Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Firewall pre pracovnú stanicu s OS Linux

Diplomová práca

2022
Bc. Jaroslava Kokavcová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Firewall pre pracovnú stanicu s OS Linux

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Jaroslav Janáček, PhD.

Bratislava, 2022
Bc. Jaroslava Kokavcová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jaroslava Kokavcová
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Firewall pre pracovnú stanicu s OS Linux
Firewall for Linux Workstation

Anotácia: Súčasťou OS Linux je subsystém firewallu, ktorý však neumožňuje priamo zohľadniť príslušnosť komunikácie ku konkrétnej aplikácii, ani momentálne požiadavky používateľa. Práca sa venuje analýze možností, návrhu a implementácii firewallu pre pracovnú stanicu s OS Linux, ktorý umožní filtrovať komunikáciu aj na základe príslušnosti ku konkrétnej aplikácii a na základe rozhodnutí používateľa, čím umožní používateľovi získať prehľad a kontrolu nad komunikačnými tokmi jednotlivých aplikácií.

Cieľ:
- analyzovať požiadavky na kontrolu komunikácie aplikácií používateľom
- preskúmať súčasný stav problematiky
- analyzovať možnosti riešenia
- navrhnúť a implementovať riešenie
- porovnať riešenie s existujúcim stavom a demonštrovať jeho prínos pre bezpečnosť

Vedúci: RNDr. Jaroslav Janáček, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 28.11.2019

Dátum schválenia: 05.12.2019 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Zo srdca ďakujem môjmu školiteľovi RNDr. Jaroslavovi Janáčkovi PhD. za jeho pomoc, čas, odborné konzultácie ako aj priateľský a ľudský prístup.

Abstrakt

Bežné firewally fungujú na transportnej vrstve, a teda umožňujú filtrovať komunikáciu na základe IP adries, portov, protokolu a podobne. To nám však v niektorých prípadoch nemusí stačiť. Môže byť užitočné vedieť identifikovať aplikáciu, ktorá paket odoslala, a túto informáciu využiť pri vytváraní firewallových pravidiel.

V tejto práci sme preskúmali možnosti aplikačného firewallu pre pracovné stanice s OS Linux. Navrhli sme vlastné riešenie takéhoto firewallu, ktorý sme implementovali a otestovali. Výsledky sme porovnali s podobným riešením.

Kľúčové slová: aplikačný firewall, Linux, bezpečnosť počítačových sietí

Abstract

Conventional firewalls operate at transport layer therefore they are able to filter network communication based on IP addresses, ports, protocol etc. In some cases it is not enough. It could be useful to be able to identify application sending packets and use this information to create firewall rules.

In this thesis possibilities of application firewall for Linux workstation were explored. Our own firewall was designed, implemented and tested. Results were compared with similar solutions.

Keywords: application firewall, Linux, network security

Obsah

Zoznam obrázkov

Úvod

Počiatky internetu siahajú do šesťdesiatych rokov minulého storočia, kedy vzniklo prvé prepojenie dvoch počítačov, jeden sa nachádzal na univerzite Stanford a druhý na univerzite UCLA. Tak vznikla sieť známa pod menom ARPAnet, do ktorej sa postupne pripájali ďalšie počítače z vedeckej sféry. V sedemdesiatych rokoch vznikol operačný systém UNIX, ktorý podporoval prácu v sieti, čo umožnilo väčšiemu množstvo počítačov sa pripojiť. Neskôr vznikli aj ďalšie siete, ktoré v osemdesiatych rokoch boli pospájané využitím TCP/IP protokolu. No v tomto období, kedy počítačová sieť spájala stále prevažne len vedeckú komunitu, začali vznikať aj prvé malvéry. Pravdepodobne prvým, ktorý sa šíril internetom, je Morrisov červ, ktorý útočil špeciálne na jeden unixový systém.

S rozširovaním internetu vďaka TCP/IP protokolu, ktorý umožňoval prepájať rôzne siete, vznikla aj prirodzená potreba vytvoriť bariéru, ktorá bude oddeľovať verejný internet od malých lokálnych sietí. V roky 1989 Jeff Mogul navrhol prvý firewall na filtrovanie paketov.

V deväťdesiatych rokoch začali vznikať stavové firewally. Postupne pribudlo grafické rozhranie a ďalšie bezpečnostné prvky, napríklad VPN či integrované antivírusové programy. S rozšírením a zrýchľovaním internetu sa začala viac vynárať otázka výkonnosti daných firewallov, čo ovplyvnilo smer, akým sa firewally uberali ďalej.

V súčasnosti sa firewally posunuli z transportnej vrstvy na aplikačnú, pričom sa snažia preskúmať paket čo najviac do hĺbky. Umožňujú detegovať neoprávnené vniknutie do siete, podporujú VPN a ďalšie funkcie, ktoré vo veľkom využívajú firmy na zabezpečenie svojej siete.

Okrem ochrany siete, firewall je dôležitým ochranným prvkom koncového zariadenia. Výhodou je, že môžeme robiť nastavenia týkajúce sa konkrétneho zariadenia. Taktiež sú zvyčajne ľahšie na konfiguráciu. Ďalšou výhodou je, že takýto firewall nás ochráni aj pred zariadeniami pripojenými do tej istej siete, čo je obzvlášť dôležité pre zariadenia pripájajúce sa do rôznych sietí, ako napríklad mobily či notebooky. Správne nastavený firewall výrazne znižuje šancu napadnutia malvérom, či ukradnutia dát.

Kapitola 1

Firewall v Linuxe

Každému zariadeniu pripojenému na internet hrozí potenciálne nebezpečenstvo. Jeden zo základných prvkov zabezpečujúcich ochranu je firewall. Chráni nás pred neautorizovaným vzdialeným pripojením či blokuje nechcený obsah. V tejto kapitole si predstavíme vstavaný firewall v operačnom systéme Linux a spravíme prehľad existujúcih riešení.

1.1 Firewall

Firewall je sieťové zariadenie, ktoré monitoruje a kontroluje tok dát medzi dvoma sieťami, zvyčajne medzi vnútornou a vonkajšou sieťou. Môže byť riešený hardvérovo alebo softvérovo. Na koncovom počítači firewall filtruje komunikáciu do počítača a z počítača von. Na routeri kontroluje komunikáciu medzi sieťami a zabezpečuje preklad sieťových adries (NAT).

Pakety sú kontrolované na základe pravidiel, ktoré sa skladajú z podmienky a akcie. Postupne sa prechádza všetkými pravidlami, a ak paket spĺňa danú podmienku, vykoná sa akcia. Akcie môžu byť zahodenie paketu a prípadne aj vygenerovanie správy o zahodení alebo jeho akceptovanie.

Firewally delíme na bezstavové a stavové. Bezstavové posudzujú každý paket samostatne. V podmienkach sa môžeme pýtať na informácie v hlavičkách sieťovej a transportnej vrstvy, napríklad východzia a zdrojová adresa, východzí a zdrojoví port či použitý protokol. Pri použití štandardných portov vieme takýmto spôsobom sledovať komunikáciu bežných aplikácií a služieb. Stavové firewally si navyše vedia pamätať aj informácie o spojeniach, napríklad pri TCP protokole, či ide o snahu vytvoriť nové spojenie alebo paket je prenášaný v rámci už vytvoreného spojenia. To dovoľuje komplexnejšie zabezpečenie, avšak nevýhodou je, že potrebuje viac pamäte aj výkonu.

Okrem toho môžeme firewally deliť podľa toho, na ktorej vrstve TCP/IP sa nachádzajú. Firewally, ktoré umožňujú filtráciu paketov spôsobmi opísanými vyššie, fungujú

na sieťovej vrstve. Na aplikačnej vrstve sú firewally, ktoré kontrolujú komunikáciu medzi aplikáciami a nižšími vrstvami. Tento typ firewallov umožňuje aplikovať pravidlá na jednotlivé procesy a aplikácie, teda nie na základe portov. Zvyčajne sa nepoužívajú samostatne, ale v kombinácií so sieťovými firewallmi.

1.2 Architektúra firewallu v Linuxe

V tejto časti si predstavíme firewally, ktoré sa nachádzajú v Linuxovom jadre.

1.2.1 Ipchains

Ipchains[15] je softvér na filtrovanie paketov v Linuxovom jadre od verzie 2.2. Všetky pakety sú rozdelené do troch tried – prichádzajúce, preposielané a odosielané. Pravidlá sú rozdelené do reťazí – Input, Forward a Output. Okrem toho, používateľ si môže zadefinovať ďalšie reťaze, čím sa môže proces filtrovania rozvetvovať. Okrem zavolania inej reťaze sa s paketom môžu vykonať nasledujúce akcie[20]:

- ACCEPT – ukončí spracovávanie paketu akceptovaním a prenechá ho príslušnému handleru.
- DENY – zahodí paket
- REJECT – zahodí paket a pošle odosielateľovi chybovú správu o nedostupnosti destinácie
- REDIRECT – používa sa na presmerovanie prichádzajúcich paketov na iný port. To sa využíva napríklad pri proxy.

Keď počítač prijme paket, ktorý je preň určený, najskôr sa skontrolujú kontrolné súčty a potom je spracovaný na základe pravidiel v reťazi input. V prípade paketu, ktorý je preposielaný do inej siete, paket postupne prejde všetkými reťazami. Najprv prejde cez input reťaz. Potom sa skontroluje, či bol paket upravovaný pomocou NAT. Ak áno, preskočí sa forward reťaz a paket hneď prejde na output reťaz. Ak paket nebol upravovaný pomocou NAT, je spracovaný postupne forward reťazou a následne output reťazou. Paket odosielaný z daného počítača je spracovaný iba pomocou output reťaze.

Nevýhodou ipchains je to, že paket sa rozdeľuje do triedy iba na základe adresy, a podľa toho je ďalej spracovávaný, čo komplikuje vytváranie pravidiel. Okrem toho podporuje iba bezstavové filtrovanie.

1.2.2 Netfilter/iptables

V Linuxovom jadre od verzie 2.4 nájdeme subsystém Netfilter[16], ktorý slúži na manipuláciu s paketmi, ako napríklad filtrovanie paketov, prekladanie sieťových adries a portov. Obsahuje iptables, ktoré nahradili predtým používané ipchains. Nájdeme tu aj ďalšie podobné moduly pre iné protokoly ako napríklad ip6tables pre IPv6. Jednotlivé moduly sú dynamicky pridávané do Linuxového jadra hookingom. My sa bližšie pozrieme na hooky pre IPv4 protokol.

Hooky a reťaze

Ako prvé po prijatí je paket spracovaný pomocou NF_IP_PRE_ROUTING. V tomto hooku je paket spracovaný ešte pred samotným smerovaním. To môže byť užitočné napríklad na NAT, detekciu denial of service útoku či účtovanie za prichádzajúce pakety. Ďalší hook je NF_IP_LOCAL_IN, ktorý filtruje prijaté pakety, ktoré zostávajú na danom počítači. NF_IP_FORWARD spracováva pakety, ktoré nie sú určené pre daný počítač, ale budú preposlané ďalej. V tomto prípade sú spracovávané aj pakety, ktoré boli modifikované pomocou NAT. NF_IP_LOCAL_OUT slúži na spracovávanie a filtrovanie paketov odchádzajúcich z daného počítača. Posledným v poradí je NF_IP_POST_ROUTING hook, ktorý sa zavolá na odchádzajúce pakety. Môže byť použitý pri účtovaní za odchádzajúce pakety[20].

Jednotlivé hooky spúšťajú základné reťaze – PREROUTING, INPUT, FORWARD, OUTPUT a POSTROUTING[3]. Okrem nich si vie používateľ zadefinovať vlastné reťaze, vďaka čomu vzniká vetvenie.

Tabuľky

Ako napovedá samotný názov, pravidlá sú organizované do tabuliek, ktoré hovoria, akou úlohou sa dané pravidlá zaoberajú. Nájdete tam nasledujúce tabuľky:

- filter – slúži na filtrovanie paketov. O každom pakete sa rozhodne, či sa pustí ďalej alebo nie. Je to najčastejšie používaná tabuľka.
- nat – sa používa na prekladanie sieťových adries. Podľa príslušných pravidiel upraví zdrojovú a cieľovú adresu.
- mangle – slúži na úpravu IP hlavičiek. Okrem toho, umožňuje označiť pakety v rámci kernelu, čo môže byť využité pri ďalšom spracovávaní paketu či už firewallom alebo inými procesmi.
- raw – narozdiel od ipchains, iptables je stavový firewall. Táto tabuľka slúži na to, aby niektoré pakety mohli byť vynechané z procesu sledovania spojenia.

