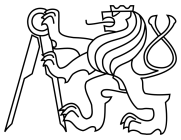
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Kryptograficky bezpečné metody port knocking a single packet authorization
Student:	Bc. Petr Klejch
Vedoucí:	Mgr. Rudolf Bohumil Blažek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové systémy a sítě
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Prostudujte a popište metodologii komunikace mezi hostiteli pomocí uzavřených síťových portů s cílem autorizace a změn konfigurace firewallu. Zaměřte se na kryptografickou bezpečnost vybraných implementací. Diskutujte rozdíly mezi metodami port knockingu a single packet authorization. Navrhněte, implementujte a otestujte kryptograficky bezpečnou metodu pro bezpečné připojení na služby systému MS Windows tak, aby její použití bylo co nejvíce transparentní pro uživatele. Otestujte vytvořený nástroj na vlastním počítači a ověřte jeho praktickou použitelnost.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 12. listopadu 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Diplomová práce

Kryptograficky bezpečné metody port knocking a single packet authorization

Bc. Petr Klejch

Vedoucí práce: Mgr. Rudolf Bohumil Blažek, Ph.D.

9. května 2017

Poděkování

Děkuji vedoucímu diplomové práce Mgr. Rudolfu Bohumilu Blažkovi, Ph.D. za připomínky a rady. Rodině a přátelům za podporu během studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. května 2017 .....

Odkaz na tuto práci

Klejch, Petr. *Kryptograficky bezpečné metody port knocking a single packet authorization.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá analýzou a srovnáním existující řešení pro komunikaci mezi hostiteli pomocí filtrovaných síťových portů s cílem ověření identity a provedení změn konfigurace firewallu. V této práci jsou diskutovány a porovnány dva koncepty – port knocking a single packet authorization. Na základě analýzy je navrhnut a implementován kryptograficky bezpečný nástroj, který umožňuje bezpečné připojení na služby systému operačního systému MS Windows.

Klíčová slova port knocking, single packet authorization, firewall, MS Windows

Abstract

This thesis analyses and compares existing solutions for communication between hosts using filtered network ports to verify identity of clients and change configuration of a firewall. Two approaches are discussed and compared – port knocking and single packet authorization. Based on the analysis it is designed and implemented cryptographically secure tool, which allows secure connection to services of MS Windows operating system.

Keywords port knocking, single packet authorization, firewall, MS Windows

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Firewall je jedním ze základních prvků počítačové síťové bezpečnosti a pomáhá chránit počítač či síť před celou řadou síťových útoků. Použití firewallu je nezbytné pro celou řadu kritických služeb, někdy však není snadné odlišit oprávněného uživatele od případného útočníka.

Koncept port knocking předpokládá, že si oprávněný uživatel dohodne s druhou komunikující stranou (obvykle serverem) tajnou sekvenci čísel. Po odeslání této tajné sekvence čísel, druhá strana tuto sekvenci rozpozná a upraví svoji konfiguraci firewallu ve prospěch uživatele. Uživatel tedy získá přístup ke kritické službě.

V této diplomové práci jsou nejprve v kapitole „Analýza“ analyzovány koncepty port knocking a single packet authorization a následně jsou porovnány jejich bezpečnostní aspekty. V další části této kapitoly jsou analyzovány a porovnány různé implementace těchto konceptů.

V kapitolách „Návrh“ a „Implementace“ je na základě analýzy existujících řešení navrhnut a implementován nástroj, který pomocí konceptu single packet authorization umožňuje vzdálenou autentizaci s cílem změn konfigurace firewallu v prostředí operačního systému MS Windows.

Hlavními výhodami implementovaného nástroje je použití kryptograficky bezpečné metody single packet authorization a automatické zasílání autentizačních paketů při detekci odchozí komunikace se serverem. Nástroj lze navíc nastavit tak, aby byl zpětně kompatibilní s existujícím řešením knockknock.

V poslední kapitole „Testování“ je implementovaný nástroj otestován v několika různých sítích a na různých verzích operačního systému MS Windows.

Kapitola 1

Analýza

1.1 Síťové modely a firewally

1.1.1 Síťové modely

Jelikož se velká část této práce týká síťové komunikace a jejího filtrování, budeme se nejprve zabývat síťovými modely, vrstvami a principu fungování firewallu.

Síťová architektura je popsána pomocí síťových modelů, tyto síťové modely jsou rozloženy do vrstev. Rozložení architektury do vrstev umožňuje nezávislost implementace jednotlivých vrstev a tedy poskytuje možnost vyměňovat protokoly v rámci vrstvy za jiné a to stále za zachování funkčnosti celé architektury.

Prvním diskutovaným síťovým modelem je referenční model ISO/OSI [4], který se skládá ze sedmi vrstev. V tab. 1.1 (sloupec vlevo) lze vidět jednotlivé vrstvy modelu. Tento model se v praxi příliš neujal [5], ale používá se pro teoretický popis síťové architektury.

Dalším síťovým modelem je TCP/IP [6], tento model má čtyři vrstvy. Jak lze vidět v tab. 1.1 (uprostřed), tento model sloučil první dvě vrstvy ISO/OSI modelu (fyzická a linková vrstva) do jedné vrstvy, která se nazývá vrstva síťového rozhraní. Dalším rozdílem je vypuštění prezenční a relační vrstvy, odpovědnost těchto vypuštěných vrstev je přenechána na aplikační vrstvě a z části na vrstvě transportní.

Dalším poměrně často používaným modelem je pětivrstvý model. Jak lze vidět z tab. 1.1 (vpravo), tento model vychází z TCP/IP modelu, ale rozdělil sloučenou vrstvu síťového rozhraní na dvě vrstvy – fyzickou a linkovou.

V této práci bude věnována pozornost zejména síťové a transportní vrstvě, tedy třetí a čtvrté vrstvě referenčního ISO/OSI modelu (tento model bude v této práci použit pro popis síťové architektury). Konkrétně diskutované protokoly budou IP, TCP a UDP.

Nejrozšířenějším protokolem síťové vrstvy je IP ve verzi 4 (IPv4) [7]. Hla-

<table>
<tr><td>aplikační vrstva</td><td rowspan="2">aplikační vrstva</td><td rowspan="2">aplikační vrstva</td></tr>
<tr><td>prezenční vrstva</td></tr>
<tr><td>relační vrstva</td><td rowspan="2">transportní vrstva</td><td rowspan="2">transportní vrstva</td></tr>
<tr><td>transportní vrstva</td></tr>
<tr><td>síťová vrstva</td><td>síťová vrstva</td><td>síťová vrstva</td></tr>
<tr><td>linková vrstva</td><td rowspan="2">vrstva síťového rozhraní</td><td>linková vrstva</td></tr>
<tr><td>fyzická vrstva</td><td>fyzická vrstva</td></tr>
</table>

Tabulka 1.1: Různé síťové modely. Zleva: Referenční model ISO/OSI, TCP/IP model a pětivrstvý model.

vičku protokolu IPv4 si lze prohlédnout v tab. 1.2. Protokol IPv4 je však postupně nahrazován novějším protokolem IPv6, z důvodu vyčerpání IPv4 adres. Jelikož jsou oba protokoly navzájem nekompatibilní, byly vyvinuty určité mechanismy, které usnadňují přechod z IPv4 na IPv6. Mezi tyto mechanismy dle [8] patří například: dvojitý zásobník, IPv6 v IPv4 tunelu, NAT překladač protokolu a další. Dle [9] přistupuje ke službám firmy Google přes IPv6 pouze 17 % uživatelů, z tohoto důvodu bude v následující práci diskutován pouze protokol IPv4.

<table>
<tr><td colspan="32">0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</td></tr>
<tr><td colspan="4">verze</td><td colspan="4">velikost hlavičky</td><td colspan="8">typ služby</td><td colspan="16">celková délka</td></tr>
<tr><td colspan="16">identifikace</td><td>0</td><td>DF</td><td>MF</td><td colspan="13">offset fragmentu</td></tr>
<tr><td colspan="8">TTL</td><td colspan="8">číslo protokolu</td><td colspan="16">kontrolní součet hlavičky</td></tr>
<tr><td colspan="32">zdrojová adresa</td></tr>
<tr><td colspan="32">cílová adresa</td></tr>
<tr><td colspan="32">rozšířená nepovinná nastavení</td></tr>
</table>

Tabulka 1.2: Hlavička IPv4.

Hlavička protokolu IPv4 (viz tab. 1.2) má velikost 20 až 60 bajtů a obsahuje 13 povinných a jednu nepovinnou položku. Obsah položek je dle [7] následující:

Verze Číslo verze IP. V případě IPv4 je tato hodnota vždy čtyři.

Velikost hlavičky Délka hlavičky ve 32bitových slovech. Minimální validní hodnota je 5, maximální 15.

Typ služby Pomocí této položky lze nastavit prioritu datagramu. Datagramy s vyšší prioritou jsou v některých sítích zpracovávány přednostně.

Celková délka Délka celého datagramu v bajtech (včetně hlavičky a samotného obsahu). Maximální možná velikost je tedy 65 535 bajtů.

Identifikace Tato položka se využívá při defragmentaci datagramů. Příjemce fragmentovaného datagramu podle tohoto identifikátoru pozná, které fragmenty patří logicky k sobě.

Příznaky Příznaky tvoří celkem tři bity, první je rezervován a je nastaven vždy na hodnotu nula. Další příznak (Don't Fragment) ovlivňuje, zdali může být datagram fragmentován. Poslední příznak (More Fragments) indikuje další příchozí fragmenty.

Offset fragmentu Hodnotou této položky je relativní pozice fragmentu vůči počátku původního nefragmentovaného datagramu. Jednotka této položky je osm bajtů.

TTL Tato položka určuje životnost datagramu, každý síťový prvek, který zpracovává datagram sníží tuto hodnotu minimálně o jedna. Pokud je hodnota TTL nula, musí být datagram zahozen.

Číslo protokolu Tato položka specifikuje typ protokolu použitý ve vyšší vrstvě (např. TCP, UDP, ICMP, atd.).

Kontrolní součet hlavičky Tato položka obsahuje kontrolní součet hlavičky. Pokud je tato položka nastavena na nulu, znamená to, že má být kontrolní součet dopočítán.

Zdrojová adresa Tato položka obsahuje zdrojovou IPv4 adresu.

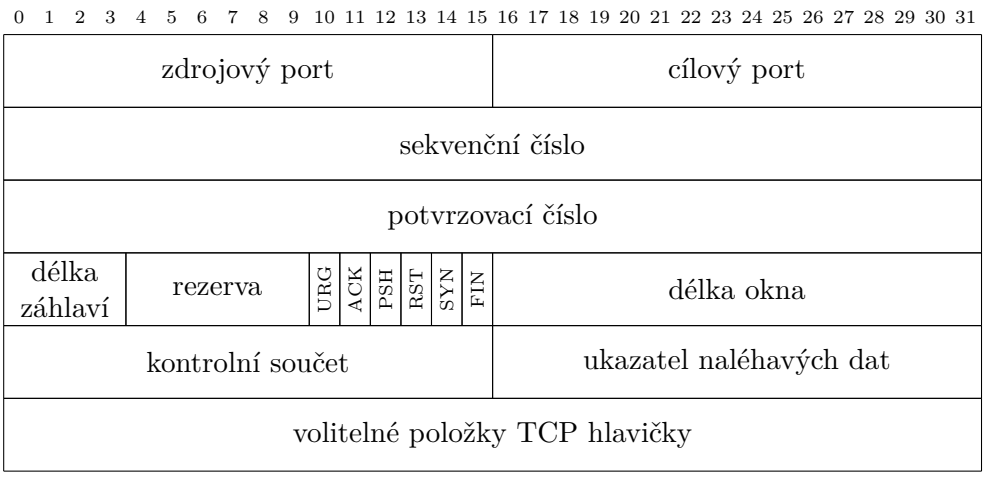
Cílová adresa Tato položka obsahuje cílovou IPv4 adresu.

Rozšířená nepovinná nastavení Tato položka obsahuje volitelné položky. Tato položka musí být zarovnána na délku dělitelnou čtyřmi bajty. Jako zarovnání se používají nuly.

Nad síťovou vrstvou je vrstva transportní (viz tab. 1.1 vlevo). Mezi nejrozšířenější protokoly transportní vrstvy patří TCP a UDP. TCP na rozdíl od UDP zachovává pořadí paketů, umí se vypořádat s duplicitními pakety a zaručuje spolehlivý přenos všech paketů.

Hlavička TCP protokolu (viz tab. 1.3) má velikost 20 až 60 bajtů, obsahuje deset povinných a jednu nepovinnou položku. Obsah položek je dle [10] následující:

Zdrojový port Číslo zdrojového portu.



Tabulka 1.3: Hlavička TCP.

Cílový port Číslo cílového portu.

Sekvenční číslo Pokud není nastaven příznak SYN, tato položka označuje aktuální číslo sekvence prvního bajtu dat v tomto segmentu. Pokud je nastaven pouze příznak SYN, tato položka označuje počáteční sekvenční číslo a tím definuje, že první bajt dat má číslo sekvence rovno hodnotě počátečního sekvenčního čísla+1.

Potvrzovací číslo Pokud je nastaven příznak ACK, tato položka označuje hodnotu dalšího sekvenčního čísla, kterou odesilatel tohoto segmentu očekává od druhé komunikující strany. Jakmile je spojení úspěšně navázáno, potvrzovací číslo (včetně příznaku ACK) je vždy odesláno v každém segmentu.

Délka záhlaví Hodnota této položky určuje, kde začínají samotná data za TCP hlavičkou.

Rezerva Hodnota této položky je nula. Tato položka je vyhrazená pro možné budoucí použití. První bit má již své experimentální využití dle [11] a další dva bity mají navržené použití dle [12].

Příznaky TCP hlavička obsahuje celkem šest příznaků. Příznak URG značí, že položka ukazatele naléhavých dat má význam, příznak ACK značí, že položka potvrzovacího čísla má význam. PSH příznak značí, že odesilatel odeslal všechna data a příjemce má doručená data ihned doručit do aplikace. RST příznak značí, že spojení bylo resetováno a informuje druhou stranu o okamžitém ukončení spojení. RST příznak se posílá např. v případě, že přišel požadavek na port, na kterém neposlouchá žádná služba.

SYN příznak značí snahu o synchronizaci sekvenčních čísel. Příznak FIN značí požadavek na ukončení spojení, na rozdíl od příznaku RST však odesilatel příznaku FIN stále čeká na potvrzení od druhé strany.

Délka okna Počet bajtů, které je odesilatel schopen přijmout.

Kontrolní součet Kontrolní součet tzv. pseudo-hlavičky, což je TCP hlavička včetně některých položek z protokolu nižší vrstvy.

Ukazatel naléhavých dat Pokud je nastaven příznak URG, tato položka označuje bajt relativně vůči sekvenčnímu číslu, kde začínají naléhavá data.

Volitelné nastavení Tato položka obsahuje volitelné položky. Tato položka musí být zarovnána na délku dělitelnou čtyřmi bajty. Jako zarovnání se používají nuly.

Při využití TCP se musí před samotnou výměnnou dat provést tzv. trojcestné potřesení rukou (three-way handshake). To se skládá ze tří následujících kroků:

1. Klient odešle serveru segment s příznakem SYN, který obsahuje náhodně vygenerované sekvenční číslo x.
2. Server na tuto žádost odpoví segmentem, který obsahuje příznaky SYN a ACK, náhodně vygenerované sekvenční číslo y a potvrzovací číslo $x+1$.
3. Klient nakonec odpoví segmentem s příznakem ACK, který obsahuje sekvenční číslo nastavené na $x+1$ a potvrzovací číslo nastavené na $y+1$.

V tuto chvíli je spojení navázáno a oba účastníci spojení mohou začínat posílat data. Při výměně dat si oba účastníci pamatují čítače pro potvrzovací a sekvenční číslo. Příklad takové komunikace vypadá následovně:

1. Klient odešle požadavek na server o délce 76 bajtů, odešle tedy segment, který bude mít nastaven jako sekvenční číslo hodnotu $x+1$ a potvrzovací číslo hodnotu $y + 1$.
2. Na tento požadavek odpoví server potvrzovacím TCP segmentem s příznakem ACK, který bude mít hodnotu sekvenčního čísla $y + 1$ a potvrzovacího čísla $x + (1 + 76)$.
3. Následně server odešle klientu odpověď na jeho předchozí požadavek o délce 285 bajtů. Sekvenční číslo v tomto segmentu tedy bude mít hodnotu $y + 1$ a potvrzovací číslo hodnotu $x + (1 + 76)$.
4. Klient po přijetí odpovědi na svůj požadavek odešle serveru potvrzovací TCP segment s příznakem ACK, který bude mít hodnotu sekvenčního čísla $x + (1 + 76)$ a potvrzovacího čísla $y + (1 + 285)$.

Sekvenční číslo tedy na každé komunikující straně označuje počet úspěšně odeslaných bajtů dat (relativně vůči vygenerovanému počátečnímu sekvenčnímu číslu + 1), potvrzovací číslo označuje počet úspěšně přijatých bajtů dat (opět relativně vůči náhodně vygenerovanému sekvenčnímu číslu protistrany + 1).

Dalším diskutovaným protokolem transportní vrstvy je UDP [13]. Tento protokol na rozdíl od TCP negarantuje úspěšné doručení paketů, zachování pořadí či neexistenci duplicitních paketů, je však velmi jednoduchý a má tak velmi malou režii. Tato vlastnost se dle [14] hodí do aplikací, kde je potřeba rychlost a jednoduchost, tedy např. streamování videa, VoIP a u protokolů DNS, SNMP, DHCP a RIP.

Hlavičku UDP si lze prohlédnout v tab. 1.4.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
zdrojový port	cílový port
délka dat	kontrolní součet

Tabulka 1.4: Hlavička UDP.

UDP hlavička má fixní velikost osm bajtů, obsahuje dvě volitelné a dvě povinné položky. Obsah jednotlivých položek je dle [13] následující:

Zdrojový port Číslo zdrojového portu. Tato položka je nepovinná, pokud není zdrojový port nastaven, hodnota této položky je nula.

Cílový port Číslo cílového portu.

Délka dat Celkový počet dat včetně hlavičky v bajtech, minimální možná hodnota je osm bajtů (tedy pouze UDP hlavička).

Kontrolní součet Tato položka obsahuje kontrolní součet a je nepovinná při použití s IPv4 (pro IPv6 je dle [15] tato položka povinná), pokud tato položka není použita, její hodnota by měla být nula. Kontrolní součet se počítá z tzv. pseudo-hlavičky, která obsahuje položky z IP hlavičky (zdrojová a cílová adresa, typ použitého protokolu a délka UDP hlavičky a samotných dat).

1.1.2 Firewally

Následující část bude krátce věnována principu fungování firewallu. Dle [16] existuje několik druhů bezpečnostních bran, např. (stavové) paketové filtry, aplikační brány, atd. V této práci budou diskutovány pouze stavové paketové filtry.

Paketový filtr provádí filtrování paketů na základě nakonfigurovaných pravidel. Tato pravidla jednoduše specifikují, z které zdrojové adresy a portu na jakou cílovou adresu a port může být paket poslán. Tato pravidla se vyhodnocují od nejobecnějších po konkrétnější, obecná pravidla tak mohou být dále upravena více konkrétními pravidly.

Navíc stavový paketový filtr udržuje stav spojení, a proto může o aktuálním paketu rozhodovat také stav spojení. Stavový paketový filtr tedy operuje nejčastěji na třetí a čtvrté síťové vrstvě referenčního modelu ISO/OSI (síťová a transportní vrstva).

Síťové spojení je definováno jako pětice: zdrojová a cílová adresa, zdrojový a cílový port a protokol. UDP je ze své definice (na rozdíl od TCP) bezstavový protokol, avšak i spojení využívající UDP mohou udržovat stav, tento stav je udržován do vypršení časového okna. Pokud tedy klient odešle UDP datagram na server ze zdrojového portu 54 321 na port 53, vrácená odpověď serveru z portu 53 na port 54 321 klienta je brána v rámci jedné relace.

Uvažujeme-li například následující konfiguraci stavového paketového filtru.

```
povol veškeré odchozí pakety
zakaž veškeré příchozí pakety
povol příchozí pakety v rámci již navázaného spojení
povol příchozí pakety s cílovým portem 80
```

Stavový paketový filtr povolí všechny odchozí pakety a příchozí pakety s cílovým portem 80. Navíc pokud bylo spojení navázáno zevnitř, příchozí odpovědi v rámci tohoto spojení jsou také povoleny.

Síťové porty hostitele (např. serveru) se proto mohou pro druhou komunikující stranu jevit v několika stavech. Např. nástroj nmap dle [17] rozlišuje šest stavů, ve kterých se port může nacházet. Tři základní stavy, které rozlišuje nástroj nmap jsou dle [17] následující:

Otevřený port Otevřený port přijímá TCP spojení nebo UDP datagramy. Najití otevřených portů je často hlavním účelem techniky port scanning. Otevřený port tedy poskytuje prostor k potencionálnímu útoku.

Uzavřený port Port je dostupný, ale žádná aplikace na tomto portu nenaslouchá. Takové porty může být vhodné později znovu zkontrolovat, zdali nejsou již otevřené. Při pokusu o zaslání UDP datagramu na tento port se vrátí ICMP paket typ 3, kód 3, který značí, že port je nedostupný. Při zaslání TCP segmentu se vrací TCP odpověď s příznakem RST.

Filtrovaný port Nelze určit zdali je port otevřený, protože paketový filtr nepovolil příchozí spojení. Nejčastěji není na příchozí spojení dána žádná odpověď a útočníkovi tak tento port neposkytne žádnou informaci. Někdy může být firewall nakonfigurován tak, aby vracel ICMP paket typ 3, kód 13, který značí, že port je filtrován.

Příkladem stavového paketového filtru je např. Windows Firewall v operačním systému MS Windows a iptables v operačním systému Linux.

1.2 Port knocking

Koncept port knocking vychází z předpokladu, že porty serveru jsou filtrovány z důvodu bezpečnosti. Jelikož jsou porty filtrovány, firewall neumožní jakékoliv příchozí spojení. Ačkoliv lze firewall nakonfigurovat tak, aby povolil příchozí spojení z určitých zdrojových IP adres, tato konfigurace není vždy dostačující. Pokud by se uživatel nacházel se svým osobním počítačem v nějaké jiné síti, jeho zdrojová adresa by byla jiná než ta, která je nakonfigurována ve firewallu.

Vyřešit tento problém se pokusil koncept port knocking. Český překlad konceptu port knocking znamená „klepání na porty“, což popisuje ideu tohoto konceptu. Předpokládejme, že klient i server mají předem domluvenou tajnou sekvenci čísel portů. Pokud klient „zaklepe“ na tyto předem domluvené porty ve správném pořadí, server tuto sekvenci rozpozná a upraví konfiguraci firewallu tak, aby byl předem domluvený port otevřený pro IP adresu klienta.

