VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Low-interaction Honeypot na jednodeskovém počítači
Low-interaction Honeypot on a Single-board Computer

2020 Jiří Jakuba

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jiří Jakuba**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Low-interaction Honeypot na jednodeskovém počítači
Low-interaction Honeypot on a Single-board Computer

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit jednoduchý honeypot s nízkou (popř. střední) úrovní provázání služeb (low-interaction, medium-interaction) na platformě malého jednodeskového počítače (např. Raspberry Pi). Honeypot bude poskytovat běžné aplikační protokoly (HTTP/HTTPs, SSH, SMTP), popř. přesměrovávat provoz na jiný cíl. Pro realizaci jednotlivých služeb je možné využít existující implementaci honeypotu (Cowrie/Kippo, tanner/snare, apod.) nebo implementovat vlastní řešení.

1. Nastudujte jak fungují honeypoty, a jaké jsou rozdíly mezi jejich typy.
2. Prostudujte existující řešení honeypotů a popište jejich typické vlastnosti. Zaměřte se zejména na low-interaction honeypoty.
3. Navrhněte honeypot, který bude realizovat běžné služby simulující klientskou stanici resp. webový server a bude provozován na jednodeskovém počítači (jako běhové prostředí bude použita distribuce Raspbian/Armbian resp. LEDE/OpenWRT).
4. Implementujte rozhraní, které umožní nasadit jednotlivé služby, notifikovat vlastníka honeypotu o probíhajícím útoku a analyzovat útok po jeho ukončení resp. v jeho průběhu.
5. Výsledné řešení otestujte a zhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

[1] Niels Provos, Thorsten Holz: Virtual Honeypots: From Botnet Tracking to Intrusion Detection, Addison-Wesley Professional, 2007.
[2] Chris Sanders and Jason Smith: Applied Network Security Monitoring: Collection, Detection, and Analysis, Syngress, 2013, ISBN: 978-0124172081.
[3] Jose Nazario: Awesome Honeypots. [online] [cit. 2018-02-28]. Dostupné z: <https://github.com/paralax/awesome-honeypots>

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Moravec, Ph.D.**

Datum zadání: 01.09.2018

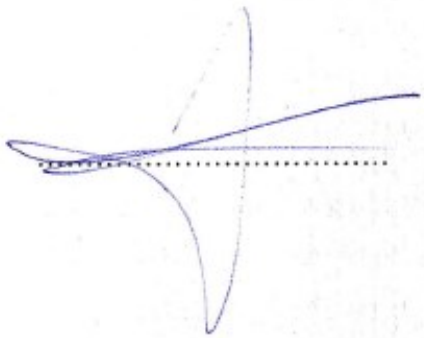
Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. května 2020



Na tomto místě bych rád poděkoval Ing. Pavlovi Moravcovi, Ph.D. za pomoc, věcné připomínky, návrhy a velkou trpělivost při vypracování této bakalářské práce.

Abstrakt

Cílem této bakalářské práce je návrh a řešení implementace nasazení honeypotu, který poskytuje detekci a analýzu útoků na webový server. Honeypot je provozován na jednodeskovém počítači Raspberry Pi 3+. Skládá se z několika existujících implementací honeypotů s nízkou nebo střední mírou interakce jako služeb pro poskytnutí detekce útoků prostřednictvím běžných aplikačních protokolů (SSH, SMTP, HTTP), ty jsou nainstalovány na webovém serveru. Součástí řešení je Java webová aplikace, jež slouží k nasazení, nastavení služeb a zobrazení informací o jednotlivých útocích, které na honeypotu proběhly.

Klíčová slova: honeypot, webový server, Raspberry Pi, Java, Python, Apache, webová aplikace, Debian

Abstract

The purpose of this bachelor thesis is to design and implement a solution of a honeypot deployment, which provides detection and analysis of malicious attacks against a web server. Honeypot is running on a single-board computer Raspberry Pi 3+. It consists of several existing implementations of honeypots with low and medium amount of interaction used for detecting attacks by commonly used application protocols (SSH, SMTP, HTTP). These are installed on the web server. Solution includes the Java web application, which provides deployment, configuration of the services as well as information about the malicious attacks which were performer on the honeypot.

Key Words: honeypot, web server, Raspberry Pi, Java, Python, Apache, web application, Debian

Obsah

Seznam použitých zkratek a symbolů 9
Seznam obrázků a ilustrací 10
Seznam tabulek 11
1 Úvod 12
2 Historie 13
3 Honeypot 14
3.1 Charakteristika 14
3.2 Výhody honeypotů 15
3.2.1 Velká hodnota získaných dat 15
3.2.2 Jednoduchost 15
3.2.3 Využití prostředků 15
3.3 Nevýhody honeypotů 16
3.3.1 Omezená působnost 16
3.3.2 Fingerprinting 16
3.3.3 Riziko využití 16
3.4 Klasifikace 16
3.5 Úrovně interakce 17
3.5.1 Honeypoty s nízkou úrovní interakce 17
3.5.2 Honeypoty se střední úrovní interakce 17
3.5.3 Honeypoty s vysokou úrovní interakce 18
4 Srovnání existujících implementací honeypotů 19
4.1 Popis jednotlivých řešení honeypotů 19
5 Použité technologie 23
5.1 Raspberry Pi 23
5.2 Java 23
5.3 IntelliJ IDEA 24
5.4 WebStorm 24
5.5 React 24
5.6 Spring 24
5.7 Apache Tomcat 24
5.8 Java JSch 25

5.9 Bootstrap 25
5.10 PostgreSQL 25
6 Návrh řešení 26
6.1 Hardware platforma 26
6.2 Zvolené implementace honeypotů 26
6.3 Rozhraní 26
7 Implementace aplikace 27
7.1 Funkce aplikace 27
7.1.1 Případy užití 28
7.2 Ukládání dat o hostu 29
7.2.1 Databáze honeypotapp 29
7.2.2 Přihlašovací údaje 31
7.2.3 Funkce cacheManager 31
7.3 Třída SSHConnection 32
7.4 Instalace a administrace honeypotů v aplikaci 33
7.4.1 Cowrie 33
7.4.2 Tanner 36
7.4.3 Snare 37
7.5 Třída HoneypotServiceImpl 38
7.5.1 Metoda updateState 38
7.5.2 Metoda saveConnectionDetails 38
7.6 Frontendová část aplikace 39
7.6.1 Přidání a úprava hosta ve frontendové části 39
7.7 Spuštění webové aplikace 41
8 Testování honeypot prostředí 42
8.1 Testování honeypotu Cowrie 42
8.2 Testování honeypotů Snare a Tanner 44
9 Závěr 47
Literatura 48
Obrázky 50
A Přílohy 51

Seznam použitých zkratek a symbolů

API	Application Programming Interface
CSS	Cascading Style Sheets
DDoS	Distributed Denial of Service
ER	Entity-Relationship
FTP	File Transfer Protocol
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IIS	Internet Information Services
IP	Internet Protocol
JS	JavaScript
JSch	Java Secure Channel
JSON	JavaScript Object Notation
MVC	Model View Controller
NAT	Network Address Translation
ORM	Object Relational Mapping
PHP	Hypertext Preprocessor
POP3	Post Office Protocol
RAM	Random Access Memory
SCP	Secure Copy
SFTP	Secure File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SSH	Secure Shell
TCP	Transmission Control Protocol
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	eXtensible Markup Language

Seznam obrázků a ilustrací

1: Deception ToolKit 13
2: Ukázka prostředí sítě Honeynet 14
3: Shadow Deamon UI 20
4: Raspberry Pi 3 Model B+ 23
5: Diagram případů užití 27
6: Tabulka host 30
7: Uživatelské rozhraní pro přidání hosta do databáze 31
8: Příkazy nmap a ifconfig 42
9: Příkaz nmap 43
10: Výpis souboru /etc/passwd/ 43
11: Logy útoku na honeypot Cowrie 44
12: Tanner log 45
13: Tanner log 45
14: Výstup testování nástroje Nikto 46

Seznam tabulek

1: Srovnání existujících implementací honeypotů 19

1 Úvod

V dnešní době se stále častěji zpracovávají informace prostřednictvím informačních technologií. Tyto informace mohou mít velkou hodnotu pro jejich vlastníky (např. vládní dokumenty, bankovní účty, osobní údaje, obchodní transakce), a proto je třeba je chránit tak, aby se zamezilo jakémukoli neoprávněnému zacházení s nimi. Je to jeden z hlavních důvodů, proč v informatice obor počítačová bezpečnost, která se zabývá ochranou informací v počítačových sítích a systémech, prevencí a odhalením nežádoucí manipulace s informacemi či zařízeními, vůbec vznikl.

Důležitost informační bezpečnosti se zvyšuje s čím dál tím větším využitím moderních počítačových technologií v našem každodenním životě. Ať už se jedná o internet, lokální sítě, bezdrátové sítě nebo vzestup tzv. „chytrých zařízení“, jako jsou chytré telefony, televize, zařízení patřící do sítě Internet of Things (IoT). Počítačová bezpečnost se zaměřuje na ochranu všech prostředků souvisejících s informačními technologiemi pomocí aplikování různých postupů a pravidel. Jedním z postupů ochrany je využití honeypotů. Jedná se o počítačové systémy, které mají za cíl na sebe přitahovat a detekovat zákeřné útoky (malware, hacker), následně tyto aktivity analyzovat a pomoci nich jim předejít.

Téma bakalářské práce jsem si vybral, jelikož problémy spojené s počítačovou bezpečností nás dennodenně obklopují. Tuto problematiku považuji za zajímavou a mám zájem o prohloubení vědomostí v tomto směru.

Cílem předložené bakalářské práce bylo v teoretické části se seznámit s problematikou a implementací honeypotů, zjistit rozdíly mezi jejich typy. Dále popsat již existující řešení honeypotů a srovnat jejich typické vlastnosti, např. typ služby, pro kterou jsou určeny, úroveň integrace (nízká, střední, vysoká), technologie, kterou jsou implementovány, platforma, pro kterou je honeypot určen a byl na ní testován. Většina z vybraných honeypotů jsou řešení s nízkou (střední) úrovní provázání služeb v souladu se zadáním práce. Následně bylo v práci popsáno navržení honeypotu, výběr a popis jednotlivých služeb, protokolů a platforem využitých ke zpracování práce.

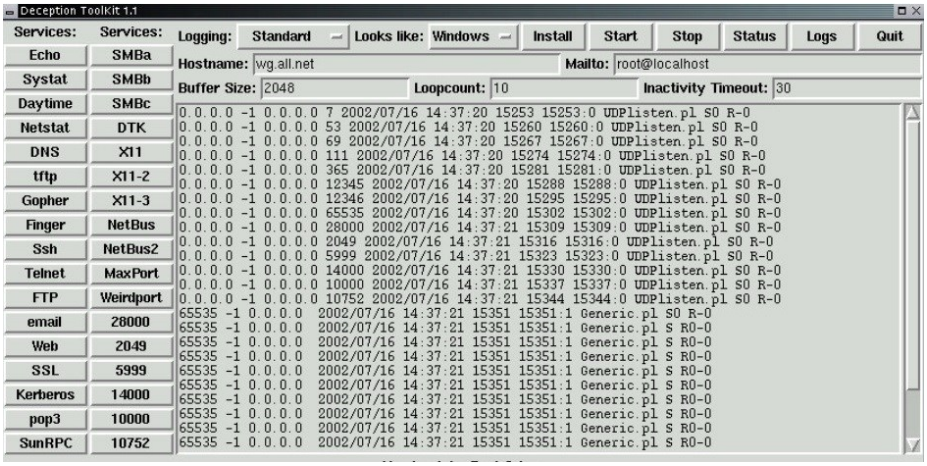
Praktická část byla zaměřena na instalaci honeypotu, jeho konfiguraci pro jednotlivé služby (HTTP, SSH, SMTP). Honeypot byl provozován na jednodeskovém počítači Raspberry Pi 3 Model B+ obsahující prostředí distribuce Raspbian. Pro simulaci klientského prostředí byl použit webový server Apache. Pro celé prostředí bylo navrženo grafické rozhraní, jenž umožňuje spustit a konfigurovat jednotlivé služby, notifikovat vlastníka nebo jiného uživatele honeypotu o probíhajícím či již proběhlém útoku, a analýzu útoků. Implementace rozhraní byla provedena v jazyce Java, respektive v jeho webové technologii Java Enterprise Edition. Ovládání a všechny funkce byly podrobně popsány. Ověření funkčnosti proběhlo formou testování.

2 Historie

Až do poloviny 80. let 20. století se o honeypotech ve světě příliš nemluvilo. Debaty spustil až hackerský útok západoněmeckého hackera Markuse Hesse v roce 1986, který se naboural do vládních institucí např. USA, Evropy, západní i východní Asie a prodával jejich informace sovětské tajné službě KGB. Jeden z jeho útoků byl cílen na vládní laboratoře Lawrence Berkeley Laboratory, kde pracoval Clifford Stoll, který FBI pomohl za pomocí fyzického honeypotu Markuse Hesse dopadnout [1,2].

Významnou osobností v oblasti počítačové bezpečnosti je Wiliam R. „Bill“ Cheswick, který v roce 1991 publikoval svou vědeckou práci „An Evening with Berferd“. V ní popisuje své zkušenosti při vytváření jeho prvního prostředí pro monitorování a zaznamenávání hackerské aktivity. Vyvinul systém obsahující falešné služby, soubory s hesly a přidal jej do produkčního prostředí firmy AT&T. Také vytvořil skript, který zaznamenával veškerou aktivitu v systému do logu. Pro celý systém následně udělal samostatné prostředí „Jail“, jenž připomínalo dnešní honeypotové prostředí [1,3].

První honeypot vytvořen dr. Fredem Cohenem za účelem ochrany systémů byl Deception ToolKit (DTK) v roce 1997. Jeho principem bylo čekání na vstup příkazu od útočníka a odpovídat na něj. Všechna komunikace byla zaznamenávána do logů. Využívala množství služeb např. FTP, Telnet, POP3, SSH. Později umožňovala na útočníka zaútočit zpět pomocí oklamání (deception). Lze jej využít i dnes [6].



