VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE ÚTOKŮ NA SLUŽBU SSH NA ÚROVNI NETFLOW

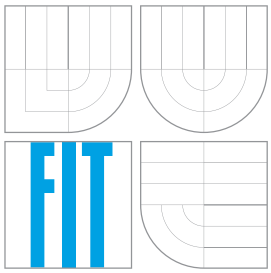
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARCEL MAREK

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE ÚTOKŮ NA SLUŽBU SSH NA ÚROVNI NETFLOW

SSH ATTACKS DETECTION ON NETFLOW LAYER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARCEL MAREK
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZBYNĚK MICHLOVSKÝ
SUPERVISOR

BRNO 2011

Abstrakt

Práce krátce popisuje základní principy protokolu SSH, jeho architekturu a používané šifrování. Dále se práce zabývá dolováním dat z nižších vrstev síťové komunikace a využitím těchto informací k detekci útoků. Také popisuje použité slovníkové útoky na službu SSH a s využitím NetFlow ukazuje možnosti dalšího zvýšení síťové bezpečnosti.

Abstract

This bachelor's thesis briefly describes the basic principles of SSH protocol, its architecture and used encryption. The thesis is mainly focused on datamining information from low-level network communication and usage of its results for attacks detection. It also describes dictionary attacks used on SSH service and with NetFlow shows further possibilities of increasing network security.

Klíčová slova

ssh, honeypot, detekce botnetů, černá listina, slovníkové útoky, netflow, dolování dat, klasifikace, random forests, weka

Keywords

ssh, honeypot, botnet detection, blacklist, dictionary attacks, netflow, data-mining, classification, random forests, weka, learning machine

Citace

Marcel Marek: Detekce útoků na službu ssh na úrovni netflow, bakalářská práce, Brno, FIT VUT v Brně, 2011

Detekce útoků na službu ssh na úrovni netflow

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zbyňka Michlovského.

.....
Marcel Marek
17. května 2011

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Zbyňku Michlovskému za jeho vstřícný přístup, ochotu a věcné připomínky.
Také bych rád poděkoval mé rodině, která mě při tvorbě této práce nesmírně podporovala.

Obsah

Kapitola 1

Úvod

V dnešní době se počítače a připojení k internetu stalo pro většinu lidí samozřejmou součásti jejich života. Spousty jich svěřují své důvěrné informace při registraci, nebo při využívání různých služeb, jako např. Jabber, Facebook, e-mail, atd., těmto organizacím a vystavují se potenciálnímu riziku zneužití dat při napadení některého serveru, na němž daná služba běží. Proto je důležité nejenom zabezpečení přihlášení na server silným heslem, případně osobním klíčem, ale i detekce potencionálního útočníka. Například, pomocí detekce botnetů, což je ovšem při dnes se více rozmáhajících distribuovaných útocích, kdy žádný z útočníků nepoužívá stejný slovník, ale pouze část ze společně sdíleného slovníku a dostatečným intervalem mezi jednotlivými pokusy, dost obtížné.

Jedním ze způsobů detekce chování útočníků je nasazení honeypotu. V takovém případě se útočník nejprve snaží prolomit zabezpečení této „návnady“, čímž získáme jednak čas, a jednak právě profil jeho chování. Zjištěný profil nám může dát představu o tom jaké slabiny systému se snaží využít. Díky této informaci můžeme provést úpravu zabezpečení na opravdu důležitých datech. Na druhou stranu je potřeba brát na vědomí, že v případě prolomení zabezpečení se útočníkovi otevírá díra do naší sítě, a proto je doporučeno mít takový server oddělen od zbytku sítě pravidly, jaká jsou aplikována i na případného útočníka.

Tato bakalářská práce se v druhé kapitole věnuje základům protokolu SSH, jeho architektuře a používanému šifrování. Dále teoretickými pojmy jako honeypot, NetFlow nebo slovníkový útok.

V další kapitole je popsána hardware a software konfigurace stroje, na kterém běží zaznamenávání útoků.

Ve čtvrté kapitole jsou zmapovány slovníkové útoky vedené proti *honeypotu*, nejčastější kombinace jmen, hesel, IP adres, časové rozložení těchto útoků. Porovnáme seznam zaznamenaných útočníků s dostupnými databázemi.

V páté kapitole jsou popsány způsoby detekce botnetů na základě použitého slovníku jednotlivými útočníky. Implementované metody rozebereme a zhodnotíme jejich výsledky. Nakonec se zamyslíme nad dalšími možnými způsoby detekce při využití všech dostupných dat.

V šesté kapitole rozebereme možnosti a omezení převodu dat síťové komunikace z nižších vrstev do formátu NetFlow pomocí hardware a software sondy. Popíšeme detekci útoků na úrovni NetFlow pomocí programu *Weka*. Představíme si základní principy tohoto programu a algoritmu použitého při klasifikaci. Uvedeme také základní charakteristiky ARFF, jež je formát dat používaný programem *Weka*. Zhodnotíme dosažené výsledky.

Poslední kapitola obsahuje závěrečné zhodnocení práce.

Kapitola 2

Teoretický úvod

2.1 SSH protokol

SSH - Secure Shell je síťový protokol určený pro vzdálené zabezpečené přihlášení a další síťové služby přes nezabezpečenou síť. SSH je náhradou za zastarávající protokol TELNET, který komunikaci nijak nešifruje, a je tak nevhodný na vzdálené přihlášení a ostatní podobné služby, při nichž se posílají důvěrná data jako jsou hesla. Převážně využíván na UNIXových systémech pro vzdálené přihlášení, vykonávání příkazů, případně tunelování, nebo přesměrování TCP/IP portů, respektive X11 spojení. Používá kryptografii veřejných klíčů pro ověření vzdáleného počítače. Využívá klasický model klient-server. Server typicky běží na TCP portu 22. SSH protokol tvoří tři části.

- **Přenosová vrstva SSH** je zabezpečený, nízkoúrovňový transportní protokol. Poskytuje autentizaci a ověření důvěryhodnosti serveru, zaručení integrity komunikace, volitelně i kompresi. Typicky běží na TCP/IP spojení, ale může pracovat nad jakémkoli spolehlivém spojení, jako AppleTalk, IPX/SPX a další. Autentizace probíhá pouze na straně serveru, neprovádí autentizaci uživatele. Může být použit jako základ pro řadu dalších zabezpečených síťových služeb. Pomocí tohoto protokolu se vyjednává metoda výměny klíčů, algoritmus veřejného klíče, symetrického šifrování, autentizační zprávy a algoritmus pro hashování. Samotná výměna klíčů začíná výměnou seznamu podporovaných algoritmů, více viz 2.1.4. Zpracováno na základě informací v [30].
- **Vrstva pro ověření uživatele** ověřuje klientskou stranu uživatele vůči serveru. Předpokládá, že běží na zabezpečeném protokolu transportní vrstvy, který zajistil autentizaci serveru, ustanovil šifrovaný komunikační kanál a vypočítal unikátní identifikační číslo sezení pro tuto relaci. Poskytuje jeden ověřený tunel pro vrstvu spojení. Čerpáno z [27].
- **Vrstva spojení** je navržena tak, aby pracovala nad protokolem pro ověření uživatele a transportní vrstvy. Rozděluje šifrované spojení do několika logických kanálů. Obě strany mohou otevřít kanál. Kanály jsou identifikovány číslem na obou koncích, přičemž čísla mohou být různá. Využity informace v [28].

2.1.1 Šifrování

SSH poskytuje širokou paletu algoritmů pro šifrování dat přenášených přes síť. Šifrování je založeno na náhodně generovaných jednorázových klíčích, které jsou bezpečně dohodnuty

pro každé sezení a po ukončení jsou zahozeny. Podporovaných algoritmů je celá řada např. 3DES, AES, Blowfish, DES, IDEA, ... Celý seznam aktuálně podporovaných algoritmů je uveden v [11, 4.11.1, p. 16].

Šifrovací algoritmus a klíč jsou dohodnuty během výměny klíčů. Pokud šifrování běží, délka paketu, doplněné zarovnání, náklad a doplněná pole každého paketu musí být zašifrována daným algoritmem. Všechna šifrování jsou v tzv. režimu CBC (Cipher Block Chaining), více viz podsekce 2.1.2 a měla by používat efektivní klíč dlouhý alespoň 128 bitů. V každém směru může být zvoleno šifrovaní nezávislé na opačném směru, pokud místní pravidla umožnují použití více algoritmů. Nicméně je doporučeno použití stejných algoritmů v obou směrech. Při tvorbě tohoto odstavce bylo čerpáno z [1, 3.1].

Zmiňme podrobněji alespoň dva algoritmy - `3des-cbc`, jež musí podporovat každá implementace, ačkoliv se nedoporučuje, a `aes128-cbc`, který je naopak volitelný, ale doporučený.

- `3des-cbc`

  Jak bylo zmíněno výše, tuto šifru musí podporovat všechny implementace, ačkoliv se nedoporučuje, protože délka efektivního klíče tohoto algoritmu je pouze 112 bitů [19] a SSH specifikace říká, že by měl být větší než 128 bitů [30, § 6.3]. 3DES je třikrát aplikovaná DES (Data Encryption Standard)[19, p. 280] šifra s trojicí klíčů (šifrování-dešifrování-šifrování), kde prvních 8 bajtů je určené k zašifrování, dalších 8 bajtů k dešifrování a posledních 8 bajtů opět k zašifrování, to vyžaduje 24 bajtů (192 bitů), ačkoliv využito je pouze 168 bitů, protože klíče mají délku 56 bitů.

  CBC režim u této šifry znamená, že „triple-DES“-zašifrovaný 64 bitový (8 B) datový blok je XORován [23] s nešifrovaným následujícím blokem (8 B), díky čemuž jsou všechny bloky závislé na předcházejících. Na rozšifrování určitého textu je zapotřebí znát šifrovaný text, klíč a šifrovaný text předcházejícího bloku. První blok nemá žádný předchozí blok, takže nezašifrovaný text je XORován s 64 bitovým číslem zvaným Inicializační Vektor (zkráceně IV). V případě přenosu přes síť, což je náš případ, se chyba přenosu, přidání/smazání, dat šíří do všech následujících bloků, protože každý blok je závislý až na posledním. Pokud nastane chyba, která data pouze pozmění, což je častější případ, ovlivní pouze změněný blok a odpovídající bity bloku následujícího. XOR celému procesu šifrování dodává další vrstvu.

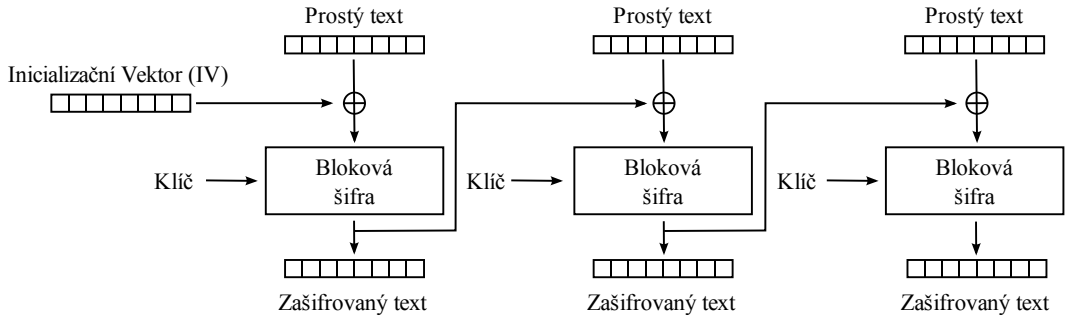
- `aes128-cbc`

  AES (**A**dvanced **E**ncryption **S**tandard) [13] je „pokročilý šifrovací standard“ přijatý Americkou vládou. Šifrovací algoritmus vychází z algoritmu Rijndael [4], používá 128 bitové (16 B) bloky a klíče délky 128, 192 a 256 bitů, v našem konkrétním případě 128 bitů. Operuje na poli 4×4 bajty. Většina AES výpočtů je dělána ve speciálním konečném rozsahu. AES je definován jako opakování transformačních kol, které mění vstupní prostý text na šifrovaný. Každé kolo obsahuje několik výpočetních kroků, včetně jednoho, který je závislý na šifrovacím klíči. Naopak sada opačných kol (opakování) se použije pro dešifrování použitím stejného klíče.

2.1.2 CBC režim

CBC (Cipher Block Chaining) režim byl vyvinut firmou IBM v roce 1976 [6]. V tomto režimu je každý blok nezašifrovaného textu XORován s předchozím blokem zašifrovaného textu před tím, než je šifrován. Díky tomu je každý zašifrovaný blok závislý na všech předchozích rozšifrovaných blocích. CBC režim vyžaduje inicializační vektor (IV), který je stejně

dlouhý jako délka bloku použité šifry. Použití náhodně generovaného IV zabraňuje generování identických bloků šifrovaného textu z paketů, které mají identická data v prvním šifrovaném bloku. IV je XORován s prvním blokem nešifrovaného textu před tím než je zašifrován. Pro následující bloky je předchozí šifrovaný blok XORován s aktuálním nezašifrovaným textem před tím než je zašifrován, viz obrázek 2.1.



Obrázek 2.1: Cipher Block Chaining (CBC) šifrovací režim

2.1.3 Metoda výměny klíčů

Metoda výměny klíčů určuje jakým způsobem jsou generovány jednorázové klíče pro šifrování a ověření serveru a jakým způsobem je ověření provedeno. Jsou definovány dvě metody, které jsou vyžadovány – `diffie-hellman-group1-sha1` a `diffie-hellman-group14-sha1`.

2.1.4 Výměna klíčů

Výměna klíčů začíná tím, že každá strana zašle seznam podporovaných algoritmů. Obě strany mají preferovaný algoritmus v každé kategorii a předpokládá se, že většina implementací, v danou dobu, používá stejné preferované algoritmy. V odpovídajících polích jsou zasílány seznamy podporovaných algoritmů. Jednotlivá jména jsou oddělená čárkou. Každá strana musí zaslat alespoň jeden algoritmus pro každou kategorii. Všechny podporované algoritmy musí být uvedeny. Algoritmy jsou seřazeny podle preferovanosti. První algoritmus v každém seznamu musí být preferovaný – tzv. hádaný.

Obě strany se můžou pokusit uhádnout jaký algoritmus používá druhá strana, a mohou poslat počáteční paket výměny klíčů hádaného algoritmu. Pokud hádaný algoritmus splňuje požadavky preferovaného algoritmu druhé strany, musí být tento hádaný počáteční paket zpracován jako počáteční paket, naopak pokud se algoritmus nepodaří uhádnout, musí se počáteční paket zahodit. Metoda výměny klíčů používá explicitní ověření serveru v případě, že zpráva výměny klíčů obsahuje podpis, nebo důkaz ověření serveru. V případě, že server musí dokázat, že zná „sdílené tajemství“ `K`, zasláním zprávy, nebo odpovídajícího ověřovacího kódu zprávy (MAC), používá implicitní ověření.

Výměna klíčů se dokončí zasláním zprávy `SSH_MSG_NEWKEYS` šifrované původním algoritmem a klíčem. Všechny následující zprávy již musí používat nový algoritmus a klíč. Pokud se nenajde algoritmus splňující požadavky obou stran, spojení selže a obě strany se musí odpojit.

Další informace o jednotlivých částech protokolu SSH můžeme najít v použitých RFC 4251 [29], 4252 [27], 4253 [30] a 4254 [28].

2.2 Data mining

Data mining - dolování dat je způsob získávání informací z nasbíraných dat. Jelikož množství nashromážděných dat se každé tři roky zdvojnásobí [14], stává se dolování dat důležitým nástrojem pro přeměnu těchto dat na informace. Tzn. získání informace z dat, která není na první pohled zřejmá. Při dolování dat se většinou nepracuje s celými daty, ale pouze se vzorky, proto je nutné správně zvolit reprezentativní vzorky.

2.3 Honeypot

Honeypot je návnada určená k detekci pokusu o neoprávněné použití serveru, počítače, či zneužití služby. Většinou se jedná o počítač, který je volně dostupný z internetu a tváří se, že nabízí nějakou z běžně dostupných služeb jako www, FTP, e-mail, nebo jako v našem případě SSH server. Nasazení takového stroje, zaměstná útočníka a naopak dává čas správci k provedení protiopatření, na základě zjištěného chování.

2.4 Slovníkový útok

Slovníkový útok je technika prolomení šifrování nebo autentizačního mechanismu k zjištění šifrovacího klíče nebo hesla, zkoušením pravděpodobných možností. Slovníkový útok využívá upravenou techniku útoku „hrubou silou“ [22], při které se nezkouší všechny možné kombinace, ale pouze kombinace z předem generovaného seznamu. Typicky všechny kombinace výrazů ze slovníku určitého jazyka, případně jazyků. Slovníkové útoky mají velkou šanci na úspěch, protože uživatelé pro zabezpečení často používají pouze jednoduchá hesla, založená na běžných slovech, která se vyskytují ve slovnících, případně na konec přidají číslici.