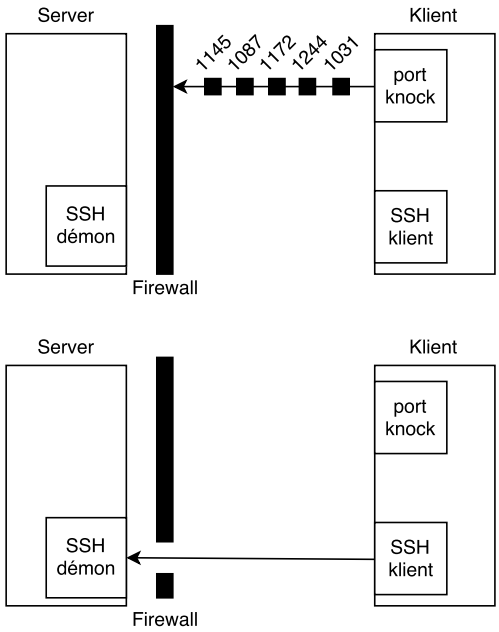
Samotný proces „klepání“ může mít různou podobu, může se například jednat o sérii TCP paketů (s různými příznaky), UDP paketů či jejich kombinace. Předem domluvená sekvence je nejčastěji definována pomocí cílových portů těchto paketů (některé implementace používají zdrojové porty).

Způsob detekce a rozpoznání sekvence na straně serveru, také není nijak pevně definován. Může se například jednat o čtení souboru s událostmi (logovací soubor), který vytváří firewall nebo čtení hlaviček příchozích paketů pomocí nějakého nástroje pro odposlouchávání síťové komunikace. Některé implementace přímo poslouchají na definovaném UDP či TCP portu a čtou zdrojové porty příchozích paketů.

Ukázku celého konceptu lze vidět na obr. 1.1. Klient se chce připojit pomocí protokolu SSH na server, jenže port 22 (výchozí port pro SSH, na kterém poslouchá SSH démon) je na straně serveru filtrovaný. Klient tedy odešle sekvenci paketů, které mají nastavený jako cílový port čísla 1145, 1084, 1172, 1244 a 1031. Server tuto sekvenci rozpozná a upraví konfiguraci firewallu tak, aby byl otevřen port 22 pro IP adresu klienta.

Tento jednoduchý koncept má však několik slabin. Například tato autentizační sekvence může být případným útočníkem lehce odposlechnuta. Útočník pak může tuto sekvenci snadno přehrát (tzv. replay attack, útok přehráním), server by poté sekvenci úspěšně rozpoznal a upravil by konfiguraci firewallu ve prospěch útočníka.

Tomuto útoku se lze bránit několika způsoby. Některé implementace například vyhradí určitý rozsah portů pro komunikaci skrz filtrované porty. Předpokládejme, že server vyhradil rozsah portů 50 000 až 50 255 pro účely autentizace klienta. Klient pak může odeslat paket na 256 různých portů a tedy v zásadě odeslat jeden bajt zprávy. Klient následně pomocí zasílání paketů na



Obrázek 1.1: Diagram konceptu port knocking, obrázek převzat z [1].

tyto předem definované porty může serveru odeslat zprávu, ačkoliv jsou tyto porty filtrované.

Obsahem této zprávy může být například port, který chce klient otevřít a navíc nějaká proměnlivá část, například časové razítko či náhodné číslo. Jelikož zpráva obsahuje proměnlivou část, autentizační sekvence bude mít pokaždé jinou podobu.

Obsah zprávy může být navíc zašifrován (např. symetrickou šifrou) a opatřen autentizačním kódem. Jelikož pouze klient a server zná tajný klíč použitý k zašifrování zprávy a díky použití autentizačního kódu, server dokáže ověřit integritu a autenticitu přijaté zprávy.

Proti útoku přehráním se tedy může server bránit například pamatováním přijatých sekvencí a zneplatněním již vícekrát přijaté sekvence (jelikož díky náhodnému číslu bude každá sekvence jiná), či ověřením časového razítka v těle paketu (časové razítko není starší než definovaná hranice).

Tento přístup má však slabinu, jelikož se k přenosu zprávy používají TCP SYN či UDP pakety, na které klient nedostává žádné odpovědi, nemůže si být jist, zdali pakety reprezentující bajty zprávy vůbec dorazily či dorazily ve správném pořadí. Tuto slabinu lze z části eliminovat rozšířením rozsahu definovaných portů určených k autentizaci klienta. Pokud by server vyhradil celý rozsah, tedy porty 0 až 65 535, klient by mohl odeslat v jednom paketu až dva bajty zprávy. To by vedlo ke snížení počtu odesílaných paketů.

Evolucí konceptu port knocking je koncept single packet authorization (SPA). Hlavní idea zůstává stejná, tedy autentizovat se na straně serveru, který následně upraví konfiguraci firewallu ve prospěch klienta. Zatímco u konceptu port knocking byla autentizace provedena pomocí sekvence paketů, u konceptu single packet authorization se autentizace provádí pomocí jediného paketu.

Ačkoliv se koncept jmenuje single packet authorization, tedy „autorizace jediným paketem“ cílem tohoto konceptu je především autentizace na straně serveru. Celý proces se většinou skládá ze tří kroků: identifikace (zjištění totožnosti klienta), autentizace (ověření totožnosti klienta) a autorizace (ověření, že klient má oprávnění provést danou akci). Různé implementace pak provádí různé kroky tohoto procesu. Některé implementace provádí pouze identifikaci a autentizaci, tedy zjistí, který klient se chce připojit a poté ověří jeho totožnost. Některé implementace provádí i autorizaci, jelikož různí uživatelé mohou provádět různé akce (např. žádat o otevření různých portů). V následujícím textu se bude o paketu, který odesílá klient serveru za účelem upravení konfigurace firewallu, mluvit jako o autentizačním paketu.

Koncept SPA spočívá v odeslání jediného paketu, který obsahuje informace, které autentizují klienta. Tento paket většinou obsahuje zprávu, ve které se nachází požadavek na otevření určitého portu. Stejně jako v případě konceptu port knocking může být obsah zprávy zašifrován a opatřen autentizačním kódem. Obsahem zprávy také může být opět nějaká proměnlivá část, například časové razítko či náhodné číslo. Z tohoto důvodu bude znemožněn útok přehráním.

Stejně jako v případě konceptu port knocking není ani zde přesně definováno, jak má takový paket vypadat. Některé implementace používají UDP paket a zprávu vloží do těla paketu, jiné implementace rozdělí zprávu na několik částí a ty vloží do různých položek hlavičky TCP SYN paketu.

Některé implementace se již vzdálily od původního konceptu SPA a používají několik paketů k provedení celého autentizačního mechanismu (včetně odpovědí ze strany serveru). Tyto implementace však nebudou v této práci diskutovány.

1.3 Srovnání bezpečnostních aspektů port knocking a single packet authorization

Jak již bylo zmíněno výše, nevýhodou konceptu port knocking je fakt, že se pro úspěšnou autentizaci musí zaslat více paketů. Tento fakt nejenom zvětšuje šanci neúspěšné autentizace z důvodu ztráty či záměny pořadí odeslaných paketů, ale také zvětšuje případný prostor k útoku.

Koncept port knocking je dále náchylný vůči útoku odepření služby (denial of service), případný útočník by mohl zasílat pakety s podvrženou zdrojovou IP adresou a narušovat tím legitimní klientovu autentizační sekvenci.

Při použití statických sekvencí je port knocking náchylný k útoku přehráním. Stejnou zranitelností však může trpět i SPA, pokud není implementován dodatečný autentizační mechanismus.

U obou řešení lze zajistit důvěrnost, integritu a autenticitu přenášené zprávy použitím šifry a autentizačního kódu. Bezpečnost obou konceptů velmi závisí na použití správných kryptografických primitiv.

Obě řešení mohou být náchylná vůči útoku Man-in-the-middle (MITM), případný útočník může podvrhnout zdrojovou IP adresu autentizačního paketu či zdrojovou IP adresu paketů autentizační sekvence. Jelikož by server tento autentizační paket či sekvenci úspěšně rozpoznal, upravil by konfiguraci firewallu ve prospěch útočníka.

Některé implementace konceptu SPA mohou být zranitelné vůči útoku buffer overflow, především pokud čtou obsah těla příchozího paketu bez pečlivé kontroly.

Bezpečnost obou řešení může být ohrožena při použití síťového překladu adres (NAT). Pokud se legitimní klient i útočník nachází ve stejné privátní síti se síťovým překladem, oba dva vystupují z pohledu serveru pod stejnou IP adresou. Pokud se klient úspěšně autentizuje, port bude otevřený i pro útočníka.

Ačkoliv žádný z obou konceptů není dokonalý a každé řešení má své zranitelnosti, při selhání těchto konceptů je útočníkovi povolen přístup k službě, to však nemusí znamenat kompromitaci bezpečnosti, jelikož případný útočník by musel prolomit další autentizační mechanismy.

Port knocking i single packet authorization tedy fungují jako dodatečná bezpečnostní vrstva nad existujícími bezpečnostními mechanismy.

1.4 Analýza existujících řešení

V této části budou diskutovány existující implementace konceptu port knocking či single packet authorization. Podrobněji budou analyzovány tři nejrozšířenější implementace: knockd, knockknock a fwknopd.

Ostatní implementace nebudou již analyzovány s takovou podrobností. Důvodem je buď malá rozšířenost nástroje, nedostatečná kryptografická bezpečnost či fakt, že pouze znovu implementují již implementovaný koncept (např. do jiného jazyka nebo na jinou platformu).

1.4.1 knockd

Nástroj knockd [18] je jedna z nejrozšířenějších implementací konceptu port knocking. Hned několik velkých webů, např. [19], [20] a [21], které poskytují návody a rady týkajících se konfigurace serverů, zmiňují právě tuto implementaci. Tento fakt je pravděpodobně způsoben tím, že je tento program dostupný i ve formě instalačního balíčku a poskytuje tak snadnou instalaci a odinstalaci.

Instalační balíček je dostupný ve formátu deb pro linuxovou distribuci Debian a dalších odvozených distribucí (Ubuntu, atd.) již od roku 2004. Dále je k dispozici také jako instalační balíček ve formátu rpm, tedy např. pro distribuce Fedora, RHEL, CentOS apod.

Program obsahuje serverovou i klientskou část, obě části (server a nativní klient) jsou napsány v jazyce C. Zatímco serverová část je kompatibilní s unixovými operačními systémy, např. Linux, FreeBSD a Mac OS, existují implementace klientské části i pro operační systém MS Windows a mobilní operační systémy Android a iOS.

1.4.1.1 Princip fungování klientské části

Klientská část je velmi jednoduchá, programu se jako argumenty příkazové řádky předá sekvence TCP a UDP cílových portů a doménové jméno či IP adresa cílového serveru. Program pak vytvoří soket a v závislosti na použitém protokolu zavolá následující funkci:

TCP Soketu je nastaven příznak O_NONBLOCK, tento příznak způsobí, že při pokusu o připojení, program nebude čekat na dokončení pokusu o připojení. Následně program zavolá funkci connect(), která odešle TCP SYN paket a pokusí se o navázání spojení. To však nebude navázáno, protože na cílovém počítači firewall blokuje příchozí pakety.

UDP Program zavolá funkci sendto(soket, „“, 1, ...). Pošle tedy paket o velikosti jednoho bajtu, který obsahuje pouze ukončovací znak „\0“.

Mezi jednotlivými položkami sekvence je možnost specifikovat krátkou prodlevu, aby se eliminovala (či alespoň snížila) možnost, že jednotlivé pakety autentizační sekvence přijdou ve špatném pořadí.

1.4.1.2 Princip fungování serverové části

Program vychází ze základního konceptu port knocking a tedy pouze detekuje předem definované, statické sekvence cílových portů příchozích paketů.

Jako obrana proti útoku přehráním (replay attack) zde však existuje možnost specifikovat soubor, který obsahuje na každém řádku sekvenci portů. Po každé úspěšně přijaté sekvenci portů je řádek označen znakem „#“, který označí sekvenci za použitou. Tímto způsobem lze docílit používání dynamických sekvencí. Je však nutné synchronizovat klienta se serverem, aby klient věděl, jakou sekvenci má použít při příští autentizační sekvenci.

Knockd podporuje transportní protokoly TCP a UDP, TCP je použit jako výchozí. Definované sekvence mohou být libovolnou kombinací TCP a UDP cílových portů. U TCP lze navíc specifikovat konkrétní příznaky (FIN, SYN, RST, PSH, ACK, URG), program pak bude brát v potaz pouze příchozí TCP pakety s těmito příznaky. TCP příznaky lze kombinovat a lze specifikovat, že paket nesmí obsahovat některý z TCP příznaků, aby byl brán v potaz.

Knockd po úspěšné autentizaci nemusí pouze přidat pravidlo do firewallu, které umožní klientovi přístup, ale umí spustit předem definovaný, libovolný skript či program. Při specifikovaní programu či skriptu existují dvě možnosti konfigurace:

1. Specifikuje se pouze příkaz, který se provede po úspěšné autentizaci klienta.

2. Specifikují se dva příkazy (direktivy start_command a stop_command) a navíc časová prodleva (direktiva cmd_timeout). Po úspěšné autentizaci klienta se spustí příkaz v direktivě start_command a po uplynutí časové prodlevy v direktivě cmd_timeout se spustí příkaz v direktivě stop_command.

Při použití druhé možnosti konfigurace, jak lze vidět v ukázce konfiguračního souboru 1.1, lze pomocí jedné sekvence např. přidat pravidlo do firewallu, které umožní přístup klientovi a po uplynutí časové prodlevy bude toto pravidlo odstraněno. Klient mezitím může využít časového okna k připojení např. k SSH serveru a využívat navázaného SSH spojení i po odstranění pravidla z firewallu. To je docíleno tím, že ve výchozí konfiguraci firewallu jsou zakázána pouze nová příchozí spojení (NEW), avšak spojení již vytvořená (ESTABLISHED) jsou povolena.

Konfigurační soubor 1.1: Ukázka konfiguračního souboru programu knockd

```
[options]
      logfile = /var/log/knockd.log

[openccloseSMTP]
      one_time_sequences = /etc/knockd/smtp_sequences
      seq_timeout        = 15
      tcpflags           = fin,!ack
      start_command      = /usr/sbin/iptables -A input -s %IP% -p
  tcp --dport 25 -j ACCEPT
      cmd_timeout        = 5
      stop_command       = /usr/sbin/iptables -D INPUT -s %IP% -p
  tcp --dport 25 -j ACCEPT
```

Pokud není přijat další paket v sekvenci do časového intervalu, který je specifikován v konfiguračním souboru (direktiva seq_timeout), je sekvence prohlášena za neplatnou a klient musí začít s autentizační sekvencí od začátku.

Pro odposlouchávání příchozích paketů se používá knihovna pcap. Výhodou této knihovny oproti samostatné službě, která poslouchá na určitém portu a autentizuje sekvence je fakt, že lze detekovat příchozí pakety, které jsou filtrovány firewallem. Na příchozí pakety tedy nedává server žádnou odpověď.

Před samotným spuštěním odposlouchávací funkce v knihovně pcap musí být nejprve nastaven filtr, který bude určovat, které pakety budou odposlouchávány. Jelikož příchozích paketů může být velké množství, musí být filtr nastaven tak, aby odposlouchával pouze relevantní příchozí pakety. Knockd tedy vytvoří filtr, podle následujících pravidel:

1. Odposlouchávány budou pouze příchozí pakety, tedy pakety, které mají jako cílovou IP adresu některou z adres přiřazenou místnímu počítači.

2. Odposlouchávány budou pouze TCP pakety, které mají jako cílový port nastavený některý z cílových portů, které byly definovány v konfiguračním souboru.

3. Odposlouchávány budou pouze TCP pakety, jejichž příznaky odpovídají nastavení z konfiguračního souboru.

4. Odposlouchávány budou pouze UDP pakety, které mají jako cílový port nastavený některý z cílových portů, které byly definovány v konfiguračním souboru.

1.4.1.3 Analýza bezpečnosti

Tato implementace je velmi jednoduchá a je založena na původním konceptu port knocking. Proti útoku přehráním (replay attack) poskytuje jen velmi malou obranu ve formě souboru s jednorázovými sekvencemi. Využití souboru s jednorázovými sekvencemi nemusí být úplně snadné, jelikož dodávaný nativní klient tento soubor využít neumí. Tento problém by se dal vyřešit jednoduchým skriptem, který by na straně klienta přečetl soubor s jednotlivými sekvencemi a pomocí dodávaného klientského programu, by tuto sekvenci odeslal.

Problém by však nastal v případě, pokud by sekvence nebyla správně autentizována na straně serveru. Tím pádem by mohlo dojít k desynchronizaci aktuální sekvence. Musel by tedy implementován dodatečný mechanismus pro synchronizaci jednorázových sekvencí.

1.4.2 knockknock

Matthew Rosenfeld, nejčastěji však znám pod pseudonymem Moxie Marlinspike, je spoluautor kryptografického protokolu Signal, který byl implementován do aplikací jako WhatsApp [22], Facebook Messenger [23] a Google Allo [24]. Mezi jeho další tvorbu patří nástroj sslstrip, který slouží k automatickému útoku na protokol HTTPS [25], objevil chybu v implementaci Microsoft CryptoAPI [26], která umožňovala skrytý Man-in-the-middle útok na prohlížeč Internet Explorer. Dále objevil chybu v implementacích SSL protokolu týkajících se certifikátů a ukončovacích znaků („\0“) v názvu subjektu

(Common Name) [27] a objevil slabinu v protokolu MS-CHAPv2, která umožní dešifrovat hesla použitá v protokolech PPTP či WPA2 [28].

Autor nástroje knockknock dle [29] zmiňuje, co nechce aby implementace konceptu port knocking dělala. Mezi jeho některé požadavky patří:

- Nástroj, který běží v jádře systému.
- Služba, která poslouchá na konkrétním portu.
- Nástroj, který používá knihovnu pcap a monitoruje každý paket.
- Nástroj, který používá UDP.
- Nástroj, který posílá více jak jeden paket skrz síť.
- Nástroj, který generuje snadno detekovatelný požadavek na otevření portu, který by útočníkovi prozradil, který port bude otevřen.
- Nástroj, který používá nedokonalou kryptografii a nesplňuje ověřitelně vlastnost IND-CCA.

Výsledkem je tedy nástroj, který implementuje koncept single packet authorization (SPA), je dostupný pro operační systém Linux a nevyužívá knihovnu pcap.

Autor dodává program jako Python balíček včetně instalačního skriptu. Obsahem balíčku je skript pro klienta (knocknock), server (knockknock-daemon), skript pro generování profilů (knockknock-genprofile) a skript pro automatické zasílání autentizačních paketů (knockknock-proxy).

V následující části bude popsán princip fungování obou částí nástroje knockknock.

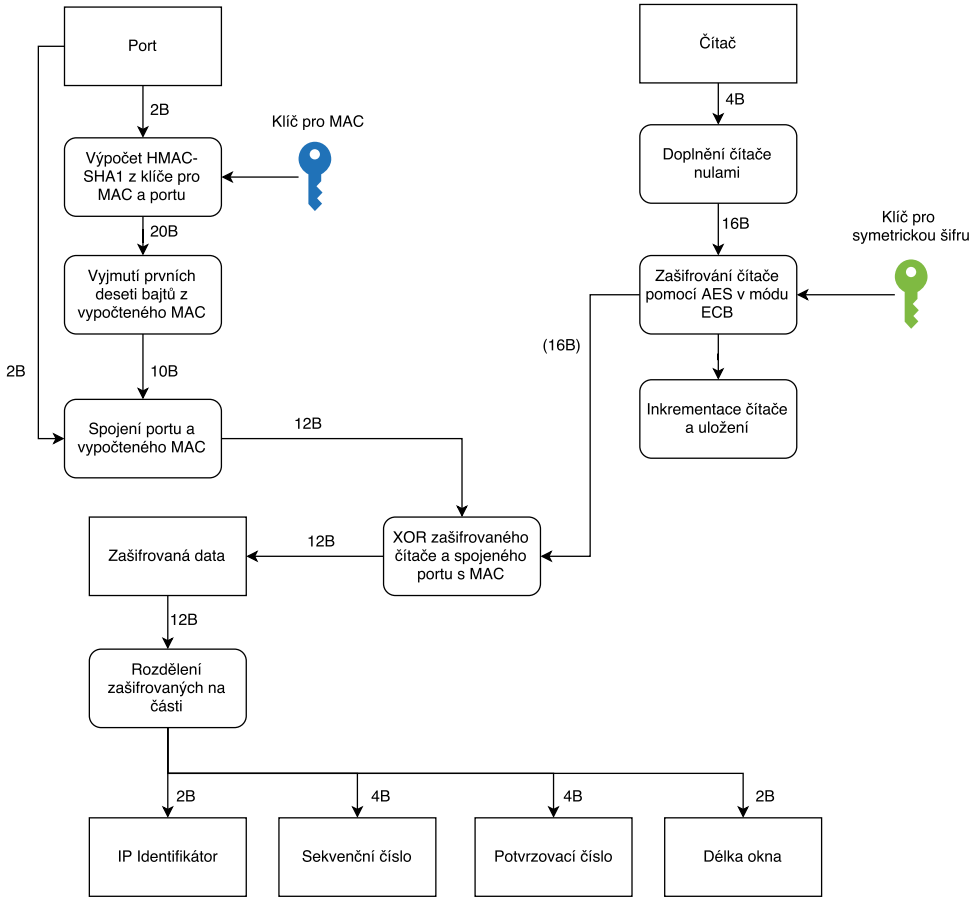
1.4.2.1 Princip fungování klientské části

Nejprve je na serveru vygenerován profil. Profil je soubor, který identifikuje uživatele, který se chce na serveru autentizovat. Při vytváření profilu uživatel specifikuje jméno profilu a port. Port je určen k identifikaci různých uživatelů na serveru, každý uživatel má přiřazen svůj unikátní port. Při přijetí autentizačního paketu server podle cílového portu pozná, který uživatel se chce autentizovat.

Při generování profilu jsou dále vytvořeny tyto soubory: counter (čítač) a dva různé klíče: cipher.key (klíč pro šifrování) a mac.key (klíč pro autentizační kód zprávy). Klíče jsou generovány pomocí pseudonáhodného generátoru /dev/urandom a jsou dlouhé 16 bajtů (128 bitů). Čítač je soubor obsahující číslo, které má výchozí hodnotu nula. Soubory profilu jsou uloženy ve složce /etc/knockknock.d/profiles/<název profilu>/.

Profil (včetně čítače a obou klíčů) je následně bezpečně dopraven na počítač klienta. Na straně klienta jsou soubory profilu zkopírovány do složky /home/<uživatel>/.knockknock/<server>. Server je identifikován pomocí IP adresy nebo doménového jména.

Při odesílání autentizačního paketu se skriptu knockknock předají dvě informace: IP adresa či doménové jméno serveru a port, který má být na serveru otevřen. Skript nalezne odpovídající profil v adresáři /home/<uživatel>/.knockknock/<server> podle doménového jména či IP adresy serveru a načte čítač a oba klíče.