- security – slúži na označovanie paketov pre SELinux.

Tabuľky nám organizujú pravidlá podľa toho, akú majú úlohu. Okrem toho, pravidlá sú v reťaziach, ktoré vedia, kedy sa majú zavolať. Tieto dve veci potrebujeme medzi sebou prepojiť. Je dôležité si uvedomiť, že jednotlivé tabuľky neobsahujú všetky reťaze. Taktiež vzniká otázka, v akom poradí sa budú volať jednotlivé reťaze patriace do rôznych tabuliek, ale do toho istého hooku. Ako prvé sa spúšťajú reťaze PREROUTING. Tie nájdeme v tabuľkách raw, mangle a nat. V tomto poradí bude paket prechádzať pravidlá v daných tabuľkách. Reťaze INPUT sa budú spúšťať postupne pre tabuľky mangle, filter, security a nat. Reťaz FORWARD nájdeme v tabuľkách mangle, filter a security. Reťaz OUTPUT sa nachádza vo všetkých tabuľkách, pričom sa budú spúštať v poradí raw, mangle, nat, filter a security. Nakoniec sa spustia reťaze POSTROUTING v tabuľkách mangle a nat. Niektoré reťaze z tabuliek môžu byť za istých okolností vynechané. Napríklad, pri použití NAT sa vyhodnocuje iba prvý paket v rámci spojenia. Všetky ostatné pakety v rámci spojenia sa už automaticky prekladajú bez ďalšieho vyhodnocovania pravidiel v nat tabuľke firewallu[3].

To, ktoré hooky sa zavolajú, záleží od toho, či paket prichádza, odchádza alebo len prechádza. Pre paket prichádzajúci na daný počítač sa použijú reťaze PREROUTING a INPUT. Prechádzajúci paket bude vyhodnotený reťazami PREROUTING, FORWARD a POSTROUTING. Na rozdiel od ipchains, prechádzajúci paket nie je vyhodnocovaný hookmi INPUT ani OUTPUT. Na paket, ktorý vznikol na danom počítači a bude posielaný von, sa použijú pravidlá z OUTPUT a POSTROUTING reťazí.

Na zhrnutie si uveďme pár príkladov. Prišiel nám paket, no skôr, ako sa dostane na soket a bude spracovaný príslušným procesom, bude vyhodnocovaný firewallom. Ako prvé prejde reťazou pravidiel PREROUTING v tabuľke raw. Potom sa vyhodnotí jeho stav v rámci spojenia. Následne sa upraví hlavička podľa reťazí PREROUTING v tabuľkách mangle a potom nat. V tejto chvíli môže byť vykonané samotné smerovanie paketu. Teraz sa zavolajú reťaze INPUT z tabuliek mangle, filter, security a nat v tomto poradí. Až keď prejde všetkými reťazami, paket je uvoľnený na ďalšie spracovávanie iným procesom. Keď máme paket, ktorý chceme odoslať z nášho počítača, najprv sa urobí smerovanie. Vyhodnotí sa reťaz OUTPUT z tabuľky raw. Potom sa pozrie na stav spojenia. Následne sa zavolajú ostatné reťaze OUTPUT a potom POSTROUTING.

Pravidlá

Reťaze obsahujú zoznam pravidiel, ktoré sa postupne prechádzajú a vyhodnocujú. Každé pravidlo má dve časti – porovnávaciu a cieľovú. V porovnávacej časti je nejaké kritérium, ktorým sa vyhodnocuje paket. Následne, ak paket spĺňa kritérium, vykoná sa akcia, ktorá je uvedená v cieľovej časti.

V prvej časti pravidla sa môžeme pýtať na rôzne informácie, ako napríklad protokol,

zdrojová a cieľová IP adresa, port a ďalšie informácie dostupné v TCP resp. UDP hlavičke. Okrem toho modul podporuje rôzne rozšírenia[17], ktoré nám umožňujú budovať komplexnejší súbor pravidiel. Medzi rozšíreniami nájdeme možnosť, ako sa pýtať na viac adries alebo portov naraz, zistiť stav v rámci spojenia a kontrolovať MAC adresu. Navyše sa môžeme pozerať na vlastníka paketu a jeho skupinu, ktorý ho vytvoril. Toto má, samozrejme, zmysel len pre odchádzajúce pakety.

V minulosti umožňovali iptables pýtať sa aj na proces ID. Táto funkcia bola zrušená v kerneli verzie 2.6.14, pretože nefungovala správne[8].

V cieľovej časti sú uvedené akcie, ktoré sa vykonajú, ak daný paket spĺňa kritérium uvedené v porovnávacej časti. Tieto akcie môžu byť terminujúce a neterminujúce. Terminujúce akcie ukončia vyhodnocovanie v rámci danej reťaze a na základe návratovej hodnoty linuxové jadro paket ponechá alebo zahodí. Ak sa paket ponechal, bude ďalej spracovávaný ďalším hookom v poradí. Po vykonaní neterminujúcej akcie sa pokračuje ďalej vo vyhodnocovaní paketu danou reťazou. Každá reťaz nakoniec skončí nejakou terminujúcou akciou, avšak predtým môže byť vykonaných viacero neterminujúcich akcií[3]. Najbežnejšie akcie sú ACCEPT a DROP, ktoré príjmu resp. zahodia daný paket. Ďalšou akciou je REJECT, ktorá okrem toho, že paket zahodí, pošle aj používateľom zvolenú chybovú správu. Môžeme skákať na iné reťaze alebo sa z nich vrátiť späť na pôvodnú reťaz. Taktiež vieme robiť akcie, ktoré upravujú IP hlavičky v tabuľkách nat a mangle alebo zalogovať informácie o prijatom pakete. Navyše môžeme paket zaradiť do fronty, odkiaľ je dostupný programom v užívateľskom priestore, ktoré rozhodnú o jeho osude[20].

Spojenie

Okrem vyhodnocovania paketov na základe informácií v hlavičkách, iptables nám umožňuje sledovať komunikáciu v rámci spojenia. To znamená, že nemusíme spracovávať jednotlivé pakety samostatne, ale môžeme sa na ne pozerať aj v kontexte ďalšej komunikácie, ktorá prebehla v rámci daného spojenia.V prípade UDP nám to dovoľuje rozlišovanie medzi odpoveďami a inými správami. Taktiež nám to umožňuje priradiť prijatú chybovú správu ku konkrétnej komunikácií. V prípade TCP paketu, ktorý v sebe už obsahuje informácie o spojení, môžeme urýchliť filtrovanie paketu tým, že preskočíme ďalšie spracovávanie a povolíme paket, ak pochádza z povoleného spojenia[17].

Každé spojenie je jednoznačne určené protokolom, zdrojovou a cieľovou adresou, zdrojovým a cieľovým portom. Tieto údaje sa ukladajú do hašovacej tabuľky, kde potom nájdeme aj ďalšie informácie o spojení. Jednotlivé spojenia môžu byť v nasledujúcich stavoch:

- NEW – spojenie vzniká týmto paketom, a zatiaľ ešte neprišla žiadna odpoveď. V prípade TCP protokolu to zodpovedá SYN paketu.

- ESTABLISHED – ide o prebiehajúce spojenia. V TCP sú to správy s ACK príznakom.
- RELATED – pakety, ktoré nejakým spôsobom súvisia s existujúcim spojením, ale nie sú priamo jeho súčasťou. Medzi takéto pakety patria napríklad chybové správy ICMP protokolu.
- INVALID – pakety, ktoré nepatria k žiadnemu známemu spojeniu a ani nezačínajú korektné spojenie. Príkladom môže byť správa echo-reply, ktorá prišla bez predchádzajúcej echo-request správy.

Nevýhodou je, že si potrebujeme pamätať všetky prebiehajúce spojenia, čo môže zaberať veľa miesta. Zvyšuje sa aj časová náročnosť spojená so spravovaním tabuľky spojení, obzvlášť pre prípady, keď sa v rámci spojenia odošle len zopár správ. Taktiež treba nastaviť časové limity, kedy sa majú staré spojenia zahodiť, pretože v prípade UDP protokolu nevieme identifikovať koniec komunikácie.

1.2.3 Netfilter/nftables

V rámci projektu Netfilter vznikol nový subsystém nftables[18], ktorý bol zaradený do Linuxového jadra od verzie 3.13. Čiastočne nahrádza moduly z Netfilter, ktoré vo veľkej miere aj využíva.

Nftables používajú virtuálny počítač, vďaka čomu umožňujú zadefinovať a používať premenné a spúšťať programy na vyhodnocovanie paketov. Podobne ako iptables, aj tu nájdeme hooky prerouting, input, forward, output a postrouting. Rozdiel je, že sa tu nenachádzajú preddefinované tabuľky. Používateľ má možnosť si sám zadefinovať tabuľky, ktoré potrebuje. Do tabuliek môžeme pridávať reťaze pravidiel. Každá reťaz musí mať uvedené, ktorý hook ju má spustiť a prioritu, ktorá určuje poradie v rámci hooku. Taktiež musí mať uvedený typ – filter, route alebo nat. Reťaze obsahujú pravidlá, ktoré sa skladajú z výrazu a príkazu. Príkazy v nftables môžu byť:

- accept – skončí vyhodnocovanie paketu jeho akceptovaním
- continue – prejde na ďalšie pravidlo
- drop – potichu zahodí paket
- reject – zahodí paket a pošle chybovú správu
- goto – zavolá inú reťaz, ktorá bude spracovávať paket, no už sa nevráti k volajúcej reťazi
- jump – zavolá inú reťaz a po skončení sa vráti na volajúcu reťaz

- return – vráti sa späť na volajúcu reťaz
- limit – hovorí, čo sa má stať s paketom, keď počet paketov spĺňajúcich daný výraz prekročí istú hranicu
- log – zapíše informácie o pakete do log súboru
- queue – odošle paket do užívateľského prostredia

Výrazy sa môžu pýtať na informácie z hlavičiek, ako protokol, adresa alebo port. Ďalej sa môžeme pýtať na metadáta, ako napríklad ktorá sieťová karta paket prijala alebo UID používateľa, ktorý daný paket vytvoril. Okrem toho sa môžeme pozerať aj na stav spojenia – new, established, related, a invalid, ktoré majú rovnaký význam ako v iptables. Pribudol nám tu stav untracked, ktorý hovorí, že pre tieto pakety nesledujeme ich spojenie. Výrazy sa môžu pýtať na viacero údajov a sú vyhodnocované lineárne zľava doprava. Narozdiel od iptables, jedno pravidlo môže obsahovať viac akcií, ktoré sa majú vykonať. Ďalšou výhodou je, že nftables umožňujú načítať pravidlá z externého súboru[17].

1.2.4 bpfilter

Napriek tomu, že nftables sa v linuxovom jadre nachádzajú od roku 2014, nie sú veľmi používané v praxi. Jeden z dôvodov je, že administrátorom sa nechce budovať nový firewall, keďže syntax nftables sa líši od iptables, a teda nie je jednoduché znovu napísať všetky pravidlá. A tak v roku 2018 bol predstavený nový firewall vychádzajúci z iptables – bpfilter[2].

Berkeley Packet Filter (BPF) [14] je virtuálny stroj v Linuxovom jadre, ktorý umožňuje vykonávanie strojového kódu. Pôvodne vznikol ako pomocný nástroj pre rýchle filtrovanie paketov v jadre, aby sa zabránilo zbytočnému kopírovaniu, napríklad pre nástroj tcpdump, ktorý sleduje TCP komunikáciu. Neskôr bol rozšírený o just-in-time kompilátor a začal sa viac využívať.

Firewall bpfilter umožňuje filtrovať pakety vykonaním BPF programu, ktorý ich preskúma a následne vyhodnotí, či majú byť akceptované alebo zahodené. Výhodou je, že môžeme zobrať pravidlá z iptables a bpfilter ich vie preložiť na BPF programy. Zatiaľ obsahuje len niektoré z podmienok a akcií, ale autori sľubujú, že pridajú všetky funkcie iptables. Nevýhodou je, že bpfilter je zatiaľ len vo vývoji, preto nie je vhodný na použitie v produkcií.