Obrázek 1: Deception ToolKit

V roce 1999 Lance Spitzner poprvé použil slovo „honeypot“ ve své práci „To Build a Honeypot“. Slovo odvodil podle anglických slov honey (med) a pot (hrnec, nádoba). Zdůvodnil to podobou principu medu v hrnci, který slouží jako návnada pro mravence a děti. Návnadou v informačních technologiích jsou citlivá data a mravenci jsou útočníci (hackeři). Sám legitimizoval honeypoty jako samostatnou disciplínu v oboru počítačové bezpečnosti [3,4,5].

Na přelomu tisíciletí byl založen The Honeynet Project, což je nezisková organizace složená z bezpečnostních odborníků po celém světě zabývajících se nejnovějšími druhy útoků, chováním útočníků a vývojem nástrojů pro ochranu a obranu systému. Je aktivní dodnes [7].

Princip honeypotů zůstal od 90. let minulého století prakticky nezměněn. V počítačové bezpečnosti se používají dodnes, avšak již v omezené míře. Jejich funkce dnes spočívá především k zjištění útoků v produkčních prostředích a při tvoření virových definic pro antivirové programy.

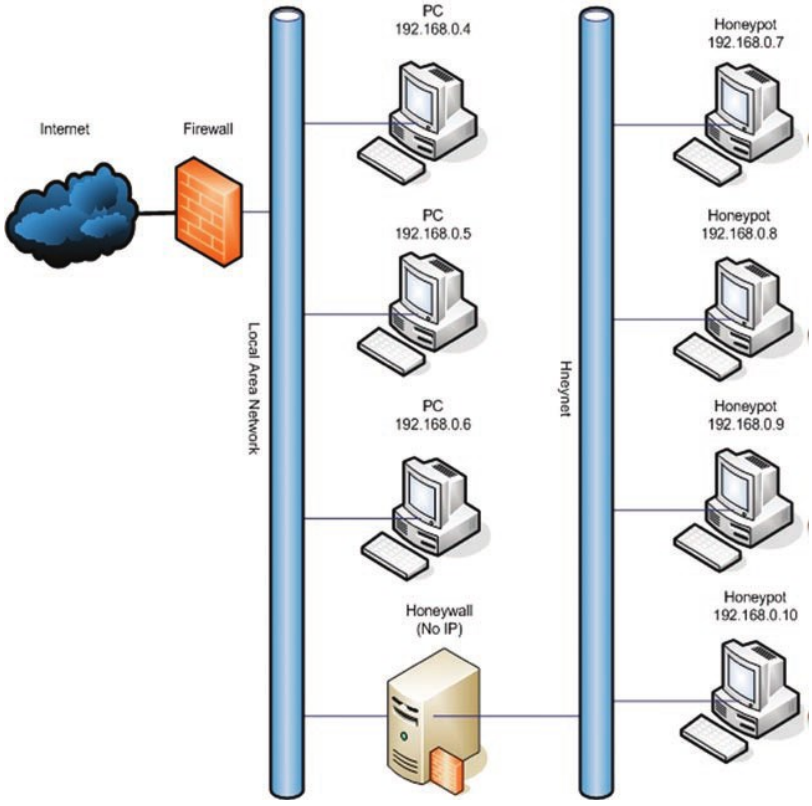
3 Honeypot

3.1 Charakteristika

Honeypot je bezpečnostní informační systém, jehož účelem je na sebe přitahovat nebezpečné aktivity, kterými mohou být počítačové viry, spam, pokus o neoprávněný zisk dat nebo čtení osobní komunikace. Jeho principem je nalákat útočníka, aby na honeypot zaútočil a pomocí logování akcí provedených hackerem umožňuje tyto útoky studovat a chránit se proti nim. Většinou se jedná o záměrně špatně zabezpečený systém, který se chová jako jeden z uzlů v síti nebo produkčním prostředí. Honeypoty velmi často nemají pro firmy žádnou produkční hodnotu, jsou určeny výhradně ke svému původnímu účelu – detekci. Tato skutečnost prakticky vylučuje možnost výskytu falešných odhalení útoku, protože jakákoli komunikace a interakce s honeypotem není žádoucí [1,3,4].

Nemusí se ovšem jednat jen o počítač. Existují tzv. Honeytokens, což jsou honeypoty, kterými nejsou počítače, ale různé informační objekty např. textový soubor, databáze, falešný uživatelský účet. Pokud se někdo takový soubor pokusí otevřít nebo použije nikým nepoužívaný účet, s největší pravděpodobností se jedná o útočníka [3].

Honeynets jsou velké sítě tvořené honeypoty simulující reálné počítačové sítě. Jsou připojené a komunikují mezi sebou stejně jako objekty v síti. Pro útočníka se jeví jako obyčejná produkční síť. Potřebují mnoho finančních prostředků a času pro vytvoření, ale poskytují značné množství informací o tom, jak se útočníci v napadených sítích chovají a mezi jednotlivými zařízeními pohybují. Díky jejich složitosti může útočníkovi trvat déle, než zjistí, že se pohybuje v prostředí obsahující výhradně honeypotové systémy [8].



Obrázek 2: Ukázka prostředí sítě Honeynet

3.2 Výhody honeypotů

3.2.1 Velká hodnota získaných dat

V dnešní době organizace zpracovávají každodenně velké množství dat, ať se jedná o systémové logy, firewall logy nebo záznamy pokusů o vniknutí do systémů. Množství těchto dat je obrovské a část těchto informací nemusí mít vůbec žádnou hodnotu. Je tak velmi těžké odvodit si jejich cenu. Honeypoty, na rozdíl od zmíněných způsobů získání dat, jich sice poskytují mnohonásobně méně, ale zato jsou tato data obvykle velmi cenná. Jedná se o důsledek konceptu honeypotu, jakákoliv aktivita provedena v jeho systému je nežádoucí. Velikost dat se pohybuje kolem několika megabajtů za den i méně, ovšem jakýkoliv záznam je pravděpodobně pokus o útok nebo vniknutí do systému [5].

Informace poskytnuté honeypotem jsou utvořeny tak, aby jim bylo možné snadno a rychle porozumět. Analýza těchto dat je mnohem jednodušší a rychlejší. Například již dříve zmíněný The Honeynet Project přijme každý den jen několik megabajtů dat za den. I když se jedná o opravdu malé množství dat, obsahují hlavně data o zákeřných aktivitách. Tato data mohu následně být použita pro statistické modelování, zlepšení detekce útoků, analýzu trendů i studium chování samotných útočníků [7].

3.2.2 Jednoduchost

Lance Spitzner ve své knize Honeypots: Tracking Hackers považuje jednoduchou implementaci a koncept honeypotů jako jejich největší výhodu. Není potřeba vymýšlet žádné složité algoritmy, administrovat složité databáze ani špatně nakonfigurovat pravidla. Čím jednodušší je koncept samotného honeypotu, tím je většinou účinnější [5].

3.2.3 Využití prostředků

Velkým problémem, se kterým se většina bezpečnostních prvků potýká, je jejich využití prostředků (např. RAM) a jejich vyčerpání. Vyčerpání prostředků nastává, pokud bezpečností prvek už nedokáže pokračovat ve své obvyklé funkci kvůli jeho zahlcení. Příkladem mohou být firewally, které selhávají kvůli nedostatků prostředků, jejich tabulka připojení je plná nebo už nejsou schopny monitorovat všechna připojení. Firewall zablokuje veškerý síťový provoz místo blokování pouze neautorizovaných aktivit. Existují i případy, kdy se systémům pro detekci podezřelých aktivit naplní buffer (vyrovnávací paměť) a dojde ke ztrátě paketů, což může vést ke ztrátě informací o potenciálním útoku [5].

U honeypotů zpravidla nedochází k velkému využití prostředků. Výjimkou mohou být honeypoty simulující produkční systémy. Na rozdíl od jiných systému pro detekci útoků jimi neprochází tolik síťového provozu, protože monitoruje pouze aktivity zaměřené jen na sebe, a ne na celou síť. Z toho důvodu honeypoty nevyžadují velké investice do jejich hardware, jakými mohou být drahé výkonné procesory, velké kapacity pamětí RAM nebo software [3].

3.3 Nevýhody honeypotů

3.3.1 Omezená působnost

Největší nevýhodou honeypotů je jejich velmi omezená působnost [5]. Honeypot je schopen zaznamenávat a analyzovat útoky zaměřené jen na sebe. Pokud dojde k útoku na honeypot i ostatní systémy v síti, nemá honeypot žádnou možnost zjistit, že k útoku došlo, pokud není sám napaden. Jestliže útočník honeypot objeví, může se útoku na něj vyhnout a infiltrovat zbytek sítě. Administrátoři se o útoku nemusí nikdy dozvědět, síť zůstane náchylnější k dalším infiltracím, např. útočník si vytvoří tzv. „backdoor“ (zadní vrátka), které může využít k budoucím útokům na stejnou síť nebo systém.

3.3.2 Fingerprinting

Fingerprinting je jev, ke kterému dochází, pokud je útočník schopen pomocí různých charakteristik a chování systému zjistit, že se jedná o honeypot. Např. honeypot emuluje webový server. Dojde na něm k chybě a v hlášce se nachází špatně zapsaný HTML příkaz (div je špatně napsán jako dav). Tato chyba, ke které by obyčejně nedošlo, protože se jedná jen o emulaci webového serveru, ne jeho správnou implementaci v produkčním prostředí, je fingerprintem (otiskem prstu) honeypotu. Útočník tuto chybu rychle zaznamená a útok zastaví. Může využít takové chyby a poslat falešné útoky z jiných systému ve stejném prostředí, v nímž se honeypot nachází. Když následně budou administrátoři vyšetřovat příčinu těchto falešných útoků, útočník infiltruje ostatní systémy.

Nejhorší dopad má fingerprinting na výzkumné honeypoty. Útočník může na honeypot zasílat falešné informace, což obvykle vede ke špatným poznatkům o těchto útocích [10].

3.3.3 Riziko využití

Další nevýhodou honeypotů je riziko, které přináší prostředím, v nichž se nachází. Pokud útočník zaútočí na honeypot, může jej využít k infiltraci nebo napadení ostatních počítačových systémů. Každý honeypot poskytuje různou úroveň takového rizika. U některých honeypotů je riziko velmi malé, avšak jiné druhy mohou útočníkovi sloužit jako vstupní brána, ze které se infiltruje do jiných systémů. Čím jednodušší je honeypot, tím zpravidla menší je jeho riziko používání. Např. honeypot emulující jen malé množství základních služeb je velmi těžké využít jako prostředek pro napadení jiných systémů. Honeypoty, které fungují jako tzv. vězení, obsahují plnohodnotné operační systémy. Útočník může z takového vězení „utéct“ a použít ho k vysílání pasivních a aktivních útoků na ostatní systémy. Dalším faktorem rizika využívání honeypotů je jejich implementace. Vzhledem k této nevýhodě nelze honeypotem nahradit jiná zařízení pro zabezpečení informačních systémů, jakými jsou firewally nebo systémy pro odhalení průniku (Intrusion Detection Systems) [5].

3.4 Klasifikace

Honeypoty je možné dělit podle různých charakteristik. Existují dva hlavní druhy honeypotů – honeypoty pro produkční prostředí a výzkumné honeypoty. Honeypoty určené pro výzkum jsou většinou využívány armádou, výzkumnými nebo vládními institucemi (v USA např. FBI, NSA) pro zisk informací o chování a postupu útočníků při útoku. Jsou vhodné k identifikaci počítačových červů, tedy

programů, které se dokážou samy šířit přes síť nebo jiná média. Získané poznatky o útocích mohou být využity k budoucí zlepšení ochrany a detekce a celkovému zvýšení bezpečnosti produkčních prostředí.

Honeypoty v produkčních prostředích mají za úkol pomoci se snižováním rizika útoku, nezabraňují jim však zcela. Lákají útočníka, aby cílil svůj útok na honeypot, který se vydává za cíl s důležitými daty. Tento způsob bezpečnosti vede k oklamání útočníka, ale získává tak čas pro administrátory, aby ztráta důležitých dat byla co nejmenší [9].

3.5 Úrovně interakce

Nejčastějším rozlišením jednotlivých honeypotů je mezi jejich úrovněmi interakce, která je honeypotem útočníkům umožněna. Čím vyšší úroveň interakce má prostředí honeypotu, tím podobnější a věrohodnější je k opravdovému produkčnímu prostředí. To může vést k získání detailnějším a přesnějším informacím o útoku. Určitá nevýhoda honeypotů s vysokou mírou provázání služeb nastává díky jejich výrazně složitější instalaci, konfiguraci a správě oproti honeypotů s nízkou úrovní interakce.

3.5.1 Honeypoty s nízkou úrovní interakce

Honeypoty s nízkou úrovní interakce jsou obyčejně nejlehčí na instalaci, konfiguraci a správu. Je to dané jejich jednoduchou implementací a funkčností. Zpravidla emulují jen služby (SSH, Telnet). Jsou určeny k získání informací o útocích, které již jsou známé a je předem zřejmé, jak se bude útočník chovat. Nejsou vhodné k odhalování nových hrozeb nebo jako výzkumné honeypoty. Příkladem je napodobení unixového systému obsahujícího základní služby (FTP, Telnet, SSH). Útočník se může prostřednictvím protokolu Telnet pokusit o infiltraci systému útokem hrubou silou (Brute force attack). Honeypot zaznamenává do logů všechny informace o těchto pokusech. Dalším příkladem je SMTP honeypot představující mail server, který zachytává všechnu nevyžádanou poštu a ukládá ji do logů [3,4,5,11].

Největší předností honeypotů s nízkou úrovní interakce je jejich detekce nežádoucích pokusů o připojení a jejich správa. Většinou se jedná o program nainstalovaný na hostitelském systému honeypotu, který si jeho správce může nakonfigurovat podle potřeby. Zároveň poskytují nejmenší riziko využívání. Není nainstalován žádný operační systém, který by mohl útočník využít k dalším útokům [3,4,5,11].

Tyto honeypoty, ale poskytují nejméně informací o útocích. Zaznamenávají jen údaje o provedených akcích (příkazech) a jen část aktivit spojených se službami, jež obsahuje. Takovými údaji jsou většinou čas a datum útoku, jaké služby bylo využito a jejího portu, IP adresu zdroje útoku a cíle nebo zadané příkazy [3,4,5,11].

