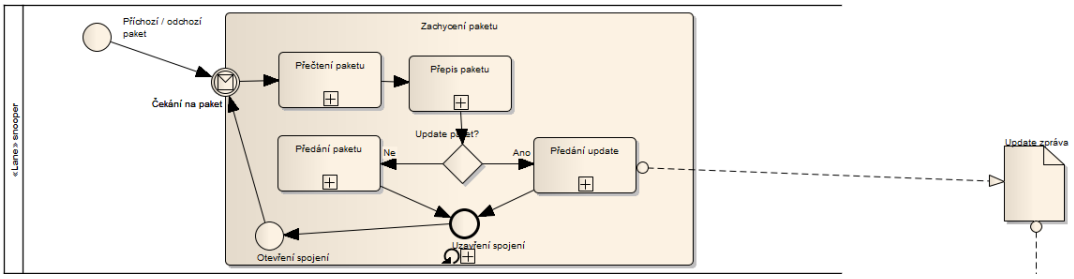
Primárním vláknem vzhledem k potřebám aplikace je rozhraní pro zachytávání a přepis síťové komunikace na úrovni jednotlivých paketů.

Sekundární vlákno představuje analytickou část aplikace. Po uplynutí nastaveného časového intervalu, který slouží pro četnostní vymezení průchozí komunikace, začne procházet seznam zachycených zpráv a podle potřeby je podrobovat analýze.

Kvůli nutnosti oddělit operace čtení a zápisu do stejného seznamu objektů a shromáždění četností výskytu identické komunikace, bylo vytvořeno třetí vlákno, které odpovídá za přetočení bufferu a tím i změnu zaměření předchozích dvou vláken. Současně kvůli zajištění schopnosti aplikace správy rozdílných nastavení jednotlivých uživatelských profilů je vlákno odpovědné za přepínání kontextu těchto profilů.

Poslední dvě vlákna jsou určena pro podporu chodu aplikace v rámci místní sítě. Prvním je rozhraní určené pro aplikování aktualizačních zpráv zachycených od ostatních počítačů v lokální síti, zatímco druhé odpovídá za vytvoření a odesílání aktualizačních zpráv z místního počítače do lokální sítě.

V následujících částech budou blíže popsané odpovědnosti jednotlivých vláken.



Obrázek 6 - Diagram činnosti vlákna "snooper"

3.2.1 Vlákno "snooper"

Přeloženo označuje odposlouchávání nebo slídila. Činnost vlákna přesně odpovídá jeho jménu. Hlavním úkolem je zachycovat a přepisovat veškerou komunikaci procházející daným síťovým rozhraním. Jedná se o jediné vlákno, jehož činnost lze z prostředí uživatelského rozhraní spouštět a vypínat.

Pokud je povolena činnost zachytávání, nachází se vlákno převážně v čekacím stavu na průchozí paket. Vzhledem k principu komunikace s rozhraním, kdy se zachycuje každý jednotlivý paket, je vlákno schopno reagovat i na odchozí komunikaci.

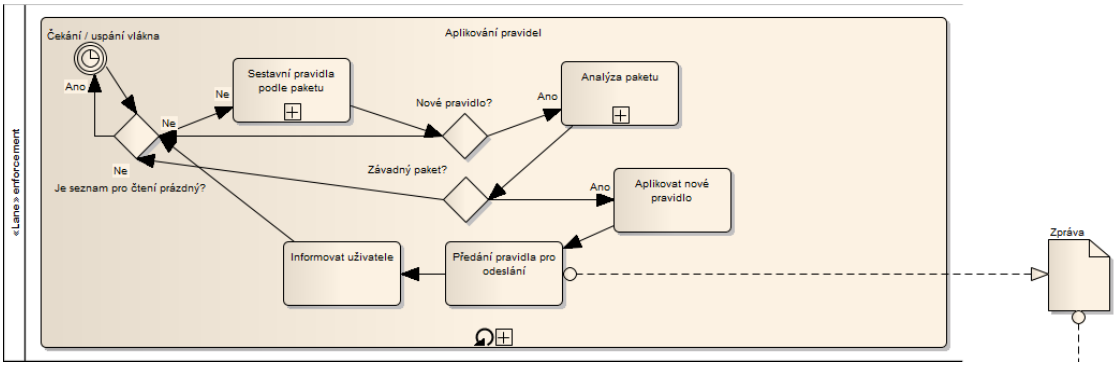
Po zachycení je nutné obsah paketu přečíst a přepsat důležité údaje do interních objektů pro další správu. V tomto okamžiku dochází k rozlišení průchozí komunikace a aktualizačních zpráv. V obou případech je objekt reprezentující paket předán na další zpracování, ale každý směřuje jinam. Komunikace je zařazena do akumulačního zásobníku a v případě vícenásobného výskytu je inkrementován počet zachycení. Aktualizace je předána do bufferu pro analýzu vláknem odpovědným za aktualizaci místního seznamu pravidel pro firewall.

3.2.2 Vlákno "enforcement"

Vlákno pro vynucování pravidel je odpovědné za analýzu přepisu zachycených paketů, případně vytvoření pravidla pro firewall a jeho zanesení do potřebných struktur.

Vlákno čeká do otočení seznamu pro čtení. Následně odebírá položky představující přepis zachycených paketů v interních strukturách. Vzhledem k náročnosti samotné analýzy, dojde k vytvoření pravidla a jeho porovnání s pracovní kopií aplikovaných pravidel v rámci firewallu. Jedná se o kontrolu existence identického pravidla, pokud už bylo zařazeno mezi aktivní, není nutná další analýza a objekt je zahozen.

V případě neznámého pravidla je zdrojový objekt podroben analýze skrze samostatnou třídu. Výsledkem je jednoduchá zpráva zda lze paket považovat za nezávadný a kdy tedy bude zahozen. V opačném případě dojde k aplikování pravidla a jeho zanesení do všech potřebných struktur - do pracovního seznamu, do souboru s pravidly dle uživatelského profilu a zanesení do firewallového rozhraní. V posledním kroku je aktuální pravidlo zobrazeno v uživatelském rozhraní.

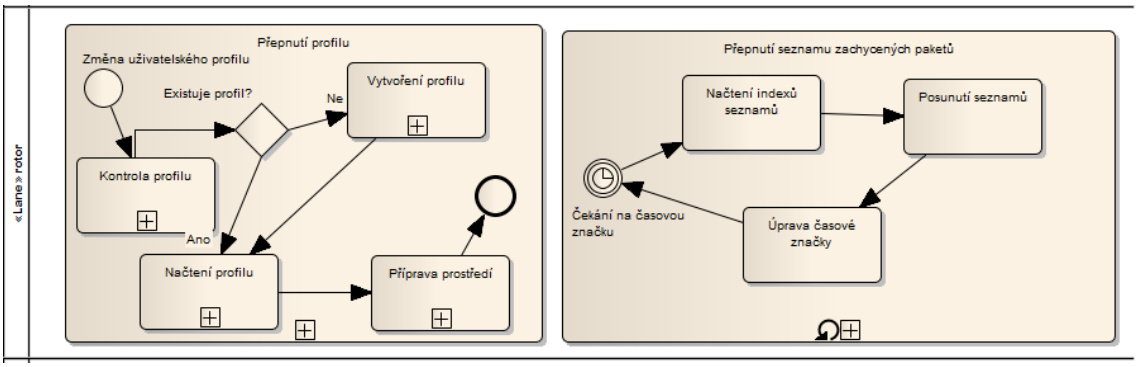


Obrázek 7 - Diagram činnosti vlákna "enforcement"

V případě nastavení rozesílání aktualizačních zpráv dojde k přidání pravidla do seznamu pro odeslání, který je postupně zpracováván vláknem "updater".

Vlákno nese však ještě jednu odpovědnost. Po spuštění aplikace před samotným zahájením své běžné činnosti prochází všechna dosud vytvořená pravidla a provádí jejich vzájemné porovnání. Hledá podobnost ve zdrojové adrese. V případě nalezení většího počtu pravidel se stejným zdrojem, může všechna individuální pravidla vymazat a nahradit je novým univerzálním záznamem. Hranici, kdy mají být pravidla vyměněna, lze nastavit skrze uživatelské rozhraní.

Tato činnost je vykonána vždy pouze jednorázově po spuštění aplikace nebo po přepnutí uživatelského profilu za běhu aplikace kvůli vysoké režii spojené s porovnáním seznamu vytvořených pravidel vůči jemu samému. V okamžiku, kdy bude zanesených pravidel více, mohlo by při pravidelné kontrole docházet k nežádoucímu zpomalení jak aplikace, tak i celého operačního systému.



Obrázek 8 - Diagram činnosti vlákna "rotor"

3.2.3 Vlákno "rotor"

Ačkoliv se může zdát bezvýznamným, plní vlákno "rotor" důležitou činnost v rámci aplikace. Reaguje na dvě samostatné události, které se mohou vyskytnout.

První spočívá v uplynutí časového intervalu, který lze z prostředí uživatelského rozhraní nastavit, určujícího dobu sběru paketů a získávání počtu výskytu jednotlivých zpráv. Při dosažení dané časové značky dojde k přetočení seznamu zachycených paketů, čímž se otevře jeden ze seznamů pro čtení a analýzu objektů reprezentujících komunikaci a prázdný seznam je k dispozici pro další zachytávání.

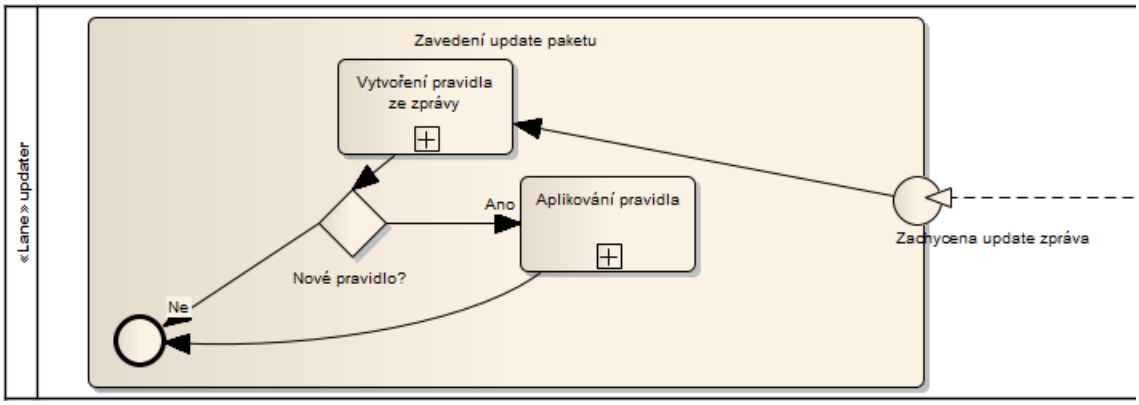
Druhá činnost vlákna spočívá ve vlastnosti aplikace, která podporuje více uživatelských profilů a možnost jejich okamžité změny. Každý profil může být v rámci daných parametrů libovolně nastaven a tím dojde k celkové změně chování aplikace i firewallových pravidel při načtení nového profilu.

Přepnutí kontextu uživatelského účtu je provedeno po volbě uživatele aplikace. Změna spočívá v kontrole uloženého profilu. Tento krok je důležitý, protože aplikace je schopna samostatného vytváření profilů při prvním spuštění po instalaci. Současně se jedná o ochranu při porušení souboru s nastavením aplikace.

Následně dojde k načtení nastavení daného uživatelského účtu a jejich aplikování do prostředí aplikace. Při této činnosti je důležité, aby nedocházelo k zásahům do aplikovaných pravidel, proto je vlákno schopno přerušit činnost analytického vlákna dokud nedojde k úplnému přepnutí kontextu uživatele.

3.2.4 Vlákno "updater"

Vykonává svoji činnost při zachycení aktualizační zprávy z jiného počítače v rámci místní sítě. Většina akcí může být podobná či přímo identická s analytickým vláknem, liší se však právě v absenci analýzy objektu.



Obrázek 9 - Diagram činnosti vlákna "updater"

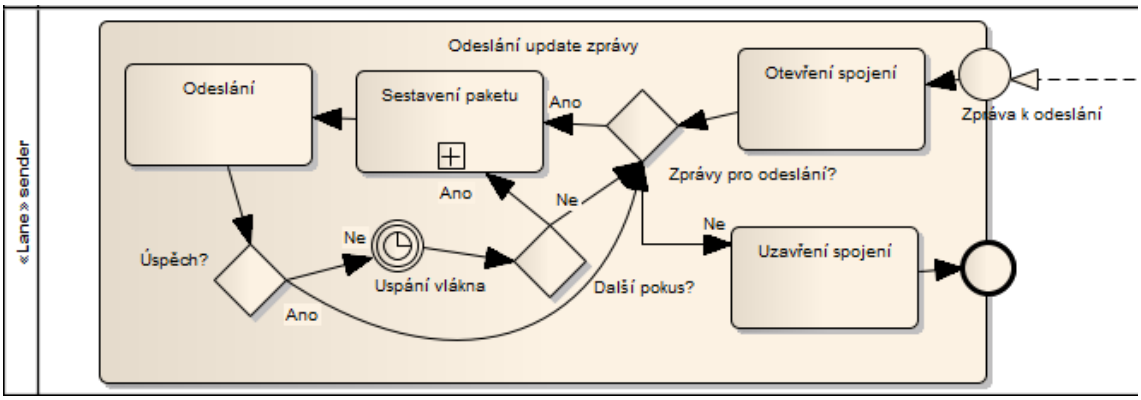
Jakmile je předána zachycená aktualizační zpráva, vlákno přečte objekt a z nákladu paketu vytažené pravidlo jednoduše zapíše do potřebných seznamů. Tato situace však nastane pouze tehdy, pokud podobné pravidlo už není obsaženo v pracovní kopii seznamu aplikovaných pravidel.

3.2.5 Vlákno "sender"

Poslední vlákno odpovídá za tvorbu a odesílání aktualizačních zpráv, pokud uživatel tuto funkcionalitu povolil v rámci nastavení aplikace.

Vlákno zpracovává seznam nově přidaných pravidel určených pro odeslání do místní sítě, která byla předána vláknem "enforcement". Pro každou zprávu je sestaven nosný paket a text pravidla je zabalen do nákladní oblasti.

Následuje pokus o odeslání. Protože je spojení vytvářeno na stejném rozhraní, které je využíváno pro odchyt komunikace, dochází k určité neúspěšnosti přenosu. Kvůli této skutečnosti bylo doplněno opakované odesílání identické zprávy po uplynutí vyčkávacího intervalu. Pokud v jejich průběhu nedojde k úspěchu, je zpráva zahozena a přechází se k další v zásobníku.



Obrázek 10 - Diagram činnosti vlákna "sender"

3.3 Model tříd

Díky využití objektově orientovaného programovacího jazyka je jedině vhodné rozdělit odpovědnosti za určité oblasti do různých tříd, potažmo balíčků, které by zapouzdřovali určité společné rysy. Tím se jednak návrh rozpadne do menších celků, u kterých je mnohem jednodušší definovat odpovědnost a potřebné vlastnosti, a zároveň vytvoří i logicky uspořádaný model představující navrhované řešení.

Na základní úrovni je vhodné rozdělení podle příslušnosti obsahu balíčku vzhledem k řešené problematice a je zastoupena balíčky ***app***, ***gui***, ***log*** a ***model***.

Balíček ***app*** představuje vstupní bod programu. Obsahuje jedinou třídu odpovědnou za spouštění celé aplikace.

3.3.1 Balíček gui

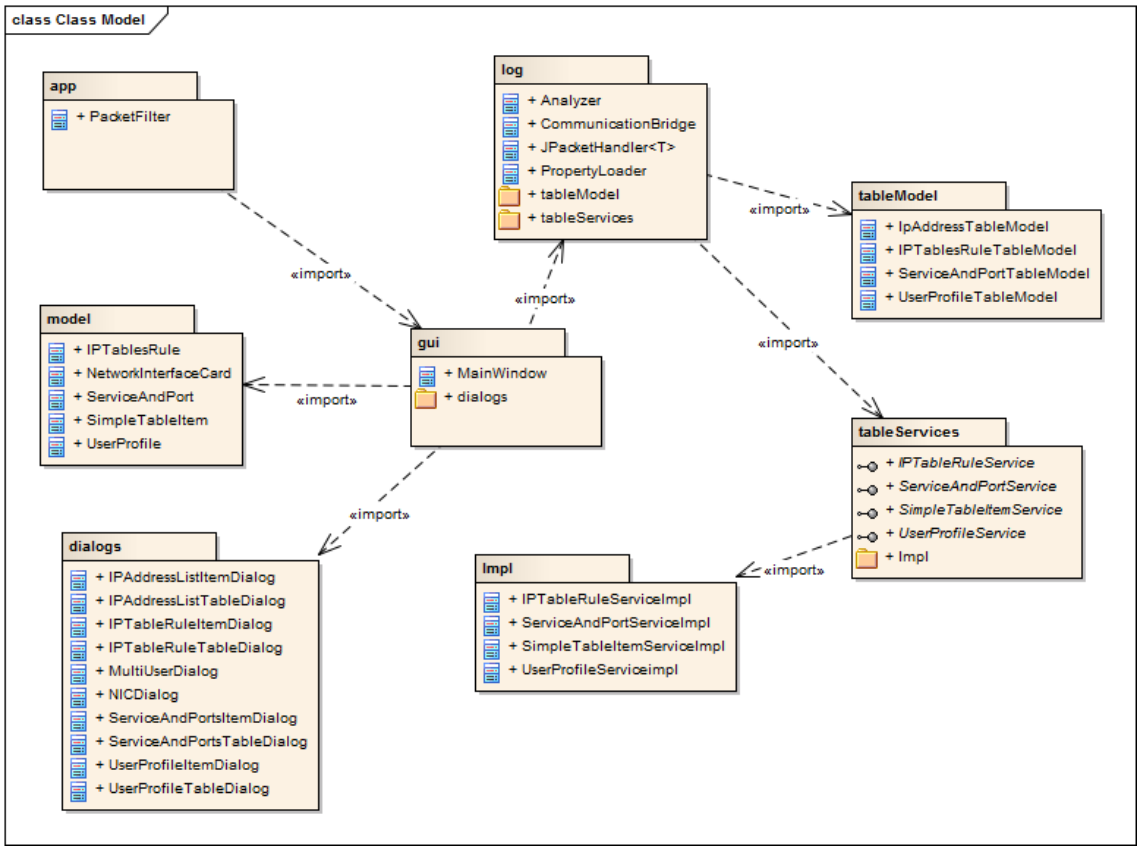
Veškerý obsah balíčku je odpovědný za zobrazování a reakce na uživatelské podněty. Jedná se o komponenty odpovědné za vytvoření hlavního ovládacího okna celé aplikace a dialogových oken zpřístupňujících detailnější informace a prvky v rámci návrhu.

Kvůli množství samostatných dialogových oken, z nichž každé je reprezentováno zvláštní třídou, byla všechna umístěna do balíčku ***dialogs***.

3.3.1.1 Třída MainWindow

Jedná se o hlavní třídu celé aplikace, která představuje propojovací prvek mezi všemi ostatními segmenty programu. Obsahuje odkazy a reference na objekty poskytující rozšířené operace se soubory a daty, stejně jako většinu modelových tříd reprezentujících interní struktury aplikace.

Představuje společné paměťové rozhraní celé aplikace pomocí sady privátních seznamů a objektů, které jsou přístupné skrze reference předávané při volání metod nebo v rámci vlastních rozhraní. Příkladem může být paměťová struktura, sada propojených seznamů, určená pro řízení komunikace vláken a předávání interních objektů zastupujících zachycenou komunikaci na síťovém rozhraní.



Obrázek 11 - Diagram tříd a balíčků

Převážná část odpovědnosti třídy je zaměřena na samotné vytvoření aplikačního prostředí, jakož i vytvoření a spuštění jednotlivých vláken. Následně vyčkává na požadavky uživatele, který skrze kontrolní prvky může upravovat chování, zobrazené informace či běh aplikace.

Obsahuje i metody pro ukončení celé aplikace, kdy nejdříve zastavuje provádění jednotlivých vláken a následně provede jejich zrušení, aby mohl být program ukončen bez ponechaných artefaktů. V rámci zastavování vláken dochází i k uzavření všech otevřených spojení do souborového systému, pokud nějaká existují, nebo k síťovému hardwaru.

3.3.2 Balíček log

Obsahem balíčku jsou objekty, které v rámci aplikace odpovídají za logiku a správu některých zdrojů. Jedná se především o komunikační rozhraní určené pro zasílání povelů do rozhraní shell operačního systému, dále o spojení se souborovým systémem kvůli přístupu k ukládaným datům a v neposlední řadě i objekty pro čtení a analyzování probíhající komunikace na daném síťovém rozhraní počítače.

Dalším obsahem jsou podpůrné struktury určené k načtení a zobrazení podrobných informací v tabulkovém rozložení pro jednotlivá dialogová okna rozšiřující možnosti aplikace. V tomto případě se jedná o další dva balíčky ***tableModels***, sdružující definice tabulek pro zobrazení, a ***tableService***, představující rozhraní pro práci s prvky zobrazenými skrze tabulkové rozložení.

3.3.2.1 Třída Analyzer

Analytická třída nesoucí odpovědnost za rozhodnutí ohledně akce spojené se zachycenou zprávou. Samotná analýza probíhá v několika krocích a každé rozhodnutí bude popsáno důvodem pro daný výsledek, který je dostupný skrze metodu třídy. Proces analýzy paketu bude detailněji probrán v kapitole **3.7 Analýza paketu**.

Současně představuje rozhraní pro překlad masek síťových adres mezi číselným a bitovým vyjádřením. Důvodem je nutnost použití bitového tvaru ve spojení s odpovídající síťovou adresou pro stanovení platné části sítě, zatímco číselné vyjádření je vhodné pro reprezentaci v rámci interních struktur nebo zobrazení uživateli skrze uživatelské rozhraní.

3.3.2.2 Třída CommunicationBridge

Komunikační rozhraní aplikace pro spojení s shellem operačního systému nebo souborovým systémem.

Kromě možnosti vytvářet a posílat vlastní příkazy, samozřejmě odpovídající struktury v rámci operačního systému, je možné i přijímat odpovědi a v rámci aplikace je podle potřeby upravit a případně na ně i reagovat. Primárně se této funkcionality dá využít při zpětném získání seznamu aplikovaných pravidel v rámci struktury iptables [14], jejich vyžádáním a zpětnou transformací do interního objektu pro další zpracování.

Druhá odpovědnost spočívá ve spojení se souborovým systémem. Třída je schopna načítat soubory nesoucí nastavení aplikace a profily uživatelů nebo záznamy jednotlivých uživatelských účtů s platnými pravidly pro rozhraní iptables [14]. Vzhledem k potřebě dynamické správy těchto souborů, při vytvoření nebo zrušení uživatelského profilu, je třída schopna vytvářet soubory s náhodným jménem, které je v rámci aplikace unikátní, stejně jako je odstraňovat, pokud už nadále nejsou potřebné. Současně umožňuje dynamicky do souborů zapisovat ve dvou režimech, úplné přepsání obsahu nebo připojení nového obsahu k předchozímu.

3.3.2.3 Třída JPacketHandler

Rozhraní založené na knihovně JNetPcap [20] určené pro zachycení a přečtení síťového paketu. Každý jednotlivý paket je tímto rozhraním zpracován. Objekt, reprezentující zachycenou komunikaci, je postupně rozebrán podle jednotlivých vrstev ISO/OSI modelu a požadované informace jsou extrahovány z odpovídajících atributů.

Zároveň dochází k základní identifikaci zdroje paketu. Pokud byl odeslán z jiného počítače v rámci místní sítě a adresován pro všechna zařízení v lokální síti, je "náklad" paketu obsahující aktualizační zprávu extrahován.

Mezi poslední činnosti náleží identifikace protokolu, který daný paket vytvořil. Za tímto účelem se hledá odpovídající hlavička paketové obálky, poslední před nákladním prostorem, aby bylo možné získat název protokolu pro jeho další identifikaci v rámci samotné analýzy.

3.3.2.4 Třída PropertyLoader

Kvůli nastavení aplikace vznikla potřeba struktury pro uložení vlastností a jejich hodnot. Problém se objevil při zamýšlení implementace více uživatelských profilů, z nichž každý může mít nezávislé nastavení v daných bodech a tedy není předem jasný počet těchto profilů. Z toho vyplynula potřeba rozhraní, které by danou paměťovou strukturu dokázalo i spravovat.

Výsledkem je třída *PropertyLoader*, ta je založena na rozhraní *Properties*, které zapouzdřuje, čímž odstraňuje nutnost ošetření výjimek v každém místě použití volání metody přistupující do souboru a přidává některé nové možnosti pro pohodlnější načítání rozličných druhů dat, například umožňuje načtení a transformaci hodnoty do podoby seznamu nebo zajištění jedinečné hodnoty skrze více rozdílných klíčů.

Struktura properties souboru je tvořena dvojicemi klíč a hodnota, kdy klíč musí být v rámci celého souboru unikátní.

3.3.3 Balíček model

Balíček sdružuje interní objekty, které zastupují nebo vytvářejí prvky s dostatečným popisem pro snadnou manipulaci v rámci aplikace. Jejich hlavním účelem je poskytnou sdružené informace, které jsou snadno přístupné skrze standardní rozhraní objektu a nezřídka obohacují původní zástupný objekt o nové údaje.

Vzhledem k návrhu celé aplikace jsou modelové třídy vhodné i pro funkcionality používané v rámci dialogových oken a tabulkového zobrazení dat.

3.3.3.1 Třída IPTablesRule

Hlavní modelová třída aplikace. V základním použití představuje zástupný objekt pro reprezentaci zachyceného paketu na síťovém rozhraní. Kvůli tomu disponuje řadou atributů, které popisují původní data v zachycené zprávě.

Protože je nutné potencionálně závadnou komunikaci převést na formu pravidla použitelného v rámci rozhraní iptables [14] operačního systému, disponuje třída převodem do formy textového řetězce. Pro podporu více rozličných typů pravidel, ať už chováním nebo umístěním, představuje třída i statické rozhraní s konstantami použitelnými v rámci konstrukce povelu pro iptables [14].

Kromě standardního rozhraní, sady konstruktorů, setterů a getterů, je dostupná i metoda schopná zpětného převodu z textového řetězce do objektu. Díky tomu je možné celý objekt jednoduše uložit do souboru a následně obnovit, aniž by došlo ke ztrátám nesených informací nebo nadbytečné zátěži při rekonstrukci. Díky skutečnosti, že se pro uložení používá struktura textového řetězce založená na formátu pravidla pro rozhraní iptables [14], je možno převádět i pravidla již obsažená v tomto rozhraní.

Současně vzhledem k potřebě identifikovat stejné objekty nabízí třída statické rozhraní pro porovnání, které bere v rámci testu pouze podstatné údaje. Tím je umožněno nalezení identického objektu, který však může nést některé odlišné popisné informace.

3.3.3.2 Třída NetworkInterfaceCard

Jednoduchá struktura pro seskupení údajů nutných pro správnou identifikaci rozhraní síťové karty. V rámci atributů je udržována i informace pro podporu vlastní komunikace jako je přidělená síťová adresa, hardwarová adresa a korespondující adresa pro broadcast, tedy vysílání zaměřené na všechna zařízení v rámci lokální sítě.

3.3.3.3 Třída ServiceAndPort

Jedná se o strukturu pro přepis záznamů určených pro analýzu komunikačních paketů zachycených v rámci aplikace na daném síťovém rozhraní. Atributy objektu reprezentují číslo portu, transportní protokol a odpovídající službu.

Třída zároveň obsahuje přepracované rozhraní pro zjištění ekvivalentního objektu, aby byla umožněna kontrola i nekonzistentních instancí objektu.

3.3.3.4 Třída UserProfile

Struktura určená pro tvorbu a správu uživatelských profilů v rámci aplikace. Objekt předpokládá existenci profilu s daným jménem v rámci properties souboru, odkud probíhá načítání hodnot interních atributů. Modelová třída je určena především pro hromadné zobrazení uživatelských účtů v rámci odpovídající tabulkové struktury dialogového okna pro správu profilů aplikace.

3.4 GUI

Aplikace by měla nabízet pohodlné uživatelské rozhraní, které by umožnilo její ovládání a přizpůsobení potřebám konkrétního uživatele.

Základním prvkem je tedy obrazovka. Ta by zprostředkovala určité výsledky automatického chování aplikace uživateli, který by na ně měl možnost reagovat.

Vzhledem k náročnosti vystavění aplikace na příkazovém principu, který je tak typický pro operační systému typu GNU/Linux, byla vybrána varianta počítající se vznikem kontrolních prvků, tlačítek a formulářových oken rozšiřujících základní obrazovku programu.

Ve spojitosti s možnostmi a potřebami aplikace došlo k navržení funkcionalit, které by umožnili její celkové ovládání skrze grafické uživatelské rozhraní.

V základním stavu aplikace dovoluje uživateli zobrazit seznam aktuálně aktivních pravidel, včetně možnosti jejich smazání, úpravy či vlastní tvorby skrze přídavné formulářové okno. Dále má uživatel možnost upravit seznam platných uživatelských profilů včetně nastavení jejich provázání kvůli šíření pravidel i do seznamů neaktivních uživatelských účtů. Může upravovat seznamy povolených a zakázaných adres skrze dialogové okno. Poslední možnost představuje seznam kontrolních hodnot přiřazující

čísla portů k daným službám. Kromě přímé editace může uživatel přidávat k jednotlivým záznamům i vlastní komentáře.