1.3 Aplikačné Firewally v Linuxe

V predchádzajúcej časti sme si predstavili firewall, ktorý je zabudovaný v Linuxovom jadre. To, čo mu zatiaľ chýba, je možnosť filtrovať komunikáciu na základe aplikácie, ktorá sa snaží komunikovať. Tu si predstavíme zopár firewallov, ktoré to umožňujú.

1.3.1 Kerio Control

Kerio Control[7] je komerčný firewall spoločnosti GFI Software. Okrem toho, že umožňuje bežné filtrovanie komunikácie na sieťovej vrstve, dovoľuje budovať pravidlá aj pre aplikácie, ktoré môžu komunikovať cez internet. Navyše umožňuje zakázať používateľovi navštevovať konkrétne stránky alebo typy stránok, prípadne nedovoliť prístup na internet v určenom čase. Nevýhodou je, že ide o platený softvér.

1.3.2 Douane

Douane[5] je firewall, ktorý podobne ako OpenSnitch filtruje komunikáciu aplikácií. Sleduje internetovú komunikáciu a vždy, keď rozpozná nové spojenie, zablokuje ho a spýta sa používateľa či chce danú komunikáciu povoliť alebo nie a na základe toho si vytvorí pravidlá. Implementuje vlastný modul a používa netfilter v Linuxovom jadre. Je dostupný pod GPL licenciou.

1.3.3 OpenSnitch

OpenSnitch[13] je linuxový firewall založený na Little Snitch firewalle pre MacOS, ktorý sa zameriava na komunikáciu aplikácií. Pre každú aplikáciu si pamätá, či smie alebo nesmie pristupovať na internet. V prípade, že pre danú aplikáciu ešte žiadne pravidlo nemá, vyskočí okno, ktoré sa spýta, či chce používateľ dovoliť danej aplikácií prístup na internet. Využíva na to NFQUEUE z iptables. Je dostupný pod GPL licenciou a nájdeme ho aj na Arch User Repository (AUR). V čase, keď sme začínali robiť túto diplomovú prácu, OpenSnitch bol vo vývoji a ešte nevyšla ani oficiálna prvá verzia. Program obsahoval veľa chýb a nefungoval tak, ako by mal, preto nebol vhodný.

Počas robenia tejto práce napredoval aj vývoj firewallu OpenSnitch. Pri vyhodnocovaní našej práce sa pozrieme na tento firewall bližšie a porovnáme ho s naším riešením.

1.3.4 SELinux

SELinux[21] je modul Linuxového jadra, ktorý implementuje povinné riadenie prístupu (MAC). Každému používateľovi, procesu, súboru, soketu apod. je priradený kontext,

ktorý obsahuje meno, rolu a typ (doménu). Súbor pravidiel, politika, hovorí, čo môže ktorý proces vykonať.

V kombinácií s iptables, ktoré podporujú SELinuxové typy v tabuľke security, je možné postavať firewall, ktorý filtruje komunikáciu na základe procesu.

Nevýhodou tohto prístupu je, že SELinux je príliš komplikovaný pre väčšinu používateľov a použitie spolu s iptables je málo zdokumentované.

Kapitola 2

Návrh riešenia

V tejto kapitole sa pozrieme, aké požiadavky by mal spĺňať náš firewall pre Linux a navrhneme, ako by mal vyzerať.

2.1 Požiadavky

Skôr, ako sa pustíme do navrhovania firewallu, je dôležité si uvedomiť, aké požiadavky by mal spĺňať. Taktiež musíme brať ohľad na to, kto budú používatelia nášho produktu.

Hlavným cieľom tohto firewallu je zvýšenie bezpečnosti pracovných staníc s OS Linux tým, že umožníme používateľovi filtrovať komunikáciu podľa aplikácie, ktorá sa snaží komunikovať.

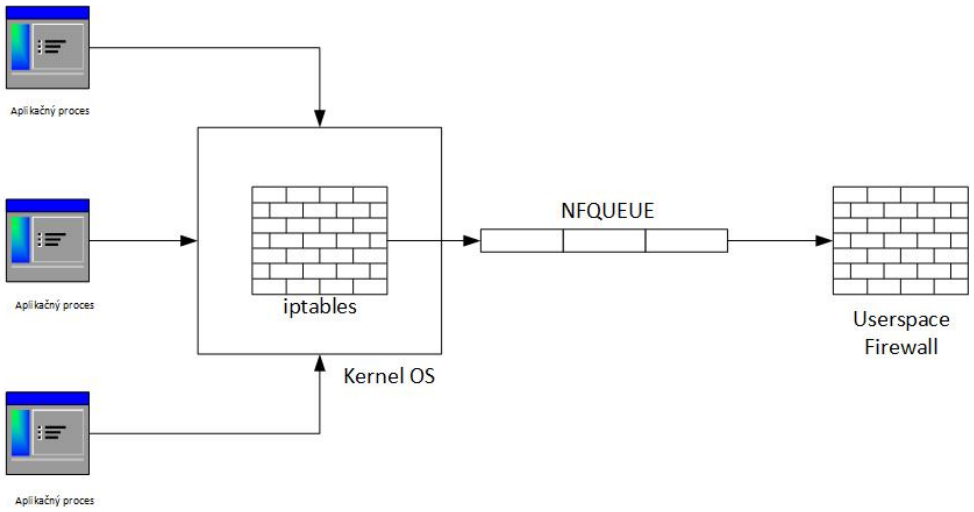
Náš softvér je určený pre zariadenia s operačným systémom Linux. Firewall umožní používateľovi sledovať, aké pakety ich zariadenie odosiela a prijíma. Zobrazí informácie o pakete, ako IP adresa, port a protokol. Pokiaľ je to možné, zistí aj to, ktorá aplikácia daný paket odoslala / príjme. Používateľ má možnosť sa pre každý paket rozhodnúť, či ho ponechá alebo sa zahodí. Zároveň má možnosť povedať, aby sa rovnaká akcia vykonala aj s ďalšími paketmi, ktoré patria tomu istému spojeniu alebo aplikácií. Tieto rozhodnutia sa ukladajú a je možné ich následne zmeniť.

Náš firewall je primárne určený pre bežných používateľov, ktorí nemajú veľké skúsenosti s administrovaním Linuxu, no záleží im na bezpečnosti ich zariadenia. Preto je potrebné, aby sa firewall jednoducho ovládal pomocou grafických prvkov.

Okrem funkčných požiadaviek kladieme dôraz aj na bezpečnosť nášho softvéru, aby sme minimalizovali možnosti zneužitia útočníkom. Preto máme požiadavku, aby okrem kritických častí náš firewall nebežal s privilegovanými právami.

2.2 Architektúra

Na základe vyššie uvedených požiadaviek navrhneme náš firewall.



Obr. 2.1: Na obrázku je znázornená vysoko-úrovňová architektúra. Aplikácie posielajú pakety, ktoré následne spracováva jadro OS. V ňom sa okrem iného nachádza iptables, ktoré cez NFQUEUE odošlú paket do užívateľského prostredia, kde ho spracuje náš firewall. Ten po spracovaní oznámi verdikt jadru, ktoré paket ponechá alebo ho zahodí.

Z pohľadu architektonických vzorov sa jedná o monolitickú aplikáciu riadenú udalosťami.

2.2.1 Vysoko-úrovňová architektúra

Základnú časť firewallu tvorí démon bežiaci na pozadí. Tento démon beží ako služba v rámci systemd, čo je štandardné riešenie. Náš firewall postavíme na linuxovom firewalle iptables, ktoré využijeme na zachytávanie paketov a ich odoslanie z linuxového jadra do užívateľského prostredia. Tam bude čakať náš démon, ktorý paket spracuje a pošle späť do jadra akciu, čo sa má s daným paketom stať.

Architektúru si bližšie priblížime nasledujúcimi scenármi. Máme na počítači spustenú aplikáciu, ktorá chce odoslať paket. Po opustení aplikácie paket prejde do jadra Linuxu, ktorý ho začne spracovávať. Okrem iného bude spracovaný aj iptables. Tie pomocou NFQUEUE pošlú paket do užívateľského prostredia, kde čaká náš firewall. Keď mu príde paket, spracuje ho a odošle verdikt späť do jadra. Pokiaľ náš firewall rozhodol, že paket má byť zahodený, jadro ho zahodí. Ak náš firewall rozhodol, že paket má byť ponechaný, jadro paket spracuje ďalej až ho nakoniec odošle. To môžeme vidieť znázornené ar na obrázku 2.1. V prípade, že dostaneme paket z internetu, znova bude spracovávaný jadrom a iptables, ktoré odošlú paket do užívateľského rozhrania, kde ho spracuje náš firewall. Odošle verdikt späť do jadra, ktoré paket buď zahodí, ak dostal taký verdikt, alebo si ho ponechá a po spracovaní ho odovzdá aplikácií, ktorá na daný paket čaká.

2.2.2 Komponenty

Náš firewall je tvorený nasledujúcimi komponentmi:

- Daemon – sleduje NFQUEUE. Keď príde paket, povie fronte, aby vykonala danú callback funkciu.
- Packet Handler – obsahuje callback funkciu. Prečíta z hlavičiek informácie o pakete a spýta sa na proces, ktorý paket odoslal / príjme. Následne sa opýta, či pre daný proces a spojenie už je uložená akcia, čo sa má s daným paketom stať. Ak áno, vráti verdikt. Ak nie, zavolá vytvorenie okienka, ktoré sa opýta používateľa, čo sa má s paketom stať.
- Process finder – zisťuje, ktorá aplikácia daný paket odoslala / príjme.
- Storage – má na starosť pamätanie si odpovedí od používateľa. Zároveň oznamuje uložené akcie Packet Handler komponentu.
- Window – vytvára okienko, ktoré sa pýta používateľa, aká akcia sa má s daným paketom vykonať a vracia odpoveď.
- NFQUEUE – je modul na rozhraní jadra a užívateľského rozhrania. Poskytuje API na spracovávanie paketov v užívateľských programoch.

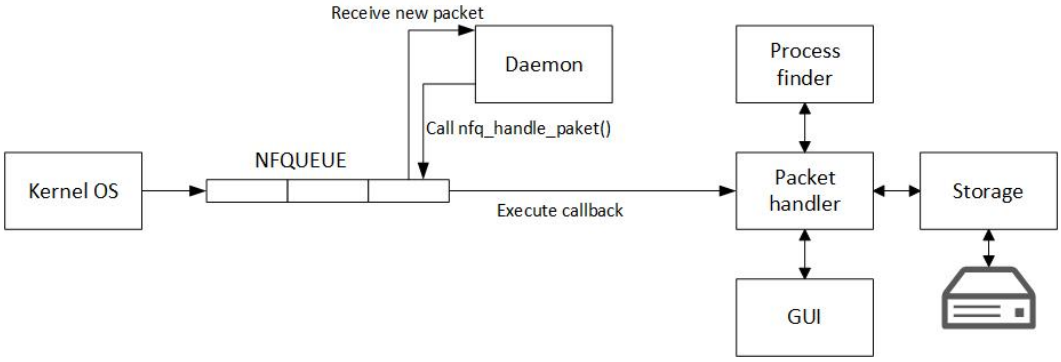
Prvý návrh

V prvom návrhu náš firewall sa skladal z jedného procesu, ktorý sledoval, či neprišiel nejaký paket do fronty. Ak prišiel, začal ho spracovávať. Pohľadal, či už má uloženú informáciu pre daný proces a spojenie. Pokiaľ nie, spýtal sa používateľa, čo má spraviť s daným paketom a následne vykonal danú akciu. Po spracovaní paketu náš firewall znova skontroloval frontu, či neprišiel ďalší paket. Spracovávanie paketu môžeme vidieť na obrázku 2.2.