2.5 NetFlow

NetFlow je síťový protokol vytvořený firmou Cisco Systems pro sběr statistických informací o přenášených datech. Používá se k monitorování přístupu na internet, účtování přenášených dat, zjišťování odkud a kam míří přenášená data, monitorování DDoS útoků a dalších užitečných věcí především pro poskytovatele internetových připojení (ISP). Tvoří jej dva klíčové prvky - Exporter a Collector

- NetFlow exporter bývá nejčastěji směrovač s podporou sběru statistik ve formě NetFlow záznamů. Všechny příchozí pakety (při zapnutém vzorkování každý n-tý) na monitorovaný bod sítě jsou prozkoumány a z nich je vytvořen Flow (proud). Informace z těchto proudů jsou exportovány na NetFlow Collector ve formě NetFlow záznamů. Odesílání probíhá buď v případě ukončení daného proudu (zjištění FIN nebo RST bitu v TCP), nebo pokud byl daný proud po určitou dobu neaktivní, nebo při dlouho trvajících proudech.
- NetFlow Collector přijímá exportované NetFlow záznamy z jednoho, či více, exportérů a zpracované záznamy ukládá.

Proud je definován jako jednosměrná sekvence paketů, které sdílí 7 hodnot:

1. zdrojová IP adresa,
2. cílová IP adresa,
3. zdrojový port,
4. cílový port,
5. protokol transportní vrstvy,
6. typ služby,
7. vstupní rozhraní routeru nebo switche.

Kapitola 3

Systém pro získávání dat

V této kapitole si nadefinujeme pojem honeypot jak je chápán v této práci a vysvětlíme jakým způsobem je zajištěno, aby se náš *honeypot* tak opravdu choval. Popíšeme kde se nachází, jakým způsobem je připojen k internetu a také jeho hardware a software konfiguraci. Rozebereme rozdíl v implementaci upraveného SSHD (fake SSHD) pro zachycování útoků a SSHD pro administraci *honeypotu*.

3.1 Definice

Obecná definice honeypotu je uvedena v sekci 2.3. V našem případě se jedná o počítač, který se tváří, že nabízí možnost vzdáleného zabezpečeného přihlášení pomocí SSH. Opatření zajišťující nemožnost přihlášení se, je v souboru `/etc/hosts.allow` 3.1, který umožnuje přihlášení pouze počítačům z rozsahu VUT a to `147.229.0.0/16`. A druhá podmínka pro splnění označení honeypot, spočívá v získání informací z takového útoku. Toto zprostředkovává upravené SSHD, viz sekce 3.3, které zaznamenává čas útoku, útočníkovu IP adresu, použité přihlašovací jméno a heslo. Druhou užitnou hodnotou je zachycování síťové komunikace na portu 22, na kterém ono upravené SSHD běží, pomocí *tcpdump*.

3.2 Základní informace

Server je fyzicky umístěn v kanceláři A222 areálu VUT FIT na nefiltrované síti (tzn. mezi ním a CESNETem není žádný FW ani jiné zařízení, které by zvyšovalo bezpečnost). Aby se na *honeypot* nemohl připojit ani případně úspěšný útočník, je v souboru `/etc/hosts.allow`, viz výpis 3.1 pravidlo umožňující připojení pouze z rozsahu adres VUT. Má staticky přidělenou IP adresu `147.229.7.6`.

Kód 3.1: Výpis souboru `/etc/hosts.allow`

```
1 # /etc/hosts.allow
2 #
3 sshd: 147.229.
4 # End of file
```

Na portu 22 běží fake SSHD a korektní na portu 2323, které slouží k potřebám administrace serveru, jak můžeme vidět ve výpisu 3.2. Pro korektní připojení stačí jednoduše `ssh root@147.229.7.6 -p 2323`, stejně tak při použití `scp` přes parametr `-P 2323`. Příklad výpisu spojení při připojení přes korektní SSHD, viz výpis 3.3.

Konfigurace *honeypotu*	
procesor	AMD Athlon(tm) 64 Processor 3000+ (1800MHz)
paměť	1024MB
distribuce	Arch Linux
verze jádra	2.6.31
verze OpenSSH (fake SSHD)	5.0p1
port (fake SSHD)	22
verze OpenSSH	5.4p1
port	2323
IP adresa	147.229.7.6

Tabulka 3.1: Přehled základních parametrů

Kód 3.2: Výpis otevřených souborů

```
[root@buslab-5 ~]# lsof -i
COMMAND  PID USER   FD   TYPE  DEVICE SIZE/OFF NODE
NAME
sshd    1221 root    3r  IPv4 3275513      0t0  TCP
147.229.7.6:2323->244.248.broadband11.iol.cz:23210 (ESTABLISHED) ustavene
spojeni s~korektnim SSHD
sshd    1281 root    3u  IPv4    4894      0t0  TCP
*:2323 (LISTEN)->korektni SSHD
sshd    1895 root    3u  IPv4    7593      0t0  TCP
*:ssh (LISTEN) ->fake SSHD
sshd    1895 root    4u  IPv6    7598      0t0  TCP
*:ssh (LISTEN) ->fake SSHD
```

Kód 3.3: Výpis spojení

```
[root@buslab-5 ~]# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address    Foreign Address
State       PID/Program name
tcp        0      0 0.0.0.0:2323     0.0.0.0:*
LISTEN      1281/sshd
tcp        0      0 0.0.0.0:22       0.0.0.0:*
LISTEN      1895/sshd ->fake SSHD
tcp        0      0 147.229.7.6:2323 90.178.248.244:23210
ESTABLISHED 1221/0
tcp        0      0 :::22            :::*
LISTEN      1895/sshd ->fake SSHD
```

3.3 Fake SSHD

Fake SSHD je určeno k získání dat od útočníka. Je zkompilováno z openssh-5.0p1 s upraveným souborem `auth-passwd.c`, viz kód 3.4. Konkrétně funkce `int auth_password(Authctxt *authctxt, const char *password)`. Konfigurační soubor `/root/sshd/etc/sshd_config` je ponechán na výchozích hodnotách s výjimkou vynucení verze protokolu 2 a nastavením cesty k alternativní verzi sftp

`/root/sshd/libexec/sftp-server`. Od SSHD pro administraci běžící na portu 2323 se liší proměnnou prostředí `$PATH` rozšířenou o cestu `/root/sshd/bin`.

Kód 3.4: Upravená funkce auth_passwd

```
#include <time.h> //hlavickova knihovna pro praci s~casem
#include <sys/stat.h>
...
//funkce pro overeni hesla
int auth_password(Authctxt *authctxt, const char *password){
...
  if(!result){
    FILE *garp;
    //otevre log pro pridani
    garp = fopen("/var/log/sshd_logged", "a");
    chmod("/var/log/sshd_logged", 0600);//nastavi prava
    //zapise cas:uzivatele:heslo:ip_adresu
    fprintf(garp,"%i:%.100s:%.100s:%.200s\n",
    time(NULL),authctxt->user,password,get_remote_ipaddr());
    fclose(garp);//zavre log
  }
}
```

Všechny pokusy o spojení se díky tomu zaznamenávají do souboru `/var/log/sshd_logged` ve formátu `unix_cas:login:heslo:ip_adresa`. Kde `unix_cas` reprezentuje čas útoku v UNIXovém formátu (počet vteřin od 00:00:00 1.1.1970), `login` přihlašovací jméno, `heslo` použité heslo a `ip_adresa` IP adresu, ze které byl pokus prováděn. V adresáři `/root/src` je umístěn skript `ssh_dump`, viz kód 3.5, pro spuštění tcpdumpu `tcpdump -i eth0 -s 1500 -w /root/log/ssh-${DAT} port 22`, který ukládá jednotlivé soubory do `/root/log` se jmény ve formátu `ssh-RRRR-MM-DD:HHMMSS`. Pro případ pádu tcpdumpu je v crontabu umístěn záznam, který spouští `/root/src/ssh_dump` každé 2 minuty a zajišťuje tak téměř nepřerušený běh. V případě restartu serveru je nutné spustit ručně soubor `/root/sshd/sbin/sshd` pro běh fake SSHD.

Kód 3.5: Soubor ssh_dump

```
#!/bin/sh

SKRIPT=‘pidof -x tcpdump‘;
DAT=‘date +"%Y-%m-%d:%H%M%S"‘;

if [ -z "$SKRIPT" ]; then
        echo "$DAT Starting smtp_dump" >> /root/log/dumplog;
        tcpdump -i eth0 -s 1500 -w /root/log/ssh-${DAT} port 22;
        if [ "$?" -ne 0 ]; then
                echo "$DAT ERROR tcpdump start perror" >> /root/log/dumplog;
        fi
fi
```

3.4 Korektní SSHD

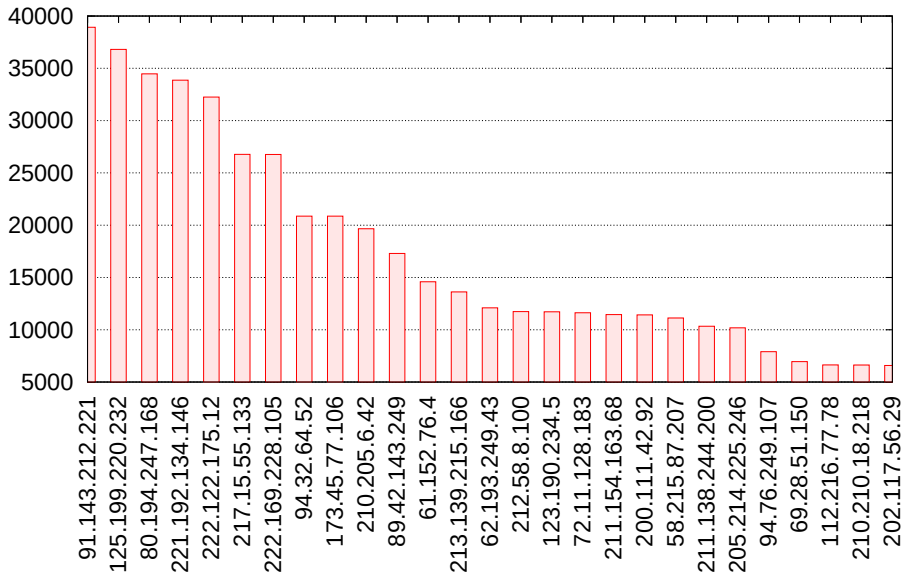
Korektní SSHD je standardní verze OpenSSH pro Linuxovou distribuci Arch Linux. Běží na TCP portu 2323 a slouží pro správu a administraci *honeypotu*. Kromě zmíněného portu má navíc oproti výchozímu nastavení povolenou autentizaci pomocí PAM.

Kapitola 4

Slovníkové útoky na *honeypot*

4.1 Souhrnné informace

Zpracovávaná data byla sesbírána za období od 21:41:40 26. července 2008 do 03:39:45 16. května 2010 středoevropského letního času. Za toto období bylo zaznamenáno celkem 758026 pokusů o přihlášení pocházejících z 730 jedinečných IP adres. Bylo použito 71307 jedinečných přihlašovacích jmen a 103729 jedinečných hesel během celého období. Počet pokusů o přihlášení v jednotlivých útocích se pohyboval v rozmezí od jednoho, či dvou, až po stovky, někdy až tisíce. Nejaktivnější IP adresy vidíme na obrázku 4.1.



Obrázek 4.1: Nejčastější IP adresy

4.2 Častá přihlašovací jména a hesla

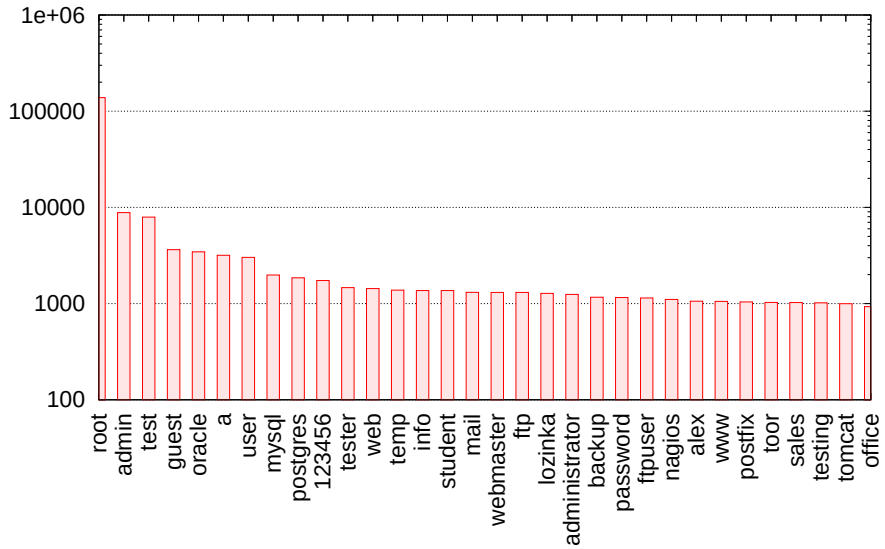
Jak se dalo očekávat, přihlašovací jméno, které bylo používáno nejčastěji, bylo `root`. Přihlašovací jméno `root` bylo použito v 138378 pokusech, což dává 18,25%, a v 139439 pokusech,

pokud počítáme i různé varianty jako `root123`, `rootadmin`, či `r00troot`, což dává 18,29% z celkového počtu všech pokusů o přihlášení.

Uživatelské jméno	**Použití**	
	počet	**%**
root	138378	18,25
admin	8835	1,16
test	7929	1,04
guest	3636	0,47
oracle	3452	0,45
a	3175	0,41
user	3027	0,39
mysql	1983	0,26
postgres	1853	0,24
123456	1735	0,22
tester	1467	0,19

Tabulka 4.1: Procentuální rozložení nejčastějších uživatelských jmen

Dalšími často cílenými uživatelskými jmény byly dočasné účty jako `test`, `guest`, `temp`, nebo `user`. Systémové účty byly také častým cílem. Celý přehled nejčastějších přihlašovacích jmen a jejich procentuální rozložení najdeme v tabulce 4.1 a grafickou reprezentaci na obrázku 4.2, kde je na ose y zvolena logaritmická stupnice z důvodu absolutně převyšujícího se výskytu jména `root`.



Obrázek 4.2: Nejčastější přihlašovací jména

Mimo uživatele `root`, dočasných a systémových jmen, velká většina přihlašovacích jmen použitých při útocích byla anglická křestní jména jako `john`, `amber`, `richard`. Naopak velice

malá snaha zaměřená na přihlašovací jména používaná ve velkých organizacích, různých informačních systémech, apod., které kombinují použití křestního jména a příjmení.

Hesla odvozená od přihlašovacího jména (např. `root/root123`, `guest/guestguest`) měla vůbec největší zastoupení v útocích. Celkem 359803 pokusů o přihlášení využívalo tuto techniku, což nám dává takřka polovinu všech pokusů, přesněji 47,5%. A 247364 pokusů (32,6%) použilo totožné přihlašovací jméno a heslo (např. `root/root`, `guest/guest`).

Nejčastější způsob změny hesla spočíval v přidání posloupnosti čísel např. „123“ na konec uživatelského jména. Pokusů s touto variací bylo celkem 83674 (11,03%). Mezi další způsoby úpravy hesla patřilo použití zkrácené verze uživatelského jména. Pro jméno `christine` použití hesla `chris`, případně naopak, pro kratší variantu jména použití jeho delší varianty jako heslo. Také byl použit způsob, při kterém se jednoduše zdvojí uživatelské jméno a vytvoří například `rootroot` z jména `root`.

V tabulce 4.2 jsou nejčastější hesla použitá při útocích spolu s počtem použití a jejich procentuálním rozložením v celkovém počtu pokusů o přihlášení. Varianty hesel popsaných výše jsou vyjádřeny jako *%username%*.

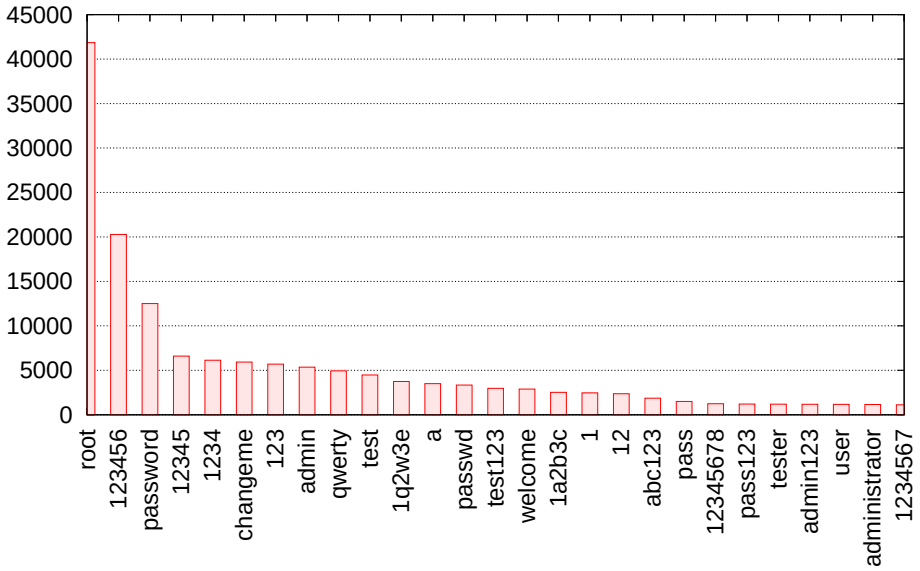
Heslo	**Použití**	
	počet	**%**
%username%	359803	47,5
root	41853	5,52
123456	20280	2,67
password	12511	1,65
12345	6602	0,87
1234	6138	0,80
changeme	5929	0,78
123	5697	0,75
admin	5352	0,71
qwerty	4941	0,65
test	4484	0,59

Tabulka 4.2: Procentuální rozložení nejčastějších hesel

Poměrně častým heslem je samotná číselná řada jako `123456`, `123` a další její varianty, kdy se mění počet číslic.