Obrázek 1.2: Diagram šifrování dat pro autentizační paket programu knockknock.

Na obr. 1.2 lze vidět jak je vytvořen autentizační paket. Nejprve se definovaný port, který uživatel specifikoval, převede na bajty reprezentující celé číslo bez znaménka o velikosti dvou bajtů v kódování big-endian (což je pořadí bajtů používané pro síťovou komunikaci). Z tohoto portu a načteného klíče pro

autentizační kód zprávy je vypočten autentizační kód zprávy (HMAC) pomocí kryptografické hašovací funkce SHA-1, která vytvoří výstupní haš o délce 20 bajtů (160 bitů).

Z tohoto haše je následně vyjmuto prvních 10 bajtů, a těchto 10 bajtů je spojeno s dvěma bajty reprezentující port, který specifikoval uživatel. Výstupem této části je tedy 12 bajtů dlouhá zpráva, která obsahuje číslo portu, který má být otevřen a autentizační kód tohoto portu.

V další části je načten soubor s čítačem. Obsah souboru obsahující čítač je nejprve načten jako celé číslo a následně je převedeno na bajty reprezentující čtyřbajtové celé číslo bez znaménka v kódování big-endian. Tyto čtyři bajty jsou následně doplněny nulami na velikost 16 bajtů. Bajty reprezentující čítač jsou dále zašifrovány pomocí symetrické šifry AES v módu ECB za pomocí šifrovacího klíče načteného z profilu. Také se inkrementuje původní čtyřbajtový, načtený čítač o jedničku a je znovu uložen do souboru. Výstupem této části je tedy zašifrovaný čítač o velikosti 16 bajtů.

Následně je použita operace XOR na 12 bajtů zprávy, která obsahuje číslo portu a jeho autentizační kód, a 16 bajtů zašifrovaného čítače. Vzhledem k rozdílné délce obou částí je použito pouze prvních 12 bajtů ze zašifrovaného čítače, poslední čtyři bajty nejsou použity. Výstupem této části je tedy zašifrovaná zpráva o délce 12 bajtů obsahující číslo portu, který má být na straně serveru otevřen a autentizační kód tohoto portu.

Zašifrovaná zpráva je následně rozdělena na čtyři části:

- První dva bajty (bajty 0 a 1) jsou převedeny na celé číslo bez znaménka v kódování big-endian. Toto číslo bude následně vloženo do IP hlavičky autentizačního paketu na místo identifikace paketu.

- Další čtyři bajty (bajty 2, 3, 4 a 5) jsou převedeny na celé číslo bez znaménka v kódování big-endian. Toto číslo bude vloženo do TCP hlavičky autentizačního paketu na místo sekvenčního čísla.

- Následující čtyři bajty (bajty 6, 7, 8 a 9) jsou převedeny na celé číslo bez znaménka v kódování big-endian. Toto číslo bude vloženo do TCP hlavičky autentizačního paketu na místo potvrzovacího čísla.

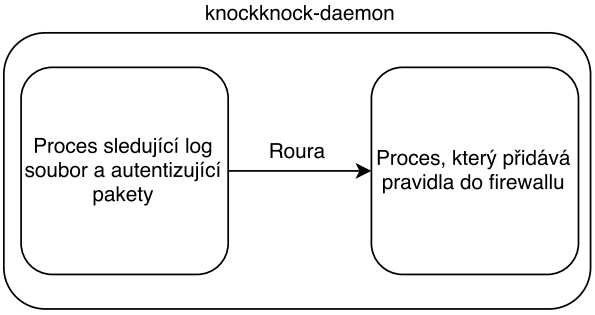
- Poslední dva bajty (bajty 10 a 11) jsou převedeny na celé číslo bez znaménka v kódování big-endian. Toto číslo bude vloženo do TCP hlavičky autentizačního paketu na místo reprezentující délku okna.

Nakonec se pomocí programu hping [30] pošle jeden TCP SYN paket s prázdným obsahem, který obsahuje ve výše zmíněných položkách hlaviček IP a TCP zašifrovanou zprávu. Tento paket je odeslán na cílový port, který je definován v profilu uživatele.

Pro pohodlnější odesílání autentizačního paketu autor poskytuje dodatečný nástroj knockknock-proxy. Tento nástroj slouží jako SOCKS proxy [31], která automaticky odesílá autentizační pakety. Uživatel tedy nakonfiguruje svoji aplikaci (např. webový prohlížeč či emailový klient), aby využívala SOCKS proxy a při každém pokusu o připojení na nakonfigurovaný server tato proxy automaticky zašle autentizační paket, který upraví konfiguraci firewallu na straně serveru.

Ačkoliv autor uvádí, že sama aplikace musí podporovat možnost SOCKS proxy, bylo experimentálně ověřeno, že lze využít dodatečného nástroje, který přesměruje veškerý provoz téměř libovolné aplikace přes SOCKS proxy. Mezi takové nástroje patří např. proxychains-ng [32] či redsocks [33].

1.4.2.2 Princip fungování serverové části



Obrázek 1.3: Diagram serverové části nástroje knockknock.

Serverová část nejprve načte všechny profily z adresáře /etc/knockknock.d/profiles/ a také načte svoji vlastní konfiguraci ze souboru /etc/knockknock.d/config.

Jak lze vidět na diagramu serverové části 1.3, nástroj se pomocí funkce fork() rozdělí na dva procesy, jeden proces, který přidává pravidla do firewallu a druhý proces, který čte logovací soubor, hledá nově přijaté pakety a autentizuje je. Jako nástroj pro meziprocesovou komunikaci autor zvolil rouru.

Jelikož program manipuluje s konfigurací firewallu, musí běžet pod administrátorským oprávněním. Autor však kvůli zvýšené bezpečnosti, sníží oprávnění procesu, který čte logovací soubor na nutné minimum. Administrátorská práva jsou tak ponechána pouze procesu, který provádí velmi malou část kódu.

Jelikož autor nechtěl využívat knihovnu pcap a chtěl, aby byl nástroj co nejjednodušší, použil pro získávání informací o paketech standardní logovací soubor programu iptables.

Jak lze vidět z instalační příručky [29], autor nastaví pomocí následujících příkazů logování programu iptables.

```
sudo iptables -N REJECTLOG

sudo iptables -A REJECTLOG -j LOG --log-level debug \
--log-tcp-sequence --log-tcp-options --log-ip-options \
-m limit --limit 3/s --limit-burst 8 --log-prefix "REJECT "

sudo iptables -A REJECTLOG -p tcp -j REJECT \
--reject-with tcp-reset

sudo iptables -A REJECTLOG -j REJECT
```

Server pak jednoduše čte nové řádky na konci soubor /var/log/kern.log. Díky zvýšené úrovni logování a několika dodatečným přepínačům lze z jednoho záznamu v logovacím souboru získat všechny relevantní informace potřebné k autentizaci paketu.

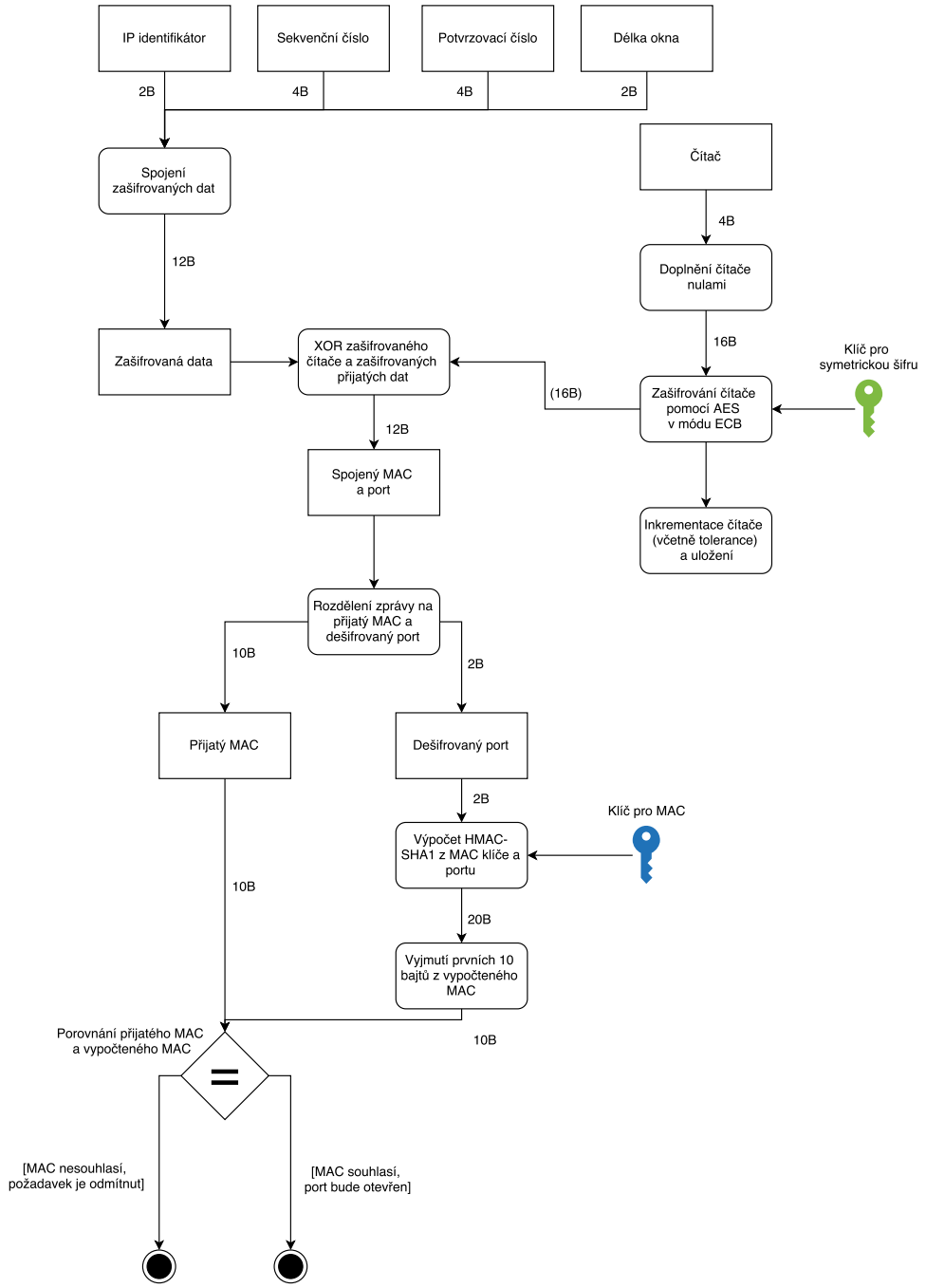
Jak lze vidět na obr. 1.4 nejprve musí nástroj ze záznamu v logovacím souboru získat relevantní data, tedy čtyři položky: IP identifikátor, sekvenční číslo, potvrzovací číslo a délku okna. Následně jsou tato čísla převedena na bajty podle následujícího předpisu:

- IP identifikátor je převeden na dva bajty reprezentující celé číslo bez znaménka v kódování big endian.
- Sekvenční číslo je převedeno na čtyři bajty reprezentující celé číslo bez znaménka v kódování big endian.
- Potvrzovací číslo je převedeno na čtyři bajty reprezentující celé číslo bez znaménka v kódování big endian.
- Délka okna je převedena na dva bajty reprezentující celé číslo bez znaménka v kódování big endian.

Tyto bajty jsou pak spojeny ve výše zmíněném pořadí, výstupem je tedy 12 bajtů dlouhá zašifrovaná zpráva.

Dle cílového portu autentizační paketu je nalezen odpovídající profil. Dále je z tohoto profilu načten čítač a dva klíče. Stejně jako v případě klientské části, je načten čítač, převeden na bajty reprezentující čtyřbajtové číslo bez znaménka a je doplněn nulami na velikost 16 bajtů.

Bajty reprezentující čítač jsou pak zašifrovány pomocí symetrické šifry AES v módu ECB. Jelikož se jedná o symetrickou šifru je použit stejný šifrovací klíč pro šifrování i dešifrování. Výstupem je tedy zašifrovaný čítač o velikosti 16 bajtů.



Obrázek 1.4: Diagram dešifrování dat autentizačního paketu nástroje knockknock.

Následně je použita operace XOR na 12 bajtů dlouhou zašifrovanou zprávu a 16 bajtů dlouhý zašifrovaný čítač. Stejně jako v případě šifrovaní je použito pouze prvních 12 bajtů ze zašifrovaného čítače. Výstupem je dešifrovaná zpráva o délce 12 bajtů.

Dešifrovaná zpráva o velikosti 12 bajtů je rozdělena na dvě části. První dva bajty obsahují port, který má být otevřen a dalších 10 bajtů obsahuje autentizační kód. Následně je tedy vypočten autentizační kód z přijatého portu, je z něj vyjmuto prvních 10 bajtů a tento autentizační kód je porovnám s přijatým autentizačním kódem.

Pokud se oba dva autentizační kódy zprávy shodují, je paket úspěšně autentizován a je poslána zpráva skrz rouru druhému procesu, aby bylo přidáno pravidlo do firewallu, které otevře daný port pro zdrojovou IP adresu tohoto autentizačního paketu. Pokud se autentizační kódy zprávy neshodují je požadavek zamítnut.

Pokud byl požadavek úspěšně autentizován je přidáno příslušné pravidlo do firewallu, zároveň je vytvořeno nové vlákno, které po uplynutí časového okna toto pravidlo z firewallu odstraní. Délka časového okna se specifikuje v konfiguračním souboru serverové části.

1.4.2.3 Analýza bezpečnosti

Samotné zašifrování obsahu zprávy zajistí důvěrnost zprávy, ale nikoliv její integritu a autenticitu. K tomu je nutné doplnit zašifrovanou zprávu o autentizační kód zprávy (MAC). Autor dle [29] uvádí, že nástroj používá šifrovací paradigma autentizuj-pak-zašifruj (MAC-then-encrypt).

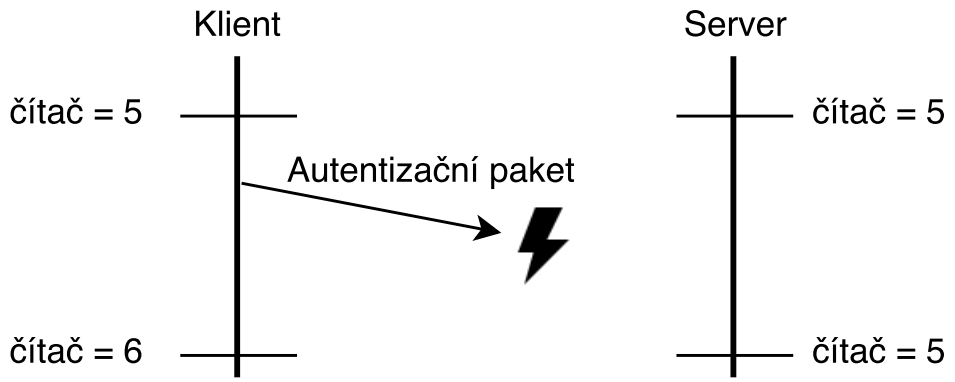
Ačkoliv autor uvádí, že jeho implementace má vlastnost IND-CCA, dle článku [34] však paradigma autentizuj-pak-zašifruj nesplňuje vlastnost IND-CCA. Další paradigmata a splnění vlastnosti IND-CCA bude diskutováno v kapitole návrh 2.1.2.

Jako šifrovací algoritmus autor zvolil algoritmus AES [35] s délkou klíče 128 bitů a velikostí bloku také 128 bitů. Jako implementaci autentizačního kódu zprávy autor zvolil algoritmus HMAC [36] ve spojení s kryptografickou hašovací funkcí SHA-1. Oba tyto algoritmy jsou dostačující dle [37].

Ačkoliv k zašifrování čítače se používá základní operační mód blokové šifry ECB, použití zašifrovaného čítače a následné operaci XOR s otevřeným textem se říká čítačový (CTR) mód, čítačový mód mění blokovou šifru na synchronní proudovou šifru. Tento mód je rovněž dostačující dle [37].

Jelikož se používá symetrická šifra a tedy se používá stejný (nebo snadno odvoditelný) klíč k šifrování a dešifrování, je nutné aby byly oba klíče přepraveny na druhý počítač zabezpečeným kanálem, např. pomocí programu scp nebo pomocí šifrované emailové komunikace.

Jak lze vidět na obr. 1.5 může se stát, že se autentizační paket ztratí, pak dojde k desynchronizaci čítače na straně klienta a serveru. Jelikož klient nikdy nedostává zprávu od serveru o výsledku autentizace (což je úmyslná



Obrázek 1.5: Diagram desynchronizace klienta a serveru.

vlastnost tohoto autentizačního protokolu), vždy předpokládá úspěšnou autentizaci a vždy inkrementuje svůj čítač po odeslání autentizačního paketu. V tuto chvíli se však při opětovném odesláním autentizačního paketu nemůže klient autentizovat z důvodu různé hodnoty čítače.

Pro opravení této desynchronizační chyby server při neúspěšné autentizaci paketu zkusí čítač zašifrovat znovu, tentokrát však inkrementovaný o jedničku. Pokud byla autentizace opět neúspěšná, server opět inkrementuje čítač a zkusí ho zašifrovat znovu. Počet těchto pokusů se odvíjí od direktivy window, která je specifikovaná v konfiguračním souboru. Jako výchozí hodnotu zvolil autor hodnotu 20.

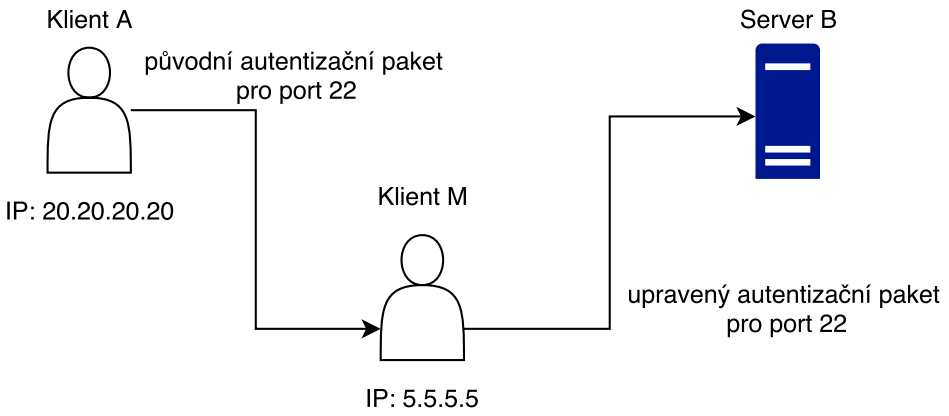
Pokud server úspěšně autentizoval paket, uloží svůj aktuální inkrementovaný čítač, tímto dojde k opětovné synchronizaci čítačů na obou stranách.

Větší hodnota direktivy window zaručí větší odolnost proti ztraceným paketům, avšak poskytne případnému útočníkovi větší prostor k útoku.

Jelikož se čítač při každém novém pokusu o autentizaci inkrementuje, nikdy se nestane, že by se šifroval dvakrát stejný čítač, tento předpoklad je nezbytný pro správné fungování čítačového (CTR) módu. Navíc díky neustále se měnícímu čítači se mění i zašifrovaný text, který se vloží do příslušných hlaviček TCP a IP. Z tohoto důvodu dva autentizační pakety s požadavkem na otevření stejného portu budou mít různou hodnotu hlaviček TCP a IP a proto bude znemožněn útok přehráním.

Jako bezpečnostní slabina však může být vnímán fakt, že server po úspěšné autentizaci klienta, upraví konfiguraci firewallu tak, že povolí přístup na port, který byl zašifrován v autentizačním paketu, pro zdrojovou IP adresu autentizačního paketu.

Jak lze vidět na obr. 1.6, pokud by případný útočník (klient M) zachytil autentizační paket od právoplatného odesilatele (klient A), mohl by změnit zdrojovou IP adresu za svoji IP adresu. Server B by tedy úspěšně autentizo-



Obrázek 1.6: Ukázka útoku na autentizační paket.

val tento autentizační paket, ale změnil by konfiguraci firewallu ve prospěch útočníka (klient M).

1.4.3 fwknopd

Nástroj fwknopd [38] (což je zkratka pro Firewall Knock Operator Daemon) je jeden z nejrozšířenějších nástrojů implementující koncept single packet authorization, stejně jako dříve zmíněný nástroj knockd (viz 1.4.1) je i tento nástroj dostupný jako instalační balíček pro linuxové distribuce CentOS, Fedora, Debian a Ubuntu.

První verze nástroje byla napsána ve skriptovacím jazyce Perl, ale následně byl tento nástroj přepsán do jazyka C. Tento nástroj obsahuje klientskou i serverovou část.

Díky velkému počtu režimů, vlastností a díky pečlivě zvoleným kryptografickým algoritmům je tento nástroj pravděpodobně nejbezpečnější a nejkompletnější implementace konceptu single packet authorization.

1.4.3.1 Princip fungování klientské části

Klientská část je k dispozici nejen pro unixové operační systémy, ale také pro operační systémy Android, iOS a MS Windows. Existují verze jak s grafickým rozhraním, tak s řádkovým rozhraním.

Stejně jako při použití nástroje knockknock (viz 1.4.2) je nejprve nutné pro každého klienta vygenerovat profil, který obsahuje šifrovací klíče. Při vytváření profilu s výchozími parametry jsou vytvořeny dva klíče: jeden pro symetrickou šifru AES, druhý pro autentizační kód zprávy (který je tvořen pomocí HMAC-SHA-256). Klíče jsou generovány pomocí pseudonáhodného generátoru /dev/urandom, délka vygenerovaných klíčů je v případě šifrovacího klíče

16 bajtů a v případě klíče pro autentizační kód zprávy 32 bajtů. Profil je uložen do souboru $HOME/.fwknoprc.

Klíč pro autentizační kód zprávy nemusí být generován, jelikož použití autentizačního kódu zprávy není nutné, pokud si to uživatel nepřeje, autor však ve svém průvodci [39] výrazně doporučuje používat autentizační kód.

Klient není omezen jen na použití symetrické šifry AES, která je použita v módu CBC, ale k dispozici je i možnost použití asymetrické kryptografie využívající GnuPG.

Na rozdíl od nástroje knockknock (viz 1.4.2), který zašifrovaná data vložil do hlaviček protokolů TCP a IP autentizačního paketu, tento nástroj posílá zašifrovaná data v těle paketu. Ve výchozím režimu tento nástroj využívá pro přenos protokol UDP, avšak poskytuje možnost využití protokolů TCP, ICMP, či HTTP.

Z důvodu možného podvrhnutí zdrojové IP adresy (tento typ útoku je diskutován v sekci 1.4.2.3), tento nástroj vkládá veřejnou zdrojovou IP adresu klienta do těla paketu. Klient má možnost buď specifikovat konkrétní zdrojovou IP adresu, která bude obsahem paketu, či zapnout automatickou detekci veřejné IP adresy.