Verzia s jedným procesom však bola nevyhovujúca, pretože pokiaľ sa čakalo na vstup od používateľa, nemohli sa spracovávať ďalšie pakety vo fronte, pričom možno tie ďalšie už majú záznam a len by sa na nich vykonala akcia bez potreby čakať na rozhodnutie od používateľa. V prípade dlhého čakania by sa navyše mohla fronta zaplniť a ďalšie pakety by systém začal zahadzovať. Preto bolo potrebné zmeniť pôvodný návrh tak, aby umožňoval spracovávať ďalšie pakety z fronty, kým sa čaká na vstup od používateľa.

Druhý návrh

Problémom v prvom návrhu bolo, že pakety boli spracovávané sériovo len jedným procesom. Preto pri druhom návrhu hľadáme možnosti, ako pakety, ktoré vieme rýchlo



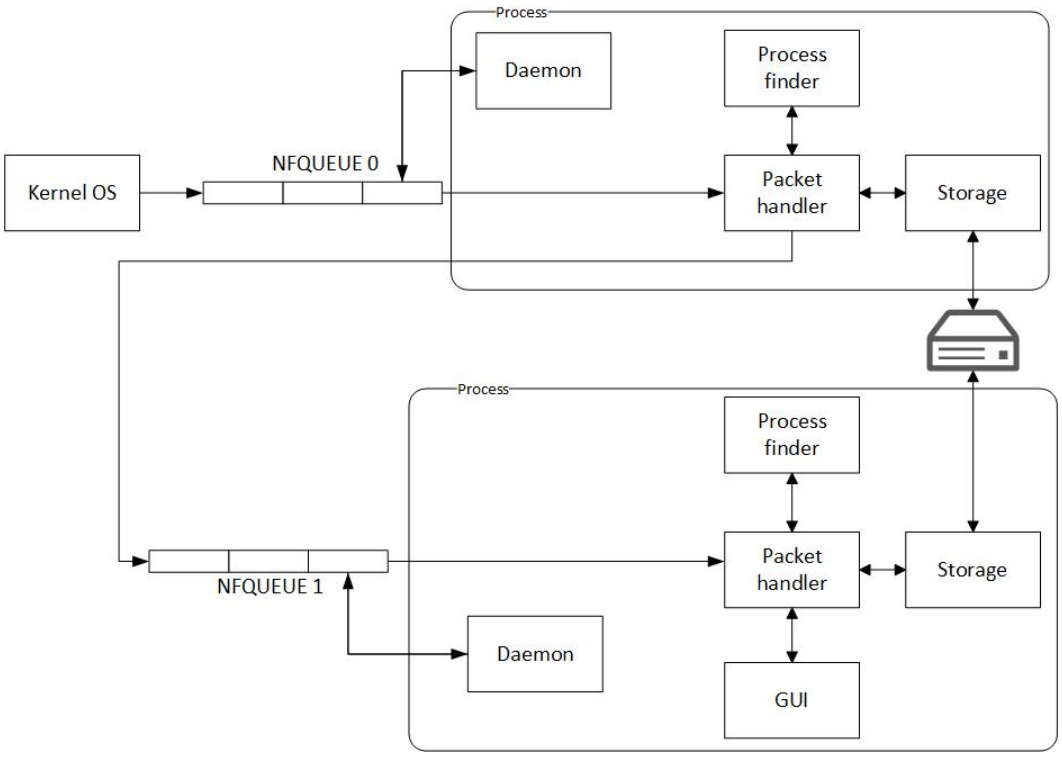
Obr. 2.2: Pôvodný návrh komponentov s jedným procesom. Tento návrh nám nevyhovoval, pretože pokiaľ sme čakali na vstup od používateľa, nemohli byť spracovávané ďalšie pakety z fronty.

spracovať, neblokovať čakaním na pakety, pri ktorých musí používateľ zadať vstup. Treba si uvedomiť, že to, či pakety budú čakať na používateľa, sa dozvieme až počas spracovávania paketu, teda nevieme to vopred.

Prvý nápad bol, že budeme každý paket spracovávať ako samostatný proces. Keď Daemon príjme paket, vytvorí nový proces, ktorý zavolá callback funkciu. Tento nový proces spracuje celý paket – zistí informácie z hlavičiek; nájde aplikáciu; pozrie, či je už uložený záznam a ak áno, použije ho, inak vytvorí okienko a spýta sa používateľa, vráti odpoveď a proces skončí. Nevýhodou tohto nápadu je, že môže vytvoriť ľubovoľne veľa procesov, čo môže vyčerpať zdroje systému.

Ďalší nápad bol, že vytvoríme druhú NFQUEUE. Pakety začneme spracovávať pôvodným procesom. Keď zistíme, že je potrebné vytvoriť okienko a spýtať sa používateľa, čo sa má s daným paketom stať, namiesto vytvorenia okienka, vložíme paket do druhej NFQUEUE, čím z pohľadu prvej fronty ukončíme spracovávanie paketu (vloženie paketu do NFQUEUE je jeden z cieľov iptables rovnako ako akceptovanie a zahodenie paketu). To nám dovolí začať spracovávať ďalší paket z prvej NFQUEUE. Máme druhý proces s rovnakým démonom, ktorý počúva na druhej NFQUEUE. Keď tam príde paket, zavolá sa callback funkcia a spracuje sa paket. Znova sa skontroluje, či medzitým nepribudol záznam pre daný paket, a ak nie, vyskočí okienko, ktoré sa opýta používateľa, čo sa má s paketom stať. Celý čas bežia dva procesy. Ako to prebieha, môžeme vidieť na obrázku 2.3.

Tretí nápad používa len jednu NFQUEUE. Pakety opäť začneme spracovávať rovnako, ako predtým. No keď prídeme na to, že by sme potrebovali zistiť akciu od používateľa, vytvoríme sa kópiu paketu a pôvodný paket, ktorý bol NFQUEUE zahodíme. Tým sa nám uvoľní pôvodný proces, ktorý môže začať spracovávať ďalší paket. Kópiu prvého paketu odovzdáme ďalšiemu procesu, ktorý vytvorí okienko a zistí od používateľa, čo sa má s daným paketom stať. Pokiaľ používateľ rozhodne, že sa má zahodiť,



Obr. 2.3: Upravený návrh komponentov pozostávajúci z dvoch procesov. Prvý proces spracováva pakety, no ak by mal vytvoriť okienko pre používateľa, neurobí tak a namiesto toho zaradí paket do fronty, ktorú spracováva druhý proces. Ten pracuje tak, ako v pôvodnom jednoprocesovom návrhu.

druhý proces nemusí spraviť nič (pretože paket už je zahodený). Ak sa ho rozhodne akceptovať, proces odošle kópiu paketu. Paket musíme označiť, aby sa nestalo, že firewall odchytí túto kópiu paketu a začne ju znova spracovávať.

Z vyššie uvedených nápadov sme zvolili druhý uvedený, teda použitie druhej NFQUEUE. Tento nápad sa nám zdal byť najlepší. Oproti prvému je lepší v tom, že v našom riešení vždy budeme mať dva procesy narozdiel od prvého nápadu, kde ich môže vzniknúť ľubovoľne veľa. V treťom nápade by boli použité tiež len dva procesy. No uprednostnili sme druhý nápad, pretože je výrazne ľahší na implementáciu.

Kapitola 3

Implementácia

V tejto kapitole sa bližšie pozrieme na implementáciu nášho riešenia, problémy, na ktoré sme narazili a ako sme ich vyriešili.

Ako prvé sme museli zvoliť vhodný programovací jazyk. Keďže ide o programovanie nízkej úrovne, zvolili sme jazyk C. Na druhú stranu, potrebovali sme spraviť aj grafické prvky, na čo jazyk C nie je ideálny. Síce už existujú knižnice, ktoré nám umožňujú robiť grafiké prostredie, avšak napriek tomu nie sú jednoduché na použitie. Preto sme sa rozhodli urobiť GUI v jazyku Python použitím knižnice *Tkinter*. Na zachytávanie a spracovávanie paketov používame knižnicu *libnetfilter_queue* v jazyku C.

Náš firewall sme rozdelili do viacerých modulov. Program začína v *main.c*, kde sa pridajú do iptables tabuľky potrebné pravidlá, pokiaľ sa tam už nenachádzajú. Následne sa vytvorí démon v *daemon.c*. Vytvoria sa dve kópie, ktoré sa pripoja k svojej NFQUEUE a čakajú na paket. Funkcie na spracovanie paketu sa nachádzajú v *paket_handler.c*. Modul *find_process.c* zisťuje, ktorému procesu patrí paket a poskytuje potrebné informácie o danom procese. Na vytvorenie okienka, ktoré sa opýta používateľa, čo chce s paketom urobiť, slúži *window.c*, ktorý cez API volá *window.py*.Uloženie a následné získanie zapamätaných akcií s paketom pre dané spojenie zabezpečuje *storage.c*. Zbavenie sa a znovuzískanie práv je implementované v *privilages.c*. Použité konštanty sa nachádzajú v *constants.c*. Samostatnú časť tvorí administrátorské centrum, v ktorom môžeme meniť uložené pravidlá. V priečinku *admin* sa nachádza rovnomenný pythonový modul, ktorý implementuje toto centrum. Na zjednodušenie kompilácie a inštalácie je vytvorený *Makefile* a *install.sh* skript.

3.1 Hľadanie procesu

Jeden z problémov, ktoré sme museli riešiť, bolo, ako nájsť proces, ktorý paket odoslal. Kedysi táto informácia bola dostupná priamo v dátach z iptables, ale táto funkcia bola odstránená. Preto sme museli hľadať, ako z dostupných dát, ktoré o pakete máme,

môžeme zistiť, z akej aplikácie pochádza.

Najprv sa pozrieme, aké informácie o pakete vieme zistiť, ktoré by sme mohli použiť na nájdenie procesu, ktorý ho odoslal. Môžeme sa pozrieť na jeho IP hlavičky, kde nájdeme okrem iného zdrojovú a cieľovú IP adresu a protokol. Navyše vieme zistiť zdrojový a cieľový port. Máme aj ďalšie informácie o pakete, podľa toho, či bol paket vytvorený na danom zariadení alebo prišiel zvonka.

Jeden zo spôsobov, ako nájsť proces využitím dostupných dát, by bolo použiť rôzne terminálové príkazy ako `netstat -p` a `lsof`. No tieto príkazy nemajú vytvorené knižnice pre jazyk C, ktoré by sme mohli použiť a volať `system()` na vykonanie daných príkazov je príliš drahé. Okrem toho, tieto nástroje na pozadí čítajú dáta z priečinkov */proc/net* a */proc/<PID>/fd*, preto sme sa rozhodli, že my budeme čítať tieto dáta priamo.

Najprv použijeme */proc/net* na nájdenie inode soketu, ktorý počúva na danom porte. V tomto priečinku sa nachádzajú informácie a štatistiky týkajúce sa sieti. Pre nás budú dôležité súbory */proc/net/tcp*, */proc/net/udp*, */proc/net/raw*, */proc/net/udplite* a */proc/net/icmp*. Dané súbory obsahujú informácie o práve aktívnych spojeniach využívajúcich daný protokol. Pozrieme sa na */proc/net/tcp*, keďže ostatné súbory majú rovnakú štruktúru.

Prvý riadok obsahuje hlavičku. Nasledujú informácie o spojeniach, každé spojenie na samostatnom riadku. Záznam o spojení pozostáva z čísla, lokálna IP adresa a port v hexadecimálnej sústave nasledované vzdialenou IP adresou a portom. Ďalej môžeme vidieť stav spojenia, prenosovú a prijímaciu frontu. Nasledujúce číslo hovorí o tom, ktorý časovač je aktívny a počet jiffy, kým časovač vyprší. Záznam pokračuje počtom nezotavených RTO (retransmission time-out). Za tým sa nachádza ID používateľa.Nasleduje počet pokusov pri nulovom okne, na ktoré neprišla odpoveď. Ďalšie číslo je inode soketu a počet referencií. Nasledujúce čísla sa už líšia podľa konkrétneho protokolu[9].

V našom firewalle prechádzame po riadku jednotlivé záznamy a hľadáme záznam, v ktorom sa zhodujú IP adresy a porty s hodnotami v IP hlavičke zachyteného paketu. Vyhovuje nám aj záznam, v ktorom sedí lokálna IP adresa a port a vzdialená IP adresa a port v súbore sú 0, čo znamená, že daný soket počuva a čaká na ľubovoľné spojenie. Keď nájdeme zhodujúci sa záznam, získame z neho inode soketu.

