VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2018 David Karger

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

HONEYPOT: NÁSTROJ V BOJI PROTI MALWARE
HONEYPOT: A TOOL FOR FIGHTING MALWARE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **David Karger**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jan Hajný, Ph.D.**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**
Ústav telekomunikací

Student: David Karger ***ID:*** 186526
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Honeypot: Nástroj v boji proti malware

POKYNY PRO VYPRACOVÁNÍ:

Práce je zaměřena na nasazení Honeypotů pro boj s malwarem. Cílem je nastudování problematiky tzv. honeypotů a jejich použití při detekci a analýze škodlivého kódu, analýza existujících metod a nástrojů z hlediska: phishingu (zachycení podvodných emailů), IoT (identifikace útoků, extrakce vzorků) a PC (získávání souborů). Analýza existujících nástrojů bude zaměřena především na oblast IoT (chytrá zařízení v domácnosti) a průmyslové sítě. Výstupem práce bude funkční honeypot. Požadavky na výsledné řešení jsou: schopnost systému získávat informace o aktuálně se šířících hrozbách, automatizace nasazení a autonomnost systému (odesílání reportů, vzorků atd.).

DOPORUČENÁ LITERATURA:

[1] SPITZNER, Lance. Honeypots: tracking hackers. Boston: Addison-Wesley, c2003. ISBN 978-0321108951.

[2] The Honeynet Project [online]. [cit. 2017-09-12]. Dostupné z: https://www.honeynet.org.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 29.5.2018

Vedoucí práce: doc. Ing. Jan Hajný, Ph.D.
Konzultant: Michal Salát, Avast

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Tato bakalářská práce se zabývá nasazením honeypotu pro boj s malwarem. Cílem bylo nastudování problematiky honeypotů a jejich použití při detekci a analýze malware. První část je věnována malware, jeho historii a jednotlivým typům. Dále je popsán tzv. botnet. Poslední část je věnována samotnému honeypotu a jeho rozdělení. Praktická realizace je uskutečněna přes honeypot Cowrie a Mailoney.

Klíčová slova

malware, bezpečnost, honeypot, detekce útoků, detekce zranitelností

Abstract

This bachelor thesis is focused on deploying a honeypot to fight malware. The aim was to study the issue of honeypots and their use in detection and analysis of malware. The first part is dedicated to malware, its history and individual types. The so-called botnet is described in the next part. The last part is devoted on the honeypot itself and its distribution. The practical realization is done through honeypots Cowrie and Mailoney.

Keywords

malware, security, honeypot, detection of attacks, detection of vulnerabilities

Bibliografická citace:

KARGER, D. Honeypot: Nástroj v boji proti malware. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. 53 s. Vedoucí bakalářské práce doc. Ing. Jan Hajný, Ph.D..

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma „Honeypot: Nástroj v boji proti malware“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona c. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonu (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne **28. května 2018** ……
podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce doc. Ing. Janu Hajnému, PhD. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.
Mé poděkování také patří konzultantovi panu Michalu Salátovi z firmy Avast za vstřícnost a pomoc při získávání údajů pro praktickou část práce.

V Brně dne **28. května 2018** ……

podpis autora

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Obsah

Seznam obrázků

Úvod

Dnešní doba se vyznačuje prudkým nárůstem a vývojem technologií. Velká část informací a dat se nachází nejen v počítačích, ale i v sítích. Často nejde pouze o informace obecné, ale ukládáme data soukromá, důvěrná i tajná. Mnoho uživatelů žije v domnění, že data uložená v jejich zařízeních jsou v bezpečí.

Společně s rozmachem moderní techniky se však rozvíjí i možnosti útočníků. Při každém vylepšení ochrany dat se zákonitě snaží protistrana najít řešení, jak tuto ochranu obejít a ke zdroji dat se dostat.

Z hlediska boje proti napadení je nutné znát aktivity a metody těch, kdo se snaží ochranu překonat. Jednou z možností detekce je i honeypot, kterému se věnuje tato bakalářské práce.

V první kapitole je vysvětlen pojem malware a jeho klasifikace. Malware je často používán útočníky, aby mohli získat neoprávněný přístup do zařízení.

V druhé kapitole je vysvětlen pojem botnet. Botnet je nejčastěji využíván k provedení tzv. DDoS útoku.

Třetí kapitola je věnována honeypotu a jeho rozdělení. Následně jsou uvedeny příklady honeypotů k jednotlivým úrovním interakce. Kapitola je zakončena popisem honeypotů v sítích SCADA.

V praktické části je popsáno propojení z honeypotů Cowrie a Mailoney s databází a jejich zobrazení v Grafana. Dále je popsáno automatizované nasazení použitých komponent.

Cílem bylo vyzkoušet propojení honeypotů s databází a následné zobrazení získaných dat. Nakonec pak vytvoření automatizovaného nasazení komponent a udržení jejich jednotných konfiguračních souborů.

1 MALWARE

1.1 Historie

Jako první malware jsou odborníky uznávány programy The Creeper a Brain. Tyto programy a několik dalších, které přišly v krátkém období po nich, jsou v současnosti nahlíženy spíše jako experiment nebo vtip, některé z nich působily škody pouze díky chybě v kódu, nikoliv účelově, jako je tomu dnes.

První známé malware se šířily přes floppy diskety, například zapsáním na bootací sektor, do dalšího počítače se následně dostaly při vložení diskety do mechaniky. V současnosti se malware šíří přes internet, ať už autonomně (například počítačoví červi), nebo skrytě, pod hlavičku jiného legitimního softwaru (například trojští koně).

V dnešní době může malware napadnout kromě počítačů i mobilní telefony a jiná chytrá zařízení. S příchodem bitcoinu se rozšířila i služba nazvaná Malware as a Service (MaaS). MaaS je byznys na černém trhu, kde je možné si koupit nějaký celý funkční malware nebo části kódu. Pro použití takto získaného malwaru nemusí být majitel žádným expertem.

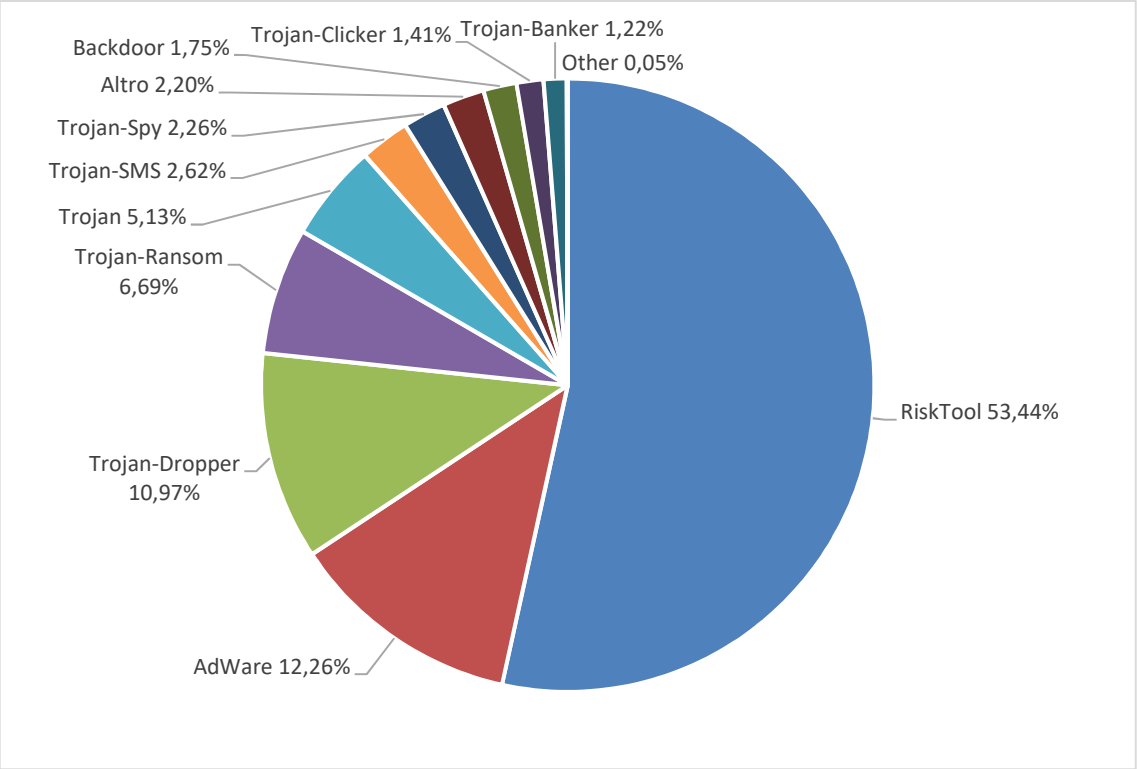
Historie čerpána z [4] a [5].

1.2 Co je to malware?

Malware, zkrácený název z Malicious Software, je program s jedním jediným účelem, poškodit svoji oběť, ať už poškozením dat v počítači nebo vydíráním peněz prostřednictvím zašifrovaného pevného disku. Co malware může v nakaženém počítači způsobit a jak se šíří, záleží na jeho typu.

Na grafu (Obr. 1.1) podle webového serveru Securelist můžeme vidět četnost jednotlivých typů malwaru na mobilních zařízeních ve třetím čtvrtletí roku 2017. Nejznámější typy – trojský kůň, počítačový červ, Backdoor, Adware a Spyware – jsou probrány níže.

RiskTool, také nazývaný Riskware, je nejpoužívanější na mobilních zařízeních, ale není ve své podstatě malware. Riskware je legitimní aplikace, na kterou může být použit exploit pro získání neoprávněného přístupu. Exploit je speciální program, data, nebo sekvence příkazů, které využívají zranitelnosti aplikace. Tou je chyba ve zdrojovém kódu aplikace, která způsobí nezamýšlenou činnost a umožňuje získaní přístupu k datům nebo uživatelskému účtu, k čemuž bychom se legitimním způsobem nedostali.



Obr. 1.1 Četnost malware podle typu. [7]

1.3 Typy malware

1.3.1 Trojský kůň

Název typu malwaru trojský kůň pochází z řecké mytologie, z příběhu o dobytí Tróje, kdy Řekové proniknou přes opevnění Tróje pomocí dřevěného koně, v němž jsou ukryti řečtí válečnici, kteří v noci město dobyli. Tento příběh zmiňuji, protože činnost malwaru, nesoucího toto jméno, je velice podobná.

Trojský kůň je uživateli skrytá část programu, která se do počítače dostane přes jinou, neškodně vypadající aplikaci – například instalační balíček aplikace, stažený z neoficiálních stránek. Po instalaci a spuštění uživatelé ve většině případů spustí to, co si nainstalovali, ale v pozadí také spustí pravý účel daného trojského koně, což může být například otevření backdooru pro hackery, těžení bitcoinů nebo jiná nežádoucí činnost.

Rozdíl mezi trojským koněm a počítačovým virem je ve způsobu šíření. Trojský kůň oproti viru neinfikuje další programy a také neinfikuje další počítače jako červ. Existují však červi, kteří do napadeného počítače trojské koně instalují nebo z nainstalovaných programů trojské koně vytvářejí.

1.3.2 Ransomware

Ransomware je vyděračský software, který zablokuje infikovaný systém a požaduje po uživateli výkupné, za které dostane kód pro odblokování. Nejčastější typ zablokování je zašifrování celého pevného disku symetrickou šifrou.

1.3.3 Červ

Počítačový červ je program, který se sám dokáže rozkopírovat na další počítače. Tohoto rozkopírováni je docíleno tak, že červ po infikaci počítače přebere kontrolu nad síťovými prostředky a začne se rozesílat na další počítače. Aby červi mohli infikovat ostatní počítače, tak k tomu využívají zranitelnosti operačních systémů nebo aplikací. Červi v sobě obsahují tzv. Payload, neboli samotný kód, který bude nějak škodit danému systému, např. omezením funkčnosti, zašifrováním obsahu (Ransomware), zpřístupněním backdooru pro hackery, shromažďováním citlivých údajů. Zvláštní skupinou jsou červi, kteří z infikovaných počítačů vytváří botnety, které jsou detailněji popsány dále.

1.3.4 Backdoor

Backdoor je metoda, umožňující hackerům snadnější přístup do nakaženého počítače. Spekuluje se, že původní myšlenkou backdooru měla být technická podpora a backdoor byl na zařízení záměrně nainstalován výrobcem. Bohužel backdoor padl do rukou hackerům a ti získali nástroj, jak se velmi jednoduše dostat do systému.

1.3.5 Spyware

Spyware je špionážní software, který odesílá informace o uživateli bez jeho vědomí. Jaké informace o uživateli se posílají, záleží na typu spywaru – ať už jsou to jednotlivé stisknuté znaky na klávesnici, nebo citlivé informace nějaké firmy.

1.3.6 Adware

Adware je reklamní software, který znepříjemňuje život otravnými reklamami. Oproti spyware nemá přímo za účel škodit napadenému systému. Adware může mít různou úroveň agresivity – od bannerů po neustále vyskakovací pop-up okna, nebo změnu domovské stránky v prohlížeči bez schválení uživatele.

2 BOTNET

Botnet je síť počítačů infikovaných speciálním softwarem, který je řízen z kontrolního počítače. Tento speciální software se do počítače dostane například prostřednictvím červa nebo trojského koně a uživatel ani neví, že se jeho počítač stal součástí botnetu. Takovému počítači se přezdívá „zombie“. Pokud kontrolní počítač vyšle do sítě botnetu příkaz na spam nebo DDoS útok, všechny infikované zombie počítače, které jsou součástí daného botnetu, začnou tento příkaz automaticky vykonávat.

Podle způsobu komunikace řídícího počítače s boty můžeme botnet rozdělit na distribuovaný (síť peer-to-peer) a síť Tor.

V peer-to-peer síti boti komunikuji s dalšími nejbližšími boty a přeposílají si instrukce. V této síti neexistuje centrální bod, odkud by chodily příkazy, takže je může poslat kdokoliv, kdo se nachází v této síti. Nevýhodou sítě peer-to-peer je možnost izolace botů od sebe, čímž by se mohli oddělit od původního botnetu a vytvořit nový. Další nevýhodou je možnost zaslání falešných instrukcí, které by boti vykonali, protože nemají možnost zjistit, zda jsou instrukce pravé.

Dalším typem je komunikace přes síť Tor. Tor je zašifrovaná síť navržená za účelem zajistit uživatelům anonymitu. Bot, připojující se v Tor síti ke skryté službě, ze které by přicházely příkazy, by byl těžký na oklamání a izolování, stejně jako v případě peer-to-peer.

DDoS (zkratka pro Distibuted Denial of Service) je útok, kdy zombie z botnetu zahltí server, firewall na směrovači nebo směrovač samotný, a znemožní legitimním uživatelům přístup k tomuto serveru; v nejhorším případě server, směrovač nebo firewall vyřadí z provozu. Pokud by útok směřoval pouze z jednoho počítače, jednalo by se o DoS (Denial of Service), ale dopad může být stejný.