Automatická detekce se provádí tak, že klient provede HTTPS požadavek na URL https://www.cipherdyne.org/cgi-bin/myip, která jako odpověď vrátí veřejnou IP adresu klienta. HTTPS požadavek se provádí pomocí nástroje wget [40]. Autor však ve svém průvodci [39] doporučuje používat pevně specifikované IP adresy a nepoužívat automatickou detekci, jelikož tato akce provádí DNS i HTTPS požadavky, které jsou snadno detekovatelné pro případného útočníka. Navíc se tento HTTPS požadavek může stát obětí Man-in-the-middle útoku.

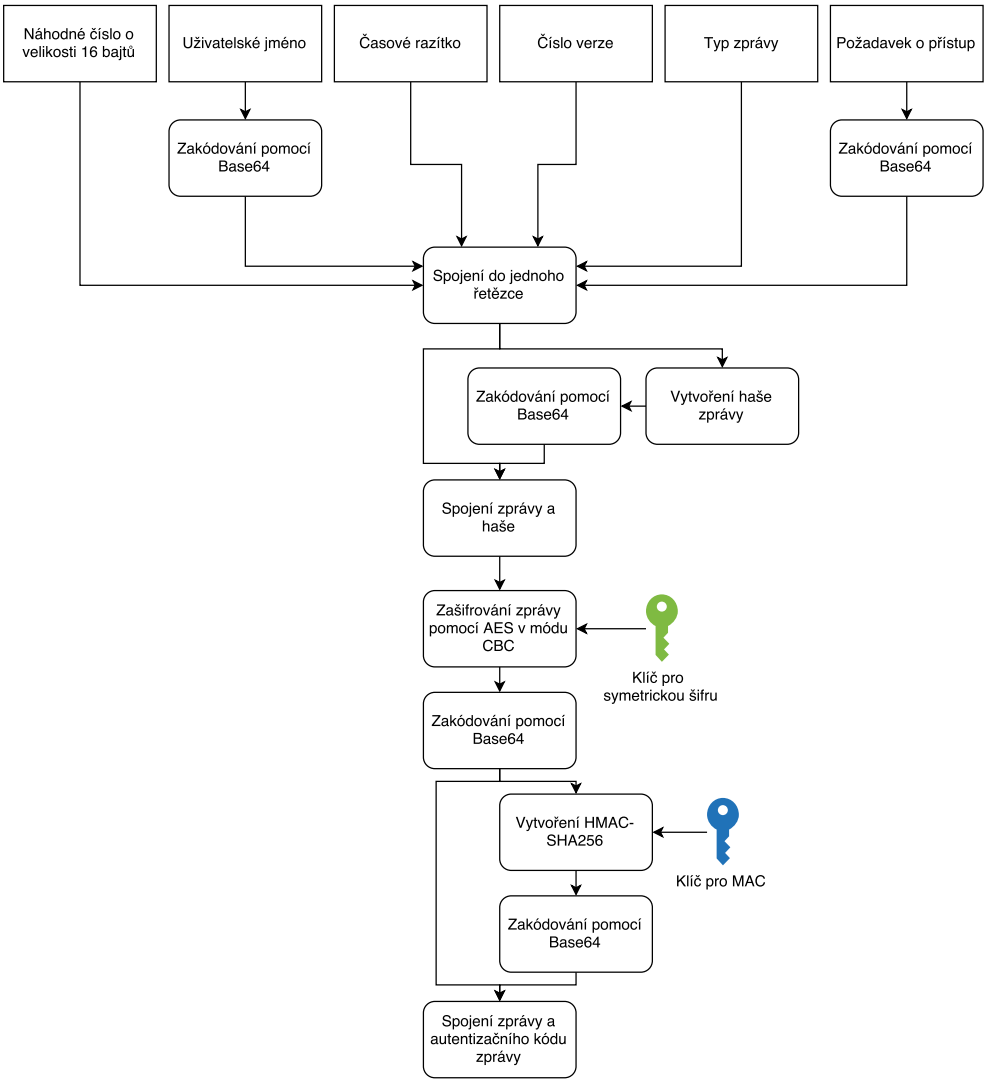
Díky této vlastnosti je klientská část kompatibilní se síťovým překladem adres (NAT) na straně klienta.

Dle [39] vypadá obsah autentizačního paketu následovně. Mezi povinné položky patří:

1. 16 bajtů náhodných dat (celé číslo bez znaménka),
2. lokální uživatelské jméno,
3. lokální časové razítko,
4. verze fwknop,
5. typ zprávy SPA,
6. požadavek o přístup nebo příkaz k provedení (např. řetězec „1.1.1.1,tcp/22“, který žádá o upravení konfigurace firewallu a povolení přístupu pro IP adresu 1.1.1.1 na TCP port 22),
7. haš zprávy (ve výchozím nastavení SHA-256).

Mezi nepovinné položky patří:

1. požadavek o přesměrování portu (DNAT),
2. autentizační informace pro software třetí strany,
3. časové okno pro platnost pravidla upravující firewall.



Obrázek 1.7: Diagram vytváření autentizačního paketu programu fwknop.

Tyto položky jsou od sebe odděleny znakem „:“, části, které by mohly obsahovat znak „:“ jsou zakódovány pomocí Base64 [41]. Na obr. 1.7 lze vidět,

jak je tento autentizační paket konstruován za použití výchozích nastavení. Z obr. 1.7 je také vidět, že nástroj používá paradigma zašifruj-pak-autentizuj.

Po spojení všech částí zprávy do jednoho řetězce, je následně tento řetězec zašifrován, zakódován pomocí Base64 a opatřen autentizačním kódem zprávy HMAC (ten je také následně zakódován pomocí Base64). Takto vytvořený paket je následně odeslán po síti. Autor ve svém průvodci [39] uvádí, že velikost obsahu takové paketu je přibližně 200 bajtů (při použití výchozího nastavení).

Ve výchozím nastavení se tento paket odešle serveru pomocí UDP na cílový port 62 201. Dalšími možnostmi je využití protokolu TCP či HTTP.

Zajímavou možností je využít protokolu HTTP pro přenos autentizačního paketu. Jelikož jsou po zašifrování data zakódována pomocí Base64, autor využil tohoto zašifrovaného a zakódovaného řetězce jako identifikátor zdroje. Nejprve však převede znaky „+“ na „−“ a „/“ na „_“, takto vytvořený řetězec je pak použit v HTTP protokolu následovně:

```
GET /<zakódovaný řetězec> HTTP/1.0
další položky HTTP hlavičky
```

Na straně serveru je požadavek na tento zdroj detekován a změněné znaky jsou nastaveny na původní hodnotu. Server může nyní paket autentizovat.

1.4.3.2 Princip fungování serverové části

Autor uvádí, že serverová část nástroje je kompatibilní s operačními systémy Linux, FreeBSD, OpenBSD, MacOS X a podporuje různé implementace firewallu jako iptables, ipfw a pf.

Serverová část obsahuje dva konfigurační soubory: obecný konfigurační soubor, který je uložen v /etc/fwknop/fwknopd.conf a konfigurační soubor, který upravuje přístupová oprávnění pro jednotlivé uživatele, ten je uložen v souboru /etc/fwknop/access.conf.

Na rozdíl od nástroje knockknock (viz 1.4.2), který prováděl pouze autentizaci, tento nástroj provádí také autorizaci. Správně autentizovaný paket tedy musí být ještě úspěšně autorizován, zdali má uživatel práva k provedení akce. Profily uživatelů, které jsou následně zkopírovány na server, mohou mít tedy různé úrovně oprávnění.

Pro jednotlivé uživatele tak lze specifikovat jaké mohou otevírat porty (např. všechny či pouze několik specifikovaných), z jakých IP adres se mohou připojovat (lze nastavit všechny nebo konkrétní IP adresy či sítě pomocí CIDR notace), atd.

Server může fungovat v několika režimech:

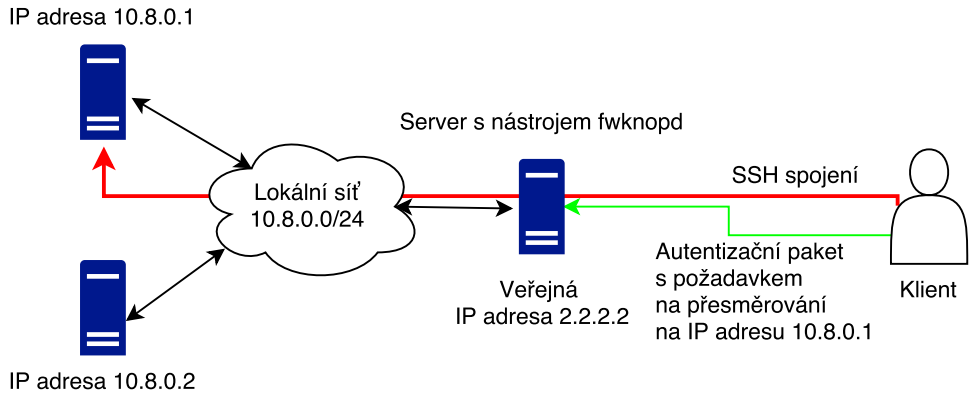
- Přímo číst příchozí pakety z lokálního UDP portu 62 201 (nebo jiného specifikovaného portu). Pak není třeba žádná další knihovna. Tento port však musí být otevřený, jelikož však server nevrací žádnou odpověď, jeví se port jako filtrovaný pomocí firewallu.

- Využívat knihovnu pcap a odposlouchávat tak pakety, které odpovídají filtru. Tento režim se dá využít při použití všech možných protokolů UDP, TCP i HTTP.
- Použít knihovnu Netfilter Queue. Tato knihovna se využívá jako alternativa ke knihovně pcap.

Při využití TCP, na specifikovaném portu pak poslouchá falešný TCP server, který neprovádí trojcestné potřesení rukou (three-way handshake), ani nijak nepotvrzuje přijaté pakety. Při tomto režimu se využívá knihovna pcap ke čtení samotného obsahu paketu.

Serverová část poskytuje mnoho možností konfigurace, mezi některé možnosti konfigurace patří:

- Server může kontrolovat časové razítko z přijatého autentizačního paketu a autentizovat pouze pakety, které nejsou starší než definovaná hranice.
- Specifikovat vlastní příkazy, které se provedou po provedení úspěšné autentizaci paketu.
- Možnost zapnout síťový překlad adres (NAT) na straně serveru.



Obrázek 1.8: Diagram překladu síťových adres na straně serveru s nástrojem fwknopd.

Jak již bylo zmíněno v části 1.4.3.1 klientská část podporuje překlad síťových adres. Na straně serveru se dá také využít překladu síťových adres. Po zapnutí této vlastnosti se při odesílání autentizačního paketu na straně klienta specifikuje, na jakou lokální IP adresu a port mají být jeho požadavky přesměrovány. Pokud tedy uživatel vloží do autentizačního paketu žádost o přesměrování na lokální IP adresu 10.8.0.1 a port 22, po úspěšné autentizaci na serveru budou jeho následující SSH spojení na server přesměrovávána na lokální IP adresu 10.8.0.1. Diagram této konfigurace si lze prohlédnout na obr. 1.8.

1.4.3.3 Analýza bezpečnosti

Nástroj fwknopd využívá moderních kryptografických algoritmů. Pro hašování využívá SHA-256 (ve výchozím nastavení), ale lze použít i hašovací funkce SHA-384 či SHA-512, tyto hašovací funkce lze také využít pro autentizační kód zprávy HMAC. Pro šifrování se používá v případě symetrické kryptografie šifra AES a v případě využití asymetrické kryptografie se využívá GnuPG.

Proti útoku přehráním (replay attack) se tento nástroj brání pomocí náhodné hodnoty v těle paketu (tato hodnota je získána z pseudonáhodného generátoru /dev/urandom) a také pomocí časového razítka. Z tohoto důvodu i dva stejné autentizační pakety (co se týče požadavku na změnu konfigurace firewallu) budou mít rozdílný obsah těla paketu.

Proti útoku podvrhnutí zdrojové IP adresy se tento nástroj brání vložením IP adresy do těla paketu a jeho následným zašifrováním a opatřením autentizačním kódem.

Nástroj používá paradigma zašifruj-pak-autentizuj (encrypt-then-MAC) a tedy dle [34] splňuje vlastnost IND-CCA.

1.4.4 The Doorman

Nástroj The Doorman [42] obsahuje klientskou část jménem knock a serverovou jménem doormand. Obě dvě části jsou napsány v jazyce C a mají pouze řádkové rozhraní.

Klientská část je kompatibilní s unixovými operačními systémy, ale také s operačním systémem MS Windows. Klientská část obsahuje konfigurační soubor s následujícími čtyřmi direktivami:

Skupina Skupina, které slouží k identifikaci uživatelů na serveru.

Heslo Heslo, které slouží jako klíč pro HMAC-MD5 autentizační kód zprávy.

Port Cílový port, na který se posílá autentizační paket.

Příkaz Příkaz, který se provede neprodleně po odeslání autentizačního paketu, např. spuštění SSH klienta. V tomto příkazu je možnost použít zástupného makra „%H%“, na jehož místo se doplní IP adresa či doménového jméno serveru.

Při spuštění klientské části uživatel specifikuje jako argumenty programu IP adresu či doménové jméno serveru a port, který chce otevřít.

Tento nástroj implementuje koncept single packet authorization. Odesílá jeden UDP paket na předem konfigurovaný port. Obsahem tohoto UDP paketu jsou následující položky, které jsou oddělené znakem mezera „ “ (0x20):

- Číslo portu, který má být zpřístupněn po úspěšné autentizaci a autorizaci.

- Jméno skupiny.
- Náhodné číslo, které je vytvořeno pomocí funkce rand(). Jako semínko tohoto generátoru je zvolen aktuální čas.
- HMAC-MD5 autentizační kód zprávy vytvořený z předcházejících položek. Jako klíč je použito heslo z konfiguračního souboru.

Serverová část je kompatibilní s unixovými operačními systémy (podporuje implementace firewall jako jsou iptables, ipchain, ipfw, pf a ipf). Serverová část je konfigurována pomocí dvou konfiguračních souborů: hlavní konfigurační soubor, který je uložen v /usr/local/etc/doormand/doormand.cf a konfigurační soubor upravující uživatele v /usr/local/etc/doormand/guestlist.

Hlavní konfigurační soubor nastavuje např. port a síťové rozhraní, ze kterého jsou čteny příchozí UDP autentizační pakety, cestu k logovacímu souboru, úroveň podrobností v logovacím souboru nebo cestu k archivnímu souboru.

Archivní soubor je soubor, který obsahuje HMAC-MD5 autentizační kódy úspěšně autentizovaných a autorizovaných paketů. Při přijetí každého autentizačního paketu zkontrolováno, zdali archivní soubor již neobsahuje tento autentizační kód, pokud ano je autentizační paket zneplatněn a je vypsáno varování o probíhajícím útoku přehráním.

Konfigurační soubor upravující oprávnění uživatelů obsahuje na každém řádku čtyři následující položky:

Skupina Jméno skupiny slouží jako identifikátor. Při přijetí autentizačního paketu je dle tohoto identifikátoru načten odpovídající řádek s konfigurací.

Heslo Pomocí hesla je vytvořen z příchozího autentizačního paketu autentizační kód HMAC-MD5 a je porovnám s přijatým autentizačním kódem.

Porty Každá skupina má určeno, které porty mohou být otevřeny pro danou skupinu.

IP adresa Pro každou skupinu lze specifikovat z jakých zdrojových IP adres mohou být odeslány autentizační pakety. Do konfiguračního souboru lze zadat konkrétní IP adresu nebo síť v CIDR notaci, lze také specifikovat doménové jméno.

Server odposlouchává pakety pomocí knihovny pcap, odposlouchávány jsou všechny UDP pakety s cílovým portem, který je specifikovaný v hlavním konfiguračním souboru. Samotný obsah těla paketu je také čten pomocí knihovny pcap, proto může tento sledovaný port filtrován firewallem.

Při přijetí je paket autentizován a autorizován následujícím způsobem:

- V paketu musí být obsaženo jméno skupiny, které je přítomno také v konfiguračním profilu upravující uživatele. Tímto se načte odpovídající řádek konfigurace.

- Pomocí přidruženého hesla je znovu vypočten autentizační kód a je porovnán s přijatým autentizačním kódem. Pokud oba kódy souhlasí, je paket úspěšně autentizován.

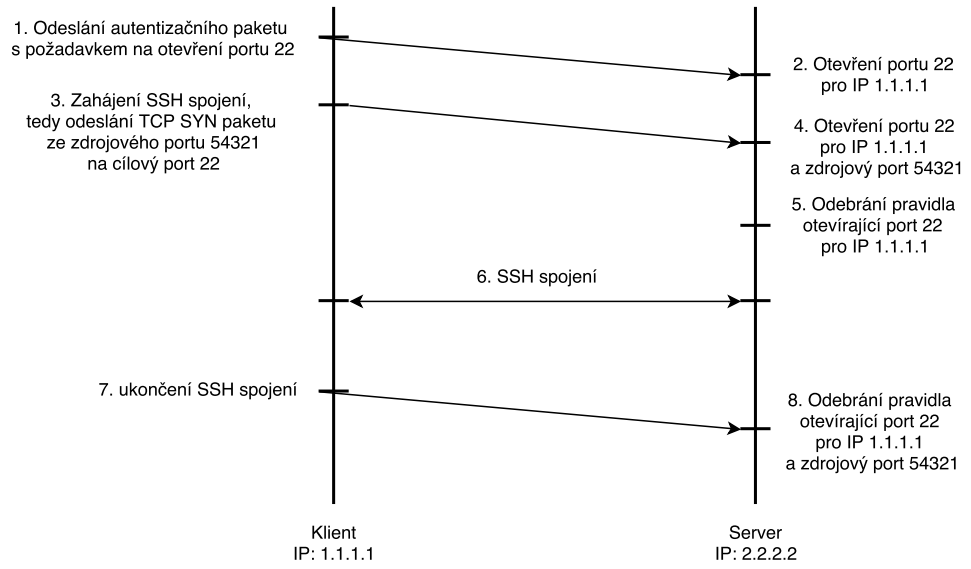
- Zdrojová IP adresa paketu musí odpovídat IP adrese či síti specifikované v konfiguračním souboru.

- Požadavek na otevření portu musí odpovídat portům, které jsou specifikovány v konfiguračním souboru.

- V archivním souboru je vyhledán autentizační kód přijaté zprávy, pokud nebyl v archivním souboru nalezen je paket úspěšně autorizován.

Po úspěšné autentizaci a autorizaci je do firewallu přidáno obecné pravidlo, které povoluje příchozí spojení z dané IP adresy na port, který byl obsažen v těle autentizačního paketu. Následně začne nástroj čekat na příchozí TCP SYN paket od klienta. Po úspěšném přijetí TCP SYN paketu je přidáno více konkrétní pravidlo do firewallu, které povolí spojení z dané IP adresy na port, který byl v těle autentizačního paketu a navíc pouze ze zdrojového portu, ze kterého pocházel TCP SYN paket. Poté je odstraněno první, obecnější pravidlo a je ponecháno pouze druhé, konkrétní pravidlo.

Následně server sleduje v časových intervalech výstup programu lsof, který vypisuje aktuálně navázaná spojení. Pokud je spojení ukončeno, je odebráno i konkrétní pravidlo z firewallu a port je tedy znovu filtrován.



Obrázek 1.9: Ukázka funkcionality programu doormand.

Na obr. 1.9 lze vidět konkrétní ukázku funkcionality nástroje doormand a knock a způsob vytváření konfiguračních pravidel pro firewall.

Ačkoliv nástroj používá autentizační kód zprávy (vytvořený pomocí HMAC-MD5), zprávu nijak nešifruje a tedy pro případného útočníka je velmi snadné zjistit, který port bude otevřen. Navíc se pro tvorbu autentizačního kódu HMAC používá zastaralá hašovací funkce MD5, která je dle [37] nevyhovující. Nástroj je dále náchylný vůči útoku podvržení zdrojové IP adresy.

1.4.5 Py-Port Knocking project

Nástroj Py-Port Knocking project [43] je napsán v jazyce Python a autor uvádí kompatibilitu s operačními systémy Linux, MS Windows a zmiňuje možnou kompatibilitu i s unixovými operačními systémy MacOS, *BSD a Solaris.

Klientská část je funkčností prakticky totožná s klientskou částí nástroje knockd (viz 1.4.1), podporuje však pouze TCP.

Serverová část je také velmi podobná serverové části nástroje knockd (viz 1.4.1), avšak možnosti konfigurace jsou mnohem chudší. Nástroj využívá knihovnu pcap (popř. WinPcap v operačním systému MS Windows) k odposlouchávání příchozích paketů. Autor využil Python modulu pcapy, který poskytuje propojení knihovny pcap (WinPcap) s jazykem Python. Filtr pro knihovnu pcap je nastaven velmi obecně, nástroj odposlouchává všechny TCP pakety.

V konfiguračním souboru se dají specifikovat libovolné příkazy, které se provedou po úspěšné autentizaci. Mezi další možnosti konfigurace patří časové okno pro dokončení autentizační sekvence a možnost specifikace na kterém síťovém rozhraní bude tento nástroj odposlouchávat pakety.

Díky možnosti specifikovat libovolné příkazy v konfiguračním souboru a díky použití multiplatformního jazyka Python, může serverová část tohoto nástroje operovat i pod operačním systémem MS Windows.

Co se týče bezpečnosti, díky použití statických sekvencí je nástroj náchylný vůči útoku přehráním a také je náchylný k podvrhnutí zdrojové IP adresy.

1.4.6 winKnocks

Nástroj winKnocks [44] je napsán v jazyce Java, obě části (klientská i serverová) obsahují grafické rozhraní vytvořené pomocí knihovny Swing. Tento nástroj je kompatibilní s operačním systémem MS Windows.

V klientské části se nejprve musí vytvořit konfigurační profil ve formátu XML. Tento profil obsahuje akce, které se provedou po úspěšné autentizaci sekvence a také sekvence paketů, které budou identifikovat tento profil.

Tyto sekvence mohou být tvořené pomocí paketů TCP, UDP a ICMP. V případě paketu TCP lze navíc specifikovat sekvenční číslo, potvrzovací číslo, velikost okna a příznaky. U všech paketů může uživatel specifikovat obsah jejich těla.

Jako akci lze specifikovat libovolnou kombinaci z následujících možností:

- Požadavek na otevření portu, lze konfigurovat, zdali bude port otevřen pouze pro zdrojovou IP adresu či pro všechny IP adresy.
- Požadavek na uzavření portu, touto akcí lze přidat konkrétní pravidlo, které uzavře specifikovaný port.
- Spustit libovolný skript nebo program.