3.5.2 Honeypoty se střední úrovní interakce

Honeypoty se střední úrovní interakce obsahují větší rozsah funkcí, se kterými je útočníkovi umožněno pracovat než honeypoty s nízkou úrovní. Není jejich součástí zcela funkční nezávislý operační systém, jak je obvyklé u honeypotů s vysokou úrovní integrace. Jsou zpravidla navrženy tak, aby byly schopny rozpoznat škodlivou činnost a reagovat na ni sofistikovaněji než řešení s nízkou úrovní integrace. Příkladem může být honeypot napodobující Microsoft IIS webový server obsahující služby

spojené s běžným IIS serverem. Tento webový server lze nastavit tak, aby obsahoval jakoukoli funkcionalitu, kterou správce požaduje pro monitorování specifických hrozeb. Pokud by na tento systém útočník zaútočil (červ), honeypot útočníkovi poskytne stejné prostředí jako u obyčejného IIS webového serveru. Úroveň interakce je větší než u low-interaction honeypotu, který by představoval např. jen část HTTP kódu na rozdíl od virtuální aplikace [5,12].

Dalším příkladem je honeypot tvořen virtuálním operačním systémem uvnitř jiného operačního systému tzv. chroot jail. Vnitřní virtuální systém se chová a vypadá jako běžný operační systém a lze ho ovládat prostřednictvím toho vnějšího, ve kterém je vytvořen. Poté, co útočník získá kontrolu nad virtuálním systém, mohou být jeho aktivity ovládány a monitorovány zvnějška. Tento přístup je ovšem velmi složitý na implementaci. Je velmi obtížné zařídit, aby simulovaný systém byl co nejpodobnější reálným systémům. Navíc platí, že čím reálnější systém a čím více funkcí virtuální prostředí obsahuje, tím snazší je pro útočníka se z „klece“ dostat a infiltrovat reálný operační systém, v němž se prostředí nachází. Proto většina chroot jail honeypotů není implementována jako plnohodnotný operační systém.

Honeypoty se střední úrovní interakce jsou obvykle náročnější na konfiguraci než honeypoty s nízkou úrovní interakce. Často jsou řešení pouze připravena k úpravě a přizpůsobení podle požadavků jednotlivých organizací. Všechny úpravy musí být provedeny tak, aby nedošlo k zneužití chyb v implementaci útočníkem. Větší možnosti interakce s prostředím znamená větší nebezpečí k vytvoření takových chyb. Časté aktualizace honeypotu jsou žádoucí, neboť stále dochází k vytváření nových hrozeb a postupů k útoku na systém [5,12].

3.5.3 Honeypoty s vysokou úrovní interakce

Honeypoty s vysokou úrovní interakce umožňují získat největší množství informací o útočnících, ovšem za cenu nejvyšší míry rizika. Jsou zároveň nejnáročnější na vytvoření a údržbu. High-intraction honeypoty poskytují útočníkům celé reálné operační systémy bez jakýchkoli restrikcí. Tyto honeypoty umožňují organizacím objevovat nové možnosti obrany, chyby v operačních systémech, bezpečnostní díry, chyby v aplikacích a prostředky, jak útočníci se systémem jako takovým pracují [5,12,14,17].

Aby byl systém co nejvěrohodnější, nesmí v něm být provedeny téměř žádné změny. Často jsou naprosto totožné systémům běžně používaných ve světě. Jediným rozdílem mezi honeypotem a běžným systémem je jeho nulová produkční hodnota. Jeho jediným účelem je být cílem útoku, což znamená velká rizika provozu honeypotu. Po získání přístupu na systém může útočník pokračovat ve své zákeřné činnosti např. útočit na jiné systémy v organizaci, zachycovat produkční aktivitu [5,12,14,17].

Nejčastěji jsou tyto honeypoty umístěny v kontrolovaném prostředí tak, aby nebylo možné zaútočit na jiné systémy např. za firewallem. Firewally většinou slouží jako prostředek, jak udržet útočníka pouze v daném kontrolovaném prostředí a obsahují speciální přístupová pravidla [5,12,14,17].

Prostředí jsou velmi náročná na instalaci a konfiguraci. Musí obsahovat řadu různých technologií, např. firewally, IDS (systémy pro odhalení průniku). Všechny zařízení musí být pečlivě nakonfigurována a jejich častá údržba stejně jako neustálé monitorování operací prováděných na honeypotu je žádoucí. Čím větší interakce je útočníkovi umožněna, tím větší je riziko. Pokud je honeypot s vysokou úrovní interakce implementován správně a bez bezpečnostních chyb, poskytuje největší množství dat a nejcennější informace o chování útočníka ze všech druhů honeypotů [5,12,14,17].

4 Srovnání existujících implementací honeypotů

Tato kapitola bude zaměřena na srovnání existujících implementací honeypotů. Údaje v tabulce a popisy jednotlivých řešení byly převzaty především z online zdroje Awesome Honeypots od autora Jose Nazario [18]. Informace jsou aktuální ke dni 12. 4. 2020. Tabulku jsem zpracoval na základě analýzy informací, kterou jsem provedl v rámci rešerše vztahující se k této bakalářské práci.

Kvalitu dokumentace jsem hodnotil podle její propracovanosti, srozumitelnosti na třístupňové stupnici špatná, dobrá, velmi dobrá.

Tabulka 1: Srovnání existujících implementací honeypotů

Název	Typ služby	Úroveň integrace	Techno-logie	Poslední aktualizace	Způsob konfigurace	Kvalita dokumentace	Platforma
Tanner/Snare	Web	Nízká	Python	Duben 2020	Python	Dobrá	Linux
Servletpot	Web	Nízká	Java	Květen 2013	Bash	Špatná	Linux
Shadow Daemon	Web	Vysoká	PHP, Perl, Python	Březen 2016	UI	Velmi dobrá	Linux
HoneyHTTPD	Web	Nízká	Python	Květen 2018	Python, JSON	Dobrá	Linux
MTPot	Web	Nízká	Python	Březen 2017	JSON	Dobrá	Linux
phpMyAdmin Honeypot	Web	Nízká/ Střední	PHP	Červenec 2015	PHP	Dobrá	Linux
Shockpot	Web	Nízká	Python	Prosinec 2015	Python	Dobrá	Linux
Cowrie	SSH/ Telnet	Střední	Python	Září 2018	Python	Velmi dobrá	Linux
Kippo	SSH	Střední	Python	Srpen 2016	Python	Dobrá	Linux, Windows
Kojoney	SSH	Nízká	Python	Únor 2010	Python	Velmi dobrá	Linux
Kojoney2	SSH	Nízká	Python	Leden 2015	Python	Velmi dobrá	Linux
SSH-honeypotd	SSH	Nízká	C	Srpen 2018	C	Špatná	Linux
Mailoney	SMTP	Nízká	Python	Květen 2018	Python	Dobrá	Linux
Conpot	Server	Nízká	Python	Září 2018	Python	Velmi dobrá	Linux

4.1 Popis jednotlivých řešení honeypotů

Tanner/Snare

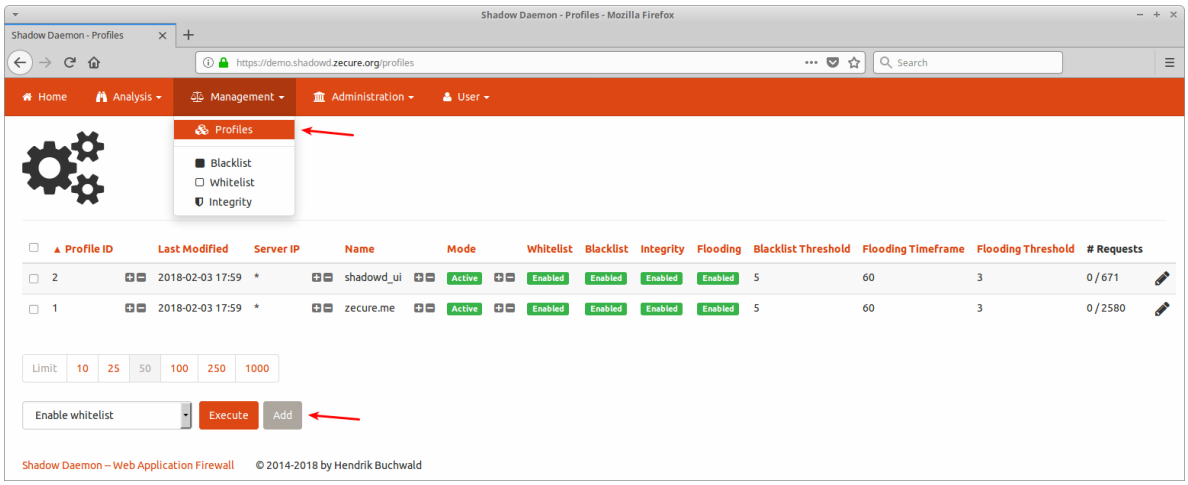
Snare (Super Next generation Advanced Reactive honEypot) je webový honeypot s nízkou úrovní interakce, jehož cílem je zjišťovat informace o útocích prostřednictvím internetu (hlavně protokolu HTTP). Webová stránka je vytvořena klonováním reálné webové aplikace a přidáním známých bezpečnostních chyb. Tanner slouží jako nástroj pro datovou analýzu, zpracování HTTP požadavků a odpovědí pro zprávy poskytnuté ze Snare. Honeypot je implementován v Pythonu a testován na platformě Linux [7].

Servletpot

Honeypot s nízkou úrovní interakce implementován jako webová aplikace v jazyce Java. HTTP komunikace je vytvořena pomocí Apache HttpClient knihoven a MySQL komunikace je vytvořena prostřednictvím MySQL konektoru od Oracle. Honeypot nemá příliš propracovanou dokumentaci, je velmi stručná [18].

Shadow Daemon

Jedná se o honeypot s vysokou úrovní interakce. Slouží k detekci, analýze a zabránění útoků na webové aplikace. Funguje jako aplikační firewall, který filtruje požadavky s parametry útoků. Je open source, implementován v několika jazycích (Perl, PHP, Python) a lze jej konfigurovat přes vlastní uživatelské rozhraní. Velmi kvalitně zdokumentován na vlastních webových stránkách [19].



Obrázek 3: Shadow Deamon UI

HoneyHTTPD

Honeypot představující webový server. Uživatel má možnost vytvořit několik falešných webových serverů a monitorovat na něj zasílané požadavky. Je implementován v Pythonu a konfigurován pomocí JSON souborů [18].

MTPot

MTPot byl vytvořen pro monitorování útoků Mirai, který využívá vzdáleně ovládaných síťových zařízení (botů) k rozsáhlým síťovým útokům (např. DDoS). Zaznamenává druh Mirai útoku, zdrojovou IP adresu a délku spojení. Je Implementován v Pythonu [9,20].

phpMyAdmin Honeypot

Lze jej upravit tak, aby fungoval jako honeypot s nízkou úrovní interakce, ale i se střední úrovní interakce. Jedná se o falešnou webovou aplikaci, je velmi jednoduchý, implementovaný v PHP. Monitoruje jen příchozí požadavky [18].

Shockpot

Slouží k detekci Shellshock (Bashdoor) útoku, což je bezpečností chyba v unixovém bash shell, přes niž může útočník získat neautorizovaný přístup do systému. Je velmi jednoduchý, nemá prakticky žádné jiné využití. Je implementován v jazyce Python [9].

Cowrie

Honeypot se střední úrovní interakce zaměřený na monitorování Brute Force útoků přes protokoly SSH a Telnet. Obsahuje falešný souborový systém ve vlastním virtuálním prostředí. Rozšiřuje Kippo několika novými prvky: SFTP a SCP podpora, SSH exec příkazy, přeposíláni SMTP připojení SMTP honeypotům [18].

Kippo

Kippo je předchůdce Cowrie. Funguje stejně jako Cowrie má vlastní virtuální prostředí a souborový systém, ale neobsahuje rozšířené prvky popsané v předchozím odstavci. Je implementován v Pythonu [18].

Kojoney

Honeypot s nízkou úrovní interakce slouží k monitorování Brute Force útoku emulováním SSH serveru implementovaném v Pythonu (Twisted knihovny). Byl inspirací pro vytvoření Kippo [21].

Kojoney2

Na rozdíl od Kojoney se jedná o honeypot se střední úrovní interakce. Kojoney2 simuluje SSH prostředí, které umožní útočníkovi autentifikaci a práci v něm. Dovoluje útočníkovi stahování a nahrávání falešných souborů. Zaměřuje se na zaznamenávání fingerprintingu útoku. Je stejně jako Kojoney implementován v Pythonu s využitím Twisted knihoven [21].

SSH-honeypotd

Velmi jednoduchý honeypot s nízkou úrovní interakce. Stejně jako ostatní SSH honeypoty se zaměřuje na zaznamenávání informací o Brute Force útocích. Loguje IP adresy, přihlašovací jména a hesla. Neobsahuje vlastní virtuální prostředí jako např. Cowrie, Kojoney [18].

Mailoney

SMTP honeypot, který je určen pro monitorování spamu a pokusu o přihlášení. Je implementován v Pythonu. Není příliš podrobně popsán [18].

Conpot

Conpot je joneypot s nízkou úrovní interakce sloužící pro monitorování řídících systémů (ICS – Industrial Control Systems). Je navržený tak, aby bylo možné vytvářet složité infrastruktury

představující falešné průmyslové komplexy (letiště, montážní linky). Je implementován v Pythonu a lze jej nainstalovat i na Docker [9,22].

5 Použité technologie

5.1 Raspberry Pi

Raspberry Pi je série malých jednodeskových počítačů vytvořených anglickou organizací Raspberry Pi Foundation. Cílem organizace je seznámit a ulehčit učení práce na počítači prostřednictvím velmi levné alternativy k obyčejným stolním počítačům. Raspberry Pi dnes slouží např. ve školách pro děti k výuce programování jazyků Scratch nebo Python. První verze (Raspberry Pi 1 Model B) byla představena v únoru roku 2012 [23]. Momentálně je dostupná již třetí generace těchto jednodeskových počítačů v několika verzích.