Zastoupeny jsou také hesla jako `password` (12511 – 1,65%), přeloženo jako „heslo“, a `changeme` (5929 – 0,78%), „změň mě“. Vyskytla se i tzv. *leet* [24] hesla, kdy se některé znaky nahradí číslicí připomínající tento znak např. místo `password` se použije `pa55w0rd`. Další skupinou jsou posloupnosti znaků podle jejich fyzického rozložení na klávesnici např. `qwerty` (4941 – 0,65%), `1q2w3e` (3740 – 0,49%).

Dosavadní prezentované výsledky se s určitou odchylkou shodují s těmi prezentovanými dříve v [15, 17].



Obrázek 4.3: Nejčastější hesla

4.3 Časové rozložení útoků

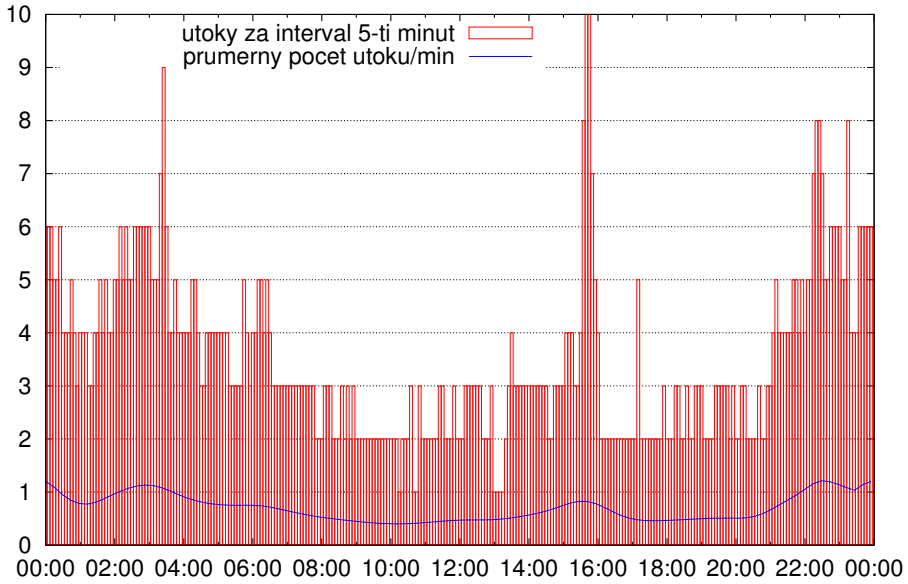
V této části je prezentováno rozložení útoků v čase.

Na obrázku 4.4 jsou data sesbíraná za celou dobu, promítnuta do časového rozsahu 24 hodin. Data jsou rozdělena do časového intervalu 5-ti minut a vydělena počtem dnů za celé období (658), z čehož nám vznikne daný histogram. Ten je proložen Bézierovou křivkou vyjadřující průměrný počet útoků za minutu. Jak můžeme vidět, průměrný počet útoků za celé období kolísá okolo 0,8 útoku za minutu. Což vychází na zhruba 1152 útoků za den.

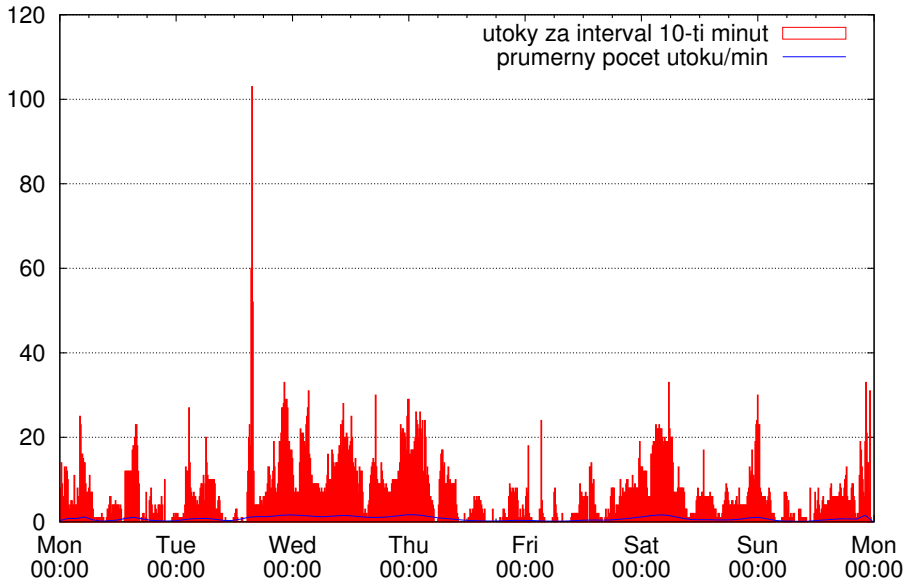
Obrázek 4.5 znázorňuje rozložení do jednoho týdne. Interval sběru dat je 10 minut a za celý týden tak máme 2016 hodnot, díky čemuž dosáhneme dostatečně nízkého kroku při generování grafu.

Obdobně je to pro obrázek 4.6, který reprezentuje rozložení za jeden kalendářní měsíc. V úseku od 30. do 31. dne v měsíci dochází ke zkreslení, protože ne všechny měsíce mají plných 31 dnů.

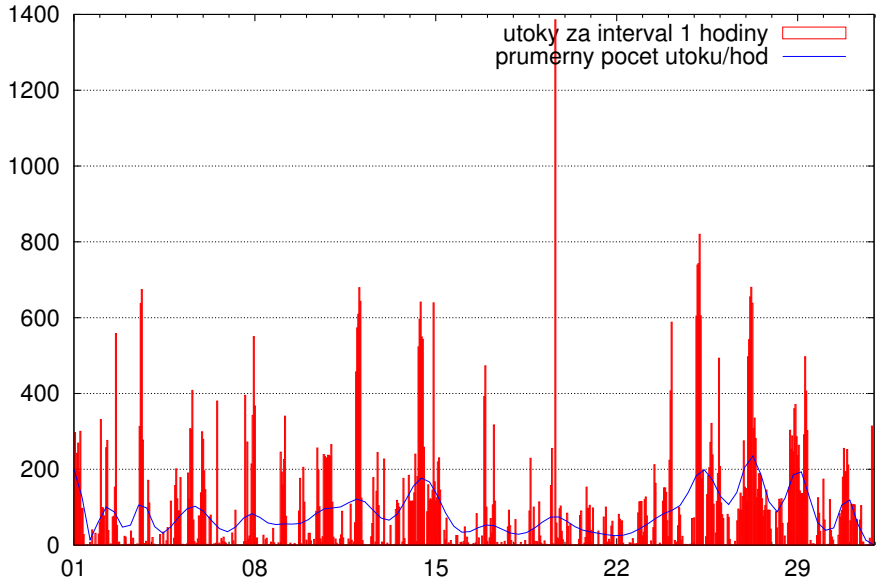
A konečně obrázek 4.7 zobrazuje rozložení za jeden celý rok.



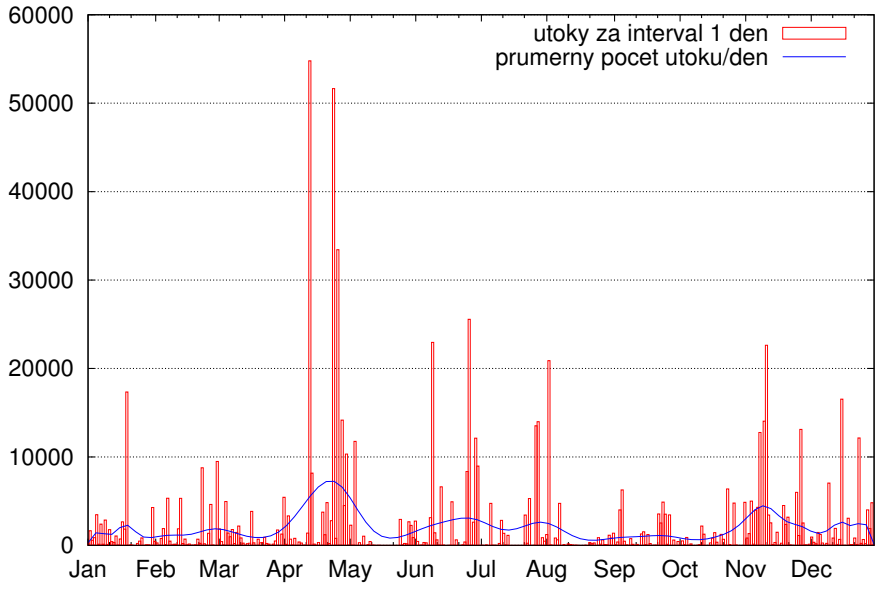
Obrázek 4.4: Histogram útoků promítnutý do 24 hodin



Obrázek 4.5: Histogram útoků promítnutý do týdne



Obrázek 4.6: Histogram útoků promítnutý do měsíce



Obrázek 4.7: Histogram útoků promítnutý do roku

4.4 Typy útoků

4.4.1 Jednoduchý slovníkový útok

Tento typ útoku používala většina útočníků. Jde o útok, kdy útočník jedná velice přímočaře a snaží se v co nejkratším čase vyčerpat všechny možnosti v použitém slovníku. Jednotlivé útoky se tak liší především délkou období, během kterého útok probíhal. To se odvíjí od velikosti použitého slovníku. Tento způsob útoku využíval například botnet identifikovaný jako `type-A-botnet-8` v kapitole 5.

4.4.2 Distribuovaný slovníkový útok

Takový útok je obtížné zachytit, protože útočník nejenom, že vždy použije pouze část kombinací, ale útok je rozložen mezi desítky až stovky počítačů. Tento typ útoku je typický pro botnet, který právě sdružuje nějakým způsobem ovládané velké množství počítačů. Detekce samotných botnetů je probírána v kapitole 5.

V našem případě se jednalo o celou podsíť `190.169.254.0/24`. Útok probíhal od 7. března 17:15:12 2009 do 7. března 17:58:03 2009 středoevropského času. Za tu dobu bylo provedeno celkem 1406 pokusů ze 116 IP adres v rozsahu `190.169.254.1` – `190.169.254.119`. Počet pokusů z jedné adresy se pohybuje v rozsahu 1–13, přičemž z tohoto rozsahu vystupuje IP adresa `190.169.254.119`, z které bylo vedeno celkem 945 pokusů (67,2%). Přehled procentuálního rozložení 10-ti nejaktivnějších IP adres je v tabulce 4.3. Byly použity výhradně kombinace s identickým přihlašovacím jménem a heslem. A pro uživatele `root` čísla od `1980` do `2010`. Pouze 7 kombinací se opakovalo dvakrát.

	Použití	
IP adresa	**počet**	**%**
190.169.254.119	945	67,21
190.169.254.65	13	0,92
190.169.254.78	12	0,85
190.169.254.9	11	0,78
190.169.254.85	9	0,64
190.169.254.121	9	0,64
190.169.254.11	9	0,64
190.169.254.71	8	0,57
190.169.254.7	8	0,57
190.169.254.4	8	0,57

Tabulka 4.3: Procentuální rozložení útoku mezi IP adresy

4.5 Srovnání s dostupnými databázemi útočníků

Pro srovnání IP adres použitých při útocích na službu SSH s blacklist databázemi jsme našli pouze dva dostupné zdroje. V obou případech se jedná o textový soubor obsahující jednotlivé IP adresy.

První dostupný z adresy `http://www.openbl.org/lists/base.txt` je aktualizován každých pár minut, obsahuje IP adresy hostitelů ze kterých byl prováděn *brute-force* útok na

některý z aktuálně 19-ti hostitelů (platné k 10. 4. 2011) používající SSH protokol. Tito hostitelé jsou umístěni v Německu, Spojených státech, Spojeném království, Francii, Ukrajině, Číně, Austrálii, České republice. Je nastaveno upozorňování a zaznamenávání těchto pokusů do centrální databáze. Text je převzatý z http://www.sshbl.org/. Námi testovaný soubor byl aktuální k 13:22:03 2. května 2011.

Druhý nalezený soubor je dostupný na adrese http://atlas-public.ec2.arbor.net/public/ssh_attackers. Doména `arbor.net` patří společnosti Arbor Networks, která je celosvětovým poskytovatelem prevence DDoS útoků a síťové bezpečnosti. Bohužel se nám na jejich webových stránkách http://www.arbornetworks.com/ nepodařilo najít nějaké bližší informace o tom, jak často je aktualizován, zda jsou IP adresy po určité době odstraněny apod. Testováno bylo několik souborů, které jsme postupem času z tohoto zdroje získávali. Zobrazené statistiky jsou ze souboru, který měl největší shodu, a ten byl stažen v 1:13:54 21. dubna 2010.

V tabulce 4.4 vidíme přehled shodných IP adres v jednotlivých zdrojích s naším referenčním seznamem 732 jedinečných IP adres. Velmi nízká shoda 0,9 % se souborem z `arbor.net` je pravděpodobně kvůli odstraňování starších IP adres, protože při srovnání dvou verzí souboru se zhruba půlročním odstupem, se neshodovala ani jediná IP adresa. IP adresy hostitelů s největším počtem útoků jsou uvedeny tabulkách 4.5 pro `sshbl.org` a 4.6 pro `arbor.net`.

	Počet		
Zdroj	**IP adres**	**shodných IP adres**	
sshbl.org	8471	145	19,8 %
arbor.net	356	7	0,9 %

Tabulka 4.4: Přehled počtu shodných IP adres v blacklist databázích

IP adresa	**Počet útoků**
222.122.175.12	32249
89.42.143.249	17291
61.152.76.4	14590
211.154.163.68	11451
194.106.203.100	5034
82.207.66.14	4745
61.158.205.224	4334
61.178.74.42	2299
89.171.143.147	2140
210.51.184.105	1776

Tabulka 4.5: Nejaktivnější IP adresy z `sshbl.org`

Stejný důvod ovlivňuje shodu se seznamem z `sshbl.org`. Tam se sice staré IP adresy nemažou, ale uživatel může požádat o odstranění ze seznamu např. pokud se stal sám terčem úspěšného útoku a z jeho počítače byl veden útok na další stroje. Dalším důvodem nízké shody může být cílení útoků pouze na určitý rozsah, případně zeměpisnou lokalitu. Ovšem při existenci hostitele oznamujícího útoky nacházejícího se v České republice je to

IP adresa	Počet útoků
202.108.59.112	3242
218.56.61.114	291
201.47.187.138	92
218.22.67.123	15
211.115.80.203	9
202.109.114.173	2
222.73.0.101	1

Tabulka 4.6: Nejaktivnější IP adresy z `arbor.net`

nepravděpodobné, ale nevíme, kteří konkrétní hostitelé z České republiky se na této činnosti podílejí, a jaký byl stav v době útoků.

Celkem 7 IP adres se vyskytovalo v obou testovaných seznamech a pouze dvě ve všech třech, tedy dvou databázích útočníků a našem referenčním seznamu, konkrétně `218.22.67.123` a `218.56.61.114`. Z těchto dvou IP adres bylo za celé období zaznamenáno celkem 306 pokusů o připojení.

Z porovnání prakticky pouze jednoho relevantního zdroje (`sshbl.org`) jsme moc informací nezískali. Pro lepší výsledky by srovnání muselo probíhat v době, kdy probíhají samotné útoky, případně určité periodě po útoku než se stihnou aktualizovat seznamy útočníků.

Kapitola 5

Detekce botnetů

V této kapitole se zaměříme na to, zda je možné na základě použitého slovníku (přihlašovací jméno/heslo) objevit síť počítačů použitou k útokům na službu SSH bez vědomí majitele (Botnet). Navrhneme několik variant, popíšeme jejich implementaci a zhodnotíme dosažené výsledky. V závěru navrhneme další možné způsoby detekce.

Nejprve si tedy nadefinujeme botnet.

5.1 Botnet

Jedná se o síť (**net**work) nakažených počítačů, označovaných jako **bot**, k nejrůznějším zákeřným činnostem, v našem případě útoku na službu SSH. Bývají centrálně řízené přes IRC protokol. Pro botnet je charakteristické, že všechny počítače při útoku využívají společný slovník, ze kterého čerpají kombinace přihlašovací jméno/heslo. V případě méně sofistikovaných botnetů používají všichni členové celý slovník.

5.2 Návrh implementace

Při návrhu implementace si musíme určit typy útoků a detekci cílit na konkrétní typ. Podle míry interakce je můžeme rozdělit na jednoduchý a distribuovaný.

- Při jednoduchém útoku pracuje útočník samostatně a používá celý slovník. Proto útočníky se stejnými slovníky označíme jako členy botnetu *Type-A* v případě shody pořadí použití jednotlivých kombinací. Pro *Type-B* už nebudeme vyžadovat shodu v pořadí, ale pouze shodný slovník.