Protože aplikace podporuje více uživatelských profilů, ale vždy může být aktivní pouze jeden, je dostupná funkce přepnutí účtu za běhu aplikace. Tato vlastnost již byla po technické stránce popsána v kapitola **3.2.3 Vlákno "rotor"**.

Všechny formuláře přijímající uživatelem vložené hodnoty jsou ochráněny proti neplatným údajům, pokud je možné jejich rozlišení.

3.5 Nastavení

Aplikace podporuje přizpůsobení svého chování podle řady parametrů. Veškeré nastavení je dostupné v samostatném souboru v rámci umístění aplikace, který lze editovat i mimo běžící aplikaci; ačkoliv se tato činnost nedoporučuje kvůli zachování integrity informací. Princip uchování nastavení tak sleduje trend operačních systémů typu GNU/Linux, které mají své konfigurační soubory též přístupné a upravitelné běžnými prostředky.

3.5.1 Globální možnosti

Upravují chování aplikace bez ohledu na aktivní profil. Při aktivování jiného profilu nedochází v průběhu přepnutí kontextu uživatele ke změně i těchto hodnot. Pokud však uživatel v rámci editace účtu provede změny týkajících se těchto možností, budou promítnuty do všech současných i budoucích uživatelských účtů v rámci aplikace.

První z možností ovlivňuje chování aplikace hned po spuštění. Aplikace dokáže nastavit výchozí profil podle zadaného jména uživatelského účtu a tím načíst i ostatní možnosti přizpůsobení aplikace vázané na daný profil.

Dále je možnost určit, kdy má aplikace začít se svojí činností. Buď má vyčkat až na pokyn uživatele, který skrze uživatelské rozhraní může zachycování a analýzu paketů kdykoli spustit nebo přerušit, nebo ihned po inicializování všech komponent aplikace. Jedná se tedy o automatické zahájení činnosti po spuštění aplikace.

Poslední dvě možnosti ovlivňují komunikaci aplikace v rozsahu místní sítě. První z možností určuje chování aplikace při zachycení aktualizačního paketu, který může být použit pro rozšíření aplikovaných pravidel, získaných na základě činnosti místní instance programu. Druhou možností je jeho zahození a ignorování nesené zprávy. Tato volba se může hodit například pro stanici umístěnou těsně za vstupním bodem do místní sítě, která by neměla přijímat aktualizace z interní sítě, ale pouze dále distribuovat vlastní zprávy.

Poslední možnost ovlivňuje odchozí aktualizační zprávy produkované programem v případě nalezení podezřelé komunikace přicházející na lokální síťové rozhraní počítače. V případě povolení této funkcionality bude po vygenerování lokálního pravidla vytvořena zpráva obsahující odpovídající pravidlo a odeslána skrze aktivní síťové rozhraní počítače do místní sítě.

3.5.2 Lokální možnosti

Nastavení chování aplikace v závislosti na aktuálně zvoleném profilu. V průběhu přepínání kontextu uživatelského profilu dochází ke změně všech možností podle hodnot uložených pod novým uživatelských účtem.

Mezi základní volby patří nastavení prahu pro započtení průchozí komunikace jako nežádoucí. Jedná se o číselné vyjádření počtu identických objektů reprezentujících zachycený paket v daném cyklu. Při překročení této hodnoty výskytu bude vygenerováno pravidlo pro filtrování odpovídající komunikace pomocí iptables [14].

Jelikož činnost aplikace může generovat velice podobná pravidla, lišící se například jenom jiným číslem portu, může si uživatel nastavit hodnotu, kdy mají být tato pravidla sdružena do jediného. Při každém startu aplikace tak bude docházet k přezkoumávání všech pravidel a pokud počet podobných pravidel překročí uživatelem definovaný práh, budou odstraněna a nahrazena univerzálním. To bude zapsáno do seznamu uživatelem zakázaných adres.

V rámci nastavení profilu si uživatel může určit délku intervalu, v rámci jehož trvání dochází k zachycování komunikačních paketů a jejich hromadění pro získání počtu výskytu identického objektu. Vždy po uplynutí intervalu bude následovat analýza zpráv zachycených v jeho průběhu.

Pro automatické generování pravidel je důležité zvolit výchozí typ, který má být generován. K tomu slouží volba akce paketu v případě pozitivního výsledku analýzy.

Struktura iptables [14] nabízí kromě samotných pravidel, která vynucují nějakou akci, i globální nastavení, které je aplikováno na průchozí komunikaci nevyhovující žádnému jinému pravidlu. Tato možnost se nazývá politika. V rámci nastavení aplikace je uživatel schopen určit výchozí politiku firewallu.

Vzhledem k jednotlivým fázím analýzy zachycených paketů je vhodné definovat akci v okamžiku sporného výsledku. Tato volba rozhodne, zda ukončit testování s kladným nebo záporným výsledkem, který by propustil paket bez dalšího omezení. Spor ve výsledku může nastat v okamžiku nesjednoceného pojmenování napříč službami nebo chybějícími údaji v těle zachyceného paketu.

Při vytvoření nového profilu je důležité jeho zacílení na dané síťové rozhraní počítače. Každý uživatelský účet musí být zaměřen na rozhraní, které je považováno za aktivní a je přes něj vedena komunikace z místního počítače. Zde se nabízí možnost využití jednotlivých profilů při přenosu aplikace mezi počítači nebo pro ochranu mobilního zařízení či laptopu, který může v různém prostředí komunikovat skrze různé druhy síťových adaptérů jako je síťový port, adaptér bezdrátového připojení a podobně.

Další volba ovlivňuje spojení mezi jednotlivými uživatelskými profily. Aktivní profil bude veškerá pravidla vytvořená na základě analýzy probíhající komunikace ukládat i do souborů připojených profilů. Umožňuje to aktualizovat pravidla i v daném okamžiku neaktivních účtů, čímž se zajistí maximální efektivita při přepnutí kontextu uživatele v rámci aplikace.

Každý uživatelský profil si může definovat dva seznamy pro urychlení filtrování a analýzy probíhající komunikace. V případě adres, které nemají být kontrolovány a v žádném případě zablokovány, je lze přidat do seznamu povolený, tzv. whitelist.

V rámci analýzy jsou pak všechny komunikační pakety směřující do a nebo z dané adresy propuštěny bez vytvoření pravidla i v případě, kdy by jeho vytvoření bylo z hlediska procesu analýzy vhodné.

Druhou možností je seznam zakázaných adres, tzv. blacklist. Každý záznam v tomto seznamu je jednoduše aplikován jako omezující pravidlo do iptables [14] a následně jsou všechny komunikační pakety porovnávány s danou adresou. Při nalezení shody, která by nějakým způsobem pronikla skrze firewallové rozhraní, je doplněno nové pravidlo a komunikace s danou lokací zakázána.

V obou případech je porovnávána adresa ze seznamu včetně masky sítě, která vymezuje rozsah platnosti daného záznamu od jedné konkrétní adresy až po celý dostupný rozsah.

3.6 Uživatelé

Aplikace podporuje více uživatelských profilů, které jsou do určité míry navzájem nezávislé. Každý profil je tvořen sadou nastavení, které charakterizují chování aplikace a bezpečností politiku. Zároveň při založení nového účtu dochází k vytvoření souboru, který je s tímto uživatelem propojen a obsahuje seznam pravidel, která byla vygenerována aplikací. Těmi mohou být pravidla získaná za aktivního běhu daného profilu, tak na základě propojení od jiného účtu, spojení je vždy pouze jednosměrné.

Možnost svázání uživatelských profilů se nabízí pro využití v rámci přenositelnosti zařízení nebo samotné aplikace. Je tak možné odlišit profily nejen podle síly jejich ochrany, ale i například podle lokace, kde se aplikace nachází a současně sdílet generovaná pravidla pro neaktivní účty.

Každý účet je jednoznačně identifikován jménem, které představuje jedinou později nezměnitelnou položku. Všechny ostatní hodnoty, které utváří daný profil, jsou upravitelné skrze uživatelské rozhraní. Nabízí se samozřejmě i možnost úplného odstranění účtu. V tomto případě však dochází k ochraně aplikace, kdy nelze vymazat primární účet. Tím je považován aktuálně nastavený spouštěcí profil v rámci globálního nastavení programu, který lze samozřejmě skrze rozhraní též změnit.

3.7 Analýza paketu

Stěžejní činnost celé aplikace spočívá v analýze zachyceného komunikačního paketu a rozhodnutí o dalším kroku s ním spojeným. V následující části bude blíže popsán princip této analýzy a možné výsledky v závislosti na hodnotách získaných ze zachyceného paketu.

V prvním kroku dochází k odlišení komunikace směřující ven z lokálního počítače a do něj. Využívá se k tomu atributu struktury reprezentující daný paket v rámci celé aplikace. Jednoduše se zjistí, do které tabulky v rámci rozhraní iptables [14] by patřilo případné pravidlo a podle výsledku se nastaví odpovídající hodnota do objektu.

Následuje kontrola zaměřena převážně na činnost samotného uživatele aplikace. Porovnání zdrojové adresy paketu vůči dvojici definovaných seznamů adres s daným rozsahem, whitelist a blacklist. Zjišťuje se, zda zdrojová ip adresa paketu spadá do rozsahu kterékoli z uložených adres. Pokud je taková skutečnost zjištěna při porovnávání s obsahem whitelistu, je paket, zde reprezentovaný ve formě interního

objektu, označen za nezávadný a celý proces analýzy ukončen. Pokud je nalezena shoda vůči adrese zanesené v blacklistu, je paket označen jako závadný, do interního objektu je doplněn důvod tohoto rozhodnutí, v tomto případě reflektující nalezenou shodu v seznamu zakázaných adres, a analýza je ukončena s pozitivním výsledkem. Další akce spojené s tímto výsledkem jsou už mimo samotný analytický proces, nicméně se jedná o vytvoření a aktivování pravidla blokujícího daný rozsah komunikace přes rozhraní iptables [14].

Za předpokladu, že všechny předchozí kontroly byly negativní, dojde k porovnání hodnot hardwarových adres zachyceného paketu. Tím se ověřuje cesta, kterou daný paket urazil v rámci lokální sítě a sice, zda je určen výhradně pro chráněné síťové rozhraní počítače a pochází od zařízení, které předává běžnou komunikaci. Za normálních okolností lze za takové zařízení považovat router, switch, hub nebo ekvivalentní typ síťového hardwaru. Na základě údajů získaných ze zachycené komunikační zprávy dojde ke kontrole, zda je paket určen pro dané rozhraní počítače jednoduchým porovnáním cílové MAC adresy a platné MAC adresy síťové karty místního počítače. Následuje kontrola fyzické adresy zařízení, se kterým síťová karta běžně komunikuje a její porovnání se zdrojovou, respektive cílovou MAC adresou paketu, v závislosti na směru průchodu skrze síťové rozhraní. Pokud by byl zjištěn rozpor, tedy původ či cíl paketu nedopovídá běžnému zařízení, je komunikace označena jako závadná a analýza ukončena s pozitivním výsledkem.

V další fázi je kontrolován počet výskytů identického objektu v daném časovém intervalu. Identickým objektem je myšlena shoda prakticky ve všech získatelných parametrech po oddělení nákladní části, která nese samotnou zprávu. Počet těchto výskytů je tedy porovnán s hodnotou uloženou v properties souboru, kterou lze skrze uživatelské rozhraní libovolně nastavit. Pokud zjištěný počet výskytů přesáhne stanovenou mezní hodnotu, je zpráva označena jako závadná a analýza ukončena.

Poslední fáze kontroly zachyceného komunikačního paketu je z hlediska času nejnáročnější. Každá komunikace směřující ke konkrétní aplikaci nebo službě nese v rámci jedné z hlaviček paketu číslo portu a název používané služby. Pokud by jediná z těchto hodnot chyběla, bude analýza ukončena s pozitivním výsledkem.

Za předpokladu, že paket obsahuje obě požadované hodnoty bude zahájena jejich analýza. Vzhledem k velkému počtu služeb a ne zcela sjednocenému názvosloví, probíhá tato konkrétní kontrola dvakrát ve dvou různých směrech.

Nejprve se z údajů získaných ze zachyceného paketu sestaví interní objekt, který se následně porovnává se seznamem hodnot určujících vazbu mezi portem a danou službou. Pokud je nalezen odpovídající objekt, lze paket označit za nezávadný, ukončit analýzu a nezahajovat druhý krok kontroly.

V druhém případě, kdy nebyl nalezen odpovídající objekt, je nutné provést kontrolu ještě obráceně, tedy hledat odpovídající hodnoty v seznamu. Dochází tedy k průchodu kontrolními hodnotami, kdy se hledá číslo portu odpovídající hodnotě v daném paketu a následně i jméno služby korespondující s tímto portem.

Rozdíl oproti předchozí fázi, kdy se porovnával dočasný interní objekt s objekty reprezentujícími kontrolní hodnoty, spočívá ve skutečnosti, že se kontrolují přímo

jednotlivé hodnoty, které lze použít pro definovaní objektu. Pro schválení tak stačí i částečná shoda, jelikož se odstranil obalový objekt přidávající další vrstvu informací.

Pokud však tato kontrola neskončila negativním výsledkem, nastává problém, jak s daným komunikačním paketem naložit. Řešením je definovaná základní akce při dosažení sporného výsledku kontroly. Tato akce je skrze uživatelské rozhraní upravitelná.

3.8 SMART přístup

Velmi zjednodušeně popsáno spočívá princip SMART v ulehčení práce uživatele pomocí předvídání či automatizace. V rámci této kapitoly se pokusím popsat některé funkcionality aplikace, které lze považovat za SMART přístup, protože mnohdy dokážou ušetřit velké množství času, které by jinak bylo potřeba pro jejich provedení.

Už z principu svého účelu, tvorby pravidel pro firewallové rozhraní iptables [14], je aplikace schopna ušetřit velké množství času. Existují sice přístupy pro vytvoření jednorázového skriptu, který nastaví firewall operačního systému, ten se však nedokáže přizpůsobit změnám v komunikaci a rozhodně není jednoduché změnit jeho části jinak než opětovným vygenerováním s odlišnými nastaveními. V tomto ohledu je aplikace výhodná díky skutečnosti, že negeneruje jediný funkční skript, ale navzájem nezávislá pravidla, která jsou následně aplikována podle platného uživatelského účtu. Čímž je opět možné velice rychle změnit chování síťové ochrany pomocí pouhého přepnutí na odlišný profil, který může být zcela jinak nastaven.

Pokud hovořím o nastavení, jsou samozřejmě myšleny hodnoty, které byly probrány v kapitole **3.5 Nastavení**. Dále je však možné odlišovat jednotlivé profily i podle seznamu pravidel, která může uživatel, po jejich automatickém vytvoření, libovolně upravovat nebo i doplňovat o vlastní.

Spousta aplikací navíc vyžaduje při svém prvním spuštění velké množství údajů, které mají sloužit k úvodnímu nastavení aplikace. Tento program požaduje pouze jediný údaj a sice identifikaci aktivního síťového rozhraní, které má být zastřeženo.

Všechny ostatní potřebné údaje si dokáže získat sám z prostředí operačního systému. Jméno přihlášeného uživatele operačního systému poslouží pro název profilu v rámci aplikace, seznam kontrolních hodnot čísel portů a korespondujících služeb je načten ze systémového souboru. Tyto a další údaje je aplikace schopna načíst během několika okamžiků po svém spuštění nebo skrze inicializaci uživatelských nastavení podle defaultních hodnot ve skrytém profilu či analýzou komunikace počítače. Poslední zmíněná metoda je využita především pro získání rozsahů a hodnot potřebných síťových ukazatelů, jako jsou ip a MAC adresy počítače a prvního připojeného zařízení v rámci místní sítě, většinou výchozí brány.

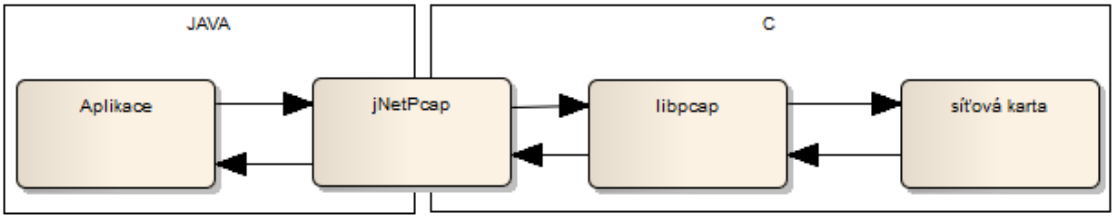
Většina firewallových aplikací pracuje na principu porovnávání se vzorem s následným povolením nebo zakázáním komunikace. Tento program však vytváří přímo pravidla, která toto chování uplatňují a současně je schopen je i zpětně editovat. Při obou procesech bere v úvahu hodnoty, které jsou automaticky nastaveny nebo následně upraveny uživatelem v rámci možností daného profilu. V určitých ohledech tedy dochází i k dynamickému přizpůsobování chování aplikace.

Aplikace navíc spravuje všechna uživatelská nastavení a generovaná pravidla v dynamicky vytvářených souborech a záznamech v rámci properties souboru. Jakýkoli zásah do těchto údajů je aplikací opětovně uložen pro zachování změněných hodnot i po vypnutí a opětovném spuštění aplikace.

4 Popis navrženého řešení

V předchozí kapitole byla popsána logika, kterou bude aplikace implementovat pro dosažení správné funkcionality. V této části bude prezentován a popsán zdrojový kód představující jednotlivé logické celky dohromady tvořící aplikaci.

Pro vývoj byly zvažovány dva programovací jazyky a sice objektově orientovaný jazyk JAVA [21] a jazyk C [22], který byl použit pro vytvoření operačních systému typu GNU/Linux. Vzhledem k větším zkušenostem s jazykem JAVA [21] a existencí potencionálního řešení bylo rozhodnuto pro použití této programovací platformy.



Obrázek 12 - Schéma komunikace aplikace se síťovým rozhraním

Aplikace potřebuje přistupovat k základním funkcím operačního systému, aby bylo možné získat kopii průchozí komunikace na daném síťovém zařízení. Tady bylo nutné využít implementaci překladové knihovny jNetPcap [20], která dokáže zpřístupnit některá volání jádra systému i aplikačnímu jazyku, jakým JAVA [21] je kvůli svému běhovému prostředí. Knihovna je z části napsaná v obou zvažovaných programovacích jazycích a dokáže tedy fungovat jako spojovací most mezi virtuálním běhovým prostředím a jádrem operačního systému.

4.1 Vlákna

Hlavní činnost celé aplikace je rozdělena mezi šest vláken, kdy hlavní vlákno programu představuje převážně vykreslování uživatelského rozhraní a reakce na uživatelské akce. Též odpovídá za spouštění ostatních vláken při startu aplikace.

```
init();
initAction();
initGui();
pack();
setResizable(false);
threadGeneration();

snooper.start();
enforcement.start();
rotor.start();
updater.start();
sender.start();

if(Boolean.parseBoolean(properties.getProperty("autoStartScan")))
    toogleScan();
```

Ukázka ze spouštěcí metody hlavního vlákna, které je představováno třídou *MainWindow*. Volání jednotlivých metod postupně inicializuje potřebné proměnné a vykresluje uživatelské rozhraní. V dalších krocích dochází ke generování kódu

pro ostatní vlákna aplikace, jejichž kód bude blíže popsán v následujících kapitolách, spuštění všech vláken a nakonec kontrola automatického zahájení činnosti. V tomto případě se čte hodnota uložená v properties souboru a parsuje se do objektu typu boolean.

```
private void init(){
    whoami = properties.getProperty("startupProfile");
    if(whoami.equalsIgnoreCase("default")) whoami = whoAmI(true);

    checkUserExist(whoami);
    prepareEnvironment();

    if(bridge.fileExist(sAPLocation))
          rewriteListToServicesAndPorts();
    else setupPorts();

    ipTablesRulesIndex.add(0);
    ipTablesRulesIndex.add(1);
    ipTablesRulesIndex.add(2);
    ipTablesRulesIndex.add(3);

    ipTablesRulesList.add(ipTablesRulesA);
    ipTablesRulesList.add(ipTablesRulesB);
    ipTablesRulesList.add(ipTablesRulesC);
    ipTablesRulesList.add(ipTablesRulesD);
}
```

Metoda *init()* provádí inicializaci uživatelského profilu po spuštění aplikace. Přičemž posloupnost se liší při první aktivaci, kdy je navíc vytvořen nový profil podle jména přihlášeného uživatele v operačním systému, a je nastaven jako výchozí pro další spouštění programu.

Následně dochází ke kontrole existence uživatelského profilu a všech potřebných hodnot v properties souboru s nastavením aplikace. Pokud by profil neexistoval, aplikace zde předpokládá, že je jediná, která zasahuje do obsahu souboru, dojde k jeho vytvoření podle zdrojového profilu, který nelze nijak skrze aplikační prostředí upravit.

Detaily procesu vytvoření uživatelského účtu a potřebných struktur budou blíže popsány v kapitole **4.3.1 Uživatelské profily**.

Po úspěšném načtení profilu je nutné upravit prostředí aplikace podle uživatelských voleb. Této činnosti bude věnován větší prostor v následující kapitole **4.1.1. Nastavení prostředí**.

Dalším krokem aplikace je nalezení nebo inicializace seznamu, který určuje propojení mezi číslem portu, a tedy aplikací, a službou, která může být použita v rámci komunikační zprávy.

```
private void setupPorts(){
    List<String> l = bridge.loadList("/etc/services");
    List<String> temporal = new ArrayList<>();
    String regex = "@";
    for(String line : l)
          if(!line.isEmpty() && !line.startsWith("#")){
                String[] p = line.split(" ");
                String temp = "";
                for(int j = 0; j < p.length; j++){
```

```
                        if(!p[j].isEmpty()){
                              String pl = p[j];

                              if(Character.isDigit(pl.charAt(0))
                                          && pl.contains("/"))
                                    pl = pl.replaceAll("/", regex);

                              if(pl.contains("#")){
                                    j++;
                                    while(j < p.length){
                                          pl += " " + p[j++];
                                    }
                              }
                              temp += (temp.isEmpty() ? "" : regex)
                                          + pl;
                        }
                  }
                  p = temp.split(regex);
                  ServiceAndPort sAP = new ServiceAndPort(
                              p[0], p[1], p[2]);

                  if(p[p.length - 1].startsWith("# "))
                        sAP.setComment(p[p.length - 1]);

                  serviceAndPort.add(sAP);
                  temporal.add(sAP.archive());
            }

      bridge.saveList(sAPLocation, temporal, false);
   }
```

Metoda *setupPorts()* je volána pouze při úvodním spuštění aplikace, nebo v případě nenalezení odpovídajícího souboru mezi strukturami samotné aplikace. Jejím hlavním účelem je přepsání aktuálního seznamu portů a služeb z prostředí operačního systému typu GNU/Linux, který je dostupný v souboru *services*. Skrze třídu pro práci se soubory je z tohoto souboru přečten a vytvořen seznam, který bude následně přepsán do interní struktury a uložen v lokálním adresáři aplikace pro budoucí použití a úpravy, které může provádět uživatel skrze grafické rozhraní.

Každý záznam je přečten, s výjimkou prázdných řádků nebo komentářů ve zdrojovém souboru. Z ostatních řádků jsou extrahovány informace definující číslo portu, službu a transportní protokol rozřezáním textového řetězce, který reprezentuje daný řádek. Pokud obsahuje i komentář, je též získán a uložen v interních strukturách.

Automatické rozřezání využívá znak mezery jako dělícího symbolu a v následných cyklech dojde k určení, které získané segmenty jsou platné a popřípadě tvoří jedinou informaci. Celý proces přeskládá požadované údaje z formy zdrojového souboru do textového řetězce, který je následně transformován do interní struktury. Tato struktura bude blíže popsána v kapitole **4.4 Model**.

```
Původní řetězec [23]:
msp             18/tcp          # message send protocol

Výsledný řetězec:
msp//18//tcp//# message send protocol
```

Nakonec je vytvořený seznam uložen do lokálního souboru, aby při dalších startech aplikace nemuselo docházet k jeho opětovnému přepisu. Lokální soubor obsahuje řetězce, které pro transformaci do interního objektu nevyžadují speciální úpravu.

Poslední činností této metody je inicializace a propojení objektů, které slouží jako vyrovnávací paměť představující komunikační rozhraní mezi jednotlivými vlákny aplikace. Přesný popis, jak vlákna tuto strukturu využívají a jejich "přístupová práva" budou popsána v kapitole **4.1.2 Komunikace mezi vlákny**.

Hlavní vlákno je též odpovědné za řádné ukončení ostatních vláken. Přestože se nejedná o kritickou činnost, virtuální běhové prostředí dokáže samo zajistit jejich zastavení a zrušení, je nezbytná pro zajištění konzistence všech souborů aplikace, neboť některá vlákna mohou přistupovat do konfiguračních souborů.

```
private void performExit(){
    running = false;

    try {
          enforcement.join(500);
          rotor.join(500);
          snooper.join(500);
          updater.join(500);
          sender.join(500);
    } catch (InterruptedException e) {
          e.printStackTrace();
    } finally {
          System.exit(0);
    }
}
```

Před samotným vypnutím aplikace je skrze globální proměnnou *running* typu boolean předána zpráva všem vláknům, aby ukončila svoji činnost. Jelikož běží v nekonečném while cyklu zajišťující jejich aktivitu, dojde změnou výše uvedené proměnné k ukončení tohoto cyklu.

Následně jsou všechny vlákna postupně volána a skrze zděděnou metodu dochází k jejich bezpečnému ukončení. Zároveň ale není vhodné čekat na zaseknuté vlákno nepřiměřeně dlouho, proto je v rámci volání metody předán i časový okamžik určující, kdy nejpozději musí být vlákno ukončeno.

Ukončování vláken může vytvářet výjimky. Kvůli situaci aplikace však není nezbytné obsloužení těchto stavů a aplikace se tak okamžitě ukončí. Pokud by se daného chybového stavu nedosáhlo, dojde k vypnutí aplikace stejným způsobem.

4.1.1 Nastavení prostředí

Aplikace při každém spuštění, a v určitých okamžicích, kdy reaguje na uživatelské akce, potřebuje vystavět některé vnitřní proměnné a samozřejmě převzít správu nad firewallovým rozhraním iptables [14].

Mezi základní hodnoty patří informace spojené s konkrétním rozhraním síťové karty počítače. Aplikace vyžaduje, aby každý jednotlivý uživatelský účet byl propojen s platným rozhraním v okamžiku svého vytvoření. Tento údaj je následně uložen v rámci lokálních hodnot profilu a je použit při úvodním nastavení prostředí aplikace po načtení profilu.

```
private void prepareNIC(){
    String intName = null;
    if((intName = properties.getProperty(
                                    whoami + ".interface")) != null){
          nic = Analyzer.getNICByName(intName);
          if(properties.getProperty(whoami + ".interfaceMAC")
                                    == null)
                properties.setProperty(
                            whoami + ".interfaceMAC",
                            FormatUtils.mac(nic.getMac()));
    } else {
          boolean done = false;
          ...
```

Aplikace se nejdříve pokusí nastavit rozhraní podle jména obsaženého v properties souboru. V rámci podmínky se tedy kontroluje návratová hodnota při čtení z daného umístění, respektive obsah daného klíče v rámci souboru. Vrátí-li se požadovaný údaj, aplikace se pokusí načíst odpovídající rozhraní pomocí volání statické metody na Analytické třídě, která vrací interní objekt určený pro sdružení a správu všech požadovaných dat o síťovém hardwaru. Ty jsou následně použity pro kontrolu platného nastavení v rámci properties souboru. Pokud by chyběla požadovaná hodnota, tedy hardwarová adresa rozhraní, byla by okamžitě doplněna.

```
public static NetworkInterfaceCard getNICByName(String name){
    List<PcapIf> interfaces = getAllDevs();
    for(PcapIf i : interfaces)
          if(i.getName().equalsIgnoreCase(name))
                return new NetworkInterfaceCard(i);

    return null;
}
```

Pro načtení odpovídajícího síťového rozhraní, za předpokladu, že na počítači existuje více ať už fyzických nebo logických jednotek, stačí znát pouze označení daného rozhraní. Jedná se také o atribut volání dané metody.

I přes znalost jména neexistuje jednoduchý způsob, jak získat jediné konkrétní rozhraní ze všech dostupných přímým přístupem. Proto se využívá funkcionalita knihovny jNetPcap [20], která vrátí seznam objektů této knihovny reprezentující všechna dostupná síťová rozhraní místního počítače.

```
public static List<PcapIf> getAllDevs(){
    StringBuilder errorBuffer = new StringBuilder();
    List<PcapIf> interfaces = new ArrayList<>();
    List<PcapIf> interfacesToExport = new ArrayList<>();
    Pcap.findAllDevs(interfaces, errorBuffer);

    for(PcapIf i : interfaces){
          boolean rewrite = false;
          for(PcapAddr a : i.getAddresses())
                if(a.getAddr().getFamily() == PcapSockAddr.AF_INET)
                      rewrite = true;

          if(rewrite) interfacesToExport.add(i);
    }
```

```
        return interfacesToExport;
    }
```

Kvůli požadavků metody knihovny je potřeba vytvořit struktury pro zápis. Jedná se o seznam, do kterého mají být nalezené objekty zapsány, a pro případ chybového výstupu i nástroj pro vytvoření textového řetězce, zde zastoupený objektem StringBuilder.

Samotné volání *Pcap.findAllDevs(interfaces, errorBuffer)* následně naplní předaný seznam v případě bezchybného průběhu. V tomto okamžiku se ale jedná o všechna dostupná rozhraní v počítači, z nichž některá nemusejí vyhovovat potřebám aplikace. Je tedy nutné výsledný seznam projít a omezit hodnoty, které mají být vráceny.