Raspberry Pi Foundation vytvořila vlastní linuxovou distribuci Raspbian (jádrem je Debian), což umožňuje využití velkého množství dostupného software, často open source. Lze jejich prostřednictvím vytvořit řadu zajímavých projektů např. Raspberry Pi směrovač, server nebo domácí stanice.

V rámci vypracování bakalářské práci byl využit modelu Raspberry Pi 3 Model B+. Model obsahuje čtyřjádrový procesor ARM Cortex-A53 s podporou 64bitových instrukcí, 1 GB LPDDR2 SDRAM, 2.4 GHz a 5 GHz WiFi 802.11 b/n/g/ac, grafický čip VideoCore IV (300 MHz). Datovým uložištěm je MicroSDHC karta SanDisk Ultra s kapacitou 32 GB. Výhodou tohoto modelu je zpětná kompatibilita se staršími generacemi. Jako operační systém byl zvolen OS Rasbian 4.14.

Obrázek 4: Raspberry Pi 3 Model B+

5.2 Java

Java je dnes jeden z nejpoužívanějších objektově orientovaných programovacích jazyků. Její vývoj začal ve firmě Sun Microsystems již v roce 1991. Největší předností tohoto programovacího jazyka měla být už při jeho prvním uvedení přenositelnost. Jakýkoli program by měl být zpracovatelný a spustitelný nezávisle na platformě, na které je používán. Později byla tato přednost využita i pro tvorbu různých přenosných programů pro prostředí internetu. Hlavním principem pro tak rozsáhlou přenositelnost je vytvoření tzv. bajtkódu, což je optimalizovaná sada instrukcí, navržena pro běhové prostředí Javy, kterým je virtuální stroj (Java Virtual Machine – JVM) [24]. Dnes je v Javě implementováno velké množství a široká škála druhů programů od uživatelských rozhraní operačních systémů (Android) až po počítačové hry (Minecraft).

5.3 IntelliJ IDEA

IntelliJ IDEA je integrované vývojové prostředí pro programování aplikací napsaných v různých programovacích nebo skriptovacích jazycích např. Java, Kotlin, XML, Groovy, Python, SQL. Je vyvíjen firmou Jet Brains ve dvou verzích: Ultimate a Community. Pro tvorbu této bakalářské práce byla využita jinak placená Ultimate edice, kterou však mohou studenti vysokých a některých středních škol po autorizaci své ISIC karty využívat zdarma. Community edice je volně dostupná jako open source software, avšak neobsahuje některé prvky podporované Ultimate verzí např. podporu při využívání knihovny Spring, SQL, Javascriptu.

5.4 WebStorm

Webstorm je stejně jako IntelliJ IDEA integrované vývojové prostředí od firmy Jet Brains. Slouží k tvorbě webových aplikací, především k vývoji v jazyce JavaScript a jeho knihovnách (Angular, React.js, Vue.js, Node.js). Na rozdíl od IntelliJ IDEA neexistuje Webstorm IDE ve volně stažitelné verzi. I zde lze po autorizaci studenta využívat aplikaci plně zdarma. Aplikace obsahuje řadu pokročilých funkcí: spolupráce se systémy správy verzí, testování jednotlivých modulů, interaktivní konzole.

5.5 React

React.js je knihovna skriptovacího jazyka JavaScript. Slouží k tvorbě interaktivního uživatelského rozhraní webových aplikací. Principem je využití více jednoduchých pohledů na stav jednotlivých komponent a případnou změnu těchto stavů. Zapouzdřením komponent, které si svůj stav upravují podle potřeby, je docíleno snadnější sestavení komplexních řešení pro jakýkoli projekt. Knihovnu React lze využít i v kombinaci s jinými knihovnami jazyka JavaScript jako je např. Node.js [28]. Součástí bakalářské práce bylo využití knihovny React současně s knihovnou Redux. Redux je nezávislá knihovna stavových kontejnerů pro aplikace napsané v jazyce JavaScript.

5.6 Spring

Spring je světově nejpoužívanější Java aplikační framework. Usnadňuje programování Java aplikací, jež obsahují práci s přístupem k datům z databáze (transakce, ORM), jinými Java technologiemi (Java Management Extensions, Java EE Connector Architecture, Java Message Service) nebo umožňuje snadnější a rozšířené testovaní pomocí vlastních škálovatelných testovacích procesů (Spring MVC Test) [27]. V aplikaci jsem využil jednu z mnoha rozšíření Spring frameworku zvanou Spring Boot. To vytváří samostatné aplikace založené na frameworku Spring, které obsahují automaticky nakonfigurované knihovny a závislosti jednotlivých modulů. Dále obsahuje vlastní Tomcat webový server, externí konfiguraci a všechny možnosti svého mateřského rámce.

5.7 Apache Tomcat

Apache Tomcat je webový server pro aplikace tvořené v Java EE. Apache Tomcat je vyvíjen organizací Apache Software Foundation, avšak je umožněno se po krátké registraci na projektu podílet komukoli, jelikož je tento software veden jako open source.

5.8 Java JSch

Java Secure Channel je API, které je implementací protokolu SSH2 v jazyce Java. Jeho součástí je funkce pro připojení na sshd server, přesměrování portů nebo přenos souborů. Funguje na principu vytvoření relace (session), v níž dojde k připojení k SSH serveru, a poté dojde k otevření kanálu, ve kterém lze zadávat příkazy pro daný server. Celá knihovna byla vyvinuta firmou JCraft jako prostředek pro práci s relacemi pro své klienty [25].

5.9 Bootstrap

Bootstrap je sada nástrojů pro tvorbu webových aplikací. Usnadňuje vývoj především frontendové části projektu prostřednictvím různých šablon vytvořených v jazycích HTML, CSS a JavaScript. Obsahuje předpřipravenou stylizaci velkého množství komponent např. tlačítek, hlavních nabídek, textu atd. Původně byl vyvíjen vývojářem z firmy Twitter. Dnes je zdrojový kód tohoto frameworku volně stažitelný z internetu [26]. V dnešní době je velmi populární.

5.10 PostgreSQL

PostgreSQL je open source objektově relační databázový systém. Vývoj začal v roce 1986 jako součást POSTGRES projektu na University of California v Berkeley, USA. K zhotovení této bakalářské práce byl vybrán především díky své dostupnosti, kompatibilitě s řadou operačních systému (Linux, macOS, Windows), podpoře většiny prvků standardu SQL [29]. Pro administraci databáze je využita aplikace pgAdmin 4, která je volně dostupná v balíčku s kompletní PostgreSQL Core distribucí. Verze použitá v této práci je 12.2.

6 Návrh řešení

Tato kapitola se věnuje návrhu vypracování zadání bakalářské práce. Cílem práce bylo naimplementovat jednoduchý honeypot s nízkou (popř. střední) úrovní provázání služeb (low-interaction, medium-interaction) na platformě malého jednodeskového počítače. Honeypot by měl poskytovat běžné aplikační protokoly (HTTP/HTTPs, SSH, SMTP), popř. přesměrovávat provoz na jiný cíl. Pro realizaci jednotlivých služeb je možné využít existující implementaci honeypotu nebo implementovat vlastní řešení.

6.1 Hardware platforma

Pro provoz služeb, které jsou součástí praktické části práce, byl vybrán jednodeskový počítač Raspberry Pi 3 Model B+. V době vypracování práce se jedná o druhý nejvýkonnější typ minipočítače platformy Raspberry Pi. Běhovým prostředím je operační systém Raspbian verze 4.19 s LXDE desktopovým prostředím pro snadnější administraci.

6.2 Zvolené implementace honeypotů

Řešení poskytuje služby aplikačních protokolů HTTP a SSH. Jelikož samotná implementace vlastního řešení jednotlivých honeypotů není účelem této práce, byly použity již vytvořené implementace pro tyto aplikační protokoly. Pro HTTP to jsou Tanner a Snare, pro SSH Cowrie. Všechny byly vybraný kvůli jejich robustnosti a shodě mezi zadáním práce a funkcemi, které každý z nich nabízí.

6.3 Rozhraní

Součástí práce je implementace rozhraní, které slouží jako prostředek pro správu jednotlivých honeypotů, kontrolu nebo analýzu útoků a notifikace o probíhajícím útoku.

Pro tvorbu rozhraní byl zvolen programovací jazyk Java. Knihovna JSch, implementovaná v jazyce Java, slouží k připojení k SSH serveru z aplikace pro snadnější instalaci a konfiguraci honeypotů. Rozhraním je webová aplikace, která je zprovozněna na aplikačním serveru Tomcat. Backend je vytvořen pomocí Java frameworku Spring Boot. Pro komunikaci mezi backendem a frontendem aplikace je použita JavaScript knihovna React.js. Pomocí Bootstrapu, sada nástrojů pro HTML, CSS a JavaScript, byl vytvořeno uživatelské rozhraní aplikace. Jednotlivé hosty a jejich informace (IP adresu, jméno atd.), udržované pro jejich následnou správu, umožňuje aplikace uchovávat v PostgreSQL databázi.

Největší předností navržené implementace je její přenositelnost. Díky vlastnostem jazyka Java a knihoven Spring a React je možné tuto aplikaci nainstalovat a spustit na jakémkoliv systému. Jelikož je běhovým prostředím unixový operační systém a k připojení používá protokol SSH, jsou všechny příkazy napsány pro unixový shell, a proto většina funkcí nebude funkčních na jiných operačních systémech.

7 Implementace aplikace

V této kapitole se nachází popis jednotlivých částí vytvořené webové aplikace, jejich funkce a řešení. Kapitoly věnující se implementaci tříd aplikace neobsahují všechny části kódu, jen vybrané zajímavé implementace daných problémů.

7.1 Funkce aplikace

Následující UML diagram případů užití stručně popisuje všechny činnosti, které může uživatel chtít po systému vykonat. V aplikaci figuruje pouze aktér uživatel. Kvůli větší přehlednosti diagramu, jsou všechny honeypoty, jež jsou v aplikaci využívány, nahrazené pouze slovem honeypot. Jedná se o akce: nainstalovat honeypot, odinstalovat honeypot, spustit honeypot, vypnout honeypot, vypsat logy honeypotu. Diagram byl vytvořen pomocí aplikace diagrams.net [31].

Obrázek 5: Diagram případů užití

7.1.1 Případy užití

Najít hosta

Host je systém, na kterém je nasazený honeypot. Tento případ užití je stěžejní pro všechny situace, kdy dochází k manipulaci s daty hostů. Pokud uživatel použije nějakou funkcionalitu přidat hosta, odebrat hosta, upravit hosta, nebo vypsat logy honeypotu, systém se pokusí o vyhledání hosta v databázi. Podle výsledku vyhledání následují různé výstupy, ať už se jedná o požadované provedení akce, nebo vypsání chybové hlášky, pokud systém nemůže požadovanou akci provést.

Přidat hosta

Uživatel může přidat hosta, na kterého chce některý z honeypotů nainstalovat nebo odinstalovat. Musí však zadat identifikační číslo hosta, jeho název, IP adresu, uživatele přidaného do skupiny sudoers, uživatelovo heslo. Všechny údaje, kromě těch přihlašovacích, a navíc implicitně zadaný stav všech honeypotů na nenainstalován, jsou následně uložené do PostgreSQL databáze.

Upravit hosta

Uživatel může kdykoli upravit jednotlivé hosty uložené v databázi. Nemůže však změnit jejich identifikační číslo, které slouží v tabulce jako unikátní klíč.

Odebrat hosta

Stejně jako má možnost uživatel hosta honeypotu přidat, může jej i odebrat. Uživateli stačí vybrat hosta podle identifikačního čísla a aplikace kompletně vymaže jeho záznam z databáze.

Vypsat logy honeypotu

Aplikace umožňuje zobrazit logované události honeypotů pro další analýzu. Uživateli stačí vybrat honeypot, jehož logy si chce vypsat, a aplikace je zobrazí.

Nainstalovat honeypot

Tato funkce zprostředkovává plnou instalaci všech druhů honeypotů (Cowrie, Tanner, Snare). Podle toho, který honeypot si uživatel vybere, se honeypot nainstaluje na uvedeného hosta. Také se změní jeho stav v databázi na nainstalován. Jestliže je již vybraný honeypot nainstalován, provede se pouze jeho aktualizace, případně se nic nestane.

Odinstalovat honeypot

Aplikace také obstarává odinstalaci honeypotů. Pokud je vybraný honeypot na hostu nainstalován, odinstaluje ho. Stejně jako u předchozího případu užití se změní stav v databázi, tentokrát na nenainstalovaný. Pokud není nainstalován, nic se nestane.

Spustit honeypot

Po instalace nebo startu systému je vhodné vybraný druh honeypotu spustit, aby mohl vykonávat svou funkci. Jelikož má každý honeypot jiný styl spouštění, je opět nutné zadat, který honeypot si uživatel přeje prostřednictvím aplikace nastartovat. Zároveň se změní stav spuštěného honeypotu v databázi na spuštěný. Pokud je honeypot ve stavu spuštěný, nelze ho odinstalovat.

Vypnout honeypot

Pokud si uživatel rozhodne spuštěný honeyspot vypnout, existuje v aplikaci i tato možnost. Služba honeypotu se vypne a honeypot přestane logovat události. Stejně jako v případě nainstalování honeypotu se změní stav v databázi na vypnutý. Aplikace pak umožňuje honeypot odinstalovat nebo opět spustit.

7.2 Ukládání dat o hostu

7.2.1 Databáze honeypotapp

Součástí vytvořené aplikace je ukládání informací o honeypot hostu do PostgreSQL databáze. Toto řešení bylo vybráno především kvůli snadnější administraci a manipulaci s daty v rámci aplikace. Především kvůli změnám stavů jednotlivých honeypotů, a také pro větší praktičnost pro uživatele, protože data jsou uložena a opět přístupná pro další práci. Byl vybrán open source PostgreSQL ve verzi 12.2. Jedná se o rychlý, pro účely této práce vhodný, objektově relační databázový systém.

