UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

DETEKCIA HONEYPOTOV

2020 **Bc. Lucia KOKUĽOVÁ**

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

DETEKCIA HONEYPOTOV

DIPLOMOVÁ PRÁCA

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

Košice 2020

Bc. Lucia KOKUĽOVÁ

Univerzita P. J. Šafárika v Košiciach
Prírodovedecká fakulta

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Lucia Kokuľová

Študijný program: Informatika (Jednoodborové štúdium, magisterský II. st., denná forma)

Študijný odbor: Informatika

Typ záverečnej práce: Diplomová práca

Jazyk záverečnej práce: slovenský

Sekundárny jazyk: anglický

Názov: Detekcia honeypotov

Názov EN: Detection of honeypots

Cieľ:
1. Analýza bezpečnosti honeypotov a vytvorenie odporúčaní pre tvorbua používanie honeypotov z pohľadu nedetekovateľnosti
2. Porovnanie a vyhodnotenie existujúcich prístupov k detekcii honeypotov3. Návrh, implementácia a vyhodnotenie nástroja/prostredia na detekciu honeypotov

Literatúra:
[1] NG, Chee Keong; PAN, Lei; XIANG, Yang. Honeypot Frameworks andTheir Applications: A New Framework. Springer, 2018.
[2] UITTO, Joni, et al. A Survey on Anti-honeypot and Anti-introspectionMethods. In: World Conference on Information Systems and Technologies. Springer, Cham, 2017. p. 125-134.
[3] JOSHI, R. C.; SARDANA, Anjali. Honeypots: a new paradigm toinformation security. CRC Press, 2011.

Vedúci: RNDr. JUDr. Pavol Sokol, PhD.

Konzultant: RNDr. Tomáš Bajtoš

Oponent: Ing. Miron Kuzma, PhD.

Ústav : ÚINF - Ústav informatiky

Riaditeľ ústavu: RNDr. Ondrej Krídlo, PhD.

Dátum schválenia: 31.03.2020

Poďakovanie

Týmto sa chcem poďakovať vedúcemu tejto diplomovej práce RNDr. JUDr. Pavlovi Sokolovi, PhD. a taktiež konzultantovi RNDr. Tomášovi Bajtošovi za ich cenné rady a pomoc s prípravou a písaním tejto práce.

Abstrakt v štátnom jazyku

Práca sa zaoberá testovaním jednej z najdôležitejších vlastností honeypotov, a to ich detekovateľnosťou. Princíp honeypotu spočíva v jeho neautorizovanom využití a je preto dôležité, aby honeypot nebolo možné v rámci počítačovej siete detegovať. V práci z tohto dôvodu rozoberieme rôzne techniky detekcie honeypotov. Našim cieľom bude navrhnúť systém, ktorý bude honeypoty detegovať, čím odhalí ich slabé stránky, ktoré následne môžu byť odstránené. Výsledkom našej práce bude nástroj, ktorý na vstupe dostane IP adresu a jeho výstupom bude vyjadrenie, či daný stroj je honeypot, alebo reálny operačný systém, spôsob detekcie a navrhnuté opatrenie proti detekcii.

Kľúčové slová: honeypot, honeynet, podvodné systémy, detekcia, informačná bezpečnosť

Abstrakt v cudzom jazyku

This paper deals with testing one of the most important properties of honeypots, namely their detectability. The principle of honeypot lies in its unauthorized use, and it is therefore crucial that honeypot cannot be detected within the computer network. For this reason, we will discuss various techniques of honeypot detection. Our goal is to design a system that detects honeypots, revealing their weaknesses, which can then be removed. The result of our paper will be a tool that gets an IP address on the input, and its output will be a statement whether the machine is a honeypot or a real operating system, a method of detection and a proposed measure against detection.

Keywords: honeypot, honeynet, deception systems, detection, information security

Obsah

Zoznam ilustrácií

Zoznam tabuliek

Zoznam skratiek a značiek

ADB	Android Debug Bridge
FTP	File Transfer Protocol, protokol prenosu súborov
HTTP	Hypertext Transfer Protocol, hypertextový prenosový protokol
ICMP	Internet Control Message Protocol,
IDS	Intrusion Detection System, systém detekcie narušenia
KVM	Kernel-Based Virtual Machine, virtuálny stroj založený na jadre
LXC	Linux Containers
MAC	Media Access Control, identifikátor sieťového zariadenia
QEMU	Quick EMUlator
RTT	Round-Trip Time, obojsmerné omeškanie
SMTP	Simple Mail Transfer Protocol, jednoduchý protokol na prenos pošty
SSH	Secure Shell, zabezpečený prístup k príkazovému interpretovaču
TCP	Transmission Control Protocol, protokol riadenia prenosu
UDP	User Datagram Protocol, používateľský datagramový protokol
UML	User Mode Linux, užívateľský režim Linuxu

Úvod

V sieti internet, tak ako ju poznáme dnes, sú pripojené milióny zariadení. Na tieto zariadenia každý deň smerujú útoky, či už s cieľom zneškodniť ich, alebo získať od nich rôzne informácie. Útočníci sa každým dňom vyvíjajú a ich útoky sú čím ďalej vyspelejšie. Je preto dôležité ich pozorovať, porozumieť ich technikám, a tým sa naučiť odvracať bezpečnostné hrozby, ktoré predstavujú. Na tieto účely existuje mnoho nástrojov, ktoré majú za úlohu útočníka odhaliť, poprípade analyzovať. V tejto práci sa budeme venovať jednému z nich, konkrétne honeypotu.

Honeypot je bezpečnostný nástroj na detekciu a prípadné odvrátenie neautorizovanej manipulácie so systémom. Tento nástroj sleduje aktivitu útočníka v počítačovej sieti, čím ho dokáže efektívne identifikovať a následne monitorovať jeho aktivitu. Jeho hodnota spočíva v tom, že ho útočník použije, honeypot by sa teda mal útočníkovi javiť ako dostupný cieľ. Jednou z najdôležitejších vlastností honeypotov je však skutočnosť, že útočník nemá vedomosť o tom, že sa pripája na honeypot.

Napriek tomu, že útočník má byť schopný neautorizovane použiť honeypot, nemôže byť schopný odhaliť to, že s ním komunikuje. V tomto prípade by hrozilo, že ukončí, resp. pozmení svoju aktivitu, čim sa cieľ honeypotu – identifikácia správania útočníka, nepodarí zrealizovať. Napriek skutočnosti, že nedetekovateľnosť je základná vlastnosť honeypotov, niektoré typy honeypotov je možné detegovať na základe ich charakteristických čŕt. Pri detekcii je možné sledovať napríklad služby ponúkané systémom, o ktorej chceme zistiť či je honeypotom. Inými vlastnosťami, ktoré pri detekcii môžeme využiť, môže byť nezvyčajné správanie či používanie špecifických hardvérových zariadení.

Jednou z možností, ako vyriešiť tento problém, je pokúsiť sa obmedziť možnosti detekcie honeypotov. Cieľom našej práce je z tohto dôvodu návrh a implementácia nástroja na detekciu honeypotov, ktorého úlohou bude odhaliť honeypot v rámci počítačovej siete, na základe čoho bude možné odstrániť vlastnosti, ktoré dopomohli k jeho odhaleniu. Súčasné riešenia detekcie honeypotov ponúkajú rôzne spôsoby, ako odhaliť honeypot zapojený v sieti, pričom na ich detekciu používajú ich známe vlastnosti, či predvolené nastavenia. Táto detekcia je teda založená na predpoklade, že používateľ, ktorý honeypot nasadil, mu ponechal jeho pôvodné nastavenia, či ho nedostatočne zabezpečil.

Cieľom tejto práce je určiť všeobecný spôsob detekcie honeypotov. Predstavíme si preto pojem podvodné systémy a taktiež spôsoby podvodov, ktoré tieto systémy používajú. Honeypoty taktiež zaradzujeme k podvodným systémom, keďže ich cieľom je nalákať útočníka na ich zneužitie. Pomocou týchto pojmov budeme schopní klasifikovať honeypoty vo všeobecnosti do šiestich základných skupín, ktoré popisujú typy podvodov používaných v honeypotoch, a to maskovanie, obalenie, zatienenie, napodobňovanie, vynájdenie a návnada. Táto klasifikácia nám pomôže viac pochopiť správanie honeypotov a tým nám uľahčí nájdenie všeobecného spôsobu, ako tieto podvodné systémy detegovať.

Práca je rozdelená do šiestich základných kapitol. V prvej kapitole sa venujeme honeypotom, ich základným vlastnostiam a deleniam. Popísali sme taktiež čo sú to honeynety a aká je ich základná funkcia. V samostatnej podkapitole sme sa zaoberali nástrojom T-Pot, ktorý predstavuje kolekciu viacerých honeypotov. Na tomto nástroji sme testovali našu implementáciu a preto sme v úvode tejto práce popísali aké honeypoty obsahuje a ako tento nástroj pracuje.

Druhá kapitola tejto práce predstavuje jej jadro, zaoberá sa podvodnými systémami a podvodnými metódami. V tejto kapitole popisujeme metódy maskovanie, obalenie, zatienenie, napodobňovanie, vynájdenie a návnadu. Ku každej z týchto podvodných metód sme ako príklady priradili jednotlivé honeypoty z nástroja T-Pot a popísali sme, ako využívajú vlastnosti týchto podvodných metód.

V tretej kapitole sa venujeme podobným prácam a existujúcim riešeniam na detekciu honeypotov. Popísali sme detekcie na základne rôznych vlastností honeypotov a tiež existujúce nástroje. Predstavili sme nástroje Checkpot, Honeybee a Shodan Honeyscore, ktoré sa zameriavajú na detekciu honeypotov a popísali sme ich funkcionalitu.

Poslednou kapitolou našej práce je popis návrhu a implementácie nášho riešenia na detekciu honeypotov. Navrhli a implementovali sme nástroj na detekciu honeypotov, ktorý predstavuje spojenie skenovania siete a vytvárania odtlačku systému. Na detekciu sme využili fingerprinting, teda odtlačok, ktorý je pre každý systém jednoznačný a preto sa dokonale hodí na detekciu. V praxi sa rovnako stáva, že útočník identifikuje zraniteľný systém pomocou jeho odtlačku [1]. V rámci implementačnej časti sme sa rozhodli upraviť nástroj na detekciu honeypotov tak, aby identifikáciu uskutočňoval na základe podvodných metód, ktoré používajú a na základe tvorby odtlačkov systému.

1 Honeypoty a honeynety

V tejto kapitole si zadefinujeme najdôležitejšie pojmy týkajúce sa tejto práce, honeypot a honeynet. Pozrieme sa taktiež na delenie honeypotov a honeynetov z rôznych hľadísk a posúdime, ktoré z týchto hľadísk je pre našu prácu významné.

1.1 Honeypot

Honeypot je systémový prostriedok, ktorý sa javí ako služba, množina služieb alebo celý operačný systém či sieť a jeho úlohou je nalákať útočníka [2]. Cieľom honeypotu je vyzerať pred útočníkom ako ľahký, zaujímavý alebo cenný cieľ. Potom ako útočník honeypot napadne, teda skúsi ho využiť vo svoj prospech, honeypot monitoruje každú činnosť útočníka v systéme, ako je prihlasovanie, úprava súborov či spustené procesy. Na rozdiel od firewallu či systému na detekciu narušení (Intrusion Detection System, IDS) honeypot poskytuje oveľa zložitejšiu formu ochrany používateľa. Stratégiou honeypotu je nalákať útočníka, aby ho zneužil. Týmto spôsobom odtiahne jeho pozornosť od reálneho systému, v ktorom je používaný, čím chráni jeho údaje.

Pre honeypoty zvyčajne platí, že nemajú žiadnu špecifickú vlastnosť, podľa ktorej by mohli byť rozdelené do jednoznačných skupín. V rámci odbornej literatúry z tohto dôvodu nájdeme mnoho delení podľa rozličných kritérií, pričom je časté, že kategórie honeypotov sa navzájom prekrývajú a dopĺňajú. V nasledujúcich podkapitolách si predstavíme niekoľko kritérií a delení honeypotov, ktoré sú pre našu prácu dôležité.

1.1.1 Delenie podľa použitia

Podľa využitia sa honeypoty delia na dve skupiny, a to produkčné a výskumné honeypoty. **Produkčné honeypoty** [2] sú určené na ochranu organizácií. Ich cieľom je znížiť risk napadnutia organizácie. Zvyčajne sú to nízko interaktívne honeypoty, ktoré sú ľahko nasaditeľné [3]. Honeypoty sa zvyčajne používajú v systéme spolu s firewallmi, IDS systémami a inými spôsobmi zabezpečenia, keďže takýto honeypot nie je dostačujúcou ochranou pre organizáciu. Sú určené pre zbieranie informácií o tom, akým bezpečnostným hrozbám organizácie čelia.

Druhou skupinou sú **výskumné honeypoty** [2]. Tento typ honeypotov sa používa na zistenie podrobnejších informácií o bezpečnostnej hrozbe, ktorej čelí organizácia.

Tento typ honeypotu je určený na to, aby sa učil. Honeypot teda okrem monitorovania útoku, získava údaje o útočníkovi, ktoré vie neskôr využiť na obranu voči nemu. Tým sa podieľa na dôležitom probléme zabezpečenia organizácií, a to na zistení identity a motivácie útočníka pri útoku na aktíva danej organizácie. Výskumné honeypoty sa tiež využívajú na odhalenie automatizovaných útokov.

1.1.2 Delenie podľa miery interakcie

V tejto podkapitole si predstavíme tri skupiny honeypotov, a to nízko, stredne a vysoko interaktívne honeypoty. Miera interakcie znamená, nakoľko je potenciálnemu útočníkovi umožnené zasahovať do honeypotu. Inými slovami, je to teda interakcia medzi nasadeným systémom a útočníkom. Interakcia ako kritérium delenia je pri honeypotoch ich najčastejšie využívaná vlastnosť [4].

Nízko interaktívne honeypoty sa vyznačujú minimálnou interakciou s útočníkom [2]. Zvyčajne sa javia len ako jednoduché sieťové služby (napr. SSH, HTTP a pod.). Na takýchto honeypotoch nebeží žiadny operačný systém. Nízko interaktívne honeypoty sú rozšírené, pretože sú veľmi ľahko nasaditeľné a udržiavateľné. Na druhej strane ale o útočníkovi vedia zistiť menej informácií (napr. IP adresu, sieťový port, použité prihlasovacie údaje). Takéto honeypoty sa používajú hlavne ak stredne a vysoko interaktívne honeypoty neprichádzajú do úvahy (napríklad nedostatočný hardvér na ich nasadenie) alebo ak chceme otestovať zraniteľnosti konkrétnej služby, ktorú nízko interaktívny honeypot môže predstavovať. Príkladom nízko interaktívnych honeypotov sú Dionaea [5] a Honeyd [6].

Najdôležitejšou kategóriou sú **vysoko interaktívne honeypoty** [2]. Tie útočníkovi poskytujú reálny operačný systém, s kompletným prístupom, čo umožňuje honeypotu sledovať priebeh celého útoku. Tieto honeypoty sa teda zvyčajne využívajú na výskumné účely. Vysoko interaktívne honeypoty sú náročné na vývoj, pretože na ich spustenie potrebujeme mnoho nástrojov a znalostí. Známymi honeypotmi v tejto skupine sú napríklad Sebek [7] a Argos [8].

Na hranici medzi nízko a vysoko interaktívnymi honeypotmi sa nachádzajú **stredne interaktívne honeypoty**, ktorých cieľom je skombinovať výhody spomenutých dvoch skupín [2]. Táto kategória sa pri delení podľa miery interakcie často neuvádza a je spájaná s jednou zo zvyšných dvoch skupín honeypotov, pretože obsahuje len malé rozdiely. Najčastejšie je pripojená k nízko interaktívnym honeypotom (napr. pri

honeypote Kippo). Táto situácia nastáva, ak stredne interaktívny honeypot simuluje nejakú sieťovú službu (teda by sme ho mohli zaradiť k nízko interaktívnym honeypotom). Rozdiel bude v tom, že bude poskytovať niekoľko funkcionalít naviac. Príkladom takéhoto honeypotu je honeypot Kippo [9].