S rozšířením IoT (zkratka pro Internet of Things) začali tvůrci malwaru, který vytváří ze zařízení zombie pro botnet, upravovat tento malware tak, aby mohl infikovat i IoT zařízení, jako jsou routery, IP kamery atd. Příkladem botnetu, využívajícího tyto IoT zařízení, je Mirai botnet. Mirai botnet se zaměřuje na zařízení připojená k internetu a používající operační systém Linux. Zombie z tohoto botnetu byly využity pro některé z největších DDoS útoků. Podle poskytovatele Akamai měl nejsilnější DDoS útok z Mirai botnetu sílu 623 Gbps.

3 HONEYPOT

Honeypot můžeme zařadit do bezpečnostního balíčku s ostatními bezpečnostními prvky typu IDS, IPS a SIEM, proto si tyto pojmy vysvětlíme, než přejdeme k samotnému honeypotu.

IDS – Intrusion Detection System – je obranný systém sloužící k detekci neobvyklého chování v síti nebo porušení bezpečnostní politiky. Rozdíl mezi IDS a firewallem spočívá v tom, že firewall se snaží filtrací zabránit útoku a případně blokuje všechny pakety, které neodpovídají předem stanoveným pravidlům. IDS analyzuje provoz a na základě signatury nebo anomálie informuje upozorněním, např. alertem, do systému SIEM. Pokud má funkcionalitu IDPS (Intrusion Detection and Prevention System), upraví svá pravidla tak, aby se pakety, které byly vyhodnoceny jako hrozba, nedostaly do sítě.

IDS můžeme rozdělit podle místa detekce na NIDS (Network Intrusion Detection Systems) a HIDS (Host Intrusion Detection Systems), nebo podle použité detekční metody na Signature-based IDS a Anomaly-based IDS.

NIDS je umístěn na strategické místo v síti nebo na místa, kde může monitorovat provoz všech zařízení.

HIDS je spouštěn na jednotlivých hostech nebo zařízeních v síti, kde monitoruje pouze příchozí a odchozí provoz zařízení.

Signature-based IDS používá k detekci specifické vzory, například byte sekvence v síťovém provozu nebo známé sekvence škodlivých instrukcí, používané malwarem.

Hlavní použití Anomaly-based IDS je pro detekci neznámých, například nových, útoků, které se projevují nezmapovaným chováním a je obtížné je detekovat přes detekci signatur.

IPS – Intrusion Prevention System – je spíše znám jako IDPS, protože tento systém v sobě má zabudován IDS současně se systémem zabraňujícím útoku (IPS).

SIEM – Security Information and Event Management – je software a služba, která v reálném čase analyzuje bezpečnostní alerty generované z aplikací a síťových zařízení.

Síla honeypotu se skrývá v detekci neznámých útoků, což je rozdíl oproti firewallu a IDS, které se hlavně zaměřují na obranu proti známým útokům. Například webový honeypot může sloužit ke hledání webových zranitelností aplikací nebo webových stránek.

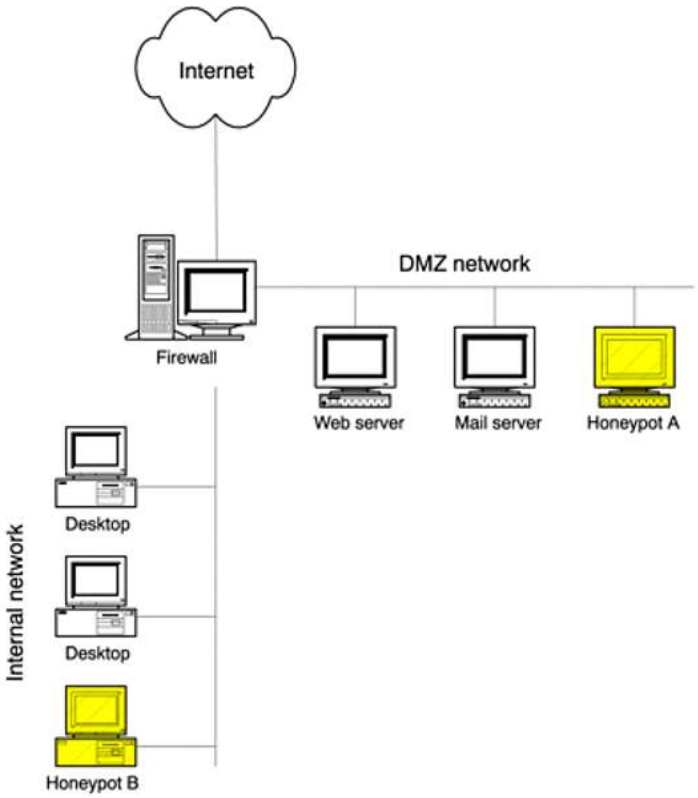
3.1 Definice honeypotu

A co to tedy honeypot je? Přesnou definici asi nenajdeme, každý chápe tuto technologii trochu jinak. Dle mého názoru nejlepší definice podle [1] zní:

Honeypot je bezpečnostní prostředek, jehož hodnota spočívá v tom, že je sondován, napaden nebo kompromitován.

K této definici bych pouze podotknul, že honeypot je v nejčastějších případech softwarové řešení, proto se instaluje na zařízení, na které nesmí přijít legitimní provoz, jinak by honeypot generoval hlavně falešné alerty, protože jakékoliv připojení učiněné s honeypotem může být bráno jako hrozba. Honeypot přichází na scénu, když bude útočník skenovat síť, aby zjistil, co se v síti nachází. Skenováním sítě si útočník zjistí, jaké služby (otevřené porty) a jaký operační systém se na daném koncovém uzlu nachází.

Pokud by na honeypot přišel sken sítě nebo by tento sken byl zachycen jinými bezpečnostními prvky v síti, jedná se nejpravděpodobněji o útočníka, který si „oťukává“ svou další oběť a můžeme provoz od útočníka nastavit tak, aby byl přesměrováván na honeypot. Proto by honeypot neměl být jediný obranný prostředek k zabezpečení sítě, ale měl by být následnou vrstvou za dalšími bezpečnostními systémy.



Obr. 3.1: Příklad umístění honeypotu v síti. [1]

3.2 Rozdělení honeypotů

Samozřejmě neexistuje jen jeden honeypot, máme dokonce dvojí rozdělení: podle úrovně interakce s útočníkem a podle účelu honeypotu. Podle úrovně interakce honeypot rozdělujeme na Low-Interaction, Medium-Interaction a High-Interaction honeypoty. Podle účelu pak honeypoty rozdělujeme na Production a Research honeypoty.

3.2.1 Podle úrovně interakce

3.2.1.1 Low-Interaction honeypot

Low-Interaction honeypoty pouze simulují služby, které jsou běžně používány, například ftp pro přenos souborů, http pro přenos webových stránek, ssh pro vzdálené přihlášení a smtp pro e-mailové služby. Tyto služby jsou simulovány pouze v malém měřítku – například základní webová stránka bez možnosti interakce, login zpráva bez možnosti úspěšného přihlášení pro logování Brute Force útoku nebo falešný soubor s hesly. Brute Force útok spočívá ve zkoušení všech možných kombinací hesla, u dlouhých hesel je tento útok velmi časově náročný kvůli velkému počtu kombinací. Malé měřítko simulovaných služeb u Low-Interaction honeypotu dovoluje emulaci více služeb zároveň a umožňuje snadnou instalaci, konfiguraci, spuštění a údržbu. Současně obnáší malé riziko, že se útočník dostane do systému, na kterém honeypot běží a tím by mohl nad systémem získat kontrolu a způsobit vyšší škodu. Malé měřítko je zároveň i nevýhodou Low-Interaction honeypotů, protože útočník rychleji zjistí, že je připojený k honeypotu a útok přeruší.

3.2.1.2 Medium-Interaction honeypot

Medium-Interaction honeypoty dovolují větší interakci než Low-Interaction, neboť u této úrovně simulujeme celou aplikaci, například Apache Web Server nebo Microsoft ISS Web, což jsou aplikace pro vytvoření http serveru. Instalací s sebou tyto aplikace přinášejí i funkcionality, které bychom u simulace http služby u Low-Interaction honeypotu nezískali. Simulací aplikace můžeme chytit červa, který hledá určitou zranitelnost uvnitř aplikace webového serveru, což by u emulované služby nemusel najít ([1]). Simulováním aplikace se zvyšuje riziko průniku do systému, protože útočníkům dáváme k dispozici celé aplikace, naopak můžeme získat více informací z útoku, například payload u červa nebo aktivitu útočníka. Další nevýhodou je zvýšená údržba oproti Low-Interaction, protože se jedná o aplikace, ke kterým mohou vycházet nové aktualizace a tím se zvyšuje i zranitelnost.

3.2.1.3 High-Interaction honeypot

High-Interaction honeypoty monitorují celý operační systém. Monitorovat můžeme změny souborů, registrů a změny systémových procesů. Protože má útočník k dispozici celý operační systém, nejen emulované aplikace jako v případě Medium-Interaction honeypotu, může útočník využít exploit pro získání přístupu do systému. Monitorování umožní získání tohoto exploitu pro zjištění, jak se útočník do systému dostal. Nevýhodou High-Interaction honeypotů je jejich náročná údržba a velké riziko, protože dáváme útočníkovi k dispozici kompletní operační systém. Tím může získat kontrolu nad operačním systémem pro provedení jiného útoku, buď do sítě, ve které se honeypot nachází nebo v horším případě, útok do cizí sítě, což je hlavní právní problém spojený s využitím High-Interaction honeypotu.

3.2.2 Podle účelu honeypotu

3.2.2.1 Production honeypot

Production honeypoty hlídají vnitřní sítě, aby zvýšily jejich celkovou bezpečnost. Vnitřní síť může také obsahovat tzv. demilitarizovanou zónu, což je část sítě, kde se nachází servery přístupné z internetu. Production honeypoty jsou jednodušší na použití a většinou používají Low-Interaction honeypoty. Jejich úkolem je pouze útok zjistit a upozornit příslušnou osobu nebo další bezpečnostní systémy.

3.2.2.2 Research honeypot

Research honeypoty mají za hlavní úkol zkoumat motivy a prostředky používané „black hat“ komunitou, proto využívají hlavně High-Interaction honeypoty. Díky tomu je možné objevit nové zranitelnosti a typy exploitů používaných útočníky. Vzhledem k popularitě honeypotů se také objevily honeypoty, sloužící pro analýzu malware.

U některých High-Interaction honeypotů je možné například vidět, které soubory jsou měněny a jak jsou měněny, a díky tomu můžeme v kontrolovaném prostředí honeypotu zkoumat útoky.

3.3 Příklady honeypotů

Popsat všechny typy honeypotů, které se vyskytují na trhu je téma na několik knih, proto se zaměříme jen na některé, níže uvedené typy honeypotů. V žádném případě nelze říct, zda tento nebo tento honeypot je nejlepší; nejlepší honeypot je vždy ten, který se nejvíce hodí pro použití v dané situaci. Toto téma rozdělíme podle úrovně interakce.

3.3.1 Příklad Low-Interaction honeypotu – HoneyD

HoneyD vytváří na síti virtuální klienty, kteří mohou být nakonfigurováni k běhu libovolných služeb a nastavení odpovědí ze služeb, jež budou zasílány útočníkovi, čímž můžeme zaručit jedinečnost systému. HoneyD byl vytvořen a spravován Nielsem Provosem z Michiganské univerzity, poprvé byl uveden v dubnu 2002. HoneyD je určen pro Unix platformu a distribuován pod licencí OpenSource, díky čemuž je možné jej používat zdarma a jsou přístupné zdrojové kódy, které je možné dále upravovat. Díky OpenSource získává uživatel také možnost upravovat simulované služby dle vlastní potřeby, což útočníkovi ztíží identifikaci honeypotu. Současně je umožněno nastavit pouze ty porty, které budou monitorovány.

HoneyD oproti ostatním honeypotům umožňuje detekci i těch útoků, které nejdou přímo na jeho vlastní IP adresu. Docílí toho tím, že pokud se útočník pokusí připojit k systému, který neexistuje (tedy cílová adresa není přidělena žádnému koncovému uzlu), HoneyD přijme požadavek na připojení a předstírá identitu cílového systému a odpovídá útočníkovi na zprávy. Toho HoneyD docílí díky tomu, že může monitorovat miliony neexistujících IP adres pro připojení, současně převzít IP adresy tisíců koncových uzlů a aktivně komunikovat s útočníkem. Tato schopnost má své vlastní uplatnění nazvané „Blackholing“. Blackholing používá koncept honeypotu a aplikuje ho na celé sítě. Účelem Blackholingu není identifikace jediného útoku, ale identifikace trendů, nebo zkoumání aktivity uvnitř internetu.

Další funkcí HoneyD je možnost simulace různých systémů současně a to dokonce na IP úrovni. Znamená to, že pokud přijde sken sítě, útočník dostane odpověď, že se na koncovém uzlu nachází takový operační systém, jako je emulovaný. Tato funkce byla u honeypotu Specter emulována na vyšší vrstvě a útočník mohl ze skenu zjistit 2 různé operační systémy. Specter také umožňoval emulaci několika operačních systémů, ale nikoliv současně jako HoneyD.

3.3.2 Příklad Medium-Interaction honeypotu – Cowrie

Cowrie je SSH a Telnet honeypot vytvořený za účelem logování Brute Force útoků a útočníkovy interakce s SSH nebo Telnet. Cowrie obsahuje plnohodnotný falešný souborový systém připomínající Debian 5, ve kterém lze přidávat, popřípadě odebírat soubory. Cowrie ukládá soubory, které jsou staženy přes wget nebo curl, popřípadě nahrány přes SFTP nebo scp. Cowrie dovoluje přeposlání SMTP připojení na SMTP honeypot.

Wget a curl slouží pro přenos souborů, používají protokoly http, https, ftp, ftps a další.

SFTP je zkratka pro SSH File Transfer Protocol a označuje protokol a zároveň i program pro zabezpečený přenos souborů.

Scp je protokol pro přenos souborů mezi lokálním a vzdáleným hostem.

3.3.3 Příklad High-Interaction honeypotu – Capture-HPC

Capture-HPC je Client honeypot, jinak také Honeyclient. Client honeypot je bezpečnostní technologie, která dovoluje hledání škodlivých serverů na síti. Capture-HPC detekuje škodlivé servery pomocí interakce s potenciálními škodlivými servery použitím dedikovaných virtuálních strojů a sledováním změn systémových stavů. Pokud je detekována změna systémového stavu, a protože nebyla žádná další aktivita u dedikovaného klienta, je server, se kterým Capture-HPC komunikoval, označen jako škodlivý.

Capture-HPC automaticky sbírá malware, který mohl být uložen na kompromitovaný klientský systém. Capture-HPC dokáže monitorovat souborový systém, registry a systémové procesy na kernel úrovni. Centralizované logy umožňují mít přehled, které stránky byly honeypotem navštíveny, jednotlivé klasifikace serverů a změny systémových stavů při navštívení škodlivých serverů.

3.4 Honeypoty pro sítě SCADA

3.4.1 Co je SCADA?

SCADA, plným názvem Supervisory Control And Data Acquisition, je obvykle software, který z centrálního počítače monitoruje průmyslová a jiná technická zařízení a procesy a umožňuje jejich ovládání.

