Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gregor Robert Krmelj

Dinamično dodeljevanje dostopa do omrežnih naprav

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Mojca Ciglarič
Somentor: asist. dr. Matjaž Pančur

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Preučite delovanje požarnih pregrad s poudarkom na možnosti za dinamično dodeljevanje dostopa do omrežnih naprav. Preverite predlagane pristope in morebitne odprtokodne implementacije. Preučite njihove prednosti in slabosti. Pojasnite pojma preverjanje pristnosti z enim paketom in programsko določanje roba omrežja in na podlagi tega zasnujte lasten pristop, ki bo odpravljal zaznane slabosti in zagotavljal višji nivo varnosti. Sistem tudi implementirajte in preverite njegovo delovanje.

Iskreno se zahvaljujem svoji mentorici izr. prof. dr. Mojci Ciglarič za njeno spodbujanje in pomoč pri izdelavi diplomske naloge. Zahvaljujem se tudi svojemu somentorju asist. dr. Matjažu Pančurju za svetovanje pri sami implementaciji programske opreme in potrpežljivo odgovarjanje na moja številna vprašanja. Poleg tega bi se zahvalil asist. Mihu Groharju za njegovo sodelovanje in podporo pri naših pogovorih o SPA in SDP.

Posebna zahvala tudi vsem prijateljem in moji družini, ki so me spodbujali in podpirali med študijem in med pisanjem diplomske naloge.

Za razvoj bolj varnih aplikacij na internetu.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
IP	internet protocol	internetni protokol
UDP	user datagram protocol	nepovezalni protokol za prenašanje paketov
PK	port knocking	trkanje na vrata
SPA	single packet authorization	preverjanje pristosti z enim paketom
SDP	software defined perimeter	programsko določen rob omrežja
CSA	cloud security alliance	neprofitna organizacija, ki spodbuja varnost v oblačnih storitvah

Povzetek

Naslov: Dinamično dodeljevanje dostopa do omrežnih naprav

Avtor: Gregor Robert Krmelj

Programska oprema postaja vse bolj kompleksna, tako po številu vrstic kot tudi po funkcionalnosti. V želji, da bi programski izdelek čim prej tržili velikokrat programska oprema vsebuje določene pomanjkljivosti zaradi katerih izdelek nima zadostne zaščite. Obstoječi varnostni mehanizmi, ki naj bi zagotavljali, da aplikacije ne bi bile dostopne za neznane oziroma nezaželene uporabnike, se ponavadi aktivirajo šele v višjih slojih aplikacij. Z obstoječimi varnostnimi mehanizmi praviloma težje določimo, ali ima uporabnik pravice dostopa na nižjih slojih. V primeru, ko so ranljivi mehanizmi avtentikacije in avtorizacije, je ogrožena splošna varnost programske storitve.

Diplomsko delo predstavi možnost omejitve dostopa do sistema na omrežnem sloju, tako da dinamično dodelimo pravice uporabniku v požarnem zidu. Posledično to omogoča, da storitev vzpostavimo na internetu in da je le ta omrežno nevidna vsem neavtoriziranim uporabnikom.

Ključne besede: omrežje, internet, požarni zid, SDP, SPA, varnost, OpenSPA, OpenSDP, skrite storitve.

Abstract

Title: Dynamic Access Control to Network Devices

Author: Gregor Robert Krmelj

Today's software is getting more complex by the number of lines of code as well as the number of included features. Due to the rising complexity of software and market demands to release new products, the number of new vulnerabilities is on the rise too. Current mechanisms to defend against unauthorized access are usually implemented in higher layers of the network stack.

Limiting access in the application layer is a common practice, while lower layers access rights are harder to implement. The problem becomes evident when an application is vulnerable and the mechanisms of authentication and authorization are threatened. This thesis proposes a method of user authentication and authorization which functions on the network layer by dynamically assigning firewall rules. This in turn facilitates deployment of dark network applications on the internet - applications which are accessible on the network layer only to authorized users.

Keywords: network, internet, firewall, SDP, SPA, security, OpenSPA, OpenSDP, hidden services.

Poglavje 1

Uvod

V današnjem svetu informacijske povezljivosti in hitrega razvoja programske opreme je precejšnja verjetnost, da v programski opremi obstajajo določene pomanjkljivosti, zaradi katerih oprema nima zadostne zaščite. Težava lahko nastane, ko storitve, ki vsebujejo določene ranljivosti pri mehanizmih zaščite dostopa, priključimo v omrežje, predvsem na internet. Možnosti zlorabe programske opreme, oz. stopnja ranljivost storitve, se premosorazmerno poveča z večjo povezljivostjo naprav.

Mehanizmi s katerimi branimo aplikacije pred neznanimi uporabniki so ponavadi implementirane v višjih slojih sklada IP. Ponavadi določimo avtorizacijo na aplikacijskem sloju, ker z obstoječimi mehanizmi težko dodelimo pravice uporabniku na nižjih slojih. To velja še posebej danes, ko uporabljamo številne različne omrežne povezljive naprave. V primeru, ko je aplikacija ranljiva in s tem njeni mehanizmi avtentikacije in avtorizacije, je ogrožena splošna varnost programske storitve.

Četudi imamo splošno gledano varno aplikacijo, to še ne pomeni, da smo tudi varni pred vsemi možnimi zlorabami oziroma zunanjimi napadi. Varnost lahko definiramo tudi kot skupek zaščitnih mehanizmov s katerimi v posameznih predvidenih situacijah v čim večji meri onemogočimo zunanje nezaželene vdore. Ko potrebno vloženo delo napadalca bistveno presega vrednost, ki jo napadalec lahko pridobi z zlorabo sistema, takrat velja, da je

aplikacija varna.

En sloj zaščite praviloma ne zadostuje za ponudbo varne programske storitve. Vsak poskus vdorov oz. zlorab s strani nepooblaščenih oseb je potrebno čim prej zaznati in jim pri tem preprečiti dostop. Požarni zidovi sicer lahko služijo kot prvi obrambni mehanizem pred nepooblaščenimi uporabniki. Slabost požarnih zidov pa je statična določitev dovoljenih omrežnih naprav. Današnje naprave pogosto nimajo fiksnega omrežnega naslova. Posledično požarni zidovi danes ne omogočajo enostavnega načina omejevanj omrežnega prometa samo določenim uporabnikom.

1.1 Pregled področja

V diplomskem delu smo se osredotočili na odprtokodne projekte. Ti so nam omogočili pregled podrobnosti njihovega delovanja in možnosti analize slabostih. Nekateri koncepti, predstavljeni v delu, niso v široki uporabi, kar pomeni, da razpolagamo s pomanjkljivim znanjem o njihovih implementacijah, in sicer tako odprtokodnih kot tudi komercialnih.

Koncept dinamičnih požarnih zidov za potrebe dostopa je znan pod imenom trkanje na vrata (angl. *port knocking*) in preverjanje pristnosti z enim paketom (angl. *single packet authorization*). Koncepta nista povsem nova in obstajajo številne implementacije. V diplomskem delu predstavimo, zakaj je preverjanje pristnosti z enim paketom boljši pristop, in opišemo projekt fwknop, ki je referenčna implementacija koncepta.

Programsko določen rob omrežja (angl. *software defined perimeter*) predstavlja nadgradnjo dinamičnega dostopa. Koncept je še vedno v fazi razvoja delovne skupine organizacije CSA (*Cloud security alliance*), kar pomeni, da ni na voljo veliko implementacij. Poleg nekaterih komercialnih izdelkov, katerih podrobnosti implementacije ni možno analizirati, obstaja odprtokodni projekt podjetja Waverley Labs. Waverley Labs razvija odprtokodno implementacijo programsko določenega roba omrežja, kot tudi komercialno različico. V diplomskem delu opišemo, kako deluje odprtokodna različica, kot

tudi specifikacijo programsko določenega roba omrežja od CSA.

1.2 Cilj in struktura diplomskega dela

Cilj diplomskega dela je začrtati razvoj takšne programske opreme, ki bi uporabnikom dinamično omejevala dostop do storitev na omrežnem sloju.

Uvodni del diplomske naloge je posvečen delovanju požarnih zidov, s poudarkom na nekaterih njihovih ključnih pomanjkljivostih. V nadaljevanju je podan opis dveh možnih tehnik dinamičnega dodeljevanja dostopa. Uvodno poglavje se zaključi s predstavitvijo programske opreme, ki uporablja preverjanje pristnosti z enim paketom. Pri tem so podane prednosti ter slabosti obravnavane metode.

Naslednje poglavje, pod naslovom *'Lastna implementacija preverjanja pristnosti z enim paketom: OpenSPA'* natančneje predstavi načrtovan protokol OpenSPA. Protokol omogoča preverjanje pristnosti uporabnika z uporabo enega paketa na način, ki ne razkrije obstoja strežnika. Sledi opis programske implementacije in možne izboljšave.

Četrto poglavje, z naslovom *'Programsko določen rob omrežja'* podrobneje opisuje programsko opremo, ki poenostavi delovanje preverjanja pristnosti uporabnikov z enim paketom. V poglavju je predstavljena specifikacija, ki opisuje delovanje takšnega sistema. V nadaljevanju je podan opis lastne odprte kodne implementacije, ki nadgradi predstavljeno implementacijo preverjanja pristnosti z enim paketom. Poglavje se zaključi z analizo možnih izboljšav.

Diplomska naloga se zaključi s primerom uporabe, ki združuje razvito programsko opremo in demonstrira njihovo uporabnost. Primer ponazarja, kako dva uporabnika, ki imata različne pravice do storitev, uporabljata razvito programsko opremo. Kot primer neavtoriziranega uporabnika je predstavljen napadalec, ki neuspešno poskuša dostopati do zaščitenih storitev.

Poglavje 2

Dinamična dodelitev dostopa z uporabo požarnega zidu

Požarni zidovi so ena izmed najbolj osnovnih omrežnih komponent za gradnjo varnih omrežij vseh velikosti. Konfiguracija požarnih zidov ponavadi poteka z dodajanjem pravil, ki dovolijo ali zavrnejo omrežni promet [10].

2.1 Konfiguracija požarnih zidov dandanes

Glavna naloga požarnega zidu je odobritev omrežnega prometa, ki se pretaka po omrežju. Ponavadi konfiguracija poteka tako, da omrežni administrator nastavi pravila. Pravila se navezujejo na značilnosti prometa, kot so:

- Izvorni naslov IP
- Ciljni naslov IP
- Izvorna vrata
- Ciljna vrata
- Protokol (npr. TCP, UDP, ICMP, itd.)
- Tip paketa (npr. pri protokolu ICMP: *Echo reply* [15])

- Stanje povezave (npr. pri protokolu TCP: *ESTABLISHED* [15])

Z uporabo pravil je konfiguracijo možno izpeljati na dva načina: po principu bele liste (angl. *whitelist*) ali črne liste (angl. *blacklist*) [13]. Razliko med obema načinoma predstavlja privzeta omrežna politika za promet, ki se ne ujema z nastavljenimi pravili. Če je požarni zid konfiguriran po principu bele liste, potem je ves promet blokiran, razen tistega prometa, ki je eksplicitno dovoljen po pravilih bele liste. Pri črni listi je ves promet dovoljen, razen tistega, ki je eksplicitno blokiran. Konfiguracija požarnega zidu po principu bele liste predstavlja boljšo prakso kot črne liste. Razlog je predvsem zaradi lažje definicije dovoljenega prometa z manjšim številom pravil, v primerjavi s črno listo.

2.2 Dinamična konfiguracija

Tipična konfiguracija požarnega zidu je statična nastavitev pravil, ki odraža omrežno politiko, kot na primer, kdo ima dovoljenje komunicirati z različnimi storitvami na omrežju.

Konfiguracija požarnega zidu z metodo bele liste je primerna za sistem, ki bi dodelil omrežni dostop samo tistim uporabnikom, ki imajo dostop do storitve, in le takrat, ko ga potrebujejo. Ustvarili bi lahko spletno storitev, kjer bi uporabniki lahko zahtevali dostop do želenih storitev preko grafičnega vmesnika. Takšen sistem je zaradi možnosti konfiguracije požarnega zidu zelo vabljiv za poskuse nepooblaščenega posega oziroma vdora. Če je takšna storitev izpostavljena na internetu, in s tem dostopna kjerkoli na svetu, lahko postane lahka tarča vdorov. Zato je treba takšno storitev zakriti pred morebitnimi napadalci. Izziv razvoja takšnega sistema je ravno v zakrivanju takšne storitve.

Potreben je mehanizem, da se naprava omrežno skrije pred nepooblaščenimi uporabniki in da je še vedno dostopna vsem pooblaščenim uporabnikom. Tekom časa sta se razvila dva postopka kot možni rešitvi. Prvi postopek se imenuje trkanje na vrata (angl. *port knocking*). Ta postopek se je zaradi

strogih podatkovnih omejitev in varnostnih pomanjkljivostih upokojil [16]. Nadomestil ga je drugi postopek, ki se imenuje preverjanje pristnosti z enim paketom (angl. *single packet authorization*).

Tovrstni varnostni ukrepi, ki uporabljajo dinamična pravila v požarnem zidu, ne zagotavljajo popolne varnosti. Njihova uporaba pripomore k varnosti tako, da zmanjša površino možnih napadov, in to še preden takšni napadi pridejo do same aplikacije. Souporaba drugih varnostnih mehanizmov je predpogoj za varno aplikacijo. Dinamična uporaba požarnega zidu velja kot prvi varnostni mehanizem pri obrambi pred možnimi vdori.

2.3 Trkanje na vrata

Osnovni cilj postopka trkanja na vrata je posredovati informacijo o avtentikaciji. To storimo s trkanjem na vrata - vnaprej znanim zaporedjem paketov poslanih na določena omrežna vrata. Če je trkanje na vrata uspešno, strežnik odpre vrata - dostop do storitve odjemalcu. Od začetka zasnove leta 2003 se je razvilo skoraj 30 znanih implementacij trkanj na vrata [19]. Razlike med implementacijami so v tem, kako le-te dosežejo avtentikacijo.

V splošnem lahko razdelimo implementacije v tri kategorije. Najbolj preprost način je klasično trkanje na vrata (angl. *plain-text port knocking*). Princip klasičnega trkanja na vrata trdi, da je (skrivnostno) zaporedje trkanj na vrata dovolj, da avtoriziramo uporabnika [11]. Takšen način je sicer možno trivialno zlorabiti, saj z enostavnim zajetjem prometa napadalec lahko izvrši (skrivnostno) trkanje, ki posledično pomeni dostop do zaščitene storitve.

Drugi način se imenuje kriptografsko trkanje na vrata (angl. *cryptographic port knocking*). Implementacije, ki spadajo v to kategorijo šifrirajo z uporabo simetričnega ključa; podatke o naslovu IP od odjemalca in omrežnih vrat do katerih želijo dostopati [11]. Šifrirano sporočilo pošljejo strežniku, tako da pošljejo pakete na omrežna vrata, ki predstavljajo binarno vrednost sporočila. Dejstvo je, da je večina implementacij tovrstne kategorije ranljiva na napade s ponovitvijo seje [11].

Najbolj varen način implementacije trkanj na vrata se imenuje enkratno trkanje na vrata (angl. *one-time port knocking*). Seja enega trkanja velja samo enkrat. Ponavadi se to doseže z uporabo seznama gesel, gesel generiranih iz glavnega ključa ali z dodajanjem časovne oznake kriptografskega trkanja na vrata [11].

2.3.1 Omejitve

Trkanje na vrata je zanimiv pristop prikritega odpiranja omrežnih vrat. Na žalost, je ta pristop zgrajen na slabih temeljih, ki predstavljajo slabost tega sistema. Najbolj očitna težava so napadi s ponovitvijo seje. Pri klasičnem in kriptografskem trkanju na vrata takšen napad povzroči odpiranje vrat nepooblaščeni osebi [19].

V glavi paketa, ki se uporablja za trkanje na vrata je na voljo samo 2 bajta informacij [19]. Tudi princip zaporedja trkanj je problematičen, saj paketi lahko pridejo do strežnika po različnih poteh. Da se temu problemu izognemo, je potrebno vstaviti zakasnitve [20]. Posledično omejena količina podatkov v paketu in zakasnitve povzročajo, da ni možno pošiljati večjih količin podatkov s katerimi bi lahko uporabili boljše varnostne mehanizme.

Ker odjemalec pošilja pakete na različna vrata v zelo kratkem obdobju je za zunanje opazovalce podobno skeniranjem vrat [19]. Podobnost postane problematična v omrežjih, kjer je postavljen sistem za preprečevanje ali odkrivanje vdorov (angl. *intrusion detection/prevention system*). Takšni sistemi imajo možnost, da ob odkritju skeniranj vrat blokirajo omrežno napravo iz katere izhaja promet. Uporaba trkanj na vrata je v takšnih omrežjih nemogoča, saj je trkanje obravnavano kot omrežni napad in povzroči, da sistem za preprečevanje vdorov blokira upravičenega uporabnika.