V cyklu tedy dochází k průchodu všemi nalezenými objekty a kontrolují se jejich komunikační adresy. Jedno rozhraní může mít přiděleno několik adres v různých standardech, které nemusejí být v rámci aplikace platné. Pokud je u daného rozhraní nalezena adresa v požadovaném standardu, je objekt označen pro přepsání do výsledného seznamu. Po projití všech objektů je výsledný seznam vrácen.

Objekty typu *PcapIf* jsou však nesmírně robustné a získat konkrétní hodnoty bývá leckdy složité. Z toho důvodu je výsledný požadovaný objekt transformován do interní struktury aplikace, která je mnohem příznivější a přístupnější. Tato struktura bude blíže popsána v kapitole **4.4 Model**. Výběr objektu je proveden na základě zadaného jména v rámci volání metody.

Doposud byl popsán průběh, kdy dané rozhraní je zaznamenáno v rámci properties souboru s nastavením. Může ale nastat situace, kdy tomu tak nebude. Nejčastěji při úvodním spuštění aplikace a automatickém vytvoření primárního profilu.

```
            ...
            nic = Analyzer.getNICByName(intName);
            if(properties.getProperty(whoami + ".interfaceMAC")
                                    == null)
                  properties.setProperty(
                              whoami + ".interfaceMAC",
                              FormatUtils.mac(nic.getMac()));
      } else {
            boolean done = false;
            while (!done) {
                  NICDialog nicDialog = new NICDialog(owner);
                  nicDialog.startNew();
                  if (nicDialog.isOk()){
                        nic = nicDialog.getNic();
                        properties.setProperty(
            whoami + ".interface", nic.getName());
                        properties.setProperty(
            whoami + ".interfaceMAC", FormatUtils.mac(nic.getMac()));
                        done = true;
                  } else if(nicDialog.isShutdown())
                        performExit();
            }
      }
  }
```

V opačném případě dojde ke spuštění dialogového okna s výběrem dostupných síťových rozhraní počítače. Procesy a funkce spojené s tímto i dalšími dialogovými okny budou popsány v kapitole **4.3.3 Dialogy**.

Uživatel v rámci daného okna může vybrat jedno z dostupných rozhraní, může ukončit výběr bez volby, kdy bude okno okamžitě znovu zobrazeno, a nebo se rozhodnout pro okamžité ukončení aplikace. Poslední volba je nezbytná, pokud v rámci nabízených množností není k dispozici žádané rozhraní, například proto, že dosud nebylo aktivováno a nastaveno, a aplikace nemůže být spuštěna na jiném. Protože je rozhraní v rámci aplikace nezbytné, je nutné ukončit proces spouštění dokud situace v rámci operačního systému nebude napravena.

Pokud však uživatel učiní volbu a označí rozhraní, které má být střeženo programem, je objekt reprezentující toto rozhraní vrácen a uložen do globální proměnné *nic*. Současně dojde k uložení jména rozhraní do properties souboru, kde bylo v rámci přípravy prostředí původně hledáno. Dalším krokem je uložení fyzické adresy rozhraní. Tato hodnota se využívá hlavně v průběhu adaptace na místní síť, která bude probrána v rámci popisu odpovědného vlákna v kapitole **4.1.3 Vlákno "snooper"**, a v procesu analýzy, blíže popsaná v kapitole **4.1.4.3 Analýza**.

V dalším kroku přípravy prostředí převezme aplikace kontrolu nad firewallovým rozhraním iptables [14] a nastaví jeho chování podle dosavadních výsledků běhu aplikace.

```
appliesRules = bridge.loadList("./data/" +
                    properties.getProperty(whoami + ".file"));

bridge.forwardCommand(IPTablesRule.getFlushRule(
          IPTablesRule.IPTABLES_CHAIN_INPUT));
bridge.forwardCommand(IPTablesRule.getFlushRule(
          IPTablesRule.IPTABLES_CHAIN_OUTPUT));
bridge.forwardCommand(IPTablesRule.getPoliticRule(
          properties.getProperty(whoami + ".politics"),
          IPTablesRule.IPTABLES_CHAIN_INPUT));
```

Do globálního seznamu jsou načtena dosud vytvořená pravidla v rámci běhu aplikace nebo ručně, skrze uživatelské grafické rozhraní. Pravidla jsou ukládána v dynamicky vytvořených a spravovaných souborech pro každý profil odděleně. Umožňuje to jak přehlednější třídění, tak rychlé odlišení nastavení jednotlivých účtů.

Kvůli zajištění správné funkčnosti zadávaných pravidel, je nezbytné nejdříve vyčistit tabulky, které budou následně naplněny. K tomu slouží volání statické metody vytvářející tzv. flush pravidlo, které zcela vymaže obsah dvou tabulek v rámci struktury iptables [14]. Poslední krokem před samotným vkládáním je nastavení výchozí politiky podle hodnoty v možnostech uživatelského účtu v properties souboru.

```
for(String sRule:appliesRules)
    bridge.forwardCommand(new IPTablesRule()
                .constructPopulate(sRule)
                .getAppendRule());
```

Následně je každé pravidlo, uložené v rámci odpovídajícího souboru, načteno. Na jeho základě je vytvořen interní objekt, který slouží pro uchování, porovnání a přípravu

pravidel do tvaru akceptovatelného firewallovým rozhraním iptables [14]. Každé je pak skrze komunikační třídu předáno prostřednictvím příkazu v rámci prostředí shell operačního systému do struktur firewallu.

```
List<String> tempWhite = properties.getProperties(
                                        whoami + ".whitelist");
List<String> tempBlack = properties.getProperties(
                                        whoami + ".blacklist");

for(int i = 0; i < Math.max(tempWhite.size(), tempBlack.size());
                                                                i++){
    if(i < tempWhite.size()){
          String temp = tempWhite.get(i);
          bridge.forwardCommand(new IPTablesRule()
                .setAction(IPTablesRule.IPTABLES_ACTION_ACCEPT)
                .setSource(temp.substring(0, temp.indexOf("/")))
                .setsMask(temp.substring(temp.indexOf("/") + 1))
                .getInputRule());
    }

    if(i < tempBlack.size()){
          String temp = tempBlack.get(i);
          bridge.forwardCommand(new IPTablesRule()
                .setAction(IPTablesRule.IPTABLES_ACTION_DROP)
                .setSource(temp.substring(0, temp.indexOf("/")))
                .setsMask(temp.substring(temp.indexOf("/") + 1))
                .getInputRule());
    }
}
```

Poslední akce při přípravě prostředí podle uživatelského profilu je nastavení filtrů založených na seznamu povolených a zakázaných adres definovaných v rámci účtu. Jednotlivé záznamy jsou ukládány v rámci lokálních nastavení profilu v souboru properties. Pro jejich získání je využita nástavba nad strukturou properties, která dokáže jednotlivé hodnoty odlišit a vrátit jejich seznam. Blíže bude tato funkce, stejně jako další spojené s načítáním a správou hodnot v rámci souboru properties, probrána v kapitole **4.2.3 PropertyLoader**.

Pro zápis těchto pravidel je využito jediného cyklu, který bude hodnoty v obou seznamech procházet současně do dosažení velikosti většího z nich. Pro generování konkrétního pravidla se tedy kontroluje velikost průchozího indexu vůči velikosti konkrétního seznamu, protože může nastat situace, kdy jsou prvky definované pouze v jednom. Pokusy o přístup do druhého seznamu by v takovém případě vedly k havárii aplikace a vyhozením příslušné výjimky.

Pro konstrukci samotného pravidla je využito stejné struktury, která je určena pro automaticky generovaná pravidla. Odlišnost obou typů je pouze v obsahu a výsledném řetězci.

Záznamy v seznamu whitelist, povolené adresy, jsou transformovány prostřednictvím objektu IPTablesRule do akceptovatelné podoby s nastavenou akcí přijetí dané komunikace. Výsledné pravidlo je následně vloženo do struktur iptables [14] pro aktivaci. Záznamy ze seznamu zakázaných adres, blacklist, jsou vytvořeny

a aplikovány stejným způsobem, pouze s jinou definovanou akcí po obdržení odpovídajícího paketu a sice zahození dané zprávy.

Všechny výše popsané kroky a činnosti se automaticky provedou vždy při spuštění aplikace kvůli nastavení primárního účtu definovaného v rámci souboru properties. Druhým okamžikem, kdy jsou prováděny, je změna aktuálního profilu skrze akci v uživatelském rozhraní.

Proces spouštění aplikace dále pokračuje vytvořením uživatelských akcí a inicializací uživatelského rozhraní. Obě tyto činnosti budou blíže popsány v kapitole věnované uživatelského rozhraní programu **4.3 GUI** a jejích podkapitolách.

4.1.2 Komunikace mezi vlákny

Jelikož je hlavní činnost celé aplikace rozdělena do několika navzájem nezávislých vláken, je nezbytně nutný mechanismus pro vzájemnou komunikaci a synchronizaci jednotlivých vláken. Nástroje, které podporuje objektový programovací jazyk JAVA, nejsou dostatečné, aby se ospravedlnilo jejich použití, přestože to znamená i určité ústupky v rámci struktury.

První problém je synchronizace všech vedlejších vláken. Už v průběhu návrhu nástroje se však ukázalo, že tento konkrétní požadavek není ani zdaleka kritickým. Většina činnosti každého vlákna je zcela nezávislá na všech ostatních, časově i datově. Proto byla všechna vlákna tvořena jako čistě asynchronní, kdy se po volání či nějaké činnosti nemusí čekat na odpověď. I přesto byly okamžiky, kdy by na sebe vlákna musela spoléhat, minimalizovány a prakticky vyrušeny.

Zbývá tedy problém vzájemné komunikace vláken. Po odzkoušení několika možných způsobů na jednoduchém modelu pro testování komunikace, bylo nakonec rozhodnuto využít mechanismus podobný frontě zpráv.

```
private List<List<String>> ipTablesRulesList = new ArrayList<>();
private List<String> ipTablesRulesA = new ArrayList<>();
private List<String> ipTablesRulesB = new ArrayList<>();
private List<String> ipTablesRulesC = new ArrayList<>();
private List<String> ipTablesRulesD = new ArrayList<>();

private List<Integer> ipTablesRulesIndex = new ArrayList<>();
```

Tento mechanismus fronty zpráv je v rámci paměti aplikace prezentován ve formě skládaného seznamu seznamů *ipTablesRulesList*. Pro účel by stačilo, aby tento seznam obsahoval právě dva vnitřní seznamy, ale kvůli potřebě statistického hromadění počtu výskytu identického objektu komunikace i v rámci budoucího zachycování, byly implementovány dva dodatečné seznamy.

Současně je však potřeba zajistit, aby si vlákna navzájem nepřekážela a nepokoušela se editovat či přistupovat k obsahu, který by mohl být spravován jiným vláknem. Implementace zámků nebo časových razítek se ukázala jako složitá. Navíc je tu požadavek pro pravidelné vymazávání zachycené komunikace, která by se jinak mohla hromadit až do dosažení limitu dostupné paměti.

Obojí se podařilo vyřešit zavedením dodatečného seznamu, který ukazuje na dostupné seznamy v rámci *ipTablesRulesList*, do kterých může vlákno přistoupit. Tento

dodatečný seznam, struktura *ipTablesRulesIndex* tedy zajišťuje, aby si jednotlivá vlákna nepřekážela ale i smazání obsahu seznamu v okamžiku, kdy již nebude dále potřebný.

Do této fronty může přistupovat vlákno "snooper", která zanáší do jednotlivých seznamů nově zachycenou komunikaci, respektive její interpretaci v interním objektu aplikace.

```
private void importNewRule(IPTablesRule rule){
    String r = rule.toString();
    for(int i = 0; i < ipTablesRulesIndex.size() - 1; i++){
          int index = ipTablesRulesIndex.get(i);
          List<String> tempList = ipTablesRulesList.get(index);
          index = tempList.indexOf(r);
          if(index < 0){
                tempList.add(r);
                tempList.add("1");
          } else
                tempList.set(index + 1,
                      Integer.toString(
                            Integer.parseInt(
                                  tempList.get(index + 1)) + 1));
    }
}
```

Zápis je proveden předáním objektu reprezentujícího zachycenou komunikaci v rámci volání metody. Následně je tento objekt transformován do textového řetězce, který bude následně uložen.

Kvůli zanesení počtu výskytu identického paketu i do budoucího zachytávání je textový řetězec uložen celkem třikrát do tří po sobě jdoucích seznamů. Jejich indexy jsou získány ze seznamu indexů a následně je získán samotný seznam.

Dalším krokem je vyhledání potencionálně již existujícího záznamu. Pokud nebude nalezen, bude objekt ve formě textového řetězce uložen, do pozice hned za něj i počet výskytů identického paketu, a jelikož se jedná o první výskyt, bude vložena číslice 1. V opačném případě dojde pouze k aktualizaci počtu výskytu, který bude zvýšen o jedničku.

Seznam indexů slouží pro aktualizace vztahu jednotlivých seznamů vůči vláknům. Tento proces změny je svěřen speciálnímu vláknu a bude blíže popsán při popisu tohoto vlákna v kapitole **4.1.5 Vlákno "rotor"**.

Kromě této komplexnější předávací struktury, existují ještě dvě menší. Obě jsou představovány prostým seznamem, protože se neočekává možnost srážky záměrů dvou vláken. První z nich slouží pro přenos zachycené aktualizační zprávy, která byla zachycena. Vlákno "snooper" má oprávnění do tohoto seznamu vkládat nové zprávy, vždy je připojí na konec. Pokud není seznam prázdný, bude vlákno "updater" postupně odebírat položky ze začátku seznamu. Obě tyto činnosti ještě blíže budou popsány v kapitolách věnujících se odpovídajícím vláknům a sice **4.1.3 Vlákno "snooper"** a **4.1.6 Vlákno "updater"**.

Druhý seznam slouží pro odesílání vlastních aktualizačních zpráv, které budou výsledkem činnosti aplikace při vytváření pravidel pro firewallové rozhraní iptables [14]. Vlákno "enforcement" při provádění analýzy zachycené komunikace, může rozhodnout o nutnosti vytvoření pravidla. To je následně uloženo do místních struktur,

ale také do seznamu, který je následně zpracováván vláknem "sender", které jednotlivá pravidla zabalí do komunikačního paketu a odešle je do lokální sítě pro ostatní zařízení. Obě tyto činnosti budou blíže popsány v odpovídajících kapitolách **4.1.4 Vlákno "enforcement"** a **4.1.7 Vlákno "sender"**.

4.1.3 Vlákno "snooper"

Jedná se o primární vlákno, které je v rámci činnosti aplikace neustále v běhu. Všechna ostatní, včetně hlavního vlákna definujícího a obstarávajícího uživatelské rozhraní, mohou být po určitou dobu uspaná nebo ve stavu čekání na impuls / akci, kterou mají obsloužit. Pokud je však aktivní kontrola, kterou skrze grafické rozhraní lze přepínat, je toto vlákno vždy aktivní neboť čeká na příchozí nebo odchozí komunikační paket, procházející chráněným síťovým rozhraním počítače.

Odpovědnost vlákna spočívá v jediné činnosti a sice zachycení jakékoli komunikace na daném síťovém rozhraní, přepsání paketu do interní struktury aplikace pro další použití a její předání odpovědným vláknům, která se postarají o další zpracování. O rozdělení těchto činností bylo rozhodnuto po uvážení náročnosti zachycení zprávy a její následné analýzy kvůli určení návazné akce. Obě činnosti by mohly způsobit zpomalení vlákna, které by nemuselo být schopné zachycovat veškerý provoz procházející daným síťovým rozhraním.

```
public void run(){
    prepareNIC();

    if(nic != null){
          String rewriteAddress = FormatUtils.ip(nic.getAddress());
          StringBuilder errorBuilder = new StringBuilder();
          JPacketHandler<String> jph =
                      new JPacketHandler<String>(nic);

          while(running)
                if(runScan) {
                      …
                } else
                      try {
                            Thread.sleep(500);
                      } catch (InterruptedException e) {
                            e.printStackTrace();
                      }
```

Po spuštění vlákna dojde nejdříve k nastavení potřebných údajů ohledně chráněného síťového rozhraní. V rámci této přípravy je volána stejná metoda, jejíž činnost již byla popsána v kapitole **4.1.1 Nastavení prostředí**. Důvod pro tuto duplicitu je jednoduchý, může totiž dojít k problému při nastavení v rámci spouštění celé aplikace. Pro další činnost je však nezbytné, aby dané rozhraní bylo správně inicializováno. Přestože by nemělo dojít k situaci, kdy by vlákno chtělo přistupovat k danému internímu objektu před jeho inicializací, je vhodnější tento krok zopakovat.

Po úspěšné kontrole výsledku předchozího nastavení, jsou inicializovány lokální proměnné, které nesou informace nebo jsou samy důležité pro další činnost. Kvůli potřebě rozhodnout o směrování zachycené komunikace se nastaví adresa síťového

rozhraní do textové proměnné ve vhodnějším tvaru. Překlad adresy zajišťuje funkcionalita dostupná skrze rozhraní knihovny jNetPcap [20]. Následně je inicializován chybový výstup a objekt typu JPacketHandler, který nese odpovědnost za přepis údajů ze zachyceného komunikačního paketu do interní struktury. Tento objekt je vystavěn pouze pro čtení údajů ze zachyceného paketu. Jeho jediná funkční metoda je volána automaticky rozhraním knihovny v okamžiku zachycení paketu na síťovém rozhraní počítače. Blíže bude popsán v kapitole **4.2.1 JPacketHandler**.

Následně je spuštěn cyklus, který zajišťuje neustálou činnost vlákna. První podmínka určuje běh while cyklu pomocí proměnné typu boolean, která se může přepnout pouze při vypnutí aplikace skrze interní rozhraní. Následná podmínka už slouží pro ovládání běhu vlákna. Pokud je proměnná nastavena na hodnotu TRUE, dochází k čekání a zachycování komunikace, v opačném případě je vlákno uspáno před další kontrolou případné změny nastavení. Tato proměnná, *runScan*, je ovladatelná skrze uživatelské rozhraní, jak již bylo dříve v této kapitole zmíněno.

```
Pcap pcap = Pcap.openLive(
                nic.getName(),
                64 * 1024,
                Pcap.MODE_PROMISCUOUS,
                1000,
                errorBuilder);

if(pcap == null){
    runScan = false;
    setOutput("pcap is null");
} else {
    …
```

Následně dochází k otevření spojení, skrze které budou zachycovány veškeré průchozí pakety na daném síťovém rozhraní. Objekt typu Pcap je opět využíván skrze rozhraní knihovny jNetPcap [20] a je udržován v paměti do vypnutí aplikace.

Pro jeho vytvoření je využito statické volání, které se propojí se síťovým rozhraním podle jména, které ho zastupuje v operačním systému, zde získané z interního objektu *nic*. Další hodnota určuje maximální velikost paketu, která má být zachycena. Pokud by zachycený paket byl větší než zde nastavená hodnota, bude oříznut a předána bude část odpovídající udané velikosti. Jedná se o užitečný parametr, neboť umožňuje odtrhnout datovou část a soustředit se pouze na hlavičky jednotlivých komunikačních vrstev síťového modelu, které nesou podstatnou část informací o zachyceném paketu. Následně je do konstrukční metody předán parametr informující o chování objektu vůči síťovému rozhraní. Tato konkrétní hodnota značí, že veškerá komunikace bude předána dále bez filtrování podle určení. Za normálního nastavení jsou totiž zprávy neurčené pro dané rozhraní zahozeny, aniž by byly jakkoli zpracovány. Posledním parametrem je časová prodleva, která může zajistit zachycení více paketů současně před jejich následným zpracováním. Hodnota definuje čas v milisekundách před zpracováním zachyceného paketu. Následně je pouze předán chybový výstup pro zachycení případných chyb.

Je důležité zmínit, že většina nastavených údajů nemusí být vůbec využita, pokud to daný operační systém a síťové rozhraní nepodporují. Současně je funkčnost těchto

parametrů závislá na knihovnách v rámci systému, přičemž v některých verzích se mohou vyskytovat chyby bránící použitelnosti těchto nastavení. Na činnost objektu, tedy zachycení a přepsání, nemají tyto neduhy žádný větší vliv. [24]

V tomto okamžiku je důležité zkontrolovat, zda se podařilo získat platný objekt. Pokud ano, aplikace může pokračovat už v procesu samotného zachytávání. V opačném případě dojde k vypnutí zachycování a informování uživatele skrze grafické rozhraní.

```
    …
} else {
    pcap.loop(1, jph, "");
    boolean direction = !jph.getSource().equals(rewriteAddress);

    IPTablesRule rule = new IPTablesRule(
          jph.getSource(),
          jph.getDestination(),
          direction ?
                      IPTablesRule.IPTABLES_CHAIN_INPUT :
                      IPTablesRule.IPTABLES_CHAIN_OUTPUT,
          properties.getProperty(whoami + ".packetAction"),
          direction);
    if(direction) rule.setinInterfaceName(nic.getName());
    else rule.setoutInterfaceName(nic.getName());
    rule.setSourcePort(jph.getSourPort())
          .setDestinationPort(jph.getDestPort())
          .setProtocol(jph.getName())
          .setWhoami(whoami)
          .setMacSource(jph.getMacSou())
          .setMacDestination(jph.getMacDes());
    …
```

Zachycení probíhá skrze cyklus *pcap.loop*, kdy jako hlavní parametry jsou předány počet paketů pro zachycení a objekt, který slouží k jeho přepsání respektive zpracování. V tomto stavu vlákno setrvá dokud nedojde k zachycení průchozího paketu. Řízení je potom předáno objektu pro zpracování, zde je zastoupeno instancí typu JPacketHandler.

V dalším kroku je určen směr průchodu paketu síťovým rozhraním kvůli další návaznosti. Jelikož samotná struktura paketu nedokáže nést tuto informaci, je vytvořena na základě porovnání známé adresy rozhraní a zdrojové adresy zachyceného paketu. Tato informace je uložena ve formě proměnné typu boolean.

Následně je vytvořena instance interní proměnné typu IPTablesRule, která bude blíže popsána v kapitole **4.4 Model**. Instance je nejdříve vytvořena skrze konstruktor třídy, jehož parametry jsou získány z objektu typu JPacketHandler. Následně jsou dodatečné informace nastaveny skrze setter rozhraní objektu. Tento objekt je vytvořen, respektive jeho rozhraní, za využití návrhového schématu, který je označován jako "řetězení metod" [25] nebo poněkud komplexnější "plynulé rozhraní" [26].

```
    …
    if(jph.isUpdate() &&
                Boolean.parseBoolean(
                      properties.getProperty("receiveUpdates")))
          receivedUpdates.add(
                new IPTablesRule()
                      .constructPopulate(
                            jph.getUpdateRule())
```

```
                                    .setWhoami(whoami));
        else if(!jph.isUpdate() && !jph.isSendUpdate())
              importNewRule(rule);

        if(properties.getProperty(whoami + ".firstGateMAC") == null){
              String myMac = rule.isOuter() ?
                          rule.getMacDestination() : rule.getMacSource();
              String firtsGateMac = !rule.isOuter() ?
                          rule.getMacDestination() : rule.getMacSource();

              if(myMac.equals(FormatUtils.mac(nic.getMac())))
                    properties.setProperty(
                                whoami + ".firstGateMAC", firtsGateMac);
        }

        jph.reset();
        pcap.close();
    }
```

Po přepsání všech dostupných informací do interního objektu je nezbytné rozhodnout, co s daným otiskem paketu dále. Je nutné rozlišit, zda se jedná o standardní komunikační paket, nebo aktualizační zprávu odeslanou jiným programem v rámci lokální sítě. Pokud bude takový paket nalezen a současně bude v rámci nastavení aplikace povolena funkce příjmu aktualizačních zpráv, bude do fronty zpráv určených pro aktualizační vlákno zařazen nový objekt typu IPTablesRule vytvořený z textového řetězce doneseného aktualizačním paketem. Jedná se o jediný případ, kdy je převzat a využit i náklad komunikačního paketu, který v tomto případě nese pravidlo pro zanesení do firewallového rozhraní iptables [14] lokálního počítače.

Pokud by se nejednalo o zachycenou aktualizační zprávu, pořád se ještě může jednat o odeslanou. Proto se i tato podmínka kontroluje předtím, než je otisk paketu předán do seznamu, popřípadě je inkrementován jeho počet výskytů, určeného pro další analýzu a případné vytvoření pravidla pro firewall. Přidání do seznamu je provedeno skrze volání metody importNewRule, která již byla popsána v rámci kapitoly **4.1.2 Komunikace mezi vlákny**.

Ještě před vynulováním hodnot v rámci instance typu JPacketHandler a uzavřením záchytného rozhraní dochází k nastavení jedné z posledních možností aplikace. Pokud doposud není vytvořen záznam v souboru properties nesoucím informace o nastavení programu, bude probíhat kontrola paketů vzhledem k jejich předchozímu bodu v rámci lokální sítě. Zjistí-li se u paketu místo určení shodné s chráněným rozhraním, uloží se zdrojová MAC adresa z hlavičky paketu do souboru properties. Tato hodnota je využívaná v rámci analýzy pohybu paketů v místní síti a také k ověření skutečnosti, že paket nebyl nikým odchycen a případně pozměněn. Proces analýzy paketu bude blíže popsán v následující kapitole **4.1.4 Vlákno "enforcement"** věnované analytickému vláknu, které také zodpovídá za aplikování případných pravidel.

4.1.4 Vlákno "enforcement"

V této kapitole bude blíže popsáno analytické vlákno a jeho hlavní odpovědnost, správa firewallových pravidel. Tato konkrétní činnost je prováděna dvěma způsoby. Prvním je správa aktivních pravidel a jejich doplňování prostřednictvím analýzy

zachycených komunikačních paketů. V této fázi probíhá kontrola, která má zabránit vzniku duplicitního pravidla. Tato ochrana však dokáže tuto skutečnost zachytit pouze v případě vlastní tvorby pravidla. Pokud by bylo pravidlo přidáno v rámci propojení uživatelských účtů, je možné, že by bylo vloženo již obsažené.

Druhým je zjednodušení a určitá filtrace pravidel. Tato činnost se stane vždy pouze jednou a to buď po úvodním načtení profilu nebo jeho změně v průběhu aktivity programu. Tato činnost nabývá na důležitosti v okamžiku, kdy bude narůstat počet automaticky vygenerovaných pravidel, popřípadě vložených uživatelem, a ruční správa by se tak stávala nadmíru obtížnou.

4.1.4.1 Zjednodušení pravidel

Proces zjednodušení pravidel, která byla aplikovaná nebo vytvořena v rámci programu, ať už automaticky na základě analýzy nebo ručně vložena prostřednictvím uživatelského rozhraní, spočívá v porovnání seznamu s jeho vlastní kopií, kdy se hledají podobnosti mezi jednotlivými pravidly. Činností aplikace totiž mohou vznikat pravidla, která by byla určena k zablokování spojení se stejnou adresou, ale pokaždé na jiné technologii nebo portu. Proces zjednodušení by měl taková pravidla odhalit a sjednotit pod jediným globálním.

```
changedUserProfile = false;

List<String> tempStringList = bridge.loadList("./data/" +
                      properties.getProperty(whoami + ".file"));
List<IPTablesRule> tempIPTableRuleList = new ArrayList<>();

for(String s : tempStringList)
    tempIPTableRuleList.add(
                new IPTablesRule().constructPopulate(s));

int replaceThreshold = Integer.parseInt(properties.getProperty(
                           whoami + ".ruleReplacement"));
String indexes = "";
…
```

Prvním krokem je vypnutí přepínače, který celý proces může spustit. Jedná se o globální proměnnou *changedUserProfile* typu boolean. Následně je načtena nezávislá kopie aktivních pravidel ze souboru daného uživatelského účtu. Název souboru je získán prostřednictvím konfiguračního souboru properties. O samotné načtení jednotlivých záznamů se stará komunikační třída, která bude blíže popsána v kapitole **4.2.2 CommunicationBridge**.

Když je dostupný seznam pravidel, je nutné jeho transformování do seznamu interních objektů typu IPTablesRule. Tento přepis probíhá v rámci for-each cyklu, kdy je každý textový řetězec rozebrán a uložen v rámci struktury objektu IPTablesRule. Konkrétní detaily spojené s principy tohoto procesu budou blíže popsány v kapitole **4.4 Model**.

Poslední krok, před zahájením samotné kontroly a zjednodušování, je načtení prahové hodnoty, která určuje počet identických pravidel, která musí být nalezena pro jejich nahrazení.

```
…
for(int j = 0; j < tempIPTableRuleList.size(); j++){
    IPTablesRule rule = tempIPTableRuleList.get(j);

    if(!rule.getSource().equalsIgnoreCase(nic.getFormatAddress())
                        && !rule.getSource().isEmpty()){
          String ip = rule.getSource();
          int amount = 0;
          boolean first = true;

          for(int i = j + 1; i < tempIPTableRuleList.size(); i++){
                IPTablesRule tempRule = tempIPTableRuleList.get(i);
                if(tempRule.getSource().equalsIgnoreCase(ip) &&
                            !indexes.contains(Integer.toString(i))){
                      if(first){
                            indexes += (indexes.isEmpty() ? "" : ",")
                                        + Integer.toString(j);
                            amount++;
                            first = false;
                      }
                      indexes += (indexes.isEmpty() ? "" : ",")
                                  + Integer.toString(i);
                      amount++;
          }
    }
    …
```

Samotná kontrola probíhá pro každé pravidlo, které je obsaženo v souboru daného uživatelského profilu. Kontrolována jsou však pouze pravidla, která vznikla pro blokaci příchozí komunikace a zároveň mají definovanou zdrojovou adresu.