3.4.2 Příklad honeypotu pro sítě SCADA – Conpot

Conpot je Low-Interaction ICS honeypot, který byl navržen pro snadné nasazení, modifikaci a rozšíření. ICS, zkratka Industrial Control Systems, je souhrnný název pro různé typy kontrolních systémů a příslušné instrumentace a obsahující zařízení, systémy a další, které jsou používány k ovládání automatizovaných procesů ([9]).

Cílem Conpotu je získávat informace o motivech a metodách útočníků, jejichž cílem je ICS. Nabízí rozsah standardních industriálních kontrolních protokolů umožňujících emulaci komplexních infrastruktur, aby přesvědčil útočníka, že se naboural do obrovského komplexu. Conpot je pod Open Source licencí a je k dispozici na stránkách Github ke stažení. Popis Conpotu převzat z oficiálních stránek na [10].

4 PRAKTICKÁ REALIZACE HONEYPOTU

4.1 Úvod

Pro praktickou realizaci jsme požadovali honeypoty pracující s protokoly SSH, Telnet a SMTP. Dále pak odesílání statistik z honeypotů do databáze a zobrazení těchto dat v grafech. Přes protokoly SSH, Telnet a SMTP mohou útočníci nahrávat, resp. stahovat, soubory infikované malwarem a proto budeme chtít tyto soubory posílat na analýzu. Nakonec jsme se rozhodli vytvořit automatizované nasazení použitých komponent, protože budeme chtít nasadit více strojů, na kterých budou spuštěny honeypoty a bylo by časově nerozumné procházet stroj po stroji a měnit jeden řádek v konfiguračním souboru.

Honeypoty pracující s protokoly SSH a Telnet jsou:

- **Kippo** ([12]) - medium interaction SSH honeypot, který je navržen hlavně pro logování interakce útočníka s honeypotem a také pro logování tzv. „brute force“ útoků, kdy útočník zkouší veškeré možné kombinace hesla.
- **Cowrie** ([13]) - založen na Kippo a umožňuje kromě SSH také Telnet. Dále je přidána podpora nahrávání souborů SFTP a SCP, přepojování (ssh proxying) např. na další honeypoty a logování ve formátu JSON.
- **Hornet** ([14]) - medium interaction SSH honeypot, který podporuje více virtuálních strojů, kde každý virtuální stroj může být nakonfigurován samostatně a s vlastním operačním systémem.

Z těchto honeypotů jsme si vybrali Cowrie, protože je nejpopulárnější, umožňuje šest možností odesílání dat do databáze a tři možnosti odesílání souborů na analýzu.

Honeypoty pracující s protokolem SMTP jsou:

- **Mailoney** ([15]) - pracuje s moduly, které umožňují logování pokusů o přeposílání e-mailových zpráv.
- **SHIVA** ([16]) - zkratka pro Spam Honeypot with Intelligent Virtual Analyzer – je Spam honeypot, který sbírá a analyzuje přijatý spam.

Zde jsme si zvolili honeypot Mailoney, protože má jednodušší strukturu a stejně jako Cowrie je napsán v programovacím jazyce Python.

Protože Mailoney neumožňuje odesílání dat do databáze, použijeme pro tuto činnost Cowrie, kde máme k dispozici následující varianty:

- **MySQL**
- **Rethinkdb**
- **SQLite3**
- **MongoDB**
- **Splunk**

- **HPFeeds**

Z těchto možností jsme měli zkušenost pouze s databázi MySQL. Nejprve se podívejme na nástroje, které umí zobrazit získaná data. Zde máme tyto možnosti:

- **Grafana**
- **Kibana**

Vybrali jsme si Grafana, neboť je schopna pracovat s více eventualitami zdrojů dat, a protože komunita vytváří pluginy, umožňující použití např. nových zdrojů dat nebo nových panelů pro reprezentaci dat. V základu má Grafana tyto zdroje dat:

- balíček **Graphite**
- **Elasticsearch** server
- **Prometheus** server
- **MySQL** databáze
- **InfluxDB** databáze
- **PostgreSQL** databáze
- **OpenTSBD** databáze
- službu **Amazon CloudWatch**

Grafana obsahuje v základu jako zdroj dat databázi MySQL. Nakonec jsme se ale jako zdroj dat rozhodli použít balíček Graphite, resp. Graphite a Carbon, protože má jednodušší formát přijímaných dat. Dalším důvodem je, že do Carbon, tedy příjemce statistik pro uložení do databáze, se nemusíme přihlašovat a data lze poslat pomocí TCP nebo UDP paketů. Oba tyto důvody nám umožní jednodušší řešení pro odesílání dat z Mailoney a Cowrie.

Carbon vyžaduje, aby příchozí data měla časové razítko. Při přijetí dat se stejným jménem a časovým razítkem by tedy byla data přepsána. Pro agreagaci dat jsme si vybrali Statsite pro jeho jednoduchý způsob zápisu statistik. Dále není potřeba přidávat časové razítko k jednotlivým paketům, protože Statsite přidá časové razítko k agregovaným datům, které posílá po daném časovém intervalu do databáze. Časový interval pro odesílání dat ze Statsite na Carbon budeme mít nastaven na 5 minut, protože nepoužívá tak kritická data, jako například odezvu serveru, abychom museli data posílat po např. 1 min, aby byla ještě relevantní.

Pro odesílání nebo uložení souborů na analýzu máme z Cowrie tyto možnosti:

- **VirusTotal**, kde soubor projde antivirovými testy od všech známých antivirových programů, jako Avast, ESET, Kaspersky atd., a můžeme tak zjistit, jestli neobsahuje nějaký známý typ malwaru. Tato varianta předpokládá, že antivirové programy dokáží malware v souboru odhalit, což může být problém u nových druhů malwaru.

- **Cuckoo** je tzv. sandbox, kde se v kontrolovaném a monitorovaném prostředí soubor spustí a analyzuje se jeho chování. Jestliže použijeme na analýzu škodlivých souborů jiný nástroj než Cuckoo, nemusely by nástroje být schopné zpracovat přijímaný formát dat z Cowrie.
- **Amazon S3** (nebo kompatibilní „bucket“), kde S3 je placené cloudové uložiště firmy Amazon pro uložení souborů.

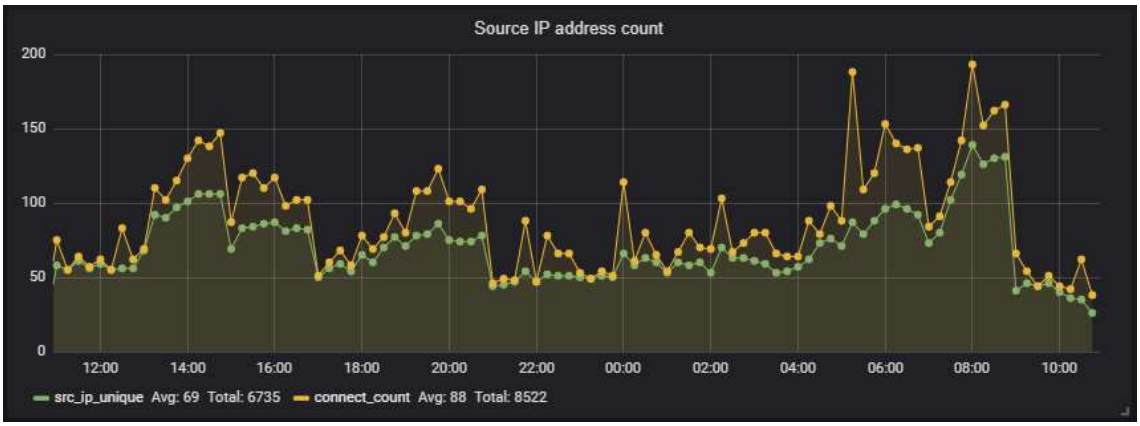
Protože jsme na této bakalářské práci spolupracovali s firmou Avast, která má již vytvořeno prostředí pro analýzu malwaru, ale pro odesílání souborů používá protokol FTP, budeme soubory z Cowrie posílat přes tento protokol.

Pro automatizované nasazení použitých komponent použijeme software Puppet. Stroje, které budou pod jeho správou, budeme rozlišovat na honeypoty (Cowrie, Mailoney + Statsite) a tzv. „backend“ (Carbon, Graphite a Grafana). Důvod nasazení Statsite na stejný stroj jako honeypoty je ten, že v Carbon můžeme oproti Statsite vytvořit tzv. „Whitelist“, tedy filtr IP adres pro příjem dat. Zmíněný Whitelist bychom mohli vytvořit také pro Statsite přes „iptables“ (firewall u distribucí Linux). Výhoda Carbon spočívá v tom, že realizuje Whitelist pomocí jednoho souboru, který obsahuje povolené IP adresy, naproti tomu řešení v „iptables“ by vyžadovalo minimálně jedno pravidlo pro jednu IP adresu.

4.1.1 Grafana

Grafana ([17]) je analytický a monitorovací software. Data si zde zobrazujeme pomocí grafů, tabulek a dalších panelů. Grafana je open source řešení, kde komunita může vytvářet podporu pro další zdroje dat a panely pro zobrazení dat.

Pro zobrazení dat jsou k dispozici tzv. „panely“, což může být graf, tabulka, „Alert List“, „Singlestat“ nebo „Heatmap“. Grafy a tabulky jsou ve své podstatě obdobou tabulek a grafů např. u Microsoft Excel či OpenOffice/LibreOffice Calc. Příklad panelu s grafem je na obrázku 4.1. Grafana podporuje tzv. „Alerty“, což jsou oznámení o neobvyklé aktivitě, např. o nedostupnosti serveru nebo jeho velké odezvě. Alerty si můžeme nastavit na jednotlivých grafech společně s definicí, kdy mají hlášení poslat. Panel „Alert List“ je tabulka varování z ostatních panelů, což mohou být např. varování o velkém počtu připojení do aplikace (možný DoS útok). Panel „Singlestat“ zobrazuje (jak z názvu vyplývá) aktuální hodnotu jedné statistiky, s možností zobrazení historie jejího vývoje na pozadí (formou grafu, ale nelze z něj číst hodnoty). Panel „Heatmap“ zobrazuje histogram v čase, kde histogram je grafická reprezentace distribuce číselných hodnot.



Obr. 4.1 Grafana, Cowrie - graf počtu připojení (žlutá) a graf počtu unikátních IP adres (zelená).

4.1.2 Graphite a Carbon

Z balíčku Graphite ([18]) využijeme požadované komponenty „graphite-web“ a „carbon-cache“ (Obr. 4.2). Komponenta „graphite-web“ je webové rozhraní, které nám umožňuje zobrazit statistiky z carbon-cache pouze pomocí grafů. Oproti Grafana nejsou grafy propracované a komponenta nepodporuje tvorbu dashboardů. K zobrazení (a přístupu z Grafana) musíme nainstalovat aplikaci Apache a povolit příslušný konfigurační soubor, který si načte potřebné součásti ze souborů komponenty.

Druhou komponentou je „carbon-cache“ (v systémovém repozitáři nazváno graphite-carbon, protože obsahuje i carbon-aggregator a carbon-relay), která se

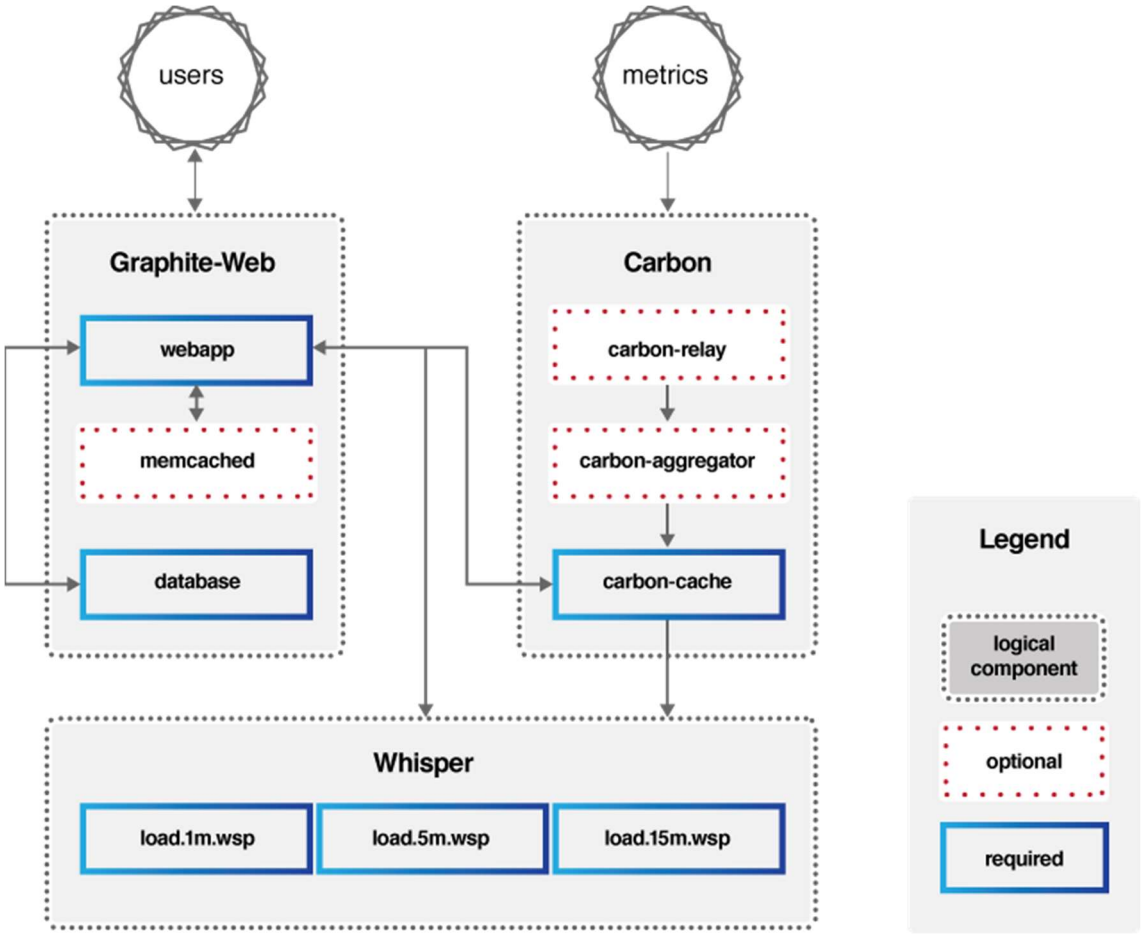
stará o příjem statistik a ukládá je do databáze. Formát pro odesílání statistik do Carbon je:

<metric path> <metric value> <metric timestamp>

kde:

- **metric path** je název statistiky
- **metric value** je hodnota dané statistiky, tedy číslo nebo řetězec znaků
- **metric timestamp** je časové razítko, kdy byla data odeslána ze zdroje.

Pro uložení přijatých dat máme k dispozici databáze Whisper a Ceres.