Metoda trkanja na vrata je lahko ranljiva na napade zaporedja trkanj (angl. *sequence busting attacks*) [19]. Takšen napad izkoristi slabost, da je edini podatek o uporabniku v glavi IP. Da napadalec lahko izvede napad, mora vriniti ponarejena trkanja, medtem ko odjemalec izvaja trkanje. Za kreiranje ponarejenega trkanja je potrebno zamenjati izvorni naslov IP

z naslovom odjemalca. Napadalec lahko kontinuirano pošilja neveljavna trkanja in tako izvaja ohromitev storitve (angl. *Denial of Service, DoS*). Zanimivo je, da lahko pride do enakega (nenamernega) učinka tudi v omrežjih NAT. Ko več uporabnikov deluje na omrežju NAT in želijo hkrati izvesti trkanje na vrata pri zunanjem strežniku. Strežnik ni zmožen ločevati paketov med uporabniki - zaradi omrežja NAT imajo v glavi IP vsi enak izvorni naslov. Končni rezultat so seveda neuspešna trkanja in nedostopnost storitve za vse uporabnike.

Zaradi omenjenih slabosti se je razvil drugi način implementacije dinamičnega dodajanja pravil v požarni zid, to je preverjanje pristnosti z enim paketom.

2.4 Preverjanje pristnosti z enim paketom (SPA)

Single Packet Authentication (SPA) je metoda, s katero uporabnik dokaže svojo identiteto tako, da pošlje samo en paket [11]. Za razliko od trkanja na vrata je celotno sporočilo zapakirano v podatkovno polje enega paketa. Paket SPA se torej pošlje le enkrat in to na točno določena vrata. Takšen pristop reši omejitve trkanja na vrata, saj omogoča pošiljanje večje količine podatkov, brez zakasnitev in prepreči napade, ki izkoriščajo ranljivosti na zaporedje trkanj.

SPA je zgrajen po principu odjemalec-strežnik [20]. SPA odjemalec pošlje en šifriran paket, ki vsebuje uporabniški naslov IP, enkratno kriptografsko število in vrata do katerih želi dostopati. Odjemalec pošlje paket SPA preko protokola IP do strežnika, ki ga obdela. V primeru, da je SPA paket veljaven in da ima uporabnik dostop do storitve, se vrata odprejo z dodajanjem izjeme v požarni zid za uporabnikov naslov IP in zahtevana vrata. V nasprotnem primeru SPA strežnik ignorira zahtevo. Odjemalec nikdar ne prejme odgovora. Edina posledica postopka SPA je, da se v primeru uspešne avtentikacije odprejo omrežna vrata.

2.4.1 Omejitve in možne rešitve

Čeprav SPA reši težave trkanja na vrata, obstajajo še vedno omejitve. Največja težava so omrežja NAT [19]. Paket SPA mora vsebovati javni naslov IP od odjemalca, saj na podlagi tega naredi izjemo v požarnem zidu. Ker je javni naslov IP od omrežja NAT pomeni, da izjema v požarnem zidu velja za vse naprave, ki so del omrežja NAT. Čeprav je NAT iz vidika nevtralnosti interneta velika slabost, saj krši princip končne povezljivosti (angl. *End-to-end principle*), je zaradi pomanjkljivosti naslovov IPv4 njegova uporaba običajna doma in tudi v podjetjih. Mehanizmi, ki poskušajo rešiti težavo, jo v celoti ne rešijo, le blažijo.

Ena od možnih rešitev je uporaba protokola IPv6 namesto IPv4. Implementacija rešitve je otežena zaradi manjše uporabe protokola IPv6, čeprav v celoti reši težavo NAT. Po mnenju nekaterih težava nastopi samo v tistih primerih, kjer se uporablja SPA preko interneta. V internih omrežjih SPA nima teh težav, ampak zaradi današnje globalne povezljivosti je delovanje SPA preko interneta pričakovano.

Kot drugo rešitev lahko v paketu SPA predložimo podatek, da je odjemalec za omrežjem NAT. Na ta način sporočimo strežniku SPA o prisotnosti omrežja NAT. Ta lahko potem odloča na podlagi omrežne politike, če je promet, ki izhaja iz omrežja NAT, dovoljen. Uporaba naslova NAT za ustvarjanje izjeme v požarnem zidu pravzaprav ni težava. Težava je, ko strežnik SPA nima podatka o tem, da je odjemalec za omrežjem NAT. To lahko vodi do tega, da strežnik dovoli promet, kateri ni skladen z omrežno politiko.

Zadnja rešitev uporablja požarni zid s podporo stanja (angl. *stateful firewall*). Takšen požarni zid hrani podatke o različnih povezavah in na podlagi le-teh lahko ustvarjamo požarna pravila. Rešitev je primerna za povezave TCP. Dodamo pravilo v požarni zid, ki dovoljuje vse povezave, ki so v TCP stanju povezane (angl. *TCP ESTABLISHED state*) [19]. Nato za zelo kratek čas, to je nekaj sekund, dovolimo povezavo odjemalcu. Toliko, da lahko odjemalec sklene povezavo TCP. Ko se uporabniku izteče dostop, ni možno več skleniti nove povezave. Obstoječa povezava, je sicer še vedno dovoljena.

Ta način delovanja poskrbi, da je možno izkoristiti omejitve naslova NAT v požarnem zidu le za nekaj sekund, v času inicializacije povezave. Ker SPA pravzaprav ne zagotavlja omrežne varnosti, bi praviloma morali biti prisotni še drugi varnostni mehanizmi. Tovrstna rešitev je varnostno sprejemljiva za večino primerov.

Druga težava, ki se lahko pojavi, so omrežja, kjer uporaba UDP protokola ni dovoljena. V takšnih primerih je potrebno uporabiti posredniški strežnik (angl. *proxy*).

2.4.2 Fwknop

Fwknop je projekt, ki je najbolj aktiven na področju SPA. Čeprav so začetki fwknop projekta predstavljali implementacijo trkanja na vrata, se je projekt razvil v implementacijo SPA dobro leto kasneje, maja 2005 [19].

Potek avtorizacije

Odjemalec ustvari paket IP v katerega zapiše potrebne podatke kot so: uporabniško ime, način dostopa (odpiranje vrat ali ukaz) in podatke o zahtevanem dostopu (številka vrat in protokol). Podatki so kriptografsko šifrirani s simetričnim (AES) ali asimetričnim (z uporabo GnuPG) ključem in poslani strežniku na vnaprej znana vrata UDP (privzeto 62201) [19]. Strežnik pridobi paket, dešifrira in preveri, če se zgoščena vrednost paketa ujema s prejšnjimi paketi[1]. Če je paket veljaven in ima uporabnik dovoljenje, se v požarni zid doda izjema.

Podrobnosti implementacije

Paket SPA v fwknop vsebuje sledeče podatke [5, 20]:

- fwknop različica

[1]Strežnik hrani seznam zgoščenih vrednosti vseh prejetih paketov. V primeru, ko dobi paket, ki ima enako zgoščeno vrednost kot v seznamu, lahko zavrže paket - možen je napad s ponovitvijo seje.

- časovna oznaka kreiranje paketa
- uporabniško ime uporabnika
- način dostopa (dostop ali ukaz)
- podatki o dostopu (število vrat ali ukaz)
- 16 bajtov naključnih podatkov (enkratno kriptografsko število)
- SHA-256 zgoščena vrednost paketa

Fwknop podpira pridobitev paketov fwknop s poslušanjem na vrata UDP ali pa s filtriranjem celotnega prometa za lastnosti SPA paketa z uporabo knjižnice *libpcap* [1].

Fwknop podpira tri različne požarne zidove: *iptables*, *ipfw* ali *PF*. Fwknop rešuje težavo NAT z uporabo omenjene rešitve kratkega dovoljenja povezav v kombinaciji s pravilom za dovoljenje že vzpostavljenih povezav. Projekt je še vedno aktiven in se v nekaterih primerih uporablja kot temeljni deljček drugih tehnologij.

Slabosti

Čeprav projekt fwknop velja kot primer dobre implementacije SPA, ima svoje slabosti. V obdobju nekaj let se je v omenjenem projektu nabralo nekaj t.im. tehničnega dolga (angl. *technical debt*). Celoten projekt je zgrajen pod predpostavko protokola IPv4. Prihodnost omrežij IP predstavlja IPv6, zato je njegova podpora nujna. Podpora za IPv6 pri fwknop se načrtuje že od leta 2011, a uporaba protokola je še vedno nepodprta [2]. Zanimivo je, da se je letos ponovno pojavilo povpraševanje za razvoj podpore IPv6. V času pisanja te diplomske naloge fwknop uradno še vedno ne podpira IPv6.

Največja slabost projekta fwknop je pomanjkanje zunanjega preverjanja avtentikacije in avtorizacije. Strežnik fwknop zahteva, da so uporabniški ključi in seznam dovoljenih storitev shranjeni v datoteki na strežniku. To predstavlja precejšnje težave, ko želimo podpreti večje število uporabnikov

- na primer v podjetjih. Pri večjem številu uporabnikov se tudi soočamo s težavami podvajanja uporabniških podatkov, zaradi pomanjkanje podpore uporabniških imenikov (Microsoft Active Directory, FreeIPA itd.). To povzroči, da moramo podvajati podatke iz obstoječih podatkovnih baz. Takšen način konfiguracije je nagnjen k napakam, kar lahko vodi do resnejših posledic za uporabnika, vključno s popolno nedostopnostjo storitve.

Princip avtorizacije je tudi zelo omejen pri fwknop. V konfiguracijo datoteke lahko dodamo samo uporabniške pravice do storitve. Konfiguracija bolj naprednih pravil, ki izražajo omrežno politiko, niso možne - kot na primer določitev avtorizacije na podlagi karakteristike odjemalčevega naslova IP (dostop iz nedovoljene države, nedovoljenega omrežja itd.).

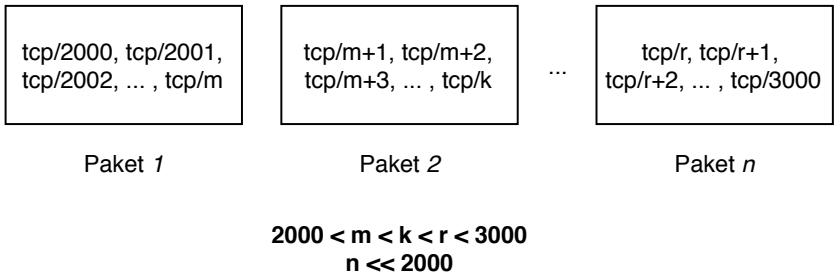
Določene omrežne aplikacije (P2P, multimedijske storitve itd.) potrebujejo razpon omrežnih vrat za komunikacijo. Fwknop paket ne podpira zahtevanje razpona vrat, le ekspliciten seznam omrežnih vrat. To povzroči, v primeru zahteve za obsežen razpon omrežnih vrat, množico paketov (slika: 2.1). Poleg nepotrebnega dodatnega procesiranja paketov in dodatnega omrežnega prometa je konfiguracija pravil v požarnem zidu dokaj neučinkovita. V najslabšem primeru, ko uporabnik zahteva celoten možen razpon vrat, je posledica dodajanje 65,535[2] pravil v požarni zid, namesto enega[3]. Požarni zidovi ponavadi uporabljajo linearno ujemanje paketov[4] s pravili [14]. To povzroči izrazito zmanjšanje zmogljivosti požarnega zidu pri filtriranju omrežnega prometa. Z večjim številom uporabnikov, ki zahtevajo takšen dostop, je upad zmogljivosti še toliko bolj očiten.

V pogovoru avtorja te diplomske naloge z Michaelom Rashem, ustvarjalcem in glavnim vzdrževalcem projekta fwknop, smo izvedeli, da se za projekt ne obeta razvoj novih funkcionalnosti. Zaradi omenjenih slabostih in večje fleksibilnosti pri razvoju novih funkcionalnosti smo se odločili za lastno implementacijo SPA.

[2]Celoten razpon omrežnih vrat je od 1 do $2^{16} - 1 = 1$–$65,535$.

[3]Požarni zidovi omogočajo določitev razpon vrat v enem samem pravilu, npr. tcp/2000-3000.

[4]Po vrsti primerja ujemanje paketa z vsakim pravilom.



Slika 2.1: Želimo poslati zahtevo za dostop do omrežnih vrat tcp/2000-3000. Ker je zapis seznama vseh vrat večji od prostora v paketu IP, je treba razdeliti seznam na več paketov.

2.5 Varnost zaradi nepoznavanja?

V razpravi o varnosti pogosto naletimo na princip varnost zaradi nepoznavanja (angl. *security through obscurity*). Ta princip pravi, da skrivanje podatkov o sistemu (arhitekturi ali načinu delovanja) ni mehanizem za zagotavljanje varnosti, še zlasti če je to edini mehanizem. V idealnem svetu bi bile aplikacije varne pred napadi in ne bi potrebovali tovrstnih mehanizmov za omejevanje dostopa. Današnji najboljši pristopi obravnavajo varnost kot sloje, kjer vsak dodaten sloj oteži delo napadalca. Naloga dinamičnih pravil v požarnem zidu ni, da zagotovi celotno varnost sistema, ampak da brani pred večino napadov, ki bi jih sicer omrežje posredovalo aplikaciji. Tovrstni princip se imenuje obramba po globini (angl. *defense in depth*), ki se je uporabljal že v času gradnje vojaških trdnjav [22]. Več kot imamo varnostnih mehanizmov, večjo težavo to predstavlja napadalcu. SPA lahko opravi to zelo učinkovito, četudi dovoli ostalim napravam za omrežje NAT dostop. Pravilna uporaba SPA je v kombinaciji z ostalimi varnostnimi ukrepi kot so: VPN, digitalna potrdila, šifrirane povezave, uporabniška gesla, enkratna gesla in dva ali večfaktorska avtentikacija.

Poglavje 3

Lastna implementacija preverjanja pristnosti z enim paketom: OpenSPA

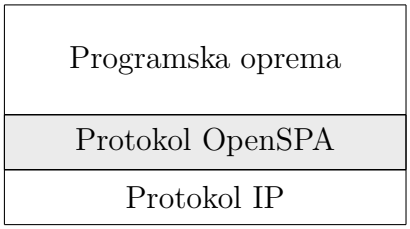
Fwknop ne omogoča prilagodljivih postopkov avtentikacije in avtorizacije. Zaradi te pomanjkljivosti smo se odločili za izdelavo lastne programske opreme z imenom OpenSPA, ki omogoča večjo prilagodljivost. Poleg tega izdelava te nove programske oprema omogoča uporabo v omrežjih IPv6 in zahtevanje dostopa do razpona omrežnih vrat.

Vizija protokola OpenSPA (slika: 3.1) je, da se uporabi kot sestavni del programske opreme. V ta namen smo razvili posebno knjižnico - *openspalib* - v programskem jeziku Golang, ki implementira specifikacijo OpenSPA in istočasno olajšala integracijo protokola OpenSPA s programsko opremo.

3.1 Tipičen potek protokola

Sledi (slika: 3.2) zelo kratek, visokonivojski pregled, kakšen je tipičen potek protokola:

1. Odjemalec ustvari zahtevo, v katero doda svoj javni naslov IP, ID naprave, protokol in omrežna vrata (od storitve) do katere želi dostopati.



Slika 3.1: Vizija za protokol OpenSPA je, da se uporabi kot sestavni del programske opreme.

Celoten paket podpiše in šifrira, predno ga pošlje strežniku OpenSPA z uporabo okvira UDP.

2. Strežnik prejme paket, ga dešifrira in preveri podpis. Če je podpis veljaven, preveri če je uporabnik avtoriziran na podlagi svoje zahteve. V primeru da je, sledi dodajanje izjeme v požarni zid. V nasprotnem primeru strežnik zavrne paket in ne odgovori odjemalcu, da je prišlo do napake.