Databáze, pojmenovaná honeypotapp, je vytvořena při prvním spuštění aplikace na webovém serveru prostřednictvím frameworku Hibernate. Hibernate je framework zprostředkovávající objektově relační mapovaní. V aplikaci figuruje jen jako jednoduchý způsob, jak založit databázi, přidat hosta a odebrat hosta. Vyznačuje se vysokým výkonem, nevyžaduje žádné speciální databázové tabulky a jeho schopnost udržovat persistenci dat, funkce podpory objektově orientovaných tříd a Java kolekcí [30], jsou více než dostačující pro požadovanou funkcionalitu. V aplikaci je tabulka Host reprezentovaná třídou *Host* v balíčku *repositor.model*. Obsahuje všechny atributy tabulky tak, aby byla správně vytvořena přes ORM Hibernate.

Pro správnou funkci programu je nutné na lokálním PostgreSQL serveru vytvořit databázi s názvem honeypotapp. Bez tohoto kroku nebude aplikace správně fungovat. Po založení databáze se při prvním spuštění aplikace automaticky vytvoří tabulka host. Dále je žádoucí, aby databáze obsahovala uživatele *postgres* s heslem *password*. Údaje jsou takto nastaveny v konfiguračním souboru knihovny Spring. Je možné je změnit, ale pouze s přístupem ke kódu. Následuje výpis ze Spring konfiguračního souboru application-dev.yml:

```
spring:
  datasource:
    driver-class-name: org.postgresql.Driver
    url: jdbc:postgresql://127.0.0.1:5432/honeypotapp
    username: postgres
    password: password
```

Obrázek [6] zobrazuje tabulku Host a celý ER diagram databáze honeypotapp. Za ní následuje stručný popis všech atributů. Diagram byl vytvořen pomocí aplikace diagrams.net [31].

Host	
PK	**hostid**
*	hostname
*	ipaddress
*	cowrie
*	snare
*	tanner

Obrázek 6: Tabulka host

- **Hostid** – Atribut hostid je primárním klíčem tabulky. Označuje identifikační číslo, které má každý host unikátní. Má nastavenou funkci automatické inkrementace, aby nedocházelo k nežádoucím duplicitním výskytům (např. při špatném zadání čísla uživatele).
- **Hostname** – Hostname označuje jméno nebo název hosta. Jedná se o povinný údaj typu character varying.
- **Ipaddress** – Atribut je záznam o IP adrese hosta, která je nutná k připojení přes SSH, které je stěžejní pro aplikaci. Bez ní není možné nainstalovat žádný honeypot. IP adresa je povinný údaj datového typu character varying.
- **Cowrie** – Atribut Cowrie určuje, v jakém stavu funkčnosti se právě nachází honeypot Cowrie na daném hostu. V aplikaci je implementován enumerátor *State*, který může nabýt hodnoty: DEPLOYED, NOT_INSTALLED, STARTED, NOT_STARTED. Tento enumerátor slouží k záznamu stavu, v jakém se právě honeypot na hostu nachází. DEPLOYED je stav po kompletní první instalaci honeypotu, NOT_INSTALLED je nenainstalován, tedy stav před instalací, STARTED nastává po startu honeypotu a NOT_STARTED po jeho vypnutí. Jelikož se jedná o řetězec, je atribut typu character varying.
- **Snare** – Stejně jako atribut Cowrie i atribut Snare stanovuje stav honeypotu. Stejně jako Cowrie může honeypot být ve stavu DEPLOYED, NOT_INSTALLED, STARTED, NOT_STARTED. Atribut je typu character varying.
- **Tanner** – Pro atribut Tanner platí stejná omezení jako pro atributy Cowrie a Snare. Jediná změna je opět druh honeypotu, atribut označuje stav honeypotu Tanner.

Obrázek 7: Uživatelské rozhraní pro přidání hosta do databáze

7.2.2 Přihlašovací údaje

Aby bylo možné vykonat instalace a administraci implementací honeypotů prostřednictvím služby SSH, je nutné znát přihlašovací údaje účtu na stanici, na kterou je požadován přístup. Je nutné, aby uživatel byl součástí skupiny sudoers, protože dochází k vytváření složek, instalaci a konfiguraci instalačních souborů každého honeypotu. Tyto operace nejsou ve většině případů povoleny uživatelům bez oprávnění správce (root). Zároveň je žádoucí, aby uživatel nemusel pro každou akci provedenou příkazem sudo zadávat své heslo, jelikož během testování došlo k chybě při vkládání hesla do příkazů. Heslo nebylo rozpoznáváno a příkaz se jednoduše neprovedl.

Toho, aby uživatel nemusel zadávat heslo při každém užití příkazu sudo, lze docílit upravením konfiguračního souboru */etc/sudoers*. První možností je povolit všem uživatelům, jenž jsou členy skupiny sudo nebo jiné skupiny, používat sudo bez hesla přidáním následujícího řádku do konfiguračního souboru využitím příkazu sudo visudo:

```
%skupina        ALL=(ALL)     NOPASSWD: ALL
```

Druhou alternativou je povolit tuto funkci jen jednomu uživateli. Opět po příkazu sudo visudo je nutné do konfiguračního souboru /etc/sudoers přidat následující řádek:

```
uzivatel        ALL=(ALL)     NOPASSWD: ALL
```

Bez tohoto nastavení nebude možné v aplikaci nainstalovat a administrovat žádný honeypot. Při spouštění honeypotu se však z bezpečnostních důvodů (např. útočníkův pokus opustit honeypot a získat přístup do systému hosta) zvýšeného oprávnění nevyužívá. Cowrie tuto možnost dokonce ani neumožňuje.

7.2.3 Funkce cacheManager

Vzhledem k bezpečnosti každého systému je žádoucí, aby přihlašovací údaje nebyly snadno dostupné, a to ať už při jejich ukládání, tak při jejich zasílání na cíl. Z toho důvodu nejsou uloženy spolu

s ostatními informacemi o hostu v databázi honeypotapp. Implicitní nastavení v kódu aplikace je také nevhodné, jelikož by bylo obtížné změnit přihlašovací údaje bez přímého přístupu do kódu aplikace.

Pro ukládání hesla a uživatele je v práci použito ukládání do cache aplikace. Data jsou uchovávána v operační paměti serveru, na kterém je aplikace spuštěna. Nevýhodou této metody uchovávání dat je jejich zánik při vypnutí serveru, což znamená, že pokud například bude nutný restart systému, musí uživatel zadat přihlašovací údaje hosta znovu. Nejedná se o nejbezpečnější řešení ukládání informací, zvolený byl především díky jeho jednoduchosti. Přečtení operační paměti, respektive způsob, jak by mohl potenciální útočník tyto data získat, je relativně náročný proces. Nabízí se řada jiných řešení, např. šifrované zasílání dat, nebo šifrování dat přímo v databázi. Při tvorbě aplikace bylo vyzkoušeno šifrování dat v databází pomocí PostgreSQL knihovny pgcrypto, které však nebylo úspěšné kvůli špatnému dešifrovaní hesel při připojení na hosta.

Pro tento účel je v rámci aplikace vytvořena funkce cacheManager, jež je součástí třídy BeanConfig.

```
public CacheManager cacheManager(){
    SimpleCacheManager appCache =  new SimpleCacheManager();
    appCache.setCaches(Arrays.asList(new ConcurrentMapCache("appcache")));
    return appCache;
}
```

7.3 Třída SSHConnection

Třída SSHConnection je stěžejní pro funkčnost celé aplikace. Její součástí jsou kompletní instalace a administrace jednotlivých honeypotů, využití dříve popsané knihovny JSch a funkce pro získání textu ze systémového výstupu při provedení příkazu.

Metoda conn

Metoda conn (String... commands) slouží k provedení příkazu na hostovi, pokud jsou dostupné přihlašovací údaje. Jejími parametry je kolekce řetězců, aby bylo možné nahrát více řetězců najednou. Funguje na principu založení SSH relace. Po založení SSH relace je otevřen kanál, do kterého jsou vloženy požadované příkazy, ty jsou následně postupně vkládány do vstupního datového proudu a provedeny. Po provedení všech příkazů je kanál odpojen. Po odpojení kanálu končí i relace. Součástí funkce je také výpis chybové hlášky a potvrzení o správném provedení. Tyto výpisy jsou však viditelné pouze na výstupu konzole vývojového prostředí.

Metodu conn využívají všechny metody pro administraci a instalaci honeypotů.

```
public String conn(String... commands) {
    String out = new String();
    try {
        Properties config = new Properties();
        config.put("StrictHostKeyChecking", "no");
        JSch jsch = new JSch();

        Session session = jsch.getSession(user, host, 22);
        session.setPassword(password);
```

```
            session.setConfig(config);

            session.connect();
            System.out.println("Connected...");

            for (String x : commands
            ) {
                ChannelExec channel = (ChannelExec) session.openChannel("exec");

                channel.setCommand(x);
                channel.setErrStream(System.err);

                InputStream in = channel.getInputStream();
                InputStream error = channel.getErrStream();
                channel.connect();
                result = out = getOutput(in, channel);
                String halo = getOutput(error, channel);
                System.out.println(halo);
                System.out.println(getResult());
                in.close();
                error.close();
                channel.disconnect();
            }

            session.disconnect();
            System.out.println("DONE!!!");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return out;
    }
```

7.4 Instalace a administrace honeypotů v aplikaci

Veškeré příkazy pro instalaci a administraci honeypotů byly převzaty a upraveny pro potřeby této aplikace z jejich dokumentace dostupné na stránkách Awesome Honeypots [18].

7.4.1 Cowrie

Cowrie je honeypot se střední úrovní interakce zaměřený na monitorování Brute Force útoků přes protokoly SSH a Telnet. Další funkcí je možnost spuštění v módu s velkou úrovní interakce, kdy funguje jako SSH a Telnet proxy pro monitorování chování útočníka na jiném systému. V rámci této práce je využito funkce se střední úrovní interakce.

Instalace Cowrie

První operací, kterou pravděpodobně bude chtít uživatel provést po přidání hosta do databáze, je instalace honeypotu. Po přeměně stavu honeypotu Cowrie v uživatelském rozhraní ze stavu nenainstalován na stav nainstalován dojde k provedení funkce deployCowrie, která je součástí třídy SSHConnection. Tato třída kompletně připraví honeypot Cowrie ke spuštění. Součástí metody deployCowrie jsou další funkce, jež budou popsány v další části kapitoly. Všechny metody mění

proměnnou command na řetězec podle jejich funkce, ten se pak provede v systému jako příkaz využitím metody conn. Ta již byla popsána v samostatné kapitole věnující se třídě SSHConnection.

```
public void deployCowrie() {
            cowrieInstall();
                conn(command);
            updateCowrie();
                conn(command);
            virtualCowrie();
                conn(command);
            activateCowrie();
                conn(command);
}
```

Metoda cowrieInstall

Metoda cowrieInstall instaluje podporu pro virtuální prostředí Python verze 3 včetně balíčků potřebných k instalaci samotného Cowrie. Je nutné, aby byl na systému předem nainstalován Python 3.

Metoda updateCowrie

Tato metoda vytváří složku pro instalaci v domovském adresáři a změní vlastníka vytvořené složky na uživatele, který je právě přihlášen prostřednictvím aplikace. Toto je nutné ke správnému provedení dalších kroků instalace. Při vytváření složky příkazem sudo dochází k změně vlastníka na uživatele root, což znamená, že vlastníkem všech ostatních souborů založených nebo vytvořených v dané složce je také uživatel root. Při testovacích instalacích nastávala nežádoucí situace. Pokud chtěl kdokoli jiný např. aktualizovat Cowrie, systém operaci nepovolil s chybovou hláškou Error 13: Povolení zamítnuto. Tato situace je vyřešena právě dříve zmíněnou změnou vlastníka složky. Po změně vlastníka složky následuje samotná instalace honeypotu Cowrie ze serveru github.com. Zároveň metoda slouží jako aktualizace Cowrie.

Metoda virtualCowrie

Metoda virtualCowrie vytváří ve vytvořené složce virtuální prostředí pro bezpečné spuštění Cowrie.

Metoda activateCowrie

Poslední metoda, která je součástí metody deployCowrie, aktivuje virtuální prostředí Cowrie a následně nainstaluje, případně aktualizuje, všechny potřebné soubory a balíčky, které jsou zaznamenány v souboru requirements.txt. Tento soubor je stažen již během provedení metody updateCowrie. Stojí zde za zmínku, že příkaz pro spuštění virtuálního prostředí není spuštěn v režimu sudo právě kvůli dříve zmíněnému zákazu spouštění Cowrie v režimu se zvýšenými právy.

```
public void activateCowrie(){
    command = "cd /home/cowrie/cowrie; source cowrie-env/bin/activate; pip install
--upgrade pip; pip install --upgrade -r requirements.txt";
}
```

Spouštění a deaktivace Cowrie

V třídě SSHConnection se nachází jedna metoda pro spuštění honeypotu a jedna metoda pro zastavení běhu honeypotu. Obě metody fungují velmi podobně. Po instalaci se ve složce /cowrie/bin/ nachází soubor cowrie, který pomocí skriptu, vytvořeného tvůrci honeypotu v jazyce Python, zajišťuje spuštění nebo vypnutí služby honeypotu. Po spuštění je Cowrie honeypot připraven monitorovat útoky přes SSH.

```
public void startCowrie(){
    command = "cd /home/cowrie/cowrie;bin/cowrie start";
    conn(command);
}

public void stopCowrie(){
    command = "cd /home/cowrie/cowrie;bin/cowrie stop";
    conn(command);
}
```

Výpis logů Cowrie

Poslední metoda související s honeypotem Cowrie je logsCowrie. Ta využívá výstupu příkazu cat pro výpis souboru, který zasílá na frontend aplikace jako řetězec. Řetězec je následně na frontendu aplikace zobrazen pro uživatele. Zároveň tato metoda slouží k analýze útoků na frontendu aplikace, kde je výpis logů Cowrie součástí administrace honeypotu.

```
public String logsCowrie(){
    command = "cat /home/cowrie/cowrie/var/log/cowrie/cowrie.log";
   return conn(command);
}
```

Přesměrování portu 22

Cowrie operuje ve svém základním nastavení na portu 2222. Aby však honeypot opravdu monitoroval všechny pokusy infiltrovat se přes službu SSH, je třeba použít standardní SSH port 22. Tato možnost není součástí implementace, neboť docházelo ke ztrátě možnosti připojit se přes SSH do systému, pokud byl zároveň spuštěn i honeypot Cowrie, což vedlo především k problémům při testování řešení hlavně ztrátě ostatních funkcionalit aplikace, protože ta vyžaduje port 22 pro připojení prostřednictvím knihovny JSch. Tento problém lze vyřešit přesunutím SSH serveru na jiný port a až následně přesměrovat provoz z portu 22 na port Cowrie (2222). Pro přesměrování komunikace lze použít příkaz pro změnu pravidla firewallu:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
```

7.4.2 Tanner

Dalším honeypotem, který aplikace nabízí k nainstalování, je Tanner. Tanner je vytvořen pro analýzu a kontrolu HTTP požadavků a tvorbu odpovědí na tyto požadavky (především pro honeypot Snare) [18].

Instalace honeypotu Tanner

Kompletní instalace honeypotu Tanner je provedena v metodě deployTanner. Ta začíná nainstalováním Redis serveru. Redis server slouží k ukládání Tanner zpráv do vlastní cache. Není nutný pro správnou funkci, ale pomáhá při vyhodnocování výsledku analýzy útoků. Následuje podobný proces instalace jako u honeypotu Cowrie. Nejdříve systém stáhne nutné soubory pro instalaci prostřednictvím systému správy verzí git. Vytvoří samostatnou složku v adresáři /home, které změní vlastníka na uživatele, jenž je u hosta uveden v přihlašovacích údajích. Poté nainstaluje potřebné soubory a balíčky pro správnou funkci. Nakonec je provedena samotná instalace honeypotu Tanner.