Pokud tyto podmínky objekt reprezentující pravidlo splní, dojde k přípravě prohledávání seznamu. Než bude samotné hledání zahájeno, je potřeba inicializovat některé proměnné. Jedná se o hodnotu ip adresy pravidla pro porovnávání, dále počítadlo identických výskytů a nakonec ukazatel, zda bylo nalezeno alespoň jedno shodné pravidlo.

Samotná kontrola je prováděna nad stejným seznamem pouze s posunutým indexem. Nemá smysl kontrolovat výskyt identického pravidla před výskytem aktuálního. Tento princip zároveň zaručuje, že nedojde k falešné identifikaci při nalezení stejné instance pravidla v rámci seznamu.

Pokud by bylo nalezeno další pravidlo s identickou ip adresou a současně již nebylo označeno pro nahrazení, dojde k jeho zapsání do dočasného ukazatele ve formě textového řetězce. V případě prvního nalezeného objektu dochází k dvojímu zapsání a sice jak samotného objektu, tak pravidla pro které byl nalezen. Při každém takovém záznamu dochází k inkrementaci počtu identických objektů. Tato činnost se opakuje, dokud není dosaženo konce seznamu.

Následně je nutné rozhodnout zda počet výskytů identického objektu přesáhl hraniční hodnotu.

```
    …
    if(amount > replaceThreshold){
          SimpleTableItemService stis =
                      new SimpleTableItemServiceImpl()
                            .setProperties(properties)
```

```
                        .setWhoami(whoami);

            List<SimpleTableItem> items = stis.getAll();
            boolean find = false;

            for(int i = 0; i < items.size(); i++){
                  SimpleTableItem item = items.get(i);

                  if(Analyzer.partOfAddress(
                              item.getAddressWithMask(), ip) == 1){
                        find = true;
                        i = items.size();
                  }
            }

            if(!find) stis.addNew(new SimpleTableItem(ip
                              + "/255.255.255.255", "blacklist"));
      } else
            for(int i = amount; i > 0; i--)
                  if(indexes.contains(","))
                        indexes = indexes.substring(
                                    0,
                                    indexes.lastIndexOf(","));
                  else if(!indexes.isEmpty()) indexes = "";
      }
}
…
```

V případě, že hraniční hodnoty nedosáhne, je nezbytné vynulovat dočasné hodnoty, aby nebyly následně chybně zpracovány. Je nutné z textového řetězce obsahujícího indexy do seznamu pravidel odstranit všechny záznamy, které do něj byly v rámci kontroly zaneseny.

Pokud bylo nalezeno množství identických pravidel převyšujících nastavenou hraniční hodnotu, dojde k vytvoření souhrnného pravidla pro blokaci celé adresy. Pro tento efekt je využit seznam uživatelem zakázaných adres, do nějž bude doplněno nové pravidlo. Nejdříve je ale nutné se ujistit, že už nebylo vloženo pravidlo, které by danou ip adresu též dokázalo zablokovat.

Pro získání platných zakázaných adres je využito třídy, která zajišťuje veškeré operace spojené se správou tohoto seznamu. Objekt *SimpleTableItemService* je určen převážně pro zobrazení zakázaných a povolených adres a obsluhu uživatelských akcí skrze grafické rozhraní. Tyto funkcionality se však výtečně hodí pro získání a úpravu seznamu i při této automatické kontrole. Princip funkce tohoto objektu bude blíže popsán v kapitole **4.3.3.1 TableService**.

Pro ověření, zda již daná adresa není blokována v rámci již existující zakázané adresy je využito funkce analytické třídy, která dokáže porovnat dvojice ip adres s daným rozsahem platnosti v podobě masky sítě.

```
String a1 = "";
String a2 = "";
String m = "";

if(addressWithMask.contains("/")){
    int index = addressWithMask.indexOf("/");
    a1 = addressWithMask.substring(0, index);
```

```
    m = addressWithMask.substring(index + 1);
    a2 = addressToCheck.substring(
          0, addressToCheck.indexOf("/") < 0 ?
          addressToCheck.length() : addressToCheck.indexOf("/"));
} else if(addressToCheck.contains("/")){
    int index = addressToCheck.indexOf("/");
    a1 = addressToCheck.substring(0, index);
    m = addressToCheck.substring(index + 1);
    a2 = addressWithMask.substring(
          0, addressWithMask.indexOf("/") < 0 ?
          addressWithMask.length() : addressWithMask.indexOf("/"));
} else return -1;

int ip = getFlatIntForAddress(a1);
int subnet = getFlatIntForAddress(a2);

if(ip < 0 || subnet < 0) return -1;

int mask = -1 << (32 - Integer.parseInt(m.length() < 4 ?
                m : numericMaskFromIpMask(m)));

if ((subnet & mask) == (ip & mask)) return 1;
else return 0;
```

Tato metoda v rámci analytické třídy přijme dvojici ip adres a podle jejich tvaru rozhodne o porovnání. Primárně přebírá masku sítě od první předané adresy, ale dokáže si poradit i v případě, kdy je maska předána až s druhou v pořadí. Pokud zadané adresy neobsahují požadovanou masku sítě, dochází k vrácení chybového stavu, jelikož metoda není schopna posoudit podobnost obou hodnot.

```
try {
    byte[] b = InetAddress.getByName(address).getAddress();
    return ((((int)b[0]) & 0xFF) << 24) |
           ((((int)b[1]) & 0xFF) << 16) |
           ((((int)b[2]) & 0xFF) << 8)  |
           ((((int)b[3]) & 0xFF) << 0);
} catch (UnknownHostException e) {
    e.printStackTrace();
}
return -1;
```

Posouzení je prováděno pomocí přepisu adresy do souhrnného čísla na základě velikosti daného oktetu a jeho pozice v řetězci. Pro každou možnou kombinaci tak vzniká jakýsi unikátní otisk, který je možné dále porovnat. Při samotném porovnání je využita maska sítě, která se binárně přičte k oběma otiskům a výsledek je porovnán.

Za předpokladu nenalezení odpovídajícího záznamu v rámci seznamu zakázaných adres, je nový záznam doplněn skrze volání metody pro přidání na instanci objektu *SimpleTableItemService*. Byl-li nalezen, není potřeba provádět žádné další akce.

```
…
String[] indexesField = indexes.split(",");
List<Integer> indexesIntField = new ArrayList<>();

for(int i = 0; i < indexesField.length; i++)
    if(!indexesField[i].isEmpty())
          indexesIntField.add(Integer.parseInt(indexesField[i]));
```

```
Collections.sort(indexesIntField);

for(int i = indexesIntField.size(); i > 0; i--)
    tempStringList.remove((int)indexesIntField.get(i - 1));

bridge.saveList("./data/" + properties.getProperty(
                whoami + ".file"), tempStringList, false);

prepareEnvironment();
```

Po skončení prohledávání a porovnávání seznamu aktivních firewallových pravidel a případném vytvoření zástupných globálních pravidel, je nezbytné odstranit předchozí záznamy. Kvůli tomuto účelu byly postupně ukládány pozice těchto pravidel v seznamu do textového řetězce. Nyní se textový řetězec rozřeže a přepíše do nového seznamu číselných indexů. Každý záznam je před transformací kontrolován zda doopravdy obsahuje hodnotu, aby se vyvarovalo případné výjimce. Následně je seznam seřazen podle velikosti daného indexu.

V cyklu, který prochází pozpátku skrze seznam indexů, jsou všechny označené položky postupně odmazávány z dočasného seznamu, který byl použit pro kontrolu. Jakmile je čištění dokončeno, je tento upravený seznam uložen do souboru uživatelského profilu skrze odpovědnou třídu.

Poslední krokem je volání metody, která provede inicializaci a nastavení prostředí podle nově upravených pravidel. Tento princip byl popsán v kapitole **4.1.1 Nastavení prostředí**.

Následuje ukázka efektu zjednodušení generovaných pravidel a výsledný efekt na seznam zakázaných adres. Veškerá data byla získána v rámci zkušebního provozu aplikace v prostředí virtuálního stroje s operačním systémem CentOS 6.5.

Ukázka souboru uživatelského profilu před zjednodušením pravidel:

```
INPUT -t filter -s 23.64.15.32/255.255.255.255 -d 192.168.200.130/255.255.255.255 -i eth0 -p Tcp
--sport 80 --dport 38512 -j DROP -whoami root $ms 00:50:56:E0:BE:2A $md 00:0C:29:28:97:5B
_Unknown MAC address _ possible spoof packet attack
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 239.255.255.250/255.255.255.255 -i eth0 -p
Udp --sport 1900 --dport 1900 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md 01:00:5E:7F:FF:FA
_Exceeded user's threshold limit 100 for packet's occurence by 102
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 255.255.255.255/255.255.255.255 -i eth0 -p
Udp --sport 57034 --dport 8612 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md FF:FF:FF:FF:FF:FF
_Neither service or port were unknown
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 255.255.255.255/255.255.255.255 -i eth0 -p
Udp --sport 57039 --dport 8612 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md FF:FF:FF:FF:FF:FF
_Neither service or port were unknown
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 255.255.255.255/255.255.255.255 -i eth0 -p
Udp --sport 51464 --dport 8612 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md FF:FF:FF:FF:FF:FF
_Neither service or port were unknown
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 255.255.255.255/255.255.255.255 -i eth0 -p
Udp --sport 51469 --dport 8612 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md FF:FF:FF:FF:FF:FF
_Neither service or port were unknown
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 255.255.255.255/255.255.255.255 -i eth0 -p
Udp --sport 51474 --dport 8612 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md FF:FF:FF:FF:FF:FF
_Neither service or port were unknown
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 255.255.255.255/255.255.255.255 -i eth0 -p
Udp --sport 51479 --dport 8612 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md FF:FF:FF:FF:FF:FF
_Neither service or port were unknown
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 255.255.255.255/255.255.255.255 -i eth0 -p
Udp --sport 65239 --dport 8612 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md FF:FF:FF:FF:FF:FF
_Neither service or port were unknown
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 255.255.255.255/255.255.255.255 -i eth0 -p
Udp --sport 65244 --dport 8612 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md FF:FF:FF:FF:FF:FF
_Neither service or port were unknown
```

Ukázka seznamu zakázaných adres před zjednodušením pravidel:

```
root.blacklist=1.2.3.4/255.255.255.255,1.11.22.4/255.255.255.255
```

```
Ukázka souboru uživatelského profilu po zjednodušení pravidel:
INPUT -t filter -s 23.64.15.32/255.255.255.255 -d 192.168.200.130/255.255.255.255 -i eth0 -p Tcp
--sport 80 --dport 38512 -j DROP -whoami root $ms 00:50:56:E0:BE:2A $md 00:0C:29:28:97:5B
_Unknown MAC address _ possible spoof packet attack

Ukázka seznamu zakázaných adres po zjednodušení pravidel:
root.blacklist=1.2.3.4/255.255.255.255,1.11.22.4/255.255.255.255,
192.168.200.1/255.255.255.255
```

4.1.4.2 Tvorba pravidel

Aplikace dokáže vytvářet a aplikovat firewallová pravidla prostřednictvím rozhraní iptables [14], která vznikají automatickou činností. Důležitou roli v tomto procesu hraje samozřejmě automatická analýza veškeré zachycené komunikace, respektive jednotlivých otisků komunikačních paketů. Při tvorbě a vložení nového pravidla, je však nezbytně nutné se ujistit, že již neexistuje ekvivalentní. Nejedná se ani tak o problém výkonnosti, byť by výkon firewallu mohl klesat s narůstajícím počtem duplicitních pravidel, ale hlavně o skutečnost udržitelné správy. Pokud by se každé pravidlo v seznamu vyskytovalo vícekrát, bylo by nesmírně obtížné i jeho prosté odstranění, protože by bylo nezbytné nalézt všechny jeho mutace / kopie a každou vymazat.

```
while(running){
    if(changedUserProfile) simplifyActiveRules();

    if(ipTablesRulesList.get(
          ipTablesRulesIndex.get(
                ipTablesRulesIndex.size() - 1))
                      .size() > 0){
          List<String> rulesToImport =
                ipTablesRulesList.get(
                      ipTablesRulesIndex.get(
                            ipTablesRulesIndex.size() - 1));
          while(rulesToImport.size() > 0 && stabile){
                …
          }
    } else
          try {
                Thread.sleep(500);
          } catch (InterruptedException e) {
                e.printStackTrace();
          }
    }
}
```

Vnější while cyklus slouží pro zastavení vlákna při vypnutí aplikace skrze přepínač typu boolean. Tento způsob kontroly je společný všem vláknům.

Následně je ověřováno, zda nedošlo ke změně uživatelského profilu. V kladném případě je zahájen proces zjednodušení pravidel, který byl popsán v předchozí kapitole.

Následuje podmínka, která určuje, zda již jsou dostupné otisky komunikačních paketů pro kontrolu a případné vytvoření pravidel. Ke kontrole je využíváno struktury pro komunikaci mezi vlákny, jejíž konstrukce a princip byl popsán v kapitole **4.1.2 Komunikace mezi vlákny**. Ověřuje se velikost seznamu umístěného na posledním indexu v seznamu těchto ukazatelů. Pokud by tato struktura byla prázdná, bude vlákno

uspáno. V opačném případě získá instanci daného seznamu a spustí se vnitřní while cyklus, který zajišťuje jeho vyprázdnění. Druhá podmínka slouží pro pozastavení vykonávání vlákna v případě přepínání kontextu uživatelského profilu, které bude popsáno v rámci kapitoly **4.1.5 Vlákno "rotor"**.

```
IPTablesRule rule = new IPTablesRule()
                  .constructPopulate(rulesToImport.remove(0));
String dimension = rulesToImport.remove(0);

if(!rule.toString().isEmpty()){
      boolean newRule = true;
      for(String s:appliesRules)
            if(newRule && rule.toString().equals(
                        new IPTablesRule()
                              .constructPopulate(s)
                              .setDescription("")
                              .toString()))
                  newRule = false;

      if(newRule && Analyzer.performAnalyze(
            rule.setDimension(dimension),
            whoami,
            properties,
            serviceAndPort)){
                  appliesRules.add(
                        rule.setDescription(
                                    Analyzer.getReason())
                              .toString());
                  saveRule(rule);
                  …
```

Z instance seznamu otisků komunikačních paketů je získáno a vytvořeno pravidlo a současně je do proměnné uložen i počet výskytů identického objektu v průběhu zachytávání a ukládání komunikace. Následně probíhá ověření, že vytvořené pravidlo obsahuje údaje, které lze porovnávat, tedy že nebyl vytvořen prázdný objekt.

Již bylo zmíněno, že podstatné při vytváření nového pravidla je nutné zajistit jeho jedinečnost v rámci firewallu. Prvním krokem ještě před samotnou analýzou je tedy vyhledání případného identického záznamu. K ověření této skutečnosti je využíván seznam aktuálně aplikovaných pravidel, který byl nastaven v rámci přípravy prostředí podle údajů uživatelského profilu. Pokud by bylo nalezeno shodné pravidlo, nastaví se proměnná typu boolean, která zabrání dalšímu zpracování. Jedná se především o zabránění provedení analýzy na daném otisku paketu.

Za předpokladu, že případné pravidlo není dosud obsaženo, přichází na řadu analýza otisku komunikačního paketu. Samotný proces analýzy bude popsán v následující kapitole.

Kladný výsledek označí paket jako potencionálně závadný a následně dojde k vytvoření a aplikování příslušného pravidla, které se doplní o slovní popis důvodu zahrnutí do firewallu. Tento údaj je užitečný hlavně v okamžiku, kdy uživatel prochází platná pravidla, protože ho informuje o výsledku provedené analýzy.

Následně je pravidlo uloženo do souboru odpovídajícího uživatelského profilu prostřednictvím volání metody *saveRule()*.

```
private void saveRule(IPTablesRule rule){
    List<String> multiUser = properties.getProperties(
                whoami + ".multiUser");

    for(String user:multiUser){
          List<String> temp = bridge.loadList("./data/" +
                properties.getProperty(user + ".file"));
          if(!temp.contains(rule))
                bridge.appendLineToList("./data/" +
                      properties.getProperty(user + ".file"),
                      rule.toString());
    }
}
```

Výše uvedená metoda slouží pro uložení nového pravidla do souboru k ostatním. Jelikož ale aplikace podporuje provázání jednotlivých uživatelských účtů, kdy následně dochází k ukládání nových pravidel i do seznamů aktuálně neaktivních profilů, je nutné zohlednit tuto činnost.

V úvodu dochází tedy k načtení seznamu propojených účtů, toto spojení je vždy pouze jednosměrné z aktuálního profilu do ostatních. Pokud by se mělo jednat o obousměrný vztah, je nezbytné nastavit oba případné účty. Tento záznam je získán ze souboru properties a vždy obsahuje alespoň jedno jméno uživatelského profilu a sice vlastní účet.

Pro každý nalezený propojený profil dojde k vyhledání jeho souboru se seznamem aktivních pravidel. Jakmile je seznam z daného souboru získán, je zkontrolována existence odpovídajícího pravidla. Pokud by již bylo v seznamu obsaženo, nedojde k novému uložení. V opačném případě je nový záznam připojen k existujícím v daném konfiguračním souboru.

```
                …
                if(Boolean.parseBoolean(
                            properties.getProperty("sendUpdates")))
                      rulesToExport.add(rule);

                String appendRule = rule.getAppendRule();
                setOutput(appendRule);
                bridge.forwardCommand(appendRule);
          }
    }
```

Po uložení daného pravidla do všech odpovídajících záznamů je zkontrolováno nastavení aplikace. Pokud je povolena funkce odesílání aktualizačních paketů, dojde k předání nového pravidla do seznamu pro odeslání, který bude v co nejkratším čase zpracován a odeslán do místní sítě. Tento proces i samotné vytvoření komunikačního paketu budou popsány v kapitole **4.1.7 Vlákno "sender".**

Posledním krokem je aplikování daného pravidla do firewallového rozhraní iptables [14] skrze odpovídající metodu komunikační třídy. Současně dojde k zobrazení vloženého záznamu na hlavní obrazovce aplikace, kde slouží jako upozornění pro uživatele o činnosti programu.

Následuje ukázka pravidla, které vzniklo přímou činností aplikace v rámci zkušebního běhu v prostředí virtuálního počítače na operačním systému CentOS 6.5.

```
INPUT -t filter -s 23.64.15.32/255.255.255.255 -d
192.168.200.130/255.255.255.255 -i eth0 -p Tcp --sport 80 --dport
38512 -j DROP -whoami root $ms 00:50:56:E0:BE:2A $md
00:0C:29:28:97:5B _Unknown MAC address _ possible spoof packet
attack
```

4.1.4.3 Analýza

Nejdůležitějším aspektem při rozhodování aplikace o zanesení nového pravidla na základě otisku zachycené komunikace je proces jeho analýzy. Tato činnost se skládá z několika fází, které jsou setříděny v pořadí, které by mělo přinést maximální odfiltrování prvků před dosažením výpočetně náročnějších akcí. V této kapitole bude popsán samotný proces analýzy i jeho možné výsledky, které jsou uživateli sdělovány pouze v případě nalezení potencionálně nebezpečného otisku komunikačního paketu, který byl v rámci činnosti zachycen.

Odpovědnost za analýzu nese samostatná třída, jejíž některé schopnosti již byly popsány v rámci kapitoly **4.1.1 Nastavení prostředí a** kapitoly **4.1.4.1 Zjednodušení pravidel**. Tato třída bude blíže popsána v samostatné kapitole **4.2.4 Analyzer**, kde budou představeny všechny její možnosti.

Proces analýzy byl vytvořen s maximální možností rozhodnutí a proto dokáže vrátit pouze dva typy výsledku a to pozitivní a negativní označení zkoumaného otisku paketu. Každá kontrola, která proběhne a skončí jedním z výše uvedených výsledků, okamžitě přeruší celý proces, čímž dochází i k úspoře výpočetního výkonu, protože již rozhodnutý paket není potřeba dále zkoumat.

```
rule.setOuter(rule.getChain().equalsIgnoreCase(
                IPTablesRule.IPTABLES_CHAIN_INPUT));

List<String> tempWhite = properties.getProperties(
                whoami + ".whitelist");
for(String s : tempWhite)
    if(partOfAddress(
                s,
                rule.getSource() + "/" + rule.getsMask())
                == 1)
            return false;

List<String> tempBlack = properties.getProperties(
                whoami + ".blacklist");
for(String s : tempBlack)
    if(partOfAddress(
                s,
                rule.getSource() + "/" + rule.getsMask())
                == 1) {
            reason = "Inclueded in user's blacklist";
            return true;
    }
…
```

V prvním okamžiku je nezbytné označit si kontrolované pravidlo, aby bylo jasné, kterým směrem bylo zachyceno. Jedná se o jednoduché rozlišení odchozí a příchozí komunikace.

Následuje kontrola pro ujištění, že zachycený paket nespadá do adresného prostoru uživatelem povolených nebo zakázaných adres. V obou případech se využívá metody pro porovnání dvou adres s daným rozsahem zadaným pomocí masky sítě. Princip tohoto postupu byl již popsán v rámci kapitoly **4.1.4.1 Zjednodušení pravidel**.

Pokud by bylo zjištěno, že paket spadá do oblasti označené jako povolené, je okamžitě ukončena analýza se záporným výsledkem, paket není nutné považovat za nebezpečný. V případě shody s adresou v seznamu zakázaných, je daný paket označen pozitivně a do proměnné je uložen důvod tohoto rozhodnutí. Následně je odpovědným vláknem vytvořeno a aplikováno odpovídající pravidlo, tato činnost byla popsána v předchozí kapitole.

```
…
if(properties.getProperty(
          whoami + ".interfaceMAC").equals(rule.isOuter() ?
          rule.getMacDestination() : rule.getMacSource()))
    if(!properties.getProperty(
                whoami + ".firstGateMAC").equals(!rule.isOuter() ?
                rule.getMacDestination() : rule.getMacSource())){
          reason =
                "Unknown MAC address, possible spoof packet attack";
          return true;
    }

if(Integer.parseInt(rule.getDimension()) >
          Integer.parseInt(properties.getProperty(
                whoami + ".threshold"))){
    reason = "Exceeded user's threshold limit " +
                properties.getProperty(whoami + ".threshold") +
                " for packet's occurence by " + rule.getDimension();
    return true;
}
…
```

V další fázi, za předpokladu, že paket doposud nebyl nijak označen, se kontroluje předchozí lokace v místní síti. V tomto okamžiku je použit údaj získaný při zachytávání paketů určených pro chráněné síťové rozhraní počítače a sice zdrojová hardwarová adresa. Samozřejmě se očekává, že zdrojová komunikace už v okamžiku přizpůsobení není závadná.

Nejdříve se kontroluje, zda paket je určen přímo pro dané rozhraní počítače na základě jeho cílové respektive zdrojové MAC adresy podle jeho směrování. Tento postup snižuje pravděpodobnost chybného označení paketů, které byly odeslány na broadcast adresu místní sítě. Následně je zkontrolována i zbývající MAC adresa paketu a porovnána s uloženým otiskem prvního zařízení v rámci sítě. Typicky by se jednalo o zařízení, které nějakým způsobem rozděluje místní síť nebo ji odděluje od internetu. Při zjištění rozdílu bude paket označen jako závadný a důvod tohoto rozhodnutí uložen do proměnné.

Následně je kontrolován počet výskytů identického paketu. Tato hodnota je porovnávána s hraniční, která je uložena v rámci nastavení aktivního uživatelského profilu. Pokud by došlo k jejímu překročení, je paket označen jako závadný a důvod

uložen do proměnné. V tomto případě je v rámci důvodu sdělen i přesný počet výskytů a aktuálně platná mezní hodnota.

```
…
String service = rule.getProtocol();
String port = rule.isOuter() ?
          rule.getSourcePort() : rule.getDestinationPort();

if(service.isEmpty() || port.isEmpty()){
    reason = "Neither service or port weren't specified";
    return true;
}

if(servicesAndPorts.contains(
          new ServiceAndPort().setPort(port).setService(service)))
    return false;
else {
    for(ServiceAndPort temp : servicesAndPorts)
          if(temp.getPort().equalsIgnoreCase(port) &&
                      (temp.getService().equalsIgnoreCase(service) ||
                      temp.getProtocol().equalsIgnoreCase(service)))
                return false;

          reason = "Neither service or port were unknown";
          return Boolean.parseBoolean(
                properties.getProperty(whoami + ".unknownService"));
          }
…
```

Poslední fáze kontroly testuje uvedený protokol a příslušný port vzhledem k seznamu platných v rámci aplikace. Nejdříve jsou tyto požadované údaje získány a uloženy do lokálních proměnných. Dále je otestováno, zda obě hodnoty byly v rámci paketu získány. V opačném případě je paket označen jako závadný.

Byli-li obě požadované informace získány, číslo odpovědného portu a použitý protokol, provede se vyhledání odpovídajícího interního objektu ze seznamu, který reprezentuje množinu akceptovatelných dvojic portu a služby. Tento interní objekt bude blíže popsán v kapitole **4.4 Model**. Je-li nalezen odpovídající objekt, paket je označen jako bezpečný.

Pokud však odpovídající objekt nalezen nebude, spustí se druhá fáze této kontroly. V jejím průběhu se načte každá akceptovatelná dvojice a bude docházet k přímému porovnání odpovídajících částí. Důvod pro tento krok je eliminace co největšího počtu falešných označení v případě chybného nenalezení odpovídajícího záznamu. Je-li tedy objevena shoda, je paket označen jako nezávadný. Pokud však dojde k projití celého seznamu bez kladného výsledku, je nezbytné rozhodnout o osudu paketu. Tato volba je dostupná skrze uživatelské nastavení profilu, kdy je možné určit výchozí akci při tomto výsledku. Je totiž možné, že kontrolovaný paket není závadný, pouze v seznamu chybí data. Výsledek této poslední kontroly je tak závislý na konkrétním nastavení uživatelského účtu aplikace.

4.1.5 Vlákno "rotor"

Hlavním účelem vlákna je podpora komunikace mezi vlákny. To byl primární důvod jeho vytvoření. Následně se jeho odpovědnost rozšířila i o přepínání kontextu uživatelského profilu. Z původně doplňkového vlákna se tak stala důležitá součást celé aplikace.

```
setInterval();
calendar.set(Calendar.MINUTE, calendar.get(
                              Calendar.MINUTE) + refreshInterval);
while(running){
    if(calendar.before(Calendar.getInstance())){
          setCalendar(Calendar.getInstance());
          ipTablesRulesIndex.add(0, ipTablesRulesIndex.remove(
                      ipTablesRulesIndex.size() - 1));
    } else {
          …
```

Primární odpovědnost vlákna je spojena s přetáčením seznamu obsahujícího zachycené objekty. Tímto krokem jsou zpřístupněny otisky komunikačních paketů, které byly vytvořeny činností vlákna "snooper" pro následnou analýzu, za níž odpovídá analytické vlákno "enforcement". Jelikož by přesunutí celých seznamů v dané struktuře bylo příliš náročné a mohlo by způsobit i výrazné zpomalení běhu, přetáčí se pouze indexy ukazatele do seznamu seznamů. Princip této komunikace byl detailně popsán v kapitole **4.1.2 Komunikace mezi vlákny**.

Před spuštěním cyklu, který zajišťuje činnost vlákna, je nezbytné nastavit interval pro přetočení a také časovou značku. Časová značka je vytvořena na základě okamžitého času a přičtení požadovaného intervalu.

Jakmile je aktuální čas větší než nastavená značka, dojde k přetočení ukazatelů do seznamu seznamů. Současně je aktualizována hodnota časové značky pro další aktivitu. Přetočení probíhá jednoduchým vyjmutím posledního prvku ze seznamu indexů, ukazatelů, a jeho umístění na začátek této struktury. V důsledku se tak změnilo i pořadí jednotlivých seznamů zachycené komunikace, protože se k nim přistupuje skrze tyto ukazatele.

```
          …
          try {
                Thread.sleep(1000);
          } catch (InterruptedException e) {
                e.printStackTrace();
          }

          if(!newUser.isEmpty() && !newUser.equals(whoami)){
                stabile = false;

                checkUserExist(newUser);
                whoami = newUser;
                appliesRules = bridge.loadList("./data/" +
                      properties.getProperty(whoami + ".file"));
                setInterval();
                prepareEnvironment();

                changedUserProfile = true;
```

```
                stabile = true;
                }
            }
        }
```

Pokud zatím nebylo dosaženo časového okamžiku pro přetočení, změnu pořadí jednotlivých ukazatelů, vlákno se ocitne v paralelní větvi. Nejdříve je vlákno dočasně uspáno. Následně se kontroluje, zda nedošlo ke změně uživatelského profilu. Tento krok je zprostředkován globální proměnnou, kterou modifikuje uživatel skrze akci definovanou v rámci grafického rozhraní. Tyto akce budou blíže popsány v kapitole **4.3 GUI**.

Došlo-li ke změně aktivního uživatelského profilu, je nezbytné provést i změnu kontextu aplikace, protože každý účet může být nastaven zcela odlišně a samozřejmě může obsahovat odlišná pravidla chování firewallu. Prvním krokem je upozornění analytického vlákna skrze přepínač, aby přerušilo svoji činnost.