Keď poznáme soket, potrebujeme nájsť proces, ktorý ho vytvoril. Na to budeme prehľadávať */proc/<PID>/fd*. V súborovom systéme */proc* nájdeme dátové štruktúry z jadra Linuxu, */proc/<PID>* sú podpriečinky pre každý proces a */proc/<PID>/fd* obsahuje súborové deskriptory, ktoré má daný proces otvorené. My prehľadávame tento súborový systém a hľadáme proces, ktorý má otvorený daný soket. Identifikačné číslo procesu zodpovedá názvu podpriečinka *<PID>* z cesty. Keď poznáme proces ID, môžeme zistiť o procese ľubovoľné informácie, aké budeme potrebovať.

Používateľ by radšej videl pri rozhodovaní meno aplikácie ako číslo procesu. Navyše

odpovede od používateľa si chceme ukladať. Proces ID nie je vhodné na uloženie, pretože po skončení procesu môže byť toto PID priradené inému procesu, ktorý zodpovedá inej aplikácií. Podobne tak po reštartovaní zariadenia aplikácie dostanú iné proces ID. Ako identifikáciu danej aplikácie budeme používať príkaz, ktorý ju spustil (bez argumentov). Túto informáciu získame zo súboru */proc/<PID>/cmdline*. Toto zodpovedá tomu, čo daný program dostane ako nultý argument.

Viac sme rozmýšľali nad menom, ktoré zobrazíme používateľovi. Meno procesu môžeme nájsť v */proc/<PID>/status* v prvom riadku. Vo väčšine prípadov sme dostali zmysluplný názov, ktorý používateľovi povie, o ktorú aplikáciu sa jedná. Avšak napríklad pri aplikácií Firefox meno procesu, ktorý komunikoval cez internet, bolo "GeckoMain". Tento názov by pravdepodobne používateľovi nepovedal, že ide o aplikáciu Mozilla Firefox. Ako sme zistili, */proc/<PID>/status* zobrazuje posledný komponent z cesty spúšťaného súboru, na ktorý sa odkazuje */proc/<PID>/exe*. Preto sme sa rozhodli ako meno použiť meno súboru, ktorý daný proces spustil, teda posledný komponent z nultého argumentu.

V prípade, že sa nepodarí nájsť aplikáciu, ktorá paket odoslala / príjme, používateľovi sa zobrazí meno aj cesta ako "UNKNOWN".

3.2 Pravidlá

Podobne ako iné firewally, aj ten náš si vytvára súbor pravidiel, ktorými sa riadi pri spracovávaní paketu a rozhodovaní, či ho má ponechať alebo zahodiť. Pokiaľ príde paket, ku ktorému nepasuje žiadne pravidlo, vyskočí okienko, ktoré sa používateľa spýta, čo chce s daným paketom spraviť. Používateľ má na výber dovoliť komunikovať uvedenej aplikácií len na danú IP adresu, port a protokol alebo môže povoliť komunikovať tejto aplikácií kamkoľvek. Podobne môže zamietnuť len toto spojenie alebo môže zamietnuť akúkoľvek komunikáciu danej aplikácie. Pokiaľ si používateľ nezvolí inak, toto rozhodnutie si firewall zapamätá ako nové pravidlo, ktoré bude používať pre ďalšie pakety.

Ďalšie spôsoby, ako vytvoriť a uložiť nové pravidlo, sú pomocou administrátorského centra alebo priamym editovaním súboru *firewall.data*, čo však nie je odporúčaný spôsob.

Pravidlá sú ukladané do súboru *firewall.data* v ľudsky ľahko čitateľnom formáte. Jeden riadok zodpovedá jednému pravidlu. V prvom stĺpci je cesta k aplikácií, resp. príkaz. Nasleduje vzdialená IP adresa a port. Ďalší stĺpec obsahuje protokol a v poslednom stĺpci je akcia, ktorá sa má vykonať. Jednotlivé položky sú oddelené medzerou. Akcia môže mať jednu z nasledujúcich hodnôt:

- ACCEPT – pakety, ktoré spĺňajú dané pravidlo sa ponechajú.

```
UNKNOWN 127.0.0.1 35059 tcp ACCEPT
UNKNOWN 127.0.0.1 53470 tcp ACCEPT
/opt/google/chrome/chrome 198.252.206.25 443 tcp ACCEPT
/opt/google/chrome/chrome 35.190.41.116 443 tcp ACCEPT
/opt/google/chrome/chrome * * * ACCEPT
UNKNOWN 192.168.1.254 4353 igmp ACCEPT
UNKNOWN 224.0.0.251 0 igmp ACCEPT
UNKNOWN 192.168.1.101 37892 igmp ACCEPT
/lib/systemd/systemd-resolved 192.168.1.254 53 udp ACCEPT
/lib/systemd/systemd-resolved 127.0.0.1 32781 udp ACCEPT
UNKNOWN 127.0.0.53 64405 icmp ACCEPT
/lib/systemd/systemd-resolved 127.0.0.1 43311 udp ACCEPT
/usr/sbin/NetworkManager 34.122.121.32 80 tcp ACCEPT
UNKNOWN 127.0.0.53 64484 icmp ACCEPT
UNKNOWN 127.0.0.1 771 icmp ACCEPT
UNKNOWN 127.0.0.53 64439 icmp ACCEPT
```

Obr. 3.1: Na obrázku vidíme ukážku súboru *firewall.data*. Prvý stĺpec označuje aplikáciu. V druhom stĺpci sa nachádza IP adresa, na ktorú paket posielame alebo z ktorej paket prišiel. Tretí stĺpec udáva číslo portu. Vo štvrtom stĺpci sa nachádza protokol a v poslednom stĺpci je akcia, ktorá sa má vykonať. Jednotlivé stĺpce sú oddelené medzerou.

- DENY – pakety, ktoré spĺňajú dané pravidlo sa zahodia.
- ASK – pre pakety, ktoré spĺňajú toto pravidlo, vyskočí okienko, ktoré sa spýta používateľa, čo sa má s daným paketom stať. Používateľ môže svoju odpoveď uložiť.

Čo sa týka aplikácie, okrem cesty môžeme mať ešte špeciálnu hodnotu *UNKNOWN*, čo znamená, že sa nepodarilo nájsť aplikáciu, ktorej paket patrí. Táto hodnota je vyhodnocovaná podobne ako každá iná aplikácia, teda ak povolím jednej neznámej aplikácií komunikovať na nejakú adresu, budú tam môcť komunikovať všetky neznáme aplikácie.

Ďalšia špeciálna hodnota je * (hviezdička), ktorá môže byť použitá pri aplikácií, IP adrese, porte a protokole, no nemôže byť použitá pri akcií. Hviezdička znamená ľubovoľnú hodnotu. Čiže napríklad, keď používateľ sa rozhodne povoliť akúkoľvek komunikáciu danej aplikácie, zodpovedajúci riadok bude obsahovať aplikáciu a na mieste IP adresy, portu a protokolu bude hviezdička. Štruktúru súboru *firewall.data* môžeme vidieť na obrázku

Vyhodnocovanie pravidiel funguje podobne ako v iptables. Nájde sa prvé pravidlo v zozname, ktoré vyhovuje a uvedená akcia sa aplikuje. Preto je dôležité aj poradie pravidiel.

Pridávanie nových pravidiel je trocha komplikovanejšie. V prípade, že vyskočí okienko,

ktoré sa spýta používateľa, s odpoveďou sa môžu stať dve veci - pokiaľ tam už také pravidlo existuje, ktoré sa zhoduje vo všetkom, okrem akcie, prepíše sa akcia na novú. V tomto prípade sa musí zhodovať aj žolík v podobe hviezdičky. Pokiaľ tam pravidlo ešte nie je, pridá sa na koniec.

V administrátorskom centre firewallu môžeme pridávať nové pravidlo na koniec, vymazať existujúce pravidlá, meniť akciu uloženého pravidla a meniť poradie pravidiel.

3.2.1 Bezpečné čítanie súboru

Ako vidíme na obrázku 3.1, pravidlo zodpovedá jednému riadku, ktorý treba správne rozdeliť na stĺpce. Na to používame funkciu `sscanf()`, ktorá umožňuje čítať formátovaný reťazec podobne ako funkcia `scanf()`. No táto funkcia nekontroluje, či prečítaný reťazec sa zmestí do premennej. Aby sme sa vyhli pretečeniu, môžeme použiť špecifikátor šírky, ktorý hovorí, koľko najviac znakov môžeme prečítať. Táto hodnota by mala byť o jedna menšia ako celková dĺžka premennej, keďže reťazec musí byť zakončený špeciálnym symbolom.

Keďže niektoré veľkosti reťazcov priamo nepoznáme, ale sú zadefinované v použitých knižniciach, potrebujeme môcť zadávať šírku ako premennú. K tomu sme si vytvorili makro, ktoré z premennej vytvorí formátovaný reťazec, v ktorom sú doplnené šírky načítavaných reťazcov z príslušných premenných.

3.3 Práva

Keďže cieľom tejto práce je zvýšiť bezpečnosť zariadení s Linuxom, pri implementácií sme mysleli aj na bezpečnosť. V každom väčšom programe sa môžu vyskytnúť chyby, ktoré predstavujú potenciálne zraniteľnosti, a preto je dobré čo najviac možné škody odstrániť. Z toho dôvodu uplatníme princíp najmenších práv.

Každý proces má niekoľko atribútov, ktoré hovoria, s akými právami daný proces beží. Prvým je reálne UID, ktoré hovorí,ktorý používateľ proces spustil. Ďalej máme efektívne UID, ktoré hovorí s akými právami proces vykonáva operácie. Ak máme privilegovaný proces, ktorý sa chce dočasne vzdať privilegovaných práv, táto hodnota sa uloží do uloženého UID, čo umožní neskôr ju obnoviť. Podobné hodnoty máme aj pre skupinu.

Najskôr sa pozrieme, ktoré časti potrebujú privilegované práva. Na začiatku sa musíme pripojiť k NFQUEUE. Tieto operácie potrebujú privilegované práva. Po vytvorení pripojenia na frontu sa môžeme práv vzdať. Ďalším miestom v kóde, kde potrebujeme privilegované práva, je hľadanie procesu, keď poznáme inode. Bez privilegovaných práv by sme boli schopní rozpoznať len procesy, ktoré vytvoril náš používateľ, no my by sme chceli priradiť soket aj k procesom iných používateľov. Udelenie verdiktu pre paket si

tiež vyžaduje privilegované práva.

Ako ďalšie miesto, ktoré potrebuje privilegované práva, sa ukázalo byť vytváranie okienka. Dôvod je ten, že používateľ firewall nemôže vytvoriť okienko pre používateľa, ktorý je prihlásený. Keďže volať pythonový modul s privilegovanými práva nie je z bezpečnostného hľadiska dobré, zistíme prihláseného používateľa a zmeníme práva na neho. Stále sú potrebné privilegované práva na importovanie modulu, no funkcia sa už zavolá s právami bežného používateľa.

Firewall potrebuje na niektoré operácie privilegované práva, preto nemôže bežať len pod bežným používateľom. Keď už musíme mať väčšie práva, ideálne by bolo na začiatku spraviť všetky privilegované operácie a potom sa vzdať práv, no ani to nie je náš prípad. Preto budeme meniť len efektívne UID.

Pri inštalácií vytvárame nového systémového používateľa s menom *firewall*. Vytvorenie nového používateľa má zároveň výhodu, že náš firewall nemá prístup k súborom bežných používateľov. Program sa spúšťa s privilegovanými právami. Po úvodnom nastavení sa vzdáme privilegovaných práv a zmeníme efektívne UID na ID používateľa firewall. V prípade, že znova potrebujeme vykonať privilegovaný príkaz, obnovíme efektívne UID z uloženého UID.

Okrem používateľského ID je potrebné zmeniť aj ID skupiny. Zmenu UID a GID musíme vykonať v správnom poradí, inak sa zmena nemusí podariť. Na zmenu GID je totiž potrebné mať privilegované práva. Preto pri znižovaní práv sa musíme najskôr zmeniť GID a až potom UID. Naopak, keď obnovujeme práva, najskôr nastavíme UID a až potom GID[19].