- Při distribuovaném útoku se může slovník rozdělit mezi útočníky několika způsoby. Všichni používají kompletní seznam jmen, ale každý používá pouze jedno heslo, případně malou část hesel. Takový botnet označíme jako *Type-C*. Nebo naopak všichni používají kompletní seznam hesel, ale pouze část jmen. Takový botnet označíme jako *Type-D*. V případě rozdělení slovníku na části, které nejsou propojeny jmény, nebo hesly, ale jsou rozděleny náhodně mezi jednotlivé členy, buď způsobem připomínající jednoduchý útok, kdy útočník dostane předem seznam kombinací, který je ovšem pouze část z celkového slovníku a pracuje sám s touto malou částí, nebo jednotliví členové při útoku na daný cíl postupně vybírají kombinace ze společného slovníku a informují o výsledku ostatní útočníky. Tímto způsobem se žádné dvě kombinace v útoku

neopakují *Type-E*. Takový útok není možné detekovat podle použitého slovníku, protože jak jsme si uvedli, je to právě různý slovník použitý jednotlivými útočníky, který tento útok charakterizuje.

Ještě si uvedeme přístup, který není založen na porovnávání použitého slovníku jednotlivých útočníků, ale bere do úvahy časové rozložení. Pomocí tohoto přístupu označíme za botnet útočníky, kteří provádí útok během určitého intervalu, respektive čas mezi dvěma po sobě jdoucími pokusy nepřekročí interval 3600 s. Tento způsob dokáže detekovat útok rozdělený mezi spoustu útočníků, kteří používají pouze část slovníku zmíněných výše. Dále v textu jej budeme označovat jako *Attack*.

5.3 Implementace

Vlastní detekce probíhá pomocí skriptu `map_attacks.py` přes parametr `--findBotnet`, který zavolá funkci `findBotnet`, která má jeden parametr, přes který přijímá název souboru z něhož se budou načítat záznamy o útocích. Název tohoto souboru můžeme specifikovat z příkazové řádky přes parametr `--log-name` a spolu s parametry `--work-dir` a `rel-path` můžeme spustit zpracování v jiném adresáři.

Nejprve otevřeme a načteme vstupní soubor jako seznam řádků. Postupně každý řádek rozdělíme na `time`, `ip`, `login` a `password` podle regulárního výrazu.

```
r=re.match(r'(?P<time>\d{10}):(?P<login>[^:]+):'\
'(?P<pass>.+):(?P<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})',line)
```

V případě, že některý řádek neodpovídá danému regulárnímu výrazu, tak se přeskočí. Postupně procházíme celý seznam a naplníme si slovník[1] `completeDict`, který obsahuje všechny použité kombinace login/heslo pro danou IP adresu. Obsahuje pro každou IP adresu slovník loginů obsahujících seznam[2] hesel. Tento slovník se dále využívá pro identifikaci tzv. *Type-B* botnetu. Druhý slovník `ipLoginPass` obsahuje pro každou IP adresu seznam použitých kombinací ve formátu `<login>-<password>` a využívá se pro identifikaci tzv. *Type-A* botnetu. A nakonec slovníky `ipLoginDict` a `ipPassDict`, které slouží pro identifikaci tzv. *Type-C*, respektive *Type-D* botnetu.

Po načtení všech záznamů projdeme slovník `ipLoginPass` a odstraníme všechny duplicitní kombinace loginu a hesla tak, abychom zachovali původní pořadí. Znovu procházíme tento slovník a ve vnořeném cyklu postupně porovnáváme použité kombinace všech IP adres, přičemž ze slovníku vždy odstraníme tu IP adresu, se kterou kombinace srovnáváme. Pokud najdeme shodné slovníky vytvoříme patřičný záznam ve slovníku `botnetTypeA` a přidáme obě porovnávané IP adresy a jejich kombinace. Nalezenou IP adresu odstraníme z původního slovníku, protože již byla přiřazeno do botnetu. Pro každou další shodu se do `botnetTypeA` přidává již pouze nalezená IP adresa a samozřejmě ji také odstraníme z původního slovníku. Obdobně pro naplnění `botnetTypeB` procházíme `completeDict` a slučujeme do botnetů IP adresy se stejnými kombinacemi, přičemž nezáleží na pořadí jejich přidání. Toho je docíleno použitím slovníku namísto seznamu. V případě *Type-C* a *Type-D* probíhá porovnávání na základě shody seznamu použitých jmen v `ipLoginDict`, respektive hesel v `ipPassDict`.

[1] Slovníkem se v této sekci označuje objekt typu `dict` (asociativní pole), který mapuje položky podle klíče.

[2] Seznamem se v této sekci označuje objekt typu `list`, který označuje měnitelnou uspořádanou posloupnost položek.

Kód 5.1: Část funkce `findBotnet(file)` vytvářející botnet *Type-A*

```
i=0; check=0; botnetTypeA={}
#pro kazdou polozku ve slovniku
for ipItem in ipLoginPass.items():
  ip=ipItem[0]
  combiList=ipItem[1] #list kombinaci
  if ipLoginPass.has_key(ip):
    #aktualne srovnavanou IP adresu odstranime z puvodniho seznamu
    del ipLoginPass[ip]
  #nastavime kontrolni priznak
  check=0
  for ipItem2 in ipLoginPass.items():
    ip2=ipItem2[0]
    combiList2=ipItem2[1]
    #pokud se kombinace shoduji
    if combiList == combiList2 and ip != ip2:
      #pokud botnet i jiz existuje
      if botnetTypeA.has_key(i):
        #pridame porovnavanou IP adresu
        botnetTypeA[i][ip2]=list(combiList2)
      else:
        #jinak vytvorime zaznam a pridame obe adresy
        botnetTypeA[i]={ip:list(combiList)}
        botnetTypeA[i][ip2]=list(combiList2)
      #incrementujeme si priznak pro nalezeni shody
      check+=1
      #vypiseme informaci o shode
      print 'shoda␣botnet␣'+str(i)+'␣Type␣A~'+ip+ '␣' + ip2
      del ipLoginPass[ip2]
  #pokude je pocet shod vetsi nez nula
  if check>0:
    #inkrementujeme pocitadlo botnetu
    i+=1
```

Nakonec si informace o jednotlivých botnetech zapíšeme do souborů. Nejprve si ověříme, zda existují příslušné adresáře, pokud ne, tak je vytvoříme. Před samotným zápisem statistik do souboru si vytvoříme seznam ze slovníku s botnety a jednotlivé botnety si seřadíme podle počtu IP adres a použijeme jako index původního slovníku. Postupně procházíme všechny slovníky s botnety a vypisujeme jednotlivé IP adresy daného botnetu a na konec souboru i použité kombinace. Zároveň je vytvořen soubor s regulárním výrazem pro filtrování jednotlivých pokusů o přihlášení z původního souboru pomocí skriptu `split_sshd_logged.sh` na základě shody IP adres. Zápis do souboru pro `botnetTypeA` vidíme na výpisu 5.2.

Pro zpracování z hlediska útoků slouží parametr `--chopAttacks`, respektive funkce `chopAttacks(file)`. Tato funkce načítá obdobně jako `findBotnet(file)` jednotlivé záznamy, rozdělí je podle zmíněného regulárního výrazu a rozděluje na jednotlivé útoky, pokud čas mezi dvěma po sobě jdoucími pokusy o připojení překročí 3600 s.

Kód 5.2: Část funkce `findBotnet(file)` zapisující informace do souboru

```
botnetTypeAList=list(botnetTypeA.items())
botnetTypeAList.sort(key=lambda delka:len(delka[1].keys()),reverse=True)
j=0
for i in range(len(botnetTypeA.keys())):
  if len(botnetTypeA[botnetTypeAList[i][0]].keys())>1:
    if not os.path.exists('botnet/type-A/botnet-'+str(j)):
      os.mkdir('botnet/type-A/botnet-'+str(j)+'')
    fileOut=open('botnet/type-A/botnet-'\
    +str(j)+'/type-A-botnet-'+str(j)+'.txt','w')
    fileGrep=open('botnet/type-A/botnet-'\
    +str(j)+'/type-A-botnet-'+str(j)+'.grep.txt','w')
    fileStats=open('botnet/type-A/botnet-'\
    +str(j)+'/type-A-botnet-'+str(j)+'.stats.txt','w')
    print 'botnet␣A~'+ str(i)

    for ip in botnetTypeA[botnetTypeAList[i][0]].keys():
      fileOut.write(ip+'\n')
      fileGrep.write('\\('+ip+'\\)\\|')
      print ip
    fileStats.write(str(len(botnetTypeA[botnetTypeAList[i][0]].keys()))+\
                '\t' + str(len(botnetTypeA[botnetTypeAList[i][0]][ip]))+\
                '\t' + 'type-A-botnet-'+str(j))
    fileGrep.write('\\('+ip+'\\)')
    fileOut.write('\n\n\n')
    fileOut.write(str(botnetTypeA[botnetTypeAList[i][0]][ip]))

    fileOut.close()
    fileGrep.close()
    j+=1
```

5.4 Výsledek

Z celkového počtu 730 jedinečných IP adres, skript identifikoval 34 tzv. *Type-A*, 33 *Type-B*, 46 *Type-C* a 35 *Type-D* botnetů. Největší, co do počtu IP adres, při porovnávání kombinací byl identifikován botnet označen jako `type-A-botnet-0`, respektive `type-B-botnet-0` (jsou totožné) s 1196 pokusy o přihlášení 57 IP adresami a 15 kombinacemi. Při porovnání pouze jmen je to `type-C-botnet-0` s 96-ti útočníky `type-D-botnet-0` při porovnávání použitých hesel. Z hlediska počtu kombinací pak `type-A-botnet-27`, respektive `type-B-botnet-26` s 41732 pokusy o přihlášení dvěma útočníky a 17396 jedinečnými kombinacemi, který byl zároveň největší i podle počtu pokusů o přihlášení, které tvoří 5,5 % z celkového počtu všech pokusů o přihlášení. Celkem bylo identifikováno 189 IP adres jako člen některého *Type-A*, 201 *Type-B*, 333 *Type-C* a 213 pro botnet *Type-D*. Přehled největších botnetů seřazených podle počtu IP adres je uveden v tabulkách 5.1, 5.2, 5.3 a 5.4.

Bylo identifikováno celkem 724 útoků[3]. Největší útok z hlediska zainteresovaných IP adres `attack-84` byl příkladem tzv. „Céčkového“ botnetu, kdy IP adresy všech útočníků spadají do stejné sítě při použití masky 255.255.255.0. Tento způsob prokázal svoji platnost při identifikaci právě „Céčkového“ botnetu a distribuovaného útoku, kdy každý útočník provedl přesně 3 pokusy a pokračoval další. Přehled nejaktivnějších můžeme vidět v tabulce 5.5.

[3]Za útok byla považována sekvence pokusů o připojení, při které časový interval mezi jednotlivými pokusy nepřekročil 3600 s (1 hodina).

Pro každou kategorii si popíšeme nejzajímavější botnety, ať už z hlediska počtu útočníků, množství kombinací, nebo celkového počtu pokusů.

5.4.1 *Type-A*

U všech identifikovaných botnetů patřila opět přihlašovací jména jako `root`, `admin`, `test` mezi nejčastější. Zajímavostí pro botnet `type-A-botnet-3` je, že byly použity specifické kombinace cílené pro náš stroj, kdy se jako login a heslo střídaly výrazy jako `vutbr`, `buslab-5` a `fit`.

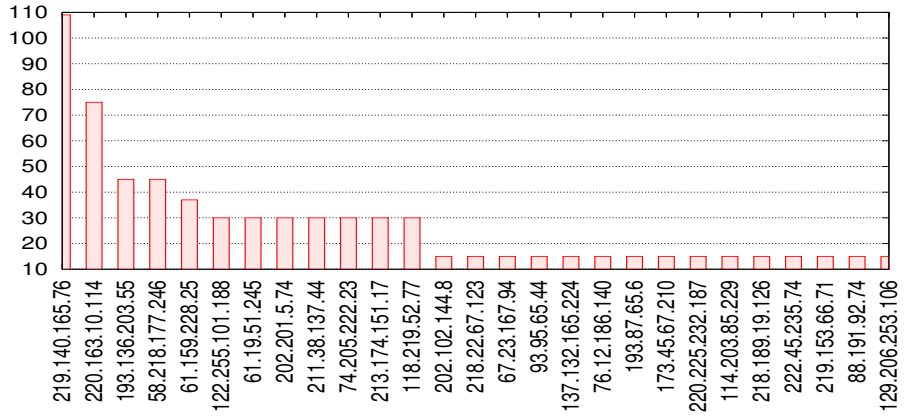
Počet			Označení
IP adresy	**Kombinace**	**Pokusy**	
57	15	1196	`type-A-botnet-0`
11	603	7537	`type-A-botnet-1`
11	67	996	`type-A-botnet-2`
9	32	288	`type-A-botnet-3`
9	165	1848	`type-A-botnet-4`
7	9	417	`type-A-botnet-5`
6	106	749	`type-A-botnet-6`
6	90	552	`type-A-botnet-7`
6	4509	34065	`type-A-botnet-8`
4	24	168	`type-A-botnet-9`
4	265	1096	`type-A-botnet-10`

Tabulka 5.1: 11 největších botnetů *Type-A* podle počtu IP adres

- `type-A-botnet-0` – jedná se o největší botnet z hlediska počtu útočníků. První zaznamenaný útok je z 03:42:38 13. srpna 2008 a poslední 02:39:45 16. května 2010. Tento botnet byl tedy aktivní po celou dobu fungování našeho *honeypotu*. Nejvíce pokusů zaznamenal útočník s IP adresou 219.140.165.76 se 109 pokusy, viz obrázek 5.1. Množství pokusů pro jednotlivé útočníky se liší podle počtu dílčích útoků vedených za celé období. Použitý slovník čítá pouze 15 kombinací a cílil především na uživatele `root`, ale také `test` a `oracle` s využitím variací *leet* hesel a číselných řad.
- `type-A-botnet-8` – má největší slovník z botnetů čítajících více než dva útočníky. Celkem 4509 jedinečných kombinací použilo 6 útočníků v 34065 pokusech o připojení. Z části byl opět cílený na účet `root`, ale obsahoval také nejrůznější variace zmíněné v kapitole 4 při použití stejného jména a hesla. Až na `211.138.244.200` s 10340 pokusy mají všichni útočníci shodný počet pokusů 4745.

5.4.2 *Type-B*

Vzhledem k tomu, že způsob detekce je velice podobný pro *Type-A* jsou i výsledky takřka totožné jako pro *Type-A*. Celkem 29 jich bylo totožných, jeden *Type-B* je pouze rozdělen na dva *Type-A*, což nám tvoří rozdíl mezi počtem identifikovaných botnetů oběma způsoby, a dva, které se částečně lišily.



Obrázek 5.1: Nejaktivnější IP adresy v `type-A-botnet-0`

Počet			Označení
IP adresy	**Kombinace**	**Pokusy**	
57	15	1196	`type-B-botnet-0`
15	35	525	`type-B-botnet-1`
11	603	7537	`type-B-botnet-2`
11	67	996	`type-B-botnet-3`
9	32	288	`type-B-botnet-4`
9	165	1848	`type-B-botnet-5`
7	24	264	`type-B-botnet-6`
7	9	417	`type-B-botnet-7`
6	106	749	`type-B-botnet-8`
6	90	552	`type-B-botnet-9`
6	4509	34065	`type-B-botnet-10`

Tabulka 5.2: 11 největších botnetů *Type-B* podle počtu IP adres

- `type-B-botnet-1` – byl nejvíce ovlivněn tím, že se nemusí shodovat pořadí, protože oproti `type-A-botnet-11`, který čítá 4 útočníky, tento identifikoval celkem 15 a počet pokusů o připojení se zvedl ze 140 na 525. Slovník je cílený pouze na účet `root` a jako hesla se vyskytují méně časté výrazy `letmein`, `fuck` a `sex`.

5.4.3 *Type-C*

- `type-C-botnet-0` – obsahuje útočníky, kteří jako přihlašovací jméno použili pouze `root`. Jedná se o 96 útočníků, kteří stáli za 14701 pokusy o přihlášení s celkem 12627 jedinečnými kombinacemi ve 103 útocích za období od 11:34:22 28. července 2008 do 12:57:28 12. května 2010.

- `type-C-botnet-1` – je oproti předchozímu pouze rozšířen o jména `test` a `oracle`. S 66-ti útočníky je v této kategorii druhý největší. Vystřídali celkem 148 kombinací

Počet			Označení
IP adresy	Kombinace	Pokusy	
96	1	14701	type-C-botnet-0
66	16	1796	type-C-botnet-1
14	67	1229	type-C-botnet-2
12	603	8171	type-C-botnet-3
11	173	2786	type-C-botnet-4
10	165	2026	type-C-botnet-5
9	32	288	type-C-botnet-6
7	24	264	type-C-botnet-7
7	9	417	type-C-botnet-8
6	106	749	type-C-botnet-9
6	90	552	type-C-botnet-10

Tabulka 5.3: 11 největších botnetů *Type-C* podle počtu IP adres

pouze se třemi uživatelskými jmény. Útoky jsou opět rozesety mezi celé období, konkrétně od 03:42:38 13. srpna 2008 do 02:39:45 16. května.