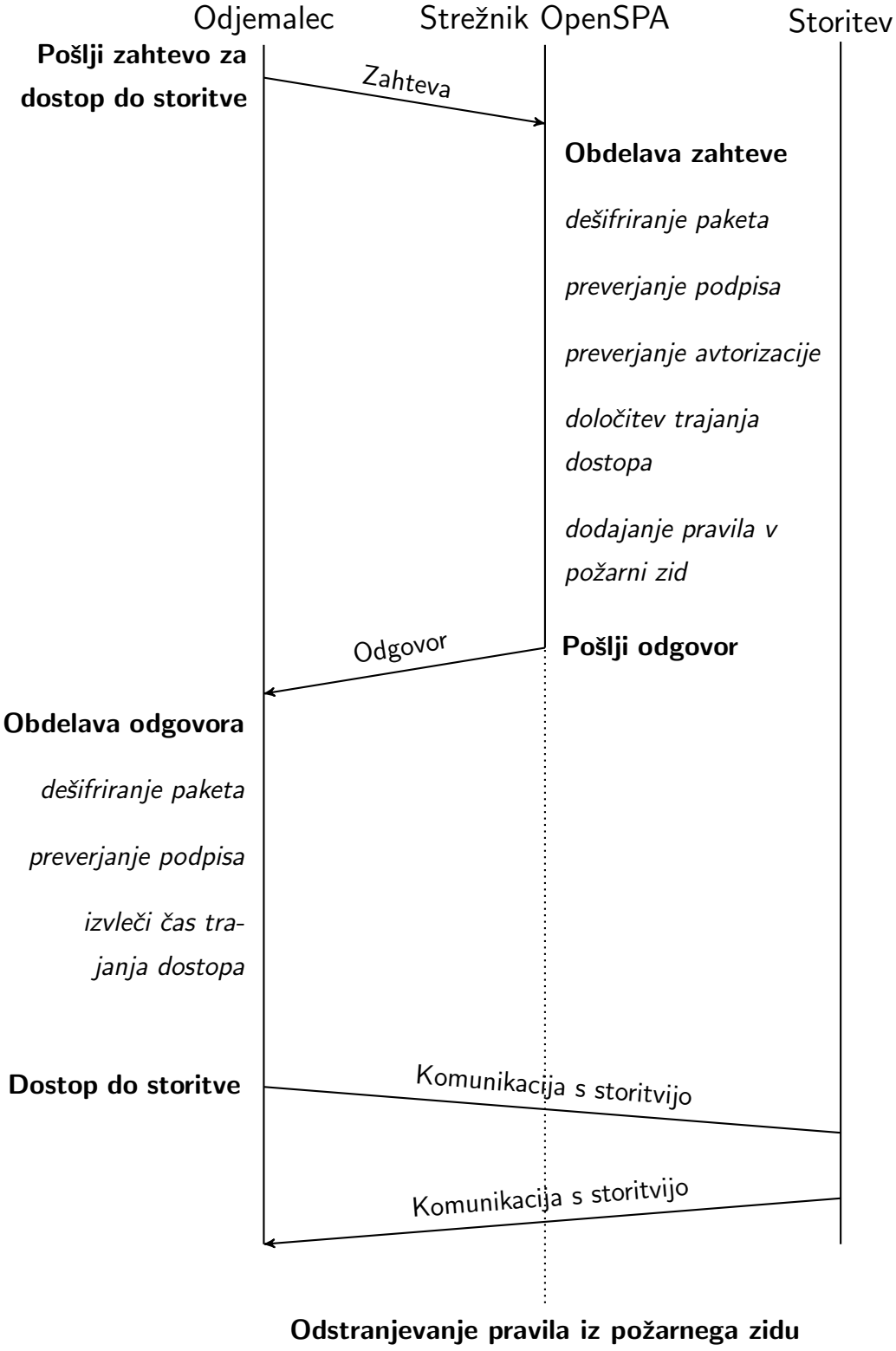
3. Strežnik kreira odgovor v katerem zapiše za koliko časa ima odjemalec dostop do storitve. Paket podpiše in šifrira, predno ga pošlje odjemalcu.

4. Odjemalec prejme odgovor, dešifrira vsebino in preveri podpis. Odjemalec se zdaj lahko poveže na storitev za toliko časa kot je zapisano v paketu.

5. Ko se čas dostopa izteče, strežnik odstrani pravilo iz požarnega zidu.

3.2 Načrtovani protokol

Pri načrtovanju in implementaciji protokola OpenSPA smo si zastavili cilj odpravo pomanjkljivosti od projekta fwknop. Izdelali smo popolnoma binarno kodiranje paketov, saj je velikost paketov ena izmed pomembnih omejitev. Pri fwknop je moč opaziti slabosti tesnega sklopa postopka, avtentikacije in avtorizacije s programsko opremo strežnika. Zaradi tega je bilo

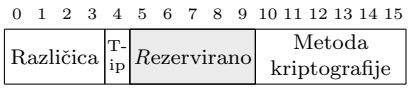


Slika 3.2: Primer tipičnega poteka protokola.

priporočljivo implementirati razširitvene skripte (angl. *extension scripts*). Le-te omogočajo kompletno programsko svobodo pri določitvi postopkov avtentikacije in avtorizacije. V paket tip zahteva smo tudi dodali možnost določitve razpona omrežnih vrat. V primerjavi s fwknop to omogoča ustvarjanje bolj učinkovitih pravil v požarnem zidu in s tem hitrejšo obdelavo paketov. Podprli smo tudi protokol IPv6, kar je vplivalo na zasnovo kodiranja paketov - zaradi daljših naslovov IPv6 je bilo potrebno povečati velikost paketa.

Dodali smo tudi paket tipa odgovor, ki ga strežnik pošlje v primeru uspešne avtorizacije. Tovrstna zasnova protokola se ne pojavi pri ostalih implementacijah trkanja na vrata ali SPA. Zaradi odločitve ločevanja implementacije požarnega zidu od strežnika izgubimo podatek o tem, koliko časa ima uporabnik na voljo, da dostopa do storitve. Paket tipa odgovor reši to težavo in omogoča dinamično določitev dostopa različnim uporabnikom in storitvam. Poleg tega ima odjemalec potrebne podatke, da ve, kdaj mora ponovno poslati zahtevo. Ta način omogoča odjemalcu komunikacijo brez prekinitev, saj paket tipa odgovor jasno definira, koliko časa bo odjemalec imel dostop. Paket tipa odgovor olajša delo z razhroščevanjem. Če storitev, ki je zaščitena z OpenSPA ne deluje, lahko sklepamo, da gre za aplikacijsko napako v primeru, ko dobimo od strežnika OpenSPA odgovor.

Poleg ločitve požarnega zidu od protokola smo želeli čim bolj ločiti tudi kriptografske mehanizme protokola. Dodali smo dve polji, kateri določata vrsto šifriranja in vrsto podpisa. Polji sta dovolj veliki, da podpirata mnogo različnih vrst kriptografije. Polje, ki določa šifiriranje, ima prostor za $2^6 = 64$ različnih metod. Polje, ki določa podpis, pa $2^8 = 256$ različnih metod. Na ta način protokol ni vezan na samo eno kriptografsko metodo, ampak lahko dodajamo novejše in bolj zmogljivejše pristope - kot so na primer digitalna potrdila.



Slika 3.3: Glava OpenSPA paketa

3.3 Protokol OpenSPA

Protokol OpenSPA je sestavljen iz dveh paketov, to je zahteve in odgovora. Omejitev velikosti celotnega paketa znaša 1232 bajtov[1].

3.3.1 Glava

Glava paketa (slika: 3.3) je identična za oba tipa paketa, zahteva in odgovor. Vsebuje naslednja polja:

- *Različica protokola (4 biti)*: Definira, katera različica protokola se uporablja.
- *Tip paketa (1 bit)*: Definira, kakšen tip paketa je v podatkovnem delu (0=zahteva/1=odgovor).
- *Rezervirano (5 bitov)*: Polje, ki je rezervirano za prihodnje funkcionalnosti.
- *Metoda kriptografije (6 bitov)*: Metoda, s katero je podatkovno polje zašifrirano.

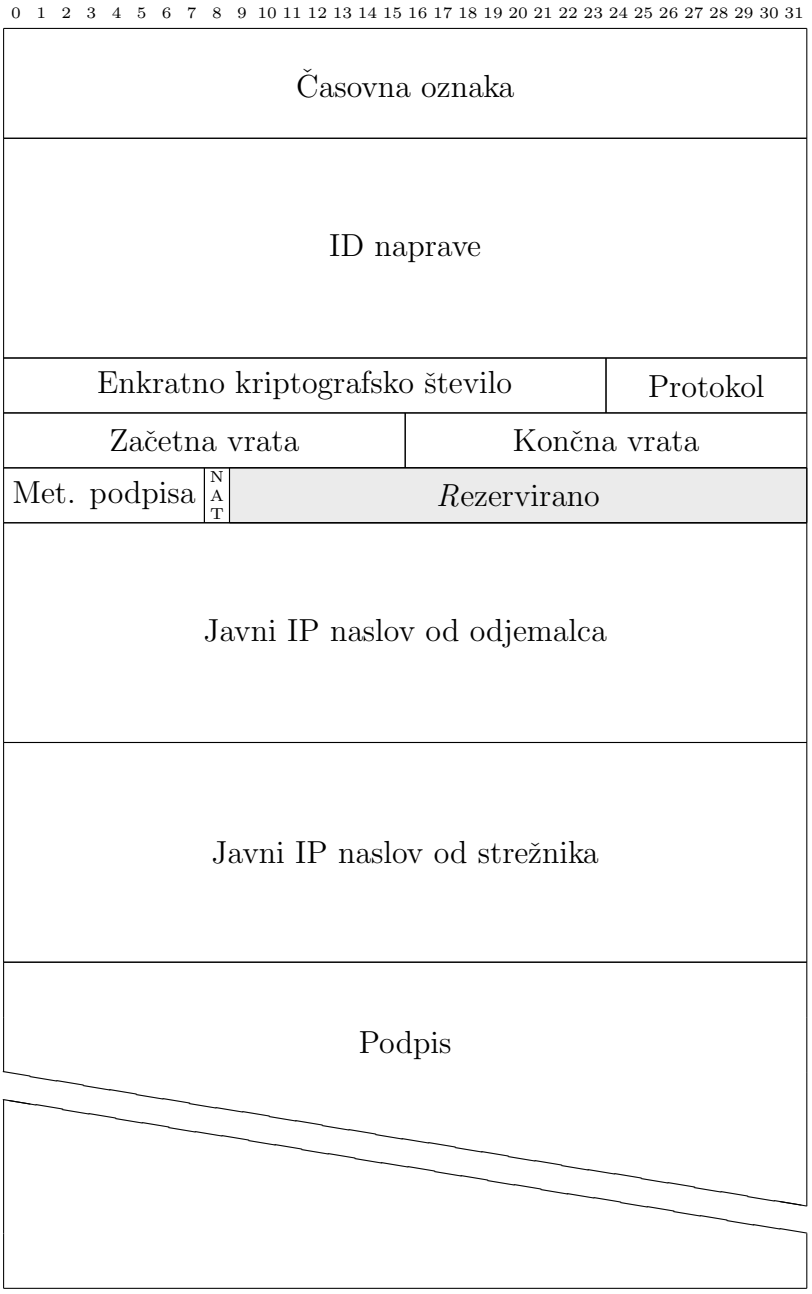
Glava OpenSPA ima velikost 2 bajta in se doda kot predpona podatkovnemu paketu, ki vsebuje zahtevo ali odgovor. Kot pričakovano se glava paketa ne šifrira, kajti vsebuje podatke o tem, katera metoda šifriranja se je uporabila pri podatkovnem delu paketa in vsebuje tudi različico protokola. S poljem različica protokola je možno ohraniti vzvratno združljivost in dodajati nova polja, ki razširijo funkcionalnost protokola.

[1]Podrobnosti, kako smo prišli do te omejitve, so razložene v podpoglavju 3.4.

3.3.2 Zahteva

Sledi seznam polj v podatkovnem delu paketa tipa zahteva (slika: 3.4).

- *Časovna oznaka (8 bajtov)*: UNIX 64-bitna časovna oznaka (angl. *timestamp*), ki označuje, kdaj je bil paket kreiran.
- *ID naprave (16 bajtov)*: UUID naprave.
- *Enkratno kriptografsko število (3 bajti)*: Naključno število, ki se uporablja za obrambo pred napadi s ponovitvijo seje.
- *Protokol (1 bajt)*: Protokol do katerega želi odjemalec dostop (npr. TCP, UDP, ICMP itd.). Število je definirano s strani IANA pod specifikacijo *Assigned Internet Protocol Numbers* [12].
- *Začetna vrata (2 bajta)*: Vrata do katerih odjemalec želi dostop.
- *Končna vrata (2 bajta)*: V primeru, da odjemalec želi dostop do razpona vrat, definira tu končna vrata, drugače pa ponovno definira *začetna vrata*.
- *Metoda podpisa (1 bajt)*: Metoda, s katero je bil paket podpisan.
- *Zastavica NAT (1 bit)*: Zastavica katero jo mora odjemalec nastaviti, če zazna, da je za omrežjem NAT.
- *Rezervirano (23 bitov)*: Polje je rezervirano za možne razširitve v prihodnosti.
- *Javni IP naslov od odjemalca (16 bajtov)*: Javni IPv4 ali IPv6 naslov od odjemalca, ki bo dodan v požarni zid za dostop.
- *Javni IP naslov od strežnika (16 bajtov)*: Javni IPv4 ali IPv6 naslov od strežnika, do katerega želi odjemalec dostop.
- *Podpis (maks. 1162 bajtov)*: Uporabniški digitalen podpis paketa.



Slika 3.4: Podatkovni del šifriranega paketa OpenSPA tipa zahteva

Največja velikost OpenSPA paketa tipa zahteva je torej 1232 bajtov[2].

3.3.3 Odgovor

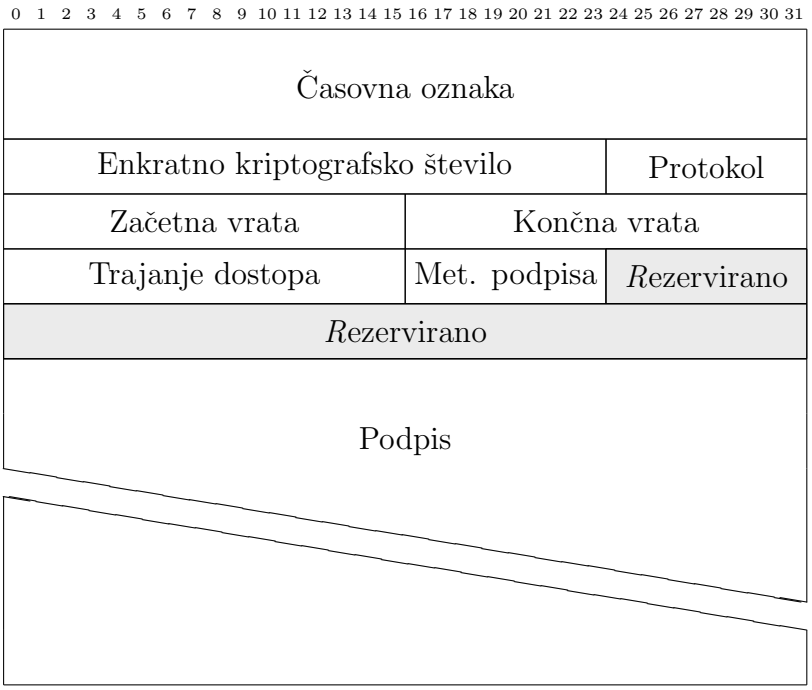
Sledi seznam polj v podatkovnem delu paketa tipa odgovor (slika: 3.5).

- *Časovna oznaka (8 bajtov)*: UNIX 64-bitna časovna oznaka (angl. *timestamp*), ki označuje, kdaj je bil paket kreiran.
- *Enkratno kriptografsko število (3 bajti)*: Naključno število, ki se uporablja za obrambo pred napadi s ponovitvijo seje.
- *Protokol (1 bajt)*: Protokol do katerega želi odjemalec dostop (npr. TCP, UDP, ICMP, itd.). Število je definirano s strani IANA pod specifikacijo *Assigned Internet Protocol Numbers* [12].
- *Začetna vrata (2 bajta)*: Vrata do katerih odjemalec želi dostop.
- *Končna vrata (2 bajta)*: V primeru, da odjemalec želi dostop do razpona vrat, definira tu končna vrata, drugače pa ponovno definira *začetna vrata*.
- *Trajanje dostopa (2 bajta)*: Za koliko sekund ima naprava dostop do omenjenih vrat.
- *Metoda podpisa (1 bajta)*: Metoda, s katero je bil paket podpisan.
- *Rezervirano (5 bajtov)*: Polje je rezervirano za možne razširitve v prihodnosti.
- *Podpis (maks. 1206 bajtov)*: Strežnikov digitalen podpis paketa.

Največja velikost OpenSPA paketa tipa odgovor je tudi 1232 bajtov[3].

[2]Glava (2 bajta) + podatkovni del paketa (1230 bajtov) = 1232 bajtov - naša zgornja omejitev celotnega paketa

[3]Poglejte opombo 2.



Slika 3.5: Podatkovni del šifriranega paketa OpenSPA tipa odgovor

3.4 Pojasnila o načrtovanem protokolu

3.4.1 Velikost paketa

Zgornja omejitev paketa je maksimalna velikost paketa na povezovalni plasti - MTU (angl. *maximum transmission unit*). V praksi je MTU pri Ethernet omrežjih omejen na 1500 bajtov. Ko imamo omrežne naprave, ki manipulirajo s paketi IP in dodajajo informacijo, se ta omejitev zmanjša. Empirično smo se odločili na omejitev paketov OpenSPA na 1280 bajtov. 1280 bajtov je omejitev na povezovalnem sloju, potrebujemo odšteti IPv4/IPv6 glavo, ki je 40 bajtov[4] in glavo UDP, ki zasede 8 bajtov. Na koncu pridemo do omejitve paketa OpenSPA na 1232 bajtov.