Obr. 4.2 Struktura Graphite. [8]

Databáze Whisper ([19]) ukládá statistiky do souborů a složek přímo na disk (například MySQL ukládá tabulky do jednoho souboru), kde mohou být roztříděny do podsložek – při odesílání statistik odděleno tečkou, např.:

cowrie.login.user.root → cowrie/login/user/root.wsp

Poslední položka v řetězci bude název souboru – „root.wsp“ ve složce „user/“. V dané složce je pak jeden soubor (s příponou .wsp) přidělen ke stejně nazvaným datům dané statistiky;

příklad:

cowrie/login/user/root.wsp

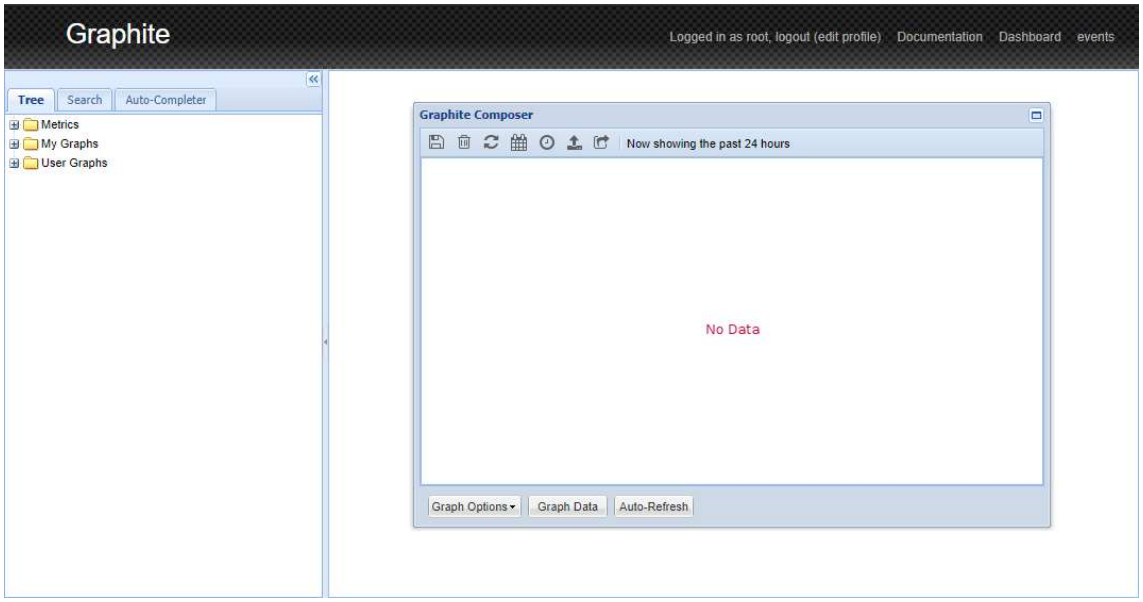
obsahuje data spojená s touto hodnotou (např. počet zadání při přihlášení do honeypotu za určitou dobu).

Databáze Ceres ([20]) měla být náhradou databáze Whisper, avšak její vývoj není momentálně aktivní.

Graphite (komponenta „graphite-web“) vyžaduje aplikaci Apache (webový server), jakožto způsob zobrazení webového rozhraní. Grafana již v sobě obsahuje obdobu Apache pro umožnění vzdáleného přístupu k webovému rozhraní. Po instalaci komponenty „graphite-web“ se ve složce:

/etc/apache2/sites-available/

vytvoří soubor „apache2-graphite.conf“. Tento soubor je potřeba v Apache aktivovat, abychom ke Graphite mohli přistupovat vzdáleně z webového prohlížeče (Obr. 4.3). Toho docílíme pomocí příkazu pro vypnutí defaultní stránky a příkazu pro zapnutí stránky s Graphite.



Obr. 4.3 Webové rozhraní Graphite.

4.1.3 Statsite

Agregační server „Statsite“ ([21]) seskupuje přijatá data a přes výstupní modul, tzv. „sink“ (v našem případě „sink“ pro Carbon) posílá data v definovaných intervalech (dle nastavení v konfiguračním souboru) do zvoleného programu, resp. na adresu a port, na kterých zvolený program přijímá data. Statsite server přijímá statistiky ve formátu „statsd“ ([22]):

klic:hodnota|typ

kde ***klic*** je jméno statistiky (klíče jsou voleny uživatelem).

Možné typy statistik:

- **Counter** (značeno: „c“) neboli čítač, přičítá při přijetí paketu zadanou hodnotu k hodnotě uložené v paměti serveru, hodnotou může být jakékoliv kladné či záporné číslo.
- **Unique Set** (značeno: „s“) spočítá počet unikátních hodnot z přijatých dat
- **Gauge** (značeno: „g“) je na začátku inicializován na danou hodnotu a tato hodnota je následně upravována, např. přičítáním nebo odečítáním
- **Key/Value** (značeno: „kv“) přiřazuje k danému klíči danou hodnotu
- **Timer** (značeno: „ms“) počítá čas odpovědi aplikace

Z těchto typů jsme využili pouze Counter a Unique Set. Counter například pro statistiku počtu připojení a Unique Set například pro počet unikátních IP adres.

Data jsou odesílána na server přes paket UDP (User Datagram Protocol) standardně na port 8125 (port lze změnit v konfiguračním souboru). Pomocí jednoho UDP paketu lze zaslat i více statistik zároveň, každá statistika musí být oddělena „\n“, neboli novým řádkem. Při posílání paketů je nutné dbát na limit maximální velikosti jednoho paketu – 1500 bytů.

4.1.4 Mailoney

Mailoney ([15]) je SMTP honeypot, který obsahuje tři moduly určující jeho funkčnost. Prvním modulem je „open_relay“, který loguje pokusy o odeslání zpráv. Druhým modulem je „postfix_creds“, tento modul loguje přihlašovací údaje z pokusů o přihlášení. Posledním modulem je „schizo_open_relay“, propracovanější možnost oproti modulu „open_relay“. Tento modul je připraven pro funkční možnost přeposílat e-maily k příjemcům, která ale není implementována.

4.1.5 Puppet

Pro automatizované nasazení (resp. instalaci a konfiguraci) těchto aplikací využijeme software Puppet ([23]), který také umožňuje udržování jednotných konfiguračních souborů pro tyto aplikace.

```
# Obsah na serveru, jméno klienta (druhá položka záznamu) je volitelné – puppetclient.*.*
192.168.1.101 puppetclient.xkarge.cz puppetclient

# Obsah na klientovi, jméno serveru (druhá položka záznamu) je volitelné – puppetmaster.*.*
192.168.1.100 puppetmaster.xkarge.cz puppetmaster puppet
```

Obr. 4.4 Obsah "/etc/hosts" na serveru a klientovi.

Puppet funguje na bázi klient-server, kde klienti zašlou na server své certifikáty. Pouze klienti, jejichž certifikát je serverem přijat, mohou obdržet tzv. „katalog“. V katalogu je popsáno, který tzv. „manifest“ má klient použít.

Manifest obsahuje úkony, které mohou např. pracovat se soubory, spouštět příkazy anebo instalovat balíčky ze systémového repozitáře. Klienti si v určitých intervalech u serveru kontrolují, zda není v katalogu něco nového, co by měli provést.

Adresu serveru klientům přidáme jako záznam do souboru:

/etc/hosts

(viz příklad Obr. 4.4), protože Puppet pracuje přes záznamy DNS a v konfiguračním souboru nelze nastavit adresu serveru přímo. K této adrese se klienti budou připojovat pro zaslání certifikátu a ke zjištění katalogu. V Obr. 4.4 je také na serveru přidána adresa klienta, abychom mohli při potvrzovaní rozlišit jednotlivé klienty.

```
# Třídy v tomto uzlu budou použity pro klienty, kteří nemají definovaný vlastní uzel
node default { }

# Šablona při přidání nového klienta
node 'puppetclient' {
    #include honeypot
    #include backend
}

# Tímto uzlem určíme, že na klientovi 'puppetclient.xkarge.cz' bude použita třída honeypot
# Jméno klienta můžeme případně zjistit při přijímaní jeho certifikátu
node 'puppetclient.xkarge.cz' {
    include honeypot
}
```

Obr. 4.5 Příklad obsahu souboru site.pp.

Pro určení, jaké manifesty mají jednotliví klienti používat, použijeme soubor „site.pp“. V našem případě bude soubor umístěn ve složce:

/etc/puppetlabs/code/environments/production/manifests

Server se do tohoto souboru dívá, když je dotázán klientem na zaslání katalogu. V souboru „site.pp“ mohou být obsaženy úkony, které mají klienti provést (příklad: některý z package/file/exec úseků v Obr. 4.6, Obr. 4.7 a Obr. 4.8), popř. použít funkci „include“ (viz Obr. 4.5). Funkce „include“ určuje třídu („class“), definovanou ve stejně pojmenovaném manifest souboru (také s příponou „.pp“ jako soubor „site.pp“), která má být aplikována na daném klientovi. Pro definování klientům, jaké třídy (resp. manifesty) mají použít, použijeme tzv. uzly („node“) s názvem klienta (Obr. 4.5, 3. řádek od konce).

```
# Zajištění, že budou nainstalovány aplikace v poli $packages
package { $packages:
        ensure => "installed",
        require => Exec['apt-update'],
}
```

Obr. 4.6 Puppet - Příklad použití funkce package.

V manifestech (souborech „.pp“) jsme nejčastěji použili bloky funkcí „package“ (Obr.4.6), „file“ (Obr.4.7) a „exec“ (Obr.4.8). Proto se na tyto funkce podíváme podrobněji, abychom věděli, co dělají a jaké parametry jsme použili. Další funkce jsou k nahlédnutí na oficiálních stránkách Puppet ([24]). Pokud je u funkce více než jeden parametr, musíme na konci atributu dát čárku pro oddělení atributů. Atribut „require“ má ve všech funkcích stejný úkol a říká, co musí být splněno, aby mohl být tento blok proveden. Například:

require => Exec['apt-update'],

říká, že musí být bez chyby proveden blok „Exec['apt-update'],“.

Funkce „package“ (Obr. 4.6) slouží k práci s balíčky, které můžeme nainstalovat pomocí standardních systémových repozitářů. Parametr „ensure“ určuje požadavky dané funkce. V případě Obr. 4.6. tedy:

ensure => "installed",

zajišťuje, že je balíček na klientovi nainstalován; pokud jej klient odinstaluje, při další kontrole klienta u serveru je tento balíček opět nainstalován.

```
# Přeposlání konfiguračního souboru Cowrie na klienta
file { 'cowrie-config':
        require => Exec['honeypot-install'],
        path => '/opt/cowrie/cowrie.cfg',
        ensure => file,
        source => 'puppet:///modules/honeypot/cowrie_config.cfg',
        owner => 'cowrie',
        mode => '644',
}
```

Obr. 4.7 Puppet - Příklad použití funkce file.

Funkce „file“ (Obr. 4.7) slouží k práci se soubory a složkami. Pomocí atributů lze tyto soubory a složky vytvářet, měnit oprávnění nebo vlastníka, anebo obsah souboru. Atribut „ensure“ určuje, zda se jedná o soubor či složku. Je-li tento atribut nastaven na hodnotu „file“ (popř. „present):

ensure => file,

říkáme tím, že se jedná o soubor, nikoliv složku. Pro složku nastavíme atribut na hodnotu „'directory'“:

ensure => 'directory',

Atribut „path“ určuje úplnou cestu k souboru či složce. Naproti tomu atribut „source“ určuje zdroj, odkud se má soubor stáhnout. Tento parametr může mít tři zdroje – soubor na klientovi, soubor z webového serveru a soubor z Puppet serveru. V případě souboru na klientovi („file:“) můžeme používat stejný zápis, jako u přímého přístupu na klientovi. U souboru z webového serveru („http:“) použijeme jeho webovou adresu (popř. jeho veřejnou IP adresu), se kterou může být soubor ze serveru stažen. Posledním zdrojem je soubor na Puppet serveru

(„puppet:“) - zde bude soubor uložen v příslušné složce se jménem modulu na serveru, v našem případě:
/etc/puppetlabs/code/environments/production/modules/(jméno)/files
Ve složce modulu musíme ještě vytvořit složku „files“, kam budou soubory nahrány; v odkazu u parametru tuto složku nezmiňujeme. Atribut „owner“ určuje, který uživatel bude vlastníkem souboru. Atribut „mode“ určuje oprávnění k souboru.

U funkce „file“ je nutné vytvářet podsložky v cestě k souboru, což Puppet neumí! Proto musíme cestu předem vytvořit pomocí skriptu, nebo musíme použít sekvenci („chain“) bloků funkce „file“, vytvářející cestu k souboru. Třetí variantou by bylo použití bloku „exec“ s příkazem pro vytváření složky a s parametrem pro vytvoření podsložek, pokud nejsou vytvořeny.

```
# Provedení příkazu v atributu „command“
exec { 'cowrie-restart':
        command => '/bin/systemctl restart cowrie',
        require => File['cowrie-config'],
        subscribe => File['cowrie-config'],
        refreshonly => true,
}
```

Obr. 4.8 Puppet - Příklad použití funkce exec.

Nakonec funkce „exec“ (Obr. 4.8) provádí spouštění spustitelných souborů (příkazů) na klientovi. Pro spuštění potřebujeme u parametru „command“ uvést úplnou cestu ke spustitelnému souboru a případně parametry ke spuštění daného souboru. Atribut „subscribe“ funguje v kooperaci s atributem „refreshonly“. Tyto atributy udávají podmínku, zda se má příkaz vykonat, jestliže je provedena změna u bloku v „subscribe“. Příkladem může být změna v konfiguračním souboru aplikace, kdy pro provedení změny je nutné aplikaci restartovat (příklad na Obr. 4.8).

4.2 Data z Honeypotů Cowrie a Mailoney

4.2.1 Data z Cowrie

Jak již bylo zmíněno, Cowrie patří mezi SSH a Telnet honeypoty. Oba protokoly vyžadují při připojení přihlášení pro získání přístupu. Poté lze zadávat příkazy, které mají být na stroji provedeny. Dále je možnost nahrávat, resp. stahovat soubory. SSH lze použít jako proxy pro připojení např. na webový server.

Důležitá data tedy jsou:

- **zdrojová IP adresa útočníka**, - zde vytvoříme statistiku počtu připojení z dané IP adresy a počet unikátních IP adres.

- **použitý protokol** pro připojení, a statistika počtu připojení přes daný protokol
- **přihlašovací údaje**, z nichž lze vytvořit seznam pro uživatele, kterých hesel by se měli vyvarovat.
- **nahrání škodlivého souboru nebo skriptu**
- pokusy o **použití SSH jako proxy**, tzv. „SSHTransport“

Nakonec vytvoříme statistiku počtu připojení na honeypot.

K zabránění vytvoření duplicitních souborů se využívá tzv. „hašovací funkce“ (anglicky: „hash function“). Tato funkce vytvoří z volitelně dlouhého řetězce znaků jeden výstupní řetězec o konstantní délce a jeho délka záleží na použité funkci. Cowrie pro hašování souborů využívá funkci „SHA-256“ (algoritmus SHA-2, [25]), která, jak napovídá název, vytváří výstupní řetězce o délkách 256 bitů. Pro reprezentaci řetězce se využívá hexadecimální soustava (viz 3. řádek u Obr. 4.11).

4.2.2 Data z Mailoney

Mailoney, jakožto SMTP honeypot, měl přes modul „schizo_open_relay“ pracovat jako e-mail „proxy“. Tím je míněno přeposílání e-mailů doručených do honeypotu k jejich příjemcům. Z této funkcionality jsme implementovali pouze její část a tou je příjem e-mailových zpráv.