5.4.4 *Type-D*

Počet			Označení
IP adresy	Kombinace	Pokusy	
58	16	1240	type-D-botnet-0
15	35	525	type-D-botnet-1
13	32	576	type-D-botnet-2
12	603	8171	type-D-botnet-3
11	67	996	type-D-botnet-4
9	165	1848	type-D-botnet-5
8	24	476	type-D-botnet-6
7	9	417	type-D-botnet-7
6	106	749	type-D-botnet-8
6	90	552	type-D-botnet-9
6	4509	34065	type-D-botnet-10

Tabulka 5.4: 11 největších botnetů *Type-D* podle počtu IP adres

- `type-D-botnet-2` – stejně jako `type-A-botnet-3` použil hesla týkající se našeho *honeypotu*, ale dokázal rozšířit počet identifikovaných útočníků z 9 na 13. Útoky jsou opět roztaženy přes celé období od 08:56:38 23. září 2008 do 19:03:36 7. května 2010.

5.4.5 *Attack*

Při tomto způsobu detekce byly všechny pokusy rozděleny celkem do samostatných 724 útoků.

Počet			
IP adresy	**Kombinace**	**Pokusy**	**Označení**
116	1339	1406	`attack-176`
5	3115	3118	`attack-88`
5	15	15	`attack-79`
4	15616	21093	`attack-281`
4	12	12	`attack-84`
4	12	12	`attack-83`
3	9	9	`attack-81`
3	7353	8039	`attack-400`
3	15	45	`attack-204`
3	34449	34519	`attack-407`
3	188	209	`attack-349`

Tabulka 5.5: 11 nejrozsáhlejších útoků podle počtu útočníků

- `attack-176` – je příklad precizně řízeného botnetu, kdy při zkoušení např. sekvence v podobě jednoznakových jmen a hesel bylo dodrženo abecední pořadí při vystřídání 10 IP adres. Použitý slovník obsahoval od systémových jmen `postgres`, `ssl`, `ssh` přes vlastní jména, abecedu zkratky vysokých škol jako `mit`, `ucla`, `harward`, státy a jejich národní domény až po numerické sekvence. Jedná se o botnet popsaný v 4.4.2.

- `attack-88` – tento útok zahrnuje celkem 5 útočníků, 3115 jedninečných kombinací a celkem 3118 pokusů, probíhal 27. listopadu 2008 od 07:03:05 do 09:15:24. Také nám ukazuje slabinu této metody, protože se ve skutečnosti jedná o samostatný útok `150.161.6.43`, během kterého proběhly další 4 útoky. To, že se jedná o samostatný útok, je patrné z použitého slovníku a rozdělením jednotlivých pokusů mezi ostatní útočníky. Útočník z `150.161.6.43` použil celkem 3103 pokusů, přičemž jednotlivé pokusy byly v abecedním pořadí a používal stejné jméno a heslo. Naproti tomu další 3 útočníci cílili svůj útok výhradně na účet `root`. Zbylý útočník sice použil obdobné kombinace (stejné jméno a heslo), ale v náhodném pořadí a je spíše členem botnetu identifikovaného v `attack-79` a dalších.

- `attack-79` – sestává z 5-ti útočníků, kde každý použil právě 3 kombinace a poté pokračoval další. Mezi jednotlivými útoky je různá prodleva od 22 s až po 840 s. Podobný vzorec mají také útoky `attack-8[1-4]` a další. Napříč všemi pokusy o připojení bylo nalezeno několik obdobných útoků, ale domníváme se, že se jedná o dva botnety, protože jedna skupina cílí svoje útoky pouze na uživatele `root`, kdežto druhá používá kombinace se stejným jménem a heslem. Interval mezi dílčími útoky se dost liší a v mnohých případech přesahuje námi zvolený limit 3600 s, proto jsou jednotlivé části roztroušeny ve více segmentech.

- `attack-281` – je obdobou `attack-88`. Hlavní útok probíhal od 12:43:01 30. června do 05:00:22 1. července 2009 s celkem 19657 pokusy o přihlášení. Během hlavního útoku probíhaly celkem 3 další, vzájemně se nepřekrývající útoky.

5.5 Vyhodnocení

Všechny implementované metody prokázaly svoji funkčnost. Úspěšnost jednotlivých metod závisí na tom, jak moc striktní pravidla jsou nastavena. V případě výsledků *Type-A* si můžeme být jistí, že identifikovaní útočníci opravdu patří do daného botnetu. Ovšem při zrušení nutnosti dodržet pořadí jednotlivých sekvencí pomocí *Type-B* jsme identifikovali stejné botnety a některé rozšířili, takže by se dalo říct, že tato metoda byla úspěšnější.

V případě výsledků *Type-C*, kdy jsme všechny útočníky, kteří použili např. pouze jméno `root`, přiřadili do jednoho botnetu, je výsledek sporný. Jednotlivé útoky totiž vykazují odlišný přístup při výběru kombinací. Zatímco některé útoky používají spíše *leet* hesla, jiné vyčerpávají dlouhé seznamy vlastních jmen s číselnými řadami. Nejde prokázat, zda jsou to tedy oddělené botnety, nebo jeden používající nejrůznější techniky, které byly zaznamenány i během útoku vedeného jediným útočníkem.

Totéž platí i pro *Type-D*, její přínos je patrný například u identifikace `type-D-botnet-2`, který rozšířil počet útočníků v již identifikovaném botnetu pomocí *Type-A*.

Neduhem implementace všech těchto metod je, že při eliminování opakovaných útoků se stejným slovníkem, se mohlo stát, že útočník po určité době svůj slovník změnil. V tom případě, se nám použitý slovník u tohoto útočníka vytvoří spojením jedinečných kombinací z obou slovníků. Toto chování může být v určitých situacích žádané a v některých bychom naopak chtěli jedinečný slovník pro každý jednotlivý útok.

Přínos přístupu analyzovat jednotlivé útoky po skupinách, spadajících do určitého intervalu, ve formě útoků, byl prokázán při identifikaci tzv. „Céčkového“ útoku, do kterého se zapojili útočníci z jedné sítě, při uvažování síťové masky C a také částečným identifikováním pomalého útoku, kdy jeden útočník nepoužil více než tři pokusy v řadě a po náhodném intervalu pokračoval další. Ovšem v případě zmíněného pomalého útoku nebyla detekce ani zdaleka dokonalá, protože mezi jednotlivými sekvencemi byly i prodlevy větší než 8179 s, čímž byl překročen námi zvolený limit 3600 s a byly tak rozděleny do několika útoků a nebo zahrnuty do některého většího.

Z výsledků námi navržené detekce botnetů vyplývá, že k objevení botnetu stačí i znalost pouze IP adres útočníků a použitých kombinací, ale pouze v případě kompletní shody použitého slovníku (*Type-B*), případně přesné shody pořadí použití jednotlivých kombinací (*Type-A*). V případě detekování i podle částečné shody, bychom museli zvážit způsob porovnávání. Jelikož je velký rozdíl v množství pokusů z jednotlivých IP adres a také to, že se mnoho „obecných“ kombinací vyskytuje ve slovníku skoro každého útočníka, tak pokud bychom porovnávali částečnou shodu měli bychom velké množství *false-positive* identifikací.

Díky této metodě byly identifikovány botnety, jako výše zmíněný „Céčkový“ a částečně také distribuovaný útok, kde jednotliví útočníci použili vždy právě 3 kombinace a pokračoval další útočník. Tyto botnety nebyly jinou metodou nalezeny.

5.6 Další způsoby detekce

Další možné způsoby detekce při použití všech dostupných dat z *honeypotu*.

- Podle střední hodnoty mezi dvěma pokusy o přihlášení. Taková detekce by byla cílena na útoky, které se snaží vyhnout detekci pomocí programů jako *fail2ban*, tedy překročení určitého počtu neúspěšných přihlášení za daný čas. Při uvažování, že členové jednoho botnetu využívají stejný počet pokusů o přihlášení v jednom útoku a stejný interval mezi jednotlivými útoky.

- Při využití dat z *tcpdump* porovnávat řetězec identifikující verzi klientské aplikace, případně seznam podporovaných a preferovaných šifrovacích algoritmů. Za předpokladu, že počítač napadený pomocí infikované stránky, nebo přílohou elektronické pošty, si nejprve stáhne program pro provádění útoku. V tom případě, by všichni útočníci používali stejnou aplikaci a s tím související i identifikační řetězec a šifrovací algoritmy.

Kapitola 6

Detekce útoků na úrovni NetFlow

V této kapitole si popíšeme způsob získávání dat, jejich formát, možnosti a omezení převodu do NetFlow formátu pomocí hardware a software sondy. Popíšeme detekci útoků na úrovni NetFlow pomocí programu *Weka* 6.3.1. Představíme si základní principy používání tohoto programu a algoritmu použitého při samotné klasifikaci NetFlow dat. Popíšeme zpracování NetFlow dat do formátu ARFF 6.3.2, který používá právě program *Weka*. Na závěr zhodnotíme dosažené výsledky.

6.1 Sběr dat

Sběr dat probíhal ve dvou etapách. V první etapě byla zachycována data z útoků pomocí počítače umístěného v areálu fakulty – *honeypot*, viz kapitola 3, a data z korektních spojení na domácím serveru (dále jen jako *lenny*).

Nejprve si specifikujeme jaká data máme k dispozici a jaká nás skutečně zajímají. NetFlow záznamy jsou v podstatě statistikou komunikace mezi dvěma počítači, respektive obecně dvěma komunikujícími entitami, kdy nás zajímají pouze informace charakterizující dané spojení, nikoliv samotná data, která se přenášejí. Zajímají nás tedy v podstatě pouze statistické informace o každém spojení, avšak data, která máme k dispozici jsou tzv. *raw* data síťového provozu uložena do souboru ve formátu *pcap*, proto je nutné získaná data ještě zpracovat.

6.1.1 Data z útoků

Získávání dat z útoků probíhalo na stroji *honeypot*, viz kapitola 3, pomocí zmíněného fake SSHD, viz sekce 3.3. Sběr dat probíhal v období od 15:25:39 17. července 2008 do 21:00:47 28. září 2010 středoevropského letního času (SELČ). Data jsou v podobě logu z programu *tcpdump* ve formátu *pcap* [21], který zaznamenával veškerou síťovou komunikaci na portu 22 a ukládal do souboru. Za celé období máme k dispozici dohromady 33 souborů o celkové velikosti 3,3 GB.

6.1.2 Data z korektních spojení

Sběr dat korektních spojení byl prováděn na domácím serveru *lenny* pomocí stejného skriptu jako na *honeypotu* viz kód 3.5. Původně mělo sbírání těchto dat probíhat na některém ze školních studentských serverů poskytujících SSH, kde je garantovaný dostatečně velký provoz, protože jsou využívané pro testování projektů, ale bohužel jsme se nedohodli se

správci kvůli bezpečnosti dat, která je minimálně diskutabilní. Proto bylo nakonec získávání dat spuštěno na serveru *lenny*. Server nemá veřejnou IP adresu, ale leží za routerem se zapnutým NATem, na kterém je nastaveno přesměrování TCP portu 22. Tím je zajištěna přístupnost serveru z internetu. Základní konfiguraci vidíme v tabulce 6.1.

Konfigurace serveru *lenny*	
procesor	AMD Athlon(tm) processor (1300 MHz)
paměť	512 MB
distribuce	Debian 5
verze jádra	2.6.26
verze OpenSSH	5.1p1
port	různý
IP adresa	192.168.1.252 (90.178.248.244)

Tabulka 6.1: Přehled základních parametrů serveru *lenny*

Protože na stroji *lenny* byly pouze dva účty, jeden správcovský účet uživatele `root` a jeden běžný uživatel, bylo potřeba vytvořit sadu účtů, na které by se mohli připojit spolužáci a známí kvůli dosažení alespoň částečně reprezentativního vzorku dat. Protože se na stroji *lenny* nacházejí i soukromá data, byl pro tyto účty vytvořen falešný kořenový oddíl pomocí sady utilit *Jailkit* [18]. Celkem bylo vytvořeno 6 těchto dočasných účtů a rozdáno mezi známé s instrukcemi pro připojení. Uživatelé byli také požádáni o připojení pomocí privátního a veřejného klíče. Každému takovému přihlášení předchází alespoň jedno, při kterém uživatel nejprve nahraje svůj veřejný klíč na server.

Data korektních spojení jsou za období od 23:49:36 22. března do 23:57:16 4. května 2010 SELČ. Bohužel, během této doby došlo několikrát k restartování systému z důvodu selhání disků, proto data nejsou za úplně celé období. Naštěstí, tato porucha se netýkala disku, na kterém byla uložena data z programu *tcpdump*. Další výpadek byl způsoben nerestartováním *tcpdumpu* při změně portu (z důvodu rozmanitosti dat), na kterém SSHD přijímá spojení.

Jelikož je *lenny* viditelný z internetu je také častým cílem útoků na službu SSH. Na odvrácení jednoduchých útoků slouží program `fail2ban` [10], který po třech neúspěšných přihlášeních přidá útočníkovu IP adresu do pravidel firewallu, a tím mu zabrání v dalších pokusech. Ale i tak, jsou data znehodnocena těmito pokusy, a proto je potřeba myslet na odfiltrování těchto adres při další analýze. A aby to nebylo vše, musíme ještě ze seznamu útočníků odfiltrovat adresy úspěšných přihlášení, protože se mohlo stát, že některý uživatel zadal na první pokus špatné heslo.

6.2 Převod dat do formátu NetFlow

Při přehrávání dat pomocí *tcpreplay* přes hardware sondu FlowMon Probe 2000, která se nachází v laboratoři C304 areálu FIT VUT, jsme narazili na několik problémů. Pokud bychom chtěli zachovat časové značky, museli bychom data přehrávat původní rychlostí. Tedy přehrávání by trvalo stejnou dobu jako zachycování, což v našem případě bylo nemyslitelné, protože by to zabralo více než dva roky.

Druhou možností bylo přehrávat data plnou rychlostí CPU, ovšem při tomto způsobu se nám ztratí časové údaje a také během každého přehrávání bylo dosaženo jiných hodnot, nejspíše z důvodu zahlcení sondy daty.

Proto byla zvolena software sonda. Po vyzkoušení několika variant (*fprobe* [20], *ntop* [5], *flow-tools* [7]), žádná neposkytovala takové výsledky jaké bychom potřebovali. Pro svou jednoduchost a přehlednost zdrojového kódu byla jako vhodný kandidát vybrána open-source aplikace *softflowd*, což nám dává případnou možnost rozšířit její funkčnost.

Ovšem, ani *softflowd* při zpracování dat ze souboru, nepracuje správně s časovými údaji. Proto bylo nutné nejprve zjistit, proč tomu tak je. Podle RFC 3954 [3], je čas vyjádřen v hlavičce každého exportovaného setu NetFlow záznamů pomocí 8 B `sys_uptime` vyjadřující počet ms od zapnutí exportujícího přístroje (sondy) a 8 B `unix_secs` vyjadřující aktuální čas počtem sekund od 0:00:00 1. 1. 1970 a 8 B `unix_nsecs` vyjadřující zbylé nanosekundy tzv. *epoch time*. Pomocí těchto hodnot spočítáme přesné datum zapnutí přístroje. Začátek a konec jednotlivých spojení je vyjádřen počtem ms od zapnutí přístroje také na 8 B. Tento způsob umožňuje dosáhnout dostatečné přesnosti při rozumné paměťové náročnosti.

Jelikož sonda při zpracování vypisovala správná data, ale kolektor – *nfcapd* z balíku utilit *nfdump*, vypisoval už data špatná, odchytili jsme několik paketů pomocí programu *Wireshark*, abychom zjistili, zda je problém na straně sondy, nebo k němu dochází až při zpracování na kolektoru. Ukázalo se, že údaje se špatným časem odesílá již sonda. Vzhledem k tomu, že program *softflowd* udává jako `unix_secs` čas, kdy byl spuštěn, je nemožné exportovat hodnoty z minulosti. Na toto chování *sofflowd* upozornil během vytváření záplaty Daan van der Sanden pomocí bug reportu [16]. Proto jsme jako první test nastavili aktuální čas (`unix_secs`) při spuštění na nejstarší datum z našich dat, aby všechna následující zpracovávaná data byla již v budoucnosti. Opět pomocí programu *Wireshark* jsme odchytili několik paketů a vše se zdálo v pořádku. Ale problém nastal při zpracování dat za delší období, to se potom časové údaje opakovaly dokola. Problém je v rozsahu proměnné udávající začátek a konec spojení. Ta dokáže vyjádřit maximálně dobu $2^{32} - 1 = 4294967295$ ms $\cong$ 49 dnů. Proto při zpracování dat ze souboru přesahujícího takový rozsah, je potřeba při exportování přepočítávat čas zapnutí sondy tak, aby rozdíl času konce posledního exportovaného záznamu v daném setu a času zapnutí sondy nepřekročil daný limit. Pořád ale docházelo k jiným výsledkům pro každé zpracování. Protože zpracování dat a jejich exportování sondou i příjem kolektorem probíhal na jednom stroji a exportování dat se provádí přes nespolehlivý UDP dochází ke ztrátě dat z důvodu vytížení počítače. Proto po exportování každého setu dat je přidáno uspání sondy pomocí volání `usleep(4000)`. Hodnota `4000` byla zvolena po sérii testů a vztahuje se na konkrétní stroj, na kterém bylo zpracování dat prováděno. V případě použití externího kolektoru toto čekání nemusí být vůbec potřeba, nebo naopak by se muselo ještě zvýšit. Dalším zkoumáním jsme zjistili, že některé spojení mají větší `flow_start`, tedy čas přijetí prvního paketu, než `flow_last` posledního, viz výpis 6.1.