```
public void deployTanner() {
    command = "sudo apt-get install redis-server;redis-server";
    conn(command);
    command = "sudo mkdir /home/tanner;sudo chown " + user + " /home/tanner;cd
/home/tanner;git clone https://github.com/mushorg/tanner.git;";
    conn(command);
    command = "cd /home/tanner/tanner; sudo pip3 install -r requirements.txt;sudo
python3 setup.py install;sudo apt install docker.io";
    conn(command);
}
```

Spouštění a deaktivace honeypotu Tanner

Spouštění a deaktivace honeypotu Tanner je prakticky stejná jako u honeypotu Cowrie.

```
public void startTanner(){
    command = "cd /home/tanner/tanner;sudo tanner &"+"\n";
    conn(command);
}
```

```
public void stopTanner(){
    command = "cd /home/tanner/tanner;bin/tanner stop";
    conn(command);
}
```

Výpis logů honeypotu Tanner

Sám Tanner příliš informací neloguje. Funguje spíše jako doplněk honeypotu Snare, kterému pomáhá logovat informace ze založených relací. Ty lze vyčíst pouze pokud je zároveň nainstalován i Snare.

```
public String logsTanner() {
    command = "cat /opt/tanner/tanner.log";
    return conn(command);
}
```

7.4.3 Snare

Snare (Super Next generation Advanced Reactive honEypot) je webový honeypot s nízkou úrovní interakce, jehož cílem je zjišťovat informace o útocích prostřednictvím internetu (hlavně protokolu HTTP). Snare je honeypot vedený jako webová aplikace, která však obsahuje přidané známé bezpečnostní chyby [18].

Instalace honeypotu Snare

Instalace honeypotu Snare, obsažená v rámci aplikace, je velmi podobná instalaci honeypotu Tanner. Rozdíl nastává po instalaci samotného honeypotu, kdy je naklonována falešná webová stránka, která slouží jako cíl pro útočníka.

```
public void deploySnare() {
    command = "sudo mkdir /home/snare;sudo chown " + user + " /home/snare;cd
/home/snare; git clone https://github.com/mushorg/snare.git";
    conn(command);
    command = "cd /home/snare/snare;sudo pip3 install -r requirements.txt; sudo
python3 setup.py install;sudo clone --target http://example.com";
}
```

Spouštění a deaktivace honeypotu Snare

Při spouštění honeypotu Snare dochází k vytvoření webového serveru na portu 8080, na kterém běží webová stránka test.com stažená během instalace. Pokud by chtěl uživatel používat honeypot v produkčním prostředí, je nutné změnit port na číslo 80, což ale není předmětem této bakalářské práce.

```
public void startSnare() {
    command = "cd /home/snare/snare;sudo snare --port 8090 --page-dir example.com
--tanner localhost &"+"\n";
    conn(command);
}
public void stopSnare() {
    command = "cd /home/snare/snare;bin/snare stop";
    conn(command);
}
```

Výpis logů honeypotu Snare

Logy honeypotu Snare se ukládají do složky /opt. Metodou logsSnare umožňuje aplikace tyto logy snadno zobrazit.

```
public String logsSnare() {
    command = "cat /opt/snare/snare.log ";
```

```
    return conn(command);
}
```

7.5 Třída HoneypotServiceImpl

Třída HoneypotServiceImpl zprostředkovává komunikaci mezi frontendem a backendem aplikace. Obsahuje veškeré akce, které může uživatel provést na webové aplikaci. Implementuje metody z rozhraní HoneypotService. Příkladem mohou být metody pro získání informací o všech honeypotech, ukládání přihlašovacích údajů, instalace honeypotů, výpis logů nebo smazání hosta z databáze.

7.5.1 Metoda updateState

Metoda updateState provádí instalaci, spouštění, vypínání a změnu stavu honeypotů v databázi. Příkladem může být host, na který ještě nebyl nainstalován honeypot Cowrie (Cowrie je v databázi ve stavu NOT_INSTALLED). Pokud uživatel v aplikaci změní jeho stav na nainstalován (v databázi jako DEPLOYED), metoda updateState honeypot na hosta nainstaluje, změní jeho stav a uloží informace do databáze. Následuje úsek z metody updateState, ve kterém se mění stav ze stavů nespuštěn nebo nainstalován na spuštěn:

```
case STARTED: {
    if (client.getHoneypot() == Honeypot.COWRIE) {
        State old = host.get().getCowrie();
        if (old == State.DEPLOYED || old == State.NOT_STARTED) {
            conn.startCowrie();
            newHost.setCowrie(State.STARTED);
            hostRepository.save(newHost);
        }
    } else if (client.getHoneypot() == Honeypot.SNARE) {
        State old = host.get().getSnare();
        if (old == State.DEPLOYED || old == State.NOT_STARTED) {
            conn.startSnare();
            newHost.setSnare(State.STARTED);
            hostRepository.save(newHost);
        }
    } else if (client.getHoneypot() == Honeypot.TANNER) {
        State old = host.get().getTanner();
        if (old == State.DEPLOYED || old == State.NOT_STARTED) {
            conn.startTanner();
            newHost.setTanner(State.STARTED);
            hostRepository.save(newHost);
        }
    }
}break;
```

7.5.2 Metoda saveConnectionDetails

Další zajímavou metodou v třídě HoneypotServiceImpl je metoda saveConnectionDetails. V ní se vytváří cache popsané v kapitole 7.2.3. Zároveň získává uživatelem zadané informace o hostu ze stránky Přidání hosta. Přihlašovací údaje ukládá do cache a zbytek vkládá do databáze.

```
public void saveConnectionDetails(ConnectionDetails details) {
    Cache appCache = cacheManager.getCache("appcache");
    Credentials credentials = new Credentials(details.getUsername(),
details.getPassword());

    if (appCache == null) {
        throw new RuntimeException("Cache nebyla nalezena!");
    }

    appCache.put(details.getHostID(), credentials);
    Host host = new Host(details.getHostID(), details.getHostname(),
details.getIpaddress(), State.NOT_INSTALLED, State.NOT_INSTALLED,
State.NOT_INSTALLED);
    hostRepository.save(host);
}
```

7.6 Frontendová část aplikace

Frontendová část aplikace je tvořena v jazyce JavaScript rozšířeného o knihovny React.js a Redux. Principem práce s těmito knihovnami je využití více jednoduchých pohledů na stav jednotlivých komponent a případnou změnu těchto stavů. Většina metod je si relativně podobná, proto je zde vybrán způsob přidání a úprava informací o hostu jako vzor. Samotné komponenty na stránce, jako jsou například tlačítka, jsou tvořeny v HTML a stylovány v CSS.

7.6.1 Přidání a úprava hosta ve frontendové části

Tato podkapitola popisuje jednotlivé části funkcionality přidávání a upravování informací o hostech. Ta je rozdělena do více souborů, které se nacházejí v kontejneru Configuration.

Soubor actions.js

V souboru actions.js se nachází akce, které vyvolávají změnu v aplikaci. Pro přidání a úpravy hosta je to akce SaveConfiguration:

```
export const saveConfiguration = (configuration)=>({
    type: API_SAVE_CONFIGURATION,
    payload: configuration
})

export const handleChange = (attributName,attributValue)=>({
    type: HANDLE_CONF_CHANGE,
    attributName,
    attributValue
})
```

Soubor constants.js

Tento soubor obsahuje definice typů akcí ze souboru actions.js. Pro již dříve zmíněné akce to jsou:

```
export const HANDLE_CHANGE = "HANDLE_CHANGE";
export const GET_HOSTS = "GET_HOSTS";
export const GET_HOSTS_SUCCESS = "GET_HOSTS_SUCCESS";
export const GET_HOSTS_FAILURE = "GET_HOSTS_FAILURE";
```

Soubor saga.js

Redux-saga je knihovna, která zprostředkovává asynchronní funkce jako je získávání dat nebo přístup k cache prohlížeče [32]. Funkce getHosts metodou get, provede požadavek, který je určen podle url v backendové části aplikace. V tomto případě se jedná o zisk dat z databáze.

```
function saveConfig(config) {
         return axios({
         method: 'put',
         url: 'http://localhost:8080/honeypots/connectiondetails',
         data: config,
         contentType: 'application/json'

     })
 }


function* watcherSaga() {
    yield takeLatest(c.API_SAVE_CONFIGURATION, workerSaga);
    yield takeLatest(c.DELETE_HOST,deleteSaga);
}
```

Soubor reducer.js

V souboru reducers se nachází specifikace toho, jak se stav aplikace mění po zpravování vyvolaných událostí. Na rozdíl od akcí reducer určuje změnu stavu aplikace (akce popisují pouze co se má stát) [32].

```
const ConfigurationReducer = (state = initialState, action) => {
    switch (action.type) {
        case HANDLE_CONF_CHANGE:
                return {...state, `${action.attributName}`]:action.attributValue};
        default:
            return state;

    }
};
```

Soubor index.js

Soubor index.js obsahuje vzhled stránky, mapování stavů ze souboru reducer.js, funkce pro provádění akcí a změny stavů a funkce, které provádí události. Metoda handleSaveConfiguration podle stavu formuláře změní svoje chování buď na formulář, ve kterém se přidává host, nebo na formulář ve kterém se host upravuje. Metoda ListHosts vypisuje informace o vybraném hostu do příslušných komponentů. HandleChange je metoda, která po události změny vybraného hosta, změní i zbytek informací tak, aby náležely právě vybranému hostovi.

```
handleSaveConfiguration = (event) => {
    const formSelect = this.props.match.params.mode;


    if (formSelect === "save") {
        this.props.onSaveConfiguration({
            username: this.props.username,
            password: this.props.password,
            hostID: null,
            hostname: this.props.hostname,
            ipaddress: this.props.ipaddress,
        })
    } else if (formSelect === "update") {
        this.props.onSaveConfiguration({
            hostID: this.props.hostID,
            hostname: this.props.hostname,
            ipaddress: this.props.ipaddress,
        })
    }

};

listHosts = () => {
    const hosts = this.props.hosts;
    if (hosts == null) {
        return null;
    }
    hosts.forEach(host => console.log(host))
    return hosts.map(host => (<option value={host.hostID}>{host.hostID}</option>))

}

handleChange = (event) => {
    this.props.handleChange(event.target.name, event.target.value);
    const id = parseInt(event.target.value);