Při odesílání paketu je do těla paketu navíc vloženo náhodně vygenerované číslo a časové razítko. Navíc však uživatel může vložit do těla paketu tzv. urgentní skript, což je příkaz nebo program, který se spustí po úspěšné rozpoznání sekvence, tento skript nemusí být předem definovaný v konfiguračním profilu (tato možnost však musí být explicitně povolena v na straně serveru). Tělo paketu je zašifrováno symetrickou šifrou DES v módu ECB. Šifrovací klíč je odvozen z hesla, které specifikuje uživatel. Odvození klíče probíhá tak, že se uživatelovo heslo zašifruje symetrickou šifrou DES v módu ECB pomocí klíče, který je napevno specifikován ve zdrojovém kódu.

Serverová část používá pro svojí konfiguraci vygenerované profily z klientské části ve formátu XML. Pro odposlouchávání paketů používá knihovnu WinPcap, což je verze knihovny pcap pro operační systém MS Windows. Pro usnadnění práce s knihovnou WinPcap používá tento nástroj knihovnu jpcap, která poskytuje propojení jazyka Java a knihovny WinPcap. Filtr pro knihovnu jpcap je buď vygenerován automaticky na základě profilů či může být nastaven ručně.

Proti útoku přehrání se tento nástroj brání v podobě náhodného čísla a časového razítka v těle paketu. Bohužel je tělo paketu šifrováno zastaralou šifrou DES (v [37] není DES uveden jako šifra splňující minimální požadavky) bez autentizačního kódu zprávy, navíc s nevyhovujícím způsobem odvození klíče z hesla. Nástroj není chráněn vůči podvržení zdrojové IP adresy.

1.4.7 KnockKnock – Port Knocking for Windows

Nástroj „KnockKnock – Port Knocking for Windows“ obsahuje klientskou část napsanou v jazyce C++ a serverovou část napsanou v jazyce Java. Obě části obsahují grafické rozhraní.

Klient obsahuje naprosto minimální funkcionalitu, uživatel má možnost pouze zadat sekvenci cílových TCP portů a specifikovat cílovou IP adresu.

Serverová část využívá programu CHX-I Packet Filter k odposlouchávání paketů a který zároveň slouží jako firewall a nahrazuje tak výchozí Windows Firewall v operačním systému MS Windows. Odposlouchávání paketů docílí tento nástroj tak, že čte soubor s událostmi o příchozích paketech.

Uživatel v serverové části nástroje specifikuje sekvenci TCP portů a port, který tato sekvence otevře. Při úspěšné autentizaci sekvence tento nástroj

upraví konfiguraci programu CHX-I Packet Filter, tak aby povolila přístup na daný port pro danou zdrojovou IP adresu.

Bohužel firma vyvíjející program CHX-I Packet Filter již zanikla a proto je tento nástroj nyní téměř nepoužitelný. Díky využití statických, předem definovaných sekvencí, je tento nástroj náchylný vůči útoku přehráním. Navíc je zranitelný vůči podvržení zdrojové IP adresy.

1.4.8 It's me (IM)

Nástroj It's me (IM) [45] obsahuje pouze klientskou část, je kompatibilní se systémem MS Windows a poskytuje pouze řádkové rozhraní.

Nástroj se konfiguruje pomocí konfiguračního souboru, který se jmenuje stejně jako binární soubor klienta. Konfigurační soubor obsahuje hlavní sekci, která obsahuje nastavení pro všechny servery (tato nastavení však mohou být přepsána v nastavení konkrétních serverů). Konfigurační soubor také obsahuje sekci pro každý server, kde se chce klient autentizovat.

Při použití programu se jako argument příkazové řádky předá název sekce, která identifikuje požadovaný server. Pokud uživatel nezadá jako argument žádný název sekce, je předpokládána sekce „Default". Sekce konfigurující server může obsahovat následující konfigurační direktivy:

HOST Tato direktiva specifikuje IP adresu nebo doménové jméno cílového serveru.

RANGE Tato direktiva specifikuje rozsah portů, které budou požity. Všechny následující specifikovaná čísla portů budou namapována do tohoto rozsahu.

CRYPT Tato direktiva specifikuje, zdali bude použito šifrování Blowfish.

PASSWORD Specifikuje heslo pro použité šifrování.

DEBUG Tato direktiva řídí úroveň podrobnosti informačních hlášek.

KNOCK-n Specifikuje port na n-té pořadí v sekvenci. Hodnotou může být konkrétní číslo nebo některé ze zástupných maker.

Jako zástupné makro lze specifikovat jednu z následujících možností:

IP-n-q Specifikuje q-tou část n-té lokální IP adresy. Tedy například direktiva IP-1-2 počítače s IP adresou 192.168.56.1 nabude hodnotu 56.

DAY Nastaví jako číslo portu aktuální den.

MON Nastaví jako číslo portu aktuální měsíc.

YEAR Nastaví jako číslo portu aktuální rok.

HR Nastaví jako číslo portu aktuální hodinu.

MIN Nastaví jako číslo portu aktuální minutu.

SEC Nastaví jako číslo portu aktuální sekundu.

P-n Nastaví jako číslo portu n-tý argument z příkazové řádky.

K-n Nastaví jako číslo portu specifikované n-té číslo portu. Lze takto referencovat existující číslo portu. Např. direktiva K-3 nastaví jako číslo portu stejné číslo jako má třetí položka v sekvenci.

RND Nastaví jako číslo portu náhodné číslo z rozsahu definovaného pomocí direktivy RANGE.

CRC Nastaví jako číslo portu součet přecházejících čísel portů modulo 255.

Po doplnění všech čísel portů do odpovídajících direktiv přichází na řadu šifrování a mapování do rozsahu. To se provádí následujícím způsobem:

1. Nejprve jsou čísla portů převedeny do binární reprezentace.

2. Následně je tato sekvence čísel rozdělena po jednotlivých bajtech.

3. Toto pole bajtů musí být nyní zarovnáno na blok o délce násobku osmi bajtů, který vyžaduje symetrická šifra Blowfish. To se provádí pomocí zarovnání PKCS#7. Toto zarovnání doplní potřebný počet bajtů do bloku pomocí bajtů o hodnotě chybějícího počtu bajtů. Tedy pokud chybí do bloku osmi bajtů tři bajty, jsou doplněny tři bajty o hodnotě tři. Pokud pole bajtů je přesně násobek délky bloku, je doplněn celý blok (osm bajtů) o hodnotě osm.

4. Toto zarovnané pole bajtů je nyní zašifrované symetrickou šifrou Blowfish v módu ECB.

5. Ze zadaného rozsahu (direktiva RANGE) je vypočtena délka rozsahu. Z délky rozsahu je vypočteno počet bitů potřebných k reprezentování tohoto rozsahu. Tedy pokud je specifikovaný rozsah 0–300, délka rozsahu je tedy 301 a počet bitů potřebných k reprezentaci tohoto rozsahu je podle vzorce $\lceil \log_2 délka\_rozsahu \rceil$ vypočten na hodnotu 9 bitů.

6. Nyní je zašifrované pole bajtů rozděleno po $\lceil \log_2 délka\_rozsahu \rceil$ bitech. Takto nově vzniklá čísla jsou převedena zpět do desítkové soustavy.

7. Nyní jsou tato čísla namapována do specifikovaného rozsahu. Všechna čísla jsou tedy přičtena k počáteční hranici rozsahu. Pokud je toto číslo mimo specifikovaný rozsah je od tohoto čísla odečtena délka rozsahu.

Pokud uživatel nenakonfiguroval šifrování je vynechán výše uvedený krok 3 a 4.

Ačkoliv nástroj poskytuje poměrně mnoho možností nastavení čísla portu dynamicky (např. dle aktuálního času), bezpečnost těchto možností je založená pouze na nevědomosti útočníka o jejich významu. Tomuto konceptu se říká security through obscurity. Pokud by tedy útočník odhalil význam jednotlivých čísel portů, mohl by snadno sestavit a zaslat vlastní autentizační sekvenci, která by byla úspěšně přijata.

Nástroj sice poskytuje možnost zašifrování sekvence a zajistí tak anonymitu odesílaných čísel portů, nepoužívá však jakýkoliv autentizační kód k ověření integrity a autenticity dat. Útočník by tak mohl snadno pozměnit zašifrovanou zprávu. Nástroj není navíc bezpečný vůči útoku podvrhnutí zdrojové IP adresy.

Určitou možností by mohlo být použití na místo portů direktivu „P-n“, která nastaví do sekvence n-té číslo z příkazové řádky. Nějaký externí program by mohl vytvořit zašifrovanou zprávu včetně zdrojové IP adresy, opatřit ji autentizačním kódem, tuto zprávu zakódovat jako sekvenci čísel portů a sekvenci vložit jako argumenty programu.

1.4.9 Shrnutí existujících implementací

V následující tab. 1.5 si lze prohlédnout srovnání implementací konceptu port knocking a single packet authorization.

Většina analyzovaných nástrojů poskytovala serverovou část pouze pro unixové operační systémy, ačkoliv klientské části byly dostupné i pro jiné operační systémy, včetně operačního systému MS Windows.

Bohužel kvalita implementací serverových částí nástrojů dostupných pro operační systém MS Windows byla velice špatná. Většinou se jednalo pouze o prototypy implementující vybraný koncept. Co se týče kryptografické bezpečnosti, tyto nástroje poskytovaly velmi slabou bezpečnost, například žádná z implementací nevyužívala kombinaci šifry a autentizačního kódu.

Nejlepší implementací, co se týče počtu funkcí a kryptografické bezpečnosti, byl nástroj fwknopd, tento nástroj však využívá UDP datagram pro přenos autentizační zprávy. Jelikož je nutné analyzovat a číst obsah přijímaného paketu, může být tento nástroj více zranitelný vůči útoku buffer overflow.

Nástroj knockknock měl také velmi dobrou kryptografickou bezpečnost, využíval jediného TCP SYN paketu, který obsahoval autentizační zprávu v hlavičkách TCP a IP a navíc byl velmi jednoduchý.

Název	knockd	knock-knock	fwknopd	door-mand	Py-Port Knocking project	win-Knocks	KnockKnock – Port Knocking for Windows	It's me (pouze klient)
Operační systém	UN*X (klient navíc pro MS Windows, iOS a Android)	Linux	UN*X (klient navíc pro MS Windows, iOS a Android)	UN*X (klient navíc MS Windows)	UN*X, MS Windows	MS Windows	MS Windows	MS Windows
Způsob detekce paketů	libpcap	žurnál	libpcap / čte příchozí UDP pakety	libpcap	libpcap	libpcap	žurnál	
Koncept	port knocking	SPA	SPA	SPA	port knocking	port knocking	port knocking	port knocking
Zabezpečení	možnost jednorázových sekvencí	šifrování + MAC	šifrování + MAC	pouze MAC	pouze statické sekvence	pouze šifrování	pouze statické sekvence	pouze šifrování

Tabulka 1.5: Porovnání různých implementací konceptu port knocking či single packet authorization.

1.5 Získávání informací o paketu v OS MS Windows

V následující části budou diskutovány možnosti získávání informací o příchozích paketech v operačním systému MS Windows.

1.5.1 Windows Firewall

Nástroje knockknock (viz 1.4.2) a KnockKnock – Port Knocking for Windows (viz 1.4.7) získávají informace o příchozích paketech z logovacího souboru. Tento logovací soubor vytváří firewall během své činnosti, kde zaznamenává informace o povolených či odmítnutých paketech.

Výchozím firewallem v operačním systému MS Windows je program Windows Firewall. Windows Firewall umožňuje záznam informací o paketech, avšak oproti programu iptables, který slouží jako výchozí firewall v operačním systému Linux, poskytuje velmi omezenou možnost konfigurace logování.

V možnostech nastavení lze ovlivnit pouze cestu k souboru, kam se budou události zapisovat, maximální velikost souboru (pokud se překročí maximální velikost souboru, nejstarší záznamy začnou být nahrazovány novými), možnost zaznamenávat odmítnuté pakety a možnost zaznamenávat úspěšně přijaté pakety.

Nastavení Windows Firewall lze provádět zvlášť ve třech profilech. Tyto tři profily se nazývají: doménový, soukromý a veřejný. Doménový profil se použije pokud je počítač připojen v doméně Active Directory, tento profil je tedy aktivní např. v podnikových sítích. Nastavení doménového profilu však může být dále upraveno pomocí doménové politiky (group policy). Další dva profily soukromý a veřejný jsou aktivní pokud je počítač připojen v soukromé či veřejné síti. Lze tak docílit rozdílného nastavení a míry zabezpečení v různých sítích.

Konfigurovat Windows Firewall lze buď pomocí grafického či řádkového rozhraní. Řádkové rozhraní dle [46] umožňuje přidávat či odebírat pravidla, vypisovat nastavenou konfiguraci či upravovat ostatní vlastnosti. Následující příklady ukazují některé možnosti využití řádkového rozhraní.

Následující příkaz nakonfiguruje ukládání informací o paketech do souboru C:\fw.log v aktuálním profilu.

```
netsh advfirewall set currentprofile logging filename "C:\fw.log"
```

Následující příkaz vypíše všechna pravidla.

```
netsh advfirewall firewall show rule name=all
```

Při konfiguraci pravidel firewallu lze využít poměrně velké množství parametrů např. zdrojová a cílová IP adresa, zdrojový a cílový port, směr paketu, typ protokolu, síťové rozhraní, atd. Při nastavování pravidel je také možné přímo zadat cestu k programu a povolit či zakázat tak jeho komunikaci.

Logovací soubor obsahuje na začátku souboru hlavičku obsahující verzi programu Windows Firewall, použitý časový formát a popis jednotlivých polí záznamu.

Každý řádek může obsahovat následující položky:

- datum,
- čas,
- akce, která se provedla s paketem,
- protokol,
- zdrojová IP adresa,
- cílová IP adresa,
- zdrojový port,
- cílový port,
- velikost paketu v bajtech,
- TCP příznaky,
- TCP sekvenční číslo,
- TCP potvrzovací číslo,
- TCP velikost okénka,
- typ ICMP,
- kód ICMP,
- dodatečná informace,
- směr paketu.

Každý záznam v logovacím souboru tak poskytuje poměrně mnoho informací o TCP/IP paketu. Nástroj knockknock (viz 1.4.2) odesílá autentizační paket, který v TCP a IP hlavičce obsahuje 12 bajtů zašifrovaných dat. Využity jsou položky: sekvenční číslo (4 bajty), potvrzovací číslo (4 bajty), délka okna (2 bajty) a IP identifikátor (2 bajty). Bohužel IP identifikátor nelze ze záznamu nijak získat, alternativou by tak mohlo být vložit 2 bajty, určené do položky IP identifikátor, do zdrojového portu (také 2 bajty).

Bohužel při analýze bylo zjištěno, že existuje poměrně velká prodleva mezi samotným přijetím (odesláním) paketu a jeho zaprotokolováním do logovacího souboru.

Měření probíhalo tak, že se sledoval logovací soubor s nastaveným filtrem, vypisující pouze relevantní záznamy. Toho bylo docíleno pomocí standardních unixových nástrojů (které byly doinstalovány do operačního systému MS Windows) tail a grep.

Filtr pro sledování relevantních záznamů byl spuštěn pomocí následujícího příkazu.

```
tail -f fw.log | grep "92.240.168.71"
```

A následně byl spuštěn program ping k odeslání ICMP paketů.

```
ping 92.240.168.71
```

Nakonec byla změřena prodleva mezi spuštěním programu ping a zobrazením odpovídajících záznamů v logovacím soubor. Při měření se délka prodlevy se poměrně lišila, pohybovala se v rozmezí 25–64 sekund, průměrně 35 sekund.

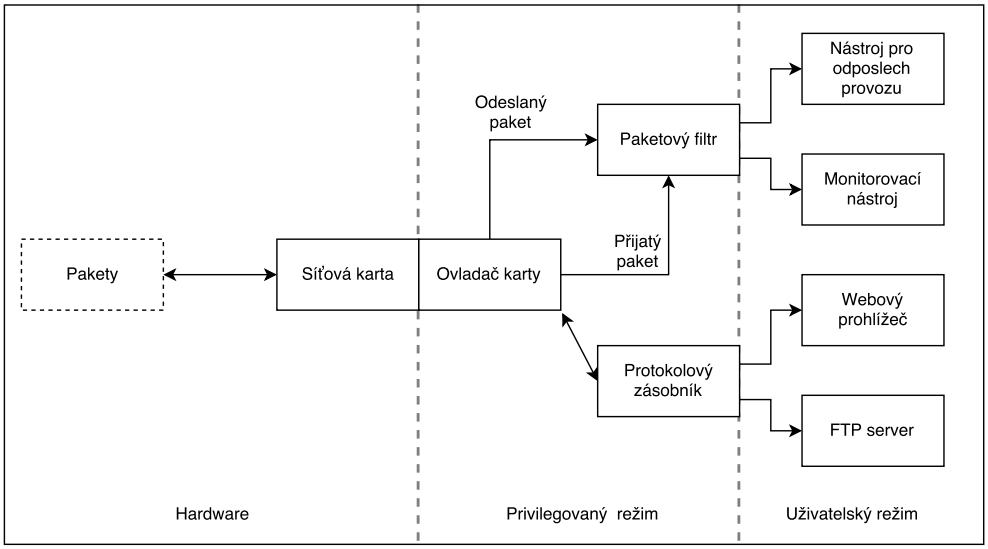
1.5.2 Knihovna WinPcap

Knihovna WinPcap [47] je implementace knihovny pcap pro operační systém MS Windows. Obě knihovny jsou srovnatelné, co se týče jejich vlastností, ale například dle [48] knihovna WinPcap umí oproti knihovně pcap zasílat vlastní pakety.

Knihovna je napsaná v programovacím jazyce C, ale je dostupná i pro další jazyky (např. Python, Java, Ruby, atd.) jako wrapper.

Knihovna pcap (resp. WinPcap) je dle [49] použita v mnoha nástrojích, mezi nejznámější patří např. Wireshark [50] nebo Tcpdump [51]. Dle [48] lze knihovnu WinPcap použít pro následující druhy nástrojů:

- analyzátory sítě a protokolů,
- síťové monitorovací nástroje,
- nástroje pro záznam síťového provozu,
- generátory síťového provozu,
- směrovač či síťový most vytvořený v uživatelském režimu,
- síťové systémy detekující útoky,
- síťové skenery,
- bezpečností nástroje.



Obrázek 1.10: Diagram jednotlivých komponent v procesu odposlouchávání paketů. Převzato z [2].

Na obr. 1.10 si lze prohlédnout jednotlivé komponenty, které se podílejí na procesu odposlouchávání paketů.

Dle [2] vypadá standardní postup práce s knihovnou následovně:

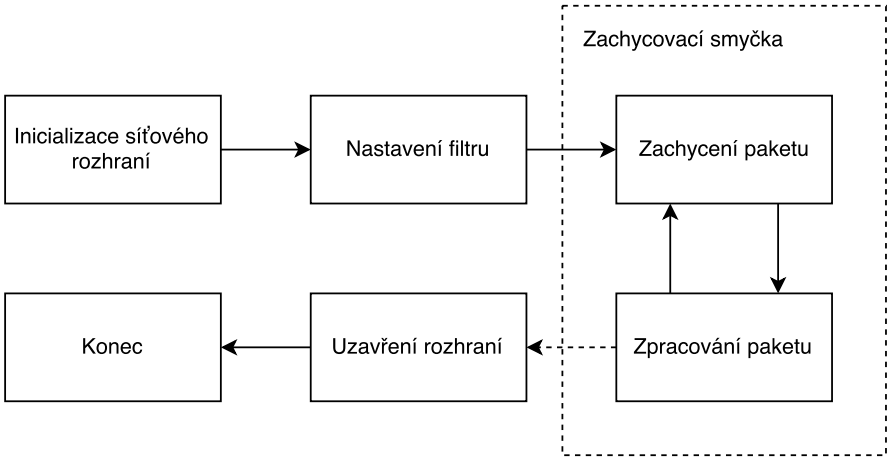
1. Nejprve se specifikuje síťové zařízení, ze kterého budou odposlouchávány pakety. Toho lze docílit např. pomocí funkce pcap_lookupdev, která vyhledá první síťové zařízení. Tato funkce se použije například v případě, kdy uživatel nespecifikoval konkrétní síťové zařízení.

2. Následně se otevře síťové zařízení pro čtení pomocí funkce pcap_open_live.

3. S otevřeným zařízením lze nyní číst pakety, nejčastěji pomocí funkcí pcap_next či pcap_loop. Funkce pcap_next vrátí první paket, který dorazí k síťovému zařízení. Funkce pcap_loop bude zpracovávat zadaný počet paketů, pokud se jako počet paketů zadá záporné číslo, bude je tato funkce zpracovávat dokud nenarazí na chybu. Zpracování probíhá tak, že se funkci předá ukazatel na obslužnou funkci. Tato obslužná funkce je zavolána pokaždé, pokud dorazí nový paket. Obslužná funkce dostane jako vstupní argument odposlechnutý paket a další uživatelem specifikované parametry.

Na obr. 1.11 si lze prohlédnout standardní postup práce s knihovnou pcap či WinPcap.

Při odposlouchávání paketů lze také využít filtru, který zajistí, že budou odposlouchávány pouze požadované pakety. Nejprve je nutné filtr zkompilovat



Obrázek 1.11: Standardní postup práce s knihovnou pcap. Převzato z [2].

pomocí funkce pcap_compile. Dle [48] tato funkce převede filtrovací výraz na program, který bude následně interpretován ve filtrovacím jádru. Následně je tento zkompilovaný filtr nastaven pomocí funkce pcap_setfilter. Nakonec lze odposlouchávat pouze relevantní pakety pomocí výše zmíněných funkcí pcap_next či pcap_loop.

Samotný odposlechnutý paket je vrácen jako pole bajtů obsahující vrstvy linkovou a vyšší. Paket tedy obsahuje hlavičky protokolů vrstvy linkové (např. Ethernet), síťové (např. IP), transportní (např. TCP) a aplikační, tedy samotné tělo paketu.

Je na uživateli, aby správně detekoval a zpracoval protokoly v daných vrstvách. Nejprve musí uživatel zjistit protokol linkové vrstvy, to lze zjistit pomocí funkce pcap_datalink, který vrátí protokol linkové vrstvy daného síťového adaptéru (nejčastěji Ethernet či 802.11). Následně musí uživatel detekovat a vyjmout hlavičku protokolu síťové a transportní vrstvy.

Dle [2] by měl uživatel pečlivě ověřovat přijatý paket, protože paket může být poškozený nebo nevytvořený dle standardu. Dále by uživatel neměl slepě věřit informaci o enkapsulovaném protokolu, měl by ověřovat TCP a IP kontrolní součty a kontrolovat, zda bajty v hlavičkách opravdu reprezentují dané informace.

1.5.3 Windows Filtering Platform

Windows Filtering Platform je aplikační programové rozhraní pro tvorbu nástrojů filtrující síťový provoz. Dle [52] lze použít WFP k tvorbě následujících nástrojů:

- bezpečnostní brány (firewall),
- síťové systémy detekující útoky,

- antivirové programy,
- nástroje pro monitorování síťového provozu,
- nástroje pro rodičovskou kontrolu.