Následuje nezbytná kontrola uživatelského účtu. V případě chybějícího účtu, což by nemělo být možné, dojde k jeho vytvoření a úvodnímu nastavení. Obě tyto činnosti budou blíže popsány v kapitole **4.3.1 Uživatelské účty**. Dále je modifikována proměnná ukazující na aktivní účet podle jeho jména a načtena platná pravidla daného profilu. Vlákno si následně nastaví nový interval pro změnu pořadí ukazatelů podle lokálního nastavení. Posledním krokem při přepínání je nastavení prostředí skrze volání metody *prepareEnvironment()*. Tato činnost byla popsána v kapitole **4.1.1 Nastavení prostředí**.

Následně je upozorněno na změnu profilu pro aktivaci funkce zjednodušení aplikovaných pravidel a uvolněn zámek omezující činnost analytického vlákna.

4.1.6 Vlákno "updater"

Doposud byla popsána vlákna, bez jejichž přímé činnosti by aplikace nemohla řádně fungovat a plnit svůj účel, tedy analýzu a případnou tvorbu pravidel pro firewallové rozhraní iptables [14]. V této a následující kapitole budou představena zbývající dvě vlákna, která mají na chod aplikace spíše doplňující a rozšiřující vliv. Pro její samotný chod však již nejsou kriticky důležitá.

Prvním takovým vláknem je "updater", který nese odpovědnost za aplikování zachycených aktualizačních pravidel. Bylo nezbytné tuto činnost oddělit, protože aktualizační zpráva nevyžaduje analýzu, stačí její aplikování mezi aktivní pravidla a zanesení do potřebných datových struktur.

```
if(receivedUpdates.size() > 0){
    while(receivedUpdates.size() > 0){
          boolean performUpdate = true;
          IPTablesRule rule = receivedUpdates.remove(0);

          if(rule.getinInterfaceName().isEmpty())
                rule.setoutInterfaceName(nic.getName())
                            .setSource("").setsMask("");
          else rule.setinInterfaceName(nic.getName())
                            .setDestination("").setdMask("");

          for(String r:appliesRules)
                if(performUpdate && new IPTablesRule()
```

```
                        .constructPopulate(r)
                              .getAppendRule()
                                    .equals(
                                          rule.getAppendRule()))
                        performUpdate = false;

            if(performUpdate){
                  bridge.forwardCommand(rule.getAppendRule());
                  appliesRules.add(rule.toString());
                  saveRule(rule);
            }
      }
} else
      try {
            Thread.sleep(1000);
      } catch (InterruptedException e) {
            e.printStackTrace();
      }
```

Nejdříve se kontroluje, zda jsou dostupné aktualizace. Pokud by žádné nalezeny nebyly, vlákno se před další kontrolou uspí.

Následně jsou procházena všechna dostupná pravidla pro aplikování získaná ze zachycených aktualizačních zpráv. Při získání instance daného pravidla je odstraněno ze seznamu. Jelikož pravidlo bylo získáno z jiného počítače v rámci místní sítě, je nezbytná jeho drobná úprava. Nastavené rozhraní, na němž bylo aplikováno pravidlo na původním počítačovém systému, se přepíše platným chráněným síťovým rozhraním místního počítače. Tím se přizpůsobí aktualizační pravidlo lokálnímu systému. Zároveň je nezbytné odstranit adresu počítače, který pravidlo vytvořil, protože pakety určené pro lokální stroj nebudou tuto adresu obsahovat. Podle směru paketu se jednoduše vynuluje zdrojová respektive cílová ip adresa pravidla.

Jelikož aplikace může sama v rámci lokálních zdrojů tvořit pravidla, která by se mohla shodovat s předaným, je nezbytná kontrola na existenci duplicitního záznamu. Pokud bude nalezeno již existující pravidlo, nastaví se přepínač typu boolean, aby se zabránilo jeho aplikování. Nové pravidlo, myšleno jako unikátní v porovnání se seznamem aktivních, je možné aplikovat a uložit do potřebných struktur skrze stejné metody využívané vláknem "enforcement". Pravidlo je skrze komunikační třídu přidáno mezi aktivní pravidla firewallového rozhraní iptables [14]. Následně dojde k jeho uložení do seznamu aktivních, který je udržován v rámci paměti aplikace. Posledním krokem je uložení pravidla do odpovídajících souborů. I v tomto případě je bráno v potaz možné propojení účtů, které je ošetřeno voláním metody *saveRule()*.

4.1.7 Vlákno "sender"

Poslední vlákno "sender" nese odpovědnost za odesílání aktualizačních paketů nesoucích nově vytvořená pravidla do lokální sítě. Jedná se o užitečnou vlastnost, která umožňuje informovat všechna zařízení s identickou aplikací v místní síti o nalezení potencionálně nebezpečné komunikace.

```
if(rulesToExport.size() > 0 && nic != null){
    StringBuilder errorBuffer = new StringBuilder();
    Pcap pcap = Pcap.openLive(
```

```
                nic.getName(),
                64 * 1024,
                Pcap.MODE_NON_BLOCKING,
                1000,
                errorBuffer);

    int secondChance = 0;

    while(rulesToExport.size() > 0){
          IPTablesRule rule = rulesToExport.remove(0);
          String message = rule.setWhoami("netUpdate").toString();
          JPacket packet = constructIpTcpPayloadPacket(message);
          …
```

Prvním krokem je nezbytnost vytvořit objekt typu Pcap z prostředků knihovny jNetPcap [20] pro navázání spojení s daným síťovým rozhraním počítače. Nastavení je identické s popsaným postupem v kapitole **4.1.3 Vlákno "snooper"** s jediným rozdílem, spojení s daným rozhraním je otevřeno v neblokujícím režimu.

Následně dochází k procházení seznamu pravidel určených pro odeslání. Při vyzvednutí dané instance je současně ze seznamu odstraněno. U daného pravidla dojde k úpravě vlastníka, aby se na druhém počítači dalo odlišit místní pravidlo od poslaného. Následně je nutné sestavit komunikační paket a připojit k němu požadované pravidlo.

```
byte[] readyMessage = null;
try {
    readyMessage = message.getBytes("UTF-8");
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
}
int packetSize = 34 + readyMessage.length;
JPacket packet = readyMessage != null ?
                new JMemoryPacket(packetSize) : null;

if(packet != null){
    packet.order(ByteOrder.BIG_ENDIAN);
    packet.setUShort(12, 0x0800);
    packet.setUByte(14, 0x04 | 0x05);
    packet.scan(JProtocol.ETHERNET_ID);

    Ethernet ethernet = packet.getHeader(new Ethernet());
    ethernet.source(nic.getMac());
    ethernet.checksum(ethernet.calculateChecksum());

    Ip4 ip4 = packet.getHeader(new Ip4());
    Payload payload = packet.getHeader(new Payload());

    if(ethernet == null || ip4 == null || payload == null)
                return null;

    ip4.length(packetSize - ethernet.size());
    ip4.ttl(32);
    ip4.destination(nic.getBroadcast());
    ip4.source(nic.getAddress());
    ip4.flags(Ip4.FLAG_DONT_FRAGMENT);
    ip4.checksum(ip4.calculateChecksum());

    payload.setByteArray(0, readyMessage);
    }
```

```
return packet;
```

Zpráva se překóduje do bitové podoby, aby bylo možné její připojení do těla generovaného paketu. Následně se nastaví délka datového paketu podle rozsahu přenášené zprávy.

Paket je vytvářen skrze objekt třídy jNetPcap [20]. Jelikož převod zprávy se nemusí vždy podařit, je nezbytné zkontrolovat výsledek. Pokud by došlo k problémům, zastaví se tvorba paketu v tomto bodě.

Za předpokladu úspěšného nastavení a vytvoření objektu paketu, je nutné nastavit požadované vlastnosti, aby bylo možné odeslat paket skrze síťové rozhraní počítače. Nutné je nastavit datovou hlavičku paketu, následně ethernetovou a ip hlavičku. Konkrétní nastavení všech odpovídajících hodnot bylo inspirováno návodem na oficiálních stránkách knihovny [27] a podnětnými radami na webu pro řešení problémů [28].

Pokud proběhlo úvodní nastavení všech zmíněných hlaviček v pořádku, jednotlivé hlavičky byly inicializovány, je možné pokračovat s připojením zprávy do nákladové oblasti datového komunikačního paketu. Po připojení zprávy je objekt paketu vrácen do místa volání metody.

```
            …
            JPacket packet = constructIpTcpPayloadPacket(message);
            if(packet != null && pcap.sendPacket(
                  ByteBuffer.wrap(
                        packet.getByteArray(
                              0,
                                    packet.size()))) == 0){
                  secondChance = 0;
            }
            else {
                  if(secondChance < 4){
                        rulesToExport.add(0, rule);
                        secondChance++;
                  } else secondChance = 0;

                  try {
                        Thread.sleep(1000);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
            }
      }
      pcap.close();
} else
      try {
            Thread.sleep(2000);
      } catch (InterruptedException e) {
            e.printStackTrace();
}
```

Vrácený paket je okamžitě odeslán skrze volání metody *sendPacket* na objektu typu Pcap, který byl dříve nastaven na odpovídající rozhraní a jeho spojení bylo otevřeno. V rámci odesílání paketu je kontrolována návratová hodnota odesílající metody.

V průběhu testování daného rozhraní bylo totiž zjištěno, že ne vždy dochází k úspěšnému poslání paketu.

Vzhledem k této skutečnosti bylo zahrnuto opakované odesílání identického paketu. Skrze číselnou proměnou je kontrolován počet těchto pokusů. Při neúspěšném odeslání je tento ukazatel zvýšen a pravidlo je opět zařazeno do seznamu pro odeslání. Tento krok může nastat maximálně čtyřikrát pro danou instanci paketu. Není však vhodné pokusit se o opětovné odeslání okamžitě po předchozím selhání. Z tohoto důvodu je vlákno vždy po chybném pokusu o odeslání uspáno, než následuje pokus nový. Pokud se dané pravidlo nepodaří v platném počtu pokusů úspěšně odeslat, je zahozeno a přechází se k dalšímu v pořadí.

Jakmile jsou všechna dostupná pravidla odeslaná, dojde k uzavření spojení na objektu typu Pcap a vlákno se dočasně uspí, než opětovně zkontroluje dostupná pravidla pro odeslání.

Následuje příklad zkonstruovaného paketu, který sloužil pro testování funkcionality odesílání aktualizačních zpráv. Po úspěšném odeslání byl paket zachycen rozhraním aplikace. Níže uvedený obraz představuje přímý převod objektu paketu do formy textového řetězce skrze metody knihovny jNetPcap [20].

```
Frame:
Frame:          number = 0
Frame:       timestamp = 2015-01-04 17:26:26.048
Frame:     wire length = 280 bytes
Frame: captured length = 280 bytes
Frame:
Eth:  ******* Ethernet - "Ethernet" - offset=0 (0x0) length=14
Eth:
Eth:      destination = e8:07:00:fc:fb:7f
Eth:                    .... ..0. .... .... = [2] LG bit
Eth:                    .... ...0 .... .... = [2] IG bit
Eth:           source = 00:0c:29:28:97:5b
Eth:                    .... ..0. .... .... = [2] LG bit
Eth:                    .... ...0 .... .... = [2] IG bit
Eth:             type = 0x800 (2048) [ip version 4]
Eth:
Ip:  ******* Ip4 - "ip version 4" - offset=14 (0xE) length=20 protocol
suite=NETWORK
Ip:
Ip:          version = 0
Ip:             hlen = 5 [5 * 4 = 20 bytes, No Ip Options]
Ip:         diffserv = 0x0 (0)
Ip:                    0000 00.. = [0] code point: not set
Ip:                    .... ..0. = [0] ECN bit: not set
Ip:                    .... ...0 = [0] ECE bit: not set
Ip:           length = 266
Ip:               id = 0x12FC (4860)
Ip:            flags = 0x2 (2)
Ip:                    0.. = [0] reserved
Ip:                    .1. = [1] DF: do not fragment: set
Ip:                    ..0 = [0] MF: more fragments: not set
Ip:           offset = 7039 [7039 * 8 = 56312 bytes]
Ip:              ttl = 32 [time to live]
Ip:             type = 0 [ip fragment of IPv6 Hop-by-Hop Option PDU]
Ip:         checksum = 0x5829 (22569) [correct]
Ip:           source = 192.168.200.255
Ip:      destination = 192.168.200.255
Ip:
Data:  ******* Payload offset=34 (0x22) length=246
Data:
0022: 4f 55 54 50  55 54 20 2d  73 20 31 39  32 2e 31 36    OUTPUT -s 192.16
```

```
0032: 38 2e 32 30  30 2e 31 33  30 2f 32 35  35 2e 32 35    8.200.130/255.25
0042: 35 2e 32 35  35 2e 32 35  35 20 2d 64  20 32 33 2e    5.255.255 -d 23.
0052: 36 34 2e 31  35 2e 32 36  2f 32 35 35  2e 32 35 35    64.15.26/255.255
0062: 2e 32 35 35  2e 32 35 35  20 2d 6f 20  65 74 68 30    .255.255 -o eth0
0072: 20 2d 70 20  54 63 70 20  2d 2d 73 70  6f 72 74 20     -p Tcp --sport
0082: 35 31 39 35  37 20 2d 2d  64 70 6f 72  74 20 38 30    51957 --dport 80
0092: 20 2d 6a 20  44 52 4f 50  20 2d 77 68  6f 61 6d 69     -j DROP -whoami
00a2: 20 72 6f 6f  74 20 5f 74  65 73 74 69  6e 67 20 72     root _testing r
00b2: 75 6e 20 65  78 61 6d 70  6c 65 20 66  6f 72 20 63    un example for c
00c2: 6f 6d 6d 75  6e 69 63 61  74 69 6f 6e  20 75 70 64    ommunication upd
00d2: 61 74 65 20  6f 76 65 72  20 69 6e 74  65 72 6e 65    ate over interne
00e2: 74 20 2f 20  64 6f 20 6c  6f 63 61 6c  20 73 69 74    t / do local sit
00f2: 65 20 74 68  61 6e 6b 73  20 74 6f 20  62 72 6f 61    e thanks to broa
0102: 64 63 61 73  74 20 61 64  64 72 65 73  73 20 66 72    dcast address fr
0112: 6f 6d 20 4e  49 43                                    om NIC
```

4.2 Komunikační rozhraní

Aplikace pro svoji činnost vyžaduje i propojení s dalšími oblastmi operačního systému. Skrze třídy představující komunikační rozhraní je možné přistupovat k jednotlivým souborům, které nesou data důležitá pro běh aplikace a její nastavení. Vzhledem k nutnosti přistupovat k více druhům souborů a v některých případech i k souborům, které nejsou ve správě tohoto programu, byl princip přístupu rozdělen podle jeho typu i do jednotlivých tříd.

Vlastní verzi rozhraní potom představuje třída odpovědná za přepis zachyceného komunikačního paketu. Jedná se o třídu, která je automaticky volána při zachycení komunikace a implementuje rozhraní knihovny jNetPcap [20] *PcapPacketHandler<T>*.

Posledním komunikačním prvkem aplikace je samotná analytická třída, která poskytuje nástroje pro práci se síťovými adresami a maskami, stejně jako zpřístupňuje jednotlivá síťová rozhraní místního počítače.

4.2.1 JPacketHandler

Důležitá součást pro přepis zachycené komunikace do interních objektů. Jelikož se jedná o automaticky volanou třídu, musí splňovat požadavky definované v rámci rozhraní, které je nezbytné řádně implementovat. Zároveň je však využito objektového principu jazyku JAVA a třída je vybavena i rozhraním pro přístup k jednotlivým atributům, které jsou v rámci procesu čtení paketu naplněny. Pro správnou činnost některých operací je potřeba pouze předat platné adresy síťového rozhraní počítače, na němž dochází k zachytávání, skrze konstruktor třídy.

```
Ethernet ethernet = new Ethernet();

if(packet.hasHeader(ethernet)){
    byte[] macDes = ethernet.destination();
    byte[] macSou = ethernet.source();

    if(macDes != null) this.macDes = macDes;
    if(macSou != null) this.macSou = macSou;
}
…
```

Přestože na konkrétním pořadí získání jednotlivých údajů nezáleží, protože objekt *packet* typu *PcapPaket* umožňuje získat jednotlivé hlavičky skrze jednoduché rozhraní,

přesto jsou prvními hodnotami hardwarové adresy směrování paketu z ethernetové hlavičky. Obě hodnoty jsou uloženy do bitového pole, odkud jsou následně překódovaný do standardizované podoby. Přeložené hodnoty jsou uloženy do lokálních proměnných. Mezi krok se ukázal být nezbytným pro kontrolu, zda jsou hodnoty skutečně obsaženy.

```
…
Ip4 ip = new Ip4();

if(packet.hasHeader(ip)){
    destination = FormatUtils.ip(ip.destination());
    source =FormatUtils.ip(ip.source());
}
…
```

V dalším kroku je získána ip adresa zdroje a cíle komunikačního paketu. Pokud daný paket obsahuje odpovídající hlavičku, což je kontrolováno v podmínce, dojde zároveň k přepsání předaného objektu hlavičky instancí dané hlavičky obsažené v objektu samotného paketu. Zdrojová a cílová ip adresa jsou opět uloženy do proměnných v rámci této třídy.

```
…
Tcp tcp = new Tcp();

if(packet.hasHeader(tcp)){
    sourPort = Integer.toString(tcp.source());
    destPort = Integer.toString(tcp.destination());
}

Udp udp = new Udp();

if(packet.hasHeader(udp)){
    sourPort = Integer.toString(udp.source());
    destPort = Integer.toString(udp.destination());
}
…
```

Následuje získání hlavičky transportního protokolu. Jelikož jsou dostupné dva odlišné druhy, je nezbytné určit, který je použitý zachyceným paketem. Každý z typů transportní technologie je zde zastoupen odpovídajícím objektem hlavičky. Pokud paket daný typ neobsahuje, nebudou z dané hlavičky odečítaná žádná data. V opačném případě jsou z údajů obsažených v rámci platné hlavičky získány čísla zdrojového a cílového portu.

```
…
Payload pay = new Payload();

if(packet.hasHeader(pay))
    if(destination.equals(FormatUtils.ip(broadcast)) &&
                !source.equals(FormatUtils.ip(address))){
          String payload = pay.getUTF8String(0, pay.size());
          if(IPTablesRule.chainEquality(payload.substring(
                      0, payload.indexOf(" ")))){
                updateRule = payload;
                update = true;
```

```
            }
        }
    …
```

Každý komunikační paket nese datovou část, která obsahuje samotnou zprávu. Její získání je v principu naprosto stejné jako v předchozích případech. Nastaví se objekt hlavičky a pokud je odpovídající záznam i v zachyceném paketu, nastaví se předaná instance na odpovídající objekt paketu.

Tento krok je však kvůli absenci algoritmů pro rozpoznávání závadného obsahu nebo virového skeneru zbytečný pro normální komunikaci. Proto se využívá pouze v případě zachycení aktualizační zprávy. Samozřejmě je nutné se ujistit, že původ zprávy je na jiném počítači v rozsahu místní sítě. Následně dojde k přeložení dat přenášených v rámci nákladového prostoru paketu do podoby textového řetězce. Dále je zkontrolována konzistence zprávy pomocí porovnání počátečního výrazu vůči akceptovaným jménům řetězců ve firewallovém rozhraní iptables [14]. Pokud vše odpovídá, nastaví se v rámci atributů třídy ukazatel na zachycenou aktualizační zprávu, aby ji bylo možné zpracovat odděleně od standardní komunikace. Tato činnost byla popsána v rámci kapitoly **4.1.3 Vlákno "snooper"**, následné zpracování takové zprávy potom bylo probráno v kapitole **4.1.6 Vlákno "updater"**.

```
…
JHeader notLast = getNotLastHeadear(packet);
name = notLast.getName();

if(source.equals(FormatUtils.ip(address)) &&
          destination.equals(FormatUtils.ip(broadcast)))
    sendUpdate = true;
```

Posledním krokem je vyhledání protokolu využitého zachyceným paketem, který je uložen v rámci předposlední paketové hlavičky. Poslední hlavičku představuje samotná zpráva a tedy důvod existence daného paketu. Pro její získání je použita metoda, která vrací instanci této hlavičky, jedná se o upravený příklad [29] z poradenského fóra tvůrců knihovny jNetPcap [20]. Po získání této informace zbývá nastavit ukazatel v případě zachyceného odesílaného aktualizačního paketu. Tímto krokem se zamezí aplikování pravidla, které vzniklo činností lokálního počítače a bylo odesláno do místní sítě.

```
private JHeader getNotLastHeadear(JPacket packet){
    int last = packet.getHeaderCount() - 1;

    if(packet.getHeaderIdByIndex(last) == Payload.ID && last > 0)
          last--;

    JHeader header = JHeaderPool.getDefault()
                      .getHeader(packet.getHeaderIdByIndex(last));
    packet.getHeaderByIndex(last, header);

    return header;
}
```

V rámci hledání předposlední hlavičky je nejdříve nastaven ukazatel na předposlední objekt v rámci daného paketu. Pokud by se na tomto místě stále

nacházela oblast nákladu, je nezbytné posunout se ještě o jednu další hlavičku dopředu, ukazatel na pozici se tedy změnší.

Následně se skrze rozhraní knihovny jNetPcap [20] a třídy poskytující metody pro objekt typu PcapPacket získá instance odpovídající hlavičky. Ta se uloží do objektu, který se inicializuje podle typu hlavičky umístěné v dané pozici v těle paketu. Následně je tento objekt vrácen do místa volání metody.

4.2.2 CommunicationBridge

Třída zpřístupňující aplikaci prostředí operačního systému. Skrze své metody umožňuje využívat prostředí shellu včetně odchytávání chybových stavů, pokud nastanou. Dále obsahuje rutiny a nese odpovědnost za fyzický přístup k souborům uložených na disku a jejich správě.

```
public String forwardCommand(String command){
    if(command.isEmpty()) return "Empty Command!";
    if(!buffer.isEmpty()) buffer = "";

    try {
          Process proc = Runtime.getRuntime().exec(command);
          if(rewrite(proc.getInputStream()).isEmpty())
                rewrite(proc.getErrorStream());
          proc.destroy();
    } catch (IOException e) {
          e.printStackTrace();
    }

    return buffer;
}
```

Mezi základní potřeby aplikace se řadí spojení s shellem operačního systému, který skrze rozhraní zpřístupňuje firewall iptables [14]. Nejdříve je nutné zkontrolovat, zda byl obdržen příkaz pro předání prostředí shellu. V případě chyby, tedy prázdného příkazu, by se na obrazovce aplikace zobrazil nápis, který by na tento stav upozornil. V dalším kroku se kontroluje řádné vyprázdnění návratového objektu.

Následně dochází k získání instance *Process* skrze volání *Runtime.exec()*. Je nutné přepsat obsah této instance, který představuje reakci na předaný povel. K získání této hodnoty slouží privátní metoda, která přijímá jako svůj jediný vstup proud, ze kterého má být čteno.

```
private String rewrite(InputStream ins){
    String line;
    BufferedReader reader = new BufferedReader(
                                  new InputStreamReader(ins));

    try {
          while((line = reader.readLine()) != null)
                buffer = buffer + line + "\n";
          reader.close();
          ins.close();
          buffer = buffer.trim();
    } catch (IOException e) {
          e.printStackTrace();
    }
```

```
        return buffer;
    }
```

Předaný vstupní proud je nejdříve řádně otevřen skrze instance *BufferedReader*. V rámci následného cyklu, který bude pokračovat dokud v proudu budou dostupná data, dochází k přepisu jednotlivých řádků odpovědi do návratového objektu. Jelikož je tento objekt, *buffer*, představován globální proměnou, není explicitně nutné jeho předávání v případě lokálního volání. I přesto je však v rámci metody vrácen pro kontrolu, zda byla nějaká data přímo přečtena.

Jakmile jsou všechny dostupné řádky přepsány, dojde k uzavření instance *BufferedReader* a následně i vstupního proudu. Posledním krokem před samotným vrácením objektu je jeho oříznutí kvůli případným mezerám před či za čteným textem.

Pokud by byl vrácen prázdný objekt, je předpokládáno, že při vykonání předaného příkazu nastala v shellu nějaká chyba. Opakuje se tedy volání privátní metody pro přepis ze vstupního proudu, ale tentokrát je jí předán chybový kanál.

4.2.2.1 Vytváření souborů

Jelikož aplikace dokáže dynamicky vytvářet uživatelské účty, je nezbytné pro ně vytvořit i ukládací struktury ve formě samostatných souborů.

```
public String createFile(PropertyLoader properties){
    String name = "";
    do{
          name = UUID.randomUUID().toString();
    } while (properties.containPropertyKey(name));

    File file = new File("./data/" + name);
    try {
          file.createNewFile();
    } catch (IOException e) {
          e.printStackTrace();
    }

    return name;
}
```

Celý proces je velice jednoduchý. Nejdříve je nutné vytvořit řetězec, který bude charakterizovat jméno. K tomuto účelu je využit náhodný generátor, jehož výsledek je kontrolován vůči seznamu již uložených hodnot, aby se zajistila jedinečnost souboru. Proces se opakuje, dokud této podmínky není dosaženo.

Následně je získána instance reprezentující nový soubor, který je v dalším kroku sám vytvořen. Metoda nakonec vrátí jméno takto nově vytvořeného souboru do místa volání metody.

Kromě tvorby uživatelských souborů pro ukládání seznamu pravidel pro rozhraní firewallu podporuje třída ještě jeden způsob tvorby a to v případě poruchy a chybějících souborů.

```
private void restoreFile(String path){
    File file = new File(path);
```

```
try {
      file.getParentFile().mkdirs();
      file.createNewFile();

      PropertyLoader pl = new PropertyLoader(path);
      String fileName = createFile(pl);

      List<String> temp = new ArrayList<>();
      temp.add("default.file=" + fileName);
      temp.add("default.multiUser=default");
      temp.add("default.threshold=100");
      temp.add("default.politics=ACCEPT");
      temp.add("sendUpdates=true");
      temp.add("receiveUpdates=true");
      temp.add("autoStartScan=true");
      temp.add("default.packetAction=DROP");
      temp.add("users=default");
      temp.add("default.refreshInterval=2");
      temp.add("startupProfile=default");
      temp.add("default.updateLimit=4");
      temp.add("default.unknownService=TRUE");
      temp.add("default.ruleReplacement=6");

      saveList(path, temp, false);
      …
```

K volání této metody může dojít pouze v případě pokusu o přístup k souboru properties, který nebyl nalezen. Aplikace pak předpokládá jeho ztrátu, smazání či první spuštění sebe sama a vytvoří si jeho náhradu. Na objektu typu *File*, který je nastaven na pozici a jméno očekávaného properties souboru, jsou volány metody pro vytvoření cesty, tedy potřebných složek, a samotného souboru.

Následně dojde k dočasnému vytvoření druhé instance rozhraní PropertyLoader v rámci celé aplikace, aby bylo možné inicializovat a vytvořit soubor s nastavením pro defaultní účet. K vytvoření souboru je využito metody, která byla popsána v předchozí kapitole.

Nebylo by efektivní přidávat jednotlivé hodnoty po jedné a proto jsou sestavené řádky s nastavením uloženy do seznamu, který je skrze odpovídající metodu uložen do nově vytvořeného souboru. Samotné uložení těchto dat bude popsáno v následující kapitole.

```
      …
      temp.clear();

      temp.add("INPUT -t filter -p icmp -j ACCEPT");
      temp.add("INPUT -t filter -i lo -j ACCEPT");
      temp.add("INPUT -t filter -p tcp --dport 22" +
                  + "-m state --state NEW -j ACCEPT");
      temp.add("INPUT -t filter -j REJECT" +
                  + "--reject-with icmp-host-prohibited");
      temp.add("INPUT -t filter -m state" +
            + "--state RELATED,ESTABLISHED -j ACCEPT");

      saveList("./data/" + fileName, temp, false);
} catch (IOException e) {
      e.printStackTrace();
```

```
    }
}
```

Po uložení je seznam vyprázdněn a opětovně využit. Tentokrát jsou v něm uskladněna pravidla pro firewall, která jsou stejnou metodou uložena do souboru obsahujícím základní chování firewallu pro ukázkový profil.

Kromě procesu vytváření zvládá aplikace i jejich smazání. Této funkce se využívá především při odstranění uživatelského účtu z prostředí aplikace.

```
public boolean deleteFile(PropertyLoader properties, String whoami){
    return new File("./data/" + properties.getProperty(whoami
                    + ".file")).delete();
}
```

Na základě zadaného jména účtu, je z properties získáno jméno odpovídajícího souboru, který je následně skrze volání metody *delete()* na instanci objektu typu *File* smazán. Výsledek této činnosti je vrácen do místa volání.

4.2.2.2 Přístup k souborům

Do souborů je možné zapisovat po jednotlivých řádcích. Je k tomu využito volání metody pro hromadný zápis se seznamem, který obsahuje jediný řádek. Odpadá tak nutnost pro každý jednotlivý zápis vytvářet lokální seznam. Tuto aktivitu dokáže obstarat volání následující metody.

```
public void appendLineToList(String fileName, String lineToSave){
    List<String> temp = new ArrayList<>();
    temp.add(lineToSave);
    saveList(fileName, temp, true);
}
```

Do dočasného seznamu je uložen předaný řádek určený k uložení. Kam se má ukládat specifikuje parametr nesoucí jméno cílového souboru.

```
public void saveList(String fileName, List<String> list,
                                                    boolean append){
    if (list != null && !fileName.isEmpty()) {
          try {
                BufferedWriter writer = new BufferedWriter(
                            new FileWriter(
                                  new File(fileName), append));
                for (String s : list) {
                      if (append) writer.append(s);
                      else writer.write(s);
                      writer.newLine();
                }
                writer.close();
          } catch (FileNotFoundException e) {
                e.printStackTrace();
          } catch (IOException e) {
                e.printStackTrace();
          }
    }
}
```

Metoda nejdříve zkontroluje, zda předaný seznam pro uložení není prázdný. V takovém případě by bylo zbytečné přistupovat k souboru. Následuje otevření souboru pro zápis skrze instanci typu *BufferedWriter*, která přijme název souboru, včetně případné cesty, a jako druhý parametr ukazatel typu přístupu. Metodu je možné využít i k úplnému přepsání obsahu souboru, nebo k jeho rozšíření připojením nových řádků na jeho konec.