Důležitými daty budou:

- **zdrojová IP adresa útočníka:** adresa, odkud přišel e-mail. Stejně jako u Cowrie vytvoříme statistiku počtu e-mailů z dané IP adresy a počet unikátních IP adres.
- **útočníkova e-mailová adresa**, kterou zadal jako odesílatele e-mailu. Statistika jako u IP adres: počet e-mailů z dané e-mailové adresy a počet unikátních zdrojových e-mailových adres.
- **e-mailové adresy příjemců**, kterým má být zpráva doručena. Zde také vytvoříme statistiku počtu e-mailů pro danou e-mailovou adresu a počet unikátních cílových e-mailových adres.

Oproti statistice počtu připojení v případě Cowrie, zde vytvoříme statistiku počtu e-mailů, které byly odeslány.

4.3 Vytvoření vlastního modulu v Cowrie

Protože budeme chtít odesílat data na server Statsite a Cowrie tuto funkci v základu neumožňuje, musíme si vytvořit vlastní modul, se kterým tohoto cíle dosáhneme.

Tvorby vlastních modulů v Cowrie pro odesílání dat je docíleno umístěním souboru s příponou .py (python skript) ve složce:

cowrie/cowrie/output

Pro správné fungování musí skript obsahovat načtení jiných tříd a definovaní potřebných metod, viz Obr. 4.9.

```
import cowrie.core.output
from cowrie.core.output import CONFIG
class Output(cowrie.core.output.Output):
    def __init__(self):
    def start(self):
    def stop(self):
    def write(self, event):
```

Obr. 4.9 Nutný obsah výstupního modulu Cowrie.

Dalším krokem je přidání příslušné položky do konfiguračního souboru Cowrie, aby ho bylo možné při restartování načíst. Toho docílíme jednoduchým vložením názvu nového modulu do hranatých závorek. Protože je modul ve složce „cowrie/cowrie/output“ musíme vložit „output_“ na začátek, např. „[output_file]“, přičemž soubor se bude jmenovat „file.py“. Vložení „output_“ v hranatých závorkách na začátek znamená, že Cowrie bude skript hledat ve složce output a jméno souboru již tuto část obsahovat nesmí. Vložení parametrů pro modul je pak docíleno jednoduchým zápisem:

parametr=hodnota

Tyto parametry můžeme následně načíst ve skriptu.

Metoda init slouží hlavně k načtení zmíněných parametrů z konfiguračního souboru a pro případné prvotní nastavení. Tyto parametry se načítají pomocí metody „CONFIG.get()“. Volání metody záleží na verzi Cowrie. Nové verze používají CONFIG.get (a potřebují jeho import z cowrie.core.output), starší verze pužívají cfg.get (naimportovaný v hlavičce metody init, viz Obr. 4.10). Zápis uvnitř volání metody je obdobný. Pro načtení parametru použijeme zápis v Obr. 4.10.

```
#Nový zápis:
def __init__(self):
    self.parametr = CONFIG.get("output_jmenomodulu",
"jmeno_parametru_v_konfiguracnim_souboru")

#Starý zápis:
def __init__(self,cfg):
    self.parametr = cfg.get("output_jmenomodulu", "jmeno_parametru_v_konfig._souboru")
```

Obr. 4.10 Zápis načtení parametru z konfiguračního souboru.

Metody start a stop jsou volány při spuštění a ukončení Cowrie. V těchto metodách může být například navázání, popř. ukončení spojení se serverem, na který se mají posílat data z Cowrie.

Poslední nutnou metodou, která je zároveň i nejdůležitější, je metoda write, která má jako jediná také parametr (proměnnou) event, resp. entry; obě mají

stejnou funkčnost. Tato metoda je volána pokaždé, když je zmíněný event vygenerován. Cílem této metody by mělo být zaslání zmíněných dat na server v takovém formátu, aby jím mohla být zpracována. Proměnná event, resp. entry, je pole hodnot, které má položky posledního vytvořeného JSON záznamu (viz příklad na Obr. 4.11) při zavolání metody write. K jednotlivým položkám pole event se přistupuje typem

klíč → hodnota

př. „event["eventid"]“, vrácenou hodnotou z příkladu na Obr. 4.11 by bylo:

cowrie.session.file_download

Toto pole může být využito, abychom vyfiltrovali, ze kterých událostí chceme brát data k dalšímu využití.


```
{
  "eventid": "cowrie.session.file_download",
  "shasum": "abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123456789",
  "url": "stdin",
  "timestamp": "2018-01-01T00:00:00.000000Z",
  "system": "SSHService 'ssh-connection' on HoneyPotSSHTransport,0,192.168.1.128",
  "isError": 0,
  "src_ip": "192.168.1.128",
  "outfile": "dl/abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123456789",
  "session": "012345abcdef",
  "sensor": "honeypot",
  "message": "Saved stdin contents with SHA-256 abcdef012345(...) to dl/abcdef012345(...)"
}
```


Obr. 4.11 Příklad JSON záznamu vygenerovaný Cowrie při stažení souboru útočníkem (upraveno).

4.4 Modul Cowrie pro nahrávání souborů na FTP server

Jedním z prvních protokolů pro přenos souborů byl protokol FTP (File Transfer Protocol). Protokol běží na bázi klient-server, klient nahrává data na server a pracuje na portech 20 a 21. Port 20 je určen pro samotný přenos dat a port 21 na instrukce spojené s přenosem. Pro jeho rozšířenost a univerzálnost u většiny hlavních operačních systému a programovacích jazyků použijeme tento protokol pro přenos škodlivých souborů nahraných na Cowrie.

Cowrie je napsán v programovacím jazyku Python, kde můžeme metody pro práci s FTP najít v knihovně „ftplib“. V knihovně jsou popsány dvě třídy: protokol FTP bez šifrování TLS:

from ftplib import FTP

a protokol FTP se šifrováním TLS:

from ftplib import FTP_TLS

Fungování a zápis těchto metod je stejný, pouze u FTP_TLS je možnost přidat certifikát klienta, aby mohl server ověřit jeho totožnost. Dále je u FTP_TLS nutné zapnout šifrovaný přenos dat pomocí příkazu:

FTP_TLS.prot_p()

Šifrovaný přenos instrukcí (přihlášení, změna adresáře atd.) je aktivní po celou dobu připojení.

Ve výstupním modulu Cowrie je vytvoření a ukončení spojení k FTP serveru možné uskutečnit v metodách start a stop. Zde by mohl nastat problém s vypršením spojení k serveru kvůli čekání na útočníka, který soubor nahrává. Z tohoto důvodu budeme připojení inicializovat a ukončovat v metodě write, po proběhnutí jedno z eventů:

cowrie.session.file_download

nebo

cowrie.session.file_upload

Touto podmínkou (Obr. 4.12, 2. a 3. řádek) máme zajištěné inicializování komunikace se serverem pouze při stažení, popř. nahrání souboru na Cowrie.

```
def write(self, event):
    if event["eventid"] == "cowrie.session.file_download" \
       or event["eventid"] == "cowrie.session.file_upload":
        self.sftp = FTP_TLS(self.hostname,self.user,self.passwd)  # Připojení k serveru
        self.sftp.prot_p()                                        # Zapnutí šifrováni dat
        self.sftp.storbinary('STOR ' + fileName, open(file,'rb')) # Nahrání souboru
        self.sftp.quit()                                          # Ukončení spojení
```

Obr. 4.12 Cowrie výstupní modul FTP (upraveno).

Na Obr. 4.12 je ve stručnosti napsána funkčnost výstupního modulu, který přenese soubory nahrané útočníkem na honeypot na server FTP. Knihovna „ftplib“ obsahuje metodu:

FTP.connect(address,port)

pro připojení k serveru. Ta je v našem případě zavolána společně s inicializací instance na řádku 4. Tuto metodu je nutno volat ručně v případě, je-li použit jiný port pro instrukce, než standartní port 21. V tomto případě musíme volat i metodu:

FTP.login(username,password)

kde parametry jsou uživatelské jméno a heslo, nutné pro přihlášení k serveru. Metodu login musíme volat i v případě, kdy při inicializaci objektu nebylo zadáno uživatelské jméno a heslo. Když nezadáme parametry do metody (tedy zavoláme jen „.login()“), budeme přihlášeni jako uživatel anonymous, pokud je serverem povolen.

Díky univerzálnosti by nám server mohl dovolit opětovné nahrání již nahraných souborů, což by u velkých souborů mohlo vést ke zbytečné zátěži. Bohužel knihovna ftplib neumožňuje kontrolu duplicitních souborů. Proto využijeme metodu pro vypsání souborů ve složce, ‚nlst()‘, která vykoná FTP příkaz „NLST“ a vrácenou hodnotou je pole jmen souborů (Obr. 4.13).

```
>>> FTP.nlst()
['soubor1.txt', 'soubor2.py', 'soubor3.sh']
```

Obr. 4.13 Příklad výstupu metody nlst.

Dále pro kontrolu duplicity využijeme zmíněný hash souboru, který použijeme při nahrávání jako nové jméno souboru. Posledním krokem je kontrola, zda není jméno (hash) souboru obsažen v poli, které bylo vráceno metodou nlst.

Funkční výstupní modul pro nahrávání na FTP server pro Cowrie je přiložen v příloze B.1.

4.5 Backend

4.5.1 Nahrávání statistik z Cowrie

4.5.1.1 Předinstalované moduly

Nyní máme vyřešeno, jak a kam budeme soubory nahrávat pomocí protokolu FTP. Zaměřme se tedy na posílání statistik do databáze a jejich následné zobrazení pomocí grafového software Grafana. V základu nabízí Cowrie více možností pro nahrávání statistik na server, resp. do databáze než u nahrávání souborů, protože existuje více serverů a databází, které mohou být použity. Pro posílání statistik máme možnost využít výstupní moduly:

- server **Elasticsearch**
- **XMPP** server
- SIEM **Splunk**
- databáze **MySQL**
- databáze **Rethinkdb**
- databáze **SQLite**
- databáze **MongoDB**

Ze zmíněných možností máme zkušenost s databází MySQL.

Databáze MySQL ([26]) využívá pro komunikaci dotazovací jazyk SQL (viz Obr.4.14, výstupy příkazů jsou v příloze A). Pro použití tohoto modulu musíme provést kroky popsané v dokumentaci:

cowrie/doc/sql/README.md

Zde je uvedeno, jaké aplikace je nutno doinstalovat a jak vytvořit tabulky. Tabulky musí být vytvořeny před začátkem nahrávání statistik, jinak by došlo k chybám a statistiky by nebyly nahrávány. Nevýhodou tohoto modulu by mohlo být množství ukládaných dat do databáze, pro která nemusíme mít využití, čímž by se zabíralo místo na disku a ve velkém množství zpomalovalo načítání dat z databáze.

```
mysql> USE cowrie;                 # Výběr, kterou databázi použijeme
mysql> SHOW TABLES;                # Zobrazení tabulek v databázi
mysql> SELECT * FROM auth;         # Zobrazení všech sloupců z tabulky auth
mysql> SELECT id,ip FROM sessions; # Zobrazení pouze sloupce 'id' a 'ip' z tabulky sessions
```

Obr. 4.14 Příklad zápisů dotazů na MySQL databázi (výstupy příkazů v příloze A).

4.5.1.2 Modul pro odesílání statistik z Cowrie do Statsite

Statistiky zasílané z Cowrie do Statsite:

- počet připojení
- počet unikátních IP adres
- počet připojení z dané IP adresy
- počet spojení přes daný protokol (SSH/Telnet)
- počet použití daného uživatelského jména a hesla
- počet nahrání daného hashe souboru
- počet použití „SSH Transport“.

Pro všechny tyto statistiky, kromě počtu unikátních IP adres, použijeme Statsite, typ dat Counter („c“), neboli čítač. Pro statistiku počtu unikátních IP adres použijeme typ Unique Set („s“).

```
# Počet připojení
"cowrie.connect_count:1|c"

# Počet unikátních IP adres
"cowrie.scr_ip_unique:{}|s".format(entry["src_ip"])

# Počet připojení z dané IP adresy
"cowrie.scr_ip.{}:1|c".format(entry["src_ip"])

# Počet spojení přes daný protokol SSH/Telnet
"cowrie.protocol.{}:1|c".format(entry["protocol"])

# Počet použití SSHTransport
"cowrie.sshtransport_count:1|c"

# Počet použití daného uživatelského jména
"cowrie.login.user.{}:1|c".format(entry["username"])

# Počet použití daného hesla
"cowrie.login.pass.{}:1|c".format(entry["password"])

# Počet nahrání daného souboru
"cowrie.file.{}:1|c".format(entry["shasum"])

# Zápis objektu socket
# Nastavení soketu na UDP
self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Odeslání paketu na danou adresu a port
self.sock.sendto("data", (self.hostname, self.port))
```

Obr. 4.15 Zápis statistik pro Statsite a zápis objektu socket.

Na obrázku 4.15 je zobrazen Statsite formát, podle kterého bychom v modulu odesílali data z Cowrie. Pro odeslání UDP paketu využijeme knihovnu socket (v Obr. 4.15 poslední 2 řádky). Parametry objektu „socket“ je rodina (IPv4 nebo IPv6) a typ protokolu (TCP nebo UDP). Parametry metody „sendto“, objektu „socket“, jsou data, která mají být poslána, a dvojice: adresa + port serveru.

Funkční výstupní modul pro nahrávání statistik z Cowrie na Statsite server je přiložen v příloze B.2.

4.6 Nahrávání statistik z Mailoney

Mailoney oproti Cowrie nemá moduly pro odesílání dat, resp. statistik. Mailoney obsahuje moduly určující funkčnost honeypotu. Proto bude implementace odesílání statistik na server Statsite vyžadovat analýzu funkčnosti jednotlivých metod v modulu, abychom mohli efektivně umístit volání metody pro Statsite.

V modulu „Schizo Open Relay“ je na konci skriptu ve třídě „SchizoOpenRelay“ metoda „process_message“, která ukládá data z honeypotu do logovacího souboru. V této metodě vložíme zavolání metody (5. řádek v Obr. 4.16) pro posílání statistik na server Statsite. Pro její funkčnost musíme ještě implementovat skript s posíláním na Statsite (1. řádek v Obr. 4.16).

```
import statsite

class SchizoOpenRelay(SMTPServer):

     def process_message(self, peer, mailfrom, rcpttos, data):
          # Zavolání metody pro odeslání dat na server Statsite
          statsite.sendData(peer[0],mailfrom,rcpttos,data)

          # Zavolání metody pro uložení dat do souboru
          log_to_file("logs/mail.log", peer[0], peer[1], data)
```

Obr. 4.16 Mailoney, modul Schizo open relay, přidání metody pro odeslání dat na server Statsite.

Tento python skript musíme vytvořit a přidat do něj metodu (5. řádek „sendData“ v Obr. 4.16), ve které budeme podobně jako u Cowrie posílat statistiky (viz Obr. 4.17) na server Statsite. U e-mailu musíme pouze myslet na možnost více příjemců („dst_email“) a kvůli tomu je nutné zprávu rozdělit, protože nemůžeme tyto e-mailové adresy poslat v jednom paketu.