[4]Pravzaprav je glava IPv4 lahko velika od 20 do 60 bajtov odvisno od uporabe opcijskih polj. Ta se sicer pogosto ne uporabljajo [21]. IPv6 ima fiksno glavo 40 bajtov - z možnostjo razširitve.

3.4.2 Polje časovna oznaka in enkratno kriptografsko število

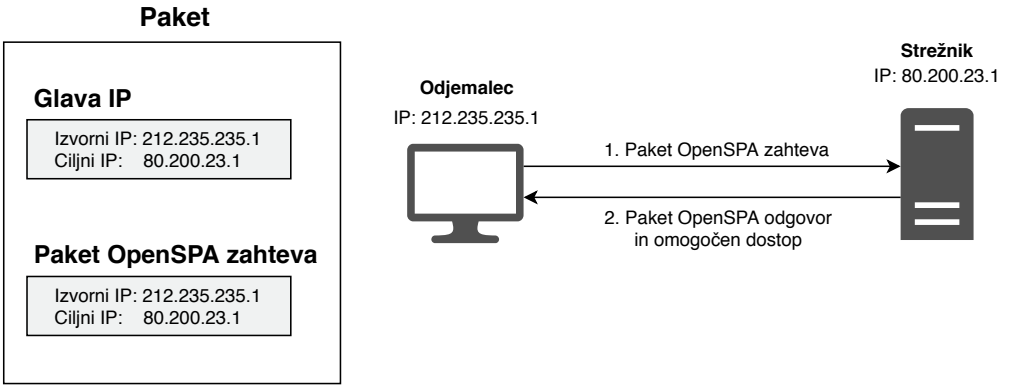
Polje, ki določa časovno oznako, omogoča strežniku filtriranje starih (neveljavnih) paketov. Dodali smo tudi polje, kjer vpišemo enkratno kriptografsko število. Kombinacija teh dveh polj omogoča obrambo pred napadi s ponovitvijo seje in filtriranje starih paketov. To velja za vse pakete, tudi tiste, ki jih ustvarimo v isti sekundi (časovna oznaka je natančna do ene sekunde). Strežnik mora shranjevati seznam zgoščenih paketov. Ob prejemu novega paketa je potrebno pregledati, če paket obstaja v seznamu. V primeru, da obstaja, zavržemo paket. Na ta način omogočimo ustreznejšo obrambo pred zunanjimi nepooblaščenimi vdori s ponovitvijo seje.

3.4.3 Naslov IP odjemalca in strežnika

Čeprav se na začetku morda ponuja sklep, da glava IP ponuja enake informacije kot podatki v podatkovnem delu paketa, ki se nanašajo na naslove IP, je to napačen sklep. Zaradi omrežnih naprav, ki manipulirajo z glavo IP, se naslovi v glavi lahko spremenijo. Polja naslov IP od odjemalca in strežnika omogočata varno in nedvoumno informacijo o zahtevanem dostopu (saj je paket digitalno podpisan). Poleg tega vpeljava dodatnih polj omogoča dodatne funkcionalnosti protokola. Sledijo štirje primeri o tem kakšno funkcionalnost lahko dosežemo.

Primer 1: En odjemalec, en strežnik

Slika 3.6 prikazuje klasičen primer, kjer imamo enega odjemalca in en strežnik. Strežnik je zaščiten s programsko opremo OpenSPA. Odjemalec pošlje zahtevo strežniku, strežnik odobri zahtevo in pošlje odgovor odjemalcu. Zdaj se lahko odjemalec poveže na strežnik, da uporabi storitev, katero je zahteval.



Slika 3.6: Primer, kjer odjemalec zahteva dostop do storitve na strežniku.

Primer 2: Dva odjemaleca, en strežnik

Slika 3.7 prikazuje primer, kjer odjemalec 1 pooblasti odjemalca 2 za dostop do storitve na strežniku, ki je zaščiten s programsko opremo OpenSPA.

Primer 3: En odjemalec, dva strežnika

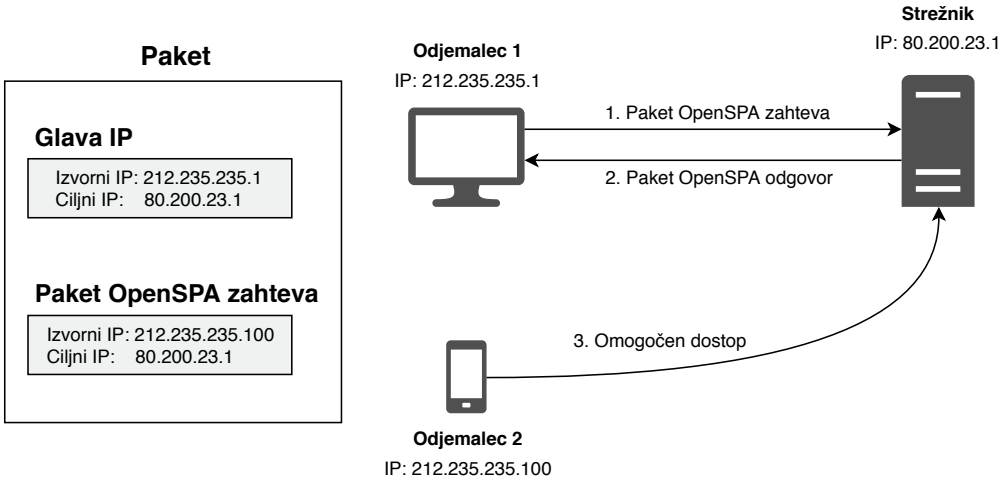
Slika 3.8 prikazuje primer, kjer odjemalec zahteva dostop do strežnika 2 in pošlje zahtevo strežniku 1 (kjer se nahaja strežnik OpenSPA).

Primer 4: Dva odjemalca, dva strežnika

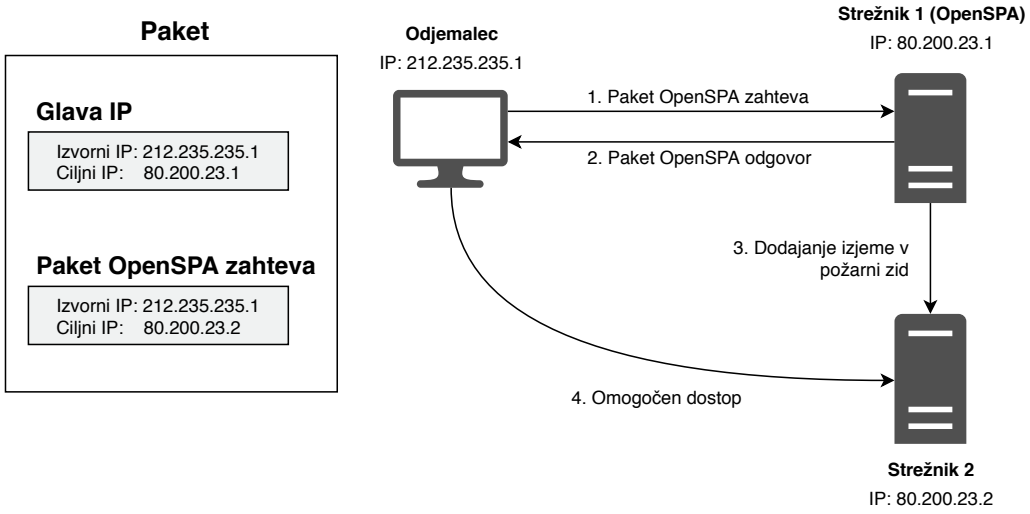
Slika 3.9 prikazuje primer, kjer odjemalec 1 pooblasti odjemalca 2, za dostop do strežnika 2, preko strežnika OpenSPA.

3.4.4 Polje začetna in končna vrata

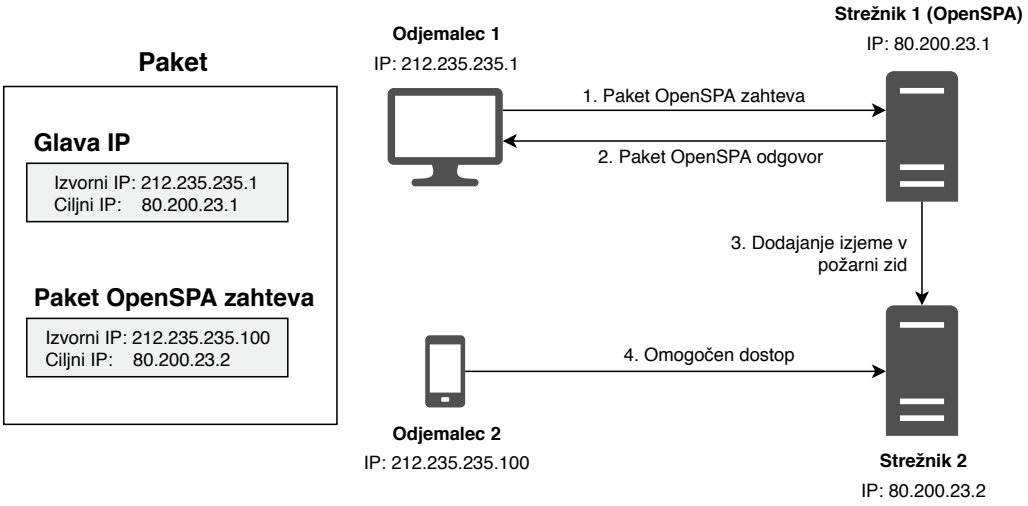
Namesto enega samega polja za število omrežnih vrat sta dodani dve. Na ta način je možno zahtevati razpon vrat do katerih bi želeli dostopati. Določene aplikacije predvsem intenzivne multimedijske, storitve za zabavo (večuporabniške igre) in P2P aplikacije potrebujejo razpon vrat za pravilno delovanje. V nasprotnem primeru bi bilo treba pošiljati številne pakete in hraniti komplek-



Slika 3.7: Primer, kjer odjemalec 1 zahteva dostop za odjemalca 2 do storitev na strežniku.



Slika 3.8: Primer, kjer odjemalec pošlje zahtevo strežniku 1 (strežnik OpenSPA) po dostopu do storitve na strežniku 2.



Slika 3.9: Primer, kjer odjemalec 1 pooblasti odjemalca 2, za dostop do strežnika 2 preko strežnika 1 (strežnik OpenSPA).

sno stanje za ponovno pošiljanje v primeru izgube paketa ali izteku časovnega dovoljenja.

3.4.5 Težava s preslikavo NAT

Kot je bilo že omenjeno, ena izmed težav SPA so primeri, kjer so odjemalci za omrežjem NAT. V podpoglavju *preverjanje pristnosti z enim paketom (SPA)*, v razdelku *omejitve in možne rešitve* so naštete možne rešitve problema NAT. Protokol OpenSPA je zasnovan tako, da podpira protokol IPv6 in hkrati vključuje zastavico NAT. Zastavico NAT vključi odjemalec, ko zazna, da se nahaja za omrežjem NAT. Strežniku je prepuščeno, če takšna zahteva dovoljena ali ne. Ker požarni zid ni tesno vezan na programsko opremo, je možno tudi konfigurirati požarni zid s pravili, ki upoštevajo stanje povezave. Ta način smo tudi predstavili v poglavju 2.4 kot možno rešitev.

3.5 Implementacija protokola OpenSPA

Za implementacijo protokola OpenSPA[5] je uporabljen programski jezik Go. Omenjeni jezik je sistemski programski jezik, ki ponuja bogato standardno knjižnico za programiranje omrežne programske opreme.

3.5.1 Razširitvene skripte

Eden izmed ciljev protokola je, da sta programska oprema protokola OpenSPA in mehanizmi avtentikacije in avtorizacije čim bolj ločena. Protokol ne definira mehanizma, kako to dosežemo. V ta namen smo razvili sistem razširitvenih skript (angl. *extension scripts, ES*). Razširitvene skripte so programi, ki so definirani v konfiguracijski datoteki strežnika. Skripte so lahko napisane v kateremkoli programskem jeziku, saj se izvajajo kot ločeni proces. Komunikacija s strežnikom poteka preko standardnega izhoda. Sledi kratek opis različnih razširitvenih skript.

Razširitvena skripta: uporabniški imenik

Vse metode šifriranja in podpisovanja so implementirane v programski opremi. Za določene metode potrebujemo zunanje podatke, kot je na primer javni ključ naprave. Za pridobitev tovrstnih podatkov priskrbi skripta *uporabniški imenik*. Ob klicu nanjo lahko z ustreznim pod ukazom *GET_USER_PUBLIC_KEY* od zahtevane naprave pridobimo javni ključ.

Razširitvena skripta: avtorizacija

Pri razširitveni skripti se avtorizacija izvede po uspešni avtentikaciji paketa. Glavni namen skripte je povpraševanje, koliko časa, če sploh, ima uporabnik na voljo za dostop do zahtevane storitve. Strežnik pričakuje na standardni izhod število, ki predstavlja čas dostopa v sekundah. Ker ima polje trajanje dostopa samo 16 bitov, je zgornja omejitev dostopa $2^{16} - 1 = 65,535$

[5]Programska oprema je na voljo: `http://openspa.org`.

sekund. Čas dostopa lahko nastavimo tudi na 0, kar pomeni da uporabnik nima avtorizacije za dostop.

Razširitvena skripta: dodajanje pravila v požarni zid

Kot je razvidno iz imena skripte, je ta skripta zadolžena za dodajanje pravil v požarni zid. Če ima uporabnik dovoljenje za dostop do omrežnih vrat, strežnik kliče skripto, da zagotovi odjemalcu dostop do vrat. Tu so možni različni programi za različne požarne zidove. Za potrebe diplomske naloge je implementirana skripta, ki uporablja za požarni zid *iptables*. Ta implementacija lahko deluje ali v načinu shranjevanja stanj[6] ali pa tudi brez (več o tem v podpoglavju 3.5.6).

Razširitvena skripta: odstranitev pravila iz požarnega zidu

Pri razširitveni skripti se ob klicu odstrani pravilo, ki ga je izvedla *skripta za dodajanje pravila v požarni zid*. Strežnik je tisti, ki sproži skripto ob izteku števca, ki meri čas dostopa. Če strežnik dobi signal od operacijskega sistema za prekinitev, strežnik tudi izvede skripto za odstranjevanje pravil. S tem počisti za sabo požarni zid in priskrbi, da uporabniki ne morejo več dostopati do sistema. V nasprotnem primeru bi uporabniško pravilo za dostop ostalo v požarnem zidu po izteku dovoljenja.

3.5.2 Primer uspešne zahteve po dostopu

Sledi primer uspešne zahteve za dostop, s poudarkom na vrstnem redu klicanja razširitvenih skript (ES) (slika: 3.10):

1. Strežnik prejme paket. Paket dekodira in na podlagi unikatnega identifikatorja naprave (angl. *device UUID*) zažene skripto *uporabniški imenik* z ukazom *GET_USER_PUBLIC_KEY*, da pridobi javni ključ od naprave.

[6] Ena izmed rešitev težave NAT.

2. Z uporabo javnega ključa preveri pristnost paketa (preveri podpis). Če, je paket veljaven, strežnik zažene skripto za *avtorizacijo*. Skripta vrne, koliko časa ima naprava dostop do želenih omrežnih vrat.

3. Ker je čas dostopa večji od nič, strežnik kliče skripto za *dodajanje pravila v požarni zid* in ustvari izjemo za napravo na podlagi podatkov, ki so v zahtevi (naslov IP, protokol, omrežna vrata itd.).

4. Po uspešnem vnosu izjeme v požarni zid, strežnik ustvari paket tipa odgovor in pošlje odjemalcu informacijo koliko časa ima naprava za dostop.

5. Odjemalec dekodira paket in preveri njegov podpis. Če je paket veljaven, izvleče podatek o tem, koliko časa je bilo odobreno za dostop. Zdaj lahko naprava dostopa do sistema.

6. Po izteku dostopa, strežnik izvede skripto za *odstranjevanje pravila iz požarnega zida* in tako odvzame napravi dostop.