Dle [52] WFP nahrazuje předešlé technologie k filtrování paketů jako Transport Driver Interface (TDI), Network Driver Interface Specification (NDIS) a Winsock Layered Service Providers (LSP).

WFP je kompatibilní s operačními systémy Windows Vista, Windows Server 2008 a vyšší. WFP bylo dle [52] použito například pro implementaci bezpečnostní brány „Windows Firewall s pokročilým zabezpečením“, která je součástí operačních systémů Windows Vista, Windows Server 2008 a vyšší.

WFP obsahuje programové rozhraní pro privilegovaný a uživatelský režim. Funkce dostupné v uživatelském režimu mají omezenou funkcionalitu (např. pakety nemohou být modifikovány), naopak funkce v privilegovaném režimu dokáží modifikovat zachycené pakety nebo je plně analyzovat.

Pro využití programového rozhraní z privilegovaného režimu je nutné vytvořit tzv. callout driver. Tento ovladač bude následně komunikovat s ovladačem řídící síťovou komunikaci.

Implementace takového ovladače je však netriviální problém, navíc pro bezproblémové fungování musí být ovladač digitálně podepsán.

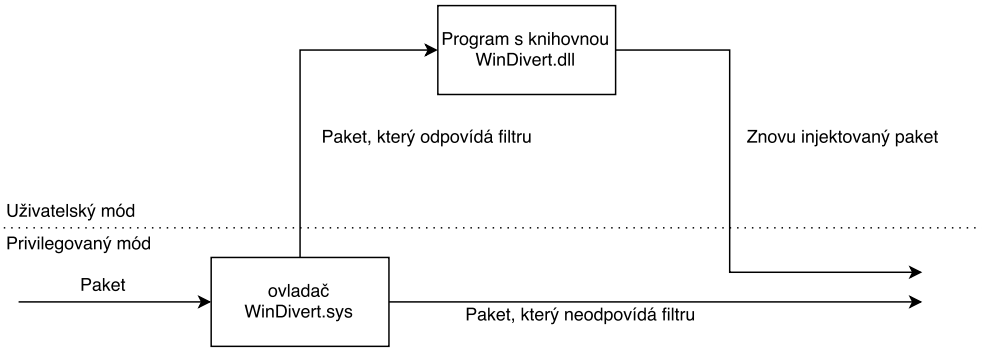
Nástroj WinDivert [53] poskytuje některou funkcionalitu z privilegovaného režimu WFP v uživatelském režimu a navíc obsahuje i digitálně podepsaný callout driver.

Nástroj WinDivert obsahuje jednak dynamickou knihovnu (DLL), která poskytuje programu rozhraní pro zachytávání, odposlouchávání, filtrování, modifikaci a (re)injektování paketů, a také vlastní ovladač, který komunikuje se síťovým ovladačem.

Diagram funkcionality si lze prohlédnout na obr. 1.12. Ovladač v privilegovaném režimu odkloní pakety odpovídající specifikovanému filtru do uživatelského režimu do aplikace s načtenou knihovnou WinDivert. Následně je rozhodnuto, jak bude s paketem naloženo, zdali bude zahozen, změněn, přečten nebo znovu injektován do síťového zásobníku.

Knihovna poskytuje velmi jednoduché rozhraní. Základní postup práce s knihovnou je následující:

1. Nejprve se otevře rozhraní se specifikovaný filtrem pomocí funkce WinDivertOpen.
2. Přijaté pakety, které odpovídají filtru se ukládají do fronty. Z této fronty je lze vyjmout pomocí funkce WinDivertRecv, která vrátí paket jako pole bajtů.
3. Zachycený paket obsahuje síťovou a vyšší vrstvu. Pomocí funkce WinDivertHelperParsePacket lze rozdělit zachycený paket na odpovídající IP,



Obrázek 1.12: Diagram funkcionality nástroje WinDivert, převzato z [3].

ICMP, TCP či UDP hlavičky. Tato funkce navíc vrátí ukazatel na první bajt obsahující samotná data z aplikační vrstvy. Funkce však neprovádí žádné další kontroly.

4. Paket lze dále modifikovat a pomocí funkce WinDivertSend ho lze znovu odeslat.

Funkce WinDivertOpen může fungovat v několika režimech. Výchozí režim zachycuje pakety, přesměruje je do aplikace ke zpracování, ale samotné odeslání paketu je již v pravomoci aplikace. V odposlouchávacím režimu aplikace zachycené pakety zkopíruje a kopii odešle do programu ke zpracování, původní paket je nezměněný odeslán dále. Poslední filtrovací režim zachycené pakety zahazuje, lze tak docílit jednoduchého paketového filtru.

Kapitola 2

Návrh

2.1 Vyhodnocení analýzy

2.1.1 Výběr vzorové implementace

Jelikož z analýzy existujících řešení vyplynulo, že neexistuje žádná uspokojivá implementace pro operační systém MS Windows, implementovaný nástroj bude určen pro MS Windows.

Při návrhu vlastního nástroje byl použit nástroj knockknock (viz 1.4.2) jako výchozí implementace. Hlavní vlastnosti tohoto nástroje, které budou převzaty jsou následující:

- Koncept single packet authorization.
- Zašifrování autentizačního paketu pomocí moderní symetrické šifry AES.
- Autentizační kód zprávy vytvořený pomocí HMAC.
- Vložení zašifrované zprávy do hlaviček TCP a IP.
- Maskování autentizačního paketu jako běžný TCP SYN paket.
- Obrana proti útoku přehráním.
- Klient bude podporovat automatické zasílání autentizačních paketů.

Nástroj knockknock je distribuován pod licencí GPL, tato licence umožňuje využívat či měnit stávající dílo za předpokladu, že nové dílo bude distribuováno pod stejnou licencí.

2.1.2 Shrnutí bezpečnostních požadavků

Mezi hlavní identifikované slabiny nástroje knockknock patří:

- Neověřování zdrojové IP adresy autentizačního paketu.
- Nenaplnění slibované vlastnosti IND-CCA.

Jelikož nástroj knockknock (viz 1.4.2) sliboval vlastnost IND-CCA, v této části bude diskutováno, co je to IND-CCA a jak této vlastnosti dosáhnout.

Notace IND-CCA znamená „Indistinguishability under Chosen Ciphertext Attack“ tedy „nerozlišitelnost pod útokem s možností výběru šifrovaného textu“, jelikož notace IND-CCA implikuje dle [54] notaci IND-CPA, bude nejprve definována notace IND-CPA.

Ačkoliv notace IND-CPA a IND-CCA byly původně definovány pro asymetrické šifry, definice následujících notací budou definovány pro symetrické šifry.

Notace IND-CPA je definována dle [55] jako hra. Uvažujme protivníka, který má přístup k šifrovacímu orákulu, avšak nemá přístup k šifrovacímu klíči K . Protivník může do orákula vkládat dvojici stejně dlouhých otevřených textů (M_0, M_1) a orákulum vrátí protivníkovi vždy jednu zašifrovanou zprávu C. Orákulum bylo inicializováno s náhodnou hodnotou $b \leftarrow \{0, 1\}$, tato hodnota určuje, který z vložených otevřených textů bude zašifrován (tato hodnota zůstává po počáteční inicializaci pořád stejná). Protivník může vkládat do orákula dvojice stejných otevřených textů (počet dotazů je však omezen) a poté se musí rozhodnout o hodnotě b. Pokud protivník dokáže uhodnout hodnotu b s pravděpodobností větší jak $\frac{1}{2}$, šifrovací schéma není bezpečné.

Formálně je tato hra dle [55] definována následovně.

```
procedura Inicializuj():
        K = vygeneruj_nahodny_klic()
        b = vyber_nahodne(0, 1)

procedura SifrovaciOrakulum(M0, M1):
        return C = Zasifruj(K, Mb)

procedura Uhadni(b'):
        return (b == b')
```

Další notací je IND-CCA, která vychází z předchozí definice, protivník má však v tomto případě k dispozici i dešifrovací orákulum. Do tohoto dešifrovacího orákula však protivník nesmí vložit zašifrovaný text C, který předtím získal ze šifrovacího orákula. Pokud by nebylo zavedeno toto omezení, hra by poté byla pro protivníka triviální. Úkol protivníka je opět stejný, uhádnout hodnotu b s pravděpodobností více jak $\frac{1}{2}$.

Formálně je hra dle [55] totožná s předchozí definicí, protivník má navíc k dispozici dešifrovací orákulum definované následovně.

```
procedura DesifrovaciOrakulum(C):
        return M = Desifruj(K, C)
```

Dle [34] šifrovací schéma AES v módu CTR splňuje notaci IND-CPA, ale nesplňuje notaci IND-CCA. Ke splnění notace IND-CCA je nutné zkombinovat šifrovací a autentizační schéma. Existuje však několik způsobů, jak kombinovat zprávu a autentizační kód zprávy, těmto způsobům se říká šifrovací paradigmata.

Tato paradigmata jsou dle [34] definována následovně. Uvažujeme zprávu M, šifrovací algoritmus $\mathcal{E}$, autentizační schéma $\mathcal{T}$ a dva různé šifrovací klíče: K_e jako klíč pro šifrování a K_m jako klíč pro autentizační kód zprávy. Operátor $\|$ značí spojení řetězců. Rozlišujeme následující tři šifrovací paradigmata:

- zašifruj-a-autentizuj (encrypt-and-MAC): $\mathcal{E}(K_e, M) \parallel \mathcal{T}(K_m, M)$
- autentizuj-pak-zašifruj (MAC-then-encrypt): $\mathcal{E}(K_e, M \parallel \mathcal{T}(K_m, M))$
- zašifruj-pak-autentizuj (encrypt-then-MAC): $C \parallel \mathcal{T}(K_m, C)$, kde $C = \mathcal{E}(K_e, M)$

Dle [34] při použití šifrovacího schémata AES v módu CTR v kombinaci s autentizačním schématem HMAC za použití paradigmata zašifruj-pak-autentizuj (encrypt-then-MAC) lze dosáhnout vlastnosti IND-CCA.

2.1.3 Možnosti odposlouchávání paketů

Z analýzy možných přístupů k získávání informací o paketech v systému MS Windows, které byly diskutovány v sekci 1.5, vyšla jako nejlepší možnost WFP s knihovnou WinDivert.

Ačkoliv nejbezpečnějším způsobem by bylo pravděpodobně čtení souboru, který generuje Windows Firewall, díky velké prodlevě mezi přijmutím paketu a jeho zaprotokolováním do souboru je tento způsob prakticky nepoužitelný. Navíc z logovacího souboru Windows Firewall nelze získat některé položky TCP či IP hlavičky, což by znamenalo omezení velikosti přenášené zprávy a tudíž by se musel dále zkrátit autentizační kód. To by mohlo vést ke snazšímu padělání autentizačního kódu.

Využití WFP a knihovny WinDivert navíc poskytne snadnou možnost pro posílání vlastních autentizačních paketů, této funkcionality bude využito při implementaci klientské části nástroje.

Jelikož bude možné využít WFP a knihovny WinDivert, budou moci být využity další položky TCP hlavičky. Díky tomu se bude moci zvětšit velikost přenášené zprávy a zvětšit tak velikost autentizačního kódu.

2.2 Výběr programovacího jazyka a dalších nástrojů

Jelikož původní nástroj knockknock (viz 1.4.2) byl implementován v jazyce Python ve verzi 2, bude i část nově implementovaného nástroje napsána v jazyce Python ve verzi 2.

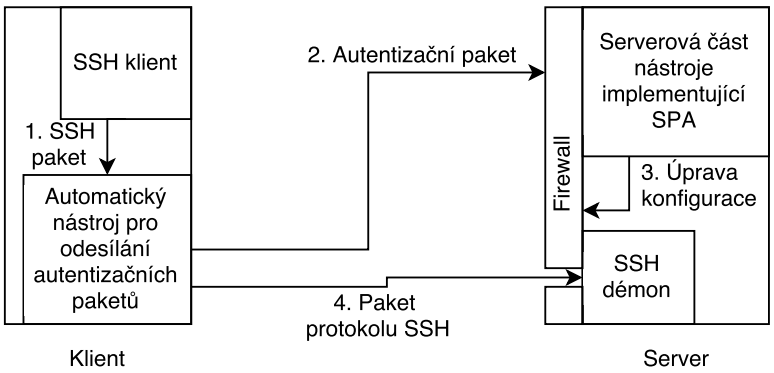
Kvůli využití WFP a knihovny WinDivert bude část nástroje napsána v jazyce C++.

2.3 Návrh nástroje

Celý nástroj bude obsahovat dvě části: klientskou a serverovou. Klientská část bude navrhnuta tak, aby podporovala automatické posílání autentizačního paketu na konfigurované servery.

Na rozdíl od výchozí implementace klientské části nástroje knockknock, která využívala jednoduchý SOCKS proxy server k automatickému odesílání autentizačních paketů (což vyžadovalo podporu samotné aplikace či dodatečnou aplikaci pro přesměrování komunikace), nově navržený nástroj bude podporovat automatické odesílání autentizačních paketů bez nutnosti dodatečného nastavení samotné aplikace či využití dodatečných aplikací.

Pokud tedy klientská část nástroje detekuje odchozí TCP či UDP paket na nakonfigurovaný server, zastaví jej, vytvoří odpovídající autentizační paket, odešle autentizační paket a po prodlevě znovu odešle odchozí paket. Ukázku funkcionality klientské a serverové části si lze prohlédnout na obr. 2.1, která znázorňuje příklad při využití protokolu SSH.



Obrázek 2.1: Návrh spolupráce klientské a serverové části nástroje.

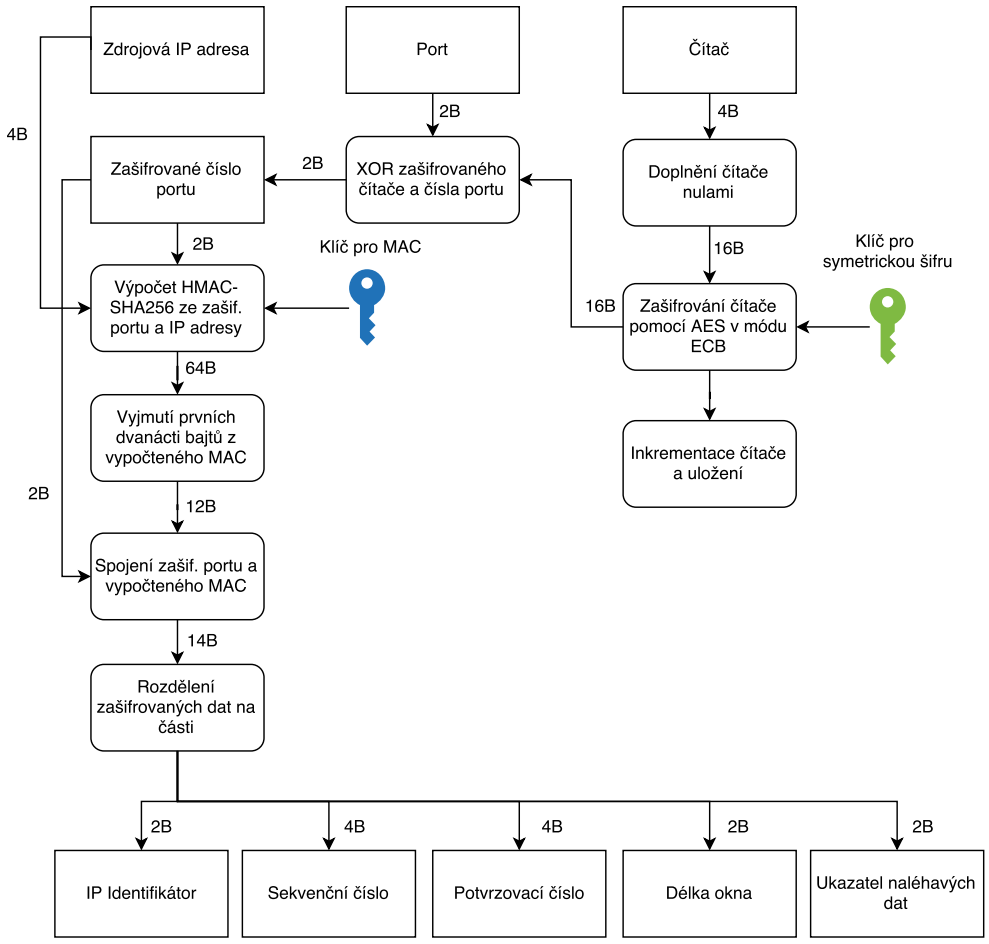
Proces vytváření autentizačního paketu je velmi podobný jako v původním nástroji knockknock. Změněno bylo však paradigma na zašifruj-pak-autentizuj a algoritmus pro vytvoření autentizačního kódu zprávy byl změněn z HMAC-SHA-1 na novější HMAC-SHA-256 (klíč pro HMAC-SHA-256 byl navíc zvětšen z původních 16 na 32 bajtů).

Využití WFP a knihovny WinDivert dovolilo využít další položky v TCP hlavičce – ukazatel naléhavých dat. Tato další položka dovolila zvětšit objem přenášené zprávy z 12 na 14 bajtů. Tyto získané dva bajty byly použity pro autentizační kód zprávy, který má nyní velikost 12 bajtů z původních 10.

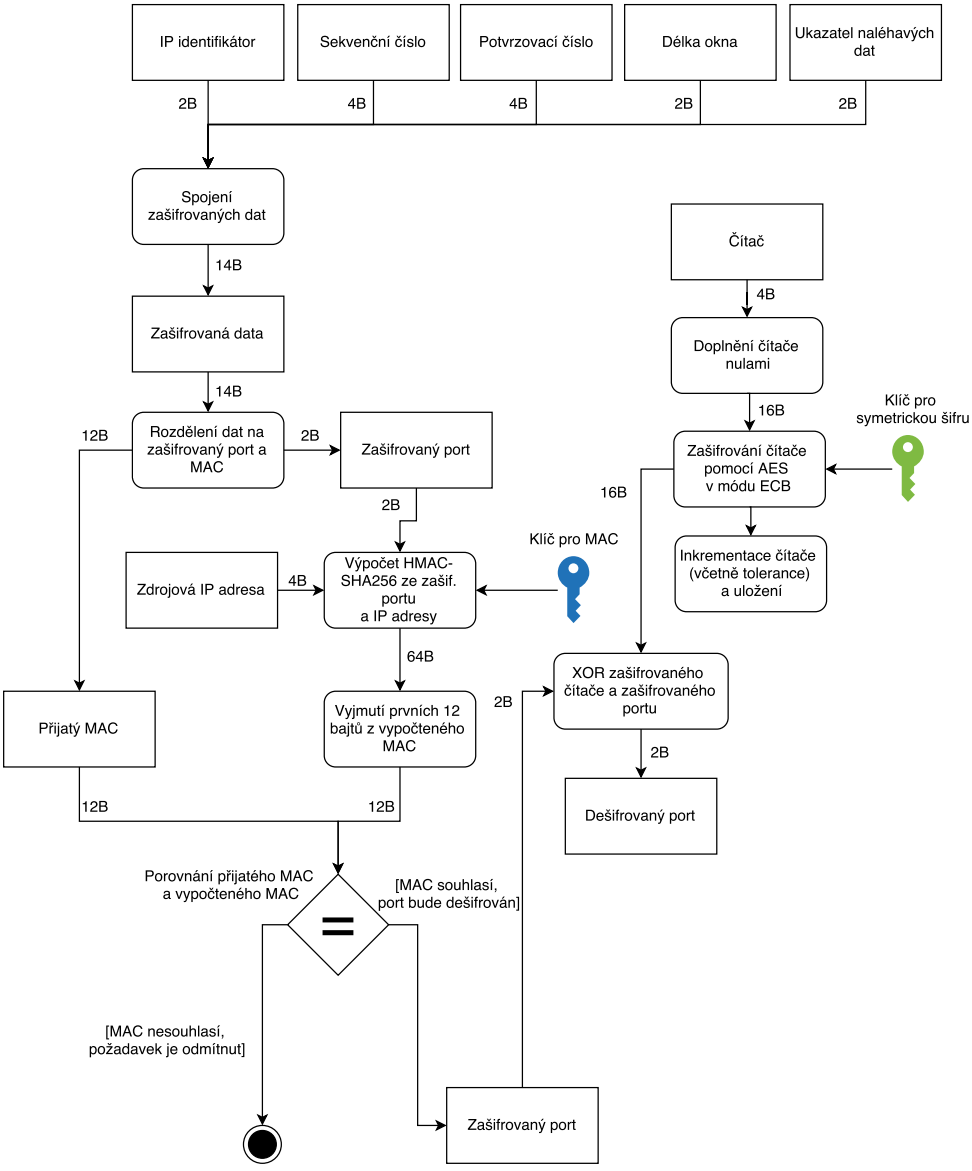
Ačkoliv bylo zvažováno použití zdrojového portu autentizačního paketu pro získání dalších dvou bajtů zprávy, tento návrh byl posléze zamítnut z důvodu možné interference se síťovým překladem adres. Síťový překlad adres by mohl změnit zdrojový port autentizačního paketu za jiný a tím by byl autentizační paket zneplatněn.

Další změnou bylo zahrnutí zdrojové adresy do autentizačního kódu zprávy, takto bude moci serverová část nástroje ověřit původního odesilatele autentizačního paketu. Celý diagram vytváření autentizačního paketu si lze prohlédnout na obr. 2.2.

Serverová část nástroje bude také velmi podobná původní implementaci nástroje knockknock. Hlavní změnou je změna šifrovacího paradigmatu, jak lze vidět na obr. 2.3, který popisuje proces přijetí paketu, jeho autentizaci a dešifrování.



Obrázek 2.2: Diagram vytváření autentizačního paketu.



Obrázek 2.3: Diagram příjmu autentizačního paketu a dešifrování zprávy.

Kapitola 3

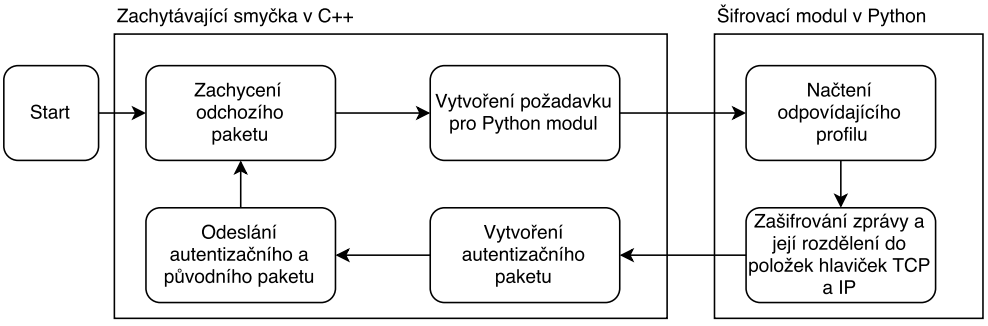
Implementace

3.1 Implementace klientské části

Z důvodu využití již existující funkcionality, zejména načítání profilů, konfigurace, samotné šifrování a rozdělení zašifrované zprávy, bylo využito v klientské části Python modulu, který obsahoval modifikace zmíněné v návrhu.