Mailoney modul pro odesílání dat na server Statsite je přiložen v příloze C.1.

```
# Počet e-mailů
"mailoney.email_count:1|c"

# Počet e-mailů z dané IP adresy
"mailoney.sender_ip.{}:1|c".format(sender_ip.replace('.', '_'))

# Počet unikátních e-mailových adres
"mailoney.sender_ip_unique:{}|s".format(sender_ip.replace('.', '_'))

# Počet e-mailů z dané e-mailové adresy
"mailoney.sender_email.{}:1|c".format(urllib.quote_plus(receiver).replace('.','_'))

# Počet unikátních zdrojových e-mailových adres
"mailoney.sender_email_unique:{}|s".format(urllib.quote_plus(receiver).replace('.','_'))

# Počet e-mailů na danou cílovou e-mailovou adresu
"mailoney.dst_email.{}:1|c".format(urllib.quote_plus(receiver).replace('.','_'))

# Počet unikátních cílových e-mailových adres
"mailoney.dst_email_unique:{}|s".format(urllib.quote_plus(receiver).replace('.','_'))
```

Obr. 4.17 Mailoney, zprávy pro server Statsite.

4.7 Automatizované nasazení pomocí Puppet

4.7.1 Automatizované nasazení honeypotů Cowrie, Mailoney a serveru Statsite

Honeypoty Cowrie a Mailoney a server Statsite nejsou dostupné skrze standardní systémové repozitáře, ale musíme je stáhnout ze stránek GitHub. Máme dvě možnosti k provedení nutných příkazů pro stažení a spuštění těchto aplikací. První možností je provedení těchto příkazů pomocí funkcí „file“ a „exec“. Nevýhodou této možnosti by byla případná nepřehlednost manifest souboru, protože by každý příkaz musel být realizován pomocí samostatného bloku, který by tento příkaz provedl. Proto využijeme druhou možnost – vytvoříme skript (např. „install.sh“), který bude potřebné příkazy obsahovat. Díky tomu bude manifest obsahovat pouze jeden blok funkce „file“ pro stažení instalačního skriptu a jeden blok funkce „exec“ pro spuštění instalačního skriptu.

U Cowrie musíme přesunout konfigurační soubor („cowrie.cfg“), vlastní výstupní moduly pro nahrávání souborů na FTP („ftp.py“) a pro nahrávání statistik na Statsite („statsite.py“).

Mailoney konfigurační soubor nemá, ale stejně jako Cowrie má moduly. Z nich budeme používat „Schizo open relay“ („schizo_open_relay.py“). Tento soubor je upraven, aby umožňoval posílat statistiky na server Statsite („statsite.py“). Funkce na posílání těchto statistik je v novém souboru. Z tohoto důvodu budeme posílat upravený python skript modulu „Schizo open relay“ a python skript pro Statsite. Nakonec u serveru Statsite potřebujeme přesunout konfigurační soubor – statsite.conf.

Posledním krokem budou bloky funkce „exec“ pro restartování aplikací v případě, kdyby se změnil některý z konfiguračních souborů aplikace.

Manifest pro nasazení Cowrie, Mailoney a Statsite je přiložen v příloze D.1. Instalační skript pro Cowrie, Mailoney a Statsite je přiložen v příloze D.2.

4.7.2 Automatizované nasazení Carbon, Graphite a Grafana

Komponenty Carbon („carbon-cache“) a Graphite („graphite-web“) lze, oproti honeypotům a Statsite, nainstalovat pomocí standardních systémových repozitářů. Tyto komponenty musíme před použitím nakonfigurovat. U Carbon stačí, abychom povolili spouštění při startu operačního systému. U Graphite budeme potřebovat vytvoření uživatele a tabulku v databázi „postgresql“ pro ukládání dat. Dále v konfiguraci musíme synchronizovat Graphite s databází a vytvořit uživatele

„superuser“, přes kterého se budeme přihlašovat. Jak jsme již zmínili, Graphite vyžaduje pro přístup webový server Apache, proto spustíme zmíněné příkazy:

a2dissite 000-default *# pro vypnutí defaultní stránky*

a2ensite apache2-graphite *# pro zapnutí stránky s Graphite*

Pro instalaci Grafana pomocí systémových repozitářů přidáme odkaz k jejímu stažení do souboru:

/etc/apt/sources.list

Manifest bude obsahovat blok „file“ pro stažení a blok „exec“ pro spuštění instalačního, resp. konfiguračního skriptu. V tomto skriptu bude nastavení Carbon tak, aby se spouštěl při startu. Dále vytvoření uživatele a tabulek v databázi „postgresql“ a uživatele „superuser“ pro Graphite. U serveru Apache přepnutí aktivní stránky na Graphite. Nakonec instalaci Grafana a zapnutí jejího spuštění při startu.

Než spustíme instalační skript pro tyto aplikace, zajistíme v manifestu přesunutí některých jejich konfiguračních souborů. U webové stránky Graphite a serveru Apache, u kterých není potřeba k nim přistupovat zvenčí, ještě přesuneme upravené soubory:

apache2-graphite.conf

apache2/ports.conf

Komponenta Carbon vyžaduje konfigurační soubory:

carbon.conf

storage-schemas.conf

storage-aggregation.conf

Tyto soubory lze přesunout po spuštění instalačního skriptu, protože Carbon je již nainstalován. Grafana je nainstalována ve skriptu, proto její konfigurační soubor („local_settings.py“) přesuneme po spuštění skriptu.

Na konci manifestu stejně jako u honeypotů a Statsite vložíme bloky „exec“ pro restartování Carbon a Grafana při změně konfiguračních souborů. U webového serveru Apache stačí tzv. „reload“, kdy se načtou potřebné konfigurační soubory bez nutnosti restartování celé aplikace. Nakonec u Graphite musíme při změně konfiguračních souborů spustit příkazy pro synchronizaci s databází:

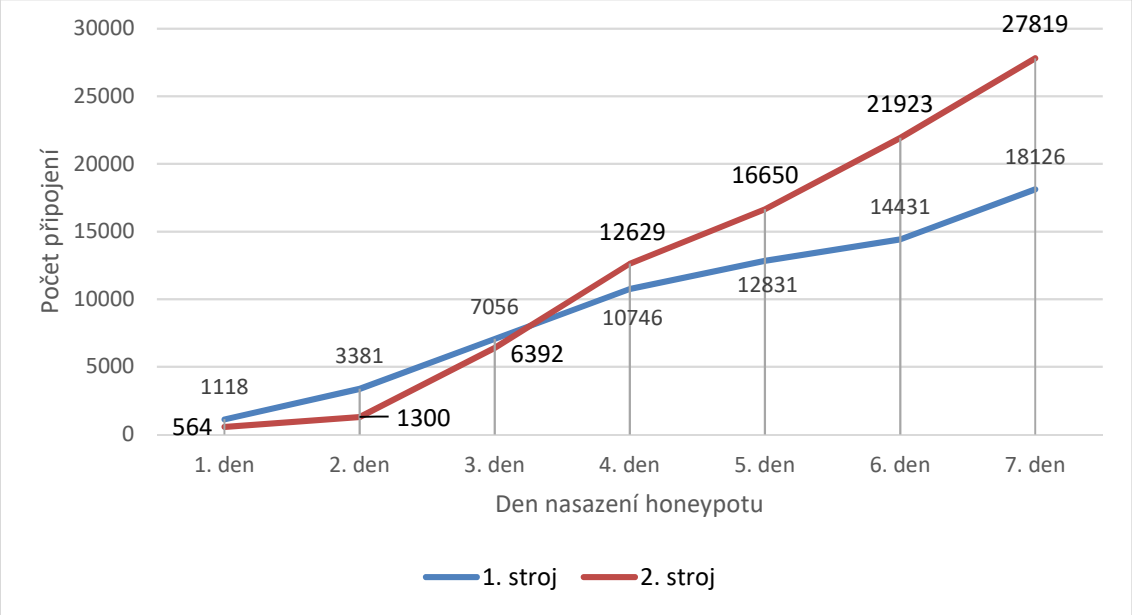
graphite-manage migrate auth

graphite-manage syncdb

Manifest pro nasazení Carbon, Graphite a Grafana je přiložen v příloze D.3. Instalační, resp. konfigurační, skript pro Carbon, Graphite a Grafana je přiložen v příloze D.4.

5 ZÍSKANÁ DATA Z COWRIE

Počet připojení s Cowrie rostl po vystavení internetu exponenciálně. V grafu na Obr. 5.1 můžeme vidět počet připojení po prvním týdnu na prvním stroji (modrý průběh) a na druhém stroji (červený průběh), který byl přidán pro odlehčení zátěže při přidání ostatních komponent na první stroj.



Obr. 5.1 Graf počtu připojení během prvního týdne nasazení honeypotu

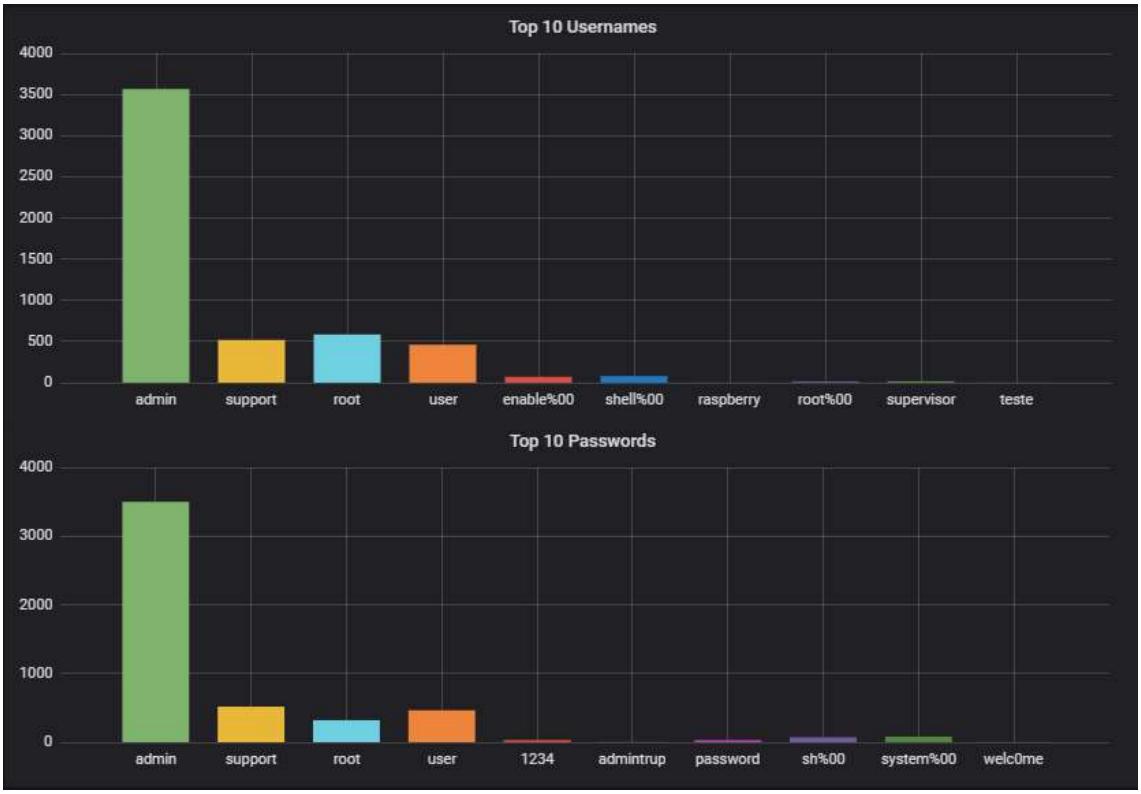
Počet různých uživatelských jmen při vytváření této práce činil okolo 1000 a počet různých hesel byl okolo 37700. Nejpoužívanější kombinace byla:

admin:admin

Jeden z důvodů populárnosti této kombinace je její použití jako prvotního přihlašovacího údaje u aplikací, ale i u např. síťových zařízení. Na Obr. 5.2 můžeme vidět deset nejpoužívanějších přihlašovacích jmen (Obr. 5.2 horní graf) a hesel (Obr. 5.2 dolní graf) během jednoho dne. Podle některých kombinací uživatelských jmen a hesel můžeme poznat zařízení, na které měli v úmyslu zaútočit. Například kombinace:

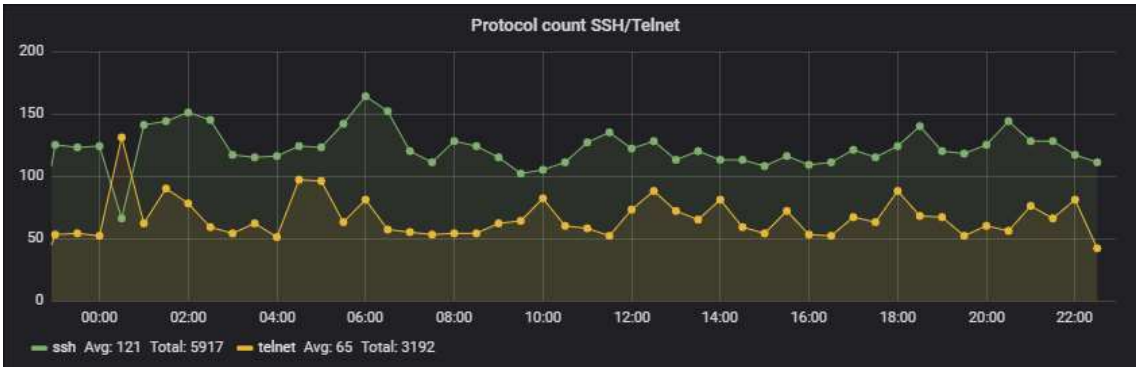
raspberry:pi

odpovídá názvu zařízení „Raspberry Pi“, což je kompaktní programovatelný počítač. Použitá kombinace mohla být defaultně nastavena na těchto zařízeních pro prvotní přihlášení.