Následně dochází k zápisu po jednotlivých položkách seznamu, dokud jsou k dispozici. Po každém připsání nové hodnoty je vložen ukazatel na nový řádek, aby se zajistilo rozdělení obsahu souboru.

Po skončení zápisu, tedy vyčerpání všech položek k tomu určených, je soubor uzavřen.

Druhým způsobem přístupu k jednotlivým souborům je čtení jejich obsahu, který mohl být zapsán prostřednictvím dříve popsané metody, ale také může náležet do prostředí operačního systému. V druhém případě je tedy nespornou výhodou, že všechny konfigurační i ostatní systémové soubory systémů typu GNU/Linux jsou vždy tvořeny čistě textovým obsahem. Odpadá totiž problém s jejich kódováním a případným převodem pro další použití obsažených údajů.

```
public List<String> loadList(String file){
    List<String> returnList = new ArrayList<>();
    String line;
    try {
          BufferedReader reader = new BufferedReader(
                                  new FileReader(new File(file)));
          while((line = reader.readLine()) != null)
                returnList.add(line);

          reader.close();
    } catch (FileNotFoundException e) {
          e.printStackTrace();
    } catch (IOException e) {
          e.printStackTrace();
    }

    return returnList;
}
```

Načtení je ekvivalentem zápisu popsaným v předchozí metodě. Požadovaný soubor je otevřen skrze instanci typu *BufferedReader* a obsah je po jednotlivých řádcích čten a ukládán do seznamu. Po přečtení celého souboru je přístup uzavřen a vytvořený seznam vrácen do místa volání metody.

Tímto způsobem probíhá například načtení seznamu portů a služeb z prostředí operačního systému při úvodním spuštění aplikace. Tento proces byl popsán v kapitole **4.1 Vlákna**.

Speciálním případem čtení obsahu souboru je zpřístupnění properties dat spravující třídě. Právě při této činnosti může dojít k vyvolání metody pro obnovení souboru, nebo jeho vytvoření při úvodním spuštění aplikace, která byla popsána v předchozí kapitole.

```
public Properties loadPropertiesFile(String propertyLocation){
    if(!fileExist(propertyLocation)) restoreFile(propertyLocation);
```

```
        File propertyFile = new File(propertyLocation);
        FileInputStream in = null;
        Properties prop = null;
        try {
              in = new FileInputStream(propertyFile);
              prop = new Properties();
              prop.load(in);
              in.close();
        } catch (FileNotFoundException e) {
              e.printStackTrace();
        } catch (IOException e) {
              e.printStackTrace();
        }

        return prop;
    }
```

Po volání metody dojde ke kontrole existence souboru. Za předpokladu jeho nalezení v udaném umístění, je načten do objektu typu *File*. Následně je k této instanci přistoupeno skrze souborový vstupní proud, který umožňuje struktuře *Properties* načtení pomocí volání metody *load(in)*. Po přečtení je soubor opět uzavřen a načtená struktura vrácena do místa původu volání této metody.

Obdobným způsobem dochází i k uložení případných změn v rámci tohoto speciálního typu souboru.

```
    public void savePropertiesFile(String propertyLocation,
                                                    Properties properties){
        File propertyFile = new File(propertyLocation);

        try {
              FileOutputStream out = new FileOutputStream(propertyFile);
              properties.store(out, null);
              out.close();
        } catch (FileNotFoundException e) {
              e.printStackTrace();
        } catch (IOException e) {
              e.printStackTrace();
        }
    }
```

Podle zadaného umístění je otevřena instance s daným souborem. Následně je připojen výstupní souborový proud, aby bylo možné zapsat všechna data. Skrze volání metody rozhraní *Properties store()* dojde k zapsání prostřednictvím předaného výstupního proudu. Hodnota *null* v tomto případě zastupuje případný komentář pro uložení do souboru.

Po zapsání je opět spojení skrze odchozí proud uzavřeno, čímž jsou všechna data, která doposud nemusela být řádně zapsána, uložena do souboru.

4.2.3 PropertyLoader

Celé nastavení aplikace, i jednotlivých účtů, je sdruženo do jediného souboru typu properties. Tento typ souboru je představován textovými daty, kdy na každém řádku je

uložen klíč, nebo též název, a za rovnítkem jeho hodnota. Následuje ukázka několika řádků z tohoto typu souboru s nastavením aplikace.

```
…
root.threshold=100
root.unknownService=true
users=default,root
default.politics=ACCEPT
default.packetAction=DROP
default.unknownService=true
…
```

V rámci celé aplikace je kvůli konzistenci hodnot uložených v tomto souboru udržována a předávána jediná instance této třídy, která odpovídá za veškeré operace nad těmito daty. Ve většině případů pouze zpřístupňuje nižší úroveň rozhraní *Properties*, které je v rámci jazyka JAVA dostupné, ale doplňuje i některé vyšší funkcionality pro potřeby aplikace.

```
public String getProperty(String key){
    String property = null;

    if(!key.isEmpty())
          property = properties.getProperty(key);

    return property;
}
```

Získání hodnoty podle vloženého klíče představuje jednu z metod, která pouze poskytuje rozhraní pro přístup k nižším voláním na samotném rozhraní struktury *Properties*. Dojde ke kontrole vloženého klíče, pokud by byl představován prázdným řetězcem, je vrácena hodnota *null*, na níž byl návratový objekt nastaven. V opačném případě je klíč předán dále do rozhraní a výsledná hodnota závisí na správnosti tohoto klíče. V případě nenalezení odpovídající položky, například kvůli špatnému jménu nebo částečně poškozenému souboru, je opět vrácena hodnota *null*.

```
public List<String> getProperties(String key){
    List<String> l = new ArrayList<>();
    String prop = getProperty(key);
    String[] field;

    if(prop != null) {
          field = prop.split(",");

          for(String value:field)
                l.add(value);
    }

    return l;
}
```

Pro potřeby aplikace bylo nezbytné zajistit možnost uložení více násobné hodnoty pod jediným klíčem. Tuto funkcionalitu rozhraní *Properties* nepodporuje. Bylo proto nezbytné zajistit její použitelnost jinými metodami.

Důležité pro funkčnost této metody je způsob uložení jednotlivých hodnot, respektive dělící symbol. Skrze volání dříve popsané metody dojde k získání jediné hodnoty, která je složena z jednotlivých údajů oddělených odpovídajícím symbolem. Tento řetězec je tedy rozřezán podle tohoto symbolu a následně přepsán do seznamu, který je vrácen.

Pokud by nedošlo k úspěšnému načtení hodnoty skrze volání metody pro získání odpovídající položky na základě znalosti klíče, je vrácen prázdný seznam, aby se omezila potřeba kontroly návratové hodnoty.

Následující dvojice metod zajišťuje změny v rámci hodnot a klíčů v souboru. Umožňují změnu hodnoty nebo smazání řádku skrze volání nižší metody na rozhraní *Properties*.

```
public String setProperty(String key, String value){
    String returnValue = null;

    if (!key.isEmpty() && !value.isEmpty()){
          properties.setProperty(key, value);
          new CommunicationBridge()
                .savePropertiesFile(propertyLocation, properties);
    }

    return returnValue;
}
```

Pro změnu, nebo nové vytvoření, je důležité zadat nejen název klíče, ale i hodnotu, která má byt pod tímto označením uložena. Nejdříve tak dochází ke kontrole správnosti těchto údajů, aby bylo možné pokračovat. Skrze volání metody *setProperty()* na rozhraní *Properties* dojde k zapsání zadaných údajů do lokální verze souboru. Aby se změny uchovali i v případě vypnutí, je nutné skrze třídu pro správu souborů uložit aktuální verzi.

```
public String deleteProperty(String key){
    String returnValue = null;

    if (!key.isEmpty()){
          properties.remove(key);
          new CommunicationBridge()
                .savePropertiesFile(propertyLocation, properties);
    }

    return returnValue;
}
```

V případě potřeby vymazat nějakou hodnotu ze souboru, stačí zadat klíč k této hodnotě a skrze volání metody *remove()* na objektu typu *Properties* dojde ke smazání. Jako v předchozím případě se změnou hodnoty daného klíče, je však nezbytné veškeré změny uložit do souboru, aby se zachovaly i při dalším spuštění aplikace.

V obou popsaných metodách dochází k vrácení hodnoty, která představuje předchozí údaj uložený v rámci souboru pod zadaným klíčem. V případě vytvoření nového klíče je však vrácena hodnota *null*, protože žádná předchozí není dostupná. Je

tak umožněna i určitá kontrola činnosti v případě, kdy bychom si nepřáli přijít o původní hodnotu.

```
public boolean containProperty(String property){
    List<String> users = getProperties("users");
    for(String s:users)
          if(property.equals(getProperty(s + ".file")))
                return true;

    return false;
}
```

Poslední užitečná metoda slouží hlavně pro kontrolu existence jména souboru s pravidly daného uživatelského profilu. Prvním krokem je načtení seznamu aktuálně platných uživatelských účtů skrze volání metody *getProperties()*, která byla již dříve v této kapitole popsána.

Následně probíhá ověření, zda některý z profilů již nemá registrovaný soubor s odpovídajícím názvem, který byl předán v rámci volání této metody. Byl-li nějaký soubor nalezen je okamžitě vykonávání metody přerušeno a vrácena hodnota *true* typu *boolean*. Při vyčerpání všech profilů a nenalezení shody je vrácena hodnota *false*, která indikuje použitelnost daného jména.

4.2.4 Analyzer

Analytická třída nenese odpovědnost pouze za tuto jedinou činnost, přestože představuje její hlavní účel. Zároveň však slouží jako převodník, který dokáže pro potřeby aplikace jednoduše převádět rozsahy a formáty jednotlivých položek.

Proces analýzy a jeho jednotlivé fáze už byly popsány v rámci kapitoly **4.1.4.3 Analýza** při popisu vlákna "enforcement".

Metody sloužící pro ověřování shody dvou ip adres včetně prolnutí jejich rozsahů definovaných pomocí masky sítě byly popsány v kapitole **4.1.4.1 Zjednodušení pravidel**. Tato funkce je samozřejmě využívána i v rámci samotné analýzy.

Odpovědnost třídy za identifikaci a získání informací o dostupných síťových rozhraních počítače byla popsána v rámci kapitoly **4.1.1 Nastavení prostředí**.

```
public static String[] ipFieldsFromIpAddress(String address){
    if (!address.isEmpty()) {
          String[] fields = new String[4];
          int k = 0;

          for (int i = 0; i < 4; i++) {
                int b = Integer.parseInt(address.substring(k,
                            address.indexOf(".", k) < 0 ?
                address.length() : address.indexOf(".", k)));
                fields[i] = Integer.toString(b);
                k = address.indexOf(".", k + 1) + 1;
          }
          return fields;
    } else return new String[4];
}
```

Metoda určená pro transformaci ip adresy v plném tvaru xxx.xxx.xxx.xxx do jednotlivých buněk v poli po jednotlivých oktetech adresy. Nejdříve je zkontrolováno, zda byla adresa skutečně předána. Následně dojde k rozřezání textového řetězce s adresou. K dělení dochází v rámci cyklu, kdy se pomocí proměnné posunuje index, který určuje hranice pro získání části řetězce. Pro rozhodnutí o dosažení konce je využita podmínka zapsaná v jediném řádku [30]. Kvůli kontrole je obsah této části řetězce zkonvertován do objektu Integer a následně zpět do podoby textového řetězce, který se uloží do výstupního pole.

```
public static String numericMaskFromIpMask(String mask){
    if (!mask.isEmpty()) {
          int resultMask = 0;
          int k = 0;

          for (int i = 0; i < 4; i++) {
                int b = Integer.parseInt(mask.substring(
                      k,
                      mask.indexOf(".", k) < 0 ?
                            mask.length() : mask.indexOf(".", k)));
                int index = 0;
                int temp = 0;
                while (b != temp)
                      temp += field[index++];

                resultMask += index;

                k = mask.indexOf(".", k + 1) + 1;
          }

          return Integer.toString(resultMask);
    } else return mask;
}
```

Někdy je vzhledem k potřebám uživatelského rozhraní nutný převod mezi plnou formou masky sítě a jejím numerických vyjádřením.

Po zkontrolování, zda byla opravdu předána adresa, se inicializují odpovědné proměnné. V cyklu, který prochází jednotlivé segmenty masky pomocí stejného principu pohyblivého indexu jako ve výše popsané metodě, se získá hodnota daného oktetu pro převod. Jeho hodnota se pak porovnává s hodnotu uloženou v globálním poli nesoucím všechny možnosti nastavení daného oktetu. Jakmile je dosaženo shodné velikosti, uloží se pozice této hodnoty v poli jako přírůstek převodníku. Posledním krokem v rámci cyklu je aktualizace pohyblivého indexu pro příští průběh.

```
public static String ipMaskFromNumericMask(String mask){
    int value = Integer.parseInt(mask);
    String resultMask = "";
    for(int i = 0; i < 4; i++)
          if(value > 8){
                value -= 8;
                if(!resultMask.isEmpty()) resultMask += ".";
                resutlMask += "255";
          } else if(value > 0){
                int result = 0;
                int index = 0;
```

```
                while(value-- > 0)
                        result += field[index++];

                if(!resultMask.isEmpty()) resultMask += ".";
                resultMask += Integer.toString(result);
            } else {
                if(!resultMask.isEmpty()) resultMask += ".";
                resultMask += "0";
            }

        return resultMask;
    }
```

Ze stejných důvodů jako v předchozím případě je však výhodné mít možnost získat zpět plnou masku sítě z její numerické hodnoty.

V prvním kroku dojde k převodu textového řetězce reprezentujícího numerickou hodnotu masky sítě do číselného formátu. Následně je tato hodnota v průběhu cyklu snižována, aby se naplnily jednotlivé oktety plné masky sítě.

V tomto případě se prováděné kroky určují podle velikosti číselného formátu adresy. Pokud je velikost větší než 8, bude hodnota odečtena a odpovídající oktet naplněn maximální hodnotou. V případě menšího ale stále kladného čísla, dochází k hledání shody s prvky v rámci globálního seznamu postupným snižováním hodnoty. Jakmile je dosaženo nuly, je prvek na posledním zaznamenaném indexu uložen jako hodnota odpovídajícího oktetu masky.

Poslední variantou je velikost číselné hodnoty rovna 0. V takovém případě se jednoduše doplní stejná hodnota i do výsledného oktetu masky.

Po čtvrtém proběhnutí cyklu je výsledná maska vrácena do místa volání metody.

4.3 GUI

Přestože hlavní prostředky celé aplikace jsou představovány automatickou činností jednotlivých vláken, nejviditelnějším prvkem je uživatelské rozhraní. Skrze něj nejenže uživatel může kontrolovat činnost celé aplikace, ale též nastavovat jednotlivé parametry, spravovat uživatelské účty v prostředí programu nebo pravidla chování firewallu.

Veškeré dostupné aktivity uživatele jsou řízeny formou obsloužených akcí mapovaných na jednotlivá tlačítka v prostředí rozhraní aplikace. Kvůli nutnosti zobrazení velkého množství informací vztažených k úzké oblasti fungování, je celé uživatelské prostředí rozděleno pomocí dialogových oken na menší funkční celky. Tato struktura umožňuje i zjemnění odpovědnosti jednotlivých obsluh a zvýšení přehlednosti chování a úpravy celé aplikace.

4.3.1 Uživatelské profily

Fungování aplikace je svázáno s existencí uživatelského účtu. Každý účet představuje soubor nastavení, které definují chování a zamření aplikace a kontroly probíhající síťové komunikace.

Každý profil je v rámci properties souboru s nastavením zaregistrován pod společným klíčem. Takový profil je následně možné aktivovat a používat v rámci

aplikace. Vytvoření nového profilu je vždy možné skrze odpovědné rozhraní, které nabízí i možnosti pro úpravu vlastností již existujících účtů.

```
users=default,root
```

Ukázka způsobu registrace účtu v rámci properties souboru pod jediným klíčem. Pro získání individuálních jmen je využito rozšířené rozhraní pro přístup k hodnotám, které bylo popsáno v kapitole **4.2.3 PropertyLoader**. V tomto případě jsou v rámci aplikace registrovany dva účty, přičemž profil se jménem "default" představuje ukázkový účet sloužící pro úvodní nastavení a případnou obnovu poškozených dat platných profilů. Tento účet není možné skrze aplikační prostředí nijak editovat ani používat.

```
root.interface=eth0
root.file=d915364b-2417-4669-a0ef-d7671ce990ea
root.multiUser=root
root.threshold=100
sendUpdates=true
receiveUpdates=true
autoStartScan=true
root.interfaceMAC=00\:0C\:29\:28\:97\:5B
root.politics=ACCEPT
root.ruleReplacement=6
root.packetAction=DROP
startupProfile=root
root.refreshInterval=2
```

Příklad platného uživatelského účtu a nastavení jednotlivých vlastností, jak jsou zaneseny v properties souboru aplikace. Nutno zmínit, že pořadí těchto hodnot si řídí aplikace automaticky při ukládání do souboru a není tedy zajištěno stejné pořadí ve všech instancích programu.

Každý profil, který je řádně zanesen v konfiguračním properties souboru, potřebuje ještě jedno fyzické úložiště. Tím se stává dynamicky vytvářený soubor, který je s daným účtem propojen skrze odpovídající hodnotu. Slouží pro ukládání a správu jednotlivých pravidel použitých pro nastavení a kontrolu chování firewallového rozhraní iptables [14] skrze aplikační prostředí a odpovědné třídy. Následuje ukázka obsahu tohoto souboru pro uživatelský profil "root", který byl použit v rámci testování chování aplikace v prostředí virtuálního počítače s operačním systémem CentOS 6.5

```
INPUT -t filter -p icmp -j ACCEPT
INPUT -t filter -i lo -j ACCEPT
INPUT -t filter -p tcp --dport 22 -m state --state NEW -j ACCEPT
INPUT -t filter -j REJECT --reject-with icmp-host-prohibited
INPUT -t filter -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i eth0 -s 1.11.22.14 -d 202.54.1.2 -p tcp --destination-port 443 -j ACCEPT
-A INPUT -i eth0 -s 1.11.22.14 -d 202.54.1.2 -p tcp --destination-port 443 -j ACCEPT
-A INPUT -i eth0 -s 1.11.22.14 -d 202.54.1.2 -p tcp --destination-port 443 -j ACCEPT
-A INPUT -i eth0 -s 1.2.3.5 -d 202.54.1.2 -p tcp --destination-port 443 -j ACCEPT
-A INPUT -i eth0 -s 1.2.3.5 -d 202.54.1.2 -p tcp --destination-port 443 -j ACCEPT
-A INPUT -i eth0 -s 1.2.3.5 -d 202.54.1.2 -p tcp --destination-port 443 -j ACCEPT
-A INPUT -i eth0 -s 1.2.3.5 -d 202.54.1.2 -p tcp --destination-port 443 -j ACCEPT
-A INPUT -i eth0 -s 1.2.3.5 -d 202.54.1.2 -p tcp --destination-port 443 -j ACCEPT
-A INPUT -i eth0 -s 1.11.22.14 -d 202.54.1.2 -p tcp --destination-port 443 -j ACCEPT
INPUT -t filter -s 23.64.15.32/255.255.255.255 -d 192.168.200.130/255.255.255.255 -i eth0 -p Tcp
--sport 80 --dport 38512 -j DROP -whoami root $ms 00:50:56:E0:BE:2A $md 00:0C:29:28:97:5B
_Unknown MAC address _ possible spoof packet attack
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 239.255.255.250/255.255.255.255 -i eth0 -p
Udp --sport 1900 --dport 1900 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md 01:00:5E:7F:FF:FA
_Exceeded user's threshold limit 100 for packet's occurence by 102
```

```
INPUT -t filter -s 192.168.200.1/255.255.255.255 -d 255.255.255.255/255.255.255.255 -i eth0 -p
Udp --sport 57034 --dport 8612 -j DROP -whoami root $ms 00:50:56:C0:00:08 $md FF:FF:FF:FF:FF:FF
_Neither service or port were unknown
```

Jelikož aplikace dokáže obnovit poškozené hodnoty všech používaných a registrovaných účtů, je schopna i jejich vytvoření, čehož se využívá především při úvodním spuštění aplikace.

```
if(properties.getProperty("autoStartScan") == null)
    properties.setProperty("autoStartScan", "true");

if(properties.getProperty("receiveUpdates") == null)
    properties.setProperty("receiveUpdates", "true");

if(properties.getProperty("sendUpdates") == null)
    properties.setProperty("sendUpdates", "true");

if(properties.getProperty("users") == null)
    properties.setProperty("users", "default");

if(properties.getProperty("startupProfile") == null)
    properties.setProperty("startupProfile", "default");
…
```

Ještě před kontrolou samotného profilu dochází ke kontrole globálních proměnných, které jsou všemi registrovanými účty sdíleny v rámci properties souboru. Pokud by kterákoliv z těchto hodnot nebyla nalezena, dojde k jejímu automatickému obnovení a nastavení původních hodnot z paměti aplikace. Není bohužel možné zjistit, které konkrétní hodnoty byly nastaveny a vrátit se k nim.

```
…
if(!properties.getProperties("users").contains(newWhoami)){
    properties.setProperty("users", properties.getProperty("users")
                + "," + newWhoami);
    properties.setProperty(newWhoami + ".refreshInterval",
                properties.getProperty("default.refreshInterval"));
    properties.setProperty(newWhoami + ".threshold",
                properties.getProperty("default.threshold"));
    properties.setProperty(newWhoami + ".multiUser", newWhoami);
    properties.setProperty(newWhoami + ".packetAction",
                properties.getProperty("default.packetAction"));
    properties.setProperty(newWhoami + ".ruleReplacement",
                properties.getProperty("default.ruleReplacement"));
    properties.setProperty(newWhoami + ".politics",
                properties.getProperty("default.politics"));
    properties.setProperty(newWhoami + ".file",
                bridge.createFile(properties));
    bridge.saveList(
          "./data/" + properties.getProperty(newWhoami + ".file"),
          bridge.loadList("./data/" +
                      properties.getProperty("default.file")),
          false);
} else {
    …
```

Následuje ověření existence jména uživatelského účtu. V případě prvního spuštění se jedná o název účtu operačního systému, pod kterým ke spuštění vlastní aplikace došlo.

Nebyl-li odpovídající záznam nalezen, je nutné vytvořit všechny potřebné struktury pro aktivování profilu.

Jméno je zaneseno do proměnné, která sdružuje všechny názvy uživatelských účtů. Následně jsou inicializovány všechny požadované položky podle hodnot patřících účtu "default".

Posledním krokem je vytvoření privátního souboru nového profilu skrze odpovídající metody třídy odpovědné za přístup k souborům, které byly popsány v kapitole **4.2.2.1 Vytváření souborů**. Hlavní je zajistit jedinečnost jména souboru, aby nemohlo dojít k nechtěnému sdílení a tedy i narušení funkčnosti jak profilu tak celé aplikace.

```
…
} else {
    if(properties.getProperty(
                        newWhoami + ".refreshInterval") == null)
          properties.setProperty(newWhoami + ".refreshInterval",
                properties.getProperty("default.refreshInterval"));

    if(properties.getProperty(newWhoami + ".threshold") == null)
          properties.setProperty(newWhoami + ".threshold",
                        properties.getProperty("default.threshold"));

    if(properties.getProperty(newWhoami + ".multiUser") == null)
          properties.setProperty(newWhoami + ".multiUser",
                        newWhoami);

    if(properties.getProperty(newWhoami + ".packetAction") == null)
          properties.setProperty(newWhoami + ".packetAction",
                properties.getProperty("default.packetAction"));

    if(properties.getProperty(
                            newWhoami + ".ruleReplacement") == null)
          properties.setProperty(newWhoami + ".ruleReplacement",
                properties.getProperty("default.ruleReplacement"));

    if(properties.getProperty(newWhoami + ".politics") == null)
          properties.setProperty(newWhoami + ".politics",
                        properties.getProperty("default.politics"));

    if(properties.getProperty(newWhoami + ".file") == null){
          properties.setProperty(newWhoami + ".file",
                bridge.createFile(properties));
          bridge.saveList(
                "./data/" + properties.getProperty(
                            newWhoami + ".file"),
                bridge.loadList("./data/" +
                            properties.getProperty("default.file")),
                false);
    }

    if(properties.getProperty(newWhoami + ".firstGateMAC") != null)
          properties.deleteProperty(newWhoami + ".firstGateMAC");
}
```

V případě, že bylo nalezeno odpovídající jméno profilu v rámci struktury konfiguračního souboru, proběhne ověření integrity jednotlivých nastavení tohoto účtu.

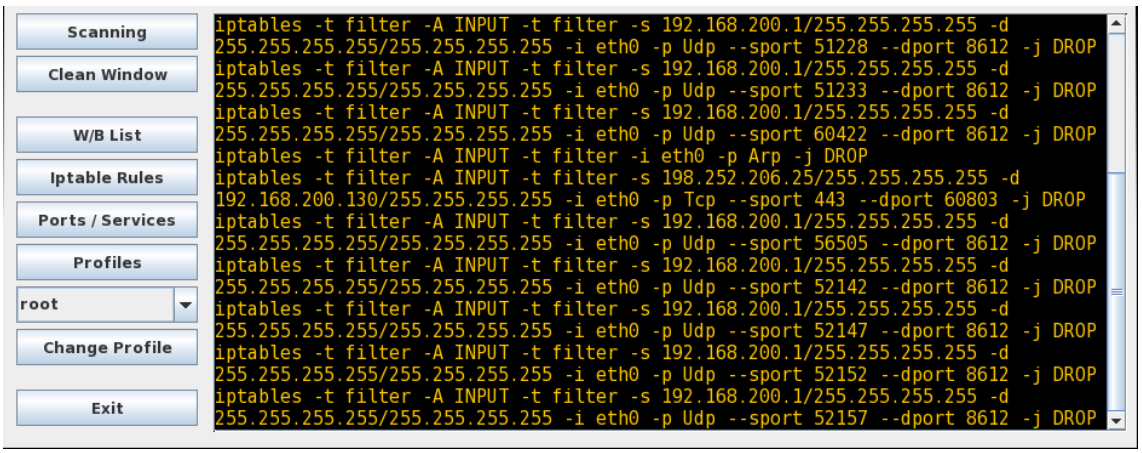
Všechny hodnoty, které jsou vázány čistě na daný uživatelský profil, jsou získány z properties souboru a zkontrolována jejich hodnota. Pokud by některé nebyly nalezeny, ať už konkrétní obsah nebo celá položka, bude záznam obnoven podle nastavení zdrojového profilu "default".

Jedinou výjimku představuje ukazatel se záznamem hardwarové adresy prvního zařízení v lokální síti, tedy takové, které přímo přijímá komunikace z místního síťového rozhraní. Pokud je tento záznam v souboru properties nalezen, je skrze volání metody pro odstranění klíče vymazán, aby se aplikace mohla opětovně adaptovat. Tento krok je důležitý zejména v případě mobilního zařízení, za předpokladu nevyužívání více samostatných účtů, nebo v případě, kdy by byla nezbytná výměna takového zařízení. V průběhu vlastní činnosti aplikace bude hodnota obnovena a nastavena na aktuální údaj, jak bylo popsáno na konci kapitoly **4.1.3 Vlákno "snooper"**.

4.3.2 Hlavní okno

Hlavní ovládací rozhraní aplikace je zastoupeno jednoduchým oknem. V levé části jsou zastoupena tlačítka, která uživateli zpřístupňují akce pro nastavování, správu a ovládání celé aplikace, zatímco pravá strana je okupována hlavní obrazovkou, která slouží pro vypisování zpráv aplikace. Převážně se jedná o informace o nově zaneseném pravidle do seznamu aktivních.

Kompletní rozhraní však bylo rozděleno pomocí dialogových oken, která vždy seskupují podobné činnosti. V případě nutnosti mohou i samotná dialogová okna vyvolat další, která nabídnou zase detailnější kontrolu nad danou oblastí.



Obrázek 13 - Hlavní okno aplikace

První dvě ovládací tlačítka představují jednoduché akce, které přímo ovlivňují chod aplikace, respektive vzhled hlavního okna. Volba "Scanning" slouží pro přepínání režimu zachytávání paketů. Při každém stisknutí je nový stav oznámen na obrazovce. Druhá možnost, "Clean Window", zajistí vymazání aktuálního obsahu hlavní obrazovky.

Tlačítko "W/B List" vyvolá dialogové okno se seznamem všech aktuálně platných povolených a zakázaných adres pro aktivní uživatelský profil.

Volba "Iptable Rules" zobrazí okno s aktuálně zanesenými pravidly pro firewallové rozhraní iptables [14].

"Ports / Service" nabízí přístup k seznamu aplikací nastavených dvojic služby a přiřazeného portu. Všechny záznamy obsahují i hodnotu přípustného transportního protokolu a volitelně i komentář či popis.