Okrem toho, dieťa si sleduje, či jeho rodičovský proces neumrel a ak áno, umrie tiež. No toto nastavenie sa vymaže pri zmene práv, preto ho potrebujeme obnoviť pri každej zmene práv[10].

3.4 Synchronizácia procesov

Keď sa aplikácia skladá z viacerých procesov, prirodzene musíme riešiť, ako tieto procesy spolu budú komunikovať a synchronizovať sa.

Oba procesy čítajú uložené pravidlá a prípadne ich môžu aj meniť. Okrem toho ich môže čítať a meniť aj proces administrátorského centra. Pokiaľ by dáta len čítali, nevadí nám, že ich číta viac procesov naraz. No pokiaľ jeden proces zapisuje, nechceme aby niekto iný tie dáta čítal, kým nezapíše všetko.

Pri výbere vhodného spôsobu zamykania sme museli myslieť aj na to, aby bol zámok dostupný v oboch použitých jazykoch a aby k nemu mali prístup všetky tri procesy. Použitie mutexov nebolo vhodné. Radšej sme zvolili uzamykanie čítaného súboru.

V Unixovým operačných systémoch je problém, že neexistuje dobré riešenie na po-

vinné zámky (mandatory locks), ktoré sú vynútené operačným systémom. Existujú nejaké implementácie, ale majú svoje chyby a vo všeobecnosti sa neodporúča ich použitie. Preto sme použili advisory zámky. Tento druh zámkov funguje dobre medzi spolupracujúcimi procesmi, ktoré o zámku vedia a používajú ho, ale ich použitie nie je vynútené operačným systémom. Na synchronizáciu našich procesov je to postačujúce, keďže väčšina používateľov si nebude pýtať zámok, nie je odporúčané, aby menili pravidlá priamym editovaním súboru *firewall.data*.

Okrem toho je dôležité, aby procesy tvoriace firewall o sebe vedeli, či ten druhý je stále nažive. Rodičovský proces dáva pozor, či nedostal SIGCHLD signál. Ak áno, postará sa o svoje mŕtve dieťa a sám skončí. Podobne detskému procesu nastavíme, aby dával pozor, či jeho rodič nezomrel. Aby sme predišli súbehu, kedy sme vytvorili nový proces, rodič skončil a až potom dieťa začalo sledovať stav rodiča (teda nezachytilo signál, že rodič umrel), musíme skontrolovať, či je jeho pôvodný rodič stále živý. To zistíme tak, že sa spýtame na PID rodiča. Proces s $PID = 1$ si adoptuje osirelé procesy, ktorým zomrel rodič. A teda, pokiaľ náš proces dostane odpoveď, že rodičovské proces ID je 1, vieme, že jeho rodič už nežije, a teda aj detský proces skončí.

3.5 iptables

Aby jadro Linuxu vedelo, že má posielať pakety do NFQUEUE, odkiaľ ich bude čítať náš firewall, treba pridať do iptables vhodné pravidlo. Najskôr skontrolujeme, či sa tam dané pravidlo nenachádza, aby sme predišli duplicitnému pravidlu. To urobíme príkazom `iptables -C OUTPUT -j NFQUEUE -queue-num 0` pre odchádzajúce pakety. Ak treba, vložíme pravidlo príkazom `iptables -A OUTPUT -j NFQUEUE -queue-num 0`. Podobne postupujeme aj pre prichádzajúce pakety, ktoré sú spracovávané INPUT reťazou.

Keďže neexistuje knižnica pre C, ktorá by nám umožňovala pridávať pravidlá do iptables, vytvorili sme skript *ibtab.sh*, ktorému zadáme ako parameter číslo fronty. Tento skript voláme priamo z programu na začiatku. Alternatívne by sme mohli nastaviť, aby systemd pred spustením firewallu spustil tento skript. Výhodou je, že ak by sme chceli zmeniť číslo použitej fronty, stačí nám upraviť hodnotu v *constatnts.c*.

K spôsobu, akým pridávame pravidlá do iptables, je potrebné poznamenať, že nie je odolný voči súbehu. Ak by sa dva procesy naraz pokúšali pridať rovnaké pravidlo, môže tam byť pridané dvakrát.

3.6 Služba

Väčšinu času náš firewall bude bežať na pozadí a čakať na prijatie paketu. Taktiež potrebujeme, aby sa program spustil pri zapnutí počítača a ak z nejakého dôvodu skončí, aby sa zapol znova. Mohli by sme síce tieto vlastnosti si naprogramovať sami, ale Linux nám ponúka oveľa jednoduchšie riešenie – systemd, ktorý slúži na spravovanie jednotlivých služieb.

Systemd má zoznam *.service* súborov umiestnených v priečinku */etc/systemd/system*, pričom každý súbor zodpovedá jednej službe. Vytvorením vhodného súboru bude náš firewall automaticky bežať na pozadí. Poďme sa teda pozrieť, ako vyzerá daný súbor a ako bude vyzerať v našom prípade.

Súbor pozostáva z niekoľkých sekcií.Prvou sekciou je [UNIT], v ktorej sa nachádzajú všeobecné informácie. Dôležitý parameter je Description, ktorý hovorí, čo daná služba budem robiť. Môžeme pridať aj manuálovú stránku. Ďalej sú tu definované závislosti, ktoré môžu byť viacerých typov:

- Requires – pri aktivácií danej jednotky budú aktivované aj všetky ostatné jednotky z tohto zoznamu. Pokiaľ niektorá zo zoznamu skonči, ukončí to aj našu jednotku
- Wants – je podobné ako Requires, ale pokiaľ niektoré jednotka zlyhá, bude to odignorované
- After – jednotka bude spustená až po všetkých jednotkách zo zoznamu
- Before – jednotka je spustená pred uvedenými jednotkami
- Conflicts – pokiaľ je spustená táto jednotka, nemôže bežať žiadna z tohto zoznamu a naopak

Ďalšou dôležitou sekciou je [Install]. Tu sú dôležité parametre WantedBy, resp. RequiredBy, ktoré nám hovoria kam sa majú pridať závislosti. Existujú typické ciele, ktoré zodpovedajú úrovniam zo SysV:

Runlevel	**Target**
0	poweroff.target
1	rescue.target
2, 3, 4	multi-user.target
5	graphical.target
6	reboot.target

Ďalšie sekcie závisia od typu jednotky. V našom prípade pôjde o službu, preto sa zameriame na tento typ.

Pri službe špeciálnou sekciou je [Service]. V nej je dôležité zadefinovať parameter ExecStart, ktorý hovorí, čo sa má spustiť, teda hlavný proces služby. Okrem toho môžeme definovať, čo sa má spustiť pred alebo po. Potom definujeme typ služby:

- simple – základný typ, kedy sa spustí hlavný proces príkazom ExecStart
- dbus – podobné ako simple, ale navyše obsahuje parameter BusName, ktorý hovorí, ako môžu iné procesy komunikovať s touto službou
- oneshot – očakáva sa, že služba skončí a až potom sa začnú spúšťať ďalšie jednotky
- notify – posiela signál, keď je naštartovaná
- forking – pôvodný proces zavolá fork a skončí. Za hlavný proces sa potom považuje dieťa procesu
- idle – táto služba je spustená až keď ostatné služby boli vybavené

Okrem toho môžeme špecifikovať ďalšie parametre[11]. Keď poznáme štruktúru súbotu, pozrieme sa na súbor *firewall.service* vytvorený k nášmu programu (obr. 3.2)

```
[Unit]
Description=Firewall service
After=network.target
StartLimitIntervalSec=0

[Install]
WantedBy=graphical.target

[Service]
Type=simple
Restart=always
RestartSec=1
User=root
Environment=DISPLAY=:0
Environment=XAUTHORITY=/home/jarka/.Xauthority
Environment=USER=jarka
ExecStart=/home/jarka/firewall/firewall
WorkingDirectory=/home/jarka/firewall
```

Obr. 3.2: Ukážka súboru *firewall.service*

Okrem už známych parametrov,naša služba ma zadefinovaný pracovný priečinok, ktorý zodpovedá priečinku, v ktorom sa nachádza spúšťateľný súbor. Okrem toho máme definovaného používateľa, pod ktorým má služba bežať. Uviedli sme roota, čo

je predvolená možnosť, pokiaľ sa neuvedie inak. Parameter Restart nám hovorí, že po skončení programu z ľubovoľného dôvodu sa systemd má pokúsiť spustiť náš firewall znova, pričom RestartSec uvádza, koľko sa má čakať medzi jednotlivými pokusmi. Parameter StartLimitIntervalSec vymedzuje časový interval, počas ktorého ak sa prekročí istý počet pokusov o spustenie, systemd sa prestane snažiť. Hodnota 0 znamená, že sa nemá prestať nikdy.

Procesy spustené zo systemd nemajú žiadne premenné prostredia, Pokiaľ teda daná služba nejaké potrebuje, musí si ich zadefinovať sama. V našom prípade potrebujeme premenné, ktoré nám umožnia zobrazovať grafické prvky. Preto nastavujeme hodnoty DISPLAY a XAUTHORITY premenných[1]. Okrem toho, nastavíme aj prihláseného používateľa, na ktorého budeme za behu meniť práva pri vytváraní vyskakovacieho okna.

Ovládanie služieb je jednoduché. Na spustenie danej služby stačí zadať príkaz `systemctl start firewall.service` a na vypnutie systemctl stop firewall.service. Automatické spustenie služby pri zapnutí počítača nastavíme príkazom systemctl enable firewall.service a zrušenie systemctl disable firewall.service. Logovacie záznamý získame príkazom `journalctl -u firewall.service`.

3.7 Administrátorské centrum

Samostatnú časť tvorí administrátorské centrum, ktoré slúži na editovanie pravidiel. Je nakódené v Pythone s využitím grafickej knižnice Tkinter. Zobrazuje pravidlá zo súboru *firewall.data*. Vieme v ňom jednoducho zmeniť akciu v danom pravidle. Ďalej máme možnosť pravidlá pridávať a odstraňovať. Okrem toho môžeme pravidlá posúvať, keďže ako sme uviedli v sekcií 3.2, na poradí pravidiel záleží. Pri pridávaní nového pravidla program nekontroluje správnosť údajov - za tú zodpovedá používateľ.

Pri čítaní a zapisovaní do súboru *firewall.data* si centrum pýta zámky, preto je to preferovaný spôsob robenia zmien v uložených pravidlách. Neodporúča sa ručné editovanie súboru. No program nedeteguje zmeny súboru urobené počas jeho behu. Čiže pokiaľ počas toho, ako máme spustené administrátorské centrum, vyskočí okienko a my uložíme odpoveď, nové pravidlo sa nezobrazí a nepridá automaticky aj do administrátorského centra. Je tam možnosť znova načítať dáta zo súboru, avšak tým prídeme o všetky zmeny, ktoré sme spravili.

Okrem synchronizácie prístupu k súboru *firewall.data* bolo treba vyriešiť aj to, ako bude administrátorské centrum vedieť, kde sa daný súbor nachádza. Narozdiel od samotného firewallu, ktorého skompilovaný binárny súbor je umiestnený v tom istom priečinku ako sa nachádza súbor *firewall.data*, administrátorské centrum bude umiestnené do priečinka, kde sa inštalujú pythonové knižnice. Preto je potrebné, aby

si pred inštaláciou uložil cestu k súboru s dátami. V inštalačnom skripte si zistíme aktuálne umiestnenie, z ktorého odvodíme absolútnu cestu k súboru *firewall.data*. Túto cestu uložíme do súboru, ktorý pribalíme k modulu administrátorského centra. Hoci by sa mohlo zdať, že toto stačí, opak je pravdou. Po priložení súboru k modulu vzniká otázka, ako sa k danému súboru dostať. K tomu je potrebné použiť knižnicu *importlib-resources*. Aby to nebolo také jednoduché, staršie verzie Pythonu potrebujú nainštalovať knižnicu *importlib_ resources* z PyPI, zatiaľčo novšie verzie môžu využiť knižnicu *importlib.resources*, ktorá je súčasťou štandardných knižníc[6].

Kapitola 4

Testovanie a vyhodnotenie

V tejto kapitole sa pozrieme na nasadenie a otestujeme náš firewall. Zameriame sa na Debian a ďalšie distribúcie založené na Debiane, keďže pri vývoji aj testovaní bola použitá jedna z týchto distribúcií. Zdrojový kód sa nachádza v prílohe a je dostupný aj na githube[12].