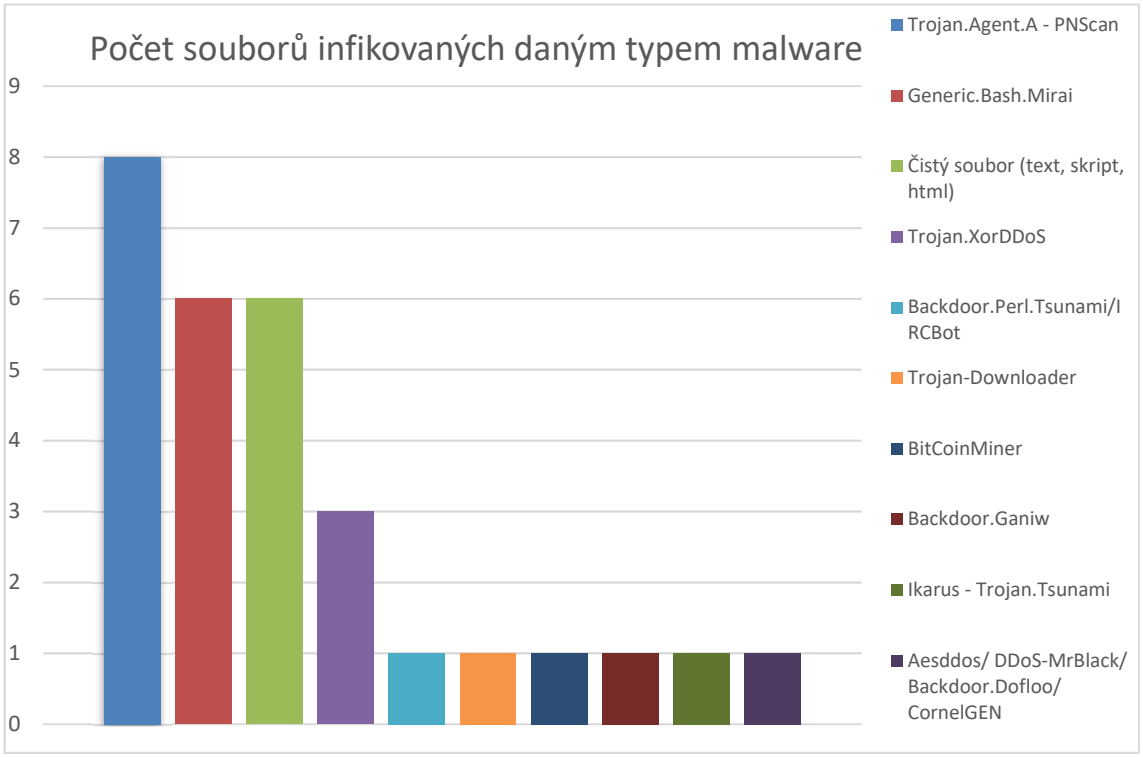
Obr. 5.3 Grafy deseti nejpoužívanějších uživatelských jmen a hesel

U Cowrie jsme použili oba protokoly, které umožňuje, tedy SSH a Telnet, přičemž SSH je zabezpečený, protože šifruje posílaná data a Telnet zabezpečený není. Z důvodu odposlechu nešifrovaných dat se v dnešní době Telnet pro standardní účely nedoporučuje používat. Z tohoto důvodu bylo překvapivé zjištění, že se jedna třetina útočníků pokoušela připojit právě pomocí Telnet, zbylé dvě třetiny připojení pak byly přes SSH (viz graf v Obr. 5.3). V obrázku 5.3 je patrné v jeden moment více připojení pomocí Telnet než přes SSH.



Obr. 5.2 Graf počtu připojení pomocí protokolů SSH (zelený průběh) a Telnet (žlutý průběh)

Dále se nám povedlo zachytit více než třicet různých souborů, které byly nahrány na Cowrie, z nichž bylo méně jak deset souborů neinfikovaných. V grafu na Obr. 5.4 jsou vypsány všechny typy malwaru a počet souborů infikovaných daným malwarem. Nejčastějším malwarem byl trojský kůň třídy PNScan. Na druhém místě je malware botnetu Mirai. Některé soubory byly na stránky VirusTotal poprvé nahrány ve stejný den, kdy se nám je podařilo zachytit na honeypotu. Podle komentářů byly také zachyceny jinými uživateli, kteří používají honeypot Cowrie.



Obr. 5.4 Graf počtu souborů infikovaných daným malwarem

6 ZÁVĚR

Cílem této bakalářské práce bylo studium problematiky honeypotu a vytvoření funkčního honeypotu. V praktické části pak ověření schopnosti honeypotu získávat informace o aktuálně se šířících hrozbách, automatizaci nasazení a autonomnosti systému.

Honeypoty mohou být silným bezpečnostním opatřením pouze v případě, pokud je zvolen správný typ honeypotu a současně je správně nakonfigurován. Honeypot však sám nezajišťuje stoprocentní získávání informací. S honeypotem by měly být součástí zabezpečení další bezpečnostní prvky typu IDS, IDPS nebo firewall.

Výstupem práce je funkční instance SSH/Telnet honeypotu Cowrie a SMTP honeypotu Mailoney, které byly nainstalovány na virtuálním stroji s operačním systémem Ubuntu 16. Cowrie odeslal veškerá důležitá data z připojení, která s ním byly učiněna, na server Statsite. Mailoney dokázal z přijatých zpráv odeslat důležitá data na server Statsite.

Některé příkazy použité útočníky nebyly v Cowrie provedeny, protože nejsou v Cowrie implementovány. Příkazy by šlo díky dobré dokumentaci doimplementovat a umožnit jejich provedení, což by mohlo umožnit získat nová data k analýze.

Podařilo se nám získat uživatelská jména a hesla, která jsou často používána útočníky pro přihlášení. Pomocí těchto údajů jsme vytvořili v Grafana grafy s deseti nejpoužívanějšími přihlašovacími údaji za posledních 24 hodin.

Jakmile byl Cowrie vystaven internetu, počet připojení rostl exponenciálně, nejčastěji útočníci zkoušeli slovníkové útoky nebo hesla hádali z konzole. Někteří útočníci se pokoušeli použít honeypot jako SSH proxy. Tuto variantu můžeme do honeypotu implementovat, abychom získali nová důležitá data, ale z důvodu náročnosti jsme od ní upustili.

Soubory, které byly nahrány útočníky, mohou být odesílány na server ftp k jejich analýze. Počet různých souborů, které byly nahrány, je přes 30, přičemž téměř všechny obsahovaly nějaký typ malwaru. Nejčastějším typem byly soubory infikované trojským koněm třídy „PNScan“, který skenuje otevřené porty počítačů v lokální síti. Zajímavým úlovkem je malwarem infikovaný soubor na těžení kryptoměny Bitcoin.

Autonomní odesílání logů jsme rozdělili na dvě fáze. První fáze je odeslání dat z honeypotů na server Statsite a druhá fáze je odeslání agregovaných dat ze Statsite do Carbon a jejich zobrazení v Grafana.

V době vypracování bakalářské práce se nám nepodařilo pomocí Mailoney zachytit žádnou komunikaci od útočníků.

Bakalářská práce směřovala k prohloubení znalosti problematiky boje s malware prostřednictvím honeypotu. V praktické části pak k propojení honeypotů s databází, resp. odesílání dat z honeypotů do databáze a automatizovaného nasazení použitých komponent. Tyto cíle byly splněny.

LITERATURA

[1] SPITZNER, Lance. *Honeypots: tracking hackers.* Boston: Addison-Wesley, c2003. ISBN 978-0321108951.

[2] *The Honeynet Project* [online]. [cit. 2018-18-05]. Dostupné z: https://www.honeynet.org

[3] *Awesome Honeypots* [online]. [cit. 2018-18-05]. Dostupné z: https://www.github.com/ paralax/awesome honeypots/blob/master/README.md

[4] *Malware Infection Vectors: Past, Present, and Future* [online]. [cit. 2018-18-05]. Dostupné z: https://www.symantec.com/connect/articles/malware-infection-vectors-past-present-and-future

[5] *Malware as a Service: Easy As It Gets* [online]. [cit. 2018-18-05]. Dostupné z: https://www.webroot.com/blog/2016/03/31/malware-service-easy-gets/

[6] *What is Riskware?* [online]. [cit. 2018-18-05]. Dostupné z: https://usa.kaspersky.com/resource-center/threats/riskware

[7] *IT threat evolution Q3 2017. Statistics* [online]. [cit. 2018-18-05]. Dostupné z: https://securelist.com/it-threat-evolution-q3-2017-statistics/83131/

[8] *What Is a Botnet?* [online]. [cit. 2018-18-05]. Dostupné z: https://www.howtogeek.com/183812/htg-explains-what-is-a-botnet/

[9] *What is SCADA?* [online]. [cit. 2018-18-05]. Dostupné z: https://inductiveautomation.com/what-is-scada

[10] *Conpot* [online]. [cit. 2018-18-05]. Dostupné z: https://www.conpot.org/

[11] *Industrial Control System* [online]. [cit. 2018-18-05]. Dostupné z: https://www.trendmicro.com/vinfo/us/security/definition/industrial-control-system

[12] *Kippo* [online]. [cit. 2018-18-05]. Dostupné z: https://github.com/desaster/kippo

[13] *Cowrie* [online]. [cit. 2018-18-05]. Dostupné z: https://github.com/micheloosterhof/cowrie

[14] *Hornet* [online]. [cit. 2018-18-05]. Dostupné z: https://github.com/czardoz/hornet

[15] Mailoney [online]. [cit. 2018-18-05]. Dostupné z: https://github.com/awhitehatter/mailoney

[16] SHIVA [online]. [cit. 2018-18-05]. Dostupné z: https://github.com/shiva-spampot/shiva

[17] *Grafana* [online]. [cit. 2018-18-05]. Dostupné z: https://grafana.com/

[18] *Graphite* [online]. [cit. 2018-18-05]. Dostupné z: https://graphiteapp.org/

[19] *Whisper* [online]. [cit. 2018-18-05]. Dostupné z: http://graphite.readthedocs.io/en/latest/whisper.html

[20] *Ceres* [online]. [cit. 2018-18-05]. Dostupné z: http://graphite.readthedocs.io/en/latest/ceres.html
[21] *Statsite* [online]. [cit. 2018-18-05]. Dostupné z: https://github.com/statsite/statsite
[22] *Statsd* [online]. [cit. 2018-18-05]. Dostupné z: https://github.com/etsy/statsd
[23] *Puppet* [online]. [cit. 2018-18-05]. Dostupné z: https://puppet.com/
[24] *Puppet Docs* [online]. [cit. 2018-18-05]. Dostupné z: https://puppet.com/docs/puppet/5.3/type.html
[25] *Secure Hash Algoritm* [online]. [cit. 2018-18-05]. Dostupné z: https://cs.wikipedia.org/wiki/Secure_Hash_Algorithm
[26] MySQL [online]. [cit. 2018-18-05]. Dostupné z: https://www.mysql.com/

Seznam příloh

A VÝSTUP DOTAZŮ NA MYSQL

```
mysql> USE cowrie;

Database changed
mysql> SHOW TABLES;
```

Tables_in_cowrie
auth
clients
downloads
input
keyfingerprints
sensors
sessions
ttylog

8 rows in set (0.00 sec)

mysql> SELECT * FROM auth;

id	session	success	username	password	timestamp
1	012345abcdef	1	root	root	2018-01-01 00:00:00

1 row in set (0.00 sec)

mysql> SELECT id,ip FROM sessions;

id	ip
012345abcdef	192.168.1.128

1 row in set (0.00 sec)

B COWRIE MODULY

B.1 Cowrie výstupní modul na nahrávání souborů na server FTP

```
# cowrie/cowrie/output/ftp.py
# U prikazu if musi byt odstraneno odradkovani a \ pro funkcnost skriptu!

import cowrie.core.output
from cowrie.core.config import CONFIG

from ftplib import FTP
from ftplib import FTP_TLS

class Output(cowrie.core.output.Output):
```

```
def __init__(self):
        addr = CONFIG.get("output_ftp", "addr").encode('utf-8')
        self.hostname = addr.split(':')[0]
        self.port = int(addr.split(':')[1])
        self.user = CONFIG.get("output_ftp", "user")
        self.passwd = CONFIG.get("output_ftp", "passwd", raw=True)
        self.directory = CONFIG.get("output_ftp", "upload_directory")
        cowrie.core.output.Output.__init__(self)

def start(self):
   pass

def stop(self):
   pass

def write(self, entry):
   if entry["eventid"] == "cowrie.session.file_download" \
      or entry["eventid"] == "cowrie.session.file_upload":
           self.connect_to_server()
           fileName = entry["shasum"]
           if self.check_if_dub(fileName) is False:
                   print('Uploading file ' + fileName + ' to FTP server.')
                   self.uploadfile(entry["outfile"],fileName)
           else:
                   print("File is allready uploaded on FTP server.")
           self.sftp.quit()


def connect_to_server(self):
    # FTP_TLS(host, user, passwd, acct, keyfile, certfile, context, timeout)
    if self.port != 21:
           self.sftp = FTP_TLS()
           self.sftp.connect(self.ip_address,self.port)
           self.sftp.login(self.user,self.passwd)
    else:
           self.sftp = FTP_TLS(self.hostname,self.user,self.passwd)
          # Automatic call of .connect() and .login()
    self.sftp.prot_p()
    self.sftp.cwd(self.directory)
    return


def check_if_dub(self,fileName):
    files_on_server = self.sftp.nlst()
    return fileName in files_on_server # Should return True or False


def uploadfile(self, file, fileName):
   self.sftp.storbinary('STOR ' + fileName, open(file,'rb'))
   return
```

B.2 Cowrie výstupní modul pro posílání statistik na Statsite server

```
# cowrie/cowrie/output/statsite.py
# U prikazu if a self.sock musi byt odstraneno odradkvani a u if i \ pro funkcnost skriptu!

import cowrie.core.output
from cowrie.core.config import CONFIG

import urllib
import socket

class Output(cowrie.core.output.Output):

    def __init__(self):
        addr = CONFIG.get("output_statsite", "address").encode('utf-8')
        self.hostname = addr.split(':')[0]
        self.port = int(addr.split(':')[1])
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        cowrie.core.output.Output.__init__(self)

    def start(self):
        pass

    def stop(self):
        pass

    def write(self, entry):
        if entry["eventid"] == "cowrie.session.connect":
            self.sock.sendto("cowrie.connect_count:1|c\n
                              cowrie.protocol.{}:1|c".format(entry["protocol"]),
                             (self.hostname, self.port))
            self.sock.sendto("cowrie.src_ip_unique:{0}|s\n
                              cowrie.src_ip.{0}:1|c".format(entry["src_ip"].replace('.', '_')),
                              (self.hostname, self.port))
        if entry["eventid"] == "cowrie.direct-tcpip.request" \
           and "HoneyPotSSHTransport" in entry["system"]:
            self.sock.sendto("cowrie.sshtransport_count:1|c", (self.hostname, self.port))
        if entry["eventid"] == "cowrie.login.success" or entry["eventid"] == "cowrie.login.failed":
            self.sock.sendto("cowrie.login.user.{0}:1|c\n
                              cowrie.login.pass.{1}:1|c".format(
                                                    urllib.quote_plus(entry["username"]),
                              urllib.quote_plus(entry["password"])), (self.hostname, self.port))
        if entry["eventid"] == "cowrie.session.file_download" \
           or entry["eventid"] == "cowrie.session.file_upload":
            self.sock.sendto("cowrie.file.{}:1|c".format(entry["shasum"]),
                             (self.hostname, self.port))
```

C MAILONEY MODUL

C.1 Mailoney modul pro odesílání dat na server Statsite

```
# mailoney/module/statsite.py
# U ".format odstranit odřádkování pro funkčnost skriptu!

# Proměnné
__statsite_ip__ = "127.0.0.1"
__statsite_port__ = 8125

import socket
import urllib

# Metoda pro odeslání statistik na server Statsite
def sendData(self, sender_ip, sender_e-mail, receivers_e-mail, data):
   self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

   self.sock.sendto("mailoney.e-mail_count:1|c", (__statsite_ip__, __statsite_port__))
   self.sock.sendto("mailoney.src_ip.{0}:1|c\nmailoney.unique_src_ip:{0}|s
                     ".format(sender_ip.replace('.', '_')), (__statsite_ip__, __statsite_port__))
   self.sock.sendto("mailoney.src_e-mail.{}:1|c".format(
                     urllib.quote_plus(receiver).replace('.','_')),
                     (__statsite_ip__, __statsite_port__))
   self.sock.sendto("mailoney.unique_src_e-mail:{}|s".format(
                     urllib.quote_plus(receiver).replace('.','_')),
                     (__statsite_ip__, __statsite_port__))
   for receiver in receivers_e-mail:
   self.sock.sendto("mailoney.dst_e-mail.{}:1|c".format(
                     urllib.quote_plus(receiver).replace('.','_')),
                     (__statsite_ip__, __statsite_port__))
   self.sock.sendto("mailoney.unique_dst_e-mail:{}|s".format(
                     urllib.quote_plus(receiver).replace('.','_')),
                     (__statsite_ip__, __statsite_port__))
   return
```