"Profiles" umožňuje správu uživatelských profilů v rámci aplikace. Při zobrazení je automaticky skryt ukázkový nebo též zdrojový profil "default", který není určen ani pro úpravy ani pro aktivní použití.

Výběrové pole slouží pro zvolení aktivního účtu. V rámci roletky jsou vždy dostupné všechny uživatelské účty, ať už vytvořené automaticky při spuštění aplikace, nebo později ručně skrze odpovídající rozhraní. Pro potvrzení volby a záměru změny uživatelského účtu slouží tlačítko "Change Profile". Po jeho stisknutí se spustí akce přepnutí uživatelského kontextu, která byla popsána v rámci kapitoly **4.1.5 Vlákno "rotor"** při popisu odpovědného vlákna.

Poslední tlačítko hlavního okna slouží k bezpečnému ukončení celé aplikace. Tento proces byl popsán na konci kapitoly **4.1 Vlákna**.

4.3.3 Dialogy

V předchozí kapitole byla několikrát zmíněna dialogová okna, která dále rozšiřují možnosti ovládání aplikace, aniž by současně zvyšovali nepřehlednost kontrolních prvků.

Všechna dialogová okna, která mají zobrazovat seznam prvků, jsou vždy tvořena pomocí struktury *AbstractTableModel*, která umožňuje velice snadné vykreslení na základě několika mála implementovaných metod. Jedná se o seřazení a rozdělení modelového objektu do jednotlivých polí řádku a samozřejmě načtení všech požadovaných objektů pro takové zobrazení. Získání objektů, pro každé dialogové okno je použit odpovídající modelový objekt, je prováděno skrze třídy implementující rozhraní, které definuje požadované schopnosti při zacházení se strukturami sdružujících tyto objekty.

```
private IPTableRuleService itrs = null;
private final String[] columns = { "table", "chain", "source",
                "destination", "inInterface", "outInterface",
                "protocol", "sourcePort", "destinationPort", "state",
                "reject", "action", "whoami", "description" };

public IPTablesRuleTableModel(IPTableRuleService itrs){
    this.itrs = itrs;
}

public void refresh(){
    fireTableDataChanged();
}

@Override
public int getColumnCount() {
    return columns.length;
}

public String getColumnName(int col){
    return columns[col];
}
```

```
@Override
public int getRowCount() {
    return itrs.getCount();
}

@Override
public Object getValueAt(int row, int col) {
    IPTablesRule rule = getItemByRowIndex(row);

    switch (col) {
          case 0: return rule.getTable();
          case 1: return rule.getChain();
          case 2: return rule.getSource() + "/" +
                Analyzer.numericMaskFromIpMask(rule.getsMask());
          case 3: return rule.getDestination() + "/" +
                Analyzer.numericMaskFromIpMask(rule.getdMask());
          case 4: return rule.getinInterfaceName();
          case 5: return rule.getoutInterfaceName();
          case 6: return rule.getProtocol();
          case 7: return rule.getSourcePort();
          case 8: return rule.getDestinationPort();
          case 9: return rule.getState();
          case 10: return rule.getReject();
          case 11: return rule.getAction();
          case 12: return rule.getWhoami();
          case 13: return rule.getDescription();
          default: return "";
    }
}

public IPTablesRule getItemByRowIndex(int row){
    return itrs.getAll().get(row);
}

public boolean isCellEditable(int row, int col){
    return false;
}
```

Ukázka třídy *IPTablesRuleTableModel* struktury *AbstractTableModel* určené pro vytvoření a zobrazení seznamu aktivních pravidel rozhraní iptables [14]. Privátní pole *columns* sdružuje názvy odpovídajících sloupců pro výslednou tabulku.

Nejzajímavější je potom metoda *getValueAt()*, která podle zadaného řádku a čísla sloupce vrátí odpovídající hodnotu z modelového objektu pro vykreslení. V tomto případě se jedná o získávání parametrů instance objektu typu *IPTablesRule*. Pro určení, která položka má být vrácena je využito struktury *switch*, která na základě indexu, zde tedy čísla sloupce, zavolá konkrétní metodu a vrátí výsledek do místa volání metody.

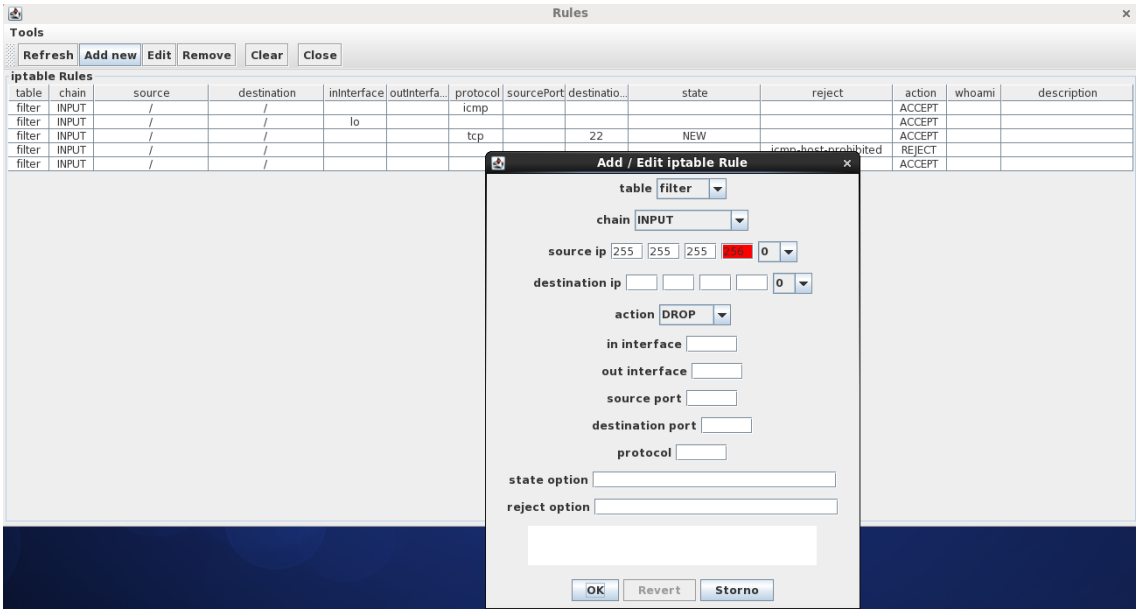
Každé dialogové okno se seznamem prvků, poskytuje metody pro jejich správu. Tyto metody jsou specifikované v rámci odpovídajícího rozhraní. Následuje ukázka rozhraní použitého pro dialogové okno vykreslené výše uvedenou třídou.

```
public interface IPTableRuleService {

    public void addNew(IPTablesRule rule);

    public void edit(IPTablesRule oldOne, IPTablesRule newOne);
```

```
    public int getCount();

    public List<IPTablesRule> getAll();

    public void clear();

    public void remove(IPTablesRule rule);
}
```

Samozřejmě ne všechny představené metody slouží pro zpracování uživatelských akcí. V tomto případě se jedná pouze o akce "addNew", "edit", "clear" a "remove".

První dvě v rámci obsloužení vyvolají dodatečná dialogová okna s formulářem pro vytvoření či úpravu pravidla zastoupeného modelovou třídou. Každý takovýto formulář obsahuje validaci všech uživatelských vstupů, které jsou vyžadovány a musí splňovat určité formáty. V úvahu samozřejmě bere i situaci, kdy daná položka není požadovaná, ale pokud by byla vložena, musí mít odpovídající tvar. Odchylky nebo nalezené problémy jsou při pokusu o uložení změn uživateli skrze samotný dialog



Obrázek 14 - Dialog pro přidání / úpravu pravidla pro firewall

prezentovány. Kromě změny či vytvoření nového objektu slouží okna i pro detailnější a přehlednější zobrazení vybraného objektu a všech jeho vlastností, které není možné přehledně zobrazit v tabulkovém formátu.

Poslední dvě umožňují vymazávání prvků, buď všech, nebo jednoho zvoleného. Ne každý dialog podporuje hromadné vymazání obsažených prvků. Tato akce není povolena u seznamu služeb a portů a u přehledu uživatelských účtů. Poslední zmíněný také obsahuje ochranu proti odstranění aktuálně nastaveného primárního účtu aplikace.

Některé dialogy podporují i vyhledávání položek podle hodnot jejich atributů. Tuto funkci podporují dialogy se zobrazením seznamu všech zakázaných a povolených adres a seznamu platných kombinací služeb a čísla portu.

4.3.3.1 TableService

V předchozí kapitole byly zmíněny uživatelské akce v rámci ovládání prostřednictvím dialogových oken. Všechny tyto akce jsou dostupné skrze implementační třídy, které představují implementace definovaných rozhraní (jedno bylo v předchozí kapitole představeno) v závislosti na konkrétním řešení. V tomto případě se jedná o komunikaci a správu údajů ukládaných do odpovídajícího souboru, tedy operace nad souborovým systémem. Stejně jednoduše lze ale vytvořit implementaci, která bude jako úložiště využívat například databázi nebo vzdálený server.

Podobně jako v předchozí kapitole i zde bude představen jeden konkrétní příklad implementace této služby pro správu pravidel firewallového rozhraní. Ostatní řešení představují identické případy pouze s jinými modelovými objekty a odlišnou strukturou pro ukládání změn.

```
public void addNew(IPTablesRule rule) {
    List<String> multiuser = properties.getProperties(whoami +
                                                       ".multiUser");
    for(String user:multiuser)
          bridge.appendLineToList(
                "./data/" + properties.getProperty(user + ".file"),
                rule.setWhoami(whoami).toString());

    bridge.forwardCommand(rule.getAppendRule());

}
```

Metoda třídy implementující rozhraní *IPTableRuleService* pro přidání nového záznamu do seznamu všech aktivních pravidel. Prvním krokem je získání seznamu propojených uživatelských účtů. Následně dochází k zápisu pravidla do souboru každého propojeného profilu skrze odpovídající metodu na třídě zpřístupňující souborový systém. Pravidlo se označí aktivním profilem jako indikátor původu. Posledním krokem je přidání nového záznamu do firewallového rozhraní iptables [14].

```
public void edit(IPTablesRule oldOne, IPTablesRule newOne) {
    List<String> temp = bridge.loadList("./data/" +
                      properties.getProperty(whoami + ".file"));
    int index = temp.indexOf(oldOne.toString());
    if(index != -1){
          temp.set(index, newOne.setWhoami(whoami).toString());
          bridge.saveList(
                "./data/" + properties.getProperty(whoami + ".file"),
                temp,
                false);
          bridge.forwardCommand(oldOne.getDeleteRule());
          bridge.forwardCommand(newOne.getAppendRule());
    }
}
```

Metoda pro úpravu pravidla změní záznam pouze pro účet, pod kterým bude provedena. Prvním krokem je načtení aktuálního seznamu ze souboru. Následně se vyhledá původní pravidlo, aby ho bylo možné nahradit.

Bude-li záznam podle předpokladu nalezen, uloží se jeho pozice v seznamu. Na základě tohoto indexu dojde k přepsání novým pravidlem, které zaujme jeho místo. Skrze metodu třídy pro přístup k souborovému systému se nový seznam uloží do souboru, čímž dojde k aktualizaci všech záznamů.

Posledním krokem je odebrání původního pravidla z firewallu a přidání nové verze.

```
public void clear() {
    if(whoami != null && properties != null){
          bridge.saveList(
                "./data/" + properties.getProperty(whoami + ".file"),
                new ArrayList<String>(),
                false);
          bridge.forwardCommand(IPTablesRule.getFlushRule(
                            IPTablesRule.IPTABLES_CHAIN_INPUT));
          bridge.forwardCommand(IPTablesRule.getFlushRule(
                            IPTablesRule.IPTABLES_CHAIN_OUTPUT));
    }
}
```

Metoda sloužící pro odstranění všech pravidel v seznamu. Vymazání probíhá uložením prázdného seznamu do souboru aktuálního profilu. Následně jsou vymazána aktivní pravidla z firewallového rozhraní iptables [14].

```
public void remove(IPTablesRule rule) {
    if(whoami != null && properties != null){
          bridge.forwardCommand(rule.getDeleteRule());
          List<String> temp = bridge.loadList(
                      "./data/" + properties.getProperty(
                                        whoami + ".file"));

          temp.remove(rule.toString());
          bridge.saveList(
                "./data/" + properties.getProperty(whoami + ".file"),
                temp,
                false);
          }
    }
```

Metoda začíná odstraněním pravidla z firewallového rozhraní. V dalším kroku je načten seznam pravidel ze souboru aktivního uživatelského účtu. Odstranění odpovídající instance probíhá jednoduchým voláním metody *remove()* na objektu seznamu. Na rozdíl od úpravy zde není nutné kontrolovat existenci pravidla. Pokud by z nějakého důvodu nebylo pravidlo nalezeno, nedojde k odstranění žádného jiného ani k chybě aplikace. Posledním krokem je uložení upraveného seznamu pravidel zpět do souboru aktivního uživatelského profilu.

4.4 Model

V této kapitole bych blíže představil a popsal modelové třídy aplikace. Přestože žádná z dále uvedených tříd nemá významný dopad na chod aplikace, jsou důležité z hlediska organizace dat a jejich správy. Většina struktury těchto tříd je tvořena jednoduchými metodami pro nastavení nebo získání hodnoty obsažené v atributech třídy, tzv. settery a gettery. Alternativu tvoří tzv. konstruktory, které umožňují vytvoření

instance dané třídy s případným úvodním nastavením jednotlivých atributů. Všechny tyto modelové objekty jsou vytvořeny s respektem k návrhovému schématu, který lze označit jako "řetězení metod" [25] nebo "plynulé rozhraní" [26], přestože se nejedná o zcela ekvivalentní označení pokud jde o jejich význam.

Prvním takovýmto objektem je třída *UserProfile*, která v rámci svých atributů sdružuje informace o uživatelském účtu a jím nastavených hodnotách. Slouží především pro přehlednou správu jednotlivých účtů prostřednictvím odpovídajícího dialogového okna aplikace.

Třída disponuje konstruktorem, který dokáže vytvořit instanci pomocí načtení odpovídajících hodnot z prostředí properties souboru skrze odpovědnou třídu, které byla věnována kapitola **4.2.3 PropertyLoader**.

Objekt *SimpleTableItem* byl primárně vytvořen čistě za účelem zpřehlednění a správy uživatelem zadaných povolených nebo zakázaných adres včetně síťové masky v odpovědném dialogovém okně. Objekt je však využíván i při procesu zjednodušování pravidel firewallu, který byl popsán v kapitole **4.1.4.1 Zjednodušení pravidel**.

Dalším objektem je třída *ServiceAndPort*, která představuje reprezentaci záznamů spojujících službu s odpovídajícím číslem portu a přenosovým protokolem. Kromě podpory pro zobrazení a správu všech pravidel skrze odpovědné dialogové okno aplikace, představuje třída i rozhraní pro kontrolu v průběhu analýzy průchozí komunikace.

```
public boolean equals(Object obj) {
    if(obj instanceof ServiceAndPort){
          ServiceAndPort object = (ServiceAndPort)obj;

          boolean equality = true;

          if(!object.getService().isEmpty())
                equality = this.getService()
                            .equalsIgnoreCase(object.getService());
          else equality = false;

          if(!object.getPort().isEmpty() && equality)
                equality = this.getPort()
                            .equalsIgnoreCase(object.getPort());
          else equality = false;

          if(!object.getProtocol().isEmpty() && equality)
                equality = this.getProtocol()
                            .equalsIgnoreCase(object.getProtocol());

          return equality;
    } else return false;
}
```

Pro účely analýzy bylo potřebné přepsat metodu *equals()*, aby brala v průběhu porovnání ohled na určité stavy objektu, kdy nemusí nutně být všechny jeho atributy získané z otisku zachyceného paketu.

Metoda se provede pouze v případě, že předaný objekt pro kontrolu je instance třídy *ServiceAndPort*. Následně dojde k přetypování předaného objektu pro účely dalšího porovnávání. Pokud předaný objekt obsahuje službu, dojde k porovnání těchto

záznamů. Výsledek je uložen do proměnné, která představuje návratovou hodnotu a určuje i další vývoj. Pokud by kterékoli porovnání skončilo záporným výsledkem, nebudou ostatní kontroly provedeny.

Obdobně probíhá porovnání i dalších atributů objektu, pokud je předaná instance obsahuje. Dochází ke zkontrolování čísla portu a přenosového protokolu. Výsledek všech analýz je následně předán jako návratová hodnota metody.

Předposledním modelovým objektem je třída *NetworkInterfaceCard*, která sdružuje informace o síťovém zařízení počítače. Jediná instance tak může skrze celou aplikaci udržovat potřebná data pro její činnost.

```
public NetworkInterfaceCard(PcapIf pcapIf){
    this.name = pcapIf.getName();
    byte[] temp = null;

    try {
          temp = pcapIf.getHardwareAddress();
    } catch (IOException e) {
          e.printStackTrace();
    }

    if(temp != null) mac = temp;

    for(PcapAddr addr : pcapIf.getAddresses())
          if(addr.getAddr().getFamily() == PcapSockAddr.AF_INET){
                this.address = addr.getAddr().getData();
                this.broadcast = addr.getBroadaddr() == null ?
                            null : addr.getBroadaddr().getData();
          }
}
```

Kromě běžné verze konstruktoru třídy, nabízí se i verze přijímající instanci typu *PcapIf* z objektů knihovny jNetPcap [20]. Prvním krokem je získání jména rozhraní z atributu předané instance. Následně je proveden pokus o získání hardwarové adresy rozhraní. Pokud se podaří data získat, jsou uložena do atributů nově vytvářené instance objektu.

Posledním krokem je získání platných adres a sice ip adresy síťového rozhraní a odpovídající broadcast verze adresy pro zajištění komunikace v rámci lokální sítě. Kvůli tomuto účelu jsou postupně procházeny všechny adresy obsažené v objektu typu *PcapIf*. Hodnoty jsou však získány pouze z objektu rodiny adres v požadovaném standardu.

Posledním objektem je třída reprezentující jak otisk zachyceného komunikačního paketu, tak potencionální pravidlo pro firewall *IPTablesRule*.

Mezi největší přednosti této modelové třídy patří schopnost kompletního převodu do formy textového řetězce a opětovné zformování. Při tomto procesu je nezbytné brát v úvahu, že ne všechny parametry mohou být nastaveny nebo využity.

```
public String toString() {
    String rule = getChain();

    if(!table.isEmpty()) rule = rule + " -t " + getTable();
    if(!source.isEmpty()) rule = rule + " -s " + getSource();
    if(!sMask.isEmpty()) rule = rule + "/" + getsMask();
```

```
    if(!destination.isEmpty()) rule = rule + " -d " +
                getDestination();
    if(!dMask.isEmpty()) rule = rule + "/" + getdMask();
    if(!inInterfaceName.isEmpty()) rule = rule + " -i " +
                getinInterfaceName();
    if(!outInterfaceName.isEmpty()) rule = rule + " -o " +
                getoutInterfaceName();
    if(!protocol.isEmpty()) rule = rule + " -p " + getProtocol();
    if(!sourcePort.isEmpty()) rule = rule + " --sport " +
                getSourcePort();
    if(!destinationPort.isEmpty()) rule = rule + " --dport " +
                getDestinationPort();
    if(!state.isEmpty()) rule = rule + " -m state --state " +
                getState();
    if(!action.isEmpty()) rule = rule + " -j " + getAction();
    if(!reject.isEmpty()) rule = rule + " --reject-with " +
                getReject();
    if(!whoami.isEmpty()) rule = rule + " -whoami " + getWhoami();
    if(!macSource.isEmpty()) rule = rule + " $ms " + getMacSource();
    if(!macDestination.isEmpty()) rule = rule + " $md " +
                getMacDestination();
    if(!description.isEmpty()) rule = rule + " _" +
                getClearDescription();

    return rule.trim();
}
```

V rámci převodu dochází k postupnému napojování hodnot atributů, pokud byly řádně nastaveny. Každý atribut je kromě své vlastní hodnoty do řetězce připojen skrze ukazatel, který je shodný s přepínači voleb firewallového rozhraní iptables [14]. Tato kompatibilita zajišťuje možnost jednoduchého přenosu pravidla přímo do prostředí firewallu. Jakmile jsou projité všechny dostupné atributy, je výsledný řetězec ještě oříznut pro odstranění případných mezer před a na konci řetězce.

Největší výhodou tohoto přístupu je ale možnost zpětného získání shodného objektu opětovným převodem z textového řetězce.

```
public IPTablesRule constructPopulate(String rule){
    if(rule.contains("iptables")) rule =
                    rule.substring(rule.indexOf(" ") + 1);
    if(rule.charAt(0) != '-') rule = IPTABLES_APPEND + " " + rule;
    String[] list = rule.split(" ");

    for(int i = 0; i < list.length - 1; i += 2){
          String name = list[i];
          String param = list[i+1];

          if(name.equals("-A") || name.equals("-I")
                      || name.equals("--insert")
                      || name.equals("--append")
                      || name.equals("--delete")
                      || name.equals("-D"))
                setChain(param);
          if(name.equals("-s") || name.equals("--source"))
                setAddressAndMask(param, true);
          if(name.equals("-d") || name.equals("--destination"))
                setAddressAndMask(param, false);
          if(name.equals("-i") || name.equals("--in-interface"))
                setinInterfaceName(param);
```

```
            if(name.equals("-o") || name.equals("--out-interface"))
                  setoutInterfaceName(param);
            if(name.equals("-p") || name.equals("--proto"))
                  setProtocol(param);
            if(name.equals("--sport")) setSourcePort(param);
            if(name.equals("--dport")) setDestinationPort(param);
            if(name.equals("--state")) setState(param);
            if(name.equals("-j") || name.equals("--jump"))
                  setAction(param);
            if(name.equals("-t") || name.equals("--table"))
                  setTable(param);
            if(name.equals("--reject-with")) setReject(param);
            if(name.equals("-whoami")) setWhoami(param);
            if(name.equals("$ms")) setMacSource(param);
            if(name.equals("$md")) setMacDestination(param);
      }

      if(rule.contains("_"))
            setDescription(rule.substring(rule.lastIndexOf("_") + 1));

      return this;
   }
```

Zpětný převod z podoby textového řetězce probíhá pomocí ukazatelů určujících typ následujících dat. Prvním krokem je úprava řetězce, kdy se odstraňuje případný začátek, který nenese žádnou informaci. Dalším krokem je případné připojení ukazatele na začátek řetězce. Následně je pravidlo rozřezáno podle dělícího symbolu, který je zastoupen mezerou.

Výsledné pole obsahující jednotlivé bloky původního textového řetězce je v rámci cyklu procházeno po dvojicích. Vždy je načteno označení, které představuje ukazatel, a následně jeho vlastní hodnota. Přestože by šlo jednotlivé přiřazování uskutečnit skrze rozhraní *switch*, je zde využito sestavy podmínek. Pokud jméno hodnoty odpovídá očekávanému tvaru, je hodnota uložena do atributu třídy.

Po projití všech hodnot v poli, získaného rozřezáním původního textového řetězce, následuje poslední kontrola a sice ověření existence komentáře. Pokud je obsažen, uloží se jeho hodnota do pole. Tato kontrola probíhá odlišně, protože samotný komentář může být tvořen téměř libovolnými symboly včetně zde použitého dělícího znaku mezery.

5 Testování realizovaného řešení

Testování vyvinuté aplikace probíhalo v několika fázích a prostředích. První testy se uskutečnily v rámci virtuálního stroje s operačním systémem CentOS 6.5, pod kterým probíhal i samotný vývoj. Tyto testy představovaly převážně ověřování funkcionality jednotlivých vlastností, popřípadě identifikace nefunkčních částí. Výstupy těchto testů vytvářely přímé požadavky na změnu nebo vylepšení designu či konkrétních částí kódu aplikace.

Právě v rámci těchto testů se ukázalo nezbytné vytvořit a upravit vzhled kontrolních prvků, které uživateli umožňují nastavování a kontrolu celé aplikace. Současně se postupně ověřila možnost využití prvků operačního systému pro úvodní nastavení filtrovacích možností aplikace.

Je však vhodné zmínit, že v průběhu testování aplikace byly i některé komponenty nakonec zrušeny. Jedná se například o přímé spojení s rozhraním iptables [14], kdy bylo umožněno i převzetí aktivních pravidel a jejich integrace v rámci aktivního profilu. Jako kompenzace bylo umožněno dynamické přepínání uživatelských profilů a tím i okamžitá změna chování firewallu skrze změny aktivních pravidel, stejně jako změnu chování celé aplikace.

Další fáze testů byla zaměřena čistě na aktivní úpravu firewallu pomocí přidávání nových pravidel na základě analýzy probíhající komunikace. V průběhu testu se ukázala potřeba úpravy času shromažďování statistických dat, počtu průchozích paketů identického původu, podle potřeb uživatele.

Současně se prokázala správnost návrhu s využitím speciálního vlákna pro zajištění předávání zpráv mezi zachytávácím a analytickým vláknem, jak bylo popsáno v kapitole **3.2 Činnosti vláken**.

V průběhu těchto testů se zkoušela i schopnost aplikace přizpůsobovat se probíhající komunikaci a vůbec schopnost analýzy jednotlivých paketů. Nejen že se prokázala funkčnost adaptabilní úpravy firewallových pravidel skrze analýzu, ale i prakticky nulový dopad na rychlost síťové komunikace. Zároveň se však prokázalo, že pevné nastavení prahového počtu identických paketů není nejvhodnější obzvláště v případě kontinuálního přístupu k webovému obsahu, jako jsou videa či soubory v cloudových uložištích. S ohledem na tuto skutečnost byl návrch aplikace rozšířen o možnost zapnout či vypnout zachytávání komunikačních paketů. V důsledku tak je možné při zachování pevné mezní hodnoty přistoupit i k datům, která by jinak byla mylně označena jako podezřelá a komunikace následně zablokována.

Příjemným bonusem je skutečnost, že aplikace dokáže nativně zpracovávat komunikaci procházející síťovým rozhraní v obou směrech a rozeznávat i směr jednotlivých paketů. Je tak možné nejen chránit počítač nebo vnitřní síť před vnějšími hrozbami, ale současně i uzamknout problematické vysílání z lokálního počítače například v důsledku virové nákazy, která se pokouší o přístup k webovým zdrojům nebo rozeslat vlastní kopie do místní sítě.

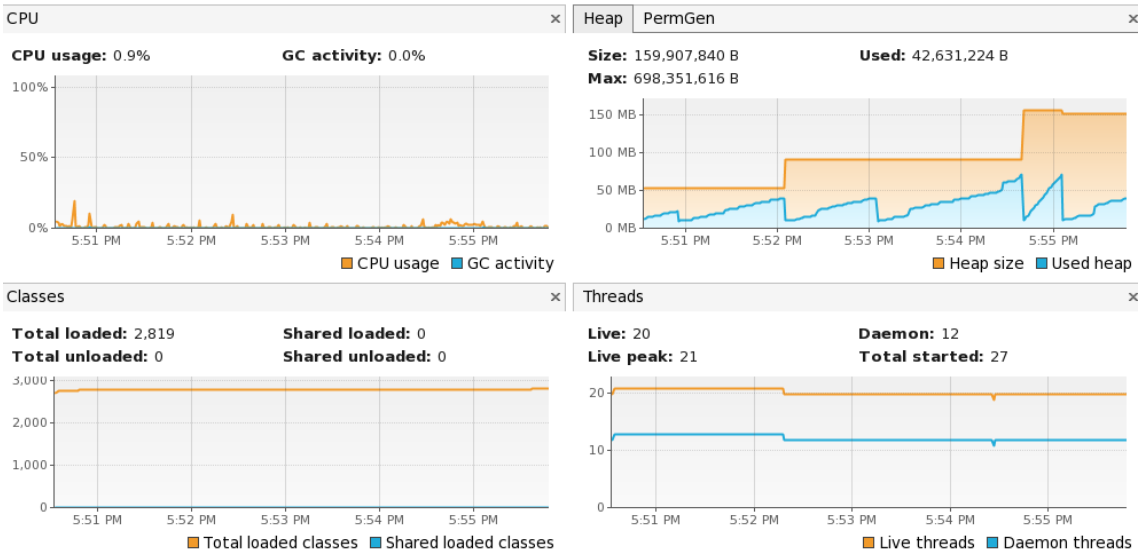
Rychlost reakce firewallu na zachycenou komunikaci je vždy limitována především nastaveným čekacím intervalem pro shromažďování odpovídajících statistických dat pro vyhodnocení. Jakmile jsou zachycená data předána k analýze, je implementace

případných nových pravidel otázka jen několika mála sekund v závislosti na okamžitém vytížení prostředků počítače a množství originální zachycené komunikace.

Poslední testy probíhaly v rámci reálného prostředí v počítačové laboratoři Univerzity Hradec Králové. Aplikace byla nastavena na počítačích s čerstvě připraveným prostředím operačního systému CentOS 6.5. Zkoušena byla především schopnost aplikace se přizpůsobovat a aplikovat jednotlivá pravidla obdobně jako v případě virtuálního počítačového systému. Dalším cílem bylo otestování funkcionality "spárování" s prvním zařízením v rámci hierarchie sítě pomocí jeho fyzické adresy jako prevence možných podvržených paketů. Tato schopnost se ukázala fungovat v rámci předpokladů, nicméně se objevil problém v případě, kdy jednotlivé počítače v rámci místní sítě nekomunikovali skrze zařízení upravující zdrojovou a cílovou fyzickou adresu průchozích paketů. V takovém případě se počítač pevně spojí s jediným dalším a jakákoli příchozí komunikace ze zdroje s odlišnou fyzickou adresou je automaticky zablokována.