Kód 6.1: Část výstupu *softflowd* při zpracování dat z honeypotu

```
user@computer:~/$ softflowd -d -v5 -n localhost:23456
 -r log/ssh-honeypot-merged_all -R
softflowd v0.9.8 starting data collection
Exporting flows to [127.0.0.1]:23456
Start time of the flow is greater than the end time
2008-12-24T00:50:00.881797 -> 2008-12-24T00:49:57.006118
...
Start time of the flow is greater than the end time
2010-05-28T23:50:00.612355 -> 2010-05-28T23:49:58.321618
Start time of the flow is greater than the end time
2010-07-14T23:50:00.290124 -> 2010-07-14T23:49:58.868765
...
```

Během programování nastaly problémy s rozsahy proměnných při vyjadřování času, takže jsme chtěli zjistit, zda je chyba v *softflowd* nebo ve skutečných datech. Pomocí zobrazení programem *Wireshark* jsme zjistili, že pakety mají opravdu špatné údaje. Vzhledem k tomu, že všechna taková spojení byla v určitou dobu, a to 01:50:00 ± 3 vteřiny, bylo podezření, že v této chvíli probíhala nějaká pravidelná údržba systému, která zahlcením procesoru by snad mohla způsobit takový problém. Ze systémového záznamu, konkrétně `/var/log/ntpdate`, bylo zjištěno, že se v tuto dobu spouštěl `ntpdate`, takže k chybě pravděpodobně došlo posunutím systémového času zpět, což zapříčinilo, že nové příchozí pakety měly menší časový údaj. A jelikož většina spojení trvala pár sekund, několikrát se stalo, že po vytvoření spojení se posunul čas o několik málo vteřin a než se tento posun stačil „dohnat“, spojení bylo ukončeno. Vzhledem k tomu, že těchto spojení bylo identifikováno pouze 12, byla tato spojení ignorována, a tedy neexportována sondou. Tento předpoklad jsme si ověřili analýzou původního logu z *tcpdump*, opět pomocí programu *Wireshark*, viz obrázek 6.1.

Delta time	Arrival Time	Source	Destination	Info
283764.0	Dec 24, 2008 01:50:00.881797	62.193.249.43	147.229.7.6	34363 > ssh [SYN] Seq=
0.000016	Dec 24, 2008 01:50:00.881813	147.229.7.6	62.193.249.43	ssh > 34363 [SYN, ACK]
0.023367	Dec 24, 2008 01:50:00.905180	62.193.249.43	147.229.7.6	34363 > ssh [ACK] Seq=
0.004640	Dec 24, 2008 01:50:00.909820	147.229.7.6	62.193.249.43	Server Protocol: SSH-2
0.023353	Dec 24, 2008 01:50:00.933173	62.193.249.43	147.229.7.6	34363 > ssh [ACK] Seq=
0.000875	Dec 24, 2008 01:50:00.934048	62.193.249.43	147.229.7.6	Client Protocol: SSH-2
0.000021	Dec 24, 2008 01:50:00.934069	147.229.7.6	62.193.249.43	ssh > 34363 [ACK] Seq=
0.000574	Dec 24, 2008 01:50:00.934643	147.229.7.6	62.193.249.43	Server: Key Exchange I
0.038486	Dec 24, 2008 01:50:00.973129	62.193.249.43	147.229.7.6	Client: Key Exchange I
0.039264	Dec 24, 2008 01:50:01.012393	147.229.7.6	62.193.249.43	ssh > 34363 [ACK] Seq=
0.034956	Dec 24, 2008 01:50:01.047349	62.193.249.43	147.229.7.6	Client: Diffie-Hellman
0.000007	Dec 24, 2008 01:50:01.047356	147.229.7.6	62.193.249.43	ssh > 34363 [ACK] Seq=
0.020170	Dec 24, 2008 01:50:01.067526	147.229.7.6	62.193.249.43	Server: New Keys
0.034755	Dec 24, 2008 01:50:01.102281	62.193.249.43	147.229.7.6	Client: New Keys
0.036777	Dec 24, 2008 01:50:01.139058	147.229.7.6	62.193.249.43	ssh > 34363 [ACK] Seq=
0.042663	Dec 24, 2008 01:50:01.181721	62.193.249.43	147.229.7.6	34363 > ssh [PSH, ACK]
0.001856	Dec 24, 2008 01:50:01.183577	147.229.7.6	62.193.249.43	ssh > 34363 [ACK] Seq=
0.000047	Dec 24, 2008 01:50:01.183624	147.229.7.6	62.193.249.43	Server: Unknown (249)[
0.048748	Dec 24, 2008 01:50:01.232372	62.193.249.43	147.229.7.6	Encrypted request packe
0.002701	Dec 24, 2008 01:50:01.235073	147.229.7.6	62.193.249.43	Encrypted response pac
0.042623	Dec 24, 2008 01:50:01.277696	62.193.249.43	147.229.7.6	Encrypted request packe
0.000019	Dec 24, 2008 01:50:01.277715	62.193.249.43	147.229.7.6	34363 > ssh [FIN, ACK]
0.000339	Dec 24, 2008 01:50:01.278054	147.229.7.6	62.193.249.43	ssh > 34363 [FIN, ACK]
-4.27193	Dec 24, 2008 01:49:57.006118	62.193.249.43	147.229.7.6	34363 > ssh [ACK] Seq=

Obrázek 6.1: Spojení ovlivněné aktualizací času

Toto rozšíření proudově orientovaného analyzátoru síťového provozu schopného exportovat data v NetFlow formátu – *softflowd* – je realizováno jako rozšíření, které se aktivuje parametrem `-R`, ale pouze v kombinaci s parametrem `-r <soubor>`, který určuje, že se data nebudou zachytávat na síťovém rozhraní, ale načítat ze souboru. Rozšíření je integrováno pouze do exportovacího protokolu verze 5. Celý program i s rozšířením je dostupný na adrese http://code.google.com/r/d0nald86-real-dates/ jako klon původní aplikace a probíhá komunikace s autorem na integrování rozšíření do standardní verze.

6.2.1 Data z útoků

Nejprve bylo nutné odstranit prázdné soubory kvůli hromadnému zpracování příkazem `tcpslice`, který nám spojil všechna data do jediného souboru `ssh-honeypot-merged_all`, pro lepší další manipulaci. Po vyřešení všech komplikací s rozšířením funkčnosti sondy jsme nejdříve zapnuli kolektor
`sudo nfcapd -w -p 23456 -b localhost -I honeypot -l replay1/` a pomocí příkazu `softflowd -d -v5 -p 23456 -n localhost -r ssh-honeypot-merged_all -R` spustili zpracování. Výsledné soubory jsme pomocí
`nfdump -R replay1/ -w netflow/honeypot.nfcapd` spojili do jednoho souboru.

Protože parametr `port 22` příkazu `tcpdump`, kterým jsme původní data získali, nerozlišuje mezi portem zdrojovým a cílovým, museli jsme odfiltrovat všechna spojení, která nemají cílovou IP adresu `147.229.7.6` a cílový port `22`, nebo nemají zdrojovou IP adresu `147.229.7.6` a zdrojový port `22`.

Pro vyřazení omylů, kdy si někdo při pokusu o přihlášení na administrativní SSHD spletl port, odfiltrujeme všechny pokusy o spojení z rozsahu VUT (`147.229.0.0/16`). A nakonec ještě několik IP adres, z kterých bylo prováděno testování funkčnosti při spouštění *honeypotu*. Všechna pravidla jsou zkombinována v souboru `filter_file`, viz kód 6.2. Pomocí těchto filtrovacích pravidel byla data ještě jednou zpracována příkazem
`nfdump -r honeypot.nfcapd -f filter_file -w honeypot.filtered.nfcapd`. Tímto jsme konečně získali data z útoků ve formátu NetFlow.

Kód 6.2: Filtrovací pravidla pro data z *honeypotu*

```
NOT (
        (
                SRC NET 147.229.0.0/16 AND DST IP 147.229.7.6
        ) OR (
                DST NET 147.229.0.0/16 AND SRC IP 147.229.7.6
        )
) AND NOT (
        (
                NOT SRC IP 147.229.7.6 AND SRC PORT 22
        ) OR (
                NOT DST IP 147.229.7.6 AND DST PORT 22
        )
)
AND NOT IP 147.229.7.5
AND NOT IP 88.208.115.106
AND NOT IP 193.85.255.30
```

6.2.2 Data z korektních spojení

Další část byla převod dat korektních spojení. Před zahájením zpracování a převodu samotných dat do NetFlow formátu bylo potřeba vytvořit filtr, pro odstranění útočníků zmíněných na začátku této kapitoly. Pro tento účel posloužil soubor `\var\log\syslog` (ze stroje *lenny*). Jako základ jsme použili soubor `failed_ip` s neúspěšnými pokusy uživatelů, respektive jejich IP adresami a druhý soubor `accepted_ip` s úspěšně přihlášenými uživateli. Ze souboru `failed_ip` jsme odstranili všechny IP adresy, které se vyskytovaly v souboru `accepted_ip`, protože se patrně jednalo o situace kdy uživatel zadal na první pokus špatné heslo, ale posléze se mu podařilo úspěšně přihlásit. Našel se i uživatel, kterému se nepodařilo přihlásit ani na 6. pokus, a to byl důkladně instruován. Proto byl ještě jednou důkladně

prozkoumán systémový záznam, zda se mezi neúspěšnými uživateli nevyskytuje uživatel, jež použil některé přihlašovací jméno z námi nově vytvořených účtů. Jelikož jména účtů měla souvislost buď se školou, nebo s konkrétní osobou, pro kterou byl účet vytvářen, bylo nepravděpodobné, že by neúspěšný uživatel s takovým jménem byl opravdu útočník. Všechny podezřelé záznamy byly pro ověření konzultovány přímo s konkrétními uživateli.

Další kroky byly obdobné jako při zpracování dat z útoků. Jednotlivé soubory jsme spojili do jednoho, pomocí software sondy exportovali na kolektor a z výsledného souboru jsme pomocí výše zmíněných filtrovacích pravidel odstranili nežádoucí záznamy.

V tabulce 6.2 vidíme přehled NetFlow statistiky pro data z *lennyho* a *honeypotu* jak v nefiltrované podobě, tak i po odstranění nežádoucích spojení.

	lenny		***honeypot***	
	nefiltrovaný	**filtrovaný**	**nefiltrovaný**	**filtrovaný**
Ident:	lenny	lenny	honeypot	honeypot
Flows_tcp:	2100	698	1377565	1376817
Packets_tcp:	33909105	24694963	19646584	19262848
Bytes_tcp:	12718325564	3862024533	3348542078	2753302719
First:	1269298177	1269298177	1216304739	1216453660
Last:	1274198972	1274198972	1285704047	1285704047
msec_first:	87	87	535	964
msec_last:	155	155	564	564
Sequence failures:	0	0	0	0

Tabulka 6.2: Přehled NetFlow statistiky

6.3 Detekce

V této části si popíšeme detekci pomocí programu *Weka*. Krátce si popíšeme vlastnosti tohoto programu. Protože možnosti tohoto programu sahají daleko za hranice naší práce, popíšeme si pouze funkce, které budeme přímo potřebovat pro řešení našeho problému. Dále zmíníme možnosti a omezení použitého klasifikačního algoritmu. V sekci 6.3.2 popíšeme výchozí formát vstupních dat používaný programem *Weka*.

6.3.1 *Weka*

Weka[1] je open source aplikace pro dolování dat napsaná v Javě. Sestává z kolekce algoritmů právě pro dolování dat. Obsahuje nástroje pro předzpracování, klasifikaci, seskupování, asociaci, výběr atributů, vizualizaci a další.

Grafické uživatelské rozhraní nám nabízí na výběr mezi čtyřmi základními režimy *Eplorer*, *Experimenter*, *KnowledgeFlow* a *Simple CLI*. V našich pokusech budeme využívat režim *Explorer*.

Hlavní funkce jsou rozděleny pomocí záložek v pořadí *Preprocess*, *Classify*, *Cluster*, *Associate*, *Select attributes* a *Visualize*.

Na záložce *Preprocess* si vybereme vstupní soubor. Poté, co si *Weka* načte všechna data, nám zpřístupní i ostatní záložky. V sekci *Filter* máme k dispozici různé filtry pro

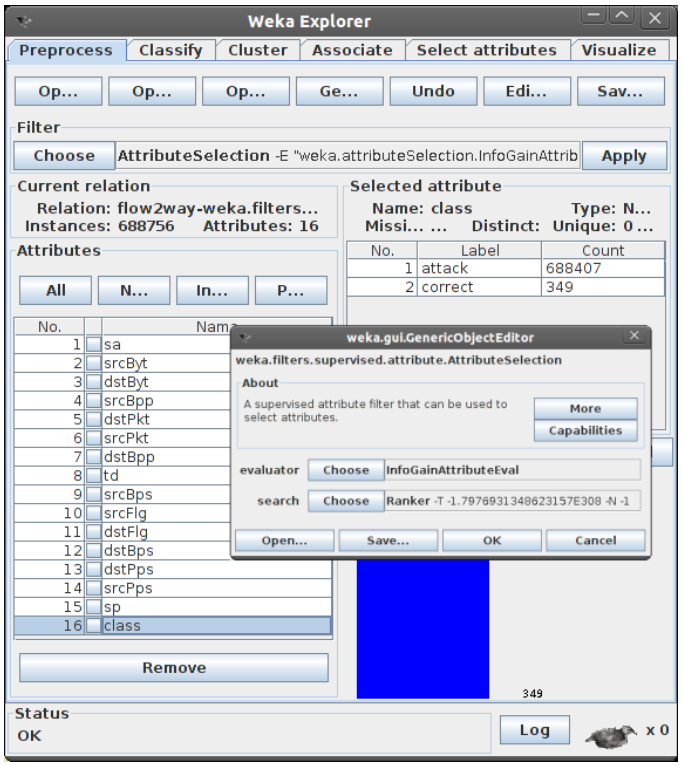
[1] Námi popisovaná verze programu Weka je 3.6.0.

úpravu konkrétních vlastností nebo celých záznamů. V sekci *Attributes* můžeme vybrat určitý atribut a jeho podrobnosti se zobrazí v sekci *Selected attribute* a pod ní grafická reprezentace rozložení tohoto atributu v záznamech.

V záložce *Classify* probíhá samotná klasifikace. Po vybrání klasifikačního algoritmu, můžeme po kliknutí na název upravit jeho parametry. Sekce *Test options* nám dává na výběr ze čtyř variant testů. Pod tlačítkem *More options...* – více možností – můžeme nastavit např. tzv. cenově závislé vyhodnocení *cost-sensitive*, kterým říkáme, že chyba pro jednu třídu je X-krát „dražší“, neboli závažnější, než pro druhou. V *Result list* – seznamu výsledků – můžeme procházet jednotlivé testy.

A konečně *Visualize* nám zobrazí dvourozměrný graf libovolných dvou atributů.

Na obrázku 6.2 vídíme spuštěný *Explorer* na záložce *Preprocess* s nahraným datasetem *flow2way*, ručně odstraněnými atributy cílové IP adresy a cílového portu a pomocí hodnotícího filtru `AttributeSelection` seřazené atributy od nejvýznamnějšího. Konkrétní hodnotící algoritmus můžeme vidět v pop-up okně, které je v obrázku rovněž vidět.



Obrázek 6.2: Náhled programu *Weka*

6.3.2 ARFF

ARFF je zkratka pro Attribute-Relation File Format. Jedná se o textový formát, který popisuje seznam záznamů sdílících stejné vlastnosti. Skládá se z hlavičky a samotných dat.

Hlavička začíná pojmenováním relace, s kterou pracujeme, např. `@RELATION flow`. Dále uvedeme všechny parametry pomocí klíčového slova `@ATTRIBUTE` a to ve formátu `@ATTRIBUTE <name> <type>`, kde `name` je námi zvolené pojmenování daného parametru a `type` vyjadřuje datový typ.

Datový typ může být jeden ze čtyř základních typů:

1. `numeric`,
2. `integer` chová se jako `numeric`,
3. `real` chová se jako `numeric`,
4. <výčtový-typ>,
5. `string`,
6. `date` [<formátovací-řetězec>].

U typu `numeric` mohou být hodnoty jak z oboru celých čísel, tak z oboru reálných čísel.

Výčtový typ je vyjádřen pomocí všech možných hodnot daného parametru např. `@ATTRIBUTE flag {hodnota1, hodnota2, ...,hodnotaN}`.

`string` se používá pro textový řetězec.

Poslední typ `date` využijeme, pokud potřebujeme vyjádřit datum. Pomocí volitelného parametru můžeme určit v jakém formátu je datum uvedeno. Výchozí formát přijímaného řetězce je podle normy ISO-8601[9]. Všechna klíčová slova určující typ `numeric`, `integer`, `real`, `string`, `date` jsou *case insensitive*.