V Tab. 1. môžeme vidieť zhodnotenie výberu honeypotov podľa požiadaviek na systém. Z tabuľky si používateľ vie určiť, aký honeypot je pre neho vhodný. To je možné určiť na základe toho, aké požiadavky kladie na náročnosť inštalácie, údržbu, na riziká pri napadnutí honeypotu či množstvo informácií zbieraných o útočníkoch. Poradie faktorov zhora nadol naznačuje zvyšujúcu sa účinnosť honeypotu a zároveň zvyšuje zložitosť, riziko a ťažkosti pri jeho nasadzovaní.

Tab. 1 Výber honeypotu podľa požiadaviek na systém [2].

Faktory	**Nízko interaktívne honeypoty**	**Stredne interaktívne honeypoty**	**Vysoko interaktívne honeypoty**
Stupeň zapojenia útočníka	Nízky	Stredný	Vysoký
Reálny operačný systém	Nie	Nie	Áno
Inštalácia	Ľahká	Ťažká	Veľmi ťažká
Údržba	Ľahká	Ľahká	Časovo náročná
Riziko	Nízke	Stredné	Vysoké
Cieľom je ohrozenie	Nie	Nie	Áno
Potrebná kontrola	Nie	Nie	Áno
Znalosti na nasadenie	Nízke	Nízke	Vysoké
Znalosti na vývoj	Nízke	Vysoké	Vysoké
Množstvo zbieraných údajov	Limitované	Stredné	Rozsiahle
Interakcia útočníka s honeypotom	Emulované služby	Požiadavky	Plná kontrola

1.1.3 Delenie podľa typu nasadenia

Jednou z dvoch skupín v tejto kategórii sú **fyzické honeypoty** [2]. Fyzický honeypot je typicky jeden stroj pripojený k počítačovej sieti a prístupný cez jednu IP adresu. Tieto honeypoty sú stále spájané s konceptom vysoko interaktívneho honeypotu, keďže predstavujú fyzický stroj, počítač s operačným systémom. Fyzické honeypoty v praxi nie sú veľmi využiteľné z dôvodu vysokých nákladov na ich nasadenie a údržbu.

Druhou skupinou sú **virtuálne honeypoty** [2]. Oproti fyzickým honeypotom sú omnoho výhodnejšie z pohľadu nákladov na ich údržbu, pretože pomocou jedného fyzického stroja s jednou IP adresou vieme vytvoriť niekoľko virtuálnych honeypotov pomocou voľne dostupných nástrojov na virtualizáciu (Napr. XEN, LXC, QEMU/KVM).

1.1.4 Delenie podľa roly

Podľa roly delíme honeypoty na dve základné skupiny, a to klientske a serverové honeypoty. Ako vyplýva z ich názvu, každý typ predstavuje jednu stranu typickej architektúry komunikácie v počítačovej sieti, teda klienta a server.

Klientske honeypoty sa javia ako zraniteľné klientske aplikácie, ktoré komunikujú so serverom a snažia sa zistiť, či je server reálny a či sa nepokúša o útok [2]. Ich cieľom je teda odhaliť takéto podvodné servery. Klientsky honeypot sa zvyčajne skladá z troch častí. Prvou je komponent, ktorý sa nazýva žiadateľ (queuer), jeho úlohou je vytvoriť list dostupných serverov, na ktoré sa má honeypot pripájať. Druhou časťou honeypotu je samotný klient, ktorý je schopný posielať požiadavky na servery identifikované žiadateľom. Po interakcii so serverom, prichádza na rad tretí komponent klientskeho honeypotu, analytická časť, ktorá je zodpovedná za určenie toho, či na klienta bol uskutočnený útok. Príkladom klientskeho honeypotu je nízko interaktívny honeypot Thug [10].

Druhou skupinou sú **serverové honeypoty**, ktoré emulujú úlohu servera. Pasívne čakajú, kým sa k nim pripojí klient a ten má po pripojení k dispozícii celú jeho funkcionalitu. Serverovými honeypotmi sú zvyčajne nízko interaktívne honeypoty predstavujúce sieťové služby, na ktoré sa pripájajú klienti. Väčšina typov honeypotov je serverová. Typickými serverovými honeypotmi sú Kippo a Honeyd.

1.2 Honeynet

Honeynet je sieť tvorená dvomi alebo viacerými honeypotmi [2]. Honeynet je zvyčajne využívaný na rovnaký účel ako honeypot, avšak pri rozsiahlejších sieťach, kde by jeden honeypot nebol dostatočný. Na rozdiel od honeypotu je teda honeynet fyzická sieť niekoľkých systémov. Architektúru honeynetu definujú tri základné elementy [2]:

- kontrola toku údajov (data control),
- zachytávanie údajov (data capture),
- zber údajov (data collection)

Kontrola toku údajov (data control) je v honeynete činnosť, ktorá znižuje riziko [2]. Znamená to, že honeynet kontroluje útočníkovu aktivitu tým, že obmedzuje čo sa môže stať, k čomu útočník má a nemá prístup. Riziko nastáva, ak sa útočník dostane do honeynetu a môže nastať situácia, že prostredníctvom neho sa vie dostať aj do reálneho systému v ktorom je honeynet nasadený. Útočník preto musí byť kontrolovaný honeynetom a mať obmedzený prístup do niektorých častí systému.

Druhou požiadavkou pre honeynety je **zachytávanie údajov (data capture)** [2]. Dôležitým faktorom v tejto fáze je, že útočník si nesmie byť vedomý toho, že niekto sleduje jeho aktivitu a nemôže byť schopný prísť na to, že sa nachádza v honeynete. Zdroje použité na zachytávanie údajov musia byť zabezpečené tak, aby údaje nemohli byť kompromitované (aby nemohla byť narušená ich integrita). Táto práca s údajmi zahŕňa mnoho krokov, ako je napríklad uskladňovanie údajov na inom mieste než priamo na honeynete, ukladanie všetkých zozbieraných údajov, či vzdialený prístup k údajom pre administrátora.

Treťou požiadavkou na architektúru honeynetu je **zber údajov (data collection)** [2]. Tento krok je dôležitý najmä pri rozsiahlych honeynetoch, kedy údaje o útočníkovi zbiera viacero nasadených honeynetov, z čoho je potom potrebné získať súhrnné informácie. Zber údajov vyžaduje štandardný formát údajov (zhodný pre všetky honeynety nasadené v počítačovej sieti) a zabezpečený prenos zozbieraných údajov od jednotlivých honeynetov k zdroju, ktorý tieto údaje následne vyhodnocuje.

Okrem zachytávania útokov na počítačovú sieť a ich monitorovania, význam honeynetov spočíva aj v tom, že sa môžu použiť ako testovacie prostredia. Honeynet ako kontrolované prostredie (administrátorom) môže byť využiteľný na analýzu zraniteľností

v nových aplikáciách alebo operačných systémoch. Prínosom teda je, že bezpečnostné riziká zistené honeynetmi sa dajú vyriešiť skôr, ako sa technológie zavedú do produkčného prostredia.

1.3 T-Pot

V tejto podkapitole si predstavíme prostredie, na ktorom budeme testovať náš nástroj na detekciu honeypotov. T-Pot [11] je nástroj pochádzajúci z projektu Deutsche Telekom AG Community Honeypot Project [12]. T-Pot je súčasne platforma obsahujúca viacero honeypotov, prepojených pomocou Dockeru [13], čo je projekt s otvoreným kódom poskytujúci rozhranie využívajúce izoláciu jednotlivých aplikácií (v tomto prípade honeypotov) do takzvaných kontajnerov [14]. V nasledujúcich podkapitolách si priblížime koncept nástroja T-Pot, popíšeme aké honeypoty obsahuje a pozrieme sa na ich znaky podvodných systémov.

1.3.1 Štruktúra nástroja

Ako sme uviedli v úvode kapitoly, T-Pot využíva Docker na správu jednotlivých honeypotov, ktoré obsahuje. Jeho výhodou je, že honeypoty sú uložené v kontajneroch, ktoré zabezpečujú izoláciu týchto systémov. Jednotlivé kontajnery obsahujú honeypot a potrebné súbory pre jeho chod, ale neobsahujú žiadny operačný systém. To zabezpečuje nižšie náklady na prevádzku a taktiež úsporu miesta.

Ďalšou dôležitou vlastnosťou T-Potu je, že je založený na takzvanom ELK Stack [15]. ELK je skratka pre tri projekty s otvoreným kódom, a to Elasticsearch [16], Logstash [17] a Kibana [18]. Elasticsearch je distribuovaný, vyhľadávací a analytický nástroj s otvoreným kódom pre všetky typy údajov, V našom prípade nástroj na vyhľadávanie údajov zozbieraných z jednotlivých honeypotov. Nástroj Logstash slúži na spracovanie údajov na strane servera, pričom zbiera údaje z viacerých zdrojov (teda kontajnerov obsahujúcich honeypoty) a následne ich odosiela ďalej do Elasticsearch. Kibana umožňuje vizualizáciu údajov zozbieraných pomocou Elasticsearch a taktiež jednoduchú navigáciu vo webovom rozhraní nástroja T-Pot.

1.3.2 Honeypoty v nástroji T-Pot

T-Pot v jeho najnovšej verzii 19.03 (ku dňu odovzdania práce) obsahuje spolu 16 honeypotov rozdelených do jednotlivých kontajnerov v Dockeri. V tejto kapitole si uvedieme, na čo tieto honeypoty slúžia a tiež, do ktorých typov podvodných systémov ich zaraďujeme.

ADBHoney [19] je nízko interaktívny honeypot navrhnutý pre protokol Android Debug Bridge (ADB) [20]. ADB je protokol určený na sledovanie emulovaných či skutočných zariadení pripojených k danému používateľovi. Je to nástroj fungujúci v príkazovom riadku, honeypot ADBHoney teda simuluje tento nástroj. O ADBHoney môžeme teda povedať, že využíva napodobňovanie. Tento honeypot môžeme rovnako zahrnúť aj do maskovania, keďže maskuje to že je honeypot tým. že je schopný vykonať niektoré konkrétne príkazy reálneho nástroja ADB (napríklad príkaz `adb connect`) [19].

Ďalším z honeypotov v kolekcii je nízko interaktívny **Cisco ASA honeypot** [21]. Cisco Adaptive Security Appliance (alebo skrátene Cisco ASA) je rodina bezpečnostných nástrojov chrániacich siete a dátové centrá rôznych spoločností [22]. Honeypot Cisco ASA je schopný detegovať zraniteľnosť CVE-2018-0101 [23], ktorá umožňuje útočníkovi bez autentifikácie vzdialene vykonať ľubovoľný kód a taktiež deteguje záplavový útok DoS (odmietnutie služby). Z hľadiska typov podvodných systémov ho môžeme zaradiť k návnade, pretože tváriac sa ako zraniteľný nástroj, láka útočníka na jeho zneužitie.

Tretím v poradí je serverový **Conpot** [24], ktorý je nízko interaktívny, ako predchádzajúce dva honeypoty. Conpot zahrňuje rôzne bežne používané protokoly využívané v priemysle, pomocou ktorých pred útočníkom simuluje štruktúru priemyselného kontrolného systému. Tento honeypot z pohľadu podvodných systémov využíva napodobňovanie lebo simuluje reálny systém, vynájdenie, pretože predstavuje systém ktorý nie je reálny a taktiež návnadu, keďže jeho cieľom je tváriť sa ako systém, ktorý sa dá zneužiť.

Na rozdiel od prechádzajúcich, ďalší z honeypotov **Cowrie** [25], je stredne až vysoko interaktívny honeypot, simulujúci služby SSH a telnet. Je určený na zaznamenávanie konzolovej interakcie útočníka a tiež útokov hrubou silou na prelomenie hesla. Cowrie zaraďujeme pod typy podvodu napodobňovanie (reálnej služby) a taktiež návnadu (prihlasovanie pod heslom).

Nasledujúcim honeypotom je **Dionaea** [26], honeypot ktorý zachytáva malvér využívajúci zraniteľné miesta služieb ponúkaných cez sieť a snaží sa pritom získať kópiu tohto malvéru. Dionaea využíva zatienenie, keďže schválne útočníkovi poskytuje známe typy zraniteľností, s cieľom aby ich zneužil. Tento honeypot môžeme tiež priradiť k vynájdeniu, pretože sa tvári že obsahuje zraniteľnosť, ktorá sa však reálne v sieti nenachádza.

Súčasťou kolekcie je tiež nástroj **ElasticPot** [27], čo je jednoduchý honeypot zameraný na zaznamenávanie útokov na vyhľadávací a analytický nástroj Elasticsearch, ktorý je taktiež súčasťou kolekcie T-Pot. ElasticPot využíva ako podvod hlavne maskovanie, keďže jeho úlohou je bez interakcie sledovať komunikáciu útočníka s nástrojom Elasticsearch.

Ďalším honeypotom v T-Pote je **Glastopf** [28], honeypot pre webové aplikácie napísaný v Pythone. Namiesto napodobňovania konkrétnej zraniteľnosti vo webovej aplikácii, napodobňuje typ zraniteľnosti, je teda viac všeobecný, čo zvyšuje jeho šancu zachytiť útočníka. Glastopf vo veľkej miere využíva napodobňovanie (zraniteľností) ale tiež zatienenie (zraniteľnosti simuluje všeobecne, nie konkrétne) a obalenie (webovú aplikáciu predstavuje ako zraniteľnú, pričom opak je pravdou).

Nasledujúcim honeypotom v zbierke je **Glutton** [29], poskytujúci SSH a TCP pripojenie k serveru. SSH proxy v tomto prípade funguje ako prostredník medzi útočníkom a serverom a navyše slúži na zaznamenávanie celej komunikácie, TCP proxy ktoré predstavuje Glutton zatiaľ logovanie komunikácie neposkytuje. Tento honeypot sa vyznačuje návnadou, pretože láka útočníka na prelomenie prihlasovacích údajov a maskovaním, pretože sa tvári ako reálna služba.

Ďalším jednoduchým honeypotom je **Heralding** [30], nízko interaktívny honeypot, ktorý používateľovi umožňuje simulovať množstvo protokolov (v súčasnej dobre je ich 14) využívajúcich prihlásenie do systému, či služby. Heralding následne zaznamenáva všetky prihlasovacie údaje zadané útočníkom pri prihlasovaní sa do jednotlivých služieb. Z rovnakých dôvodov ako Glutton, zaraďujeme Heralding k návnade a maskovaniu.

O niečo komplexnejší je honeypot **HoneyPy** [31], ktorý takisto simuluje systém zahŕňajúci viacero služieb a zaznamenáva útoky na jednotlivé z nich. HoneyPy je nízko až stredne interaktívny honeypot emulujúci služby založené na internetových protokoloch TCP a UDP. Podobne ako honeypoty simulujúce pripojenie, aj HoneyPy využíva návnadu vo forme využívania prihlasovacích údajov.

Podobným nástrojom ako HoneyPy je **HoneyTrap** [32], bezpečnostný nástroj slúžiaci na sledovanie útokov na služby využívajúce protokoly TCP a UDP. Server emuluje známu službu jednoduchým odoslaním zachytenej sieťovej prevádzky pripojenému hostiteľovi, teda potenciálnemu útočníkovi. Rovnako ako HoneyPy, HoneyTrap predstavuje návnadu pre útočníka, umožnením prihlásenia sa do jednotlivých služieb.

Ďalším honeypotom, ktorý obsahuje T-Pot je **Mailoney** [33], SMTP honeypot napísaný v Pythone. Zaznamenáva pokusy o prihlásenie a taktiež e-maily, ktoré sa útočník pokúša odoslať. Mailoney využíva maskovanie a z rovnakých dôvodov ako iné honeypoty sledujúce prihlásenie, využíva návnadu.

Funkcionálne najviac odlišným z týchto honeypotov je **Medpot** [34], takzvaný HL7/FHIR honeypot. HL7 alebo tiež Health Level Seven je spoločnosť a zároveň súbor medzinárodných štandardov na prenos a zdieľanie klinických a administratívnych údajov medzi aplikáciami, ktoré používajú rôzni poskytovatelia zdravotnej starostlivosti [35]. FHIR je štandard pre výmenu údajov o zdravotnej starostlivosti publikovaný spoločnosťou HL7 [36]. Medpot využíva návnadu, keďže predstavuje systém pracujúci s citlivými údajmi a tiež napodobňovanie (prenosu údajov).

Nasledujúcim honeypotom je **RDPY** [37], implementácia protokolu RDP (Remote Desktop Protocol), čo je sieťový protokol umožňujúci používateľovi vzdialene ovládať počítač prostredníctvom počítačovej siete [38]. RDPY ako podvod používa napodobňovanie, keďže simuluje reálny protokol a taktiež vynájdenie, keďže reálne pripojenie k vzdialenému počítaču nie je zrealizované.