5.1 Zátěž OS

V průběhu testování funkčnosti aplikace se ukázalo vhodné otestovat i dopad její přímé činnosti na výkon operačního systému a konzumaci zdrojů. Tyto testy probíhali v rámci virtuálního počítače pomocí nástroje pro měření aktivity běhového prostředí objektového jazyku JAVA VisualVM [31].



Obrázek 15 - Vytížení procesoru, využití paměti, načtené objekty a spuštěná vlákna, VirtualVM [31]

Výše uvedený obrázek reprezentuje samotné měření v průběhu testovacího běhu aplikace v rámci virtuálního počítače s operačním systémem CentOS 6.5. Zobrazený stav odpovídá přibližně patnácti minutovému běhu. V jeho průběhu se ukázaly zcela zanedbatelné výkyvy ve využití procesoru pro výpočty a to včetně probíhajících analýz zachycené komunikace. Průměrné vytížení procesoru nedosahovalo ani 1% celkového dostupného výkonu; virtuální stroj byl simulován s dvou jádrovým procesorem.

Počet načtených tříd a spuštěných vláken je v rámci běhu aplikace prakticky konstantní s nepatrnými výkyvy stavu v obou směrech.

Nejpodstatnější je však využití dostupné operační paměti. Od spuštění aplikace došlo k dvojímu zvětšení přiřazené paměti na celkových 150MB, přestože v okamžiku maximálního vytížení bylo využito přibližně 50MB paměti. Kolísání je dáno postupným shromažďováním zachytávané komunikace předtím, než bude předána analytickému vláknu pro další zpracování. V průběhu analýzy je následně pamět uvolňována, protože dochází k odmazávání zkontrlovaných objektů reprezentujících zachycenou komunikaci.

Threads: 25 **Total CPU Time [ms]:** 32,701

Thread Name	Thread CPU Time [...	Thread CPU Time [ms]
snooper		12,862.177 (0.0%)
RMI TCP Connection(1)-127.0.0.**1**		7,754.748 (0.0%)
AWT-EventQueue-0		3,653.819 (0.0%)
DisposableGC		2,583.32 (0.0%)
AWT-XAWT		1,446.207 (0.0%)
RMI TCP Connection(3)-127.0.0.**1**		1,434.46 (0.0%)
enforcement		611.253 (0.0%)
DestroyJavaVM		589.607 (0.0%)
***** JFluid Monitor thread *****		527.045 (0.0%)
JMX server connection timeout 26		523.373 (0.0%)
rotor		177.827 (0.0%)
***** Profiler Agent Communication Thread**		147.686 (0.0%)
TimerQueue		102.242 (0.0%)
Attach Listener		86.075 (0.0%)
Thread-1		80.731 (0.0%)
Thread-0		55.962 (0.0%)

Obrázek 16 - Využití procesoru jednotlivými vlákny aplikace, VirtualVM [31]

Vzhledem k využití principu více vláknové aplikace je vhodné změřit dopad jendotlivých vláken na využívání prostředků aplikace. Výše uvedený obrázek reprezentuje zachycená vlákna v rámci běhu aplikace skrze prostředí měřícího nástroje VirtualVM [31]. Dle očekávání největší aktivity dosahuje vlákno určené pro zachytávání průchozí komunikace, které je, vyjma okamžiků vypnutého sledování, vždy plně v provozu a čeká na jakýkoli průchozí paket. V rámci grafu se jedná o vlákno označené "snooper". Dalším vláknem v pořadí podle vytížení je analytické vlákno, zde reprezentováno pod jménem "enforcement". Dále následuje pomocné vlákno zajišťující komunikaci mezi ostatními a sice "rotor".

Podle výsledků se odvažuji tvrdit, že aplikace je schopna zvládnout i velké množství průchozí komunikace, aniž by příliš zatěžovala jak výpočetní prostředí počítače, tak samotné komunikační rozhraní.

6 Kritické zhodnocení testovaného řešení

Kromě výkonu samotné aplikace je nezbytné shrnout i schopnosti navrženého řešení a jejich dopad. Některé tyto vlastnosti již byly popsány v předchozí kapitole. Bylo by však nanejvýš vhodné dát tyto výsledky i do určitého kontextu, který by pomohl osvětlit jejich význam nebo účel v případě nasazení.

V první řadě je vhodné připomenout, že se jedná o firewallovou aplikaci, která se dokáže automaticky přizpůsobovat probíhající komunikace. Síla ochrany je tak závislá především na nastavení chování a vhodně zvoleném bezpečnostním prahu, který odlišuje nezávadné množství komunikace od podezřelého. Ačkoli si program nastaví tuto hodnotu podle interních záznamů, je v každém případě nezbytná její kontrola a úprava. Jen těžko lze odhadnout velikost normální průchozí komunikace. Tato hodnota závisí mimo jiné i na rychlosti síťové linky, typům přístupů, které jsou v rámci daného připojení provozovány a samozřejmě i samotném účelu chráněného stroje. Nelze očekávat, že bude vyhovovat jednotný práh pro počítač určený k prohlížení internetu a výkonnému serveru, se kterým mohou komunikovat i tisíce uživatelů. Zároveň není ani možné tuto hodnotu jakkoliv dynamicky odvodit, protože by to vyžadovalo zatížení komunikačního rozhraní v běžné hladině po určitou dobu.

Dalším faktorem je samozřejmě samotné automatické přizpůsobování aplikace a tedy i modifikace aktivních pravidel v rámci firewallového rozhraní iptables [14]. Zatímco běžné řešení funguje na základě pravidel a tedy veškerá průchozí komunikace je považována za bezpečnou, navržené řešení předpokládá, že i vlastním firewallem nezablokovaná spojení představují potencionální riziko. Může tedy docházet k nežádoucím omezením vlivem nárustu množství komunikace například při stahování většího množství dat, sledování on-line videa či odesílání objemných souborů, které nejsou za bežných okolností provozovány. V takovém případě je nutné buďto dočasně zastavit skenování komunikace, ale nezapomenout po ukončení činnosti zase ochranu aktivovat, nebo přepnout na upravený profil, který je uzpůsoben této zvláštní situaci, popřípadě si upravit stávající skrze nabízené možnosti, aby komunikaci nezahrnoval do bezpečnostních rizik a případně tedy i nezablokoval prostřednictvím výsledku analýzy zachycených paketů.

Určitým problémem může být provázání aplikace na první zařízení v rámci komunikace pomocí hardwarové adresy. Ideální je stav, kdy by se tímto zařízením stala výchozí brána pro spojení s vnějším světem. Pokud by však byla současně požadována i lokální komunikace mezi jednotlivými počítači, bude patrně nutné přidat záznam do nastavení, který by označil lokální síť jako povolenou a tedy i vyjmutou z běžného analytického procesu. Samozřejmě v takovém případě může dojít i k oslabení vlivu této ochrany, protože by případnému útočníkovi stačilo převzít adresu z interní sítě.

Dalším kritickým bodem se v průběhu jednotlivých testů ukázala i samotná instalace, která neprobíhala vždy úplně hladce. Hlavním důvodem je potřeba pomocné knihovny pro spuštění a její konkrétní verze, která do určité míry omezila schopnosti operačního systému. Jednalo se především o knihovnu libpcap, která do velké míry umožňuje síťovou komunikaci a v případě požadované starší verze se ukázaly některé bezdrátové adaptéry jako nerozpoznatelné. Následně i potřeba extrakce a zkopírování

několika podpůrných balíčků do určených adresářů v rámci operačního systému, přestože by tato činnost měla proběhnout automaticky v procesu instalace balíčku jNetPcap [20], který se však odmítal spustit a bylo tedy nutné jeho ruční zpracování.

6.1 Možná rozšíření

Přestože je aplikace plně funkční, zůstává zde i veliký prostor pro možná rozšíření a celková vylepšení jejích vlastností. Kromě zvýšení ochranných schopností se nabízí i alternativní způsoby použití.

V rámci procesu analýzy otisku zachyceného komunikačního paketu je prostor pro nasazení technologie či algoritmů, které by byly schopné odhalit případnou přítomnost závadného kódu. V současné verzi nejsou zaznamenávána data spojená s přenášeným obsahem právě kvůli absenci těchto algoritmů. Jedinou výjimku tvoří zachycené aktualizační pakety u nichž se získává i přenášená zpráva obsahující informace pro aktualizaci lokálního seznamu pravidel na základě analýzy na počítačích v rámci místní sítě.

Pro úspěšné zachycování veškeré průchozí komunikace musí být aplikace propojena s konkrétním síťovým rozhraním počítače. Toto propojení je možné definovat na úrovni uživatelského profilu. Každý jednotlivý účet však může být v daný čas propojen pouze s jediným rozhraním. Nastává tedy problematická situace u systémů, které provozují více připojení, popřípadě zastupují i roli směrovače v rámci sítě. Možným řešením se může jevit vícenásobné spuštění aplikace, kdy by každá instance chránila jedno konkrétní komunikační rozhraní. V takovém případě by však bylo nutné i správné nastavení profilů, aby se zajistilo, že nebude docházet ke ztrátám pravidel, která by byla vygenerována ostatními běžícími instancemi programu. Aplikace totiž po svém spuštění přebírá kontrolu nad firewallovým rozhraním. Prvním krokem je vymazání všech obsažených pravidel, aby nedocházelo ke kolizím s uloženými pravidly daného účtu. Bude tedy nezbytné zajistit provázání jednotlivých profilů, aby poslední spuštěná instance obsahovala vždy i pravidla vygenerována kontrolou odlišného síťového rozhraní počítače.

Jelikož by tato činnost vyžadovala veliké úsilí už jen při samotném spouštění, kdy by bylo nutné každou instanci buď spouštět z alternativního umístění, čímž by se narušila možnost sdílení pravidel v rámci uživatelských profilů, a nebo ze stejné instalace, pak by se však musel po každém spuštění přepnout aktivní uživatelský profil.

Už v současné verzi je vhodným nastavením možné využít aplikaci jako logovací a statistický program. Pokud by se extrémně snížil prah pro rozpoznávání nežádoucí komunikace a změnila výchozí akce v rámci generování pravidel, nesla by všechna vytvořená pravidla počet výskytů identického paketu a současně by neuzavírala firewall. Tímto způsobem by se dala sledovat veškerá komunikace daného počítače. Pro úplně správnou interpretaci těchto výsledků by však bylo vhodné implementovat i záznam času vytvoření odpovídajícího pravidla.

Aplikace v současnosti využívá IPv4, která je stále hojně rozšířená, přestože knihovna jNetPcap [20] obsahuje i podporu pro nadcházející standard IPv6. Pro zajištění budoucí použitelnosti aplikace je tedy vhodné implementovat i podporu pro

využití tohoto standardu, který v nedaleké budoucnosti nejspíše zcela převezme odpovědnost za směrování veškeré komunikace v prostředí internetu a lokálních sítí.

V průběhu testů se ukázala užitečná vlastnost aplikace, která by mohla posloužit i jako kontrolní mechanismus. V rámci sestavování testovací sítě bylo nezbytné ověřit spojení jednotlivých počítačů. Přestože bylo možné využít i standardní postupy pro zjišťování dostupnosti, aplikace sama byla schopna indikace aktivní komunikace. Snížením prahu je možné identifikovat příchozí komunikaci a bližší informace popisující typ zachyceného paketu. Je tak možné ověření spojení i v případě, kdy nelze použít běžné metody nebo jejich výstup není dostatečně vypovídající a současně není dostupný specializovaný nástroj pro tento účel.

Závěr

V rámci kapitoly **1 Rešerše dostupných řešení** byly představeny přístupy a principy, které by se mohly využít jako potencionální či funkční firewallová řešení s různým stupněm odolnosti proti útokům, zaměřením na problematiku nebo funkčními předpoklady. Aby však bylo možné tvrdit, že jsou opravdu odolná, bylo by potřeba několik různých nápadů či ideí zkombinovat, což už jen z některých požadavků není zcela možné.

I z tohoto důvodu byl v kapitole **3 Návrh řešení - logický model** představen nový přístup vycházející z použití knihovny jNetPcap [20] a programovacího jazyku JAVA pro vytvoření automatického firewallu pro operčaní systémy typu GNU/Linux. Byla popsána logika a rozdělění aplikace stejně jako nutnost využití několika vláken, která by představovala samotnou činnost aplikace.

Vytvořený nástroj byl následně detailně popsán v kapitole **4 Popis navrženého řešení.** Kapitola obsahuje velké množství ukázek kódu vyvinuté aplikace stejně jako ukázku její vlastní činnosti při analýze zachycované síťové komunikace a tvorbě pravidel pro firewallové rozhraní iptables [14].

Následně byla aplikace otestována v rámci virtuálního počítače, kde se testovaly samotné funcionality a výkon vyvinutého nástroje. V další fázi byla aplikace nasazena na reálný stroj s operačním systémem CentOS 6.5.

Vyvinutý nástroj prokázal schopnost aktivní analýzy komunikace a její případné blokování v okamžiku porušení podmínek, které si je schopen v základní míře i samostatně nastavit. Velkou výhodou je automatická schopnost blokování obou směrů komunikace a tedy i ochrana zbytku sítě v případě nakažení lokálního počítače a samozřejmě schopnost neustále se přizpůsobovat probíhající komunikaci. Vlastní činnost aplikace dynamicky vytváří firewallová pravidla na základě analýzy veškerého síťového provozu, který prošel chráněným komunikačním rozhraním počítače. Zajímavé je rozhodně využití hardwarových adres pro předejití přijetí podvržených paketů, přestože využití této funkce může do určité míry omezit lokální síťovou komunikaci.

I přesto, že se vyvinutý nástroj ukázal být v plné míře funkční, některé jeho funkcionality by mohly být ještě vylepšeny či rozšířeny. Z části byly tyto možnosti probrány v předchozí kapitole.

Zdroje

[1] Mishra, A., Agrawal, A., Ranjan R.: *Artificial Intelligent Firewall*, In ACAI '11 Proceedings of the International Conference on Advances in Computing and Artificial Intelligence, pp.204-207, 2011, 10.1145/2007052.2007094

[2] Hamelin, M.: *Preventing firewall meltdowns*, In Network Security, Volume 2010, Issue 6, pp.15-16, 06/2010, 10.1016/S1353-4858(10)70083-0

[3] Krueger, T., Gehl, Ch., Rieck, K., Laskov, P.: *TokDoc: A Self-Healing Web Application Firewall*, In SAC '10 Proceedings of the 2010 ACM Symposium on Applied Computing, pp.1846-1853, 2010, 10.1145/1774088.1774480

[4] Hamed, H., Al-Shaer, E.: *Dynamic Rule-ordering Optimization for High-speed Firewall Filtering*, In ASIACCS '06 Proceedings of the 2006 ACM Symposium on Information, computer and communications security, pp.332-342, 2006, 10.1145/1128817.1128867

[5] Moon, Ch-S., Kim, S-H.: *A Study on the Integrated Security System based Real-time Network Packet Deep Inspection*, In International Journal of Security and Its Applications, Volume 8, pp.113-122, 2014, 10.14257/ijsia.2014.8.1.11

[6] *NSS uncovers firewall shotcomings*, In Network Security, Volume 2011, Issue 5, pp.2,19, 05/2011, 10.1016/S1353-4858(11)70045-9

[7] *Firewall - ArchWiki* [online] 2014 [cit. leden 2015]. Dostupné z WWW: https://wiki.archlinux.org/index.php/firewalls

[8] *IP paket* [online] 2014 [cit. leden 2015]. Dostupné z WWW: http://home.zcu.cz/~lknakal/

[9] *Paket - Wikipedie* [online] 2014 [cit. leden 2015]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Paket

[10] *TCP/IP - Wikipedie* [online] 2014 [cit. leden 2015]. Dostupné z WWW: http://cs.wikipedia.org/wiki/TCP/IP

[11] *Referenční model ISO/OSI - Wikipedie* [online] 2014 [cit. leden 2015]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Referen%C4%8Dn%C3%AD_model_ISO/OSI

[12] *Firewally* [online] [cit. leden 2015]. Dostupné z WWW: http://www.fi.muni.cz/~kas/p090/referaty/2008-podzim/st/firewally.html

[13] *Firewall - Wikipedie* [online] 2014 [cit. leden 2015]. Dostupné z WWW: http://cs.wikipedia.org/wiki/Firewall#Paketov.C3.A9_filtry

[14] *iptables - ArchWiki* [online] 2002-2014 [cit. prosinec 2014]. Dostupné z WWW: https://wiki.archlinux.org/index.php/iptables

[15] *Správa linuxového serveru: Linuxový firewall, základy iptables - Linux E X P R E S* [online] 2010 [cit. leden 2015]. Dostupné z WWW: http://www.linuxexpres.cz/praxe/sprava-linuxoveho-serveru-linuxovy-firewall-zaklady-iptables

[16] *Správa linuxového serveru: Linuxový firewall, základy iptablesIV - Linux E X P R E S* [online] 2010 [cit. leden 2015]. Dostupné z WWW: http://www.linuxexpres.cz/praxe/sprava-linuxoveho-serveru-linuxovy-firewall-zaklady-iptables-3

[17] *Správa linuxového serveru: Linuxový firewall, základy iptables II - Linux E X P R E S* [online] 2010 [cit. leden 2015]. Dostupné z WWW: http://www.linuxexpres.cz/praxe/sprava-linuxoveho-serveru-linuxovy-firewall-zaklady-iptables-2

[18] *Správa linuxového serveru: Linuxový firewall, základy iptables III - Linux E X P R E S* [online] 2010 [cit. leden 2015]. Dostupné z WWW: http://www.linuxexpres.cz/praxe/sprava-linuxoveho-serveru-linuxovy-firewall-zaklady-iptables-1

[19] Buyya, R., Selvi, S. T., Chu, X.: *Object-Oriented Programming with Java: Essentials and Applications*, New Delhi: Tata McGraw-Hill, 2009, ISBN 9780070669086

[20] *jNetPcap OpenSource | Protocol Analysis SDK* [online] 2014 [cit. prosinec 2014]. Dostupné z WWW: http://jnetpcap.com/

[21] *java.com: Java + You* [online] 2014 [cit. prosinec 2014]. Dostupné z WWW: https://java.com/en/

[22] *C (programovací jazyk) - Wikipedia* [online] 2014 [cit. prosinec 2014]. Dostupné z WWW: http://cs.wikipedia.org/wiki/C_%28programovac%C3%AD_jazyk%29

[23] *services - Linux Command - Unix Commnad* [online] 2014 [cit. prosinec 2014]. Dostupné z WWW: http://linux.about.com/library/cmd/blcmdl5_services.htm

[24] *2.3 Opening a Network Interface for Capture | jNetPcap OpenSource* [online] 2014 [cit. prosinec 2014]. Dostupné z WWW: http://jnetpcap.com/?q=node/62

[25] *Method chaining - Wikipedia, the free encyklopedia* [online] 2014 [cit. prosinec 2014]. Dostupné z WWW: http://en.wikipedia.org/wiki/Method_chaining

[26] *Fluent interface - Wikipedia, the free encyklopedia* [online] 2014 [cit. prosinec 2014]. Dostupné z WWW: http://en.wikipedia.org/wiki/Fluent_interface

[27] *Create TCP Packet | jNetPcap OpenSource* [online] 2014 [cit. leden 2015]. Dostupné z WWW: http://jnetpcap.com/?q=node/621

[28] *java - Jnetpcap, preparing UDP/TCP/IP/ICMP packet - Stack Overflow* [online] 2014 [cit. leden 2015]. Dostupné z WWW: http://stackoverflow.com/questions/8141118/jnetpcap-preparing-udp-tcp-ip-icmp-packet

[29] *get protocol name | jNetPcap OpenSource* [online] 2014 [cit. leden 2015]. Dostupné z WWW: *http://jnetpcap.com/?q=node/618*

[30] *The ? : operator in Java* [online] 1997 [cit. leden 2015]. Dostupné z WWW: http://www.cafeaulait.org/course/week2/43.html

[31] *Home - Project Kenai, VisualVM* 2014 [cit. únor 2015]. Dostupné z WWW http://visualvm.java.net/

Přílohy

I. Archiv s vyvinutou aplikací: firewall.zip

Archiv ve formátu .zip obsahující nezbytné soubory pro zprovoznění vyvinuté aplikace na počítači s operačním systémem typu GNU/Linux. Obsažené soubory: app.jar - vyvinutá aplikace; jnetpcap-1.3.0-1.rhel.x86_64.tgz - balíček podprůných knihoven jNetPcap; libpcap-0.9.4-15.el5.x86_64.rpm - instalační balíček rozhraní libpcap v požadované verzi; READ.ME - popis postupu zprovoznění aplikace; start.sh - jednoduchý skript pro spuštění aplikace.

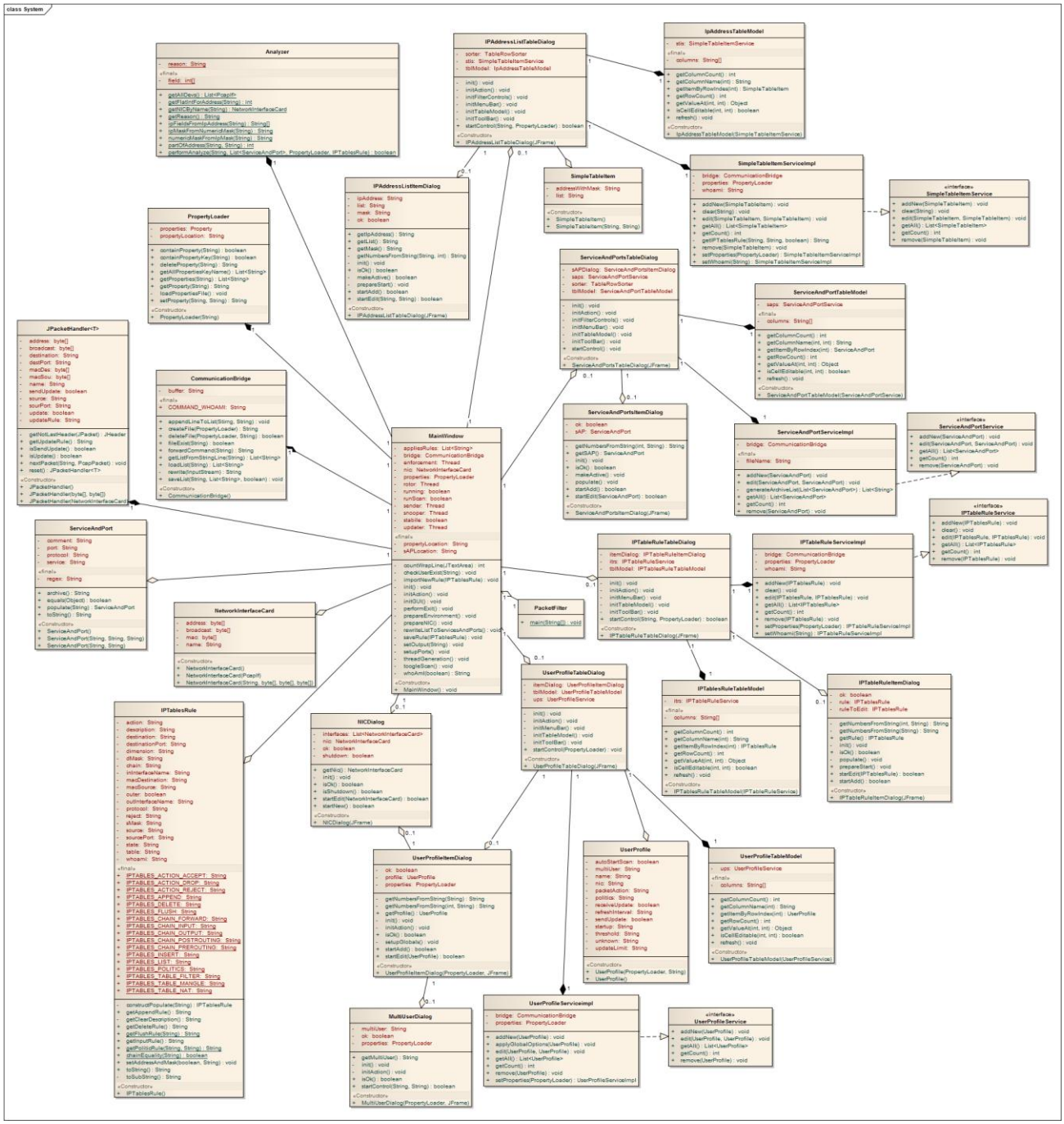
II. Archiv zdrojových souborů aplikace: packetFilter.zip

Archiv obsahující zdrojové soubory, data a vytvořenou aplikaci z programovacího prostředí Eclipse.

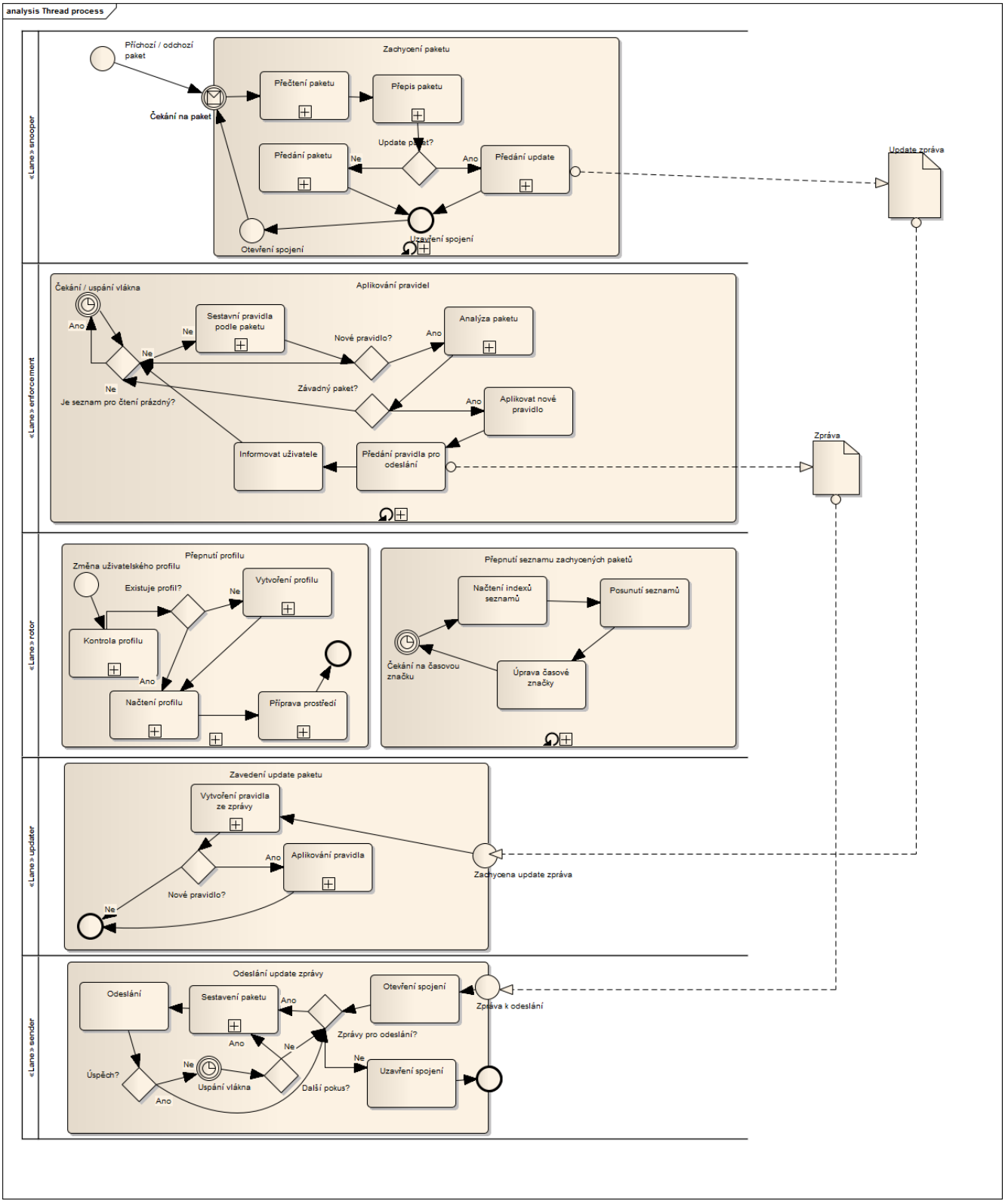
III. Soubor diagramu v prostředí Enterprise Architect: diagram.eap

Diagram tříd a balíčků a některých logických komponent aplikace (činnosti vláken) vytvořený v prostředí programu Enterprise Architect.

IV. Diagram tříd aplikace z prostředí programu Enterprise Architect



V. Diagram činnosti vláken aplikace z programu Enterprise Architect



VI. Zadání k závěrečné práci

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Petr Halický**

Obor studia: Aplikovaná informatika (2)
Jméno a příjmení vedoucího práce: **Josef Horálek**

Název práce:
Problematika a optimalizace firewall nad OS Linux

Název práce v AJ:
Problems and optimization of firewall Linux OS

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem práce je provést analýzu dostupných firewall řešení a navrhnout univerzální firewall s možností filtrování na L2 až L7 vrstvách modelu ISO/OSI realizovaný na OS Linux.

Osnova práce:
Úvod
Rešerše dostupných řešení
Představení principů firewall na OS Linux
Návrh řešení - logický model
Popis navrženého řešení
Testování realizovaného řešení
Kritické zhodnocení testovaného řešení
Závěr

Projednáno dne:

Podpis studenta Halický

Podpis vedoucího práce