D PUPPET SOUBORY

D.1 Puppet manifest pro automatizované nasazení Cowrie, Mailoney a Statsite

```
# puppetlabs/mainfests/honeypot.pp

class honeypot {
```

```
exec { 'apt-update':
      command => '/usr/bin/apt-get update'
}

file { 'statsite-run-directory':
      path => '/var/run/statsite/',
      ensure => 'directory',
}

$packages = ['git', 'python-pip', 'python-virtualenv', 'libssl-dev', 'libffi-dev', 'scons',
                  'build-essential', 'libpython-dev', 'python2.7-minimal', 'authbind', 'check',
                  'libtool', 'automake', 'autoconf', 'gcc', 'python-pytest', 'python-requests']

package { $packages:
      ensure => "installed",
      require => Exec['apt-update'],
}

file { 'honeypot-install-script':
      path => '/opt/honeypot_install.sh',
      ensure => 'present',
      source => 'puppet:///modules/honeypot/honeypot_install_script.sh',
      require => Package[$packages],
      mode => '755',
}

exec { 'honeypot-install':
      command => '/bin/bash /opt/honeypot_install.sh',
      cwd => '/opt',
      creates => '/opt/cowrie/cowrie.cfg',
      require => File['honeypot-install-script'],
      subscribe => File['honeypot-install-script'],
      refreshonly => true,
}

file { 'cowrie-config':
      require => Exec['honeypot-install'],
      path => '/opt/cowrie/cowrie.cfg',
      ensure => file,
      source => 'puppet:///modules/honeypot/cowrie_config.cfg',
      owner => 'cowrie',
      mode => '644',
}

file { 'cowrie-userdb':
      require => Exec['honeypot-install'],
      path => '/opt/cowrie/data/userdb.txt',
      ensure => file,
      source => 'puppet:///modules/honeypot/cowrie_userdb.txt',
```

```
        owner => 'cowrie',
        mode => '644',
}

file { 'statsite-config':
        require => Exec['honeypot-install'],
        path => '/etc/statsite/statsite.conf',
        ensure => file,
        source => 'puppet:///modules/backend/statsite_config.conf',
        mode => '644',
}

file { 'cowrie-output-ftp':
        require => Exec['honeypot-install'],
        path => '/opt/cowrie/cowrie/output/ftp.py',
        ensure => file,
        source => 'puppet:///modules/honeypot/cowrie_output_ftp.py',
        owner => 'cowrie',
        mode => '644',
}
file { 'cowrie-output-statsite':
        require => Exec['honeypot-install'],
        path => '/opt/cowrie/cowrie/output/statsite.py',
        ensure => file,
        source => 'puppet:///modules/honeypot/cowrie_output_statsite.py',
        owner => 'cowrie',
        mode => '644',
}

file { 'mailoney-module-schizo':
        require => Exec['honeypot-install'],
        path => '/opt/mailoney/modules/schizo_open_relay.py',
        ensure => file,
        source => 'puppet:///modules/honeypot/mailoney_module_schizo.py',
        owner => 'mailoney',
        mode => '644',
}

file { 'mailoney-module-statsite':
        require => Exec['honeypot-install'],
        path => '/opt/mailoney/modules/statsite.py',
        ensure => file,
        source => 'puppet:///modules/honeypot/mailoney_module_statsite.py',
        owner => 'mailoney',
        mode => '644',
}

exec { 'cowrie-restart':
        command => '/bin/systemctl restart cowrie',
        require => [ File['cowrie-config'],
```

```
                    File['cowrie-output-ftp'],
                    File['cowrie-output-statsite'], ],
        subscribe => [ File['cowrie-config'],
                      File['cowrie-output-ftp'],
                      File['cowrie-output-statsite'], ],
        refreshonly => true,
    }

    exec { 'mailoney-restart':
        command => '/bin/systemctl restart mailoney',
        require => [ File['mailoney-schizo-module'],
                    File['mailoney-module-statsite'], ],
        subscribe => [ File['mailoney-schizo-module'],
                      File['mailoney-module-statsite'], ],
        refreshonly => true,
    }

    exec { 'statsite-restart':
        command => '/bin/systemctl restart statsite',
        require => File['statsite-config'],
        subscribe => File['statsite-config'],
        refreshonly => true,
    }
}
```

D.2 Instalační skript pro instalaci a konfiguraci Cowrie, Mailoney a Statsite

```
# puppetlabs/modules/honeypot/files/honeypot_install.sh
# TODO – Vytvořit „systemd“ soubory u Mailoney a Statsite

#!/bin/bash

# Cowrie

if [ ! -d "/opt/cowrie" ]; then
    useradd -d /home/cowrie -s /bin/bash -m cowrie -g users
    git clone http://github.com/micheloosterhof/cowrie
    cd cowrie
    virtualenv cowrie-env
    source /opt/cowrie/cowrie-env/bin/activate
    pip install --upgrade pip
    pip install --upgrade -r requirements.txt
    chown -R cowrie:users /opt/cowrie/
    sed -i 's/AUTHBIND_ENABLED=no/AUTHBIND_ENABLED=yes/' bin/cowrie
fi

if [ ! -f "/etc/authbind/byport/22" ]; then
```

```
    touch /etc/authbind/byport/22
    chown cowrie /etc/authbind/byport/22
    chmod 770 /etc/authbind/byport/22
fi

if [ ! -f "/etc/authbind/byport/23" ]; then
    touch /etc/authbind/byport/23
    chown cowrie /etc/authbind/byport/23
    chmod 770 /etc/authbind/byport/23
fi

# Copy of service file to start/restart Cowrie via systemctl & enabling start on boot
if [ ! -f "/etc/systemd/system/cowrie.service" ]; then
    sed -i 's:/home/cowrie/cowrie:/opt/cowrie:g' /opt/cowrie/doc/systemd/cowrie.service
    sed -i '11s:Group=cowrie:#Group=cowrie:' /opt/cowrie/doc/systemd/cowrie.service
    cp /opt/cowrie/doc/systemd/cowrie.service /etc/systemd/system/cowrie.service
    chmod 777 /etc/systemd/system/cowrie.service
    systemctl daemon-reload
    systemctl start cowrie
    systemctl enable cowrie
fi


#  Mailoney

if [ ! -d "/opt/mailoney" ]; then
    useradd -d /home/mailoney -s /bin/bash -m mailoney -g users
    git clone https://github.com/awhitehatter/mailoney
    chown -R mailoney:users /opt/mailoney/
fi

if [ ! -f "/etc/authbind/byport/25" ]; then
        touch /etc/authbind/byport/25
        chown mailoney /etc/authbind/byport/25
        chmod 770 /etc/authbind/byport/25
fi

if [ ! -f "/etc/systemd/system/mailoney.service" ]; then
    echo "[Unit]" > /etc/systemd/system/mailoney.service
    chmod 777 /etc/systemd/system/mailoney.service
    sed -i "$ a Description=Mailoney daemon" /etc/systemd/system/mailoney.service
    sed -i "$ a After=network.target\n" /etc/systemd/system/mailoney.service
    sed -i "$ a [Service]" /etc/systemd/system/mailoney.service
    sed -i "$ a Type=simple" /etc/systemd/system/mailoney.service
    sed -i "$ a User=mailoney" /etc/systemd/system/mailoney.service
    sed -i "$ a WorkingDirectory=/opt/mailoney" /etc/systemd/system/mailoney.service
    sed -i "$ a ExecStart=/usr/bin/authbind python /opt/mailoney/mailoney.py -p 25
            -s yahoo.com -t schizo_open_relay" /etc/systemd/system/mailoney.service
    sed -i "$ a Restart=on-failure\n" /etc/systemd/system/mailoney.service
    sed -i "$ a [Install]" /etc/systemd/system/mailoney.service
```

```
    sed -i "$ a WantedBy=multi-user.targety\n" /etc/systemd/system/mailoney.service

    systemctl daemon-reload
    systemctl start mailoney
    systemctl enable mailoney
fi


#  Statsite

if [ ! -d "/opt/statsite" ]; then
    pip install -U pytest requests
    git clone https://github.com/statsite/statsite.git
    cd statsite
    ./autogen.sh
    sed -i '5955s/PKG_CHECK_MODULES(CHECK, check >= 0.9.7, have_check="yes",
            { $as_echo "$as_me:${as_lineno-$LINENO}: WARNING: Check not found;
            cannot run unit tests!" >&5/#PKG_CHECK_MODULES(CHECK, check >= 0.9.7,
            have_check="yes", { $as_echo "$as_me:${as_lineno-$LINENO}: WARNING:
            Check not found; cannot run unit tests!" >&5/' /opt/statsite/configure
    sed -i '5956s/$as_echo "$as_me: WARNING: Check not found; cannot run unit tests!"
             >&2;})/#$as_echo "$as_me: WARNING: Check not found; cannot run unit tests!"
             >&2;})/' /opt/statsite/configure
    ./configure
    make
    make install
fi

if [ ! -f "/etc/statsite" ]; then
    mkdir /etc/statsite
fi

if [ ! -f "/etc/systemd/system/statsite.service" ]; then
    echo "[Unit]" > /etc/systemd/system/statsite.service
    chmod 777 /etc/systemd/system/statsite.service
    sed -i "$ a Description=Statsite Deamon" /etc/systemd/system/statsite.service
    sed -i "$ a After=network.target\n" /etc/systemd/system/statsite.service
    sed -i "$ a [Service]" /etc/systemd/system/statsite.service
    sed -i "$ a Type=simple" /etc/systemd/system/statsite.service
    sed -i "$ a PIDFile=/var/run/statsite.pid" /etc/systemd/system/statsite.service
    sed -i "$ a WorkingDirectory=/opt/statsite" /etc/systemd/system/statsite.service
    sed -i "$ a ExecStart=/usr/local/bin/statsite -f /etc/statsite/statsite.conf"
            /etc/systemd/system/statsite.service
    sed -i "$ a Restart=on-failure\n" /etc/systemd/system/statsite.service
    sed -i "$ a [Install]" /etc/systemd/system/statsite.service
    sed -i "$ a WantedBy=multi-user.target\n" /etc/systemd/system/statsite.service

    systemctl daemon-reload
    systemctl start statsite
    systemctl enable statsite
```

```
fi
```

D.3 Puppet manifest pro automatizované nasazení Graphite a Grafana

```
# puppetlabs/manifests/backend.pp

class backend {

    exec { 'apt-update':
        command => '/usr/bin/apt-get update'
    }

    $packages = ['graphite-web', 'graphite-carbon', 'postgresql', 'libpq-dev', 'expect', 'curl',
                 'python-psycopg2', 'apache2', 'git', 'libapache2-mod-wsgi']

    package { $packages:
        ensure => 'installed',
        require => Exec['apt-update'],
    }

    file { 'ssl-directory':
        path => '/opt/ssl',
        ensure => 'directory',
    }
    file { 'ssl-cert':
        path => '/opt/ssl/certificate.pem',
        ensure => file,
        source => 'puppet:///modules/ssl/certificate.pem',
        require => File['ssl-directory'],
        mode => '644',
    }
    file { 'ssl-key':
        path => '/opt/ssl/cert_key.pem',
        ensure => file,
        source => 'puppet:///modules/ssl/cert_key.pem',
        require => File['ssl-directory'],
        mode => '644',
    }

    file { 'carbon-config':
        path => '/etc/carbon/carbon.conf',
        ensure => file,
        source => 'puppet:///modules/backend/carbon_config.conf',
        require => Package[$packages],
        mode => '644',
    }
    file { 'carbon-storage-schemas-config':
```

```
        path => '/etc/carbon/storage-schemas.conf',
        ensure => file,
        source => 'puppet:///modules/backend/carbon_storage_schemas.conf',
        require => Package[$packages],
        mode => '644',
}
file { 'carbon-storage-aggregation-config':
        path => '/etc/carbon/storage-aggregation.conf',
        ensure => file,
        source => 'puppet:///modules/backend/carbon_storage_aggregation.conf',
        require => Package[$packages],
        mode => '644',
}

file { 'graphite-config':
        path => '/etc/graphite/local_settings.py',
        ensure => file,
        source => 'puppet:///modules/backend/graphite_config.py',
        require => Package[$packages],
        mode => '644',
}

file { 'apache-graphite-site-config':
        path => '/etc/apache2/sites-available/apache2-graphite.conf',
        ensure => file,
        source => 'puppet:///modules/backend/apache_graphite_site.conf',
        require => Package[$packages],
        mode => '644',
}
file { 'apache-ports-config':
        path => '/etc/apache2/ports.conf',
        ensure => file,
        source => 'puppet:///modules/backend/apache_ports.conf',
        require => Package[$packages],
        mode => '644',
}

file { 'backend-install-script':
        path => '/opt/backend_install.sh',
        ensure => 'present',
        source => 'puppet:///modules/backend/backend_install.sh',
        require => Package[$packages],
        mode => '755',
}

exec { 'backend-install':
        command => '/bin/bash /opt/backend_install.sh',
        cwd => '/opt',
        require => File['backend-install-script'],
        subscribe => File['backend-install-script'],
```

```
        refreshonly => true,
    }

    file { 'grafana-config':
        path => '/etc/grafana/grafana.ini',
        ensure => file,
        source => 'puppet:///modules/backend/grafana_config.ini',
        require => Exec['backend-install'],
        mode => '644',
    }

    exec { 'carbon-restart':
        command => '/bin/systemctl restart carbon-cache',
        require => [ File['carbon-config'],
                     File['carbon-storage-schemas-config'],
                     File['carbon-storage-aggregation-config'], ],
        subscribe => [ File['carbon-config'],
                       File['carbon-storage-schemas-config'],
                       File['carbon-storage-aggregation-config'], ],
        refreshonly => true,
    }

    exec { 'graphite-manage':
        command => '/usr/bin/graphite-manage migrate auth &&
                    /usr/bin/graphite-manage syncdb',
        require => File['graphite-config'],
        subscribe => File['graphite-config'],
        refreshonly => true,
    }

    exec { 'apache-reload':
        command => '/bin/systemctl reload apache2',
        require => [ File['graphite-config'],
                     File['apache-graphite-site-config'],
                     File['apache-ports-config'], ],
        subscribe => [ File['graphite-config'],
                       File['apache-graphite-site-config'],
                       File['apache-ports-config'], ],
        refreshonly => true,
    }

    exec { 'grafana-restart':
        command => '/bin/systemctl restart grafana-server',
        require => File['grafana-config'],
        subscribe => File['grafana-config'],
        refreshonly => true,
    }
}
```