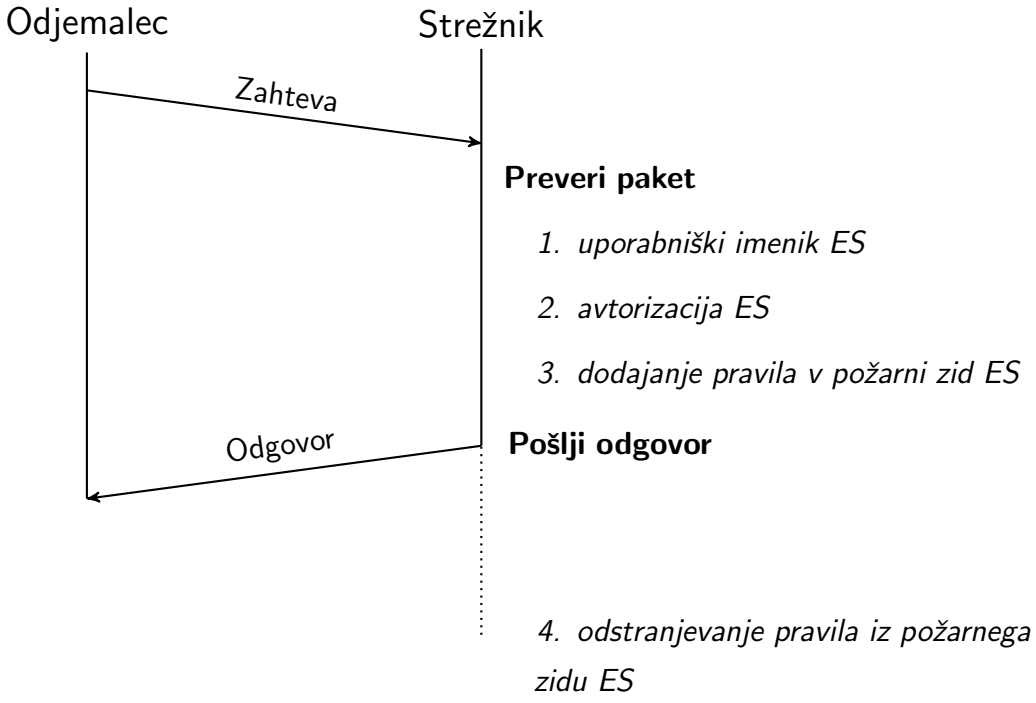
3.5.3 Kriptografija

Protokol OpenSPA v trenutni obliki ne predpisuje vseh možnih kriptografskih metod. Pri implementaciji protokola so določene prve metode podpisovanja in šifriranja (slika: 3.11).

Metoda podpisovanja poteka z uporabo RSA in zgoščevalne funkcije SHA-256. Postopek podpisovanja paketa zahteve je sledeč:

1. Odjemalec vzame glavo OpenSPA in (nepodpisani) podatkovni del paketa, sledi izračun z uporabo zgoščevalne funkcije SHA-256.

2. Zgoščeno vrednost paketa predamo funkciji, ki sprejme zasebni ključ RSA (od odjemalca) in digitalno podpiše podatke. Rezultat funkcije je digitalni podpis, ki se doda v podatkovno polje paketa.



Slika 3.10: Primer uspešne zahteve s poudarkom, katere razširitvene skripte (ES) se izvedejo.

Metoda šifriranja paketa poteka z uporabo javne kriptografije RSA in simetrične kriptografije AES v načinu CBC. Odločili smo se za uporabo javne kriptografije, da ne bi bilo treba hraniti uporabniških ključev za šifriranje na strežniku. Zaradi značilnosti RSA je dešifriranje podatkov počasneje kot z uporabo simetrične kriptografije, npr. AES [17]. V primeru, ko imamo večje pakete OpenSPA, bi bilo treba razdeliti paket na bloke in jih posamezno šifrirati z uporabo RSA. Zaradi večje zmogljivosti strežnika smo se odločili za hkratno uporabo RSA in AES.

Odjemalec in strežnik imata vsak svoj par javnih in zasebnih ključev. Odjemalec mora hraniti javni ključ od strežnika in strežnik mora hraniti javni ključ od odjemalca. Implementacija uporablja 2048 bitne ključe RSA in 256 bitne ključe AES. Kriptografija AES-256-CBC se uporablja za šifriranje podatkov. Geslo kriptograma je naključna vrednost, ki se šifrira z javnim ključem RSA. Za uporabo AES smo se odločili ker je pri RSA možno šifrirati le podatke velikosti ključa. Pri AES velja enako, da je možno šifrirati le podatke velikosti ključa. Prednost AES je v tem, da so ključi manjši in tako s tem lahko prihranimo pri velikosti paketa.

Postopek šifriranje paketa zahteve je sledeč:

1. Odjemalec preda podpisan podatkovni del paketa funkciji za šifriranje.

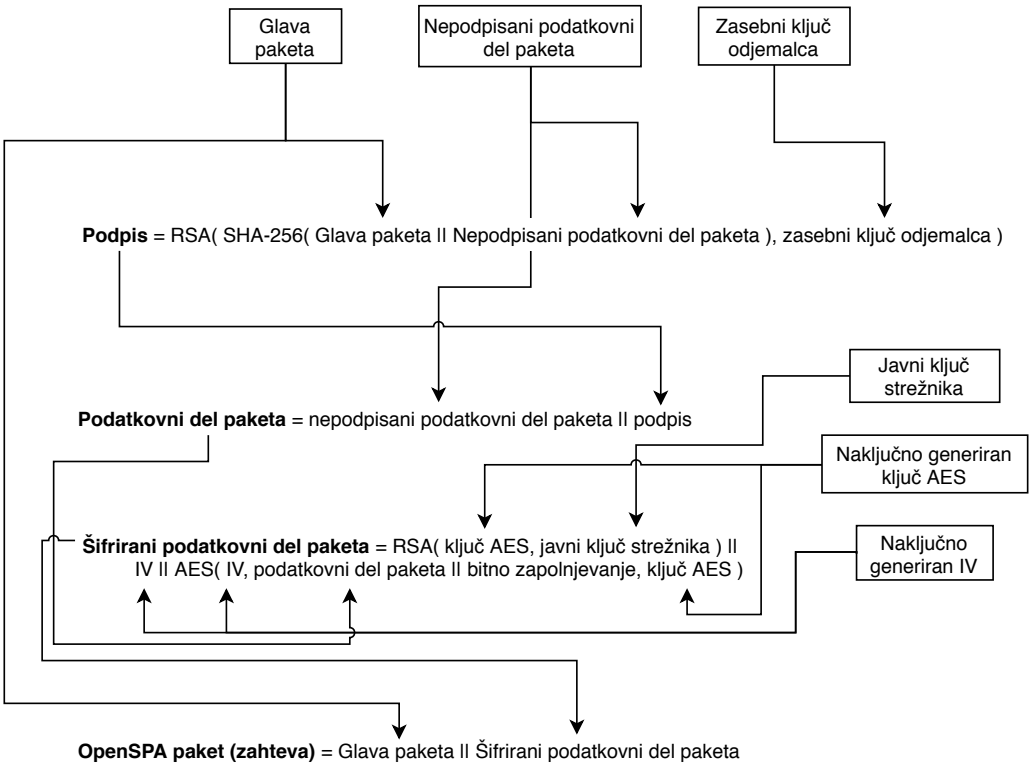
2. Funkcija naključno generira 256 bitni AES ključ in inicializacijski vektor (IV).

3. Podatke zašifriramo s kriptografskim algoritmom AES-256-CBC z uporabo naključnega ključa in IV.

4. IV dodamo kot predpono pri šifriranih podatkih.

5. Ključ AES šifriramo z javnim ključem od strežnika in dodamo kot predpono šifriranih podatkov.

6. Dodamo glavo OpenSPA in imamo šifrirani paket zahteve.



Slika 3.11: Postopek odjemalca pri podpisu in šifriranju paketa tipa zahteva.

Strežnik prejme zahtevo nato v obratnem vrstnem redu izvede nasprotne operacije (dešifriranje namesto šifriranje itd.). Strežnik na enak način podpiše in šifrira pakete tipa odgovor.

3.5.4 Uporabniška datoteka OSPA

Odjemalec mora imeti za uporabo programske opreme OpenSPA veliko različnih podatkov: naslov IP strežnika, omrežna vrata strežnika, javni ključ strežnika, zasebni ključ odjemalca itd. Za poenostavitev uporabe programske opreme smo se odločili za kreiranje uporabniške datoteke s končnico *.ospa*. Uporabniška datoteka je zapisana v formatu *yaml* in vsebuje vse potrebne podatke, da naprava komunicira s strežnikom.

Strežnik podpira tudi generacijo uporabniških datotek preko ukazne vrstice. Na ta način lahko administrator enostavno kreira uporabniško datoteko, ki

jo potem preda uporabniku v uporabo.

Primer uporabniške datoteke OSPA:

```
# Required field. Version of the OpenSPA OSPA file format
version: "0.1.0"

# Required field. Represents the name of the service, can be
# anything.
name: "My SPA server"

# Required field. The client's unique ID (should be a UUID)
# that the server uses along with the client's packet
# signature to verify authentication and authorization.
clientDeviceId: "c19541ee-4e58-4e66-98ea-7e896e4e73a1"

# Optional field. The OpenSPA server IP, to where we should
# send OpenSPA request packets. If not provided, it is
# required that you provide it with the command flag.
serverIp: 88.200.23.31

# Optional field. The OpenSPA server port, to where we should
# send OpenSPA request packets. If not provided the default
# port 22211 will be used. Can be overridden using a command
# flag.
#serverPort: 22211

# Optional field. The Echo-IP server to use to resolve the
# client's public IP and if the client is behind a NAT. If
# not provided the default Echo-IP server will be used:
# "https://ip.openspa.org". If you wish to host your own
# the source code is available here:
# "https://github.com/greenstatic/echo-ip".
```

```
# You can override this field using a command flag.
#echoIpServer: "https://ip.openspa.org"


# Required field. The client's private RSA 2048 bit key.
# This should be the private key corresponding to the
# "publicKey" field public key. This field should be
# provided as a YAML multiline string like so:
#
# privateKey: |
#   -----BEGIN RSA PRIVATE KEY-----
#   <PRIVATE KEY CONTENTS HERE>
#   -----END RSA PRIVATE KEY-----
#
# Note the "|" character after the field name, it denotes
# a multiline string with new lines preserved.

privateKey: |
-----BEGIN RSA PRIVATE KEY-----
/// redacted ///
-----END RSA PRIVATE KEY-----

# Required field. The client's public RSA 2048 bit key.
# This field should be provided as a YAML multiline string
# like so:
#
# publicKey: |
#   -----BEGIN PUBLIC KEY-----
#   <PUBLIC KEY CONTENTS HERE>
#   -----END PUBLIC KEY-----
#
```

```
# Note the "|" character after the field name, it denotes a
# multiline string with new lines preserved.
publicKey: |
-----BEGIN PUBLIC KEY-----
/// redacted ///
-----END PUBLIC KEY-----

# Required field. The server's public RSA 2048 bit key.
# This field should be provided as a YAML multiline string
# like so:
#
# serverPublicKey: |
#   -----BEGIN PUBLIC KEY-----
#   <PUBLIC KEY CONTENTS HERE>
#   -----END PUBLIC KEY-----
#
# Note the "|" character after the field name, it denotes
# a multiline string with new lines preserved.

serverPublicKey: |
-----BEGIN PUBLIC KEY-----
/// redacted ///
-----END PUBLIC KEY-----
```

3.5.5 Zajem prometa

Fwknop je dokazal, da je možno zajeti pakete SPA s filtriranjem celotnega omrežnega prometa naprave z uporabo orodja *libpcap*[7] [19]. V tem načinu delovanja mora strežnik pridobiti dovoljenje od operacijskega sistema, da lahko prejme ves omrežni promet. Strežnik fwknop poda orodju libpcap filter,

[7]Fwknop je zmožen zajeti promet tudi s poslušanjem na vrata, kot navadna omrežna aplikacija.

ki ustreza obliki paketov fwknop. Libpcap poskrbi za filtriranje prometa in posredovanja ustreznih paketov strežniku.

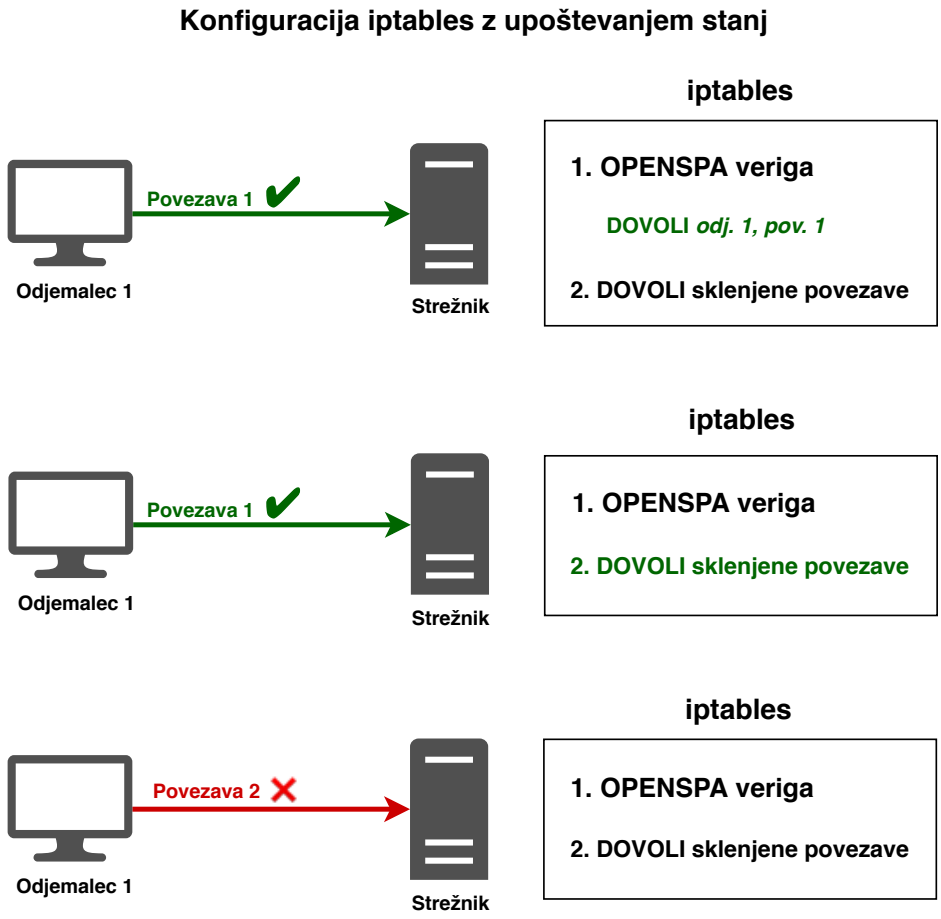
Pri izdelavi strežnika OpenSPA smo se odločili za zajem prometa s poslušanjem na omrežnih vratih. Razlog je v tem, ker strežnik v takšnem načinu delovanja ne potrebuje sistemskih pravic in ni tesno vezan na zunanje knjižnice, ki mogoče ne bi delovale v drugih okoljih ali sistemih.

3.5.6 Uporaba požarnega zidu brez upoštevanja stanj

Med implementacijo programske opreme smo naleteli na težavo pri požarnem zidu *iptables*. Da bi omogočali strežniku normalno zunanjo komunikacijo (pregled posodobitev, zunanjo komunikacijo za potrebe aplikacij itd.) smo dodali pravilo, ki odobri promet, če je povezava v stanju sklenjena. Pravilo je bilo potrebno, ker smo nastavili privzeto pravilo za promet, ki se ne ujema s pravili, da ga strežnik zavrže. Pravilo je delovalo skladno s pričakovanjem, vendar s stranskim učinkom. Ko je bila odjemalcu dovoljena povezava, je bila povezava označena kot sklenjena. Težava je nastala, ko smo odstranili pravilo. Povezava z odjemalcem je bila še vedno označena kot sklenjena in s tem posredno dovoljena. Skratka, nenamerno je bil implementiran SPA z upoštevanjem stanj v požarnem zidu (slika: 3.12).

Da bi dosegli podporo SPA brez upoštevanja stanj povezave (slika: 3.13) smo prišli do sledeče rešitve: V iptables smo dodal dodatno verigo *OPENSPA-BLOCK*. Verigo smo vstavili med pravila verige *OPENSPA*[8] in pravilom za dovoljenje sklenjenih povezav. Ko smo želeli odstraniti pravilo za dostop odjemalcu, se je premaknilo pravilo iz verige OPENSPA v verigo OPENSPA-BLOCK in s tem spremenil status iz 'dovoli' v blokiraj'. Strežnik je prejel promet od odjemalca, kateremu smo odvzeli dostop, pravilo pa je poskrbelo za blokado prometa. Največja pomanjkljivost rešitve je bila v tem, da se v verigo OPENSPA-BLOCK stalno dodajajo pravila za blokado. Sčasoma se veriga OPENSPA-BLOCK napolni in iptables potrebuje dalj časa za pregled paketa. To vodi do dodatne obremenitve strežnika in posledično počasnejšo

[8]Veriga OPENSPA se uporablja za določitev dovoljenih povezav.