4.1 Kompilovanie a inštalácia

Skôr, ako začneme firewall používať a testovať musíme ho najprv nainštalovať. Na kompiláciu potrebujeme GCC kompilátor. Okrem toho je potrebné mať nainštalovaný Pyhton3, python3-dev, python3-tk a python3-setuptools. Ďalej potrebujeme libnetfilter-queue-dev. Všetko uvedené môžeme nainštalovať pomocou príkazu `apt-get install`. Samozrejme, potrebujeme mať priečinok so všetkými súbormi, ktorý umiestnime tam, kde chceme, aby bol firewall nainštalovaný. Je potrebné, aby používateľ firewall mal prístup čítania a zapisovania do priečinka. Netreba zabudnúť nastaviť právo spúšťania na súbory *iptab.sh* a *install.sh*.

Proces kompilácie a inštalácie sme sa snažili čo najviac automatizovať, i keď niektoré kroky nebolo jednoduché spraviť automaticky, preto ich musí spraviť používateľ ručne. Prvým krokom je kompilácia. Vytvorili sme *Makefile*, ktorý skompiluje celú aplikáciu. Avšak na to, aby mohol prilinkovať knižnicu *Python.h*, ktorá nám umožňuje z programu v C volať pythonovské funkcie, potrebujeme nastaviť správne parametre, ktoré závisia od verzie Pythonu, ktorá sa na danom zariadení nachádza. Našťastie Python so sebou prináša skript *python3-config* (prípadne s iným číslom podľa verzie Pythonu), ktorý je umiestnený v */bin* priečinku. Používateľ pred kompilovaním by mal spustiť v termináli príkaz `/bin/python3-config -includes -libs -embed` a výsledok skopírovať do premennej `PYINC` v *Makefile*. Potom môžeme spustiť príkaz `make`, ktorý nám skompiluje celý firewall.

Ďalej potrebujeme vytvoriť používateľa firewall. Okrem toho treba doplniť do *fire-*

wall.service hodnoty jednotlivých parametrov a umiestniť ho na správne miesto. Potom môžeme spustiť danú službu a nastaviť, aby sa spustila pri spustení Linuxu.

Okrem toho je potrebné nainštalovať administrátorské centrum. To je nakódené ako modul v Pythone, preto na inštaláciu sme sa rozhodli použiť skript *setup.py*, čo je štandardný spôsob inštalovania modulov. Pokiaľ takýto skript existuje, na inštaláciu stačí spustiť príkaz `python3 setup.py install`.

Aby sme zjednodušili inštaláciu, kroky uvedené v predchádzajúcich dvoch odsekoch sa vykonajú automaticky spustením skriptu *install.sh*, ktorý je súčasťou nášho firewallu. Tento skript na správne fungovanie potrebuje privilegované práva.

Keďže pri inštalácií zadávame premenné prostredia, ktoré platia pre daného používateľa, ktorý je zrovna prihlásený, firewall bude správne fungovať len pre neho.

Kompilovanie a linkovanie s Pythonom nemusí prebehnúť vždy hladko. V prípade problémov môže pomôcť spustiť *python3-config* s inými argumentami, ktoré pomôžu nájsť argumenty potrebné k úspešnému skompilovaniu. Viac o tomto skripte sa môžeme dočítať v dokumentácií[4].

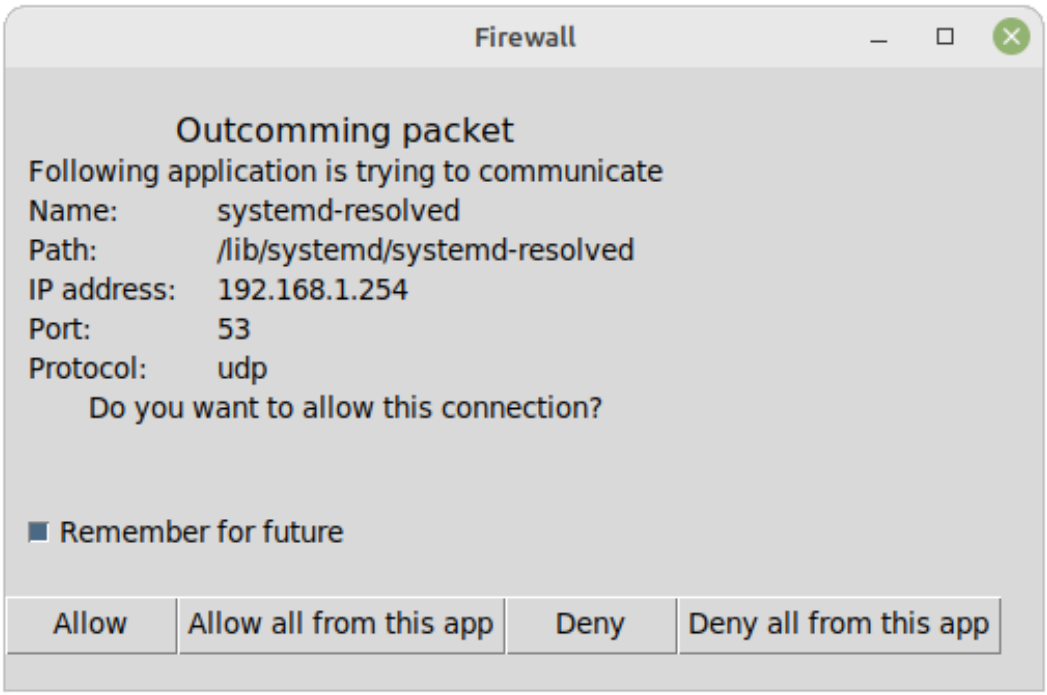
4.2 Testovanie

V tejto časti sa pozrieme na to, ako prebiehalo testovanie. Použili sme notebook s operačným systémom Linux Mint 20.3 Una. Verzia Pythonu bola 3.8.

Firewall sme nainštalovali vyššie uvedeným spôsobom a otestovali ho v bežnej prevádzke. Keďže inštalačný skript zároveň spustí službu firewall, hneď sa nám zobrazilo okno pýtajúce sa, či chceme službe systemd-resolved povoliť komunikovať cez internet. Takéto okienko môžeme vidieť na obrázku 4.1

Vyskakujúce okno nám povie, ťi ide o prichádzajúci paket alebo odchádzajúci. Ďalej tu nájdeme meno aplikácie a cestu, resp. nultý argument. Zobrazuje nám aj vzdialenú IP adresu, port a protokol. My sa môžeme rozhodnúť, či dovolíme uvedenej aplikácií komunikovať na danú adresu a port, či jej dovolíme posielať pakety kamkoľvek alebo naopak či jej zakážeme komunikovať na danú adresu a port, prípadne jej nedovolíme posielať žiadne pakety. Pokiaľ neklikneme na zaškrtávacie okno, odpoveď sa zapamätá a bude použitá pri spracovávaní ďalších paketov.

Firewall zachytil pakety a našiel aplikáciu, ktorá ho odoslala. Skúšali sme aplikácie bežne využívajúce internet, ako napríklad Google Chrome, Mozilla Firefox, Signal A Thunderbird. Ďalšie aplikácie, ktorých pakety sme zachytili, boli systemd-resolved, texstudio, NetworkManager, systemd-timesyncd, Clion, xbrlapi, node, snapd, http a https methods pre APT, git-remote-https, mintUpdate, lsof, gvfsd-smb-browse a cups-browsed. Taktiež sa vyskytlo veľa paketov, ku ktorým sa nám nepodarilo zistiť aplikáciu, ktorá ich odoslala.



Obr. 4.1: Vyskakujúce okno, ktoré sa zobrazí, keď firewall spracováva paket, ku ktorému ešte nemá uložené pravidlo

Treba podotknúť, že nie všetky uvedené aplikácie sa snažili komunikovať cez internet. Niektoré sa pripájali len na localhost. Počas implementácie sme rozmýšľali, či chceme aj tieto pakety zachytávať, no rozhodli sme sa používateľovi dať úplný prehľad paketov, ktoré na jeho zariadení chodia. Všimli sme si napríklad, že systemd-resolved sa pripája na localhost a zakaždým použije iný port, čo nám vyrobilo veľa pravidiel, keď sme ich všetky chceli ukladať. Preto sme v administrátorskom centre pridali pravidlo, ktoré umožňovalo danému procesu komunikovať ľubovoľným portom.

Ďalej sme si všimli, že Chrome okrem komunikácie potrebnej na otvorenie danej web-stránky posiela nejaké pakety na IP adresy Googlu. Keďže Google vlastní veľkú podsieť, tieto IP adresy boli rôzne, čo opäť produkovalo veľa pravidiel. Ako riešenie by sa zišla možnosť pridávať k adresám aj masky, ale to zatiaľ náš firewall nedovoľuje.

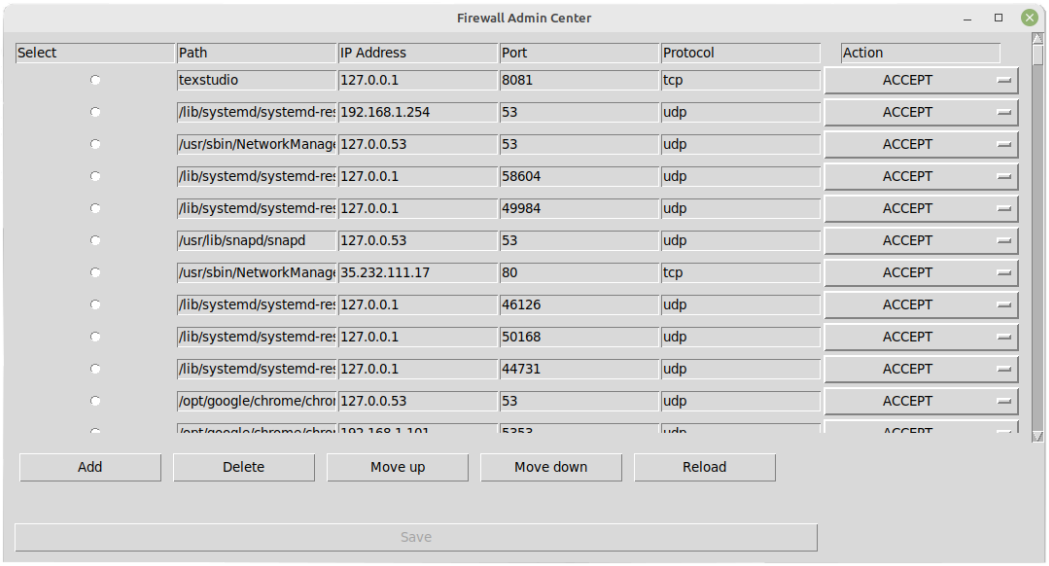
Medzi paketmi, ku ktorým náš firewall nedokázal nájsť aplikáciu, boli napríklad tie používajúce IGMP protokol pochádzajúce od routera. Zachytili sme aj pakety používajúce ICMP protokol. Ďalej išlo o komunikáciu na získanie IP adresy pomocou DHCP protokolu. Vyskytli sa aj DNS žiadosti, monitorovanie siete pomocou SNMP a pakety NTP na synchronizáciu času. No medzi neznámymi paketmi sa vyskytol aj prípad, kedy používateľ jasne vie identifikovať proces. Keď sme spustili ping, firewall pakety síce zachytil, ale nevedel rozpoznať, ktorý proces ich posiela. Navyše, keďže ping nemá určený port, tieto pakety mali zakaždým inú hodnotu portu, čo môže viesť k veľkému počtu pravidiel. Keď sme pozreli súbor */proc/net/icmp*, našli sme tam záznam. No

keď sme chceli nájsť proces, ktorý má otvorený daný soket pomocou lsof, neboli sme úspešní.

4.2.1 Administrátorské centrum

Okrem samotného firewallu sme otestovali aj administrátorské centrum.

Na spustenie centra stačí zadať do terminálu príkaz `firewall_admin`. Zobrazí sa okno obsahujúce zoznam pravidiel, ktoré môžeme vidieť na obrázku 4.2.



Obr. 4.2: Administrátorské centrum, v ktorom môžeme upravovať súbor pravidiel

V administrátorskom centre sme vyskúšali pridať nové pravidlo, zmazať existujúce pravidlo, zmeniť akciu ako aj zmeniť poradie pravidiel. Taktiež sme znovu načítali pravidlá potom, ako sme pridali pravidlo z vyskakujúceho okienka. Všetko nám fungovalo bez problémov. Je dôležité poznamenať, že na to, aby sa zmeny prejavili v správaní firewallu, je potrebné ich najskôr uložiť. V prípade problému s ukladaním treba skontrolovať, či daný proces má právo zapisovať do súboru *firewall.data*.

4.2.2 Latencia

Počas nášho testovania sme pridali celkovo zhruba 700 pravidiel. Pravidlá vznikli tým, že sme zakaždým povolili konkrétne spojenie, ale žiadnej aplikácií sme nedovolili komunikvať všade. Keď sme potom pridali na koniec zoznamu pravidlo, ktoré povoľovalo Google Chrome-u komunikovať s celým internetom a spustili video (s tým, že nemáme skoršie pravidlo umožňujúce danú komunikáciu), pri vyšších rozlíšeniach sme už videli problémy s prehrávaním.

Pri podobnom experimente so zhruba 500 pravidlami firewall zvládal aj prehrávajúce videá.

Nie je odporúčané bezmyšlienkovite vytvárať pravidlá povoľujúce jedno spojenie zakliknutím okienka. Je vhodné sa zamyslieť a vhodne zvoliť sady pravidiel. Taktiež je dobré staré pravidlá vymazávať a často používané pravidlá dať na začiatok.

4.3 OpenSnitch

Ako sme naznačili v sekcií 1.3.3, vyskúšali sme OpenSnitch, verzia 1.5.1, v čase písania práce najnovšia. Testovali sme ju vo virtuálnom stroji s OS Linux Mint 20.3.

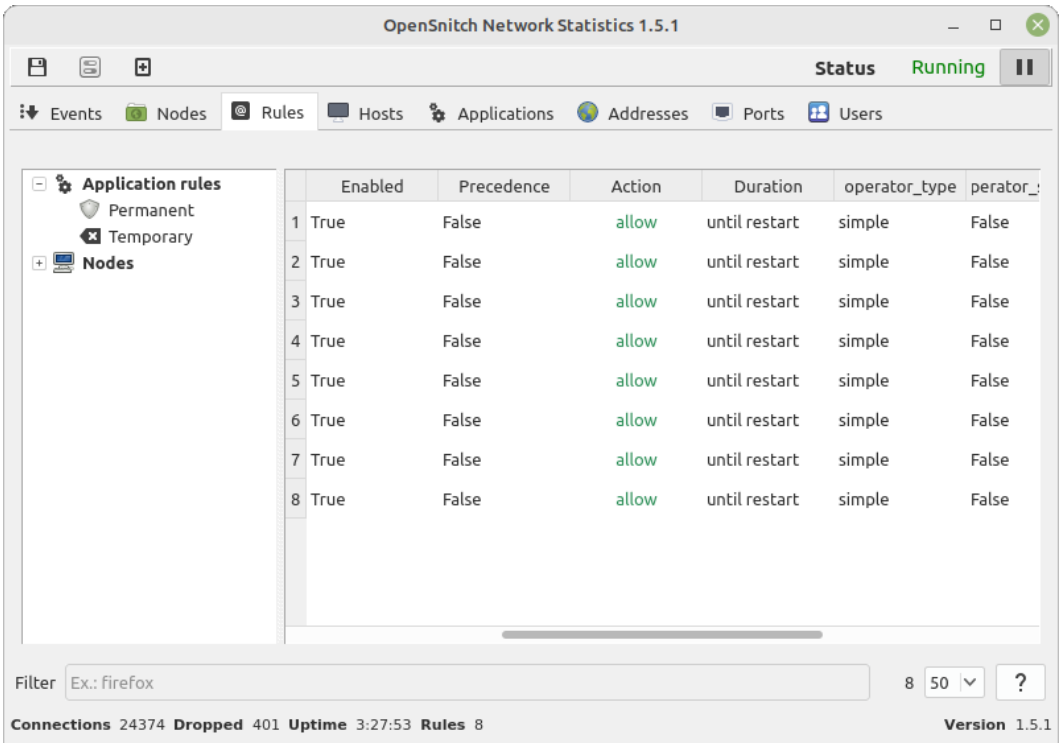
Inštalácia bola pomerne jednoduchá, stačilo použiť inštalačný balíček. Mohli sme si všimnúť, že pribudli dva procesy - opensnitchd, ktorý beží s privilegovanými právami, a opensnitch-ui s právami bežného používateľa.

Keď OpenSnitch zachytil paket, vyskočilo okienko, ktoré zobrazuje informácie o aplikácií, protokole, doméne, zdrojovej a cieľovej IP adrese, cieľovom porte, používateľovi a proces ID. Používateľ si môže vybrať, či chce pravidlo uplatniť na daný spustiteľný súbor, pre celý príkaz, pre pakety smerujúce na daný port, doménu, IP adresu (vrátane masiek), od daného používateľa alebo od procesu s daným PID. Stlačením tlačidla "+"môžeme tieto možnosti skombinovať. Ďalej máme možnosť nastaviť, ako dlho má pravidlo platiť - raz, do reštartu, navždy alebo môžeme vybrať niektorý z časových intervalov. Nakoniec zvolíme akciu. Celé okno sa zobrazí na 15s, potom sa použije predvolené akcia. Príklad takéhoto okna môžeme vidieť na obrázku 4.3.

Obr. 4.3: Vyskakovacie okno OpenSnitchu. Môžeme si všimnúť, že oproti nášmu firewallu zobrazuje viac informácií aj dáva viac možností na vytvorenie pravidla

Okrem toho máme k dispozícii GUI aplikáciu, ktorú môžeme vidieť na obrázku 4.4.

Tá zobrazuje štatistiky, posledné udalosti, ale aj pravidlá, ktoré tu môžeme meniť a pridávať nové.



Obr. 4.4: GUI aplikácia OpenSnitchu, ktorá zobrazuje štatistiky a umožňuje upravovať pravidlá ako aj vytvárať nové. Na obrázku sú zobrazené pravidlá.

Keď porovnáme OpenSnitch s naším firewallom, prvý menovaný ponúka viac možností, na čo sa môžeme pýtať v rámci pravidla. TAktiež je výhoda, že podporuje IPv6. No všimli sme si, že nezachytáva všetky pakety, ale len tie, čo majú protokol TCP, UDP, UDPLITE a ich verzie pre IPv6. Čiže napríklad vyššie spomenutý ping OpenSnitch firewall ani nezachytí. Okrem toho, počas testovania sa nestalo, že by OpenSnitch zachytil paket, ku ktorému by nevedel priradiť aplikáciu. Nevieme povedať, či je to tým, že je lepší v určovaní aplikácie (počas testovania sme videli len tie aplikácie, ktoré zachytil aj náš firewall), alebo tým, že ak neviem k paketu priradiť aplikáciu, nezobrazí používateľovi okienko. Autorke práce sa tiež nepáčilo, že okno sa zobrazí len na 15s, lebo je to pomerne krátko, pokiaľ chceme meniť predvolené hodnoty. No tento čas sa dá nastaviť. Hodnotíme pozitívne možnosť vybrať, ako dlho má dané pravidlo platiť, vďaka čomu nebude v zozname zbytočne veľa starých pravidiel, ktoré sa nikdy nepoužijú.

Ako hlavnú výhodu nášho firewallu vnímame v tom, že zachytáva viac paketov bez ohľadu na protokol. Taktiež náš firewall sa vzdáva privilegovaných práv, pokiaľ ich nepotrebuje narozdiel od OpenSnitch, ktorého démon beží s privilegovanými právami stále.

Záver

Cieľom tejto práce bolo navrhnúť, implementovať a otestovať firewall pre Linux, ktorý umožní filtrovať komunikáciu na základe aplikácie. Analyzovali sme požiadavky a navrhli riešenie. Počas implementácie sme objavili nedostatky v pôvodnom návrhu, a tak sme ho prerobili a implementovali zmeny.

Počas implementácie sme narazili na rôzne problémy, ktoré sme museli vyriešiť, aby sme dostali fungujúci a bezpečný program, ktoré sme úspešne prekonali. Na záver sme náš firewall otestovali a porovnali s iným riešením.

Vývoj firewallu ani zďaleka nie je na konci. Náš firewall by bolo dobré rozšíriť aj na IPv6, keďže zatiaľ podporuje len IPv4. Pri vytváraní pravidiel by sa nám hodila možnosť zadať masku siete, nie len konkrétnu IP adresu. Na optimalizáciu výkonu môžeme automaticky akceptovať pakety v rámci spojenia bez ďalšieho vyhodnocovania. Ďalší nápad na pokračovanie je vylepšenie administrátorského centra, aby sa pridalo doňho nové pravidlo vytvorené okienkom automaticky. Taktiež by mohla byť užitočná možnosť označiť naraz viac pravidiel a vymazať všetky naraz. Mali sme v pláne aj pridať možnosť konfigurácie, napríklad zvoliť, ktoré nfqueue sa majú použiť.

V rámci testovania by bolo dobré otestovať náš firewall aj na ďalších distribúciach.

Literatúra

[1] ArchWiki. *systemd/User*. Dostupné na https://wiki.archlinux.org/title/systemd/User#DISPLAY_and_XAUTHORITY.

[2] David S. Miller Daniel Borkman. net: add bpfilter, 2018. Dostupné na https://lwn.net/Articles/747504/.

[3] Justin Ellingwood. *A Deep Dive into Iptables and Netfilter Architecture*, August 20, 2015.

[4] Python Software Foundation. *1. Embedding Python in Another Application*. Dostupné na https://docs.python.org/3/extending/embedding.html#compiling-and-linking-under-unix-like-systems.

[5] Guillaume Hain. *Douane*. Dostupné na https://douaneapp.com/.

[6] John T. Wodder II. *Using Package Data in Python Projects with Setuptools*. Dostupné na https://github.com/jwodder/kbits/blob/master/src/posts/pypkg-data.rst#accessing-package-data-at-runtime.

[7] Inc. Kerio Technologies. *Kerio Control*. Dostupné na https://www.gfi.com/products-and-solutions/network-security-solutions/kerio-control.

[8] Linux Kernel. Changelog kernel 2.6.14, 2005. Dostupné na http://ftp.dei.uc.pt/pub/linux/kernel/v2.6/ChangeLog-2.6.14.

[9] The kernel development community. *The proc/net/tcp and proc/net/tcp6 variables*. Dostupné na https://docs.kernel.org/networking/proc_net_tcp.html.

[10] Michael Kerrisk. *prctl(2) — Linux manual page*. Dostupné na https://man7.org/linux/man-pages/man2/prctl.2.html.

[11] Michael Kerrisk. *systemd.service(5) — Linux manual page*. Dostupné na https://man7.org/linux/man-pages/man5/systemd.service.5.html.

[12] Jaroslava Kokavcová. *kokinka/firewall*. Dostupné na https://github.com/kokinka/firewalle.

[13] Simone Margaritelli. *OpenSnitch*. Dostupné na `https://github.com/evilsocket/opensnitch`.

[14] Steven McCanne and Van Jacobson. The bsd packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, USENIX'93, page 2, USA, 1993. USENIX Association.

[15] Paul "Rusty"Russell. *Linux IP Firewalling Chains*. Dostupné na `http://people.netfilter.org/rusty/ipchains/`.

[16] Paul "Rusty"Russell. *Netfilter Project*. Dostupné na `https://netfilter.org/`.

[17] Steve Suehring. *Linux Firewalls: Enhancing Security with Nftables and Beyond*. Addison-Wesley, 2015.

[18] Netfilter Core Team. *nftables*. Dostupné na `https://netfilter.org/projects/nftables/`.

[19] John Viega and Matt Messier. *Secure programming cookbook for C and C++ - recipes for cryptography, authentication, networking, input validation and more.* O'Reilly, 2003.

[20] Klaus Wehrle. *The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel*. Prentice Hall, 2004.

[21] SELinux Wiki. Main page — selinux wiki,, 2017. Dostupné na `https://selinuxproject.org/w/?title=Main_Page&oldid=1842`.