Část, ve které jsou uvedena samotná data, je uvozena klíčovým slovem `@data`. Každý záznam je uveden na vlastním řádku a jednotlivé parametry jsou odděleny čárkou – `,`. Parametry obsahující mezeru, musí být uvedeny v apostrofech. V případě, že některá hodnota pro daný záznam chybí, vyjadřuje se pomocí jednoho znaku otazníku – `?`. Text týkající se ARFF byl zpracován na základě informací v [25].

6.3.3 Vlastní detekce

To, že detekce útoků na úrovni NetFlow bude prováděna pomocí programu *Weka*, jsme zmínili na začátku této kapitoly, teď si popíšeme algoritmus, který budeme používat pro samotnou klasifikaci.

Jedná se o algoritmus *Random Forests* [2] vyvinutý Leo Breimanem a Adele Cutler, který pro hodnocení používá několik rozhodovacích stromů a výstupní hodnocení je založeno na převažujícím hodnocení jednotlivých rozhodovacích stromů. Každý rozhodovací strom je konstruován tak, že pro každý uzel se vybírá m náhodných parametrů a použije se nejlepší z nich. m by mělo být daleko menší než celkový počet parametrů M. Většinou se využívá rovnice $m = log_2 M$, kde m je počet náhodně vybraných argumentů a M je celkový počet parametrů. m zůstává konstantní pro generování všech stromů.

Random Forests byl vybrán, protože se využívá právě u problémů, kde existuje velký počet prediktorů (atributů), z nichž každý sám o sobě obsahuje pouze jen málo informací o závislé proměnné (`class`). Další významná věc je, že *Random Forest*, respektive obecně rozhodovací stromy, jsou snadno implementovatelné a kombinací použití s programem *Weka* je přímo zajištěno provázání s programovacím jazykem Java. A pomocí API je možno využívat přímo z vlastního programu, nebo si pomocí grafického rozhraní připravit model, a uložit jej jako Java Object File.

Další důležitá část je způsob vyhodnocení úspěšnosti daného modelu, přesněji předpokládaná úspěšnost. Pokud bychom pro trénování modelu i pro jeho testování použili všechna, tedy stejná, data, dosáhli bychom velmi optimistické úspěšnosti, pravděpodobně blížící se 100%. Proto potřebujeme data rozdělit na část pro trénování a část pro testování.

Ovšem takovým způsobem se nám některá data nedostanou do tréninkové množiny a jejich vyhodnocení při testování bude pravděpodobně neúspěšné. Pro vyřešení tohoto problému se používá metoda *cross-validation*, která rozdělí data na n námi definovaných dílů, s co nejrovnoměrnějším zastoupením všech tříd v jednotlivých dílech, jako v původním datasetu, označovaný jako *n-fold*. Přičemž použije $n-1$ dílů pro trénování a jeden díl pro testování. Celý proces se provádí pro každý díl, tedy n-krát. Jednotlivé dílčí výsledky se vyhodnocují dohromady. Tímto způsobem je zajištěno, že každý záznam bude použit jak při trénování, tak i při testování.

Rozsáhlé testování ukázalo, že *10-fold*, tedy rozdělení na 10 dílů, je nejlepší množství pro vyhodnocení nejrůznějších algoritmů. Nicméně to, že by to vždy muselo být právě 10, není dané a je pravděpodobné, že *5-fold*, nebo *20-fold cross-validation* budou stejně úspěšné, viz [26, 5.2, p. 150].

Protože v praxi je běžné používat jednu třetinu pro testování a zbylé dvě pro trénování, viz [26, 5.2, p .150] a také pro ušetření času při testování, budeme používat *3-fold cross-validation*.

Jelikož program *Weka* [8], který pro vstupní data používá jako výchozí formát ARFF, potřebovali jsme data v textové podobě (nfcapd soubor je binární). Proto jsme opět použili *nfdump*, kterému jsme pomocí parametru `-o` nadefinovali vlastní formát výstupních dat. Tím jsme sice dosáhli vypsání pouze pro nás zajímavých parametrů v námi zvoleném pořadí, ale možnosti vlastního formátu nejsou až tak flexibilní jak bychom potřebovali.

Trochu předběhneme, a prozradíme, že námi vybraný klasifikační algoritmus neumožňuje zpracování řetězců, proto jsme museli data ještě prohnat přes upravující skript, který IP adresy převede na čísla a hodnoty větší než 1024^2, které *nfdump* převedl na M (mega), G (giga) případně T (tera), musíme převést zpět do jejich absolutní podoby pomocí `correct.arff.awk`, viz kód 6.3.

Při převodu do ARFF jsme se snažili využít všechny parametry, které nám NetFlow nabízí, ale zase jen do takové míry, kdy nám daný parametr přináší určitou informativní hodnotu. Právě z důvodu nulového přínosu informací kvůli definování pole, na kterém je tento projekt aplikován, byly vypuštěny parametry *Address Family*, určující, jestli se jedná o protokol IPv4 nebo IPv6, *Protocol*, určující TCP, nebo UDP, *Input IF* a *Output IF* určující vstupní, respektive výstupní rozhraní a *Tos* určující typ služby. *Address Family* a *Protocol* byly vypuštěny, protože všechna data jsou ze spojení TCP přes protokol IPv4, a tudíž mají u všech spojení konstantní hodnotu. Parametry *Input IF* a *Output IF* zase nejsou použity kvůli použitému způsobu získání NetFlow dat a *Tos* z důvodu nevyužívání tohoto parametru a tudíž také konstantní hodnotou pro všechna spojení.

Po nahrání prvního datasetu `hon-lenny.comp.arff` do programu *Weka*, jsme si uvědomili, že adresa serveru přesně kopíruje atribut `class`, určující příslušnost k hodnotící třídě. Proto bylo nutné odstranit IP adresu serveru z testovaných dat, ale protože v NetFlow je spojení reprezentováno dvěma jednosměrnými proudy, objeví se původní cílová IP adresa jak v poli cílová IP adresa, tak i zdrojová IP adresa při reprezentaci datového proudu od serveru k uživateli. Z tohoto důvodu byl pro každý směr spojení vytvořen vlastní dataset pomocí filtrování výstupu z *tcpdump*, určující vždy buď cílovou `DST IP <IP adresa serveru>`, nebo zdrojovou `SRC IP <IP adresa serveru>` IP adresu serveru. Tímto jsme získali dva nové datasety, `hon-lenny.to.arff` reprezentující příchozí a `hon-lenny.from.arff` reprezentující odchozí proud z pohledu serveru. U těchto datasetů již bylo jednoduché odstranit IP adresu serveru.

Kód 6.3: Skript pro úpravu ARFF dat `correct.arff.awk`

```
#!/usr/bin/awk -f
BEGIN {
        OFS=","; FS="|"; sai=2; dai=4; pkti=6;
        byti=7; bpsi=9; ppsi=10; bppi=11;
}
{
        $sai=address($sai);
        $dai=address($dai);
        $pkti=units($pkti);
        $byti=units($byti);
        $bpsi=units($bpsi);
        $ppsi=units($ppsi);
        $bppi=units($bppi);
        print $0;
}

function units(field){
        if(field ~/.*M/){
                field*=1024*1024;
}else if(field~/.*G/){
                field*=1024*1024*1024;
}else if(field~/.*T/){
                field*=1024*1024*1024*1024;
}
        return field;
}

function address(field){
        split(field,IP,".");
        field="";
        for(i=1;i<5;i++){
                l=length(IP[i]);
                if(l==1){
                        IP[i]="00" IP[i];
}else if(l==2){
                        IP[i]="0" IP[i];
}
                field=field IP[i];
}
        return field;
}
```

Ovšem při takovém rozdělení se nám ztratila vazba mezi odchozím a příchozím proudem spojení. Proto byl vytvořen ještě formát *flow2way*, viz výpis 6.4, který oba proudy spojuje do jednoho záznamu, popisujícího původní spojení. Dva záznamy jsou spojeny dohromady na základě shody zdrojové IP adresy jednoho záznamu a cílové IP adresu druhého záznamu a naopak. Obdobně také pro zdrojový a cílový port a ještě délky spojení. Spojení je prováděno pomocí rozšíření skriptu `correct.arff.awk`.

Kód 6.4: ARFF hlavička pro relaci `flow2way`

```
@relation 'flow2way'

@attribute td integer
@attribute sa integer
@attribute sp integer
@attribute da integer
@attribute dp integer
@attribute srcPkt integer
@attribute srcByt integer
@attribute srcFlg {.A...., .AP..., .AP..F, .APR.., .APRS., .APRSF, .AP.S.,
.AP.SF, .A.R.., .A.RS., .A.RSF, .A..S., .A..SF, ...R.., ...RS., ....S.}
@attribute srcBps integer
@attribute srcPps integer
@attribute srcBpp integer
@attribute dstPkt integer
@attribute dstByt integer
@attribute dstFlg {.A...., .AP..., .AP..F, .APR.., .APRS., .APRSF, .AP.S.,
.AP.SF, .A.R.., .A.RS., .A.RSF, .A..S., .A..SF, ...R.., ...RS., ....S.}
@attribute dstBps integer
@attribute dstPps integer
@attribute dstBpp integer
@attribute class {attack, correct}
```

Další zavádějící atribut je cílový port. Vzhledem k tomu, že na *honeypotu* byl po celou dobu získávání dat konstantní a na *lennym* se několikrát měnil, nedají se tyto atributy porovnávat. A také při detekci síťového provozu v reálné situaci, by všechna spojení vedla na jeden port, ať už výchozí 22, nebo jakýkoli jiný. A poslední problémový atribut ještě zdrojová IP adresa. Při hodnocení na základě zdrojové IP adresy bychom automaticky diskriminovali určitou lokalitu a naopak. Vzhledem k tomu, že IP adresy mají danou přibližnou zeměpisnou polohu, např. stroje s adresami z rozsahu VUT se nacházejí v Brně (s výjimkou VPN připojení apod.). Ve výsledku je to právě zdrojová IP adresa, kterou hodnotíme, pouze zastoupena jednotlivými vlastnostmi/atributy spojení. Protože při reálné aplikaci detekce útoků bychom jako filtrovací pravidlo pro korektní spojení a útok použili pravděpodobně právě zdrojovou IP adresu, která nám nejlépe charakterizuje útočníka. Tuto domněnku nám potvrdil filtr pro hodnocení atributů, viz tabulka 6.3, který po odstranění cílové IP adresy a cílového portu, určil právě zdrojovou IP adresu jako nejdůležitější atribut.

Pro námi zvolený klasifikační algoritmus *Weka* nenabízí možnost vizualizace klasifikačního stromu, respektive klasifikačních stromů, a to ani v textové podobě. Takže se velice špatně dá ovlivňovat výsledný model, ať už redukcí záznamů, nebo atributů, protože nevíme, který atribut, v kterém stromu hrál největší roli při klasifikaci daného záznamu.

Chybovost algoritmu *Random Forests* ovlivňují dvě věci:

- Vzájemný vztah mezi dvěma rozhodovacími stromy v „lese“. Zvýšením vzájemného vztahu se zvětšuje i chybovost.
- Síla každého rozhodovacího stromu v „lese“. Strom s nízkou chybovostí je silný klasifikátor, proto zvýšením síly jednotlivých stromů snížíme chybovost. Snížení vzájemného vztahu dosáhneme větším množstvím atributů, ovšem při zařazení slabých atributů, snižujeme sílu jednotlivých stromů.

Jde tedy o to zvolit nejlepší počet náhodných atributů a které atributy do klasifikace ještě zahrnout, a které naopak snižují úspěšnost hodnocení a budou vyloučeny. Jak je

uvedeno v [26, 7, p. 320] míra náhodnosti může být ověřena pouze pomocí experimentu. Proto byly prováděny testy se zbývajícími atributy pro zjištění nejvhodnější kombinace. Pro vytvoření určitého výchozího bodu, byly spuštěny testy i nad všemi zmíněnými datasety. Výsledné hodnoty těchto testů jsou uvedeny na přiloženém CD.

Testy byly spouštěny s výchozími parametry `RandomForest -I 10 -K 0 -S 1`. Vytváří se tak 10 stromů s automaticky počítaným množstvím náhodných atributů a s počátečním *seed* 1. Postupně byly odebírány méně důležité atributy. Po odebrání každého atributu byl spuštěn nejprve test s automaticky generovaným počtem náhodných atributů a poté minimálně dva další. První, s o jeden menším a druhý, o jeden větším. Pořadí bylo určeno pomocí záložky *Select attributes*. Vyhodnocovací algoritmus byl zvolen `InfoGainAttributeEval` a vyhledávací metoda `Ranker -T -1.7976931348623157E308 -N 1` podle návodu v [12]. Tabulka 6.3 uvádí přehled pořadí atributů podle důležitosti při klasifikaci pro všechny výše zmíněné datasety.

Pojmenovaní jednotlivých atributů vychází z formátu používaného v *nfdump*. Pouze parametry, které se kvůli spojení dvou datových proudů objevují dvakrát, jsou uvozeny buď `src`, nebo `dst` určující příslušnost k danému proudu. Atribut `td` určuje délku spojení, `sa` zdrojovou IP adresu, `sp` zdrojový port, `da` cílovou IP adresu, `dp` cílový port, `Pkt` počet paketů, `Byt` počet bajtů, `Flg` TCP příznaky, `Bps` množství bajtů za sekundu, `Pps` počet paketů za sekundu a `Bpp` počet bajtů na paket.

Pro testy s nejlepšími výsledky byly spuštěny ještě *cost-sensitive* klasifikace. „Cena“ chyby pro třídu `correct` byla stanovena postupně na 1972, 3945 a 7890. Výchozí hodnotu ceny chyby jsme určili jako poměr mezi počtem spojení z útoků a počtem korektních spojení, pro vyrovnání množství záznamu v obou třídách. Násobky této hodnoty pak říkáme, kolikrát je daná chyba pro nás závažnější.

Přehled nejdůležitějších parametrů výsledků vybraných testů je uveden v tabulce 6.4. Všechny uvedené testy jsou z datasetu `hon-lenny.flow2way`, který obsahuje celkem 688756 záznamů, z toho 688407 jsou záznamy z třídy `attack` a 349 z třídy `correct`. Označení jednotlivých testů je ve formátu `firstX.Yf[.cost(Z)]`, kde `X` označuje počet nejdůležitějších parametrů podle tabulky 6.3, `Y` počet náhodně vybíraných atributů. Přítomnost `cost(Z)` v označení testu určuje, že se jedná o *cost-sensitive* klasifikaci a `Z` udává hodnotu použitou v cenové matici pro hodnocení *false positive* z pohledu klasifikátoru. Z pohledu třídy `correct` bychom ji označili jako *false negative*. Jedná se tedy o počet instancí třídy `correct` vyhodnocených jako `attack`. Do tohoto pořadí se nepočítají atributy `sa`, `da` a `dp`. Další sloupce vyjadřují počet chybně klasifikovaných instancí ze třídy `attack`, respektive `correct`, následované procentuálním vyjádřením chyby pro třídu `correct` a také pro obě třídy dohromady. V posledním sloupci je pak hodnota *Relative absolute error*, která je podrobněji zmíněna v další části. Kompletní přehled výsledků je uveden na přiloženém CD.

6.3.4 Vyhodnocení výsledků

Než budeme hodnotit výsledky jednotlivých modelů, musíme si určit podle jakého kritéria budeme hodnotit. Pro určení takového kritéria musíme nejprve přesně určit náš cíl. Ten byl samozřejmě správně klasifikovat všechna spojení. Ovšem jak už napovídají zmíněné *cost-sensitive* testy, správné určení korektního spojení je pro nás kritické. Protože v analogii s detekcí spamu v elektronické poště, odfiltrování regulérní zprávy může mít nedozírné následky. Obdobně zablokování oprávněného uživatele je nepřípustné. Proto pro nás není tolik důležitý celkový počet chybně klasifikovaných záznamů, ale především počet chybně klasifikovaných záznamů z třídy `correct`. Samozřejmě musíme sledovat i počet chyb pro `attack`,

	Název atributu			
Pořadí	`hon-lenny`	`hon-lenny.from`	`hon-lenny.to`	`hon-lenny.flow2way`
1.	da	sa	da	da
2.	sa	da	sa	sa
3.	byt	byt	byt	srcByt
4.	pkt	pkt	bpp	dstByt
5.	bpp	sp	pkt	srcBpp
6.	td	bpp	dp	dstPkt
7.	dp	td	td	srcPkt
8.	sp	bps	bps	dp
9.	flg	flg	flg	dstBpp
10.	bps	pps	pps	td
11.	pps	dp	sp	srcBps
12.				srcFlg
13.				dstFlg
14.				dstBps
15.				dstPps
16.				srcPps
17.				sp

Tabulka 6.3: Atributy seřazené podle důležitosti při klasifikaci

protože při neúměrně vysokém počtu chyb pro tuto třídu, by znamenalo, že klasifikátor pouze upřednostňuje třídu `correct`.

Počet chyb nám dává dobrý přehled o výkonu daného modelu, ovšem vzhledem k nevyváženému množství dat mezi našimi dvěma třídami, jedna chyba při vyhodnocení korektních spojení nám chybovost pro tuto třídu zvyšuje daleko výrazněji. Proto potřebujeme vyjádření chybovosti, které reflektuje rozdíl mezi množstvím dat v obou třídách. A klasifikátor pro predikci používá procentuální vyjádření pro obě možnosti, takže ani úspěšná predikce nemusí ještě znamenat, že model byl na 100 % úspěšný a naopak.

Ve výstupu programu *Weka* máme k dispozici hned několik hodnot vyjadřující míru chyby *Mean absolute error* 6.1, *Root mean squared error* 6.2, *Relative absolute error* 6.3 a *Root relative squared error* 6.4. V našem případě je pro nás důležitá míra chyby pro jednotlivé třídy bez ohledu na množství instancí v dané třídě. Tento pohled vyjadřuje *Relative absolute error* (RAE), protože při zvýšení počtu chyb pro `correct` se RAE zvyšuje výrazněji než při zvýšení počtu chyb pro třídu `attack` o stejné množství.

$$\frac{|p_1 - a_1| + \ldots + |p_n - a_n|}{n} \tag{6.1}$$

$$\sqrt{\frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{n}} \tag{6.2}$$

$$\frac{|p_1 - a_1| + \ldots + |p_n - a_n|}{|a_1 - \overline{a}| + \ldots + |a_n - \overline{a}|}, \text{ kde } \overline{a} = \frac{1}{n}\sum_{i}^{n} a_i \tag{6.3}$$

$$\sqrt{\frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{(a_1 - \overline{a})^2 + \ldots + (a_n - \overline{a})^2}} \tag{6.4}$$

Označení	Počet chyb pro třídu		Procento chyb		
	attack	correct	třídy correct	celkově	*Relative absolute error*
first13.4f	6	23	6,5900 %	0,0042 %	6,6361 %
first13.5f	5	24	6,8800 %	0,0042 %	6,6990 %
first13.6f	5	25	7,1600 %	0,0044 %	7,0968 %
first12.3f	5	27	7,7364 %	0,0046 %	7,1811 %
first12.4f	6	24	6,8768 %	0,0044 %	7,0925 %
first12.5f	5	24	6,8768 %	0,0042 %	6,2221 %
first11.3f	5	26	7,4499 %	0,0045 %	6,8703 %
first11.4f	4	23	6,5903 %	0,0039 %	6,5244 %
first11.5f	11	26	7,4499 %	0,0054 %	7,0795 %
first10.3f	5	25	7,1633 %	0,0044 %	6,6511 %
first10.4f	6	25	7,1633 %	0,0045 %	6,7515 %
first10.5f	5	21	6,0172 %	0,0038 %	5,7476 %
first9.3f	6	21	6,0172 %	0,0039 %	6,3637 %
first9.4f	4	21	6,0172 %	0,0036 %	6,4382 %
first9.5f	12	18	5,1576 %	0,0044 %	6,7228 %
first8.3f	7	21	6,0172 %	0,0041 %	6,5366 %
first8.4f	12	22	6,3037 %	0,0049 %	6,8369 %
first8.5f	12	20	5,7307 %	0,0046 %	6,3506 %
first7.3f	7	24	6,8768 %	0,0045 %	6,6081 %
first7.4f	6	21	6,0172 %	0,0039 %	6,3649 %
first7.5f	14	24	6,8768 %	0,0055 %	6,7654 %
first12.5f.cost(1972)	14	16	4,5845 %	0,0044 %	7,6671 %
first12.5f.cost(3945)	18	20	5,7307 %	0,0055 %	8,5132 %
first12.5f.cost(7890)	19	19	5,4441 %	0,0055 %	9,7622 %
first9.4f.cost(1972)	14	21	6,0519 %	0,0051 %	7,3129 %
first9.4f.cost(3945)	15	20	5,7307 %	0,0051 %	8,7339 %
first9.4f.cost(7890)	23	19	5,4441 %	0,0061 %	11,5538 %
first9.5f.cost(1972)	20	21	6,0172 %	0,0060 %	7,6217 %
first9.5f.cost(3945)	17	20	5,7307 %	0,0054 %	8,5381 %
first9.5f.cost(7890)	28	21	6,0172 %	0,0071 %	10,2827 %
first8.5f.cost(1972)	17	17	4,8711 %	0,0049 %	7,2660 %
first8.5f.cost(3945)	18	16	4,5845 %	0,0049 %	8,3960 %
first8.5f.cost(7890)	20	20	5,7307 %	0,0058 %	9,8978 %

Tabulka 6.4: Přehled nejdůležitějších parametrů vybraných výsledků klasifikace

p vyjadřuje předpovězenou hodnotu a a skutečnou. Výše zmíněné vzorce jsou převzaty z [26, table 5.8].

Metodou postupného odebírání atributů jsme opravdu dosahovali lepších výsledků až k `first9.5f`, který je s 18-ti chybami z 349 pro třídu `correct` v tomto ohledu nejúspěšnější. Poté už se úspěšnost začala opět snižovat. Nejlepším testem při hodnocení podle RAE vychází `first12.3f` s chybovostí 6,2221 %.

Celkově jsou jednotlivé modely zjevně přeučeny kvůli velkému množství dat z útoků. Obecně se metoda přeučení využívá právě pro ovlivnění výsledku sledované třídy, ale protože máme k dispozici pouze malé množství dat pro třídu `correct`, nemůžeme metodu přeučení použít tak, jak bychom potřebovali. Pro kompenzování rozdílného množství dat v obou třídách jsme pro testy s nejlepšími výsledky použili ještě *cost-sensitive* klasifikaci. Při těchto testech se výsledky, co do počtu chyb, opravdu zlepšovaly, ale při postupném zvyšování ceny byl efekt ve většině případů přesně opačný.

Nejlepší *cost-sensitive* byl `first12,5f.cost(1972)` s 16-ti chybami pro třídu `correct`. Dalším způsobem, kterým jsme kompenzovali přeučení na třídu `attack` bylo jednoduše zredukování počtu záznamů této třídy. Výsledky vyhodnocení modelu, vytvořeného na základě redukovaných dat, dosahovaly takřka absolutní úspěšnosti. Takový výsledek ovšem považujeme za přehnaně optimistický a pouhým vyhodnocením vytvořeného modelu na kompletních datech jsme dosahovali řádově vyšších chyb pro třídu `attack`.

Výsledky všech testů dosahují natolik dobrých hodnot, že nastavováním různých parametrů se je podařilo vylepšit už pouze částečně. Nicméně navržené postupy prokázaly svoji účinnost. Pro dosažení lepších výsledků pro třídu `correct` by bylo potřeba nejlépe stejné množství dat jako pro třídu `attack`.

Na závěr ještě zmíníme, že porovnávání spojení podle hodnot vztahujících se k rychlosti komunikace je ovlivněno tím, že data jsou ze dvou výkonově odlišných strojů a především velice odlišnou rychlostí připojení k internetu.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo nastudovat principy protokolu SSH, zmapování útoků na *honeypot*, návrh detekce botnetů a návrh detekce útočníků na úrovni NetFlow. Všechny cíle byly splněny.

V teoretické části jsem se díky pečlivému čtení, nejenom čtyř základních RFC, dozvěděl detaily o protokolu, který každý den používám k přihlášení, ať již na svůj, či školní server, a také jsem si uvědomil, jak rozsáhlá je specifikace takového protokolu.

V případě mapování útoků se výsledky s určitou odchylkou shodují s dřívějšími studiemi na toto téma.

V kapitole detekce botnetů bylo navrženo několik způsobů detekce a ty implementovány v rámci skriptu, který slouží také k dolování dat a jejich převedení do prezentovatelné formy v podobě grafů, generovaných pomocí *gnuplot*. Vyhodnotili jsme výsledky jednotlivých způsobů detekce a zdůraznili přínos jednotlivých metod. Vygenerované grafy jsou uloženy na přiloženém CD.

Šestá kapitola popisuje proces detekce útoků na úrovni Netflow od získávání dat až po samotnou klasifikaci pomocí programu *Weka*. Rozebíráme problémy spojené s převodem dat z nižších vrstev síťové komunikace do formátu NetFlow a následném použití v klasifikátoru. Navržený způsob detekce dosahoval velmi dobrých výsledků při vyhodnocování. Dalšího zlepšení bychom dosáhli použitím většího množství dat z korektních spojení, která bychom získali použitím vytíženějšího serveru, než je domací server *lenny*.

Snažili jsme se pracovat metodicky a precizně, protože jsme zpracovávali velké množství dat a než z prvotních dat dostaneme výsledný model, musíme data několikrát zpracovat. Nicméně, ani při zvýšené opatrnosti se nám nepodařilo vždy odhalit všechny chyby hned napoprvé a v některých případech to nebylo ani možné. Některé se totiž projeví až při zpracování v dalším kroku. Proto jsme jednotlivé kroky prováděli pouze se vzorkem dat, abychom odhalování chyb v jednotlivých krocích urychlili. Bohužel, některé chyby se projevily právě až při zpracování celého bloku dat. Příkladem je odhalení chybného zpracování časových údajů v programu *softflowd*.

Celkově mě tato problematika velice zajímá, protože bych se chtěl jednou uplatnit právě v oboru počítačových sítí a rád bych navázal tam, kde jsem skončil třeba v rámci diplomové práce.

Literatura

[1] Barrett, D. J.; Silverman, R. E.: *SSH, the Secure Shell: The Definitive Guide*. O'Reilly, 2001, ISBN 0-596-00011-1.

[2] Breiman, L.: Random Forests. *Machine Learning*, ročník 45, 2001: s. 5–32, ISSN 0885-6125.
URL http://dx.doi.org/10.1023/A:1010933404324

[3] Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), October 2004 [cit. 2010-11-18].
URL http://www.ietf.org/rfc/rfc3954.txt

[4] Daemen, J.; Rijmen, V.: *The design of Rijndael: AES–the advanced encryption standard*. Springer, 2002, ISBN 3-540-42580-2.

[5] Deri, L.: NetFlow, NetFlow-Lite and sFlow based Open Source Network Traffic Monitoring. [online], [cit. 2011-04-10].
URL http://www.ntop.org/

[6] Ehrsam, W. F.; Meyer, C. H. W.; Smith, J. L.; aj.: Message verification and transmission error detection by block chaining. [online], April 1976 [cit. 2010-05-17].
URL http://patft.uspto.gov/netacgi/nph-Parser?d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=4074066.PN.&OS=PN/4074066&RS=PN/4074066

[7] Fullmer, M.: flow-tools. [online], [cit. 2011-04-10].
URL http://www.splintered.net/sw/flow-tools/

[8] Hall, M.; Frank, E.; Holmes, G.; aj.: The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, ročník 11, č. 1, 2009: s. 10–18, ISSN 1931-0145.
URL http://dx.doi.org/10.1145/1656274.1656278

[9] Internatiol Organization for Standardization: ISO-8601. [online], [cit. 2011-04-04].
URL http://www.iso.org/iso/support/faqs/faqs_widely_used_standards/widely_used_standards_other/date_and_time_format.htm

[10] Jaquier, C.: Fail2Ban. [online], [cit. 2010-05-17].
URL http://www.fail2ban.org/

[11] Lehtinen, S.; C. Lonvick, E.: The Secure Shell (SSH) Protocol Assigned Numbers. [online], January 2006 [cit. 2009-11-02].
URL http://tools.ietf.org/html/rfc4250

[12] Markov, Z.; Russell, I.: An introduction to the WEKA data mining system. In *ITiCSE*, 2006, s. 367–368.
URL http://doi.acm.org/10.1145/1140124.1140127

[13] National Institute of Standards and Technology: Specification for the Advanced Encryption Standard (AES). [online], November 2001 [cit. 2010-04-24].
URL http://tools.ietf.org/html/rfc4254

[14] Peter, L.; Varian, H. R.: How Much Information. [online], 2003 [cit. 2009-11-02].
URL
http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/

[15] Ramsbrock, D.; Berthier, R.; Cukier, M.: Profiling Attacker Behavior Following SSH Compromises. *Dependable Systems and Networks, International Conference on*, 2007: s. 119–124, [cit. 2010-05-16].
URL http://doi.ieeecomputersociety.org/10.1109/DSN.2007.76

[16] van der Sanden, D.: Bug 1836 – 'undesired' behavior when using the -r flag. [online], [cit. 2011-04-10].
URL https://bugzilla.mindrot.org/show_bug.cgi?id=1836

[17] Seifer, C.: Analyzing Malicious SSH Login Attempts. [online], 2006, [cit. 2010-05-16].
URL http://www.symantec.com/connect/articles/
analyzing-malicious-ssh-login-attempts

[18] Sessink, O.: Jailkit. [online], [cit. 2010-05-17].
URL http://olivier.sessink.nl/jailkit/

[19] Shneiser, B.: *Applied cryptography: protocols, algorithms, and source code in C.* Wiley, 1996, ISBN 0-471-12845-7.

[20] The fprobe Team: NetFlow probes: fprobe and fprobe-ulog. [online], [cit. 2011-04-10].
URL http://fprobe.sourceforge.net/

[21] The Tcpdump team: Tcpdump/Libpcap public repository. [online], [cit. 2011-04-04].
URL http://www.tcpdump.org/

[22] Wikipedia: Brute force attack – Wikipedia, The Free Encyclopedia. [online], 2010, [cit. 2010-05-17].
URL http://en.wikipedia.org/wiki/Brute_force_attack

[23] Wikipedia: Exclusive or – Wikipedia, The Free Encyclopedia. [online], 2010, [cit. 2010-05-02].
URL http://en.wikipedia.org/wiki/XOR

[24] Wikipedia: Leet – Wikipedia, The Free Encyclopedia. [online], 2010, [cit. 2010-05-16].
URL http://en.wikipedia.org/wiki/Leet

[25] Wikispaces: ARFF (book version). [online], [cit. 2011-04-04].
URL http://weka.wikispaces.com/ARFF+(book+version)

[26] Witten, I. H.; Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems, San Francisco, CA: Morgan Kaufmann Publishers, druhé vydání, 2005, ISBN 0-12-088407-0.

[27] Ylonen, T.; C. Lonvick, E.: The Secure Shell (SSH) Authentication Protocol. [online], January 2006 [cit. 2009-11-02].
URL http://tools.ietf.org/html/rfc4252

[28] Ylonen, T.; C. Lonvick, E.: The Secure Shell (SSH) Connection Protocol. [online], January 2006 [cit. 2009-11-02].
URL http://tools.ietf.org/html/rfc4254

[29] Ylonen, T.; C. Lonvick, E.: The Secure Shell (SSH) Protocol Architecture. [online], January 2006 [cit. 2009-11-02].
URL http://tools.ietf.org/html/rfc4251

[30] Ylonen, T.; C. Lonvick, E.: The Secure Shell (SSH) Transport Layer Protocol. [online], January 2006 [cit. 2009-11-02].
URL http://tools.ietf.org/html/rfc4253

Dodatek A

Obsah CD

- Elektronická verze textu bakalářské práce ve formátu PDF a její zdrojový text pro LaTeX.
- Adresář `scripts` se skripty použitými při tvorbě této práce.
- Adresář `data` obsahující datové soubry z jednotlivých částí této práce.
- Adresář `graphs` s grafy vygenerovanými pomocí `map_attacks.py`.
- Archiv `openssh-5.0p1.tar.gz` s upraveným souborem `pass-auth.c`.
- Archiv `softflowd.tar.gz` s upraveným zdrojovým kódem.

Dodatek B

Seznam použitých zkratek

- **TELNET** – Síťový prokol pro obousměrnou, textově orientovanou komunikaci.
- **TCP** – Protokol transportní vrstvy podle modelu ISO/OSI.
- **UDP** – Nespolehlivý protokol transportní vrstvy podle modelu ISO/OSI.
- **SSHD** – Aplikace, zpracovávající příchozí žádosti o vzdálené přihlášení pomocí protokolu SSH, viz sekce 2.1.
- **SSH** – Služba vzdáleného zabezpečeného připojení, viz sekce 2.1.
- **PAM** – **P**luggable **A**uthentication **M**odule je modulovací autentizační systém pro přidělování oprávnění.
- **IP adresa** – Číslo, které jednoznačně identifikuje rozhraní v počítačové síti, která používá internetový protokol.
- **ARFF** – Formát vstupního souboru používaný programem *Weka* [8].
- **pcap** – Formát souboru pro ukládání síťové komunikace používaný programy jako *tcpdump* [21] a *Wireshark*.
- **NAT** – **N**etwork **A**ddress **T**ranslation je metoda překladu síťových adres průchozích paketů přes router.
- **API** – **A**pplication **P**rogramming **I**nterface označuji rozhraní pro programování aplikací.
- **DDoS** – **D**istributed **D**enial **o**f **S**ervice je typ distribuovaného útoků k zamezení určité služby.
- **PDF** – **P**ortable **D**ata **F**ormat je přenosný formát dokumentů vytvořený firmou Adobe.
- **RFC** – **R**equest **F**or **C**omments se používá pro označení řady standardů popisujících internetové protokoly, systémy, apod.
- **VPN** – **V**irtual **P**rivate **N**etwork je bezpečný způsob připojení přes veřejnou síť do privátní místní sítě.