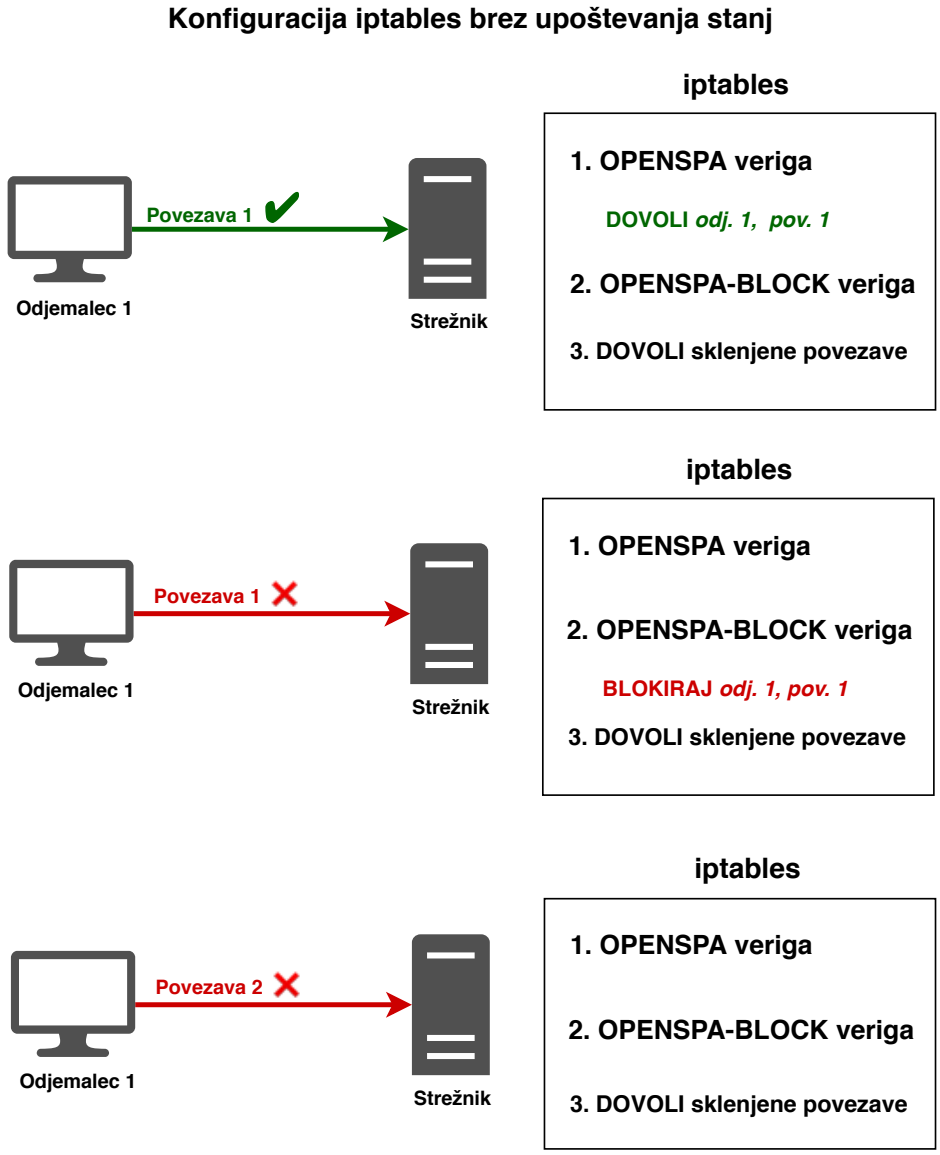
Slika 3.12: Slika predstavlja kaj se zgodi obstoječi in novi povezavi v primeru konfiguracije požarnega zidu iptables z upoštevanjem stanj.

komunikacijo. Rešitev je v periodičnem odstranjevanju pravil. Pravilo v verigi OPENSPA-BLOCK je potrebno le nekaj sekund do minute, dokler povezava ni označena kot prekinjena.

3.5.7 Težava s translacijo NAT

Predstavljena implementacija protokola poskuša rešiti težavo s translacijo NAT na sledeče načine:

- Podpora za IPv6



Slika 3.13: Slika predstavlja kaj se zgodi obstoječi in novi povezavi v primeru konfiguracije požarnega zidu iptables brez upoštevanjem stanj.

- Zastavica NAT v zahtevi
- Možnost uporabe požarnega zidu v načinu, ki uporablja stanje povezave

Vzporedno je bila razvita dodatna manjša storitev - *Echo-IP* - ki deluje kot razreševalnik javnih naslovov IP. Z uporabo Echo-IP, odjemalec lahko pošlje HTTP(S) zahtevo vnaprej določenem naslovu (npr. `https://ip.openspa.org`) in v odgovoru HTTP pridobi podatek, kateri javni IP je bil označen kot pošiljatelj zahteve[9].

Ker ima odjemalec OpenSPA mehanizem, da pridobi svoj IPv4 in IPv6 naslov (ki sta lahko le lokalna, lahko pa tudi javna), lahko primerja z odgovorom storitve Echo-IP. V primeru, ko sta naslova IP enaka (naslov IP, ki ga ima odjemalec nastavljen in odgovor storitve Echo-IP), potem ima naprava končno povezljivost. V nasprotnem primeru odjemalec nima končne povezljivosti in je tako za omrežjem NAT (v tem primeru je odjemalcu potrebno vključiti zastavico NAT pri pošiljanju zahteve OpenSPA).

3.6 Možne izboljšave

Med opisom OpenSPA so že navedene nekatere težave protokola in možne rešitve. Poleg opisanih rešitev so možne še dodatne izboljšave, ki se lahko uporabijo pri nadgradnji programske opreme oz. pri izdelavi novejše različice protokola.

3.6.1 Filtriranje v omrežni kartici

Čeprav ni nikjer omenjeno, da OpenSPA lahko brani napade DoS, je protokol dovolj dobro zasnovan, da bi bilo smiselno v prihodnje raziskovati možnosti

[9]Echo-IP podpira postavitev storitve, da deluje samo v načinu IPv4 ali IPv6. V ta namen imamo dve ločeni domeni: `https://ipv4.ip.openspa.org` in `https://ipv6.ip.openspa.org`. `https://ip.openspa.org` pa ima IPv4 in IPv6 podporo in tako vrača tisti naslov IP, ki se je uporabil za pošiljanje zahteve (ponavadi, če ima naprava IPv6 potem ima protokol IPv6 višjo prioriteto kot IPv4).

izboljšave OpenSPA za aktivno obrambo napadov DoS. Poleg možnih iptables pravil za omejevanje prometa bi bilo zanimivo raziskati možnosti za uporabo orodja, kot so: eXpress Data Path (XDP) in Berkeley Packet Filter (BPF). Tovrstna orodja omogočajo omejeno programiranje omrežne kartice in njenih gonilnikov tako, da je možno filtrirati promet na omrežni kartici. Takšen pristop razbremeni strežnik in omogoča zavračanje ogromnih količin prometa bistveno hitreje, kot bi to zmogel strežnik, ne da bi to vplivalo na izvajanje programske opreme [8].

3.6.2 Digitalna potrdila

Med izdelavo referenčne implementacije protokola je bil določen čas porabljen za raziskovanje možnosti implementacije digitalnih potrdil v zahteve OpenSPA. Z vključitvijo digitalnega potrdila[10] v paket bi razbremenili strežnik pri iskanju javnega ključa od odjemalca. Vse kar bi bilo potrebno storiti je to, da se preveri digitalno potrdilo. V primeru, da je veljavno, bi lahko strežnik pridobil javni ključ od odjemalca iz samega digitalnega potrdila in potem preveril podpis.

Poskušali smo uporabiti RSA 2048 bitno digitalno potrdilo. Poskus ni bil uspešen. Ko je bilo digitalno potrdilo podpisano z uporabo lastne generirane certifikatne agencije je velikost potrdila presegla omejitve paketa OpenSPA. Pri naslednjem koraku smo poskušali kodirati digitalno potrdilo v binarni obliki (ponavadi so kodirani v base64 obliki), a so bili rezultati enaki.

Čeprav RSA digitalna potrdila niso bila uspešna, bi bilo možno vpeljati potrdila, ki temeljijo na kriptografiji z eliptičnimi krivuljami (angl. *eliptic curve cryptography, ECC*). ECC je kriptografija z javnimi ključi, ki omogoča šifriranje z manjšimi ključi, kot so na primer RSA, z enako ali celo boljšo varnostjo. Ključ ECC, ki je dolžine 233 bitov, je primerljiv 2048 bitnemu RSA ključu [9]. Takšne velikosti bi verjetno omogočale implementacijo digitalnih potrdil ECC v protokolu OpenSPA.

[10]Digitalno potrdilo bi moralo biti podpisano s strani certifikatne agencije, ki jo strežnik OpenSPA zaupa.

3.6.3 Pošiljanje preko TCP/HTTPS

V nekaterih večjih omrežjih v podjetjih je uporaba prometa UDP nedovoljena. Ker se protokol OpenSPA zanese na UDP za pošiljanje paketov bi bila uporaba protokola v takšnih omrežjih nemogoča. Projekt fwknop je uspešno naredil nekaj raziskav na to temo. Razvili so mehanizem uporabe fwknop protokola preko TOR omrežja (angl. *The Onion Router*) - TOR omogoča le povezave TCP.

3.6.4 Uporaba v oblačnih in kontejnerskih okoljih

Pri izdelavi protokola se nismo osredotočali na oblačne storitve ali kontejnerska okolja. Tovrstne tehnologije postajajo čedalje bolj priljubljene. Priporočljivo bi bilo raziskati možnosti izboljšave protokola za lažjo uporabo v tovrstnih okoljih.

3.6.5 Avtomatična zahteva po dostopu po potrebi

Referenčna implementacija protokola trenutno lahko deluje samo v načinu ročne zahteve. To pomeni, da je treba za vsaka omrežna vrata (lahko tudi razpon vrat) zahtevati dostop z ročnim poganjanjem programa. Tovrstni postopek ni primeren za običajne uporabnike. Da bi približali protokol običajnim uporabnikom bi bilo potrebno zasnovati mehanizem, ki bi samodejno zahteval dostop v času, ko uporabnik dostopa do storitve. Tovrstni mehanizem bi poskrbel, da je protokol OpenSPA popolnoma neviden uporabniku.

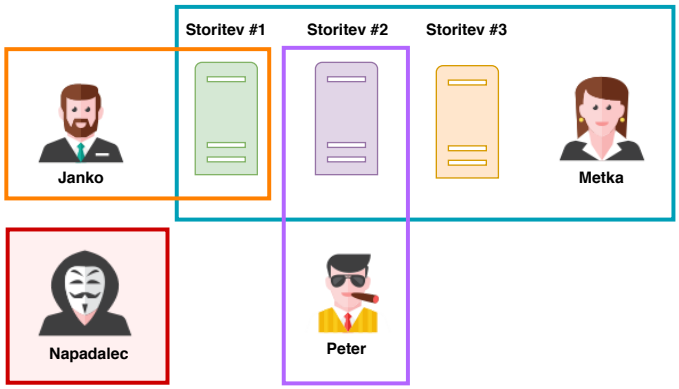
Poglavje 4

Programsko določen rob omrežja

Čeprav je SPA učinkovita tehnika skrivanja storitev, je njena uporaba omejena na primere, kjer vnaprej vemo, kje se nahajajo storitve. SPA deluje na omrežnem sloju in operira z naslovi IP in omrežnimi vrati. Programsko določen rob omrežja (angl. *Software Defined Perimeter, SDP*) je logično nadaljevanje SPA. SDP deluje na višji ravni kot SPA, to je na ravni storitev.

Naloga SDP je zagotoviti vsakemu uporabniku lasten rob omrežja (slika: 4.1). Lasten rob omrežja pomeni, da vsak uporabnik (z različnimi pravicami) vidi topologijo omrežja drugače od ostalih. To pomeni, da nepooblaščeni uporabniki nimajo možnosti dostopati do storitev za katere nimajo pravic. To omogoča večjo varnost, saj so napadi na storitev omejeni na pooblaščene naprave. Ker nas ne zanima fizična topologija omrežja, je možno postaviti storitve na internet in jim omejiti dostop z uporabo koncepta programsko določenega roba omrežja.

Leta 2013 je CSA (Cloud Security Alliance - neprofitna organizacija, ki spodbuja varnost v oblačnih storitvah) ustvarila delovno skupino z namenom, da bi razvila rešitev, ki bi varovala aplikacijsko infrastrukturo pred zunanjimi omrežnimi napadi. Prvo specifikacijo rešitve so javno objavili in koncept poimenovali SDP [3].



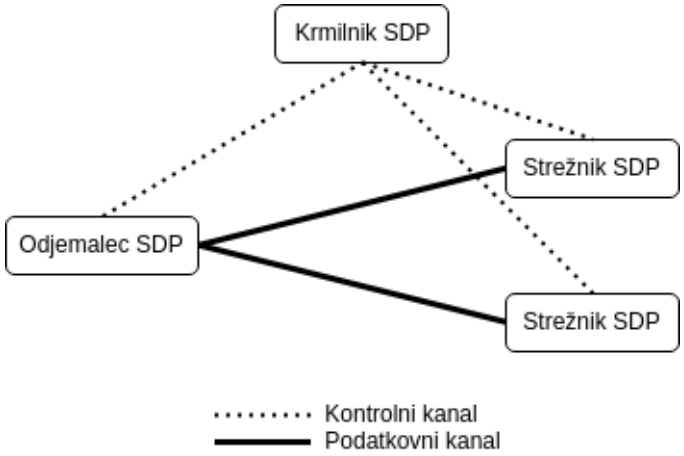
Slika 4.1: Vsak uporabnik ima določen svoj rob omrežja. Odvisno od uporabniških pravic, uporabniki med seboj vidijo različne omrežne topologije.

Principi, ki jih specifikacija SDP zahteva, niso povsem novi. Podobne tehnike, ki jih specifikacija določa, so že v uporabi v omrežjih oboroženih sil in obveščevalnih služb Združenih držav Amerike [3]. SDP posvoji njihov logični model in določi standardni potek delovanja.

SDP predpisuje model kjer imajo uporabniki omrežni dostop samo do avtoriziranih storitev. Odjemalec se mora pred komunikacijo avtenticirati in avtorizirati. Če je postopek uspešen, se ustvari šifrirana povezava med odjemalcem in aplikacijo po kateri se izmenja podatkovni promet [3].

4.1 Arhitektura SDP

Specifikacija SDP določa tri vrste komponent: odjemalec SDP (angl. *initiating SDP host*), strežnik SDP (angl. *accepting SDP host*) in krmilnik SDP (angl. *SDP controller*). Komunikacija se začne s krmilnikom, ki določa katera naprava lahko komunicira s katero. Povezave s krmilnikom so del kontrolnega kanala. Ta kanal skrbi za povezave na podatkovnem kanalu, ki predstavljajo komunikacijo med odjemalcem in storitvijo.



Slika 4.2: Prikaz kontrolnega in podatkovnega kanala storitve, zaščitene s SDP [6].

4.1.1 Tipičen potek komunikacije

1. Odjemalec SDP pošlje paket SPA krmilniku. Ob uspešni povezavi se ustvari medsebojna povezava TLS.
2. Krmilnik SDP poizvede zunanje vire, da pridobi seznam storitev do katerih ima odjemalec dostop.
3. Krmilnik sporoči vsem strežnikom SDP do katerih ima odjemalec dostop, da lahko začnejo sprejemati povezave od odjemalca.
4. Krmilnik sporoči odjemalcu do katerih storitev ima dostop.
5. Odjemalec pošlje paket SPA vsakemu strežniku. Ob uspešni povezavi, se ustvari do strežnika medsebojna povezava TLS po kateri poteka podatkovni promet s storitvami.

4.1.2 Preverjanje pristnosti z enim paketom

Prva specifikacija SDP predpisuje uporabo lastne implementacije SPA. Implementacija predpisuje uporabo enkratnih gesel, ki se generirajo po specifikaciji RFC 4226 (HOTP). HOTP (angl. HMAC-based One-time Password

algorithm) je algoritem za generiranje enkratnih gesel z uporabo HMAC. Algoritem se pogosto uporablja v programski opremi, ki implementira večfaktorsko avtentikacijo [18].

Paket SPA je zgrajen iz treh polj:

- AID (Agent ID) - 32 bitni identifikator SDP naprave
- Geslo - 32 bitno število, ki ga HOTP vrne
- Števec - 64 bitno število, ki mora biti sinhronizirano med odjemalcom in strežnikom

Protokol za SPA, ki ga predpisuje CSA-jeva specifikacija SDP, ima velike pomanjkljivosti. Kot prvo, se paket pošlje strežniku (ali krmilniku) preko povezave TCP. Takšna implementacija SPA ni najboljša. Ena od značilnosti tehnike SPA je skrivanje storitve. Skrivanje storitve pri SDP implementaciji ni možno, saj strežnik SPA sprejema pakete z uporabo TCP (kar odkrije obstoj storitve).

Podatki o uporabniškem naslovu IP se pridobijo iz glave IP. Uporaba teh podatkov je tvegana, saj podatki niso podpisani. Napadalec lahko spremeni naslov v glavi in zavede strežnik, da omogoči napadalcu dostop [7].

Paket SPA ne vsebuje podatkov o zahtevanih vratih ali protokolu. Pomanjkljivost podatkov v paketu SPA je bila namenoma izpuščena, saj je uporaba SPA pri SDP omejena na skrivanje storitve TLS na strežniku.