    if (event.target.name === "hostID") {
        const host = this.props.hosts.filter(e => e.hostID === id)[0];
        console.log(typeof (id));
        this.props.handleChange("hostname", host.hostname);
        this.props.handleChange("ipaddress", host.ipaddress);
    }
```

7.7 Spuštění webové aplikace

Prerekvizitami, potřebnými pro spuštění webové aplikace, jsou PostgreSQL databáze a Java.

Kroky pro spuštění webové aplikace

1) Spusťte příkazový řádek
2) Přejděte do složky s frontendovou částí aplikace (pojmenovanou HoneypotAppFrontend).
3) Zadejte příkaz npm install
4) Zadejte příkaz npm run start

Po kroku 4) by mělo dojít ke spuštění webové aplikace

8 Testování honeypot prostředí

Tato kapitola se věnuje testování implementovaného prostředí, především samotným nainstalovaným honeypotům. Všechny honeypoty lze nainstalovat a administrovat na stránce Administrace honeypotů webové aplikace, která je součástí řešení této práce.

8.1 Testování honeypotu Cowrie

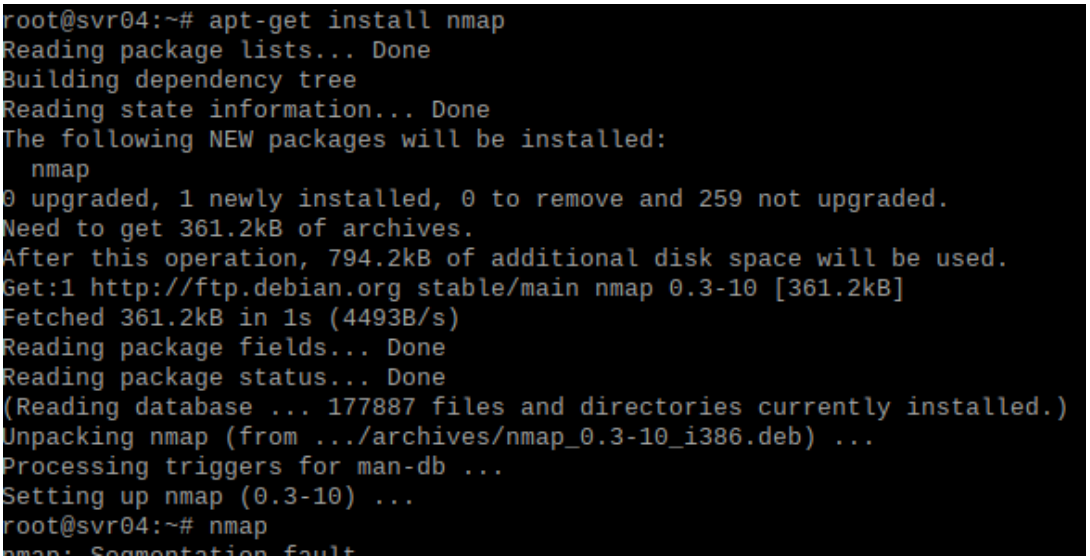
Testování honeypotu Cowrie proběhlo úspěšně. Instalace i spuštění z aplikace proběhlo bez problémů. Následoval test samotného prostředí honeypotu. Připojení přes SSH dopadlo také úspěšně. Nejdříve byly vyzkoušeny příkazy nmap a ifconfig. Příkaz nmap nebyl nalezen a ifconfig se provedl výsledný výpis lze sledovat na obrázku 8.

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@svr04:~# nmap
-bash: nmap: command not found
root@svr04:~# ls -l
root@svr04:~# ls -l /
drwxr-xr-x 1 root root  4096 2013-04-05 13:53 bin
drwxr-xr-x 1 root root  4096 2013-04-05 14:02 boot
drwxr-xr-x 1 root root  3060 2013-04-05 14:03 dev
drwxr-xr-x 1 root root  4096 2013-04-05 14:06 etc
drwxr-xr-x 1 root root  4096 2013-04-05 14:02 home
lrwxrwxrwx 1 root root    32 2013-04-05 13:53 initrd.img -> /boot/initrd.img-
.0-4-686-pae
drwxr-xr-x 1 root root  4096 2013-04-05 14:01 lib
drwx------ 1 root root 16384 2013-04-05 13:52 lost+found
drwxr-xr-x 1 root root  4096 2013-04-05 13:52 media
drwxr-xr-x 1 root root  4096 2013-04-05 13:52 mnt
drwxr-xr-x 1 root root  4096 2013-04-05 13:52 opt
dr-xr-xr-x 1 root root     0 2013-04-05 14:03 proc
drwx------ 1 root root  4096 2013-04-05 14:25 root
drwxr-xr-x 1 root root   380 2013-04-05 14:06 run
drwxr-xr-x 1 root root  4096 2013-04-05 14:03 sbin
drwxr-xr-x 1 root root  4096 2013-04-05 13:52 selinux
drwxr-xr-x 1 root root  4096 2013-04-05 13:52 srv
drwxr-xr-x 1 root root     0 2013-04-05 14:03 sys
drwxrwxrwt 1 root root  4096 2013-04-05 14:17 tmp
drwxr-xr-x 1 root root  4096 2013-04-05 13:52 usr
drwxr-xr-x 1 root root  4096 2013-04-05 13:52 var
lrwxrwxrwx 1 root root    28 2013-04-05 13:53 vmlinuz -> /boot/vmlinuz-3.2.0-
86-pae
root@svr04:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 05:d8:07:65:9f:04
          inet addr:10.152.39.3  Bcast:10.152.39.255  Mask:255.255.255.0
          inet6 addr: fed0::b7:c2ff:fe08:ab01/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:447088 errors:0 dropped:0 overruns:0 frame:0
          TX packets:288001 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:395432528 (395.4 MB)  TX bytes:19940097 (19.9 MB)


lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:110 errors:0 dropped:0 overruns:0 frame:0
          TX packets:110 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:48917378 (48.9 MB)  TX bytes:48917378 (48.9 MB)
```

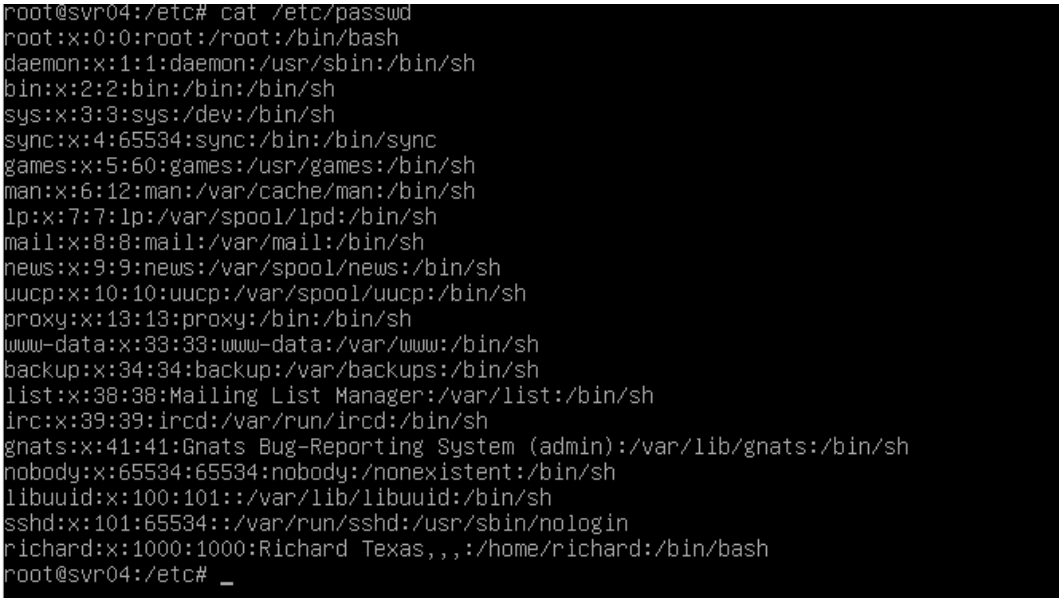
Obrázek 8: Příkazy nmap a ifconfig

Bylo otestováno i samostatné stáhnutí balíčku s příkazem nmap, které se zdálo jako úspěšné. Poté, co byl však příkaz použit znovu, vyskytla se chyba paměťové ochrany (obrázek 9)



Obrázek 9: Příkaz nmap

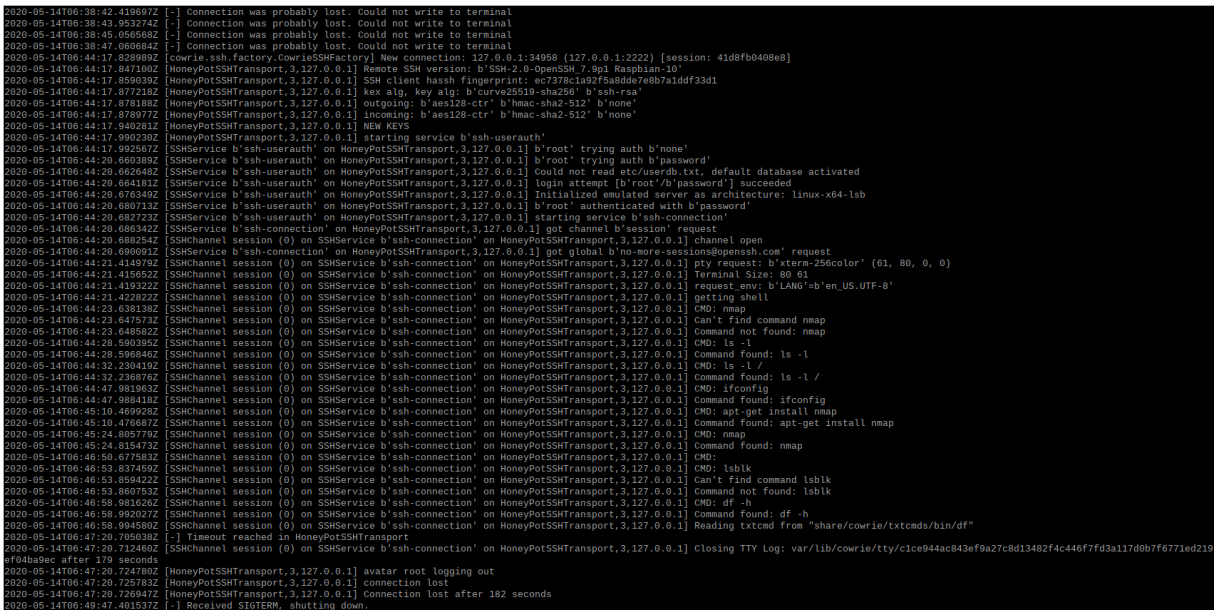
Při startu v módu emulované konzole shell, což je případ využitý v této bakalářské práci, je vytvořen falešný souborový systém, který umožňuje útočníkovi přidat nebo odebrat soubory. Zároveň je vytvořen celý falešný souborový systém představující plnou instalaci operačního systému Debian. Cowrie umožňuje i tento systém rozšiřovat vlastními soubory pro větší důvěryhodnost. Pokud útočník zadá příkaz pro zobrazení obsahu souboru /etc/passwd, který obsahuje informace o uživatelích, zobrazí se mu následující výpis:



Obrázek 10: Výpis souboru /etc/passwd/

Jak je vidět na obrázku 10, falešný konfigurační soubor /etc/passwd se od jeho reálného předchůdce prakticky neliší.

Na obrázku 11 lze vidět výpis z logů celé relace testování. Je možné z nich vyčíst přesný čas, jednotlivé příkazy, které útočník zadal, a dokonce také přihlašovací údaje, pomocí, kterých se útočník snažil získat přístup do honeypotu. V základním nastavení Cowrie umožňuje přihlášení na uživatele root s jakýmkoliv heslem. To to nastavení lze změnit v konfiguračním souboru, ovšem zároveň se snižuje šance, kdy útočník heslo opravdu uhodne, což nemusí vždy být vhodné.



Obrázek 11: Logy útoku na honeypot Cowrie

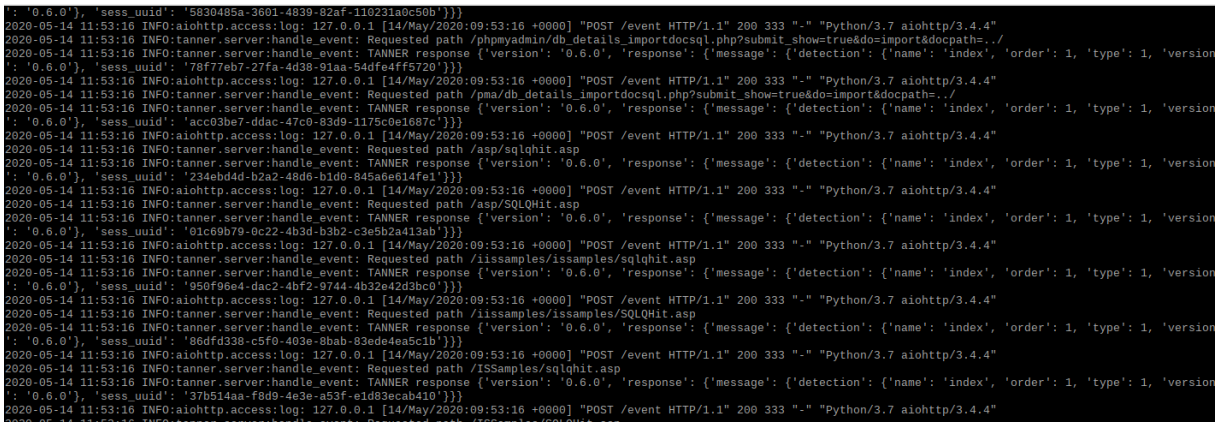
8.2 Testování honeypotů Snare a Tanner

Během tvorby této bakalářské práce byly honeypoty Snare a Tanner aktualizovány tak, že jsou na sobě zároveň závislé. Bez správně nainstalovaného Tanneru Snare nebylo možné správně spustit. Zároveň tvůrci přidali Tanneru závislosti na souborech součástí balíčku docker.io.

I díky těmto změnám došlo k testování obou honeypotů najedou. Po nasazení na jednodeskový počítač byl nejdříve spuštěn Tanner a až poté Snare. To proto, aby mohl Tanner získávat a zprostředkovávat odpovědi pro Snare.

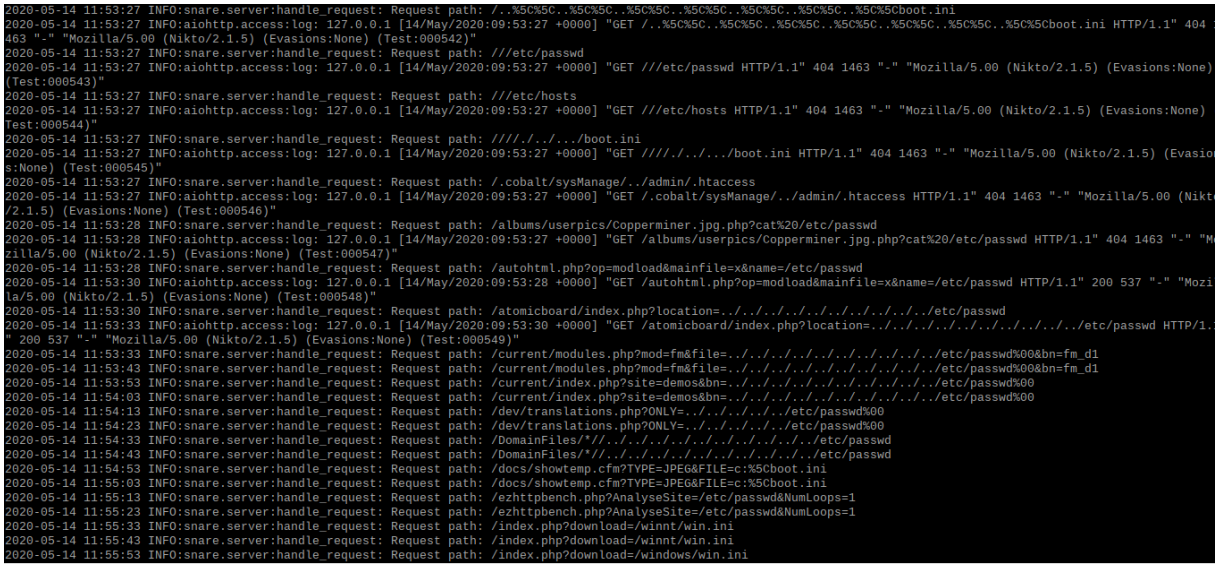
Testování proběhlo nástrojem Nikto. Ten slouží k testování různých špatných nastavení, chybových nastavení, objevování nezabezpečených souborů všech druhů na webovém serveru. Jelikož Snare vytváří falešnou zranitelnou webovou stránku, Nikto je vhodný pro účel jeho testování. Nikto zasílá různé požadavky na serveru a očekává odpovědi. Tím otestuje i honeypot Tanner, protože ten,.jak bylo již zmíněno, zprostředkovává odpovědi, které pak Snare posílá. Zároveň je i zaznamenává do svých logů.

Testování bylo úspěšné, honeypoty se podařilo nainstalovat. I přes problémy, které nastaly při instalaci (chybějící balíčky pro docker, změna funkcionality Snare), se podařilo oba honeypoty i spustit a nástrojem Nikto následně otestovat.



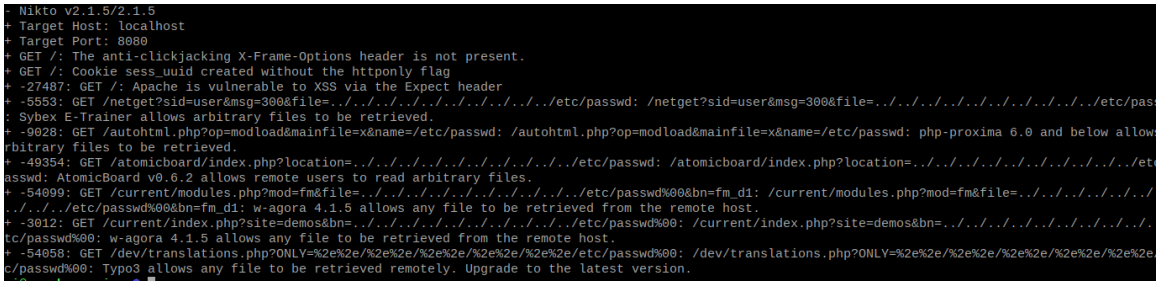
Obrázek 12: Tanner log

Na obrázku 12 je možné si povšimnout odpovědí, které Tanner zasílal na požadavky nástroje Nikto.



Obrázek 13: Tanner log

Na obrázku 13 z výpisu Snare logů jsou zřejmé požadavky nástroje Nikto zaslané na falešný webový server.



Obrázek 14: Výstup testování nástroje Nikto

Na výstupu z Nikto, obrázek 14, jsou vypsány všechny zranitelnosti, které Nikto na falešné webové stránce objevil např. zranitelnou verzi webového serveru Apache,

9 Závěr

Cílem této bakalářské práce bylo nastudovat, jak fungují honeypoty, k čemu slouží a najít různé rozdíly mezi nimi. Na základě rešerše provedené v rámci práce, bylo vytvořeno srovnání existujících řešení honeypotů a jejich stručný popis. Zároveň je popsána stručná historie vývoje honeypotů, využití v praxi, jejich nevýhody i výhody a dělení.

Součástí praktické části bylo navrhnout honeypot s nízkou (střední) úrovní interakce. Byly vybrány tyto honeypoty – Cowrie, Tanner, Snare z důvodů nenáročné instalace a konfigurace.

Webová aplikace, která je součástí řešení, umožnuje nainstalovat, spustit a vypnout různé druhy honeypotů. Díky její přenositelnosti, je aplikaci možné používat na jakémkoliv systému unixové distribuce Debian. Také zprostředkovává možnost administrace více honeypotových hostů najednou, které mohou být součástí honeypotových sítí. Aplikace je vytvořena pomocí v dnešní době moderních technologií, ať už se jedná o frameworky React.js a Redux.js, tak Java framework Spring. Všechny jsou v této době hojně využívány i pro nejnáročnější projekty.

Výsledná implementace se neobešla bez komplikací. Jednalo se například o zabezpečení Cors v prohlížeči Google Chrome, aktualizace řešení honeypotů, kdy jejich změna závislostí a migrace na jiný druh konfiguračních souborů (konfigurační soubory honeypotu Tanner byly změněny na yaml) způsobila změny při samotné instalaci a funkčnosti.

Poslední fází vypracování bylo otestování výsledného řešení. To proběhlo až na výjimky úspěšně. Problémy se vyskytly hlavně při testování honeypotů Tanner a Snare, kdy došlo ke změně jejich konfigurace. Bylo nutné změnit postup implementace.

I přes překonání většiny z popsaných problémů je však stále možné aplikaci vylepšovat. Ať už by se jednalo jak o podporu více druhů honeypotů, tak podporu nejen unixových systémů, větší možnosti konfigurace, nebo méně omezená funkcionalita např. podpora Dockeru. Dále by bylo vhodné zlepšit zabezpečení zasílaných přihlašovacích údajů např. dříve navrhovaným šifrovaným přenosem.

Celkově byly honeypoty zajímavým tématem. I když se dnes nejedná o zvlášť rozšířenou formu zabezpečování před útoky nebo jejich předcházení, stále mají honeypoty své místo v oblasti počítačové bezpečnosti.

Literatura

[1] GRIMES, Roger A. *Honeypots for Windows: An Introduction to Honeypots*. Spojené státy americké: Apress, 2005. ISBN 978-1-4302-0007-9.
[2] The Spy Who Hacked Me. *Infosecurity* [online]. Velká Británie, 2011 [cit. 2020-04-06]. Dostupné z: https://www.infosecurity-magazine.com/magazine-features/the-spy-who-hacked-me/.
[3] GIBBENS, Mathias a Harsha vardhan RAJENDRAN. *Honeypots* [online]. 2012 [cit. 2020-03-29]. Dostupné z: https://www2.cs.arizona.edu/~collberg/Teaching/466-566/2013/Resources/presentations/2012/topic12-final/report.pdf.
[4] SEIFERT, Christian, Ian WELCH a Peter KOMISARCZUK. Taxonomy of Honeypots. *Technical Report CS-TR-06/12* [online]. Victoria University of Wellington, 2006, červen 2006, 1-19 [cit. 2020-04-02]. Dostupné z: http://www.mcs.vuw.ac.nz/comp/Publications/CS-TR-06-12.abs.html.
[5] SPITZNER, Lance. *Honeypots: Tracking Hackers*. Spojené státy americké: Addison-Wesley, 2002. ISBN 0-321-10895-7.
[6] Deception ToolKit. *All.Net* [online]. Spojené státy americké, 2019 [cit. 2020-03-30]. Dostupné z: http://www.all.net/dtk/.
[7] The Honeynet Project. *The Honeynet Project* [online]. Spojené státy americké, c1999-2019 [cit. 2020-04-02]. Dostupné z: https://www.honeynet.org/.
[8] DOMINGUEZ, Andrea. The State of Honeypots: Understanding the Use of Honey Technologies Today. *SANS Institute* [online]. listopad 2017, 1-48 [cit. 2020-04-02]. Dostupné z: https://www.sans.org/reading-room/whitepapers/detection/paper/38165.
[9] Symantec. *Symantec* [online]. Spojené státy americké, 2019 [cit. 2020-01-28]. Dostupné z: https://www.symantec.com.
[10] DAHNBUL, R. N., C. LIM a J. PURNAMA. Enhancing Honeypot Deception Capability Through Network Service Fingerprinting. *Journal of Physics: Conference Series* [online]. 2017, 1-6 [cit. 2020-04-02]. DOI: 10.1088/1742-6596/801/1/012057. Dostupné z: https://iopscience.iop.org/article/10.1088/1742-6596/801/1/012057.
[11] MOKUBE, Iyatiti a Michele ADAMS. Honeypots: Concepts, Approaches, and Challenges. *ACM-SE 45 2007* [online]. 2007, 321-326 [cit. 2020-04-01]. DOI: 10.1145/1233341.1233399. Dostupné z: https://dl.acm.org/doi/10.1145/1233341.1233399.
[12] KEONG NG, Chee, Lei PAN a Yang XIANG. *Honeypot Frameworks and Their Applications: A New Framework* [online]. Singapur: Springer, 2018 [cit. 2020-03-29]. ISBN 978-981-10-7739-5. Dostupné z: https://link.springer.com/book/10.1007%2F978-981-10-7739-5#about.
[13] HECKER, Christopher, Kara L. NANCE a Brian HAY. Dynamic Honeypot Construction. *Proceedings of the 10th Colloquium for Information Systems Security Education* [online]. University of Maryland, 2006, 95-102 [cit. 2020-04-01]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.127.5761.
[14] PROVOS, Niels. Honeyd: A Virtual Honeypot Daemon: (Extended Abstract). *Proceedings of the 10th Colloquium for Information Systems Security Education* [online]. Center for Information Technology Integration University of Michigan, 2003, 1-7 [cit. 2020-04-01]. Dostupné z: https://www.researchgate.net/publication/250395424_Honeyd_A_Virtual_Honeypot_Daemon_Extended_Abstract.
[15] ALOSER, Yaser a Omer RANA. Honeyware: A Web–Based Low Interaction Client Honeypot. *Third International Conference on Software Testing, Verification, and Validation*

Workshops [online]. School of Computer Science & Informatics, Cardiff University, 2010, 410-417 [cit. 2020-04-01]. DOI: 10.1109/ICSTW.2010.41. Dostupné z: https://www.researchgate.net/publication/220719975_Honeyware_A_Web-Based_Low_Interaction_Client_.
[16] MAHAJAN, Surendra, Akshay Mhasku ADAGALE a Chetna SAHARE. Intrusion Detection System Using Raspberry PI Honeypot in Network Security. *International Journal of Engineering Science and Computing* [online]. 2016, 2792-2795 [cit. 2020-04-01]. DOI: 10.4010/2016.651.
[17] ALATA, E., V. NICOMETTE, M. KAÂNICHE, M. DACIER a M. HERRB. Lessons learned from the deployment of a high-interaction honeypot. *Proceedings of the Sixth European Dependable Computing Conference* [online]. 2006, 1-6 [cit. 2020-04-01]. DOI: 10.1109/EDCC.2006.17. Dostupné z: https://www.researchgate.net/publication/224674327_Lessons_learned_from_the_deployment_of_a_high-interaction_honeypot.
[18] NAZARIO, Jose. Awesome Honeypots. *GitHub* [online]. Spojené státy americké, 2020 [cit. 2020-03-30]. Dostupné z: https://github.com/paralax/awesome-honeypots.
[19] Shadow Deamon. *Shadow Deamon* [online]. Spojené státy americké, 2020 [cit. 2020-03-29]. Dostupné z: https://shadowd.zecure.org/.
[20] Mirai Botnet DDoS Attack Type. *Corero* [online]. Spojené státy americké, 2020 [cit. 2020-03-30]. Dostupné z: https://www.corero.com/resources/ddos-attack-types/mirai-botnet-ddos-attack.html.
[21] Kojoney – A Honeypot For The SSH Service. *Kojoney* [online]. Spojené státy americké, c2005-2006 [cit. 2020-03-30]. Dostupné z: http://kojoney.sourceforge.net/.
[22] ICS/SCADA Honeypot. *Conpot* [online]. Spojené státy americké, 2020 [cit. 2020-04-02]. Dostupné z: http://conpot.org/.
[23] Raspberry Pi. *The Raspberry Pi Foundation* [online]. Velká Británie, 2020 [cit. 2020-03-30]. Dostupné z: https://www.raspberrypi.org/.
[24] SCHILDT, Herbert. *Mistrovství – Java: An Introduction to Honeypots*. Brno: Computer Press ve spol. Albatros Media, 2014. Mistrovství. ISBN 978-80-251-4145-8.
[25] JSch – Java Secure Channel. *JCraft* [online]. Japonsko, c1998-2016 [cit. 2020-03-29]. Dostupné z: http://www.jcraft.com/jsch/.
[26] Bootstrap. *Bootstrap* [online]. Spojené státy americké, 2020 [cit. 2020-04-02]. Dostupné z: https://getbootstrap.com/.
[27] Spring. *Spring* [online]. Spojené státy americké, 2020 [cit. 2020-04-02]. Dostupné z: https://spring.io/.
[28] React – A JavaScript Library For Building User Interfaces. *React* [online]. 2020 [cit. 2020-04-02]. Dostupné z: https://reactjs.org/.
[29] PostgreSQL: The World's Most Advanced Open Source Relational Database. *PostgreSQL* [online]. c1996-2020 [cit. 2020-05-07].
Dostupné z: https://www.postgresql.org/?fbclid=IwAR1_8_bVqonqRkz-7F2KJhq9yx0Z5rpcvZWk-Lc39wWt3OLHYwxNi2Pqk9w
[30] Object Relational Mapper. *Hibernate* [online]. [cit. 2020-04-14]. Dostupné z: https://hibernate.org/orm/
[31] Flowchart Maker & Online Diagram Software. *Diagrams.net* [online]. [cit. 2020-04-15]. Dostupné z: https://app.diagrams.net/
[32] Redux-Saga. *Redux-Saga* [online]. [cit. 2020-04-14]. Dostupné z: https://redux-saga.js.org/.
[33] Redux. *Redux* [online]. c2015-2020 [cit. 2020-04-15]. Dostupné z: https://redux.js.org/ /

[34] Nikto - The Manual. *Nikto v2.1.5 - The Manual* [online]. [cit. 2020-04-14]. Dostupné z: https://cirt.net/nikto2-docs/

Obrázky

1. DTK uživatelské prostředí konfigurace – DTK uživatelské prostředí konfigurace. In: *All.Net: Deception Toolkit* [online]. 2019 [cit. 2020-04-16]. Dostupné z: http://www.all.net/dtk/
2. Honeynet – Ukázka prostředí sítě Honeynet. BARFAR, Arash a Shahriar MOHAMMEDI. In: *Honeypots: Intrusion deception* [online]. ISSA Journal, 2007 [cit. 2020-04-20]. Dostupné z: https://www.researchgate.net/figure/Figure-shows-a-network-diagram-of-a-honeynet-setup-with-four-honeypots-The-honeywall_fig1_228854989
3. Shadow Deamon UI. In: *Shadow Deamon* [online]. 2020 [cit. 2020-04-24]. Dostupné z: https://shadowd.zecure.org/overview/user_interface/
4. Raspberry Pi 3 Model B+. In: *Raspberry Pi 3 Model B+* [online]. 2020 [cit. 2020-04-24]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

A Přílohy

- Webová aplikace – honeypotapp
- Kopie bakalářské práce – 2020_JAK0087_BP.pdf
- Uživatelská příručka – prirucka.pdf