Poslednou dvojicou z kolekcie T-Pot sú honeypoty **Snare** [39] a **Tanner** [40]. Snare je honeypot pre webové aplikácie, ktorý funguje ako senzor a je lákadlom pre škodlivý softvér. Tanner funguje ako jeho druhá polovica, je teda službou, ktorá analyzuje a klasifikuje údaje z udalostí zaznamenaných honeypotom Snare a rozhoduje o tom, ako má Snare na dané udalosti reagovať. Snare aj Tanner oba využívajú maskovanie a návnadu, keďže Snare funguje ako lákadlo a Tanner ako logovacia služba.

1.4 Spôsoby určenia systému na základe odtlačku

V tejto kapitole si predstavíme dva prístupy na identifikáciu systémov pomocou fingerprintingu (vytvorenia jedinečného odtlačku systému). Tieto nástroje sa zameriavajú

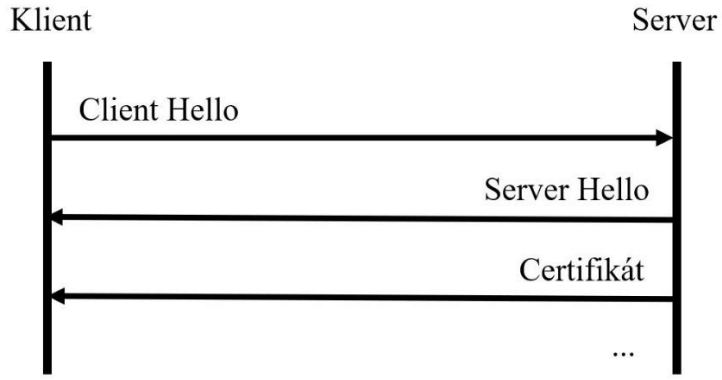
na SSL/TLS odtlačok systému a na sieťový odtlačok vytváraný pomocou SSH implementácie na strane klienta a servera. Odtlačky systému sa dajú ľahko ukladať, vyhľadávať a zdieľať vo forme MD5 odtlačku (hashu).

1.4.1 JA3 a JA3S

Algoritmus JA3 [41] predstavuje identifikáciu komunikácie pomocou odtlačku založeného na protokole TLS a jeho predchodcovi SSL. Tieto protokoly slúžia na zabezpečenie dôvernosti komunikácie tým, že danú komunikáciu šifrujú. Obdobne sú tieto protokoly zneužívané škodlivými programami (malvérom) na zakrytie svoje škodlivej aktivity

Ako je možné vidieť na Obr. 1, na začatie TLS spojenia (relácie), začínajúca strana komunikácie (klient) pošle druhej strane (server) TLS Hello Client paket. Tento paket a spôsob jeho generovania závisia od balíkov a metód použitých pri zostavovaní klientskej aplikácie. Ak server podporuje TLS komunikáciu, odpovie server Hello paketom. Ten je vytvorený na základe knižníc a konfigurácií na strane servera a tiež pomocou informácií z Client Hello paketu.

Obr. 1 Začiatok toku správ v TLS handshake protokole



Keďže správy zobrazené na Obr. 1 sa prenášajú nešifrovane, je možné pomocou nich získať odtlačok klientskej ako aj serverovej aplikácie. To je možné získať pomocou detailov nachádzajúcich sa v jednotlivých Hello paketoch. Na toto získavanie odtlačkov, slúžia metódy JA3 [41] pre klientsku aplikáciu a JA3S [41] pre serverovú aplikáciu. Pre nás je dôležité získavať odtlačky na strane servera. Najmä z dôvodu, že náš nástroj ako

začínajúca strana komunikácie, chce vedieť detegovať druhú stranu komunikácie (čiže server) ako honeypot.

Metóda JA3S spočíva v zbere desatinných hodnôt bajtov pre nasledujúce polia v pakete Server Hello:

- verzia (Version),
- akceptované šifry (Accepted Cipher) a
- zoznam rozšírení (List of Extensions).

Nástroj následne zreťazí tieto hodnoty pomocou čiarky na oddelenie jednotlivých polí a pomocou pomlčky na oddelenie jednotlivých hodnôt v poli [41]. Zreťazenie teda vyzerá nasledovne: *TLSVersion,Cipher,Extensions*. Príkladom je reťazec *769,47,65281–0–11–35–5–16*. Ak sa v Server Hello pakete nenachádzajú žiadne rozšírenia, posledné pole po čiarke ostane prázdne. Z celého tohto reťazca je následne vytvorený MD5 odtlačok (hash), ktorý je výsledným JA3S odtlačkom servera.

1.4.2 HASSH

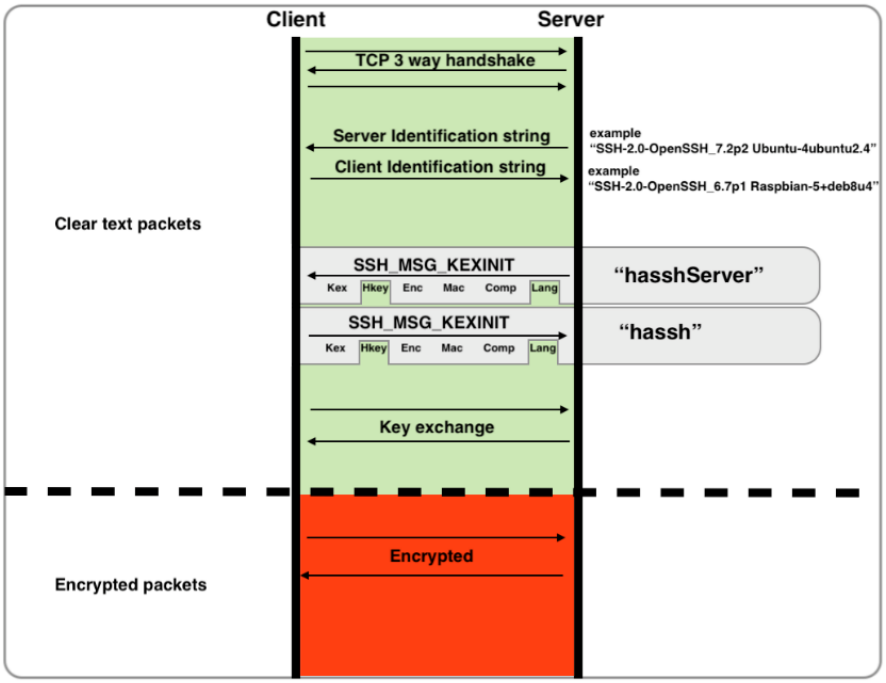
HASSH [42] predstavuje metódu profilovania pre SSH klientov a SSH servery. Predstavuje špecifický sieťový odtlačok, ktorý sa dá použiť na identifikáciu konkrétnych implementácií SSH klienta a servera.

Odtlačky hassh a hasshServer sú MD5 odtlačky (hashe) zostavené podľa špecifickej sady algoritmov, ktoré sú podporované rôznymi klientskymi a serverovými SSH aplikáciami. Ako môžeme vidieť na Obr. 2, tieto algoritmy sa v SSH komunikácii vymieňajú po počiatočnom TCP handshake-u, v podobe nezašifrovaného textu. Poznáme ich ako správy SSH_MSG_KEXINIT. Tieto správy sú neoddeliteľnou súčasťou nastavenia konečného šifrovaného SSH kanála. Výber a usporiadanie algoritmov v špecifikácii daného SSH spojenia je dostatočne jedinečné na vytvorenie jedinečného odtlačku klienta či servera. Rovnako ako v predchádzajúcej podkapitole, pre nás je dôležité získavať odtlačky na strane servera, teda využiť nástroj hasshServer.

Metóda hasshServer spočíva v zreťazení štyroch polí z SSH konfigurácie, a to metódy výmeny kľúčov (Key Exchange methods), šifrovania (Encryption), overenia správy (Message Authentication) a kompresie (Compression). Metóda zreťazí tieto hodnoty pomocou bodkočiarky a jednotlivé hodnoty v poliach oddelí čiarkami. Výsledné

zreťazenie jednotlivých polí, z ktorých sa vytvára konečný odtlačok má teda tvar *Key Exchange methods;Encryption;Message Authentication;Compression.*

Obr. 2 Tok správ v protokole SSH [42]



2 Podvodné systémy

V oblasti informačných technológií sa často spomína pojem podvod. **Podvod** je úspešný pokus o to, aby niekto uveril niečomu, čo je nepravdivé, a to buď úmyselne alebo neúmyselne. Pod podvodom môžeme rozumieť aj činnosť, ktorá využíva cudziu nevedomosť, či dôvernosť k vlastnému prospechu.

V súvislosti s informačnou bezpečnosťou používame pojem **kybernetický podvod** [43]. Je to podvod, ktorý sa vyskytuje v kybernetickom priestore. Takýto podvod môže byť v útočnom zmysle (napadnúť niekoho) alebo v zmysle obrannom (obrana proti útokom). Ofenzívne podvody majú vo zvykoch používať obmedzenú množinu metód, ako je napríklad vydávanie sa za druhú osobu, ale vždy s drobnými zmenami. Obranné podvody môžu a mali by byť rozmanitejšie. V kybernetickom priestore zvyčajne chápeme obranný podvod ako spôsob obrany proti útoku. Inými slovami, ide o aktívnu obranu voči útočníkovi [44]. Moderné útoky využívajú obranu proti zložitým počítačovým operáciám, ktoré zahŕňajú podvody [45]. Podvod môže byť použitý na rôznych úrovniach, od sieťovej až po aplikačnú úroveň, čo vyžaduje starostlivú koordináciu medzi viacerými stratégiami [46][47].

Každý kybernetický podvod začína tým, že si jeho iniciátor (aj keď možno nevedome) vyberie metódu, akou bude podvod vykonávať. Podvodné metódy môžeme rozdeliť do šiestich základných kategórií [43]:

- **maskovanie (masking)** – systém skrýva nejaký proces, resp. činnosť na pozadí – napríklad monitorovanie užívateľa,
- **obalenie (repackaging)** – niečo sa skrýva ako niečo iné - napr. vloženie malvéru do bežného programu,
- **zatienenie (dazzling)** – systém niečo skrýva tým, že to „zatieni“ inou činnosťou – napr. posielanie mnohých chybových správ útočníkom, ktorí vykonávajú škodlivú činnosť,
- **napodobňovanie (mimicking)** – systém napodobňuje niečo iné – napríklad podvrhnutý súborový systém,
- **vynájdenie (inventing)** – systém vytvára stále nové objekty (často falošné) na nalákanie útočníka,
- **návnada (decoying)** – napríklad podhodenie prihlasovacích údajov.

Honeypoty sa považujú za najznámejší a najrozšírenejší spôsob podvodných technológií v oblasti informačných technológií. Podvodné metódy môžeme napasovať na činnosti, ktoré honeypoty vykonávajú. Qassrawi a Hongli v článku [48] uvádzajú príklady využitia podvodných metód v oblasti honeypotov:

• **maskovanie** - príkladom môže byť monitorovanie používateľov, tým že modifikujú operačný systém tak, aby sa skryli jeho stopy,

• **obalenie** – príkladom môže byť vloženie softvéru, zneškodňujúceho útok, do navonok bezpečne pôsobiacej časti operačného systému,

• **zatienenie** – za túto metódu môžeme považovať posielanie mnohých chybových hlášok útočníkovi,

• **napodobňovanie** - príkladom je vybudovanie falošného súborového systému, ktorý vyzerá ako súborový systém zaneprázdneného používateľa a jeho cieľom je presvedčiť útočníka, že daný systém nie je honeypot,

• **vynájdenie** - príkladom môže byť ponechanie nejakého softvéru v honeypote, ktorý si útočník stiahne, a tým mu umožní sledovať jeho osobné údaje a aktivitu,

• **návnada** - je to napríklad zámerné ponechanie hesiel v súborovom systéme, čo nabáda útočníka aby ich použil.

Podľa toho, aký typ podvodu využíva daný honeypot, vieme dopredu určiť, aké typy podvodu bude využívať. Po zistení, akú podvodnú metódu honeypot využíva, vieme určiť, akým spôsobom bolo možné honeypot detegovať. Metóda podvodných metód sa v dnešnej dobe používa nielen pri detekcii samotných honeypotov, ale aj honeynetov [49].

2.1 Maskovanie

Cieľom tohto typu podvodu je zamaskovať, skryť vlastnosti reálneho objektu alebo činnosti, či už pred používateľom alebo pred samotným systémom. Maskovanie má zabezpečiť, že príslušný objekt nebude možné detegovať. Honeypoty pri svojej činnosti často využívajú túto techniku, pretože ich cieľom je aby neboli detekovateľné, teda často skrývajú, maskujú svoju činnosť. Príkladom honeypotu, ktorý využíva maskovanie je ADBHoney [19], ktorý sa maskuje napodobňovaním služieb reálneho systému. Ďalšími príkladmi sú ElasticPot [27] alebo Glutton [29] rovnako napodobňujúci reálnu službu

a mnoho ďalších. Maskovanie je spôsob podvodu, ktorý môžeme najčastejšie pozorovať pri správaní honeypotov, pretože patrí k ich základným vlastnostiam.

2.2 Obalenie

Pri technike obalenia, resp. obaľovania je realita skrytá takým spôsobom, aby objekt vyzeral rozdielne od toho aký je v skutočnosti. Typickým príkladom je situácia, kedy sa útok javí ako priateľský e-mail s reálnou hlavičkou, aby nalákal prijímateľa na otvorenie správy či prílohy [50].

Hoci útočník môže túto techniku použiť, aby oklamal používateľa, technika obalenia sa môže použiť aj ako obranný mechanizmus. Pri používaní honeypotov nie je zriedkavé, že honeypot, ktorý predstavuje reálny systém s prednastaveným falošným súborovým systémom. Iným príkladom sú takzvané „honey files“. Sú to súbory, ktorých úlohou je vyzerať ako bežné používateľské súbory. Slúžia však ako upozornenie pre systémového administrátora, keďže monitorujú stav a upozorňujú na situácie, keď s nimi útočník akokoľvek pracuje. Tieto súbory môžu byť rovnako vytvárané aj útočníkmi, pričom zvyčajne mávajú názvy, ktoré majú za úlohu nalákať používateľa na ich otvorenie [50]. Obalenie ako spôsob podvodu využíva napríklad honeypot Glastopf [28], tváriaci sa ako zraniteľný systém, pričom v skutočnosti zraniteľnosti neobsahuje.

2.3 Zatienenie

Zatienenie je technika, ktorej úlohou je zmiasť používateľa. Vyznačuje sa obfuskáciou a randomizáciou objektov identifikovaných v systéme [50]. Zatienenie sa snaží skryť realitu tým, že používa činnosti nesúvisiace so skrývaným objektom s cieľom zmiasť používateľa. Pri vyššie spomínanom posielaní mnohých chybových hlášok, je ich cieľom zatienenie pravej činnosti, teda napríklad toho, čo v systéme beží na pozadí. Techniku zatienenia používajú honeypoty Dionaea [26], využívajúc známe zraniteľnosti na nalákanie útočníka a takisto Glastopf [28], ktorý sa javí ako zraniteľný systém, čím zatajuje svoju činnosť.

2.4 Napodobňovanie

Rovnako ako pri zatienení, aj do tejto techniky môžeme zahrnúť podvrhnutý súborový systém, ktorý vytvárajú honeypoty. Napodobňovanie je technika, ktorá simuluje reálne črty objektu, za ktorý sa snaží vydávať. Iným príkladom sú napríklad honeypoty emulujúce konkrétne služby, ktoré napodobňujú. Popri tejto činnosti monitorujú správanie sa útočníka, ktorý ju používa (napríklad zaznamenávanie konkrétnych príkazov). Za napodobňovanie sa tiež považuje podvrhnutá webová stránka, ktorá je v skutočnosti vytvorená útočníkom. Napodobňovanie využíva viacero honeypotov, medzi inými aj ADBHoney [19] simulujúci protokol ADB, Conpot [24] simulujúci reálny systém, Cowrie [25] napodobňujúci služby SSH a telnet, či Medpot [34] simulujúci prenos údajov.

2.5 Vynájdenie

Tento typ podvodu sa vyznačuje tým, že vytvára dojem že daný objekt existuje, ale v skutočnosti je nereálny, falošný. Pri honeypotoch sa táto technika vyskytuje napríklad v situáciách, keď honeypot predstavuje počítačovú sieť so špecifickými adresami, ale v skutočnosti žiadna z nich neexistuje [50]. Ďalším príkladom môže byť podvrhnutie súboru či programu, ktorý monitoruje útočníka, ktorý si ho stiahne či skopíruje. Z honeypotov techniku vynájdenia používa napríklad Conpot [25], ktorý predstavuje priemyselný systém, ktorý však nie je reálny a honeypot RDPY [37], ktorý simuluje falošné pripojenie k vzdialenému počítaču.

2.6 Návnada

Posledný typ podvodu nazývaný návnada, rovnako ako aj pôvodný význam slova je spôsob ako odlákať pozornosť od relevantných objektov, či služieb. Interakcia medzi útočníkom a „obrancom“ počítačovej siete, môže prebiehať dvoma prípadmi – reálny systém tváriaci sa ako honeypot a naopak, honeypot tváriaci sa ako reálny systém. Honeypoty teda môžu slúžiť organizáciám ako návnady pre útočníkov, pretože donútia útočníka myslieť si, že jeden zo systémov danej spoločnosti je zraniteľný. Tým prilákajú jeho pozornosť a zároveň ju odvrátia od reálnych systémov. Návnadu pri svojej funkcionalite používajú honeypoty Cisco ASA [21] tváriaci sa ako ľahko napadnuteľný

systém, Cowrie [25], HoneyPy [31] a HoneyTrap [32] lákajúci útočníka na prelomenie ich hesiel a mnoho ďalších.

2.7 Zhodnotenie

V predchádzajúcich kapitolách sme popísali jednotlivé podvodné metódy. Ku každej z týchto metód sme priradili konkrétne honeypoty, ktoré sú súčasťou platformy T-Pot. V Tab. 2 môžeme vidieť prehľad spôsobov, ktoré honeypoty využívajú. Zo zhrnutia je zrejmé, že najčastejšie využívaná podvodná metóda je metódy návnady, čiže nalákanie útočníka. Často využívanou metódou je samozrejme aj maskovanie, ktoré by malo byť základnou vlastnosťou honeypotu, keďže zabezpečuje jeho nedetekovateľnosť.

Tab. 2 Prehľad spôsobov podvodu, ktoré využívajú jednotlivé honeypoty

Honeypot	**Masko-vanie**	**Obale-nie**	**Zatiene-nie**	**Napodob-ňovanie**	**Vynájde-nie**	**Návna-da**
ADBHoney	X			X		
Cisco ASA						X
Conpot				X	X	X
Cowrie				X		X
Dionaea			X		X	
ElasticPot	X					
Glastopf		X	X	X		
Glutton	X					X
Heralding	X					X
HoneyPy						X
HoneyTrap						X
Mailoney	X					X
Medpot				X		X
RDPY				X	X	
Snare	X					X
Tanner	X					X

3 Prehľad existujúcich riešení

V existujúcich prácach o detekcii honeypotov sa nachádzajú rôzne prístupy v otázke delenia honeypotov. Každý autor rozdeľuje honeypoty podľa rôznych kritérií a od toho sa zvyčajne odvíja aj technika detekcie použitá na odhalenie honeypotu. V mnohých prípadoch ide o detekciu konkrétneho honeypotu (implementácie), nie typu honeypotu (napríklad všeobecne klientsky honeypot). Tieto riešenia obmedzujú možnosti detekcie honeypotov vo všeobecnej rovine.

3.1 Detekcia vysoko interaktívnych virtuálnych honeypotov

Príkladom nástroja na monitorovanie virtuálnych honeypotov je **VMScope** [51]. Je to monitorovací systém založený na virtualizácii, ktorý má rovnakú schopnosť hĺbkovej kontroly systému ako existujúce interné monitorovacie nástroje, pričom je rovnako transparentný a odolný voči neoprávneným zásahom ako externé monitorovacie nástroje. VMscope je odolný voči manipulácii a je transparentný pre monitorovaný systém. Okrem toho, bez potreby akejkoľvek modifikácie monitorovaného systému, VMscope je schopný pozorovať a zaznamenávať parametre a sémantiku rôznych udalostí virtuálneho systému vrátane systémových volaní. Umožňuje hĺbkovú kontrolu virtuálnych honeypotov bez toho, aby sa vo vnútri umiestnili akékoľvek senzory.

Ďalším nástrojom je **Sebek** [52], ktorý slúži na odchytávanie údajov, určený na zachytenie činností útočníka na honeypote, bez toho, aby o tom útočník vedel. Skladá sa z dvoch komponentov. Prvým z nich je klient, ktorý beží na honeypote, jeho účelom je zachytiť všetky aktivity útočníka (stlačenia klávesov, nahrávanie súborov, heslá) a následne tieto údaje odoslať na server. Ten je teda druhou zložkou a jeho úlohou je zhromažďovať údaje z jednotlivých honeypotov.

Jiang a Wang v [51] uvádzajú, že existujúce prístupy na monitorovanie virtuálnych honeypotov možno rozdeliť do dvoch hlavných kategórií, a to na interné a externé prístupy. Externé monitorovanie zostáva pre monitorovaný honeypot neviditeľné, ale za cenu straty schopnosti zachytiť interné systémové udalosti, ako sú napríklad vykonané systémové volania.

Na druhej strane, interné monitorovanie nasadzuje senzory vo vnútri monitorovaných honeypotov, a tým poskytuje rozsiahly pohľad na rôzne aspekty systému. Senzory vo vnútri honeypotov však môže útočník detegovať a deaktivovať.

3.2 Detekcia nízko interaktívnych virtuálnych honeypotov

Mukkamala a spol. v článku [53] uvádzajú, že nízko interaktívne honeypoty môžeme detegovať na úrovni počítačovej siete alebo pomocou takzvaného TCP/IP fingerprintingu.

Pri detekcii na úrovni počítačovej siete, napríklad u honeypotu Honeyd [54] je možné využiť časová analýza ICMP ECHO požiadavky. Detekčná technika je založená na jednoduchom pozorovaní, a síce že väčšina honeypotov odpovedá pomalšie na požiadavky ICMP ECHO (ping) v porovnaní s klasickými systémami.

TCP/IP fingerprinting, tiež známy ako odtlačok TCP/IP stack fingerprinting, je analýza dátových polí v pakete TCP/IP na identifikáciu rôznych konfiguračných atribútov sieťového zariadenia. Informácie, ktoré sa z tohto dajú vyčítať, zahŕňajú typ zariadenia, z ktorého paket pochádza a operačný systém, na ktorom je spustený. Pri analýze je pre každé TCP/IP pripojenie vyextrahovaných 49 rôznych kvalitatívnych a kvantitatívnych znakov (napr. sent_packets, received_packets, total_packets). Z výsledkov prezentovaných v článku [53] je zrejmé, že zatiaľ čo Honeyd implementuje základnú funkcionalitu služby, zlyháva, keď sa službu skutočne pokúsime využiť. Autori článku testovali služby HTTP, FTP a SMTP, pričom porovnávali Honeyd a reálne systémy (Linux aj Windows) s rôznymi požiadavkami na tieto služby. Výsledky, ktoré získali, môžeme vidieť v nižšie uvedenej tabuľke. Znak začiarknutia označuje funkciu (príkaz), ktorý bol prítomný, a krížik označuje, že funkcia sa nenašla, teda ju honeypot nepodporoval. Ako môžeme vidieť v Tab. 2., Honeyd zlyhal v nadpolovičnej väčšine funkcií pri testovaní služieb HTTP a SMTP a pri službe FTP podporoval presne polovicu odskúšaných funkcií.

Tab. 3 Porovnanie honeypotu Honeyd a reálneho systému

Služba	Príkaz	Reálny systém	Honeyd
HTTP	GET	✓	✓
	OPTIONS	✓	×
	HEAD	✓	×
	TRACE	✓	×
FTP	USER	✓	✓
	PASS	✓	✓
	MODE	✓	×
	RETR	✓	×
SMTP	HELO	✓	✓
	MAIL	✓	✓
	DATA	✓	×
	VRFY	✓	×
	ETRN	✓	×

3.3 Detekcia testovaním simulovaných služieb

Podobne ako v predchádzajúcej podkapitole, aj Krawetz v článku [55] predstavil myšlienku identifikovať honeypoty tým, že skúmal funkčnosť simulovaných služieb. Vo svojej práci navrhol poslať si e-maily od honeypotu, ktorý sa javí ako SMTP server. Ak nie sú prijaté žiadne e-maily, navrhovaná funkcionalita nie je dostupná a prostredie systému je podozrivé, teda to môže byť nasadený honeypot.

Fu a spol. vo svojej práci [56] predstavili niekoľko metód na detekciu virtuálnych honeypotov, ako je Honeyd, založených na časovom správaní. Používajú Ping, TCP a UDP pakety na určenie času od odoslania segmentu až po doručenie príslušného potvrdenia (round trip time - RTT).

Dahbul a spol. v článku [57] použili modelovanie hrozieb na identifikáciu potenciálnych hrozieb, ktoré odhaľujú existenciu honeypotu, čo ho robí neúčinným. V práci sa rozoberajú rôzne protiopatrenia, ktoré sa ukázali ako účinné na zvýšenie

neidentifikovateľnosti honeypotov. Tieto protiopatrenia aj v práci otestovali na honeypotoch Glastopf, Honeyd, Kippo a Dionaea.

3.4 Detekcia fyzickej prítomnosti honeypotu

Innes a Valli v článku [58] uvádzajú rôzne metódy, ako detegovať honeypot. Prvou z nich je postup, ako detegovať VMWare honeypot (virtuálny honeypot vytvorený pomocou nástroja VMWare). Pred verziou VMWare 4.5, veľké množstvo hardvéru nebolo konfigurovateľné. To znamená, že sa honeypot dal veľmi ľahko odhaliť pomocou predvolenej konfigurácie. Ďalšou chybou je MAC adresa sieťovej karty, pretože časť výrobcu MAC adresy (prvé tri oktety) na virtuálnom sieťovom rozhraní vo VMWare je vždy jedna z konkrétnych troch hodnôt. Teda jednoduchým nástrojom na zistenie MAC adresy (napr. ipconfig v operačnom systéme Windows alebo ip v operačnom systéme Linux) zistíme, či začiatok MAC adresy je zhodný s danými hodnotami a odhalíme tak honeypot. Poslednou slabinou VMWare honeypotov je konfigurácia virtuálneho stroja počas jeho behu. Útočník by mohol potenciálne spustiť tento kód na konfiguráciu a zistiť, že príkaz, ktorý sa pokúša vykonať, bol úspešný, čím by si overil že sa nachádza vo virtuálnom prostredí VMWare (pretože v klasickom prostredí by tieto príkazy nefungovali).

Ďalšou z metód, ktorú uvádzajú autori práce, je detekcia chroot prostredia v honeypote. Chroot [59] na operačných systémoch Unix je operácia, ktorá zmení zdanlivý koreňový adresár na aktuálny proces a pre procesy, pre ktorý je proces rodičom. Program, ktorý je spustený v takomto modifikovanom prostredí, nemôže pomenovať (a teda normálne nemôže pristupovať) k súborom mimo určeného adresárového stromu. Termín chroot sa môže vzťahovať na systémové volanie chroot(2) alebo na program chroot(8). Jedným zo spoločných spôsobov zabezpečenia honeypotov je používanie prostredia chroot. Najjednoduchším spôsobom, ako zistiť, či sa nachádzame v prostredí chroot, bolo spustiť príkaz ls –lia v koreňovom adresári. Treba si všimnúť, že tento príkaz nám ukáže inode (index node) koreňových adresárov „.“ a „..“. Na štandardnom systéme sme mohli vidieť, že inode-y pre tieto dva adresáre sú obe číslo 2. Ak sme ale boli v chroote a spustili by sme rovnaký príkaz, inode-y by boli úplne odlišné (čísla budú rôzne od 2).

Autori sa pokúšali detegovať aj honeypot Honeyd. Ako uvádzajú, prvý hlavný problém s Honeyd je v tom, že má tendenciu odpovedať na pakety, na ktoré by zvyčajne reagovať nemal. Ak mu niekto pošle chybne vytvorený sieťový paket (napríklad paket, ktorý okamžite otvorí a zatvorí spojenie), Honeyd odpovie ako keby prijal platný paket. Toto je však zvyčajne prehliadnuté, keďže útočník je zvyknutý neprijímať odpovede, avšak aj táto vlastnosť môže byť zneužitá na odhalenie honeypotu.

V článku autori súčasne popisujú aj niektoré všeobecné problémy, ktoré nastávajú pri detekcii honeypotov ako takých. Vzhľadom k tomu, že honeypoty zapisujú logy, keď útočník používa zariadenie, vykonávajú sa niektoré inštrukcie navyše. Výsledkom je, že celkový čas vykonania procesu alebo príkazu by sa mohol rozšíriť až tak, že je zrejmé, že sa deje niečo podozrivé. Okrem toho, honeypoty často trpia výrazným zhoršením výkonu, ak sú podrobené vytrvalému útoku a v niektorých prípadoch môžu zlyhať úplne (napr. honeypot Kippo).

3.5 Detekcia UML honeypotu

Innes a Valli v článku [58] predstavujú spôsob, ako detegovať UML honeypot (User Mode Linux, teda linuxové jadro, ktoré môže byť spúšťané zo systému na ktorom beží Linux). UML, teda operačný systém Linux v používateľskom móde, má niekoľko problémov, ktoré môžu byť ľahko zneužité, ak ich útočník pozná. Napríklad, UML nepoužíva reálny disk na ukladanie údajov, ale virtuálne zariadenie, ktoré ukazuje na existujúci súborový systém obsahujúci obraz disku. Pomocou jednoduchých príkazov v príkazovom riadku Linuxu sa vieme pozrieť, či systém obsahuje takýto virtuálny disk. Týmto spôsobom vieme získať prvý náznak toho, že systém môže obsahovať UML honeypot.

V neskorších vydaniach UML bolo teda snahou vývojárov znemožniť odhalenie tohto virtuálneho disku. To však nepomohlo k zabráneniu toho, aby útočník zistil, že sa nachádza v UML prostredí. Toto prostredie obsahuje ďalšie vlastnosti, ktoré môžu byť zneužité na jeho odhalenie. Iný rýchly spôsob, ako ho odhaliť, by bolo napríklad pozrieť sa do adresára /proc/cpuinfo, ktorý ak si dáme vypísať do konzoly, nám vypíše zopár riadkov, z ktorých jeden obsahuje „model name: UML“, čím sme dosiahli náš cieľ a odhalili sme UML prostredie.

3.6 Nástroje na detekciu honeypotov

V nasledujúcich podkapitolách si predstavíme nástroje, pozostávajúce zo skriptov určených na odhalenie rôznych honeypotov. Všetky tieto nástroje sú napísané v jazyku Python a niektoré ich zdrojové kódy sú zverejnené v službe Github. Takéto projekty sú určené na odstránenie chýb, ktoré obsahujú rôzne honeypoty, zneužitím ktorých sa dajú v systéme odhaliť. Ich cieľom je otestovanie prostredia s honeypotom a následné odstránenie týchto chýb.

3.6.1 Checkpot Honeypot Checker

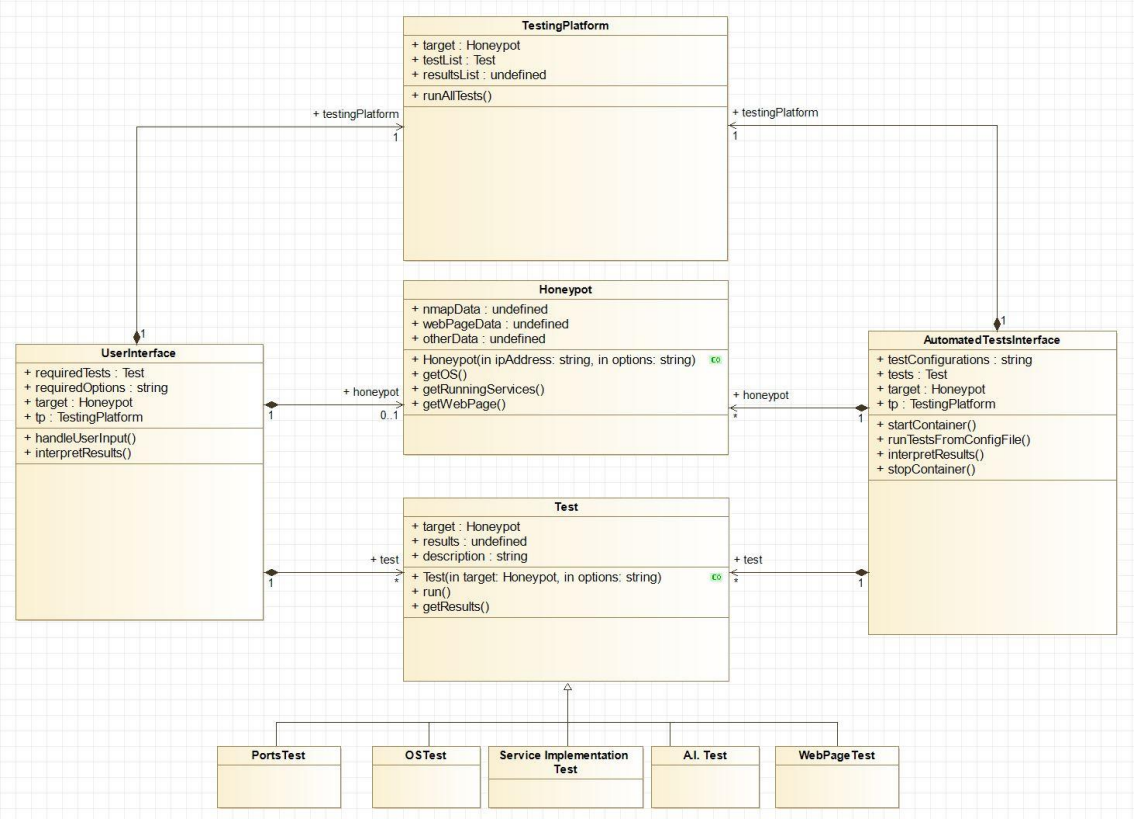
Prvým nástrojom, ktorý si priblížime je Checkpot [60]. Je to nástroj na zisťovanie chýb v konfigurácii honeypotov. Najčastejším dôvodom na odhalenie honeypotu, je ponechanie konfiguračného súboru v pôvodnom, nezmenenom stave.

Checkpot je najčastejšie využívaný na zabezpečenie honeynetu, na overenie toho či sú jednotlivé honeypoty v ňom správne nastavené a nedetekovateľné. Jeho úlohou je preskúmať systém, ktorého IP adresa je mu zadaná, vytvoriť správu so zisteniami a ich závažnosťou. Pomocou tohto nástroja môžu používatelia skontrolovať svoj systém pred uvedením do prevádzky a na základe výsledkov poprípade opraviť chyby, ktoré sa v ňom nachádzajú.

Nástroj Checkpot si po jeho spustení rozdelí vstupné parametre zadané používateľom, z ktorých najdôležitejšia informácia je IP adresa systému, ktorý má za úlohu testovať. Checkpot následne v danom systéme skenuje rôzne jeho vlastnosti, ako napríklad otvorené porty, webstránky, bežiace procesy a iné. Po úvodnom skenovaní prichádza na rad najdôležitejšia fáza, spustenie testov. Jednotlivé testy skúmajú detekovateľnosť honeypotov, pričom využívajú rôzne známe chyby v predvolených konfiguráciách, či nastaveniach. Do Checkpot-u je možné pridávať aj vlastné testy, či upravovať tie, ktoré sa v ňom už nachádzajú, je teda prispôsobiteľný pre rôzne potreby používateľa. Na Obr. 3 [61] je zobrazený UML diagram vyjadrujúci priebeh testovania v nástroji Checkpot.

Checkpot v jeho najnovšej verzii je schopný odhaliť 13 konkrétnych honeypotov (medzi inými aj najčastejšie používané Conpot, Cowrie, Dionaea, Glastopf a iné) pomocou 15 prednastavených testov.

Obr. 3 Testovanie honeypotu v nástroji Checkpot [61]



3.6.2 Honeybee

Ďalším nástrojom na detekciu honeypotov je Honeybee [62]. Je o niečo menej rozsiahly ako Checkpot a je rovnako napísaný v jazyku Python. Pozostáva z 5 skriptov, ktorých úloh je odhalenie honeypotov Amun [63], Glastopf [28] a Kippo [9].

Jeho prvotnou funkcionalitou je podobne ako u nástroja Checkpot skenovanie siete zadanej ako vstupný parameter, teda prehľadanie otvorených portov, bežiacich procesov a podobne. Následne pomocou jednotlivých skriptov Honeybee hľadá v systéme prítomnosť spomínaných 3 honeypotov na základe ich vlastností, pomocou ktorých sú tieto honeypoty detekovateľné.

3.6.3 Honeyscore

Honeyscore [64] je webová služba respektíve nástroj, ktorý je súčasťou bezpečnostnej služby Shodan [65]. Shodan používateľovi umožňuje nájsť rôzne typy systémov pripojených k internetu (napríklad webové kamery, servery, smerovače a

podobne). Shodan na rozdiel od bežných vyhľadávačov neprehľadáva len webové sídla, ale skenuje celú počítačovú sieť, kontroluje teda otvorené sieťové porty, známe sieťové služby či bežiace procesy.

Honeyscore je časť Shodanu, ktorej úlohou je identifikácia honeypotu na danej IP adrese, ktorú jej zadáme. Tento nástroj bol vytvorený na základe známych charakteristík rôznych honeypotov. Honeyscore po preskúmaní IP adresy podá vyhlásenie o tom, či sa podľa neho na danej adrese nachádza honeypot alebo reálny systém.

4 Návrh a implementácia nástroja

V predchádzajúcich kapitolách sme popísali honeypoty a podvodné systémy. Ako sme si ukázali, tieto dve témy sú úzko prepojené, keďže podvodné metódy, ktoré sa využívajú kybernetickom priestore, sú podstatou aktivít vykonávaných honeypotmi. Cieľom našej práce je návrh a implementácia prostredia, resp. nástroja na detekciu honeypotov. V podobných prácach, ktoré sme rozobrali, je vidieť, že neexistuje žiadna univerzálna metóda na detekciu honeypotov. V našom riešení je cieľom nájsť spôsob, ako detegovať honeypoty podľa typov podvodu, ktoré používajú.

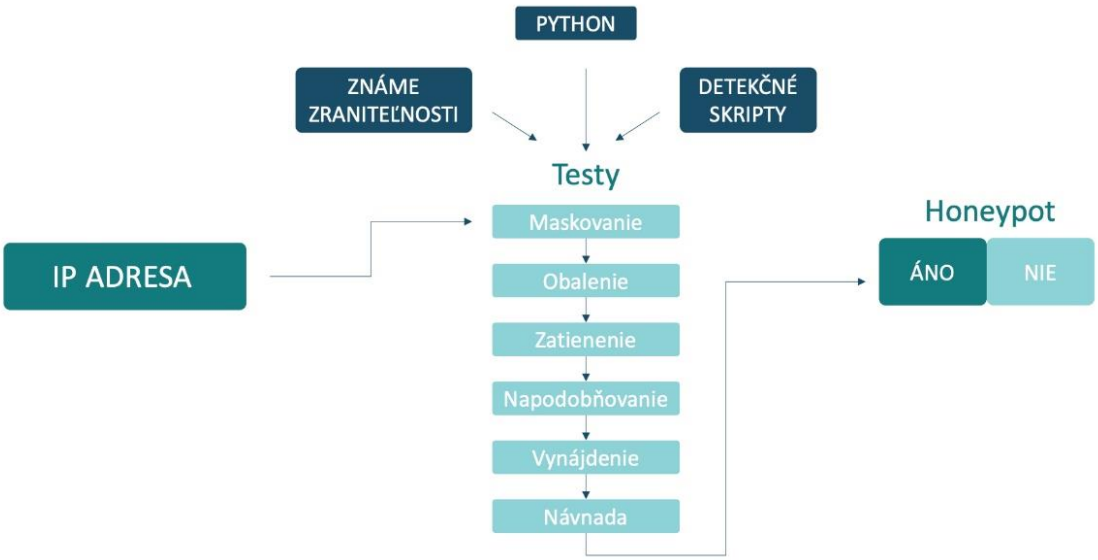
Pri detekcii honeypotov je potrebné, aby nástroj, ktorý vytvoríme, bol schopný detegovať, resp. identifikovať v počítačovej sieti ľubovoľný (vopred neznámy) honeypot. Cieľom je vytvoriť program v jazyku Python, ktorý na vstup dostane IP adresu a na základe naprogramovaných vlastností a metód, určí či táto IP adresa je adresou honeypotu, alebo ide o štandardnú sieťovú službu, resp. operačný systém. Predtým, ako budeme testovať reálne systémy, sme sa pozreli na zoznam najčastejšie sa vyskytujúcich honeypotov. Tieto honeypoty sme si rozdelili do jednotlivých podvodných metód a na základe tohto delenia budeme vedieť pracovať so systémami, o ktorých dopredu nebudeme vedieť, či sú honeypotmi, alebo ide o reálny systém.

Pri prideľovaní honeypotov k šiestim základným podvodným metódam, môže nastať situácia, že jeden honeypot bude spĺňať popis niekoľkých podvodných metód. To nám však nevadí, keďže pri analýze systému, ktorý dostane program na vstup, pre nás nebude dôležité, akú konkrétnu podvodnú metódu systém využíva. Dôležité pre nás bude, že využíva ľubovoľnú z týchto metód, čím budeme vedieť jednoznačne povedať, že sa jedná o podvodný systém. Po rozdelení honeypotov nasleduje tvorba nástroja. Tento program musíme navrhnúť tak, aby bez znalosti vstupu (teda pridelená IP adresa môže ale nemusí byť honeypot) bol schopný rozoznať tento systém.

4.1 Návrh nástroja na detekciu honeypotov

Nástroj na detekciu honeypotov, ktorý je výsledkom našej práce, deteguje honeypoty na základe ich vlastností ako podvodných systémov. Tento nástroj je implementovaný v jazyku Python a na detekciu využíva vlastnosti realizácie jednotlivých typov kybernetických podvodov. Na Obr. 4 je možné vidieť štruktúru nami navrhovaného nástroja.

Obr. 4 Návrh nástroja na detekciu honeypotov



Po zadaní IP adresy do nášho nástroja, nasleduje prvotné skenovanie daného systému, teda kontrola spustených sieťových služieb, otvorených sieťových portov a podobne. Následne táto adresa bude otestovaná skriptami v jazyku Python, ktoré budú zohľadňovať vlastnosti jednotlivých spôsobov kybernetických podvodov. Tieto vlastnosti popíšeme a implementujeme na základe už známych zraniteľností jednotlivých honeypotov a známych detekčných skriptov a nástrojov. Po analýze vlastností systému zo vstupu, nástroj rozhodne o tom, či na danej IP adrese je prevádzkovaný podvodný systém, alebo naopak, reálny operačný systém, či legitímna sieťová služba. Pri návrhu systému sme preskúmali výhody a nevýhody existujúcich riešení na detekciu honeypotov. Súčasne sme analyzovali rôzne detekčné nástroje (skripty), ktoré potencionálne môžu pomôcť pri návrhu jednotlivých modelov zohľadňujúcich jednotlivé typy kybernetického podvodu.

4.2 Implementácia nástroja na detekciu honeypotov

Výsledkom našej práce je nástroj na detekciu honeypotov, ktorý v sebe spája výhody existujúcich nástrojov, navyše využívajúc podvodné metódy a skenovanie počítačovej siete. Jeho ďalšou pridanou hodnotou oproti existujúcim riešeniam, je využitie vytvárania odtlačku systému, pomocou ktorého je nástroj schopný detegovať systém na základe jeho

nastavení v TLS/SSL a SSH komunikácii. Testovanie TLS a SSH protokolov má veľký zmysel najmä z dôvodu, že väčšina najpoužívanejších honeypotov emuluje služby využívajúce tieto protokoly. Navyše implementácie honeynetov v rámci časti zberu údajov obsahujú emuláciu TLS alebo SSH protokolu.

Ako súčasť našej práce sme sa rozhodli využiť vlastnosti nástroja Checkpot [60] tak, aby náš nástroj využíval jeho výhody a zároveň vyhodnocoval svoje testy na základe ich priradení k jednotlivým podvodným metódam. Toto vyhodnotenie spolu s našou detekciou na základe odtlačku systému, nám pomôže ľahšie detegovať honeypoty na základe ich vlastností ako podvodných systémov.

Aby sme boli schopní efektívne ohodnotiť detekciu honeypotov na základe podvodných metód, bolo nutné použiť model rozdielny od nástroja Checkpot. Implementovali sme z tohto dôvodu nástroj, ktorý sme nazvali *DecepScan*.

Nástroj Checkpot je rozdelený do 9 samostatných skriptov, ktoré predstavujú jednotlivé kategórie daných testov a sú prevažne rozdelené podľa toho, na akú službu sa zameriavajú. Testy sú rozdelené do kategórií:

- `default_ftp` – testuje výskyt predvoleného FTP baneru,
- `default_http` – testuje predvolené šablóny so štýlmi webovej stránky, domovskú stránku a správnosť certifikátov,
- `default_imap` – testuje výskyt predvoleného IMAP baneru,
- `default_smtp` – testuje výskyt predvoleného SMTP baneru,
- `default_telnet` – testuje výskyt predvoleného telnet baneru,
- `default_templates` – testuje výskyt súboru s predvolenou šablónou,
- `direct_fingerprinting` – testuje priamy výskyt honeypotu nástrojom nmap,
- `old_version_bugs` – testuje chybnú implementáciu honeypotu Kippo a
- `service_implementation` – testuje implementáciu metód pre protokoly SMTP a HTTP.

V našej implementácii sme zohľadnili prerozdelenie testov podľa toho, akú vlastnosť honeypotov využívajú na ich detekciu. Cieľom bolo zovšeobecniť testy a priblížiť ich k modulu priradzovania honeypotov k podvodným metódam. Prerozdelenením týchto kategórií testov nám vznikli 3 nové kategórie:

- `correct_implementation`,

- `default_confs` a
- `fingerprinting`.

Prvá kategória `correct_implementation` zahŕňa testy, ktoré kontrolujú korektnú implementáciu systému. Test zisťuje operačný systém obsahuje len podporované služby a či má platné certifikáty. Kategória `default_confs` vyhľadáva štandardné konfigurácie a hľadá znaky jednotlivých honeypotov, ktoré vedia honeypot jednoducho identifikovať. Príkladom je kontrola predvoleného FTP baneru honeypotu Dionaea, ktorý má špecifický tvar „`b'220 DiskStation FTP server ready.\r\n'`“.

Poslednou je kategória `fingerprinting`, ktorá obsahuje triedu testujúcu systém nástrojom nmap [66], ktorý dokáže priamo určiť, či je daný systém honeypot alebo nie. Nmap túto detekciu vykonáva na základe vyhľadávania kľúčových slov v popise systému a tiež na základe otvorených portov. Do tejto kategórie zaraďujeme aj nami vytvorené testy na detekciu honeypotov pomocou vytvorenia odtlačku systému, ktoré takisto využívajú metódu odtlačku (fingerprintingu).

Najčastejším princípom nástrojov na detekciu honeypotov je ohodnotenie ich vlastností. To znamená, že je určená premenná, ktorej sa podľa jednotlivých testov priemerne zvyšuje či znižuje priemerná hodnota. Vyhodnotenie tejto premennej po skončení programu určí, či testovaný systém s akou pravdepodobnosťou je alebo nie je honeypot.

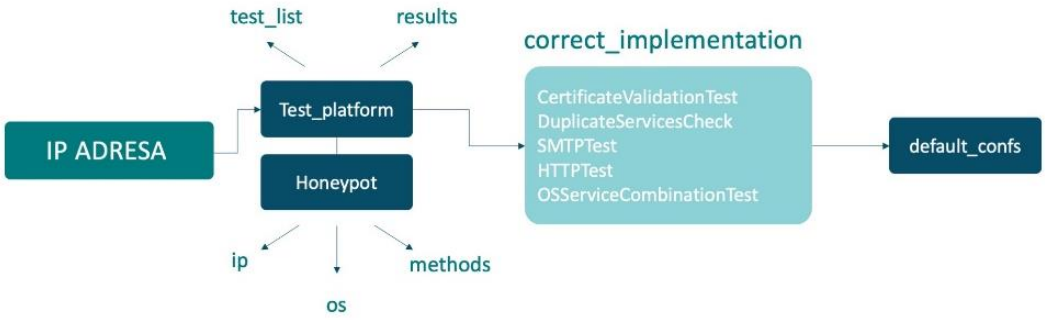
Nástroj Checkpot, ktorým sme sa inšpirovali pri našom nástroji, používa na ohodnotenie honeypotov premennú s označením `karma_value`. Táto premenná nadobúda hodnotu 0 – 100. Používateľa má informovať o dôležitosti tohto testu (0 je najmenej dôležité a 100 je veľmi dôležité) [61]. Dôležitosť testu určuje, nakoľko relevantný je test z hľadiska detekcie honeypotov. Túto premennú autori priradili každému testu podľa ich vlastného uváženia na základe typu testu, ktorému prislúcha.

V našej práci sme sa rozhodli priradiť každému testu hodnotu tejto premennej na základe toho, do akej kategórie kybernetického podvodu daný test spadá. Priradili sme z tohto dôvodu každej samostatnej triede predstavujúcej jeden test, jednu alebo viacero podvodných metód, ktoré predstavuje. Na základe tohto priradenia sme ohodnotili premennú `karma_value`. Spolu sme vytvorili ohodnotenie 18 testovacích tried, ktoré si popíšeme v nasledujúcich kapitolách.

4.3 Kategória testov korektnej implementácie

Po spustení nástroja DecepScan je daný systém postupne testovaný triedami zo všetkých troch kategórií testov. Ako prvá je spúšťaná kategória `correct_implementation`. Po zadaní IP adresy ako vstupu do nástroja DecepScan, sú vytvorené dva objekty – Test_platform a Honeypot (Obr. 5). Trieda Test_platform v sebe uchováva inštanciu výsledkov v premennej `results`, kde sa ráta výsledná hodnota `rating_value`. Nachádza sa v nej tiež zoznam testov, ktorými má daný systém prejsť, pričom predvolene sú spúšťané všetky testy. Trieda Honeypot uchováva inštanciu honeypotu, ktorý testujeme a obsahuje informácie ako IP adresu systému, operačný systém a rôzne implementované metódy, napr. metódu na vypísanie všetkých otvorených portov.

Obr. 5 Prvotný tok údajov pri spustení nástroja DecepScan



Ako prvú sme ohodnotili kategóriu testov `correct_implementation`. Tá obsahuje 5 testov, ktorých ohodnotenie sa nachádza v Tab. 4. V tabuľke je zobrazený názov testovacej triedy, jej pôvodné ohodnotenie `karma_value`, podvodné metódy, ktoré sme danému testu priradili a v poslednom stĺpci, jej nové ohodnotenie. Túto hodnotu sme nazvali `rating_value`. Podvodné metódy sme každej triede priradili podľa toho, na akú využívanú vlastnosť honeypotov sa zameriavajú. Od toho, aké podvodné metódy sme priradili jednotlivým triedam, sa odvíja aj zmena ohodnotenia tejto triedy. Testy pokrývajúce viacero podvodných metód majú prirodzene zvýšené ohodnotenie než testy, ktoré sa zameriavajú len na jednu podvodnú metódu.

Triede `CertificateValidationTest`, sme hodnotu znížili. Tento test kontroluje platnosť certifikátu daného honeypotu/systému. Hodnotu testu sme znížili najmä z dôvodu, že táto hodnota však nie je priamym ukazovateľom honeypotu. Neplatný certifikát môže mať aj legitímny systém. Triedam `DuplicateServicesCheck`, `SMTPTest` a `HTTPTest` sme naopak hodnotu zvýšili, keďže predstavujú presnejšie

označenie honeypotu. V systéme kontrolujú, či niektoré zo sieťových služieb nebežia na viacerých portoch a či sú služby SMTP a HTTP správne implementované. Inými slovami či sú implementované všetky ich metódy. Tieto ukazovatele sú typickým znakom podvodného systému. Z tohto dôvodu bolo ich ohodnotenie zvýšené. Podvodné systémy, ako napríklad honeypoty predstavujúce konkrétnu sieťovú službu, zvyknú byť ľahko detekovateľné práve na základe overenia implementácie všetkých metód daných služieb. Poslednej triede z danej kategórie s názvom `OSServiceCombinationTest` sme ponechali pôvodnú hodnotu rovnú `karma_value`, keďže trieda testuje správnu kombináciu operačného systému a služieb. Táto testovacia trieda napríklad kontroluje, či sa v operačnom systéme Linux nenachádza aplikácia z balíka Microsoft Office. Túto skutočnosť trieda považuje za náznak podvodu, označí teda nesprávne nastavený operačný systém.

Tab. 4 Ohodnotenie kategórie correct_implementation

Názov testu	karma_value	Podvodné metódy	rating_value
CertificateValidationTest	20	zatienenie napodobňovanie vynájdenie	10
DuplicateServicesCheck	30	napodobňovanie vynájdenie	40
SMTPTest	60	maskovanie obalenie napodobňovanie návnada	70
HTTPTest	60	maskovanie obalenie napodobňovanie návnada	70
OSServiceCombinationTest	90	maskovanie obalenie návnada	90

4.4 Kategória testov predvolenej konfigurácie

Ďalšou ohodnocovanou kategóriou v poradí je kategória testov predvolenej konfigurácie (`default_confs`). Táto kategória obsahuje 10 testovacích tried (Obr. 6). Z toho triedy `DefaultFTPBannerTest`, `DefaultSMTPBannerTest`, `DefaultTelnet-BannerTest`, `DefaultTemplateFileTest` a `KippoErrorMessageBugTest` mali nastavenú hodnotu `karma_value na 100`. To znamená, že tieto testy vedia stopercentne označiť daný systém ako honeypot. Tieto testy kontrolovali konkrétne predvolené banery na vybraných službách honeypotov. Banery sú v kóde uložené v premenných, je preto jednoduché si do týchto testov doplniť vlastný baner, ktorý chceme v budúcnosti v ľubovoľnom systéme identifikovať. Tieto triedy teda identifikujú predvolenú nezmenenú konfiguráciu honeypotu. V súčasnosti väčšina predvídateľných reakcií počítačových systémov poskytuje útočníkom cenné informácie o tom, ako ich infiltrovať [66]. Týmto testovacím triedam sme preto aj `rating_value` nastavili na hodnotu 100.

Obr. 6 Druhotný tok údajov v nástroji DecepScan

Testovacie triedy z kategórie `default_confs`, ktorým bolo zmenené ohodnotenie môžeme vidieť v Tab. 5. V tejto kategórii sme zvýšili ohodnotenie tried `DefaultStyleSheetTest`, `DefaultWebsiteTest` a `DefaultGlastopfWebsiteTest`. Okrem zmeny ohodnotenia sme navyše pridali do testovacej triedy `DefaultStyleSheetTest` identifikáciu nástroja T-Pot. Táto trieda generuje odtlačok (hash hodnotu) predvolenej šablóny so štýlmi webovej stránky. Do tohto testu sme teda pridali nami vytvorený SHA256 odtlačok (hash) šablóny webovej stránky T-Potu, čo zabezpečí jeho detekciu pri použití nášho nástroja DecepScan.

Triede `DefaultIMAPBannerTest` odhaľujúcej predvolený banner služby IMAP sme ohodnotenie znížili, keďže nepredstavuje natoľko presnú identifikáciu honeypotu ako iné testy s ohodnotením 90. Poslednej triede z kategórie `default_confs` s názvom `DefaultServiceCombinationTest` sme ponechali pôvodnú hodnotu 50.

Tab. 5 Ohodnotenie kategórie default_confs

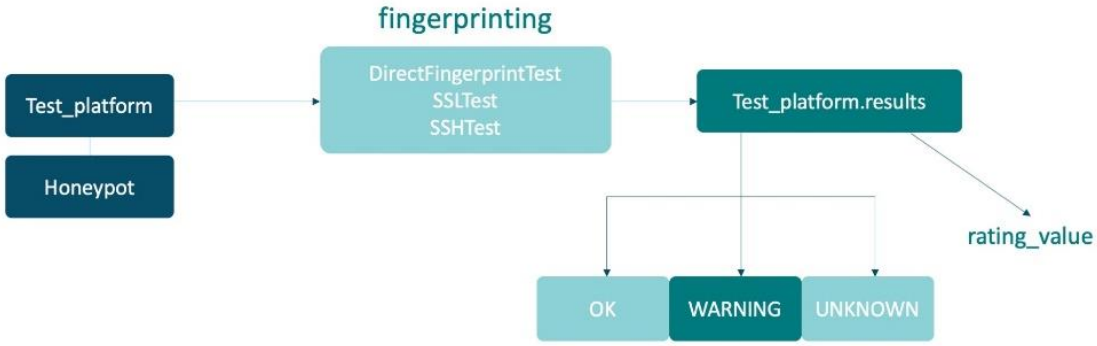
Názov testu	karma_value	Podvodné metódy	rating_value
DefaultStylesheetTest	30	maskovanie	50
DefaultWebsiteTest	60	zatienenie vynájdenie	70
DefaultGlastopfWebsiteTest	60	obalenie zatienenie napodobňovanie	80
DefaultIMAPBannerTest	90	napodobňovanie návnada	70
DefaultServiceCombinationTest	50	zatienenie vynájdenie návnada	50

4.5 Kategória fingerprinting

Poslednou kategóriou testov je kategória `fingerprinting`, obsahujúca 3 testovacie triedy (Obr. 7). Jedna z týchto tried je test nástrojom nmap a zvyšné 2 testy sme implementovali v rámci našej práce s využitím vytvárania odtlačku systému. Testovacia trieda `DirectFingerPrintingTest` testuje systém nástrojom nmap, ktorý priamo identifikuje honeypot. Jej hodnota `karma_value` bola nastavená na 100. Toto ohodnotenie sme sa rozhodli z tohto dôvodu ponechať. Našim dvom triedam `TLSTest` a `SSHTest` sme takisto priradili hodnotu 100, keďže odtlačok systému na základe špecifikácie SSL/TLS alebo SSH komunikácie predstavuje jednoznačnú identifikáciu honeypotu. Tieto dva testy identifikujú honeypot na základe odtlačku systému. Okrem toho, že sme ich pridali ako samostatné triedy do testov, sú taktiež ako metódy súčasťou

samostatného nástroja DecepScan, ktorý sme vytvorili v rámci tejto práce a popíšeme ho v nasledujúcej kapitole.

Obr. 7 Konečný tok údajov v nástroji DecepScan



Metódy TLSTest a SSHTest koncepčne vychádzajú z algoritmov JA3S a HASSH, ktoré sme popísali v predchádzajúcich kapitolách. Výhodou našej implementácie je však forma vstupu. Spomínané algoritmy a ich implementácie vyžadujú na vstupe pcap (packet capturing) súbor, teda odchytené pakety zo sieťovej komunikácie. Pred ich použitím si z tohto dôvodu používateľ potrebuje tieto súbory vytvoriť. Naša implementácia poskytuje ľahšie používanie, keďže na vstupe potrebuje iba IP adresu a celú sieťovú komunikáciu, spolu so získaním požadovaných informácií z tejto komunikácie spracuje s pomocou tohto údaju (IP adresy).

4.5.1 Testovacia trieda TLSTest

Naša trieda `TLSTest` dostane na vstupe IP adresu systému. Jej výstupom je odtlačok (hash) vybraných hodnôt nadviazanej SSL komunikácie. Na vytvorenie tejto komunikácie využívame knižnice `ssl` a `socket` jazyka Python. Využili sme triedu `ssl.SSLContext`, ktorá uchováva rôzne dáta ako napríklad možnosti konfigurácie SSL, certifikáty a súkromný kľúč [67]. Pomocou tohto kontextu si v programe vytvoríme soket, ktorý sa pripája na zadanú IP adresu a port 443, čo je štandardný port pre protokol HTTPS. Tento port je vhodný na vytvorenie našej komunikácie, pretože protokol HTTPS využíva protokol HTTP spolu s SSL alebo TLS protokolom. Z vytvoreného soketu získame informácie potrebné pre vytvorenie špecifického odtlačku tejto šifrovanej komunikácie.

```
1. from ssl import SSLContext
2. sslcontext = SSLContext()
3. sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4. sslsocket = sslcontext.wrap_socket(sock, server_hostname=host,
   do_handshake_on_connect=True)
5. sslsocket.connect((ip, 443))
```

Výstupom programu je MD5 odtlačok (hash) vybraných hodnôt nadviazanej komunikácie. Hodnoty vybrané pre vytvorenie tohto odtlačku sú názov šifry vybranej na nadviazanie komunikácie so serverom, verzia SSL protokolu, ktorá bola použitá a počet skrytých bitov komunikácie. Tieto hodnoty sú spojené čiarkou a z tohto reťazca je následne vytvorený odtlačok (hash).

Príkladom je reťazec ECDHE-RSA-AES256-GCM-SHA384,TLSv1/SSLv3,256 a jeho výsledný odtlačok 81e79e42c8734fe9e287c9929819d822. Ak sa tento odtlačok zhoduje s niektorým z vopred vytvorených odtlačkov systémov uložených v programe DecepScan, nástroj označí systém ako honeypot.

4.5.2 Testovacia trieda SSHTest

Trieda `SSHTest` rovnako dostáva na vstup IP adresu systému. Jej výstupom je takisto odtlačok systému, avšak vytvorený je z informácií získaných z nadviazanej SSH komunikácie. Pri implementácii sme využili triedu knižnicu `socket`, v ktorej pomocou metód `send()` a `recv()` komunikujeme so serverom. Program sa pripája na zadanú IP adresu a na port 22, čo je štandardný port pre SSH komunikáciu. V prípade, ak služba SSH je prevádzkovaná na inom porte, čo je bežné v prípade SSH honeypotov Kippo a Cowrie, v programe je možné daný port zmeniť podľa testu sieťových služieb vykonaných na začiatku testovania daného zariadenia nástrojom nmap.

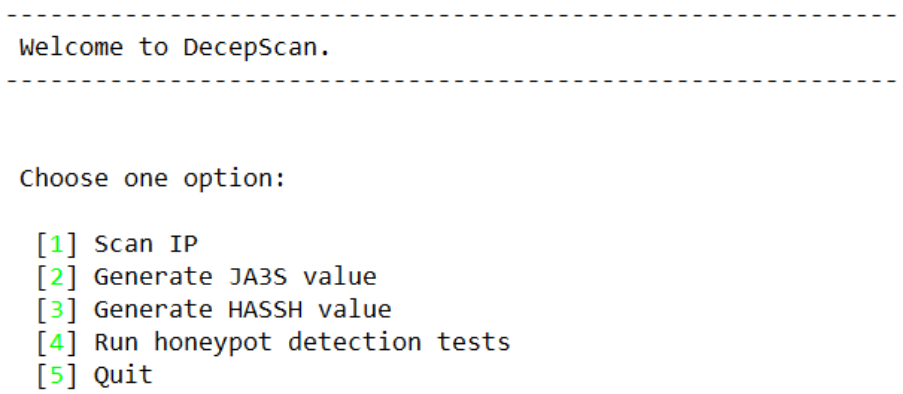
Nadviazanie SSH komunikácie začína TCP handshake-om (3 pakety), nasleduje výmena verzií (2 pakety) a napokon pre nás najpodstatnejšia časť, výmena takzvaných SSH_MSG_KEXINIT paketov [42]. Z komunikácie v programe získame informácie, ktoré obsahujú tieto pakety a z nich je vytvorený jedinečný odtlačok systému. Tento paket je rozdelený na 4 časti: `Key Exchange methods`, `Encryption`, `Message Authentication` a `Compression`. Obsahuje teda informácie slúžiace na dohodu jednotlivých strán o metódach výmeny kľúčov, šifrovaní údajov, integrite správ a o použitej kompresii. Obsah serverovského SSH_MSG_KEXINIT paketu je rovnako

ako v triede TLSTest zreťazený pomocou čiarky. Z tohto reťazca je následne vytvorený MD5 odtlačok (hash), ktorý predstavuje odtlačok systému. Ak sa tento odtlačok zhoduje s niektorým z vopred vytvorených odtlačkov systémov uložených v programe DecepScan, nástroj označí systém ako honeypot.

4.6 Nástroj DecepScan

Pre podrobnejšiu detekciu honeypotov pomocou vytvárania odtlačku systémov sme rozšírili náš nástroj DecepScan o možnosti skenovania siete a vytvorenia odtlačkov systému. Tento nástroj po jeho spustení vyzve používateľa, aby sa rozhodol aké kroky chce vykonať, čo môžeme vidieť na Obr. 5, ktorý predstavuje ukážku obrazovky po spustení programu. Na výber sú štyri základné možnosti.

Obr. 8 Spustenie programu DecepScan



Používateľ má okrem ukončenia programu na výber štyri možnosti, a to skenovanie systému pomocou zadanej IP adresy, vygenerovanie JA3S odtlačku systému, vygenerovanie HASSH odtlačku systému alebo spustenie testov nástroja DecepScan. Vytvorenie odtlačkov môže byť použité napríklad pri následnom využívaní nástroja DecepScan. Ako sme popísali v predošlých kapitolách, DecepScan obsahuje vopred vytvorené odtlačky systémov. Tieto odtlačky sú vytvorené práve týmito našimi programami. Ak si teda používateľ vytvorí jedinečný otlačok svojho systému či honeypotu, môže ho potom v budúcnosti jednoducho detegovať pridaním jedného riadku kódu do nášho nástroja.

Prvou možnosťou je celkové skenovanie systému na základe jeho IP adresy. Používateľovi je najprv ponúknutá možnosť skenovania sieťových portov, kde ako vstup programu je potrebné zadať počiatočný a konečný port na skenovanie. Na Obr. 6 môžeme vidieť ukážkový výstup programu. Po skenovaní zadaného rozsahu portov program vypíše, ktoré porty sú v danom systéme otvorené. Používateľovi umožní vykonať ďalšie skenovanie portov, alebo preskúmať daný systém pomocou nástroja `nmap`.

Ak si používateľ vyberie zobrazenie ďalších informácií získaných pomocou nástroja `nmap`, na obrazovke terminálu sa zobrazia informácie, ktoré sa skenovaním IP adresy podarilo programu získať. Táto metóda skenovania okrem základných informácií systému zobrazí otvorené porty pre protokoly TCP a UDP, operačný systém bežiaci na danej IP adrese, meno systému, posledné zapnutie a ďalšie informácie.

Obr. 9 Výstup metódy skenovania IP adresy

```
------------------------------------------------------------
 Welcome to DecepScan.
------------------------------------------------------------
Enter a host or IP to scan:  127.0.0.1
Host IP:  127.0.0.1


Initial port (eg: 80): 440
Ending port (eg: 443): 446


Starting scan, please wait...

Port 443 open
Port 445 open

Scan completed.


Do you want to start another scan? (y/n): n

Do you want to scan host with Nmap? (y/n): y
Starting scan with Nmap, please wait...
```

Možnosti vytvorenia odtlačku systémov metódami JA3S a HASSH v nástroji DecepScan sú formálne upravené testovacie triedy ktoré sme implementovali v našom nástroji. Obe z nich sú popísané v predchádzajúcich dvoch podkapitolách, fungujú teda rovnako ako triedy `SSLTest` a `SSHTest` v nástroji Checkpot. Z tohto dôvodu ich nebudeme znovu popisovať. Ich výstup po spustení v nástroji môžeme vidieť na Obr. 7 a Obr. 8, ktoré zobrazujú ukážky obrazovky po spustení jednotlivých metód.

Obr. 10 Výstup metódy TLSTest

```
--------------------------------------------------------------------------------

     Final MD5 hash is created from values: the name of the cipher being used,
                    the version of the SSL protocol that defines its use,
                    and the number of secret bits being used.

     Current string: ECDHE-RSA-AES256-GCM-SHA384,TLSv1/SSLv3,256

     Final hash:  81e79e42c8734fe9e287c9929819d822

--------------------------------------------------------------------------------
```

Obr. 11 Výstup metódy SSHTest

```
--------------------------------------------------------------------------------

    Final MD5 hash is created from functions:
      Key Exchange methods,
      Encryption,
      Message Authentication
      and Compression.

    Current string: All algorithms that can be seen in SSH_MSG_KEXINIT.

    Final hash:  34de91e4224936ec457a66b3d550f44e

--------------------------------------------------------------------------------
```

4.7 Odporúčania pre tvorbu a používanie honeypotov z pohľadu nedetekovateľnosti

V našej záverečnej práci sme rozobrali rôzne metódy detekcie honeypotov. Tým sme odhalili mnohé ich vlastnosti, ktoré tieto systémy robia detekovateľnými. V praxi musia byť systémy využívajúce kybernetický podvod atraktívne, musia ponúknuť návnadu pre útočníka a musia byť presvedčivé, aby zaistili, že útočníci nevzdajú svoj plánovaný útok [68].

Honeypoty majú rôzne spôsoby na zachytávanie a spracúvanie údajov, avšak ich najpodstatnejšou vlastnosťou je využitie kybernetického podvodu. Túto svoju vlastnosť využívajú, aby útočníkov presvedčili, že sú reálnymi systémami [69]. Najčastejšou metódou na detekciu honeypotu je overenie konfigurácie. Ak daný systém obsahuje

predvolenú nepozmenenú konfiguráciu, zvyčajne sa dá jasne označiť ako honeypot. Napríklad konfigurácia webového honeypotu Glastopf alebo SSH honeypotu Kippo. Pri takejto detekcii sa hľadajú kľúčové slová, či konkrétne reťazce identifikujúce daný podvodný systém. Pri honeypote Amun je napríklad hľadaným reťazcom „b'a200 Lotus Domino 6.5.4 7.0.2 IMAP4\r\n'“, ktorý identifikuje jeho predvolený IMAP baner.

Pri nasadzovaní honeypotov by preto prvým krokom mala byť zmena jeho predvolenej konfigurácie. V niektorých prípadoch spočíva overovanie tejto konfigurácie vo vytvorení odtlačku (hashu) prislúchajúceho súboru. Rovnako je to napríklad pri testovaní predvolenej šablóny so štýlmi webovej stránky v našom nástroji DecepScan. Ak teda v konfigurácii zmeníme ľubovoľný znak, odtlačok sa zmení a detekcia bude zmarená. Rovnako ak vidíme, že v konfigurácii sa nachádzajú akékoľvek slová identifikujúce honeypot (kľúčové slová súvisiace s honeypotmi alebo konkrétny názov), tieto informácie by sme mali pozmeniť.

Ďalšou z metód používaných na detekciu honeypotov je overenie implementácie daného systému, či služby. Pri nasadení honeypotu by sme mali ošetriť, ako daný systém reaguje na požiadavky používateľa. Ak niektoré služby či metódy honeypotu nie sú implementované, útočník by mohol ľahko prísť na to, že nekomunikuje s reálnym systémom. Príklad sme uviedli pri analýze existujúcich riešení detekcie honeypotov, kde bol týmto spôsobom detegovaný honeypot Honeyd. Z tohto dôvodu je v systéme potrebné implementovať metódy, ktoré implementované nie sú, alebo zvoliť ľahšiu cestu a to pripraviť honeypot ako odpovedať v takejto situácii. Chybové hlásenie (alert) konkrétneho honeypotu môže byť príčinou jeho detekcie, avšak chybové hlásenie metódy požadovanej útočníkom môže byť prehliadnuté ako normálny jav.

Spôsobom podobným testovaniu predvolenej konfigurácie, je testovanie predvoleného baneru (napr. výpisu v konzole po spustení programu, systému) alebo šablóny so štýlmi webových stránok. Príkladom je spomínaný IMAP baner honeypotu Amun. Toto testovanie prebieha rovnako ako v predchádzajúcom prípade, teda buď sa hľadajú konkrétne reťazce alebo sa vytvára odtlačok systému. Rovnakým spôsobom sa dá detegovať predvolená webová stránka, alebo aj používatelia systému (prihlasovacie údaje root/toor alebo admin/admin). Je preto rovnako dôležité kontrolovať si tieto súčasti honeypotu, ako kontrolovať jeho predvolenú konfiguráciu.

Záver

Honeypoty sú v dnešnej dobe jedným z najpoužívanejších nástrojov na detekciu útokov, útočníkov a ich metód. Je preto dôležité používať ich vždy so správnou implementáciou a tak, aby nebolo možné ich v systéme odhaliť. Táto vlastnosť, ktorú by mal spĺňať každý honeypot sa nazýva nedetekovateľnosť a zaoberali sme sa ňou v našej práci. Naším cieľom bolo pokúsiť sa detegovať honeypoty a prísť na to, na základe čoho je ich detekcia pri ich implementáciách možná.

Prvým cieľom našej práce bolo preskúmať nedetekovateľnosť honeypotov a analyzovať ich bezpečnosť. Tomuto cieľu sme sa venovali v tretej kapitole popisujúcej podobné práce, kde sme rozobrali rôzne prístupy k detekcii honeypotov. Popísali sme presné kroky ako boli jednotlivé honeypoty v týchto prácach detegované, čitateľ si preto môže vlastnú implementáciu upraviť tak, aby jeho honeypot nebol v systéme detekovateľný. V prvej kapitole práce venujúcej sa honeypotom vo všeobecnosti, sme popísali takisto rôzne ich delenia. V kapitole k podobných prácam, sme analyzovali rôzne detekcie honeypotov na základe týchto delení a kategórií. Na základe tejto analýzy sme určili, že najčastejším spôsobom detekcie honeypotov je detekcia ich štandardnej nepozmenenej konfigurácie. V poslednej kapitole práce sme popísali odporúčania pre tvorbu a používanie honeypotov z pohľadu nedetekovateľnosti, čo bolo taktiež jedným z cieľov tejto práce.

Hlavnou myšlienkou našej práce sú podvodné metódy a ich využitie v honeypotoch, ktoré sme popísali v druhej kapitole. Popísali sme metódy maskovanie, obalenie, zatienenie, napodobňovanie, vynájdenie a návnadu. V kapitole sme tiež uviedli ich využitie v honeypotoch. Na testovanie našej implementácie sme si vybrali nástroj T-Pot, ktorý je kolekciou 16 honeypotov a popísali sme ho v prvej kapitole. Ku každému honeypotu v tejto kolekcii sme preto priradili jednotlivé podvodné metódy a odôvodnili sme každé takéto priradenie.

Druhým cieľom našej práce porovnanie a vyhodnotenie existujúcich prístupov k detekcii honeypotov. Okrem analýzy podobných prác sme sa zamerali aj na existujúce nástroje na detekciu honeypotov a popísali sme ich v tretej kapitole. Najpoužívanejším z nich je online nástroj Shodan Honeyscore, no pre nás zaujímavejším bol nástroj Checkpot. Tento nástroj sa formou podobá implementácii, ktorú sme si vybrali v našej záverečnej práce a tou je nástroj v jazyku Python. Tento nástroj sme preto podrobnejšie rozobrali aj vo štvrtej kapitole a rozhodli sme sa pozmeniť jeho implementáciu. Tento

nástroj sme pretvorili tak, aby jeho vyhodnotenie detekcie honeypotov bolo založené na priradení podvodných metód ku jednotlivým jeho testom. Do jeho implementácie sme tiež doplnili detekciu nástroja T-Pot ako celistvej kolekcie honeypotov.

Posledným cieľom našej práce bolo vytvoriť návrh, implementáciu a vyhodnotenie nástroja na detekciu honeypotov. V rámci tejto záverečnej práce sme vytvorili nástroj DecepScan, inšpirovaný modelom nástroja Checkpot, ktorý je okrem iného schopný detegovať honeypot na základe jedinečného odtlačku systému. Odtlačok systému, teda fingerprinting metóda bola pred našou implementáciou v existujúcich nástrojoch využívaná iba s použitím nástroja nmap. Ten však nevytvára odtlačky systému, iba kontroluje jeho vlastnosti (otvorené porty, reťazce identifikujúce podvodný systém). Naša implementácia predstavuje jedinečnú detekciu na základe vytvorenej TLS/SSL alebo SSH komunikácie. Metódy, ktoré sme implementovali v jazyku Python vytvárajú odtlačky systému na základe špecifických informácií, ktoré sme získali z týchto vytvorených komunikácií.

Zoznam použitej literatúry

1. DAHBUL, R. N.; LIM, C.; PURNAMA, J., 2017. Enhancing Honeypot Deception Capability Through Network Service Fingerprinting

2. JOSHI, R. C.; SARDANA, Anjali, 2011. Honeypots: a new paradigm to information security. CRC Press.

3. AKKAYA, D.; THALHOTT, F. 2010. Honeypots in network security.

4. SPITZNER, L., 2002. Honeypots: Tracking Hackers.

5. Dionaea, 2011 [online] Dostupné z: http://honeynet.org/project/dionaea

6. Honeyd, 2008 [online] Dostupné z: http://www.honeyd.org/

7. The Honeynet Project: Know Your Enemy: Sebek2 A kernel based data capture tool, 2003.

8. Argos, 2014 [online] Dostupné z: https://www.few.vu.nl/argos/

9. Kippo - SSH Honeypot, 2016 [online] Dostupné z: https://github.com/desaster/kippo

10. Thug, 2019 [online] Dostupné z: https://buffer.github.io/thug/

11. Release T-Pot 19.03, 2019 [online] Dostupné z: https://dtag-dev-sec.github.io/mediator/feature/2019/04/01/tpot-1903.html

12. DTAG Community Honeypot Project, 2019, [online] Dostupné z: https://dtag-dev-sec.github.io/

13. Docker, 2019, [online] Dostupné z: https://www.docker.com/

14. Docker: What is a Container?, 2019, [online] Dostupné z: https://www.docker.com/resources/what-container

15. ELK Stack: Elasticsearch, Logstash, Kibana, 2019 [online] Dostupné z: https://www.elastic.co/what-is/elk-stack

16. Elasticsearch, 2019 [online] Dostupné z: https://www.elastic.co/what-is/elasticsearch

17. Logstash: Collect, Parse, Transform Logs, 2019 [online] Dostupné z: https://www.elastic.co/products/logstash

18. Kibana: Explore, Visualize, Discover Data, 2019 [online] Dostupné z: https://www.elastic.co/products/kibana

19. Honeypot ADB Honey, 2019 [online] Dostupné z: https://github.com/huuck/ADBHoney

20. Android Debug Bridge, 2019 [online] Dostupné z: https://developer.android.com/studio/command-line/adb

21. Cisco ASA Honeypot, 2019 [online] Dostupné z: https://github.com/Cymmetria/ciscoasa_honeypot

22. Cisco Adaptive Security Appliance (ASA) Software, 2019 [online] Dostupné z: https://www.cisco.com/c/en/us/products/security/adaptive-security-appliance-asa-software/index.html

23. Zraniteľnosť CVE-2018-0101, 2019 [online] Dostupné z: https://nvd.nist.gov/vuln/detail/CVE-2018-0101

24. Honeypot Conpot, 2019 [online] Dostupné z: http://conpot.org/

25. ZUHRI, F. A., 2019. The Illusion Of The Cyber Intelligence Era

26. CARROLL, T. E.; GROSU, D., 2009. A Game Theoretic Investigation of Deception in Network Security

26. Honeypot Dionaea, 2019 [online] Dostupné z: https://github.com/DinoTools/dionaea

27. Honeypot ElasticPot, 2019 [online] Dostupné z: https://github.com/schmalle/ElasticpotPY

28. Honeypot Glastopf, 2019 [online] Dostupné z: https://github.com/mushorg/glastopf

29. Honeypot Glutton, 2019 [online] Dostupné z: https://github.com/mushorg/glutton

30. Honeypot Heralding, 2019 [online] Dostupné z: https://github.com/johnnykv/heralding

31. Honeypot HoneyPy, 2019 [online] Dostupné z: https://github.com/foospidy/HoneyPy

32. Honeypot HoneyTrap, 2019 [online] Dostupné z: https://github.com/armedpot/honeytrap/

33. Honeypot Mailoney, 2019 [online] Dostupné z: https://github.com/awhitehatter/mailoney

34. Honeypot Medpot, 2019 [online] Dostupné z: https://github.com/schmalle/medpot

35. Health Level Seven International, 2019 [online] Dostupné z: https://www.hl7.org/

36. Štandard FHIR, 2019 [online] Dostupné z: https://www.hl7.org/fhir/

37. Honeypot RDPY, 2019 [online] Dostupné z: https://github.com/citronneur/rdpy

38. Understanding the Remote Desktop Protocol, 2019 [online] Dostupné z: (RDP) https://support.microsoft.com/en-us/help/186607/understanding-the-remote-desktop-protocol-rdp

39. Honeypot Snare, 2019 [online] Dostupné z: https://github.com/mushorg/snare

40. Honeypot Tanner, 2019 [online] Dostupné z: https://github.com/mushorg/tanner

41. JA3 - A method for profiling SSL/TLS Clients, 2020 [online] Dostupné z: https://github.com/salesforce/ja3

42. "HASSH" - a Profiling Method for SSH Clients and Servers, 2020 [online] Dostupné z: https://github.com/salesforce/hassh

43. ROWE, N. C.; RRUSHI, J., 2016. Introduction to Cyberdeception

44. HAN, X.; KHEIR, N.; BALZAROTTI, D., 2018. Deception Techniques in Computer Security: A Research Perspective

45. CHANGWOOK, P.; YOUNG-GAB, K., 2019. Deception Tree Model for Cyber Operation

46. DE FAVERI, C.; MONEIRA, A., 2016. Designing Adaptive Deception Strategies

47. AL-SHAER, E.; WEI, J.; HAMLEN, K. W.; WANT, C., 2019. Autonomous Cyber Deception

48. QASSRAWI, M. T.; HONGLI, Z., 2010. Deception Methodology in Virtual Honeypots

49. GARG, N.; GROSU, D., 2007. Deception in Honeynets: A Game-Theoretic Analysis

50. Honeypot Cowrie, 2019 [online] Dostupné z: https://github.com/cowrie/cowrie

51. JIANG, X.; WANG, X., 2007. Monitoring of VM-based High-Interaction Honeypots

52. Sebek, 2006 [online] Dostupné z: http://old.honeynet.org/tools/sebek/

53. MUKKAMALA, S., 2007. Detection of Virtual Environments and Low Interaction Honeypots

54. Honeyd, 2007 [online] Dostupné z: http://www.honeyd.org/

55. KRAWETZ, N., 2004. Anti-Honeypot Technology

56. FU, X., 2006. On Recognizing Virtual Honeypots and Countermeasures

57. DAHNUL, R. N.; LIM, C.; PURNAMA, J., 2017. Enhancing Honeypot Deception Capability Through Network Service Fingerprinting

58. INNES, S.; VALLI, C., 2006. Honeypots: How do you know when you are inside one?

59. chroot, 2019 [online] Dostupné z. https://en.wikipedia.org/wiki/Chroot

60. Checkpot Honeypot Checker, 2019 [online] Dostupné z: https://github.com/vladalexgit/checkpot

61. Checkpot: Tests and TestPlatform framework, 2019 [online] Dostupné z: https://checkpot.readthedocs.io/en/master/test_platform_description.html

62. Honeybee, 2019 [online] Dostupné z: https://github.com/mohitrajain/honeybee

63. Honeypot Amun, 2019 [online] Dostupné z: https://github.com/zeroq/amun

64. Honeyscore, 2019 [online] Dostupné z: https://honeyscore.shodan.io/

65. Shodan, 2019 [online] Dostupné z: https://www.shodan.io/

65. Checkpot: Writing a new Test, 2020 [online] Dostupné z: https://checkpot.readthedocs.io/en/master/writing_a_new_test.html

66. Nmap: the Network Mapper – Free Security Scanner, 2020 [online] Dostupné z: https://nmap.org/

66. ALMESHEKAH, H. M.; SPAFFORD, H. E., 2016. Cyber Security Deception

67. Python ssl.SSLContext, 2020 [online] Dostupné z: https://docs.python.org/3/library/ssl.html#ssl.SSLContext

68. AMMIREZA, N.; HAADI, J. J.; BEI-TSENG, C.; AL-SHAER, E., 2020. HoneyBug: Personalized Cyber Deception for Web Applications

69. ROWE, C. N., Honeypot Deception Tactics

Prílohy

Príloha A: CD médium – diplomová práca v elektronickej podobe, prílohy v elektronickej podobe

Príloha B: Vybrané ukážky nástroja DecepScan

Príloha C: Vybrané ukážky testov kategórie korektnej implementácie

Príloha D: Vybrané ukážky testov kategórie predvolenej konfigurácie

Príloha E: Vybrané ukážky testov kategórie fingerprinting

Príloha B: Vybrané ukážky nástroja DecepScan

```
from os import system
from sys import exit
from time import sleep
from socket import *
import nmap
import platform

host = ""
iport = 0
eport = 0


def options():
    menu()
    os = platform.system()
    if os == "Linux":
        clear = "clear"
    else:
        clear = "cls"
    try:
        print("\n")
        print(" Choose one option: \n")
        option = int(input(
            "  [\033[1;32m1\033[1;m] Scan IP\n"
            "  [\033[1;32m2\033[1;m] Generate JA3S value\n"
            "  [\033[1;32m3\033[1;m] Generate HASSH value\n"
            "  [\033[1;32m4\033[1;m] Run DecepScan\n"
            "  [\033[1;32m5\033[1;m] Quit\n\n\n \033[1;91m=>
\033[1;m"))
    except:
        print("\n\033[1;91mInvalid option.\033[1;m")
        sleep(2)

def scan():
    """ Scanning opened ports """

def scanNmap(host):
    nmScan = nmap.PortScanner()

    ip = gethostbyname(host)
    nmScan.scan(host, str(iport)+'-'+str(eport))
    print("")
    try:
        print("\033[1;91mHostname for given IP is: \033[1;m" +
nmScan[host].hostname())
        print("\033[1;91mState: \033[1;m" + nmScan[host].state())
        print("\033[1;91mOS: \033[1;m" + str(nmScan.scan(host,
arguments='-O')))
        for prot in nmScan[host].all_protocols():
            print("\033[1;91mProtocols found: \033[1;m" + prot)
```

```
        print("\033[1;91mKeys found for host: \033[1;m" +
str(nmScan[host].keys()))

        # TCP scan
        for tcp in nmScan[host].all_tcp():
            print("\033[1;91mPort/s for TCP protocol: \033[1;m" +
str(tcp))

        # UDP scan
        for udp in nmScan[host].all_udp():
            print("\033[1;91mPort/s for UDP protocol: \033[1;m" +
str(udp))

        # IP protocols
        for ip in nmScan[host].all_ip():
            print("\033[1;91mPort/s for IP protocol: \033[1;m" +
str(ip))

        # SCTP protocols
        for sctp in nmScan[host].all_sctp():
            print("\033[1;91mPort/s for SCTP protocol: \033[1;m" +
str(sctp))

    except KeyError as e:
        print(e)
        return
    exit(1)

def ssltest():
    import ssl
    import socket
    from hashlib import md5
    from ssl import SSLContext

    """JA3S test"""


def sshtest():
    import socket
    from hashlib import md5

    """HASSH test"""
```

Príloha C: Vybrané ukážky testov kategórie korektnej implementácie

```
import socket
from .test import *
import http
import ssl


class SMTPTest(Test):
    def check_smtp_implemented(self, server_address, port=25):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(10)
        try:
            s.connect((server_address, port))
        except socket.error as exception:
            self.set_result(TestResult.WARNING, "failed to connect to
smtp server: ", exception.strerror)
            return


class HTTPTest(Test):
    def check_http_implemented(self, server_address, port=80):
        try:
            conn = http.client.HTTPConnection(server_address,
port=port, timeout=5)
            conn.request('HEAD', '/')
            conn.getresponse()
            self.set_result(TestResult.OK, "HTTP implemented")
        except Exception as e:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.settimeout(10)
            try:
                s.connect((server_address, port))
            except socket.error as exception:
                self.set_result(TestResult.WARNING, "failed to connect
to http server: ", exception.strerror)
                return


class CertificateValidationTest(Test):
    def run(self):
        target_ports = []

        if self.target_honeypot.has_tcp(443):
            target_ports += [443]

        if not target_ports:
            self.set_result(TestResult.NOT_APPLICABLE, "Port 443 not
open")
            conn =
http.client.HTTPSConnection(self.target_honeypot.ip, port=port,
timeout=5)
```

```
            try:
                conn.request('GET', '/')
            except ssl.SSLError as e:
                if e.reason == "CERTIFICATE_VERIFY_FAILED":
                    self.set_result(TestResult.WARNING, "Certificate
invalid for", self.target_honeypot.ip, ":", port)

class OSServiceCombinationTest(Test):
    windows_exclusive = ['ms-sql', 'iis', 'windows', 'microsoft']
    linux_exclusive = []

    def run(self):
        os = self.target_honeypot.os
        ports = self.target_honeypot.get_all_ports('tcp')

        if os.lower() == 'linux':
            for port in ports:
                product_description =
self.target_honeypot.get_service_product('tcp', port)
                for s in self.windows_exclusive:
                    if s in product_description.lower():
                        self.set_result(TestResult.WARNING, "Linux
machine is running", product_description)
                        return

class DuplicateServicesCheck(Test):
        def run(self):
        ports = self.target_honeypot.get_all_ports('tcp')
        service_names = {}

        for service, assigned_ports in service_names.items():
            if len(assigned_ports) > 1:
                report += service + "->" + str(assigned_ports) + " "

        if report:
            self.set_result(TestResult.WARNING, "The following
services run on multiple ports:", report)
```

Príloha D: Vybrané ukážky kategórie predvolenej konfigurácie

```
from bs4 import BeautifulSoup
import urllib.request
import urllib.error
import re
import hashlib
import socket

from honeypots.honeypot import ScanFailure

class DefaultFTPBannerTest(Test):
    def run(self):
        known_banners = {
            b'220 DiskStation FTP server ready.\r\n': "dionaea",
            b'220 Welcome to my FTP Server\r\n': "amun",
            b'220 BearTrap-ftpd Service ready\r\n': "beartrap"
        }


class DefaultStylesheetTest(Test):
    def run(self):
        default_hashes = {

'1118635ac91417296e67cd0f3e6f9927e5f502e328b92bb3888b3b789a49a257':
"glastopf",

'fd0131773e1d78a6de2466d5beb9c09288e32de461bd4f4a5dea8e01e71fb0f6':
"t-pot"
        }

class DefaultWebsiteTest(Test):
    def run(self):
        default_hashes = {

'c59e04f46e25c454e65544c236abd9d71705cc4e5c4b4b7dc3ff83fec0e9402f':
"shockpot",

'd405fe3c5b902342565cbf5523bb44a78c6bfb15b38a40c81a5f7bf4d8eb7838':
"honeything",

'351190a71ddca564e471600c3d403fd8042e6888c8c6abe9cdfe536cef005e82':
"dionaea",

'576137c8755b80c0751baa18c8306465fa02c641c683caf8b6d19469a5b96b86':
"amun"
        }

class DefaultGlastopfWebsiteTest(Test):
    def run(self):
        try:
```

```
            try:
                request =
urllib.request.urlopen('http://www.gutenberg.org/files/42671/42671.txt
', timeout=10)
            except:
                request =
urllib.request.urlopen('http://www.gutenberg.org/files/42671/42671.txt
', timeout=10)

            book =
request.read().decode(request.headers.get_content_charset('utf-8'))

class DefaultIMAPBannerTest(Test):
    def run(self):
        known_banners = {
            b'a200 Lotus Domino 6.5.4 7.0.2 IMAP4\r\n': "amun"
        }

class DefaultSMTPBannerTest(Test):
    def run(self):
        known_banners = {
            b'220 mail.example.com SMTP Mailserver\r\n': "amun",
        }

class DefaultTelnetBannerTest(Test):
    def run(self):
        known_banners = {

b'\xff\xfb\x03\xff\xfb\x01\xff\xfd\x1f\xff\xfd\x18\r\nlogin: ':
"telnetlogger",
            b'\xff\xfd\x1flogin: ': "cowrie",
b'\xff\xfb\x01\xff\xfb\x03\xff\xfc\'\xff\xfe\x01\xff\xfd\x03\xff\xfe"\
xff\xfd\'\xff\xfd\x18\xff\xfe\x1f': "mtpot",
            b'\xff\xfb\x01\xff\xfb\x03': "mtpot",
            b'\xff\xfb\x01': "mtpot",
            b'Debian GNU/Linux 7\r\nLogin: ': "honeypy"
        }

class DefaultTemplateFileTest(Test):
    def run(self):
        target_ports = self.target_honeypot.get_service_ports('iso-
tsap', 'tcp')
        target_ports += self.target_honeypot.get_service_ports('s7-
comm', 'tcp')

        default1 = ['Version: 0.0', 'System Name: Technodrome',
                        'Module Type: Siemens, SIMATIC, S7-200',
                        'Serial Number: 88111222',
                        'Plant Identification: Mouser Factory',
                        'Copyright: Original Siemens Equipment']

class DefaultServiceCombinationTest(Test):
    default_ports = {"amun": [21, 23, 25, 42, 80, 105, 110, 135, 139,
```

```
143, 443, 445, 554, 587, 617, 1023, 1025, 1080,
                              1111, 1581, 1900, 2101, 2103, 2105,
2107, 2380, 2555, 2745, 2954, 2967, 2968, 3127, 3128,
                              3268, 3372, 3389, 3628, 5000, 5168,
5554, 6070, 6101, 6129, 7144, 7547, 8080, 9999, 10203,
                              27347, 38292, 41523],
                     "artillery": [21, 22, 25, 53, 110, 1433, 1723,
5800, 5900, 8080, 10000, 16993, 44443],
                     "dionaea": [21, 42, 80, 135, 443, 445, 1433,
1723, 3306, 5060, 5061],
                     "honeypy": [7, 8, 23, 24, 2048, 4096, 10007,
10008, 10009, 10010]
                     }

class KippoErrorMessageBugTest(Test):
    def run(self):
        target_ports = self.target_honeypot.get_service_ports('ssh',
'tcp')

        if not target_ports:
            self.set_result(TestResult.NOT_APPLICABLE, "No open ports
found!")
            return

        for port in target_ports:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.settimeout(5)

            try:
                s.connect((self.target_honeypot.ip, port))
                banner = s.recv(1024)  # TODO use banner?
                s.send(b'\n\n\n\n\n\n\n\n')
                response = s.recv(1024)
                s.close()
            except socket.error:
                self.set_result(TestResult.UNKNOWN, "Can't communicate
with ports")
                return

            if b'168430090' in response:
                self.set_result(TestResult.WARNING, "Old unpatched
version of Kippo detected, please update to the latest version")
                return

            if b'bad packet length' in response:
                self.set_result(TestResult.WARNING, "Old unpatched
version of Kippo detected, please update to the latest version")
                return
```

Príloha E: Vybrané ukážky kategórie fingerprinting

```
class DirectFingerprintTest(Test):
    def run(self):
        ports = self.target_honeypot.get_all_ports('tcp')

        if ports is None:
            self.set_result(TestResult.UNKNOWN, "Port request returned
None")
            return

        for port in ports:
            product_description =
self.target_honeypot.get_service_product('tcp', port)

            if 'honeypot' in product_description.lower():
                self.set_result(TestResult.WARNING, "Service on port",
port, "reported as honeypot directly by nmap")
                return

        self.set_result(TestResult.OK, "No service was fingerprinted
directly as a honeypot by nmap")

class SSLTest(Test):
    def run(self):
        import ssl
        import socket
        from hashlib import md5
        from ssl import SSLContext

        ip = str(self.target_honeypot.address)

        """ Default hashes generated with our method """
        default_hashes = {
            '1b79113ee52f48dfbd509aa9822e3004 ': "t-pot"
        }

        sslcontext = SSLContext()
        sslcontext.verify_mode = ssl.CERT_NONE
        sslcontext.check_hostname = False

        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        host = socket.gethostbyaddr(ip)[0]
        sslsocket = sslcontext.wrap_socket(sock, server_hostname=host,
do_handshake_on_connect=True)

        sslsocket.connect((ip, 443))
        ciphers = ','.join(list(map(str, sslsocket.cipher())))
        sslsocket.close()
        hash = md5(ciphers.encode('utf-8')).hexdigest()
```

```
          try:
                hash = self.target_honeypot.get_sslHash()
          except ScanFailure as e:
                self.set_result(TestResult.UNKNOWN, e)
                continue

          if hash in known_hashes:
                self.set_result(TestResult.WARNING, "Known",
known_hashes[hash], "hash used")
                return
          else:
                self.set_result(TestResult.OK, "No default hashes")

class SSHTest(Test):
    def run(self):
        import socket
        from hashlib import md5

        ip = str(self.target_honeypot.address)

        """ Default hashes generated with our method """

        known_hashes = {
            'cf1c730aa909d81aadf158b5c45c03c7 ': "t-pot"
        }

        conn = socket.create_connection((ip, 22))

        version = conn.recv(50).decode().split('\n')[0]

        conn.send('SSH-2.0-OpenSSH_6.0p1\r\n'.encode())
        ciphers = conn.recv(10000)
        hash = md5(ciphers).hexdigest()
        conn.close()

        try:
                hash = self.target_honeypot.get_sshHash()
          except ScanFailure as e:
                self.set_result(TestResult.UNKNOWN, e)
                continue

          if hash in known_hashes:
                self.set_result(TestResult.WARNING, "Known",
known_hashes[hash], "hash used")
                return
          else:
          self.set_result(TestResult.OK, "No default hashes")
```