Fwknop in OpenSPA nimata naštetih slabostih in posledično ponujata boljšo omrežno zaščito. Ustvarjalec projekta fwknop, Michael Rash, je delovni skupini SDP predstavil slabosti predpisane metode SPA in dal napotke za izboljšave. Tudi mi smo delovni skupini razjasnili slabosti njihove implementacije in predstavili naš protokol OpenSPA. V času pisanja omenjene diplomske naloge je v pripravi druga različica specifikacije SDP. Različica poskuša uveljaviti izboljšave, ki jih je Michael Rash priporočal in katera rešijo tudi določene težave, katere so izpostavljene v tej diplomski nalogi. Dokler ni končana druga specifikacija, se priporoča uporaba alternativnih implementacij SPA.

4.2 Implementacija SDP Waverley Labs

V času pisanja diplomske naloge je na trgu prisotno le malo implementacij SDP. Vpogled v podrobnosti njihovega delovanja ni mogoč zaradi nedostopnosti izvorne kode. Na srečo obstaja odprtokoden projekt podjeta Waverley Labs, ki implementira SDP[1]. Programska oprema, ki implementira krmilnik SDP, je sprogramirana z uporabo ogrodja Node.js. Odjemalec in strežnik SDP sta implementirana s posebno različico fwknop.

Vsak strežnik SDP je zaščiten z fwknop. Krmilnik SDP skrbi, da se generirajo uporabniški ključi in da se le-te pošljejo vsakemu strežniku SDP. To je potrebno ker, kot smo že opisali, fwknop zahteva, da so uporabniški ključi shranjeni na sistemu, ki ga fwknop ščiti.

Pri namestitvi programske opreme, administrator ustvari svojo (samopodpisano) certifikatno agencijo. Certifikatna agencija (CA) podpisuje digitalna potrdila od komponent SDP. CA predstavlja vsem komponentam zaupanja vredno avtoriteto. Uporablja se pri preverjanju medsebojne povezave TLS.

Krmilnik SDP je povezan na podatkovno bazo MySQL. Podatkovna baza vsebuje podatke o uporabnikih, skupinah, strežnikih SDP in o uporabniških pravicah. Krmilnik iz podatkovne baze pridobi seznam storitev do katerih ima uporabnik dovoljen dostop. Poizvedba se sproži, ko je postopek SPA uspešen in ko je medsebojna povezava TLS sklenjena. Krmilnik ob pridobitvi seznama sproži ukaz (preko medsebojne povezave TLS) pri vsaki storitvi na seznamu, da dovoli povezavo odjemalcu.

Po prejemu seznama storitev, odjemalec pošlje paket fwknop na vsako storitev, da zahteva dostop. Če je paket veljaven, se omrežna vrata od avtorizirane storitve odprejo.

[1]Programska oprema dostopna na: `https://github.com/WaverleyLabs/SDPcontroller`.

4.2.1 Slabosti

Implementacija SDP Waverley Labs uporablja fwknop za SPA, zato so prisotne slabosti, ki smo jih opisali že v prejšnjih poglavjih. To se predvsem navezuje na tesno sklopljen postopek avtentikacije in avtorizacije. Ker fwknop potrebuje ključe od vseh uporabnikov v svoji konfiguracijski datoteki, SDP krmilnik pošlje vse ključe vsaki storitvi. Poleg podvajanje podatkov in slabosti ki jih tovrstno dejanje prinaša, je shranjevanje uporabniških ključev na vsaki storitvi varnostno tvegano. Kot tudi težavi pri skalabilnosti sistema.

Ker je krmilnik SDP napisan v programskem jeziku Javascript z uporabo ogrodja Node.js, bi lahko to postalo problematično. Node.js strežnik, ki izvaja program Javascript, in se izvaja znotraj enega procesa (v eni niti). Programiranje poteka asinhrono, posledično so vhodni/izhodni klici (npr. komunikacija z bazo) zelo učinkoviti, saj ne blokirajo glavnega procesa. Slabost strežnika Node.js je izvajanje procesorsko intenzivnih poslov [4]. Klasičen primer je uporaba kriptografije. Izvajanje procesorsko zahtevnih poslov ne poteka asinhrono in posledično blokira glavni proces. To povzroči, da se zmogljivost procesiranja zahtev zmanjša. Poleg vsega programski jezik Javascript ni najboljša izbira za programsko opremo, ki izvaja posle krmilnika SDP. Za kritičen kos infrastrukture, kot je krmilnik SDP, je boljša izbira programski jezik, ki se prevede in nudi boljše zmogljivosti.

Krmilnik SDP je tesno vezan na lastno podatkovno bazo. Takšen pristop vodi do podvajanje podatkov iz sistemov, kjer že hranimo uporabniške podatke (npr. FreeIPA). V podjetjih takšna rešitev ni optimalna, saj pri usklajevanju podatkov med sistemi lahko pride do nepredvidljivih napak.

4.3 Lastna implementacija programske določitve roba omrežja

Naslednji smiseln korak za SPA je implementacija SDP. Vizija uporabe OpenSPA je, da ta predstavlja sestavni košček druge programske opreme. Uporaba

OpenSPA pri lastni implementaciji SDP omogoča iz prve roke izkusiti, kako lahko OpenSPA nadgradi programsko opremo. Pri zasnovi implementacije SDP v diplomski nalogi smo se zgledovali po implementaciji od Waverley Labs in pri tem poskušal odpraviti njene slabosti. Lastno implementacijo SDP smo poimenovali OpenSDP[2] in uporabil OpenSPA kot temeljni element programske opreme.

OpenSDP je v trenutnem stanju zgolj ovitek okoli OpenSPA in vključuje mehanizem odkrivanja storitev.

Tipičen postopek uporabe:

1. Odjemalec pošlje strežniku OpenSDP zahtevo OpenSPA za dostop do storitve OpenSDP.

2. Če ima uporabnik dostop do storitve OpenSDP se odprejo vrata.

3. Preko medsebojne povezave TLS odjemalec pošlje strežniku OpenSDP zahtevo za odkrivanje storitev.

4. Strežnik OpenSDP zaradi medsebojne povezave TLS lahko varno izve podatke o odjemalcu in poizvede do katerih storitev ima uporabnik dostop. Seznam storitev se pošlje nazaj odjemalcu.

5. Od prejema seznama storitev odjemalec pošlje vsaki storitvi zahtevo OpenSPA.

6. Storitve preverijo paket OpenSPA. Če ima odjemalec dostop, se odprejo omrežna vrata.

Končni rezultat programske opreme OpenSDP je ukaz, s katerim uporabnik pridobi dostop do vseh avtoriziranih storitev. Možno bi bilo dodati OpenSDP ukaz med programe, ki se izvajajo v ozadju med zagonom operacijskega sistema. Na ta način bi računalnik od uporabnika v ozadju zahteval dostop do avtoriziranih storitev in bi bila uporaba OpenSDP uporabniku nevidna.

[2]Programska oprema je na voljo: `http://opensdp.org`.

4.3.1 Podrobnosti implementacije

Zaradi medsebojne povezave TLS so potrebna digitalna potrdila. Na strežniku OpenSDP je potrebno ustvariti lastno certifikatno agencijo. Nadalje potreben je podpis javnih ključev od vseh uporabnikov s strani certifikatne agencije. Vsak odjemalec potrebuje podpisano digitalno potrdilo od naše OpenSDP certifikatne agencije in OSPA datoteko (konfiguracijska datoteka od odjemalca za uporabo OpenSPA). Vsak strežnik je zaščiten z OpenSPA, vključno s strežnikom OpenSDP.

Odjemalec ima možnost ročno (če mu je poznan naslov IP in vrata storitve) ustvariti zahtevo OpenSPA in poslati strežniku zahtevo za dostop. Težava je, da ponavadi odjemalec nima potrebnih podatkov o dostopnih storitvah, zato potrebuje mehanizem za odkrivanja le teh. Za odkrivanje dostopnih storitev skrbi strežnik OpenSDP.

Komunikacija z strežnikom OpenSDP poteka preko enostavnega HTTP REST vmesnika, ki je zaščiten z OpenSPA. Odjemalec, da bi lahko komuniciral z strežnikom OpenSDP mora poslati strežniku paket OpenSPA. Naslednji korak je ustvarjanje medsebojne povezave TLS med odjemalcem in strežnikom OpenSDP. Po uspešni povezavi odjemalec pošlje strežniku GET zahtevo na */discover*. Odgovor je seznam storitev do katerih ima odjemalec dostop.

Seznam dostopnih storitev se pridobi iz datotek tipa YAML, kjer so navedene vse storitve in odjemalci, ki imajo do njih dostop. Odjemalec, ki ima podatek o naslovu IP, protokolu in vratih ima vse potrebne podatke, da lahko ustvari zahtevo OpenSPA.

4.3.2 Možna načina delovanja

Neodvisni način delovanja

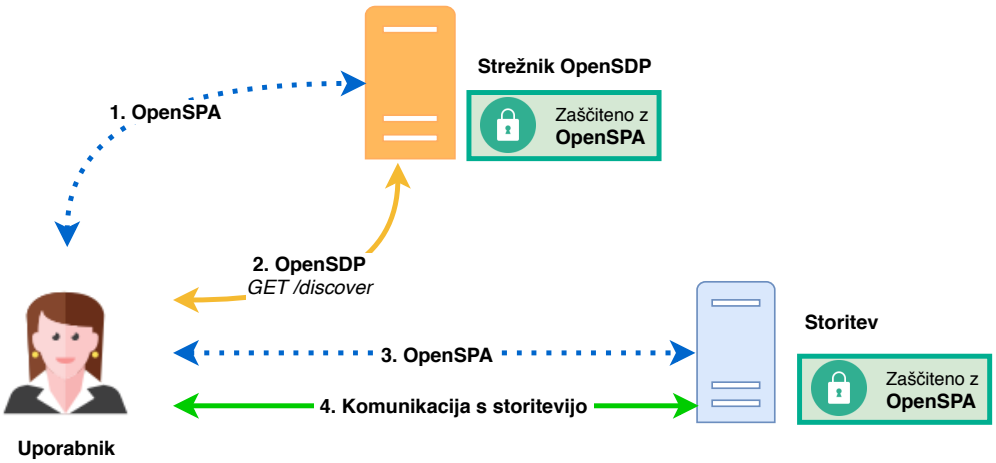
Najbolj enostavna arhitektura OpenSDP je uporaba OpenSDP izključno za odkrivanje storitev (slika: 4.5). Trenutno je to edini način delovanja, ki je podprt. V takšnem načinu delovanja imamo sisteme strogo med seboj

```
# clients.yaml
version: 0.1.0
kind: clients
clients:
- deviceId: 9f84fbb8-10e8-4b8a-abd2-bb91cbf484df
  label: alice
  services:
  # All these services need to exist in the services.yaml file
  - name: example-www
```

Slika 4.3: Primer datoteke *clients.yaml*

```
# services.yaml
version: 0.1.0
kind: services
services:
- name: example-www
  ip: 192.168.1.1
  ports:
  - [tcp, 80]
  - [tcp, 443]
  tags:
  - www
  - internal
  accessType:
  - OpenSPA
```

Slika 4.4: Primer datoteke *services.yaml*



Slika 4.5: Primer uporabe OpenSDP za odkrivanje storitve.

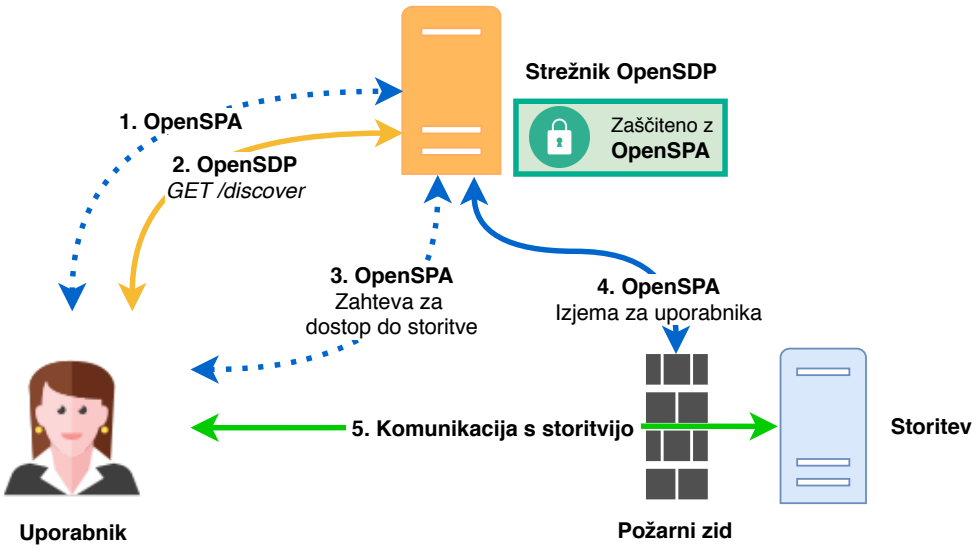
ločene. Posledično se avtentikacija izvaja dvakrat, pri strežniku OpenSDP in strežniku OpenSPA.

Upravljalski način delovanja

Drugi način delovanje je, da strežnik OpenSDP poleg izvajanje posla odkrivanje storitev tudi izvaja vlogo strežnika OpenSPA za storitev. Odjemalec bi namesto pošiljanja zahteve po dostopu direktno do storitve poslal zahtevo na strežnik OpenSDP (slika: 4.6). V takšnem načinu delovanja storitve ne potrebujejo lastnega strežnika OpenSPA. To poenostavi upravljanje celotnega sistema, beleženje aktivnosti in posodobitve. Slabost takšnega sistema je, da strežnik OpenSDP postane kritična točka odpovedi. V primeru, da odpove strežnik OpenSDP je komunikacija med odjemalci in storitvami onemogočena. Postavitev dodatnih strežnikov OpenSPA za višjo razpoložljivost s trenutno implementacijo ni možna.

4.3.3 Možnosti izboljšave

V trenutni obliki projekt OpenSDP podpira samo neodvisni način delovanja. Naslednji korak izboljšave programske opreme je podpora za upravljalski



Slika 4.6: Primer uporabe OpenSDP za odkrivanje storitve in preverjanje pristnosti.

način delovanja. Programska oprema OpenSDP bi se tako lažje postavila v obstoječih okoljih in posledično bila bolj privlačna podjetjem.

Pri postavitvi strežnika OpenSDP v neodvisnem načinu delovanja je potrebno konfigurirati uporabniške nastavitve (javni ključ za OpenSPA in seznam avtoriziranih storitev) na dveh mestih - strežniku OpenSDP in pri storitvi na strežniku OpenSPA. Takšen postopek nastavitve pa vsebuje precejšnje pomanjkljivosti ter je v celoti neučinkovit. Razviti bi bilo potrebno razširitveno skripto za strežnik OpenSPA, ki bi preko omrežja komuniciral s strežnikom OpenSDP in si tako pridobil potrebne podatke.

OpenSDP pridobi podatke o uporabnikih in storitvah preko datotek YAML. Takšna funkcionalnost je primerna v primeru, ko imamo do približno deset uporabnikov in majhno (npr. do 20) število storitev. V večjih in bolj resnih namestitvah bi bila integracija z obstoječimi uporabniški imeniki (npr. Microsoft AD, FreeIPA, Amazon IAM, itd.) obvezna. Implementacija takšne funkcionalnosti lahko sledi tehniki, ki jo razvil OpenSPA - razširitvenih skript ali z uporabo storitve preko HTTP REST vmesnika, ki implementira integracijo.

Če bi želeli, da se takšen sistem uporablja v večjih omrežjih, bi bilo potrebno razviti mehanizme nadzorovanja in beleženja aktivnosti. Takšne funkcionalnosti se pričakujejo od programske opreme, ki skrbi za omrežno varnost. Za končne uporabnike je uporaba programske opreme OpenSDP preko ukazne vrstice neprimerna. V ta namen bi bilo potrebno razviti grafični vmesnik. Grafični vmesnik bi večino časa preživel v ozadju in skrbel za nemoteno komunikacijo s storitvami. Ponujal bi tudi grafičen prikaz uporabniku dostopne storitve z vsemi omejitvami (npr. storitev ni dostopna, če je uporabnik za omrežjem NAT).

Trenutna implementacija ne podpira shranjevanja in izvajanja omrežne politike dostopa, ko uporabnik zahteva seznam storitev. Potrebno je dodati polje in določiti format sporočil, ki bodo določili omrežno politiko dostopa. Pravila omrežne politike bi se lahko nanaša na:

- Naslov IP
- Uporabo storitve samo ob določenih dnevih
- Omejitve dostopa, ko je uporabnik za omrežjem NAT
- Vrsto naprave (npr. namizni računalnik, prenosnik, telefon itd.)
- Operacijski sistem
- Različice programske opreme (npr. dostop je zavrnjen, če je različica protivirusne zaščite manjša kot 1.4)

Za shranjevanje in določitev tovrstnih pravil bi bilo potrebno razviti posebno programsko opremo. Potreben bi bil tudi pregled obstoječih rešitev in možnosti integracij.

Pogosta težava pri vpeljavi javne kriptografije je distribucija ključev. OpenSDP in OpenSPA se močno zanašata na javno kriptografijo. Generacija in distribucija potrebnih digitalnih potrdil in javnih ključev je eden izmed problemov, ki se ga oba projekta izogibata. Za uporabo programske opreme

OpenSDP v večjem okolju bi bilo potrebno razmisliti o najboljšem načinu distribucije ključev, ki se uporabljajo pri vzpostavitvi povezave.

CSA specifikacija SDP opisuje možnost tuneliranja prometa med odjemalcem in strežnikom preko vzpostavljene medsebojne povezave TLS (pod imenom Dynamical Tunnel Mode, DTM). Dodatne raziskave o implementaciji in integraciji obstoječih rešitev v OpenSDP bi bile po vsej verjetnosti nadvse zanimive. Takšna funkcionalnost bi omogočala šifriranje podatkov med odjemalcem in strežnikom, kar bi bilo zanimivo za storitve, ki ne uporabljajo šifriranih povezav. Uporaba tuneliranje povezav bi pomenila tudi, da navidezno zasebna omrežja (angl. *virtual private network*) ne bi bila potrebna. Uporaba več VPN tehnologij hkrati je problematična in pogosto uporabniku neprijazna. SDP s podporo tuneliranja predstavlja eno izmed možnih rešitev.

Podobno, kot je že bilo navedeno pri OpenSPA bi bilo potrebno razviti mehanizem zahtevanja dinamičnega dostopa po potrebi. Pri OpenSDP sta opisana dva pristopa glede zahteve o dostopu. Lahko zahtevamo dostop ročno vsake storitve posebej ali pa do vseh hkrati. Zaradi enostavnosti uporabe bi uporabniki uporabljali dostop do vseh storitev hkrati. Takšen način določitve dostopa ni varnostno najbolj primeren, ker krši princip najmanjšega privilegija (angl. principle of least privilege). Rešitev bi bila v razvoju tretjega mehanizma, to je v tako imenovanem pametnem dostopu, ki bi na podlagi aktualnih uporabniških potreb samodejno zahteval dostop do storitve.

Poglavje 5

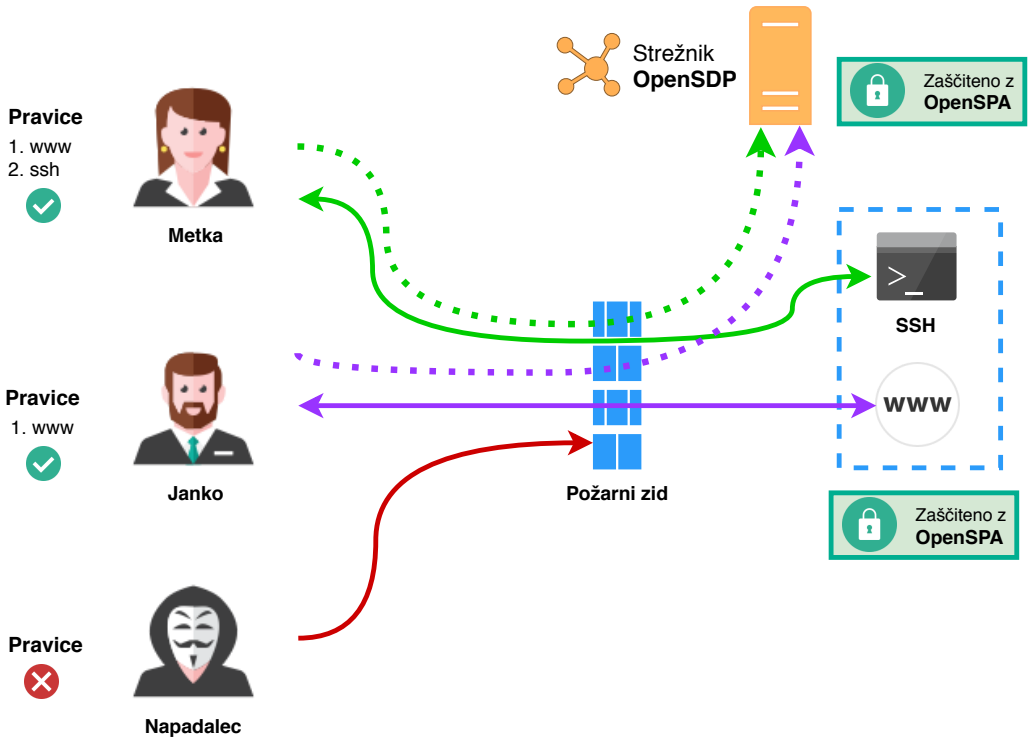
Testiranje delovanja in primer uporabe

Po uspešni implementaciji programske opreme smo testirali delovanje sistema na primeru uporabe. Primer uporabe (slika: 5.1) vsebuje dva avtorizirana uporabnika in enega napadalca. Janko in Metka imata oba dostop do OpenSPA zaščitene spletne strani *www.example.si*. Ker je Metka skrbnica strani, je tudi avtorizirana za dostop preko povezave SSH.

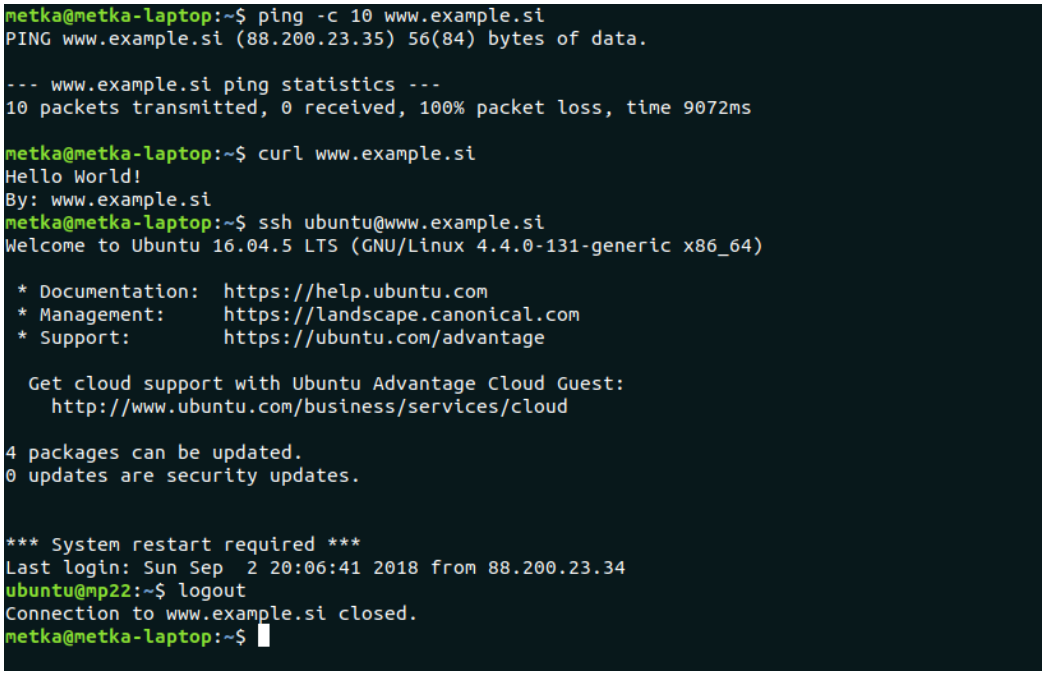
Avtorizirani uporabniki, da pridobijo dostop, morajo le zagnati ukaz:

```
$ opensdp-client access -a

level=info msg="Successfully resolved IPs" behindNat=false publicIp
    =193.2.177.239
level=info msg="Sending OpenSPA request packet" packetSize=610
level=info msg="Received OpenSPA response packet" packetSize=562
level=info msg="Successfully performed OpenSPA exchange"
level=info msg="You have been granted access to ports" duration=30 endPort
    =33311 protocol=TCP startPort=33311
level=info msg="Gaining access to all authorized services" count=1
level=info msg="Successfully resolved IPs" behindNat=false publicIp
    =193.2.177.239
level=info msg="Sending OpenSPA request packet" packetSize=610
level=info msg="Received OpenSPA response packet" packetSize=562
```



Slika 5.1: Shema primera uporabe. Janko in Metka lahko dostopata do storitev z uporabo protokola OpenSPA in OpenSDP, napadalec pa ne.



Slika 5.2: Metka ima dostop do spletne strani (ukaz curl) in storitve ssh na strežniku *www.example.si*. Nima pa dostopa do protokola ICMP (ping ne deluje).

```
level=info msg="Successfully performed OpenSPA exchange"
level=info msg="You have been granted access to ports" duration=30 endPort
    =80 protocol=TCP startPort=80
level=info msg="Again requesting access in a few seconds" seconds=15
```

Metka zažene ukaz in ima zdaj dostop do spletne strani in storitve SSH (slika: 5.2). Metka nima pravic pošiljanje prometa ICMP, kar povzroči 100% izgubo paketov.

Enako kot Metka, Janko zažene ukaz za dostop (slika: 5.3). Janko nima pravic pošiljanje prometa ICMP in povezave na storitev SSH. Ima pa dostop do spletne strani, ki deluje kot pričakovano.

Napadalec je vsak, ki ne zahteva dostopa za storitev in nima ustreznih pravic. OpenSPA priskrbi, da je napadalcu onemogočen dostop do vseh storitev (slika: 5.4) - promet ICMP, spletna stran in storitev SSH. Napadalec, brez dodatnih informacij, omrežno ne vidi strežnika - za njega strežnik ne

```
janko@janko-desktop:~$ ping -c 10 www.example.si
PING www.example.si (88.200.23.35) 56(84) bytes of data.

--- www.example.si ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9072ms

janko@janko-desktop:~$ curl www.example.si
Hello World!
By: www.example.si
janko@janko-desktop:~$ ssh ubuntu@www.example.si
ssh: connect to host www.example.si port 22: Connection timed out
janko@janko-desktop:~$
```

Slika 5.3: Janko ima dostop do spletne storitve. Do ostalih storitev na strežniku nima dostopa.

```
napadalec@ECorp:~$ ping -c 10 www.example.si
PING www.example.si (88.200.23.35) 56(84) bytes of data.

--- www.example.si ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9062ms

napadalec@ECorp:~$ curl www.example.si
curl: (7) Failed to connect to www.example.si port 80: Connection timed out
napadalec@ECorp:~$ ssh ubuntu@www.example.si
ssh: connect to host www.example.si port 22: Connection timed out
napadalec@ECorp:~$
```

Slika 5.4: Napadalec nima dostop do nobene storitve na strežniku.

obstaja. To nam omogoča skrivanje storitve na internetu, brez skrbi o nepooblaščenem dostopu.

Poglavje 6

Zaključek

V diplomskem delu smo opisali dve možni tehniki dinamičnega dodeljevanja dostopa omrežnim napravam. Ponazorjena je tako prednost kot slabost obeh predstavljenih metod. Opisali in analizirali smo tudi obstoječe projekte, ki so aktivni na področju SPA in SDP. Analize so nam omogočile razvoj skupka programske opreme, ki omogoča omrežno varovanje storitev.

Zasnovali smo lastni protokol SPA, OpenSPA, ter ga implementirali. Za delovanje programske opreme OpenSPA v omrežjih NAT, smo razvili tudi programsko opremo Echo-IP. Le-ta omogoča razreševanje javnih naslovov v omrežjih NAT, ki so potrebni za delovanje OpenSPA. Programsko opremo OpenSPA smo nadaljno izboljšali z razvojem lastne implementacije SDP. Programska oprema OpenSDP omogoča široko uporabo koncepta SPA v večjih omrežjih. Funkcionalnosti, ki jih ponuja, so zanimive za omrežja v večjih podjetjih in varnosto občutljive storitve.

Končni rezultat diplomskega dela je programska oprema, ki lahko zaščiti poljubno omrežno storitev tako, da je storitev omrežno skrita. To smo tudi demonstrirali v zadnjem poglavju, kjer smo testirali programsko opremo in predstavili primer uporabe.

Razvita programska oprema predstavlja prvo različico zasnovanega protokola. Nadaljnji tehnološki razvoj lahko še v bistveni meri pripomore k povečani varnosti storitev. Da bi prepričali potencialne uporabnike o visoki

uporabnosti programske opreme, mora biti le-ta čim bolj prijazna do uporabnika. S tem ciljem so bile nakazane dodatne možne izboljšave programske opreme OpenSPA in OpenSDP. Omenjene dodatne izboljšave bo moč realizirati v nadaljnjih raziskavah ter tako vidno izboljšati tako varnost kot tudi enostavnost uporabe programske opreme.

Za podporo programske opreme na mobilnih napravah je potreben nadaljnji razvoj. V današnjem času mobilnosti je podpora mobilnih naprav nujna. Slednje predstavlja dodatni zanimiv izziv. Mobilne naprave pogosto prehajajo med različnimi brezžičnimi in mobilnimi omrežji. Zato so tovrstne naprave naravnost idealne za uporabo programske opreme SDP.

Literatura

[1] Single packet authorization: The fwknop approach. Dosegljivo: http://www.cipherdyne.org/fwknop/docs/design-decisions.html, 2012. [Dostopano: 6. 8. 2018].

[2] Add full ipv6 support to fwknop. Dosegljivo: https://github.com/mrash/fwknop/issues/1, 2018. [Dostopano: 28. 6. 2018].

[3] CSA Software Defined Perimeter Working Group. Dosegljivo: https://cloudsecurityalliance.org/group/software-defined-perimeter, 2018. [Dostopano: 23. 8. 2018].

[4] Don't Block the Event Loop (or the Worker Pool). Dosegljivo: https://nodejs.org/en/docs/guides/dont-block-the-event-loop/, 2018. [Dostopano: 23. 8. 2018].

[5] fwknop - single packet authorization. Dosegljivo: https://github.com/mrash/fwknop, 2018. [Dostopano: 1. 9. 2018].

[6] SDP Specification 1.0). Dosegljivo: https://downloads.cloudsecurityalliance.org/initiatives/sdp/SDP_Specification_1.0.pdf, 2018. [Dostopano: 23. 8. 2018].

[7] SPA Analysis and Recommendations by Jonathan Bennett and Michael Rash. Dosegljivo: https://basecamp.com/1825565/projects/4644250/documents/9583047, 2018. [Dostopano: 23. 8. 2018, vpogled možen s prijavo.].

[8] Gilberto Bertin. Xdp in practice: integrating xdp into our ddos mitigation pipeline. In *Technical Conference on Linux Networking, Netdev*, volume 2, 2017.

[9] Simon Blake-Wilson, Nelson Bolyard, Vipul Gupta, Chris Hawk, and Bodo Moeller. Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls). RFC 4492, RFC Editor, May 2006.

[10] T. Chomsiri, X. He, P. Nanda, and Z. Tan. An improvement of tree-rule firewall for a large network: Supporting large rule size and low delay. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 178–184, Aug 2016.

[11] Reinderd Gordon Nathan deGraaf. Enhancing firewalls: Conveying user and application identification to network firewalls. 2007.

[12] IANA. Protocol numbers, Jul 2017. [Dostopano: 6. 8. 2018].

[13] Lo Janet. Whitelisting for cyber security: What it means for consumers. Technical report, Canadian Office of Consumer Affairs, 2010.

[14] Kayvan Karimi, Arash Ahmadi, Mahmood Ahmadi, and Bahram Bahrambeigy. Acceleration of iptables linux packet filtering using gpgpu. In *Symposium on Computer Science and Software Engineering (CSSE)*, 2013.

[15] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 2012.

[16] Jiun-Hau Liew, Shirly Lee, Ivy Ong, Hoon-Jae Lee, and Hyotaek Lim. One-time knocking framework using spa and ipsec. In *2010 2nd International Conference on Education Technology and Computer*, volume 5, pages V5–209–V5–213, June 2010.

[17] R. Mathur, S. Agarwal, and V. Sharma. Solving security issues in mobile computing using cryptography techniques — a survey. In *International*

Conference on Computing, Communication Automation, pages 492–497, May 2015.

[18] David M'Raihi, Mihir Bellare, Frank Hoornaert, David Naccache, and Ohad Ranen. Hotp: An hmac-based one-time password algorithm. RFC 4226, RFC Editor, December 2005.

[19] Michael Rash. *Linux firewalls: attack detection and response with iptables, psad, and fwsnort*. No Starch Press, 555 De Haro Street, Suite 250, San Francisco, CA 94107, USA, 1st edition, 2007.

[20] Michael Rash. Single packet authorization. *Linux Journal*, 2007(156):1, Apr 2007.

[21] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, Upper Saddle River, New Jersey 07458, USA, 5th edition, 2010.

[22] Martin R. Stytz. Considering defense in depth for software applications. *IEEE Security Privacy*, 2(1):72–75, Jan 2004.