Naštěstí jazyk C++ umožňuje snadné propojení s jazykem Python, viz [56]. Pomocí tohoto rozhraní je inicializován vytvořený Python modul, je mu předán požadavek na zašifrování zprávy (ten obsahuje číslo portu, který má být na straně serveru otevřen a adresu či doménové jméno serveru) a tento Python modul vrátí hodnoty jednotlivých položek do hlaviček TCP a IP.

Obsah hlaviček se následně vloží do vytvořeného autentizačního paketu, tento autentizační paket se odešle a po nakonfigurované prodlevě se odešle původní paket. Celou funkcionalitu si lze prohlédnout na obr. 3.1.



Obrázek 3.1: Diagram architektury klientské části.

Jelikož je odesílání autentizačního paketu plně v režii nástroje (za pomoci knihovny WinDivert), nemůže se klientská verze nástroje spolehnout na některé funkce jádra operačního systému, které usnadňují odesílání paketů. Mezi tyto funkce patří například počítání kontrolních součtů IP, TCP a UDP hlaviček a také nastavení zdrojového portu. Knihovna WinDivert však poskytuje funkci WinDivertHelperCalcChecksums, která dopočítá příslušné kontrolní součty.

Nastavení zdrojového portu je však zcela v režii nástroje, v první verzi implementace byl zdrojový port autentizačního paketu stejný, jako zdrojový port odchozího paketu. Tato implementace však v případě existence dlouhodobého spojení odesílala autentizační paket ze stále stejného zdrojového portu. Používání duplicitních zdrojových portů by usnadnilo detekci těchto autentizačních paketů. Proto se zdrojový port nastavuje náhodně pomocí uniformního generátoru celých čísel, který poskytuje standardní knihovna random ze standardu C++11. Rozsah generovaných portů je nastaven na 49 152 – 65 535, dle doporučení [57].

Aby nedocházelo k neustálému odesílání autentizačních paketů při každém odchozím paketu, je na straně klienta držena v paměti hašovací tabulka reprezentující aktuální spojení. Tato hašovací tabulka používá jako klíč pro přístup strukturu reprezentující spojení, tato struktura obsahuje zdrojovou, cílovou adresu, cílový port a použitý transportní protokol.

Zdrojový port není použit záměrně, jelikož některé protokoly aplikační vrstvy (např. HTTP bez nastavené možnosti keep-alive) otevírají nové TCP spojení pro každý požadavek. Při přístupu na jednu webovou stránku by požadavky na různé zdroje (obrázky, styly, skripty, atd.) vygenerovaly mnoho nových spojení s různými zdrojovými porty. To by vedlo k zasílání redundantních autentizačních požadavků pro každý zdroj.

V hašovací tabulce je jako hodnota k danému klíči definováno časové razítko posledního odeslání autentizačního paketu. Při odchozím paketu je tedy z hašovací tabulky zjištěno, kdy byl naposledy odeslán autentizační paket, pokud je rozdíl mezi aktuálním časem a časem odeslání posledního autentizačního paketu větší než nakonfigurovaná mez, je znovu odeslán autentizační paket. Navíc je aktualizováno časové razítko v hašovací tabulce.

Ačkoliv by se mohlo zdát, že by se výše zmíněný problém dal vyřešit sledováním pouze odchozích TCP SYN paketů na nakonfigurované servery a pouze v těchto případech odesílat autentizační pakety, tento přístup přináší několik problémů.

Zejména by se tímto způsobem znemožnilo odesílání autentizačních paketu při využívání UDP. Navíc jak již bylo řečeno výše, některé protokoly aplikační vrstvy vytváří mnoho nových TCP spojení (a tedy posílají mnoho nových TCP SYN paketů), to by opět vedlo k posílání redundantních autentizačních paketů.

Jelikož klientská část držící tuto strukturu v paměti může běžet velmi dlouho, mohlo by se stát, že by struktura obsahovala velké množství starých spojení a zabírala tím místo v paměti. Z tohoto důvodu klientská část vytvoří při svém startu další vlákno, které v definovaných intervalech prochází hašovací tabulku a odstraňuje stará spojení. Ve výchozím nastavení vlákno odstraní spojení, pokud byl autentizační paket odeslán déle než před dvěma minutami.

3.2 Implementace serverové části

Implementace serverové části vychází z původní architektury nástroje knock-knock-daemon (viz obr. 1.3). V původní implementaci se nástroj skládal ze dvou částí – jedna část četla logovací soubor a autentizovala pakety a druhá část upravovala konfiguraci firewallu.

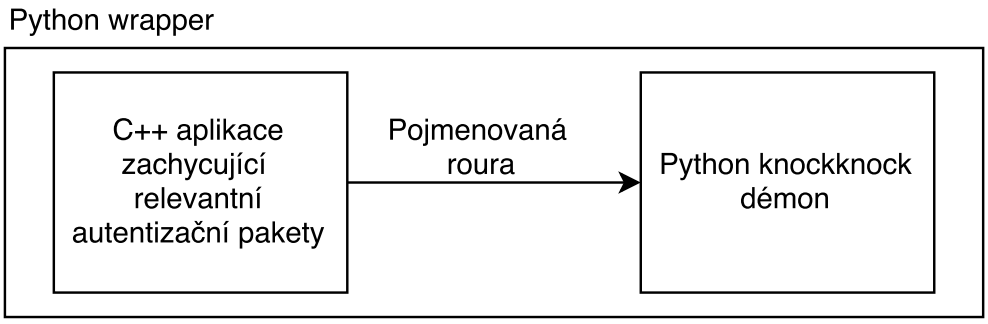
Část, která četla logovací soubor, byla zaměněna za program implementovaný v C++, který odposlouchává příchozí autentizační pakety. Tento program po úspěšném přijetí autentizačního paketu vyjme relevantní položky z hlaviček TCP a IP. Z těchto položek je vytvořena zpráva, která je zaslána skrz pojmenovanou rouru do druhého procesu. Formát posílané zprávy skrz pojmenovanou rouru vychází z formátu, který generuje program iptables při přijetí paketu.

Z tohoto důvodu nemusela být část nástroje čtoucí záznamy z logovacího souboru výrazně modifikována, pouze se změnil způsob, kterým nástroj získává relevantní záznamy o příchozím autentizačním paketu.

Jelikož bylo nutné dosáhnout meziprocesové komunikace, byly zvažovány různé možnosti meziprocesové komunikace. V první verzi implementace bylo využito souboru. Proces odposlouchávající pakety zapisoval záznamy o autentizačních paketech do souboru, druhý proces četl z tohoto souboru nově zapsané řádky a zpracovával je. Tento způsob vycházel z původní architektury nástroje knockknock. Jelikož však nebylo nutné tato data zachovávat persistentně, byl změněn způsob meziprocesové komunikace tak, aby byla využívána pojmenovaná roura.

Výhoda pojmenované roury je, že s ní lze manipulovat pomocí stejného rozhraní jako se souborem, modifikace z původní implementace využívající soubor, na novou implementaci využívající pojmenovanou rouru, byla minimální.

Serverová část nástroje je celá inicializována z Python skriptu, který nejprve spustí v samostatném procesu část nástroje odposlouchávající autentizační pakety a následně spustí druhou část nástroje, jak lze vidět na obr. 3.2.



Obrázek 3.2: Diagram architektury serverové části nástroje.

Změnou oproti původní implementaci však bylo použití programu Windows Firewall namísto programu iptables. Ačkoliv Windows Firewall poskytuje řádkové rozhraní (CLI) pro snadnou modifikaci pravidel, Windows Firewall má odlišné chování při odstranění pravidla.

Nástroj knockknock spolupracoval s programem iptables následovně. Do programu iptables byly nejprve nastaveny výchozí pravidla pro zakázání veškeré příchozí komunikace, ale povolení odchozí komunikace a již navázané komunikace. Po obdržení autentizačního paketu bylo přidáno pravidlo povolující přístup klienta na vybraný port serveru. Zároveň bylo spuštěno na straně serveru vlákno, které bylo nejprve uspáno na specifikovanou časovou prodlevu a které následně pravidlo opět odebralo. Klient tak mohl využít nakonfigurovaného časového okna pro připojení k serveru. Po odstranění pravidla však mohl klient dále komunikovat se serverem v rámci již navázaného spojení.

Pokud tedy klient navázal úspěšné připojení pomocí SSH na server v rámci časového okna, mohl využívat SSH spojení i po odstranění pravidla, jelikož pakety SSH spojení byly programem iptables identifikované jako náležící k již navázanému spojení.

Bohužel tohoto chování nebylo možné docílit při použití Windows Firewall. Windows Firewall při odstranění pravidla okamžitě přeruší již navázané spojení. Ačkoliv by se tento problém dal vyřešit pomocí velkého časového okna (např. v řádu minut či dokonce hodin) mezi přijmutím autentizačního paketu a odstraněním pravidla, byl by tím výrazně kompromitován význam celého konceptu.

Naštěstí byl tento problém vyřešen modifikací logiky serverové části nástroje. Serverová část nástroje v paměti drží hašovací tabulku, jejíž klíč je tvořen řetězcem „IP adresa klienta:požadovaný port na otevření“, hodnotou odpovídající klíči je časové razítko posledního přijatého autentizačního paketu.

Pokud tedy serverová část nástroje přijme nový autentizační paket, vloží příslušný záznam do hašovací tabulky a upraví konfiguraci Windows Firewall. Pokud serverová část přijme autentizační paket již od existujícího klienta,

který požaduje stejný port, pouze aktualizuje časové razítko v hašovací tabulce. Na serveru je při startu spuštěné vlákno, které v nakonfigurovaných intervalech prochází hašovací tabulku a maže záznamy, které překročili nakonfigurovanou mez. Se smazáním záznamu zároveň dojde k upravení konfigurace Windows Firewall a tedy k odstranění pravidla povolující danému klientovi přístup na daný port.

3.3 Tvorba služby pro OS MS Windows

Aby bylo pro uživatele používání obou částí nástroje, co nejjednodušší, byly obě dvě části implementovány tak, aby podporovaly režim služby.

Windows služba je obdoba unixového démona, tedy programu, který běží na pozadí, bez přímého kontaktu s uživatelem. Výhodou běhu nástroje jako služby je například automatický start po startu počítače, či automatické restartování nástroje v případě jeho neočekávaného pádu. Schopnost automatického restartování nástroje může být kruciální, jelikož v případě selhání serverové části nástroje by se mohlo stát, že bude server nepřístupný i pro oprávněného uživatele.

Jelikož se klientská část skládá hlavně z C++ kódu, vytvoření služby probíhá pomocí nativního WinAPI. Nejprve se musí z kódu zavolat funkce StartServiceCtrlDispatcher, této funkci se předá tabulka s informacemi o službě. Pomocí této funkce se propojí vlákno služby s kontrolním manažerem služeb (service control manager), ten pak může vláknu zasílat kontrolní žádosti (např. žádost o zastavení služby). Následně se pomocí funkce RegisterServiceCtrlHandler zaregistruje funkce, která obslouží žádosti zaslané kontrolním manažerem služeb.

Jelikož kontrolní žádost může přijít kdykoliv a je nutné na ni, co nejrychleji odpovědět, musela být klientská část nástroje upravena tak, aby podporovala asynchronní zpracování. V klientské části nástroje zejména funkce WinDivertRecv, která zachytává odchozí pakety, blokuje vlákno do doby, dokud nepřijme paket. Naštěstí autor knihovny WinDivert poskytuje neblokující funkci WinDivertRecvEx, ta funguje téměř totožně s funkcí WinDivertRecv, avšak nečeká na příchozí paket a ihned vrátí řízení spouštějícímu vláknu. Funkci WinDivertRecvEx se navíc předá tzv. overlapped struktura, která dokáže indikovat přijatý paket.

Klientská verze byla tedy upravena tak, že se pomocí funkce WinDivertRecvEx spustí zachytnutí paketu, ale tato funkce se ihned vrátí. Následně se pomocí WinAPI funkce WaitForMultipleObjects čeká na dvě možné události. První událostí je příjem paketu, což indikuje událost uvnitř overlapped struktury. V tomto případě se následně pomocí WinAPI funkce GetOverlappedResult čeká na dokončení příjmu celého paketu a ten je následně zpracován. Druhou možnou událostí je žádost o ukončení služby, v tomto případě program přeruší hlavní zachytávací smyčku a uvolní alokované prostředky.

Serverová část je tvořena z větší části Python kódem, i ten však poskytuje, pomocí knihovny PyWin32, rozhraní k vytvoření služby. Tato knihovna poskytuje rozhraní WinAPI v Python kódu, navíc zjednodušuje a zaobaluje některé rozhraní do tříd. Tvorba služby je tedy velmi jednoduchá, pouze se vytvoří třída, která je potomkem třídy ServiceFramework a následně se implementují metody SvcStop a SvcDoRun, které se provedou při příjmu žádosti o spuštění či zastavení služby.

Stejně jako v případě klientské části i zde muselo být docíleno asynchronního zpracování. Hlavní blokující funkcí byla funkce ReadFile, která četla příchozí zprávy z pojmenované roury. Knihovna PyWin32 ovšem poskytuje neblokující verzi této funkce, které se předá overlapped struktura. Logika serverové části nástroje pro příjem zprávy z pojmenované roury a čekání na události byla analogická s logikou klientské části nástroje.

3.4 Odstranění závislosti na nainstalovaném Python interpretru

Jelikož se v obou částech nástroje využívá v určité míře kód napsaný v Pythonu (včetně dalších Python knihoven), pro snadnou instalaci a používání nástroje muselo být zajištěno, aby byly veškeré závislosti přítomné na počítači uživatele.

V serverové části toho bylo docíleno pomocí nástroje PyInstaller [58], který zabalí Python kód, včetně závislostí a vytvoří spustitelný soubor s příponou „exe“. K tomuto souboru je dále přibalen interpretr Pythonu jako dynamická knihovna (DLL) a další nezbytné závislosti. Výsledkem je tedy balíček všech nezbytných souborů nutných k běhu serverové části nástroje. Pokud si uživatel nainstaluje serverovou část nástroje, nemusí mít nainstalovaný interpretr Python kódu ani žádné další moduly.

V klientské části nástroje se využívá pouze malá část kódu napsaná v jazyce Python. Ačkoliv by bylo možné pomocí nástroje PyInstaller vytvořit jeden spustitelný soubor, který bude obsahovat veškeré závislosti, a tento soubor následně spouštět jako samostatný proces z C++ části nástroje, pro snazší komunikaci části napsané v C++ a Python části, byl Python modul ponechán ve své původní zdrojové podobě.

V C++ části se následně použije knihovna Python, která obsahuje samotný interpretr a rozhraní pro volání Python kódu. Pro odstranění všech závislostí, byl ke klientské části nástroje přibalen interpretr jazyka Python jako dynamická knihovna a ručně nalezeny veškeré další závislosti. Všechny tyto závislosti jsou následně distribuovány spolu s klientskou částí nástroje a koncový uživatel nemusí mít pro správnou funkcionalitu nainstalovaný interpretr Python kódu či jiné závislosti.

Kapitola 4

Testování

Testování nástroje probíhalo ve třech různých prostředích: ve virtuální LAN, reálné LAN a WAN.

Testované byly aplikační protokoly HTTP a RDP. Navíc bylo otestované vytvoření libovolného TCP a UDP soketu pomocí nástroje netcat pro operační systémy MS Windows [59] a následným připojením pomocí nástroje telnet [60] v případě TCP či netcat v případě UDP.

4.1 Testování ve virtuální LAN

Testování ve virtuální LAN probíhalo mezi hostitelským a několika virtuálními počítači. Hostitelský počítač byl laptop Lenovo E430 s procesorem Intel Core i3-3110M a 8 GB operační paměti. Jako operační systém na hostitelském počítači byl použit 64bitový Microsoft Windows 7 Professional.

Virtuální počítače byly vytvořeny pomocí programu Oracle VM VirtualBox [61]. Celkem byly vytvořeny dva virtuální počítače s následujícími operačními systémy:

- 64bitový Microsoft Windows Server 2016 Essentials
- 32bitový Microsoft Windows 8.1 Pro

Při testování s virtuálními počítači nemohl být otestován protokol RDP ve směru připojení z virtuálního počítače na hostitelský. Jelikož při připojení pomocí RDP je nutné odhlásit přihlášeného uživatele na cílovém počítači, což však v tomto případě je uživatel, který spouští virtuální počítač. Testování z hostitelského počítače na virtuální šlo uskutečnit bez problému.

Testování mezi dvěma virtuálními počítači nemohlo proběhnout z důvodu omezené kapacity operační paměti na hostitelském počítači.

Pro účely testování HTTP byl na virtuálním počítači s OS Windows Server 2016 nainstalován webový server IIS, následně byla v programu Windows Firewall smazána explicitní pravidla povolující přístup k portu 80. Poté byla

otestována funkčnost Windows Firewall, připojení k webovému serveru z hostitelského počítače selhalo dle očekávání. Po nainstalování serverové části implementovaného nástroje, vygenerování profilu a jeho přenesení na hostitelský počítač se z hostitelského počítače podařilo úspěšně připojit (za pomoci klientské části implementovaného nástroje) k webovému serveru IIS. Jako HTTP klient byl na hostitelském počítači použit prohlížeč Google Chrome a řádkový HTTP klient cURL.

Dále byla ověřena funkčnost autentizačního mechanismu v kombinaci s RDP na obou virtuálních počítačích. Na obou počítačích se v nastavení operačního systému povolilo příchozí připojení vzdálené plochy a následně se odebralo patřičné pravidlo z Windows Firewall, které explicitně povolovalo příchozí TCP pakety na port 3389 (což je port, který využívá RDP). Následně byla ověřena funkčnost Windows Firewall, který nepovolil příchozí pokus o přípojení ke vzdálené ploše. Po konfiguraci serverové části implementovaného nástroje na obou virtuálních počítačích, došlo k úspěšnému připojení ke vzdálené ploše z hostitelského počítače.

Nakonec byla ověřena funkčnost autentizačního mechanismu v kombinaci s nástroji telnet a netcat. Na obou virtuálních počítačích byl vytvořen soket, který naslouchal na portu 50 000 pomocí nástroje netcat. Po jeho spuštění operační systém MS Windows upozornil uživatele, zdali chce přidat patřičné pravidlo do Windows Firewall a povolit tak příchozí spojení na port 50 000, uživatel má následně možnost přidat explicitní pravidlo povolující či zakazující příchozí spojení na port 50 000. V obou případech muselo být toto explicitní pravidlo smazáno a ponechána pouze implicitní pravidla Windows Firewall. Následně byla ověřena funkčnost Windows Firewall, který nepovolil příchozí spojení na port 50 000 (dle implicitního pravidla), po konfiguraci a spuštění serverové části implementovaného nástroje došlo k úspěšnému připojení na port 50 000 pomocí nástroje telnet v případě testu TCP soketu a netcat v případě testu UDP soketu.

Výše zmíněné testování TCP a UDP soketu bylo otestováno i obráceně, tedy z virtuálního počítače na klientský, i v tomto případě připojení proběhlo v pořádku.

4.2 Testování v reálné LAN

Testování v reálné LAN probíhalo mezi dvěma fyzickými počítači, prvním byl výše zmíněný laptop Lenovo E430 a druhým byl laptop HP ProBook 4510s s procesorem Intel Core 2 Duo T5870, 4 GB operační paměti a 64bitovým operačním systémem Microsoft Windows 7 Professional. Oba dva počítače byly připojeny do bezdrátové LAN s adresou 10.0.0.0/24.

Testování připojení ke vzdálené ploše proběhlo úspěšně stejně jako v předchozím případě. Tentokrát byla však testována funkčnost připojení oběma směry.

Testování TCP a UDP soketů bylo provedeno stejným způsobem jako v případě výše, včetně obou směrů komunikace. Testování bylo i v tomto případě úspěšné.

4.3 Testování ve WAN

Při testování ve WAN nebyl k dispozici žádný počítač s veřejnou IP adresou a operačním systémem MS Windows, proto byl nástroj otestován vůči virtuálnímu privátnímu serveru (VPS) s veřejnou adresou a 64bitovým operačním systémem Ubuntu 14.04. Tento server byl hostován u společnosti Hosting90.cz.

Jelikož implementovaná serverová část nástroje není kompatibilní s operačním systémem Linux, byla jako serverová část použita původní implementace nástroje knockknock (viz 1.4.2). Na straně klienta však byl však použit nově implementovaný nástroj, avšak použitý v režimu zpětné kompatibility s původní implementací.

Na straně serveru byl nainstalován a nakonfigurován původní nástroj knockknock, následně byly přeneseny vygenerované profily na počítač klienta. Na straně serveru byl nakonfigurován firewall iptables podle instalačního návodu [29], následně byla ověřena funkčnost firewallu, ten nepovolil žádná příchozí spojení dle očekávání. Po konfiguraci implementovaného nástroje na straně klienta bylo možné se úspěšně připojit na webový server Apache. Jako HTTP klient byl použit prohlížeč Google Chrome a nástroj cURL.

Tímto testem byla zároveň ověřena zpětná kompatibilita s původní implementací.

V následující tab. 4.1 si lze prohlédnout všechny provedené testy. Označení server a klient označuje role počítačů v jejich komunikaci.

Klient	Server	Protokol
Hostitelský počítač	Virt. počítač (Windows Server 2016)	HTTP
Hostitelský počítač	Virt. počítač (Windows Server 2016)	RDP
Hostitelský počítač	Virt. počítač (Windows Server 2016)	TCP
Hostitelský počítač	Virt. počítač (Windows Server 2016)	UDP
Virt. počítač (Windows Server 2016)	Hostitelský počítač	TCP
Virt. počítač (Windows Server 2016)	Hostitelský počítač	UDP
Hostitelský počítač	Virt. počítač (Windows 8.1 Pro)	RDP
Hostitelský počítač	Virt. počítač (Windows 8.1 Pro)	TCP
Hostitelský počítač	Virt. počítač (Windows 8.1 Pro)	UDP
Virt. počítač (Windows 8.1 Pro)	Hostitelský počítač	TCP
Virt. počítač (Windows 8.1 Pro)	Hostitelský počítač	UDP
Lenovo E430	HP ProBook 4510s	RDP
Lenovo E430	HP ProBook 4510s	TCP
Lenovo E430	HP ProBook 4510s	UDP
HP ProBook 4510s	Lenovo E430	RDP
HP ProBook 4510s	Lenovo E430	TCP
HP ProBook 4510s	Lenovo E430	UDP
Lenovo E430	VPS (Ubuntu 14.04)	HTTP

Tabulka 4.1: Seznam úspěšně provedených testů.

Závěr

Cílem této diplomové práce bylo analyzovat metody komunikace mezi hostiteli pomocí filtrovaných síťových portů s cílem autentizace a úpravy konfigurace firewallu. Dále popsat rozdíly mezi koncepty port knocking a single packet authorization a analyzovat existující implementace se zaměřením na kryptografickou bezpečnost. Na základě získaných informací následně navrhnout, implementovat a otestovat nástroj pro bezpečné připojení na služby systému MS Windows.

V kapitole „Analýza“ byly nejprve diskutovány koncepty port knocking a single packet authorization a jejich vzájemné rozdíly. Dále byla analyzována existující řešení, zejména tři nejrozšířenější implementace: knockd, knockknock a fwknopd. Nakonec byly analyzovány různé přístupy k odposlouchávání síťové komunikace s cílem získání informací o příchozím paketu.

Na základě analýzy existujících implementací, byl zvolen nástroj knockknock jako výchozí implementace pro další návrh. Zároveň byly objeveny slabiny této implementace.

V kapitole „Návrh“ byly diskutovány změny původní implementace k dosažení vyšší kryptografické bezpečnosti při zachování původního jednoduchého konceptu single packet authorization a také změna nekompatibilních částí, aby byl nástroj kompatibilní s operačním systémem MS Windows.

Dle návrhu byl implementován nástroj pro bezpečnou autentizaci na serveru s cílem změn konfigurace Windows Firewall. Implementovaný nástroj poskytuje obranu proti útoku přehráním a také proti útoku MITM. Tento nástroj byl následně úspěšně otestován v několika sítích, včetně sítě WAN, na počítačích s různými verzemi operačního systému MS Windows.

Výsledkem této diplomové práce je dvojice nástrojů pro klient a server, které implementují koncept single packet authorization. Klientská část navíc podporuje automatické odesílání autentizačních paketů pro větší komfort uživatele. Nástroj lze také použít v režimu zpětné kompatibility v kombinaci s nástrojem knockknock.

Při implementaci obou částí nástroje jsem prohloubil své znalosti programování v operačním systému MS Windows a také si lépe osvojil jeho principy.

Výhled do budoucna

Při implementaci tohoto nástroje nebyl brán v potaz stále se rozšiřující protokol IPv6. Klientská část nástroje by mohla být upravena tak, aby v případě detekce odchozí komunikace využívající IPv6 odeslala autentizační paket rovněž využívající IPv6.

Literatura

[1] Aycock, J.; Jacobson, M.; aj.: Improved port knocking with strong authentication. In *Computer Security Applications Conference*, IEEE, 2005.

[2] Garcia, L. M.: Programming with Libpcap – Sniffing the Network From Our Own Application. *Hakin9*, ročník 3, č. 2, Únor 2008: s. 38–46, [cit. 2017-04-21].

[3] GitHub: *WinDivert README* [online]. 2016, [cit. 2017-04-21]. Dostupné z: `https://github.com/basil00/Divert/blob/master/README`

[4] (International Organization for Standardization: *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model* [online]. Listopad 1994, [cit. 2017-05-08]. Dostupné z: `http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip`

[5] Meyer, D.; Zobrist, G.: TCP/IP versus OSI. *IEEE Potentials*, ročník 9, č. 1, 1990: s. 16–19.

[6] Braden, R.: *Requirements for Internet Hosts – Communication Layers* [online]. Říjen 1989, [cit. 2017-05-08]. Dostupné z: `https://tools.ietf.org/html/rfc1122`

[7] Postel, J.: *Internet Protocol* [online]. Září 1981, [cit. 2017-04-24]. Dostupné z: `https://tools.ietf.org/html/rfc791`

[8] Raicu, I.; Zeadally, S.: Evaluating IPv4 to IPv6 transition mechanisms. In *Telecommunications*, ročník 2, IEEE, 2003, s. 1091–1098, [cit. 2017-04-24].

[9] Google: *IPv6 Statistics* [online]. 2017, [cit. 2017-04-24]. Dostupné z: `https://www.google.com/intl/en/ipv6/statistics.html`

[10] Postel, J.: *Transmission Control Protocol* [online]. Září 1981, [cit. 2017-04-24]. Dostupné z: https://tools.ietf.org/html/rfc793

[11] Spring, N.; Wetherall, D.; Ely, D.: *Robust Explicit Congestion Notification (ECN) Signaling with Nonces* [online]. Červen 2003, [cit. 2017-04-24]. Dostupné z: https://tools.ietf.org/html/rfc3540

[12] Ramakrishnan, K.; Floyd, S.; Black, D.: *The Addition of Explicit Congestion Notification (ECN) to IP* [online]. Září 2001, [cit. 2017-04-24]. Dostupné z: https://tools.ietf.org/html/rfc3168

[13] Postel, J.: *User Datagram Protocol* [online]. Srpen 1980, [cit. 2017-04-25]. Dostupné z: https://tools.ietf.org/html/rfc768

[14] IPv6.com: *UDP - User Datagram Protocol* [online]. 2008, [cit. 2017-04-25]. Dostupné z: http://ipv6.com/articles/general/User-Datagram-Protocol.htm

[15] Deering, S.; Hinden, R.: *Internet Protocol, Version 6 (IPv6) Specification* [online]. Prosinec 1998, [cit. 2017-04-25]. Dostupné z: https://tools.ietf.org/html/rfc2460

[16] Cheswick, W. R.; Bellovin, S. M.; Rubin, A. D.: *Firewalls and Internet Security: Repelling the Wily Hacker*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., druhé vydání, 2003, ISBN 020163466X.

[17] nmap.org: *Port Scanning Basics* [online]. [cit. 2017-04-26]. Dostupné z: https://nmap.org/book/man-port-scanning-basics.html

[18] Vinet, J.: *knockd* [online]. GitHub, 2015, [cit. 2017-04-09]. Dostupné z: https://github.com/jvinet/knock

[19] Ellingwood, J.: *How To Use Port Knocking to Hide your SSH Daemon from Attackers on Ubuntu* [online]. DigitalOcean, 2014, [cit. 2017-04-09]. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-use-port-knocking-to-hide-your-ssh-daemon-from-attackers-on-ubuntu

[20] Krčmář, P.: *Port knocking: zaklepejte na svůj server* [online]. Root.cz, 2009, [cit. 2017-04-09]. Dostupné z: https://www.root.cz/clanky/port-knocking-zaklepejte-na-svuj-server/

[21] Valencia, A.: *How to use Port Knocking on Ubuntu to hide the SSH port* [online]. HowtoForge, [cit. 2017-04-09]. Dostupné z: https://www.howtoforge.com/tutorial/how-to-use-port-knocking-to-hide-the-ssh-port-from-attackers-on-ubuntu/

[22] Metz, C.: *Forget Apple vs. the FBI: WhatsApp Just Switched on Encryption for a Billion People* [online]. Wired, 2016, [cit. 2017-04-10]. Dostupné z: https://www.wired.com/2016/04/forget-apple-vs-fbi-whatsapp-just-switched-encryption-billion-people/

[23] Greenberg, A.: *'Secret Conversations:' End-to-End Encryption Comes to Facebook Messenger* [online]. Wired, 2016, [cit. 2017-04-10]. Dostupné z: https://www.wired.com/2016/07/secret-conversations-end-end-encryption-facebook-messenger-arrived/

[24] Greenberg, A.: *With Allo and Duo, Google Finally Encrypts Conversations End-to-End* [online]. Wired, 2016, [cit. 2017-04-10]. Dostupné z: https://www.wired.com/2016/05/allo-duo-google-finally-encrypts-conversations-end-end

[25] Rosenfeld, M.: *sslstrip* [online]. GitHub, 2011, [cit. 2017-04-10]. Dostupné z: https://github.com/moxie0/sslstrip

[26] Rosenfeld, M.: *Internet Explorer SSL Vulnerability* [online]. Moxie.org, 2002, [cit. 2017-04-10]. Dostupné z: https://moxie.org/ie-ssl-chain.txt

[27] Rosenfeld, M.: *More Tricks For Defeating SSL* [online]. Youtube.com, 2011, [cit. 2017-04-10]. Dostupné z: https://www.youtube.com/watch?v=ibF36Yyeehw

[28] Rosenfeld, M.: *chapcrack* [online]. GitHub, 2012, [cit. 2017-04-10]. Dostupné z: https://github.com/moxie0/chapcrack

[29] Rosenfeld, M.: *knockknock* [online]. Moxie.org, 2009, [cit. 2017-04-10]. Dostupné z: https://moxie.org/software/knockknock/

[30] Sanfilippo, S.: *hping* [online]. GitHub, 2014, [cit. 2017-04-10]. Dostupné z: https://github.com/antirez/hping

[31] Leech, M.; Ganis, M.; Lee, Y.; aj.: *SOCKS Protocol Version 5* [online]. Březen 1996, [cit. 2017-04-19]. Dostupné z: https://tools.ietf.org/html/rfc1928

[32] *proxychains-ng* [online]. 2016, [cit. 2017-04-19]. Dostupné z: https://github.com/rofl0r/proxychains-ng

[33] Evdokimov, L.: *redsocks* [online]. 2016, [cit. 2017-04-19]. Dostupné z: https://github.com/darkk/redsocks

[34] Bellare, M.; Namprempre, C.: *Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm* [online]. 2007, [cit. 2017-04-11]. Dostupné z: http://cseweb.ucsd.edu/~mihir/papers/oem.pdf

[35] Daemen, J.; Rijmen, V.: *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography, Springer Berlin Heidelberg, 2013, ISBN 9783662047224.

[36] Krawczyk, H.; Bellare, M.; Canetti, R.: *HMAC: Keyed-Hashing for Message Authentication* [online]. Únor 1997. Dostupné z: https://tools.ietf.org/html/rfc2104

[37] *Vyhláška č. 316/2014 Sb., o bezpečnostních opatřeních, kybernetických bezpečnostních incidentech, reaktivních opatřeních a o stanovení náležitostí podání v oblasti kybernetické bezpečnosti (vyhláška o kybernetické bezpečnosti).* 2014, [cit. 2017-04-11]. Dostupné z: https://portal.gov.cz/app/zakony/download?idBiblio=83170&nr=316~2F2014~20Sb.&ft=pdf

[38] Rash, M.: *fwknop* [online]. 2017, [cit. 2017-04-12]. Dostupné z: https://github.com/mrash/fwknop

[39] Rash, M.: *A Comprehensive Guide to Strong Service Concealment with fwknop* [online]. 2016, [cit. 2017-04-12]. Dostupné z: https://www.cipherdyne.org/fwknop/docs/fwknop-tutorial.html

[40] Nikšić, H.: *GNU Wget* [online]. 2017, [cit. 2017-04-12]. Dostupné z: https://www.gnu.org/software/wget/

[41] Josefsson, S.: *The Base16, Base32, and Base64 Data Encodings* [online]. Říjen 2006. Dostupné z: https://tools.ietf.org/html/rfc4648

[42] Ward, J.: *The Doorman* [online]. 2005, [cit. 2017-04-15]. Dostupné z: http://doorman.sourceforge.net/

[43] *Py-Port Knocking project* [online]. 2016, [cit. 2017-04-12]. Dostupné z: http://billiejoex.altervista.org/Prj_Py_port_knock.htm

[44] Malavolta, I.: *winKnocks* [online]. 2010, [cit. 2017-04-12]. Dostupné z: http://winknocks.sourceforge.net/

[45] Prinz, R.: *It's me (IM)* [online]. 2006, [cit. 2017-04-17]. Dostupné z: https://www.min.at/prinz/o/software/port/

[46] Microsoft TechNet: *Netsh AdvFirewall Firewall Commands* [online]. 2009, [cit. 2017-04-19]. Dostupné z: https://technet.microsoft.com/cs-cz/library/dd734783(v=ws.10).aspx

[47] Riverbed Technology: *WinPcap* [online]. 2013, [cit. 2017-04-21]. Dostupné z: https://www.winpcap.org/

[48] Riverbed Technology: *WinPcap Documentation* [online]. 2007, [cit. 2017-04-21]. Dostupné z: https://www.winpcap.org/docs/docs_40_2/html/main.html

[49] Stearns, W.: *Libpcap, winpcap, libdnet, and libnet applications and resources* [online]. 2008, [cit. 2017-04-21]. Dostupné z: http://www.stearns.org/doc/pcap-apps.html

[50] Combs, G.; aj.: *Wireshark* [online]. 2017, [cit. 2017-04-21]. Dostupné z: https://www.wireshark.org/

[51] Jacobson, V.; aj.: *Tcpdump* [online]. 2017, [cit. 2017-04-21]. Dostupné z: http://www.tcpdump.org/

[52] Microsoft: *Windows Filtering Platform* [online]. 2017, [cit. 2017-04-21]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa366510.aspx

[53] reqrypt.org: *WinDivert* [online]. 2016, [cit. 2017-04-20]. Dostupné z: https://reqrypt.org/windivert.html

[54] Bellare, M.; Desai, A.; Pointcheval, D.; aj.: *Relations among notions of security for public-key encryption schemes* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, ISBN 978-3-540-68462-6. Dostupné z: http://dx.doi.org/10.1007/BFb0055718

[55] Bellare, M.; Rogaway, P.: Introduction to modern cryptography. *UCSD CSE*, ročník 207, 2005: str. 207.

[56] Python Software Foundation: *Embedding Python in Another Application* [online]. 2017, [cit. 2017-04-28]. Dostupné z: https://docs.python.org/2/extending/embedding.html

[57] Larsen, M.; Gont, F.: *Recommendations for Transport-Protocol Port Randomization* [online]. Leden 2011, [cit. 2017-05-05]. Dostupné z: https://tools.ietf.org/html/rfc6056

[58] PyInstaller Development Team: *PyInstaller* [online]. 2017, [cit. 2017-05-01]. Dostupné z: http://www.pyinstaller.org/

[59] Wysopal, C.: *Netcat for Windows* [online]. 2004, [cit. 2017-05-02]. Dostupné z: https://eternallybored.org/misc/netcat/

[60] Microsoft: *Telnet Client* [online]. 2017, [cit. 2017-05-02]. Dostupné z: https://technet.microsoft.com/en-us/library/cc771275(v=ws.10).aspx

[61] Oracle: *Oracle VM VirtualBox* [online]. 2017, [cit. 2017-05-02]. Dostupné z: https://www.virtualbox.org/

[62] Russell, J.: *Inno Setup* [online]. 2012, [cit. 2017-05-03]. Dostupné z: `http://www.jrsoftware.org/isinfo.php`

[63] Microsoft: *CryptGenRandom function* [online]. 2017, [cit. 2017-05-03]. Dostupné z: `https://msdn.microsoft.com/en-us/library/windows/desktop/aa379942(v=vs.85).aspx`

Příloha **A**

Seznam použitých zkratek

AES Advanced Encryption Standard

CBC Code Block Chaining

CIDR Classless Inter-Domain Routing

CLI Commnad-line Interface

CTR Counter

DHCP Dynamic Host Configuration Protocol

DLL Dynamic-Link Library

DNS Domain Name System

ECB Electronic Codebook

GPL General Public License

HMAC Keyed-hash Message Authentication Code

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IIS Internet Information Services

IND-CCA Indistinguishability under chosen ciphertext attack

IND-CPA Indistinguishability under chosen plaintext attack

IP Internet Protocol

IPv4 Internet Protocol verze 4

IPv6 Internet Protocol verze 6

LAN Local Area Network

MITM Man-in-the-middle

MS Microsoft

NAT Network Address Translation

OS Operační systém

PGP Pretty Good Privacy

PPTP Point-to-Point Tunneling Protocol

RDP Remote Desktop Protocol

RIP Routing Information Protocol

SNMP Simple Network Management Protocol

SOCKS Socket Secure

SPA Single packet authorization

SSH Secure Shell

TCP Transmission Control Protocol

UDP User Datagram Protocol

VoIP Voice over IP

VPS Virtuální privátní server

WAN Wide Area Network

WFP Windons Filtering Platform

WPA2 Wi-Fi Protected Access 2

XML eXtensible Markup Language

Příloha B

Návod k instalaci a používání

Pro zjednodušení celého instalačního procesu byly vytvořeny instalační balíčky pomocí programu Inno Setup [62].

Instalační balíčky byly vytvořeny zvlášť pro klientskou a serverovou část nástroje. Oba nástroje lze jednoduše nainstalovat pouhým poklepáním na instalační balíček.

Nástroj byl otestován na operačních systémech MS Windows 7, MS Windows 8.1 a MS Windows Server 2016. Zároveň by měl být kompatibilní s operačními systémy MS Windows 8, MS Windows 10 a ekvivalentních serverových verzí (MS Windows Server 2008 R2, MS Windows Server 2012 a MS Windows Server 2012 R2).

Nástroj je k dispozici také ve veřejném repozitáři `https://github.com/pklejch/knockknock-win`.

B.1 Pokyny k instalaci

Jelikož je instalace obou částí nástroje totožná, následující instalační pokyny se týkají obou částí nástroje.

Při spuštění instalátoru se spustí průvodce, ve kterém uživatel specifikuje kam chce nástroj nainstalovat a zdali chce vytvořit zástupce na ploše.

Po úspěšné instalaci má uživatel na výběr zdali chce nainstalovat nástroj jako službu a zdali chce tuto službu po instalaci spustit.

Obě části tohoto nástroje vyžadují nainstalovaný balíček „Visual C++ Redistributable for Visual Studio 2012 Update 4“ pro 32bitovou architekturu (x86). Ten lze stáhnout na URL `https://www.microsoft.com/en-us/download/details.aspx?id=30679`.

B.2 Konfigurace serverové části nástroje

V adresáři s nainstalovanou serverovou částí nástroje lze najít konfigurační adresář (conf), ten má následující strukturu:

```
conf ..... Konfigurační adresář
├── config ..... Hlavní konfigurační soubor
└── profiles ..... Adresář s uživatelskými profily
    ├── petr ..... Nakonfigurovaný profil uživatele
    └── maruska ..... Nakonfigurovaný profil uživatele
```

Hlavní konfigurační soubor obsahuje tři následující direktivy:

error_window Tato direktiva specifikuje kolikrát se serverová část pokusí inkrementovat svůj čítač v případě desynchronizace čítačů. Hodnotou je celé číslo větší než 0.

delay Tato direktiva specifikuje počet sekund mezi přijetím posledního autentizačního paketu a odebráním pravidla z Windows Firewall. Hodnotou je celé číslo větší než 0.

moxie Tato direktiva specifikuje, zdali se použije režim kompatibility s původní implementací. Hodnotou je 1 či 0.

Pokud výše zmíněné direktivy nebudou přítomné v konfiguračním souboru, serverová část použije tyto výchozí hodnoty: error_window s hodnotou 20, delay s hodnotou 15 a vypnutým režimem kompatibility.

Uživatelské profily je možné generovat pomocí přiloženého nástroje „knockknock_genprofile“. Pro vygenerování profilu musí uživatel spustit tento program z příkazové řádky se dvěma argumenty: název profilu a číslo portu. Ukázka vygenerování profilů vypadá následovně.

```
knockknock_genprofile jirka 10001
```

Výše zmíněný příkaz vygeneruje a uloží profil „jirka“ do adresáře conf/profiles/. Název profilu i číslo portu musí být v rámci serverové části unikátní. Číslo portu slouží k identifikaci různých uživatelů, každý uživatel má definovaný svůj port, na který bude zasílat autentizační pakety.

Obsahem profilu jsou čtyři soubory: klíč pro symetrickou šifru, klíč pro HMAC, čítač profilu a konfigurační soubor profilu. Oba klíče jsou vygenerovány pomocí Python funkce os.urandom, která volá funkci CryptGenRandom z Windows CryptoAPI, která je dle [63] kryptograficky bezpečná. Čítač je nastaven na výchozí hodnotu nula.

B.3 Konfigurace klientské části nástroje

Po vygenerování profilu v serverové části nástroje se musí tento profil bezpečně přenést na počítač klienta. Následně se tento profil musí umístit do konfiguračního adresáře (conf) klientské části nástroje, ten se nachází v adresáři s nainstalovanou klientskou částí nástroje.

Tento profil se musí pojmenovat jako doménové jméno či adresa serveru, kterou tento profil reprezentuje.

Každý profil obsahuje čtyři soubory: klíč pro symetrickou šifru, klíč pro HMAC, konfigurační soubor daného profilu a čítač profilu.

Konfigurační soubor profilu obsahuje pět následujících direktiv:

knock_port Číslo portu, které reprezentuje profil uživatele. Na tento port budou odesílány autentizační pakety. Port je definován při generování profilu.

delay Tato direktiva specifikuje počet sekund, které musí uběhnout od odeslání posledního autentizačního paketu pro odeslání nového. Tato direktiva musí mít podobnou hodnotu jako direktiva delay specifikovaná v konfiguračním souboru serverové části. Hodnotou je celé číslo větší než nula.

source_ip Zdrojová IPv4 adresa, která bude použita pro autentizaci klienta. Tato IP adresa musí být stejná jako je zdrojová IP adresa autentizačního paketu. Pokud je tedy klient v privátní síti se zapnutým překladem síťových adres (NAT), hodnotou této direktivy musí být IP adresa brány. Lze specifikovat hodnotu „auto“, ta způsobí, že se provede HTTPS požadavek na URL „https://httpbin.org/ip“ a jako IP adresa se použije hodnota z HTTPS odpovědi. Konfigurace na hodnotu „auto“ však nebude fungovat při připojení v rámci LAN.

auth_packet_delay Tato direktiva specifikuje čas v milisekundách, mezi odesláním autentizačního paketu a odesláním původního odchozího paketu. Hodnotou je celé číslo větší než nula.

moxie Tato direktiva specifikuje, zdali se použije režim kompatibility s původní implementací. Hodnotou je 1 či 0.

Struktura konfiguračního adresáře v klientské části nástroje je následující.

```
conf ..... Konfigurační adresář
├── klejch.cz ..... Adresář s profilem reprezentující server klejch.cz
│   ├── cipher.key ..... Soubor s klíčem pro symetrickou šifru
│   ├── mac.key ..... Soubor s klíčem pro HMAC
│   ├── config ..... Konfigurační soubor profilu.
│   └── counter ..... Čítač profilu
└── 192.168.56.2 ... Adresář s profilem reprezentující server 192.168.56.2
    ├── cipher.key
    ├── mac.key
    ├── config
    └── counter
```

B.4 Používání nástroje

Klientskou i serverovou část nástroje lze ovládat pomocí ovládacího rozhraní služeb systému Windows. Toto ovládací rozhraní lze spustit pomocí klávesové zkratky Win+R a spuštěním programu services.msc či pomocí nabídky Start a napsáním klíčového slova „Služby“ (anglicky „Services“).

V tomto ovládacím rozhraním je k dispozici aktuální seznam nainstalovaných služeb. Klientská část nástroje se jmenuje „Auto Knocker“, serverová část nástroje se jmenuje „Knockknock Daemon“. V tomto rozhraní lze obě dvě části nástroje spustit, zastavit či restartovat. Lze také nastavit automatické spuštění po startu počítače. Dále lze nastavit automatický restart služby v případě neočekávaného pádu.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `bin` ..... adresář s instalačními balíčky implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `client` ..... zdrojové kódy klientské části implementace
    - `server` ..... zdrojové kódy serverové části implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF