Možnosti zapojení Honeypotů pro detekci narušitelů

Bc. Milan Janovič

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Milan Janovič**
Osobní číslo:	**A22343**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Kybernetická bezpečnost**
Forma studia:	**Kombinovaná**
Téma práce:	**Možnosti zapojení Honeypotů pro detekci narušitelů**
Téma práce anglicky:	**Honeypot Connection Options for Intruder Detection**

Zásady pro vypracování

1. Popište možnosti nasazení honeypotů a honeynetů.
2. Proveďte výběr vhodného honeypotu do testovací infrastruktury.
3. Navrhněte implementaci honeypotu a definujte jeho řízení.
4. Proveďte implementaci Vašeho řesení v testovací infrastruktuře.
5. Ověřte reakci honeypotu na útoky.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ADARSH, S a Kurunandan JAIN. Capturing Attacker Identity with Biteback Honeypot. _2021 International Conference on System, Computation, Automation and Networking (ICSCAN), System, Computation, Automation and Networking (ICSCAN), 2021 International Conference on_ [online]. 2021, , 1-7 [cit. 2021-11-30]. ISBN 9781665439862. ISSN edseee.IEEEConferenc. Dostupné z: doi:10.1109/ICSCAN53069.2021.9526371
2. BUZZIO-GARCIA, Jorge. Creation of a High-Interaction Honeypot System based-on Docker containers. _2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4), Smart Trends in Systems Security and Sustainability (WorldS4), 2021 Fifth World Conference on_ [online]. 2021, , 146-151 [cit. 2021-11-30]. ISBN 9781665400961. ISSN edseee.IEEEConferenc. Dostupné z: doi:10.1109/WorldS451998.2021.9514022
3. WANG, Ping a Hubert D'CRUZE. Honeypots and knowledge for network defense. _Issues in Information Systems_ [online]. 2021, **22**(3), 241-254 [cit. 2021-11-30]. ISSN 15297314. Dostupné z: doi:10.48009/3_iis_2021_259--272
4. WU, Chwan-Hwa a J. David IRWIN. _Introduction to computer networks and cybersecurity_. Boca Raton: CRC Press, c2013, xxxix, 1336 s. ISBN 9781466572133.
5. SELECKÝ, Matúš. _Penetrační testy a exploitace_. Brno: Computer Press, 2012, 303 s. ISBN 9788025137529.
6. STALLINGS, William a Lawrie BROWN. _Computer security: principles and practice_. Fourth edition. Chennai: Pearson, [2020], 800 atrn. ISBN 978-93-534-3886-9.
7. KOLOUCH, Jan a Pavel BAŠTA. CyberSecurity. Praha: CZ.NIC, z.s.p.o., 2019. CZ.NIC. ISBN 978-80-88168-31-7.

Vedoucí diplomové práce: **Ing. David Malaník, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Jméno, příjmení: Bc. Milan Janovič

Název bakalářské/diplomové práce: Možnosti zapojení Honeypotů pro detekci narušitelů

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Milan Janovič, v.r.

Ve Zlíně, dne …….

podpis diplomanta

ABSTRAKT

Táto diplomová práca sa zaoberá implementáciou a modifikáciou vybraných honeypotov, špecificky zvolených podľa potrieb cieľovej infraštruktúry. Práca rovnako definuje a implementuje spôsoby tvorby, zberu, úpravy a vizualizácie logov z daných honeypotov. V teoretickej časti teda práca poskytuje základný prehľadom o problematike honeypotov. Vysvetľuje čo to vlastne honeypot je, aké typy poznáme, ako s nimi môžeme interagovať a definuje motiváciu stojacu za ich vznikom a implementáciou. V praktickej časti potom práca popisuje samotný proces implementácie, vyhodnocuje dosiahnuté výsledky a navrhuje niekoľko možných vylepšení.

Kľúčové slova: kybernetická bezpečnosť, kybernetický zločin, infraštruktúra, hacking, honeypot, monitoring, logovanie, vizualizácia

ABSTRACT

This master's thesis deals with the implementation and modification of selected honeypots, specifically chosen according to the needs of the target infrastructure. The thesis also defines and implements methods for creating, collecting, modifying, and visualizing logs from these honeypots. In the theoretical part, the thesis provides a basic overview of the issue of honeypots. It explains what a honeypot actually is, what types are known, how we can interact with them, and defines the motivation behind their creation and implementation. In the practical part, the thesis describes the implementation process itself, evaluates the achieved results, and suggests several possible enhancements.

Keywords: cybersecurity, cybercrime, infrastructure, hacking, honeypot, monitoring, logging, visualization

POĎAKOVANIE

Chcel by som sa touto formou poďakovať pánovi Ing. Davidovi Malaníkovi, Ph.D. za jeho ochotu, rady, trpezlivosť, pohotovú komunikáciu, dôveru a v neposlednom rade jeho odbornosť v oblasti, ktorú práca rieši, bez ktorej by tvorba tejto práce nebola možná.

PREHLÁSENIE

Prehlasujem, že odovzdaná verzia diplomovej práce a verzia elektronická, nahraná do IS/STAG, sú totožné.

OBSAH

ÚVOD

„Knowledge is Power" – Sir Francis Bacon

Práve táto fráza charakterizuje štádium, v ktorom sa aktuálne nachádza (nielen) odbor kybernetickej bezpečnosti. Asi nikoho z nás neprekvapuje, že s príchodom čoraz sofistikovanejšej a výkonnejšej výpočetnej techniky a celkovej digitalizácie nášho bežného a pracovného života sa zvýšila aj vynaliezavosť útočníkov. Tento fakt výrazne zvýšil nároky na zabezpečenie kladené ako na organizácie, tak aj jednotlivcov. Je teda nevyhnutné neustále vyvíjať, zdokonaľovať a v neposledom rade aj implementovať nástroje slúžiace na ochranu informačných systémov a dát ako takých. Aj napriek snahám je väčšina útokov (a spôsobov akým útočníci daný systém napadli) známa až po danom fakte, resp. po danom útoku. Ako teda docieliť, aby sme zámery a nové spôsoby akými útočníci napadajú systémy odhalili ešte predtým, než k samotnému útoku dôjde?

Jedným z možných riešení je práve implementácia Honeypot-ov, ktoré na rozdiel od systémov ako IDS (Intrusion Detection System) alebo IPS (Intrusion Prevention System), nemonitorujú aktuálny stav siete alebo konkrétneho zariadenia, snažiac sa nájsť známky neštandardného, podozrivého prípadne priamo škodlivého chovania. Honeypoty vo svojej podstate slúžia ako návnady pre útočníkov. Jedná sa teda o (zámerne) zraniteľné a ideálne dôkladne monitorované a izolované systémy (alebo celé siete), ktoré majú prilákať pozornosť útočníkov. Primárnym cieľom je prilákať útočníkov, nechať ich daný honeypot napadnúť, s úmyslom pozorovať a zaznamenať aké prostriedky, zraniteľnosti, postupy a pod. útočníci použili. Takto získané poznatky nám následne umožňujú lepšie pochopiť kybernetické hrozby, s ktorými sa stretávame a v neposlednom rade ich vieme aj využiť na zabezpečenie „reálnych" – produkčných systémov, ktoré by následne týmto útokom mali odolať. Existuje teda zjavná motivácia implementovať v našej infraštruktúre práve honeypoty, ktoré simulujú nami využívané služby, aby sme poznatky získané z útokov vedeli použiť na lepšie zabezpečenie našej infraštruktúry. Nemá pre nás zjavný význam získavať poznatky o tom, aké nové spôsoby útočníci v súčasnosti využívajú na napadnutie konkrétneho software, služby, protokolu atď., ak tento software, službu alebo protokol nevyužívame. Výnimkou je implementácia honeypotov za účelom výskumu.

Motivácia za implementáciou honeypotov je teda, ako dúfame, zrejmá. V tejto práci sa teda najskôr pozrieme a hlbšie vysvetlíme čo to honeypot je, typy honeypotov a iné. Taktiež

uvedieme príklad nasadenia reálneho systému s množstvom honeypotov na verejnom internete, ktorý sme v rámci prípravy k tejto práci vykonali.

Následne v praktickej časti implementujeme nami vybrané honeypoty, ktoré budú simulovať služby cieľovej infraštruktúry - do testovacej infraštruktúry. Popíšeme využité technológie, spôsoby implementácie, vykonané zmeny, logovanie, zhromažďovanie a vizualizovanie dát z honeypotov a iné.

Cieľom práce je teda vytvoriť a implementovať sadu honeypotov, ktoré by nám mali umožniť lepšie pochopiť hrozby, ktorým cieľová infraštruktúra čelí a pomôcť jej správcom s efektívnou obranou voči nim.

I. TEORETICKÁ ČASŤ

1 ZÁKLADNÉ POJMY

Predtým, než si v nasledujúcich kapitolách, okrem iného, bližšie rozoberieme motiváciu za vznikom honeypotov (na praktickom príklade), teóriu honeypotov, technológie použité v praktickej časti a praktickú časť samotnú, pozrieme sa na niekoľko pojmov, ktoré by čitateľovi tejto práce mohli pomôcť pochopiť prácu ako takú alebo mu aspoň pripomenúť niektoré z jeho doterajších poznatkov. Je nutné podotknúť, že sa v žiadnom prípade nejedná o vyčerpávajúci výpis všetkých pojmov (a ani o ich vyčerpávajúce vysvetlenie), ktoré by sa v práci mohli objaviť, ale skôr tých, ktorých sa práca týka, sú s ňou úzko spojené alebo sú pre prácu v istom zmysle kritické.

1.1 Informačná bezpečnosť

Organizácie (ale výnimočne aj jednotlivci) potrebujú na zabezpečenie svojho fungovania získavať, spracovávať, prenášať a uchovávať obrovské množstvo dát, resp. informácií. Manuálna práca s dnešným objemom dát, je vo svojej podstate nemožná. Riešením bolo zavedenie informačných systémov a komunikačných technológií. Informačná bezpečnosť sa teda zaoberá ochranou informácií a dát pred neoprávneným prístupom, zneužitím, zmenou alebo zničením (zámerným ale aj náhodným). Je to teda širší pojem, ktorý zahŕňa nie len technické aspekty práce s dátami, ale aj procesy, politiky, štandardy, právne vzťahy a ľudský faktor, pričom na informácie kladie niekoľko tzv. bezpečnostných požiadaviek. Medzi najhlavnejšie patria [1][2]:

1. *dostupnosť* – informácia by mala byť dostupná keď ju potrebujeme,
2. *autentickosť* – príjemca by si mal byť istý tým, že údaje sú zhodné s tými, ktoré odosielateľ poslal – napríklad pomocou využitia digitálnych podpisov,
3. *integrita* – informáciu by nemalo byť možné nepozorovane modifikovať, bez toho, aby si to príjemca všimol, vzhľadom k tomu, že na takúto informáciu sa nie je možné spoliehať – využitie HASH funkcií,
4. *dôvernosť* – informácia by mala byť dostupná len tým osobám, ktorým bola určená alebo tým, ktoré sú oprávnené s ňou nakladať.

1.2 Kybernetická bezpečnosť

Je pojem veľmi úzko spojený s pojmom informačnej bezpečnosti z predchádzajúcej podkapitoly a to až natoľko, že je v niektorých prípadoch označovaná ako informačná bezpečnosť kybernetického priestoru – teda má s pojmom informačnej bezpečnosti takmer

synonymický charakter. Môžeme teda tvrdiť, že sa jedná o množinu postupov, procesov, nástrojov atď., ktorých úlohou je ochrana systémov a dát na nich uložených v kybernetickom priestore, pred útokmi, ktoré by tieto systémy chceli narušiť – zvyčajne za účelom získania, poškodenia alebo zmeny dát na nich uložených. Čím sa opäť dostávame k bezpečnostným požiadavkám z predchádzajúcej podkapitoly. Je teda zrejmé, že zatiaľ čo pojem informačná bezpečnosť sa zaoberá ochranou dát, kybernetická bezpečnosť sa špecializuje na ich ochranu v rámci kybernetického priestoru. [1] [3]

1.3 Kybernetická kriminalita

Bohužiaľ podobne ako v predchádzajúcich dvoch prípadoch nie je pre pojem kybernetickej kriminality všeobecne daná definícia. Vychádzajúc z predchádzajúcich definícií pre informačnú bezpečnosť a kybernetickú bezpečnosť, by sme kybernetickú kriminalitu mohli definovať ako jednanie namierené proti počítačovým systémom a dátam na nich uloženým. Pričom sa pod tento pojem zvykne všeobecne zaradovať akákoľvek trestná činnosť, ktorá je vykonávaná prostredníctvom informačných a/alebo komunikačných technológii, bez ohľadu na jej cieľ (systém, počítačová sieť, dáta, fyzická osoba atď.). [4][5]

S pojmom kybernetickej kriminality sa úzko spája pojem kybernetický útok, resp. teda počítačová bezpečnostná udalosť, počítačový útok, počítačový trestný čin. Tento pojem je najčastejšie definovaný ako nezákonná, nepovolená, či neautorizovaná činnosť, ktorá je mierená voči iným počítačovým systémom, sieťam, jednotlivcom pohybujúcim sa v kybernetickom priestore. Okrem toho sem, ale zaraďujeme aj zločiny (vykonávané v kybernetickom priestore), ako obťažovanie, spreneveru, šírenie či držanie detskej pornografie a iné. [4][5]

1.4 Kryptológia

Kryptológia je téma, ktorú by bolo možné spracovať aj ako samostatnú prácu, v našom prípade sa na ňu pozrieme len veľmi okrajovo, hlavne kvôli vysvetleniu princípov fungovania symetrických a asymetrických šifrovacích systémov, resp. teda šifrovacích kľúčov.

Kryptológia je asi najčastejšie označovaná ako veda o utajení tajných správ a ochrane informácie pri jej prenose, pričom do nej spadajú tri pododbory [6]:

1. *Kryptografia* – obor zaoberajúci sa návrhom, následným vývojom ako aj štandardizáciou šifrovacích systémov (systémov slúžiacich na premenu nešifrovanej informácie – tzv. otvorený text na informáciu šifrovanú – tzv. šifrový text, pomocou matematický princípov).
2. *Kryptoanályza* – je vo svojej podstate opakom kryptografie z bodu 1., vzhľadom k tomu, že jej cieľom je vyhľadávať slabiny v navrhnutých šifrovacích systémoch a testovať ich odolnosť voči útokom.
3. *Steganografia* – jej cieľom nie je znemožniť čitateľnosť danej informácie a následne ju v takejto forme preniesť (ako je to v prípade kryptografie), ale skryť samotný fakt, že informácia vôbec existuje. V tomto prípade hovoríme o metódach ako neviditeľné atramenty, úprave obrazu alebo zvuku, tak aby z neho bolo možné danú informáciu odčítať a pod.

1.4.1 Šifrovacie systémy

Symetrické šifrovacie systémy sú systémy, ktoré na zašifrovanie a dešifrovanie dát využívajú ten istý kľúč. Teda šifrujeme a dešifrujeme tým istým kľúčom. Výhodou je relatívna jednoduchosť procesu tvorby šifrovacích kľúčov, ako aj šifrovacích systémov samotných. Nevýhodou je naopak nutnosť zdieľať daný šifrovací kľúč predom, čím ho môžeme vystaviť riziku odhalenia. [7]

Asymetrické šifrovacie systémy sú systémy, ktoré na zašifrovanie využívajú jeden kľúč – tzv. verejný kľúč (VK) a na dešifrovanie kľúč druhý – tzv. privátny kľúč (PK). Pričom platí, že medzi týmito kľúčmi existuje istá matematická závislosť – hovoríme o tzv. kľúčovom páre, kedy VK je verejne známy a dostupný a PK by mal poznať len jeho vlastník. Výhodou týchto systémov je eliminácia nutnosti dopredného zdieľania kľúča medzi účastníkmi komunikácie, vzhľadom k tomu, že informáciu zašifrujeme VK príjemcu a následne ju je schopný dešifrovať len daný, konkrétny príjemca pomocou svojho PK. Nevýhodou je zvyšujúca sa komplexnosť tvorby kľúčových párov, ako aj samotných šifrovacích systémov. [7]

Hybridné šifrovacie systémy sa snažia kombinovať silné stránky oboch predchádzajúcich typov šifrovacích systémov. [7]

1.5 Počítačová sieť

„Množina počítačov a iných zariadení, ktoré sú vzájomne prepojené, aby bolo možné medzi nimi zdieľať zdroje a informácie.“ [8]

Jedná sa teda o množinu počítačov (alebo celých systémov), ktoré sú navzájom prepojené (teda sem radíme aj komunikačnú infraštruktúru), za účelom vzájomnej komunikácie a výmeny dát. Počítačové siete sa delia z rôznych hľadísk, medzi najrozšírenejšie patrí [4]:

- *Delenie podľa rozľahlosti* – zaraďujeme sem tzv. siete PAN, LAN, MAN a WAN.
    a) Siete PAN (Private Area Network) sú privátne siete zväčša v rozsahu zariadení jednotlivca alebo domácnosti – mobilné telefóny, inteligentné zariadenia (práčky, televízory a pod.), stolové počítače, laptopy atď.
    b) Siete LAN (Local Area Network) – je väčším typom siete ako sieť PAN, ale taktiež pokrýva len malé geografické územie. Jedná sa teda zväčša o prepojenie uzlov v rámci jednej budovy, prípadne viacerých budov. Napríklad sa môže jednať o sieť organizácie, univerzity a pod.
    c) Siete MAN (Metropolitan Area Network) – je ešte väčším typom siete, v porovnaní so sieťami LAN. Jedná sa zväčša o sieť spájajúcu niekoľko sietí LAN, v rámci mesta alebo jeho časti.
    d) Siete WAN (Wide Area Network) – je najväčším typom siete. Siete WAN do seba spájajú jednotlivé siete MAN, pričom pokrývajú veľké geografické územia. V kontexte sietí WAN hovoríme o sieťach v rozsahoch štátov, kontinentov, či dokonca celého sveta.
- *Delenie podľa hierarchie uzlov v sieti* – poznáme dva typy sietí podľa typu hierarchického postavenia jednotlivých uzlov. Siete typu P2P a siete typu klient-server.
    a) Siete typu klient-server sú siete, kde je jeden (tzv. server) alebo viacero sieťových prvkov hierarchicky nadradených ostatným prvkom v sieti (tzv. klient). Tento typ siete je možné centrálne spravovať a zvyčajne platí, že klient žiada server o služby, ktoré daný server poskytuje.
    b) Siete typu P2P (Peer-to-peer) sú siete, kde jednotlivé systémy resp. používatelia danej siete spolu komunikujú priamo, bez centrálneho prvku (servera) - teda hovoríme o komunikácii klient-klient. Každý systém v sieti, teda môže plniť funkciu ako klienta, tak aj servera pre ostatných klientov.

- *Delenie podľa vlastníctva* – podľa vlastníctva delíme siete na tri typy – privátne siete, verejné siete a VPN siete.
    a) Privátne siete sú siete využívajúce privátne rozsahy IP adries, ktoré boli na tento účel špeciálne vyčlenené a nevyužívajú sa vo verejných sieťach. Pokiaľ chce zariadenie s privátnou IP adresou pristupovať do verejného internetu musí dôjsť k prekladu danej privátnej adresy na verejnú alebo musí využiť proxy server. Tento proces bol zavedený najmä, kvôli nedostatku IPv4 adries.
    b) Verejné siete sú siete dostupné širokej verejnosti a môže sa k nim pripojiť prakticky ktokoľvek.
    c) VPN siete umožňujú bezpečné prepojenie počítačových systémov (alebo celých sietí) prostredníctvom nedôveryhodných (verejných) sietí, tak akoby dané systémy boli pripojené k jednej a tej istej dôveryhodnej (privátnej) sieti. Komunikácia medzi zariadeniami býva, z pravidla, šifrovaná a pred začiatkom komunikácie medzi systémami je nutné overiť ich totožnosť – pomocou certifikátov, hesiel a pod.

1.6 Protokol

Zariadenia v počítačových sieťach nie sú jednotné, líšia sa svojím typom, štruktúrou, výrobcom, architektúrou, internými procesmi, účelom a mnohým iným. Aj napriek tomu, sú tieto zariadenia schopné navzájom komunikovať a zdieľať informácie. Toto je docielené práve pomocou tzv. protokolov. Protokol je predom definovaná množina pravidiel, ktoré udávajú ako sú dáta prenášané medzi danými zariadeniami, bez ohľadu na vyššie spomínané rozdiely. Protokoly rozoberajú procesy, ktoré je nutné vykonať na menšie časti, úlohy alebo funkcie, ktoré sú následne vykonávané. Aby bolo možné tieto menšie časti a k nim prislúchajúce modely hierarchicky zoradiť a definovať tak akúsi štruktúru danej komunikácie, boli zavedené tzv. modely, medzi najpoužívanejšie patrí model OSI (zobrazuje **Obrázok 1**) a model TCP/IP (zobrazuje **Obrázok 2**). Je jasné, že vzhľadom k tomu, že fungovanie protokolov je založené na predpoklade, že sú verejne známe a rozšírené, nie je praktické, aby každý mohol voľne vytvárať svoje vlastné proprietárne protokoly. Medzi organizácie, ktoré protokoly publikovali patrí IEEE (The Institute of Electrical and Electronic Engineers), IETF (The Internet Engineering Task Force), ISO (The International Organization for Standardization), ITU (The International Telecommunications Union), W3C

(The World Wide Web Consortium). Protokoly sú teda kľúčovou časťou sieťovej komunikáciu a sú vedome, či nevedome využívané miliardami ľudí každý deň. [4][9]

OSI Model				
Data Unit (protokolová datová jednotka)		Layer (Vrstva)		Function (Funkce)
Host Layers	Data	7	Aplikační	Definuje způsob, jakým komunikují se sítí aplikace, například databázové systémy, elektronická pošta nebo programy pro emulaci terminálů. Používá služby nižších vrstev, a díky tomu je izolována od problémů síťových technických prostředků. Je softwarová.
	Data	6	Prezentační	Specifikuje způsob, jakým jsou data formátována, prezentována, transformována a kódována. Řeší například háčky a čárky, CRC, kompresi a dekompresi, šifrování dat. Je softwarová.
	Data	5	Relační	Koordinuje komunikace a udržuje relaci tak dlouho, dokud je potřebná. Dále zajišťuje zabezpečovací, přihlašovací a správní funkce. Je softwarová.
	Segments (Segmenty)	4	Transportní	Vlastní přenos dat. Definuje protokoly pro strukturované zprávy a zabezpečuje bezchybnost přenosu (provádí některé chybové kontroly). Řeší například rozdělení souboru na pakety a potvrzování. Je softwarová.
Network Layers	Packets (pakety)	3	Síťová	Definice protokolů pro směrování dat. Adresování a směrování dat v síti od zdroje k cíli. Definuje protokoly pro směrování dat, jejichž prostřednictvím je zajištěn přenos dat do požadovaného cílového uzlu. Je hardwarová, ale když směrování řeší PC s dvěma síťovými kartami je softwarová.
	Frames (rámce)	2	Linková	Zajišťuje integritu toku dat z jednoho uzlu sítě na druhý. V rámci této činnosti je prováděna synchronizace bloků dat a řízení jejich toku. Je hardwarová.
	Bits (bity)	1	Fyzická	Definuje prostředky pro komunikaci s přenosovým médiem a s technickými prostředky rozhraní. Dále definuje fyzické, elektrické, mechanické a funkční parametry týkající se fyzického propojení jednotlivých zařízení. Je hardwarová.

Obrázok 1 - Model OSI [4]

Obrázok 2 - Model TCP/IP [4]

1.6.1 Typy protokolov

Aj napriek tomu, že si v nasledujúcej podkapitole, bližšie – aj keď stručne, priblížime niekoľko z protokolov, ktoré sú pre prácu kľúčové, nie je možné v aktuálnej kapitole vymenovať všetky, vzhľadom na ich počet (počítame v stovkách, ak nie tisícoch). Preto ich všeobecne rozdelíme do troch kategórií, podľa funkcie, ktorú spĺňajú [9]:

- *Komunikácia* – komunikačné protokoly sú protokoly, ako názov napovedá, zabezpečujúce komunikáciu zariadení v počítačových sieťach. Zabezpečujú teda procesy od zasielania správ až po prístup na internet.
- *Správa siete* – procesy slúžiace na správu siete, sú procesy zvyčajne procesy na pozadí, ktoré zabezpečujú plynulý chod siete. Patria sem aj procesy, ktoré využívajú správcovia sietí na ich administráciu.
- *Bezpečnosť* – sú protokoly zabezpečujúce bezpečný prenos dát v rámci počítačových sietí, pričom využívajú kryptografické funkcie (viz. **Kapitola 1.4**)

1.6.2 HTTP(S)

Protokol HTTPS je protokol aplikačnej vrstvy, ktorý kombinuje protokol HTTP a SSL (predtým TLS), aby sme zabezpečili komunikáciu medzi webovým serverom a klientom. Protokol HTTP komunikuje so serverom na porte (viz. **Kapitola 1.8**) 80 a protokol HTTPS na porte 443 (viz. **Kapitola 1.8**). Bežný používateľ môže rozdiel medzi HTTP a HTTPS

spozorovať ako rozdiel na začiatku URL (webovej adresy), kedy HTTP adresa začína ako *http://* a HTTPS adresa potom ako *https://*. V dnešnej dobe, už je využívanie HTTPS štandardom, čiže aj samotné webové prehliadače používateľov upozorňujú, keď nie je použitý. Pri použití HTTPS šifrujeme nasledujúce časti komunikácie [10][11][12]:

- URL dokumentu, ktorý od serveru požadujeme a jeho obsah
- obsah formulárov, ktoré užívateľ na danom webe vyplní,
- cookies, ktoré si server a klient vymieňajú,
- obsah HTTPS hlavičky.

Pri použití HTTPS, je možné taktiež overiť pôvod resp. pravosť dokumentov, ktoré boli odoslané pomocou tzv. certifikátov, ktoré server odosiela spolu s požadovanými dátami. Vo veľmi zjednodušenej forme, certifikát pozostáva z verejného kľúča servera, od ktorého požadujeme dáta a zašifrovaného verejného kľúča servera, od ktorého požadujeme dáta, pričom tento verejný kľúč je zašifrovaný privátnym kľúčom tzv. certifikačnej autority – tretej strany, ktorej dôveruje ako odosielateľ (server), tak prijímateľ (klient) a ktorá vydaním daného certifikátu potvrdzuje pravosť daných dát. Klient po prijatí dát, spolu s certifikátom vyžiada verejný kľúč danej certifikačnej autority, dešifruje ho a porovná s dešifrovaným kľúčom, ktorý obdržal v certifikáte, čim overí ich zhodu a teda aj identitu odosielateľa. [10][13]

1.6.3 RDP

RDP (Remote Desktop Protocol) je proprietárny protokol spoločnosti Microsoft Windows, ktorý umožňuje používateľom pripojenie (spolu s grafický rozhraním) a ovládanie iného (než ich vlastného) zariadenia prostredníctvom počítačovej siete. RDP využíva na komunikáciu port 3389 (viz. **Kapitola 1.8**) a aj napriek tomu, že je vybavený rôznymi bezpečnostnými prostriedkami, je náchylný na kybernetické útoky. Hlavne na útoky typu MITM (Man-in-the-middle), ktorý bude vo svojej podstate využitý aj na realizáciu honyepot-u v praktickej časti tejto práce – pochopiteľne, ale nie za účelom kybernetického útoku. [14][15]

1.7 IP Adresa

IP adresa dostala názov podľa protokolu IP (Internet Protocol), patriaceho do rodiny protokolov TCP/IP - viz. **Kapitola 1.6**. Protokol IP slúži na smerovanie a prenos dát, prostredníctvom počítačovej siete. IP adresa potom slúži k jedinečnej identifikácií sieťového

rozhrania (systému) v rámci siete. V súčasnosti najpoužívanejšou verziou je verzia štyri. Jedná sa o 32 bitové číslo, rozdelené na štyri oktety (8 bitové časti) napríklad [4][16]:

192.168.100.50

(11000000.10101000.01100100.00110010)

Obmedzením IP adries verzie štyri je ich počet – 4 294 967 296, čo sa môže na prvý pohľad javiť ako pomerne veľké číslo, avšak je nutné si uvedomiť, že každé zariadenie s pripojením na internet si vyžaduje pridelenie svojej jedinečnej IP adresy, aby ho bolo možné na sieti odlíšiť (a teda adresovať) a že tento počet IP adries je nutné rozprestrieť po celom svete. Jedným z riešení, tohto problému bolo zavedenie privátnych rozsahov pre privátne siete (viz. **Kapitola 1.5**). Jedná sa o IP adresy v rozsahu 192.168.0.0 až 192.168.255.255, 172.16.0.0 až 172.31.255.255 a 10.0.0.0 až 10.255.255.255. Tieto adresy sú potom pomocou protokolu NAT prekladané na jednu verejnú IP adresu. Týmto sme síce ušetrili pomerne veľké množstvo adries, keďže privátne rozsahy sú znovu použiteľné pre všetky interné siete – teda sú jedinečné len v rámci dane internej siete, avšak toto skôr spomalilo než eliminovalo blížiaci sa nedostatok IP adries verzie štyri. Riešením je teda zavedenie IP adries verzie šesť. IP adresa verzie šesť je 128 bitové číslo rozdelené do ôsmich skupín po 16 bitov, pričom sa zapisuje v hexadecimálnej sústave, napr. [4][16]:

2001:0718:1c01:0016:0214:0000:0000:0ca5

Teória IP adries je oveľa širšou témou, než je nutné definovať pre potreby tejto práce, pre širšie pochopenie IP adries viz. [4] alebo [16].

1.8 Port

Vzhľadom k tomu, že dnešné systémy, poskytujú alebo využívajú veľké množstvo služieb udávame za IP adresou z predchádzajúcej kapitoly tzv. port, resp. číslo portu. Port je teda, v našom kontexte, 16 bitové číslo, ktoré operačný systém pevne zviaže (z angl. *bind*) so službou, ktorá na tomto porte prijíma alebo odosiela dáta, napríklad v prípade HTTPS z **Kapitoly 1.6.2** hovoríme o porte 443. Vzhľadom k tomu, že má číslo portu 16 bitov môžeme pokryť 65536 (od 0 do 65535) portov, resp. služieb, pričom sa tento rozsah podľa [17] delí na tri podmnožiny:

- (0-1023) – systémové porty,
- (1024-49151) – používateľské porty,
- (49152-65535) – dynamické a/alebo privátne porty.

Skupina systémových portov, je často označovaná aj ako množina „dobre známych portov“ (z angl. *well-known ports*), vzhľadom k tomu, že sem patria najpoužívanejšie a najznámejšie služby ako napríklad [17]:

- Port 20 a 21 – FTP,
- Port 22 – SSH,
- Port 23 – Telnet,
- Port 25 – SMTP,
- Port 53 – DNS,
- Port 65 a 67 – DHCP,
- Port 80 – HTTP,
- Port 143 – IMAP,
- Port 443 – HTTPS
- a iné.

Registrácia systémových portov k daným službám teda podlieha prísnejším pravidlám.

1.9 Virtualizácia operačných systémov

Virtualizácia operačných systémov je proces, pri ktorom nahrádzame fyzický prostriedok softwarovou abstrakciou. Pri procese virtualizácie teda vytvárame tzv. virtuálne zariadenia (z angl. *virtual machines*), ktoré pomocou software emulujú fyzický systém – v našom prípade budeme virtualizáciu využívať na simuláciu viacerých systémov s operačným systémom Ubuntu. Hlavnou výhodou virtualizácie je **efektívnejšie využívanie fyzických zdrojov hardware**, ktoré máme dostupné, vzhľadom k tomu, že na jednom zariadení môžeme naraz prevádzkovať niekoľko operačných systémov, ktoré dokonca nemusia byť toho istého typu. Dané virtuálne operačné systémy, zdieľajú fyzický hardware, na ktorom sú spustené (pamäť, procesor, grafickú kartu, úložisko atď.), pričom všetky sa chovajú ako nezávislé zariadenia a zväčša využívajú len frakciou zdrojov, ktoré daný hardware má. Medzi ďalšie výhody patrí **jednoduchšie nasadenie a správa**, keďže administrátori nemusia inštalovať separátne hardwarové zariadenia, ktoré sú následne nasadené – nehovoriac o čase potrebnom na ich objednanie, doručenie a pod., stačí len nasadiť virtuálny operačný systém na už existujúci hardware a špecifikovať aké množstvo zdrojov by mal mať dostupné. Tento proces sme schopní z veľkej časti automatizovať. Čo nás privádza k ďalšej z výhod – **zníženiu časov výpadkov**. Administrátori môžu automatizovať proces vytvorenia

a automatickej konfigurácie virtuálneho zariadenia pri výpadku, prípadne v rovnakom čase prevádzkovať viacero redundantných virtuálnych zariadení – čo by pri klasickom prístupe znamenalo kúpu viacerých redundantných serverov. Neodmysliteľnou súčasťou virtualizačného procesu sú tzv. hypervisors. Jedná sa o softwarovú vrstvu, ktorá riadi dané virtuálne zariadenie a tvorí akúsi vrstvu medzi ním a daným hardware, v neposlednej rade taktiež zabezpečuje vzájomnú izoláciu jednotlivých virtuálnych zariadení tak, aby si vzájomne nemohli a nezasahovali do hardwarových zdrojov. V súčasnosti poznáme dva typy hypervisorov [18]:

- Typ 1 alebo „hypervisor priamo na železe“ (z angl. *bare-metal hypervisor*) – úplne nahrádza operačný systém bežiaci na danom hardware. Tento typ je zväčša využívaný na serveroch, kedy server samotný nevyžaduje inštaláciu separátneho operačného systému.
- Typ 2 – je inštalovateľný ako aplikácia do už existujúceho operačného systému, ktorý je na danom hardware nainštalovaný – čo je v mnohých aspektoch jeho výhodou. Hlavnou nevýhodou je znížený výkon, keďže k hardwarovým zdrojom daného zariadenia musí hypervisor pristupovať skrz ďalší medzičlánok – už existujúci operačný systém.

2 HONEYPOT, HONEYNET

Definíciu pojmu honeypot sme v skrate načrtli už v úvode tejto práce, v nasledujúcich podkapitolách sa naň pozrieme ešte bližšie a okrem iného objasníme aké typy honeypotov poznáme na základe rôznych kategórií.

2.1 Čo je to honeypot?

Aj napriek tomu, že sme uvideli, že honeypoty na rozdiel od systémov ako IDS (Intrusion Detection System) alebo IPS (Intrusion Prevention System) nemonitorujú aktuálny stav siete alebo konkrétneho zariadenia, snažiac sa nájsť známky neštandardného, podozrivého prípadne priamo škodlivého chovania, vo svojej podstate patria medzi prostriedky detegujúce neoprávnené preniknutie (z angl. *intrusion detection tools*). Klasické prostriedky slúžiace na detekciu neoprávneného preniknutia sa vo všeobecnosti spoliehajú na dva typy analýzy [10][19]:

- *Behaviorálnu* – v tomto prípade sa snažíme vyhľadávať, detegovať, označovať a prípadne zamedzovať chovanie, ktoré vykazuje známky škodlivosti. Za takéto chovanie môžeme označiť napríklad snahu programu skopírovať svoj obsah do iného programu, resp. jeho zdrojového kódu, presmerovať užívateľa nepozorovane z ním zadanej URL na inú URL, kopírovanie a odosielanie veľkého množstva dát, či pripojenie na známe IP adresy C2C serverov – serverov ovládajúcich veľké botnety (teda skupiny nakazených zariadení, ktoré bez vedomia používateľa vykonávajú príkazy, ktoré sú im zadané ako odosielanie spamu, kybernetické útoky a pod.). Vychádzame z faktu, že existuje enormné množstvo vírusov a iného malware (škodlivého software), ktorý je nutné poznať a aktívne ho v sieti vyhľadávať, aby sme ho vedeli zachytiť. Je teda jednoduchšie zamerať sa na všeobecné vzorce správania, ktoré vykazujú známky škodlivosti – čo je spoločným znakom pre všetok malware.
- *Založenú na signatúrach* (z angl. *signature based*) – za signatúru označujeme jednoznačný znak na základe, ktorého vieme identifikovať konkrétnu hrozbu. Príkladom sú antivírusové programy, ktoré využívajú rozsiahle databázy signatúr známeho malware a aktívne ich vyhľadávajú. Jedná sa teda o presne opačný filozofický smer, než v predchádzajúcom prípade, kedy vychádzame z predpokladu, že chovanie sa mení tak rýchlo, že nie je možné sa naň spoliehať. Nehovoriac o tom, že veľké množstvo legitímnych programov môže vykazovať známky škodlivého chovania, čím bude vytvárať veľké množstvo falošných pozitívov.

Otázkou teda zostáva, načo vlastne strácať čas s tvorbou, implementáciou, monitoringom, správou, údržbou a zabezpečením honeypotov, ak vyššie uvedené typy analýzy spolu s ďalšími systémami ako firewally a pod. pokrývajú širokú škálu nástrojov na odhalenie útočníka. Odpoveďou na túto otázku je fakt, že všetky tieto prostriedky vychádzajú z už známych poznatkov V prípade behaviorálnej analýzy vychádzame zo vzorcov správania, ktoré poznáme a teda ktoré následne vieme vyhľadávať. V prípade analýzy založenej na signatúrach už je daný malware predom známy, jeho signatúra bola vytvorená a teraz ju vieme vyhľadávať – avšak niekto musel byť prvý, niekto si pravdepodobne prešiel úspešných útokom a metódou post-mortem bol následne odhalený spôsob, akým bol útok vykonaný. V prípade firewallov je nutné ich predom nakonfigurovať podľa aktuálnych osvedčených postupov (z angl. *best practices*), ktoré ale opäť vychádzajú z už známych faktov a overených spôsobov obrany. Toto predstavuje najväčšie úskalie klasického defenzívneho prístupu a ponúka možnosť využitia silných stránok honeypotov.

Honeypot je teda „falošné“ zariadenie, systém, služba a pod., ktoré je navrhnuté tak, aby bolo napadnuteľné a slúžilo ako návnada. Jedná sa o zámerne zraniteľné zariadenie, ktorého úlohou je prilákať útočníkov, zaznamenávať ich chovanie a udržať ich pozornosť po čo možno najdlhšiu dobu, za účelom neskoršej analýzy. Takto vieme odhaliť nové taktiky a spôsoby akými útočníci kompromitujú systémy skôr, než sa im tento nový spôsob podarí „vyskúšať“ na produkčnom prostredí. Honeypoty teda okrem pasce predstavujú aj prostriedok odlákania útočníkov od produkčného prostredia, keďže predstavujú príklad taktiky zmätenia (z angl. *deception*) útočníka v oblasti kybernetickej obrany. Zistené informácie následne vieme použiť pre efektívnejšiu obranu a vylepšenia zabezpečenia produkčného prostredia. [10][20][21][22]

Podľa vyššie uvedeného je jasné, že falošné dáta a samotné honeypoty, na ktorých sú umiestnené, nemajú žiadnu produkčnú hodnotu a legitímny užívateľ nemá dôvod s nimi interagovať. Z tohto dôvodu sa vo všeobecnosti akákoľvek vonkajšia interakcia s honeypotom považuje za podozrivú, keďže sa pravdepodobne jedná o útok. Rovnako platí, že akákoľvek komunikácia smerom von z honeypotu, teda pravdepodobne znamená, že bol napadnutý. [10]

2.2 Delenie honeypotov

Predtým, než si uvedieme a definuje jednotlivé možnosti klasifikácie (delenia) honeypotov, je nutné poznamenať, že nasledujúce delenia zaraďujú všetky honeypoty do všeobecných

kategórií. Teda opomíname granulárne delenie, ako napríklad delenie na webové honeypoty, ktoré môžeme ďalej deliť na WordPress honeypoty, OWA (Outlook Web Application) honeypoty atď., vzhľadom k tomu, že takto špecifické delenie honeypotov nie je pre účely tejto práce zmysluplné.

Honeypoty teda môžeme deliť podľa ich úrovne interakcie s útočníkom, podľa typu implementácie, podľa ich spôsobu využitia a podľa ich schopnosti, resp. neschopnosti škálovať. [21]

2.2.1 Delenie podľa úrovne interakcie s útočníkom

Delenie podľa úrovne interakcie s útočníkom predstavuje najširšie používané a diskutabilne najpodstatnejšie delenie. Podľa úrovne interakcie teda honeypoty rozdeľujeme na dva, resp. tri typy [10][22]:

- *Honeypoty s vysokou úrovňou interakcie* (z angl. *High interaction honeypots*) – tento typ honeypotov predstavuje najrealistickejší a najatraktívnejší cieľ pre potencionálnych útočníkov, keďže predstavuje simuláciu celého systému. Teda obsahuje operačný systém, aplikácie, služby a pod. Tento typ honeypotu sa snaží svojou realistickosťou nalákať čo najväčší počet útočníkov, udržať ich pozornosť čo najdlhšie a zozbierať čo najviac a čo najšpecifickejšie dáta o ich činnosti. Nevýhodou je nutnosť tento typ honeypotu veľmi dôkladne odizolovať od zvyšku infraštruktúry, vzhľadom k tomu, že úplný systém by mohol potencionálny útočník využiť k jej napadnutiu.
- *Honeypoty s nízkou úrovňou interakcie* (z angl. *Low interaction honeypots*) – sú opakom predchádzajúceho typu honeypotov. Zväčša sa jedná len o konkrétnu službu resp. časť systému. Tento typ honeypotov z pravidla upúta pozornosť útočníka, len zo začiatku, keďže neposkytuje simuláciu celej služby. Sme teda schopní získať informácie len o počiatočných fázach metodológie útoku, avšak nie o celom jeho priebehu. Výhodou je väčšia izolovanosť od zvyšku infraštruktúry.
- *Honeypoty so strednou úrovňou interakcie* (z angl. *Medium interaction honeypots*) – sú honeypoty, ktoré sa snažia kombinovať silné stránky oboch predchádzajúcich typov. Snažia sa teda poskytovať útočníkovi dostatočne zaujímavý cieľ, tak aby sme udržali jeho pozornosť čo možno najdlhšie a zozbierali čo najväčší počet dát. Pričom minimalizujeme riziko „úniku“ útočníka mimo honeypot.

2.2.2 Delenie podľa typu implementácie

Podľa typu (spôsobu) implementácie rozdeľuje honeypoty na dva typy a to [20][21]:

- *fyzické honeypoty* – sú honeypoty, ktoré fungujú na dedikovanom fyzickom stroji, ktorý bol na tento účel vyčlenený. Sú teda pochopiteľne drahšie a náročnejšie na implementáciu, či nasadenie,
- *virtuálne honeypoty* – sú honeypoty, ktoré na svoje fungovanie využívajú princípy virtualizácie a teda aj výhody a nevýhody tohto prístupu – viz **Kapitola 1.9**.

2.2.3 Delenie podľa spôsobu využitia

Podľa spôsobu využitia honeypoty rozdeľujeme na dva typy [21]:

- *Produkčné honeypoty* – sú honeypoty, ktorých primárnou úlohou je zvýšenie bezpečnosti konkrétnej organizácie – pomocou už spomínaných úloh a výhod honeypotov, ako je odlákanie pozornosti útočníkov, analýzy ich metodológie atď. Konkrétna implementácia honeypotu je teda závislá na potrebách danej organizácie. Logicky teda nebudeme implementovať honeypot predstavujúci falošnú implementáciu NodeJS webovej aplikácie, ak daná organizácia neprevádzkuje žiadnu takúto aplikáciu v produkčnom prostredí a teda dáta získané z prevádzky takéhoto honeypotu nemajú ako prispieť k zvýšeniu bezpečnosti produkčného prostredia danej organizácie. Rovnako je v tomto prípade nutné obzvlášť dbať na zabezpečenie izolácie honeypotu, keďže tento typ honeypotov býva implementovaný priamo v sieti danej organizácie.
- *Výskumné honeypoty* – sú honeypoty, ktoré sa snažia zozbierať, čo najväčší počet dát na výskumné účely. Bývajú teda, z pravidla, nasadzované na verejnom internete, aby mal na ne dosah najväčší počet potencionálnych útočníkov.

2.2.4 Delenie podľa schopnosti škálovať

Delenie podľa schopnosti škálovať (zvyšovať svoj počet) predstavuje najjednoduchšie – takmer až doplnkové delenie. Delíme na dva typy [20]:

- *škálovateľné honeypoty* – majú schopnosť škálovať,
- *neškálovateľné honeypoty* – nemajú schopnosť škálovať.

2.3 Honeynet

Honeynet je vo svojej podstate počítačová sieť tvorená množinou honeypotov, často spolu s aplikáciami, službami, firewallom atď. V prípade honeypotu sme hovorili o jednej „falošnej“ službe/aplikácii, prípadne celom systéme, ktorý slúžil ako návnada pre potencionálnych útočníkov. Honeynety ďalej rozvíjajú tento koncept, pričom tvoria celé „falošné“ siete. Často sa jedná o kópiu produkčnej siete danej organizácie, ktorá sa takto snaží overiť zabezpečenie celej svojej „skutočnej“ (produkčnej) siete, nie len jej konkrétnej časti, prípadne viacerých častí. Výhodou je zvýšenie autenticity z pohľadu útočníka, celá sieť spolu s firewallom, službami, aplikáciami a zariadeniami pôsobí dôveryhodnejšie, než jedno, často až nápadne zraniteľné zariadenie či služba. Výhodou je taktiež, že útočník môže prehľadávaním a napadaním jednotlivých prvkov siete stráviť oveľa dlhšiu dobu a odhaliť oveľa širšiu časť svojho arzenálu. V prípade honeynetov teda hovoríme o akomsi rozšírení myšlienky honeypotov, pričom spolu zdieľajú svoje ciele, výhody, nevýhody a riziká. [23]

II. PRAKTICKÁ ČASŤ

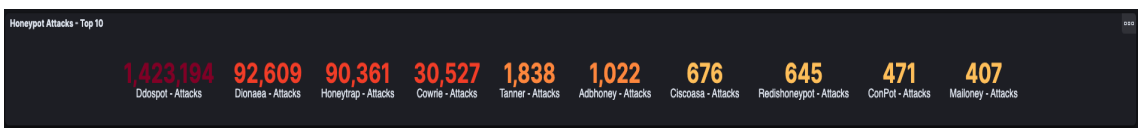
3 ÚVODNÝ EXPERIMENT

Pred začatím realizácie praktickej časti tejto práce, sme v cloudovom prostredí Vultr (viz. *https://www.vultr.com*) implementovali a približne týždeň prevádzkovali inštanciu T-Pot honeypot projektu, o ktorom si viac povieme v nasledujúcej **Kapitole 9**. V aktuálnej kapitole si pri každom honeypote uvedieme službu, ktorú simuloval. Tento experiment sme vykonali na potvrdenie hypotézy o veľkosti aktivity, ktorú útočníci denne na verejnom internete vykonávajú a teda toho, že čas strávený implementáciou, monitoringom a údržbou honeypotov môže organizáciám výrazne pomôcť v snahe zvýšiť zabezpečenie ich siete. Výsledky nášho sedemdenného pokusu si preto, v skratke (budeme sa zameriavať na honeypoty, ktoré zaznamenali u útočníkov najväčší „úspech"), odprezentujeme v tejto kapitole.

3.1 Výsledky

3.1.1 Všeobecné zhrnutie

Počet útokov podľa honeypotu zobrazuje **Obrázok 3**. Najväčší počet útokov zaznamenal Ddospot (1 423 194 útokov) – čo pochopiteľne vychádza z jeho povahy, keďže sa jedná o honeypot stavaný na prilákanie DDOS útokov. Druhý najväčší počet zaznamenal Dionaea honeypot (92 609 útokov) a hneď za ním nasledoval Honeytrap honeypot (90 361 útokov) – oba tieto honeypoty v sebe zahŕňajú veľké množstvo služieb, čo im určite pomohlo zaznamenať väčší počet útokov. Nasledoval Cowrie honeypot (30 527 útokov), ktorý simuluje SSH a Telnet služby. Piaty najväčší počet útokov zaznamenal webový honeypot Tanner resp. Snare (1 838 útokov). Hneď za ním sa nachádza ADB (Android Debug Bridge) honeypot Adbhoney (1 022 útokov). Nasleduje ho Ciscoasa honeypot (676 útokov), predstavujúci falošnú príležitosť útoku na Cisco ASA komponentu (CVE-2018-0101) a databázový Redishoneypot (676 útokov). Posledné dve miesta medzi prvými desiatimi honeypotmi – podľa počtu útokov uzatvárajú ConPot (471 útokov) a SMTP honeypot MailHoney (407 útokov). ConPot je honeypot simulujúci zraniteľnosť v priemyselných riadiacich systémov (z angl. *industrial control system*). [24]



Obrázok 3 - Vultr experiment zhrnutie, Zdroj: vlastný

Nasledujúce obrázky predstavujú zhrnutie výsledkov experimentu na základe rôznych kritérií, medzi ktoré patrí zdrojová krajina a IP adresa útočníka, AS resp. ASN útočníka, reputácia IP adresy útočníka a iné.

Obrázok 4 - Vultr experiment - ASN zobrazuje počet útokov na základe jednotlivých AS – Autonómnych Systémov (z angl. *Autonomous systems*). Tieto systémy vo svojej podstate predstavujú jasne definovaný zoznam IP adries dostupných v danej sieti (tzv. IP prefixov), ktoré môžu byť spravované jedným alebo viacerými sieťovými operátormi, pričom musia mať jasne definovanú internú smerovaciu politiku. Slúžia teda sieťovým operátorom na smerovanie v rámci daného AS a medzi inými AS, resp. inými sieťovými operátormi. [25]

AS	ASN	Počet
61588	DIGITAL PROVEDOR DE ACESSO A INTERNET LTDA	551,067
267593	B.B.S COMUNICACOES LTDA ME	88,567
267074	NEXT PROVEDORES ACESSO LTDA ME	83,042
61587	C. HOKI DA COSTA E CIA LTDA - ME	78,735
266047	Chapnet Servicos de Comunicacao Ltda	76,520
396982	GOOGLE-CLOUD-PLATFORM	55,047
267219	Netware Telecomunicacoes e Informatica EIRELI	36,417
269403	WI-FI NET TELECOMUNICACOES E MULTIMIDIA SCM EIRELI	36,337
4134	Chinanet	17,338
8452	TE Data	15,894

Obrázok 4 - Vultr experiment - ASN, Zdroj: vlastný

Obrázok 5 zobrazuje počet útokov z danej zdrojovej IP adresy. Jedná sa o počet jedinečných útokov z konkrétnej IP adresy. Preto môžeme vidieť USA na prvom mieste aj napriek tomu, že počet útokov z AS Brazílskych sieťových operátorov, bol výrazne vyšší ako počet útokov z AS pochádzajúcich z USA.

Zdrojová adresa	Počet	Krajina
34.77.127.183	47,780	USA
221.224.202.18	9,184	Čína
94.232.43.36	7,022	Holandsko
85.105.103.184	3,199	Turecko
170.238.248.127	3,185	Brazília
31.162.16.203	3,157	Rusko
41.38.72.90	3,155	Egypt
46.167.107.60	3,155	Rusko
124.106.65.166	3,154	Filipíny
186.188.255.179	3,152	Panama

Obrázok 5 - Vultr experiment - zdrojové IP, Zdroj: vlastný

Obrázok 6 vyobrazuje rozdelenie útočníkov na základe ich typu resp. reputácie a **Obrázok 7** dotvára obraz o operačných systémoch, ktoré útočníci využili.

Typ útočníka	Count
known attacker	98,604
mass scanner	7,962
bot, crawler	403
anonymizer	118
tor exit node	67
form spammer	5

Obrázok 6 - Vultr experiment - typ útočníka, Zdroj: vlastný

Operačný systém	Počet
Linux 2.2.x-3.x	168,302
Windows 7 or 8	85,896
Linux 2.2.x-3.x (barebone)	49,085
Linux 3.11 and newer	12,853
Windows NT kernel	5,066
Mac OS X	3,524
Linux 2.2.x-3.x (no timestamps)	1,345
Linux 3.1-3.10	398
Windows NT kernel 5.x	368
Linux 2.4.x	209

Obrázok 7 - Vultr experiment - OS útočníka, Zdroj: vlastný

Obrázok 8 a **Obrázok 9** poskytujú prehľad o geolokácii útočníkov spolu s počtom útokov z danej geografickej oblasti, pričom **Obrázok 8** počty útokov ďalej rozdeľuje podľa portu, na ktorý sa útočilo.

Krajina	Port	Počet
Brazil	53	1,208,881
Brazil	445	9,600
Brazil	123	2,542
Brazil	80	131
Brazil	23	43
United States	123	69,257
United States	53	19,055
United States	80	439
United States	22	353
United States	8080	343
Belgium	1025	90
Belgium	445	70
Belgium	50100	46
Belgium	8008	41
Belgium	8000	39
Hong Kong	123	25,506
Hong Kong	53	7,503
Hong Kong	6379	120
Hong Kong	135	79
Hong Kong	9200	76
China	123	8,791
China	22	4,053
China	53	1,791
China	23	499
China	80	286

Obrázok 8 - Vultr experiment - krajina a port, Zdroj: vlastný

Krajina	Počet
Brazil	1,221,492
United States	109,447
Belgium	47,873
Hong Kong	34,654
China	32,932
Russia	20,372
Vietnam	17,774
Egypt	16,090
Japan	14,432
United Kingdom	10,228

Obrázok 9 - Vultr experiment - počet útokov podľa krajiny, Zdroj: vlastný

Obrázok 10 zobrazuje popis eventu zo software Suricata spolu s ID tohto eventu a počtom koľkokrát bol daný event zaznamenaný. Suricata je software spoločnosti Open Information Security Foundation, slúžiaci na analýzu sieťovej prevádzky a odhaľovanie hrozieb. [26]

ID	Popis	Počet
2210048	SURICATA STREAM reassembly sequence GAP -- missing packet(s)	530,682
2200036	SURICATA TCP option invalid length	61,147
2210051	SURICATA STREAM Packet with broken ack	4,692
2030387	ET EXPLOIT Possible CVE-2020-11899 Multicast out-of-bound read	3,924
2200074	SURICATA TCPv4 invalid checksum	2,865
2200094	SURICATA zero length padN option	2,688
2001978	ET POLICY SSH session in progress on Expected Port	2,045
2210037	SURICATA STREAM FIN recv but no session	1,994
2100486	GPL ICMP_INFO Destination Unreachable Communication with Destination Host is Administratively Prohibited	1,943
2029054	ET SCAN Zmap User-Agent (Inbound)	1,192

Obrázok 10 - Vultr experiment - Suricata ID, Zdroj: vlastný

Obrázok 11 je ďalším výstupom zo software Suricata, ktorý v tomto prípade popisuje jednotlivé zachytené CVE (Common Vulnerabilities and Exposures). Jedná sa teda o verejne známe a zdokumentované zraniteľnosti, ktoré sa útočníci snažili využiť.

CVE-2020-11899 je zraniteľnosť protokolu TCP/IP od spoločnosti Treck pred verziou 6.0.1.66, ktorá spôsobila, že protokol pri čítaní IPv6 paketov mohol vyjsť mimo povolených hraníc. [27]

CVE-2019-12263/61/60/55 sú 4 zraniteľnosti týkajúce sa software Wind River VxWorks, ktoré spôsobovalo pretečenie zásobníku TCP komponenty daného software. [28]

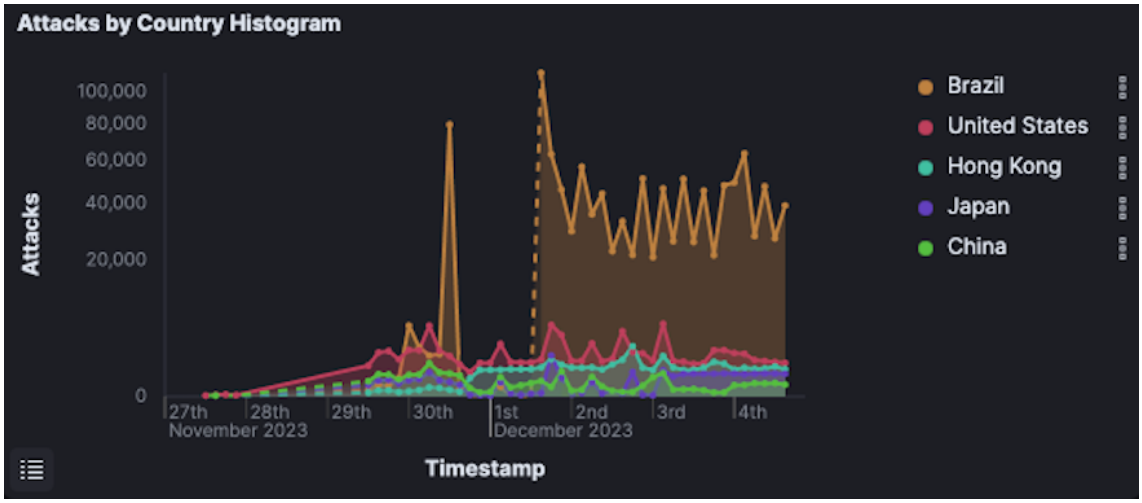
CVE-2020-11910 je obdoba **CVE-2020-11899** avšak v tomto prípade pre ICMPv4 pakety nie IPv6 pakety. [29]

CVE ID	Počet
CVE-2020-11899	3,924
CVE-2019-12263 CVE-2019-12261 CVE-2019-12260 CVE-2019-12255	8
CVE-2020-11910	2

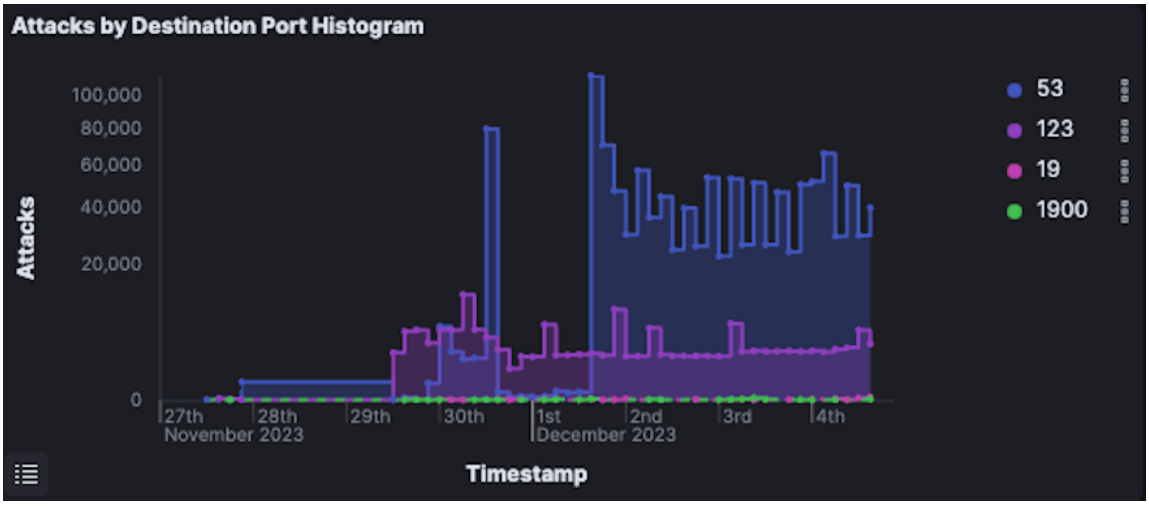
Obrázok 11 - Vultr experiment - Suricata CVE, Zdroj: vlastný

3.1.2 Ddospot

Ddospot zaznamenal najväčší počet útokov, pričom tieto útoky boli práve z Brazílie – viz. **Obrázok 12**. Toto poukazuje na koreláciu s obrázkami uvedenými vyššie. Na **Obrázok 4** vidíme veľký počet ASN z Brazílie. **Obrázok 8** potom jasne poukazuje na vysoký počet útokov z Brazílie najmä na porte 53 – čo predstavuje port, na ktorý bolo v prípade Ddospotu vedených najviac útokov (zobrazuje **Obrázok 13**). **Obrázok 9** nás v tejto hypotéze dodatočne utvrdzuje.



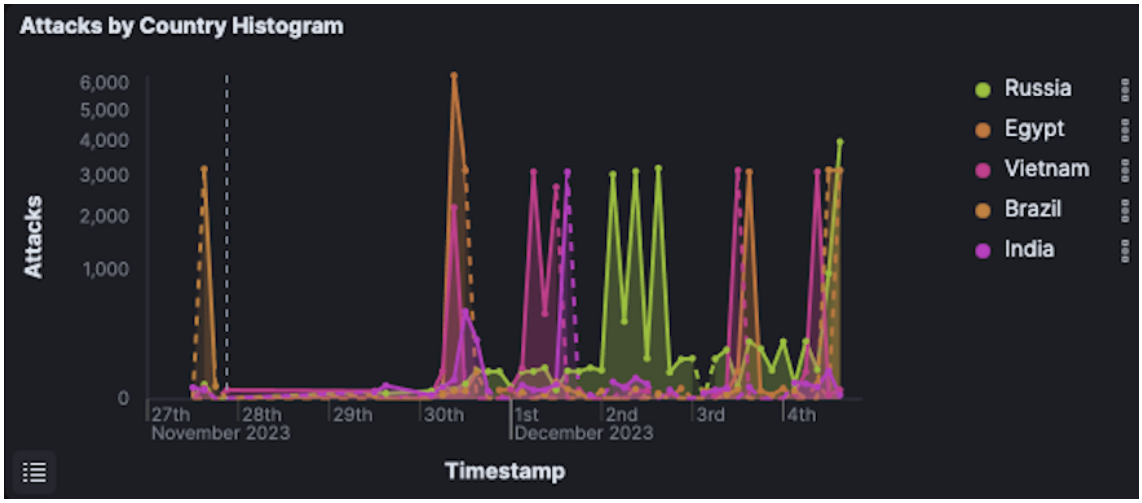
Obrázok 12 - Ddospot - zdrojová krajina, Zdroj: vlastný



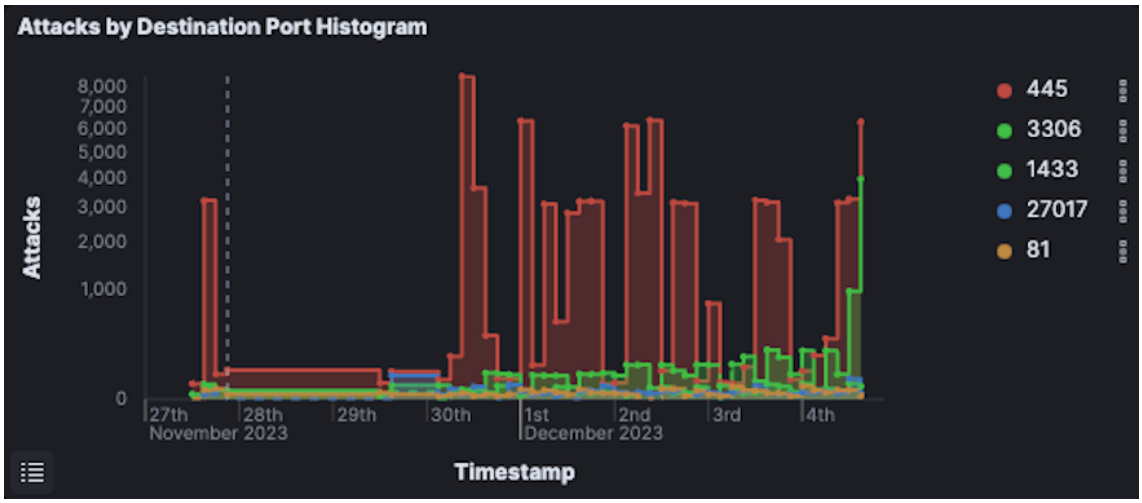
Obrázok 13 - Ddospot - cieľový port, Zdroj: vlastný

3.1.3 Dionaea

Dionaea honeypot zaznamenal druhý najväčší počet útokov vedených najmä z Ruska, Egypta, Vietnamu, Brazílie a Indie (zobrazuje **Obrázok 14**), aj napriek tomu, že Vietnam alebo India sa do zoznamu prvých desať najväčších útočníkov nedostali. V mnohom sa teda líšili aj cieľové porty na ktoré sa útočníci zameriavali (zobrazuje **Obrázok 15**).



Obrázok 14 - Dionaea - zdrojová krajina, Zdroj: vlastný



Obrázok 15 - Dionaea - cieľový port, Zdroj: vlastný

Vzhľadom k tomu, že Dionaea projekt obsahuje aj službu zbierajúcu zadané užívateľské mená a heslá, podarilo sa mu zachytiť aj tieto dáta, ktoré spolu s ich počtom zobrazujú **Obrázok 16** a **Obrázok 17**.

Zadané užívateľské meno	Počet	Zadané užívateľské meno	Počet
admin	112	SALE	111
ADMIN	111	Sale	111
Admin	111	TEST	111
BANKRUPTCY	111	Test	111
Bankruptcy	111	WAREHOUSE	111
COLLECTOR	111	Warehouse	111
Collector	111	bankruptcy	111
DECLINE	111	collector	111
DECLINES	111	decline	111
DELETE	111	declines	111
DIALER	111	delete	111
DUMP	111	dialer	111
Decline	111	dump	111
Declines	111	hold	111
Delete	111	house	111
Dialer	111	manager	111
Dump	111	newbiz	111
HOLD	111	sale	111
HOUSE	111	test	111
Hold	111	warehouse	111
House	111	root	111
MANAGER	111	sa	43
Manager	111	root	19
NEWBIZ	111	(empty)	9
Newbiz	111	anonymous	2

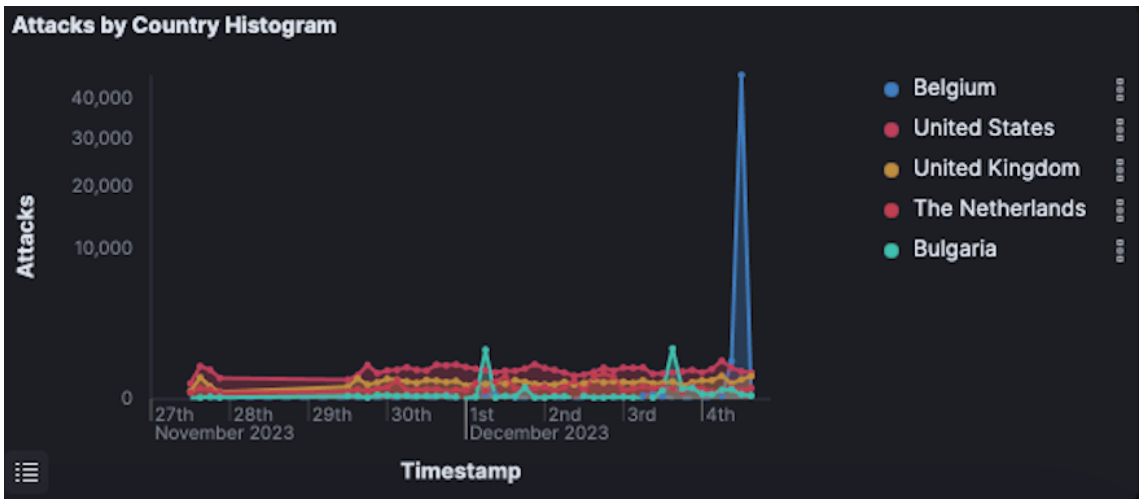
Obrázok 16 - Dionaea - zadané užívateľské mená, Zdroj: vlastný

Zadané heslo	Počet
(empty)	5,453
915	10
letmein	10
anonymous@	2
%system2016%	1
123456Aa!	1
Kq123	1
haian	1
monkey	1
trashh	1

Obrázok 17 - Dionaea - zadané heslá, Zdroj: vlastný

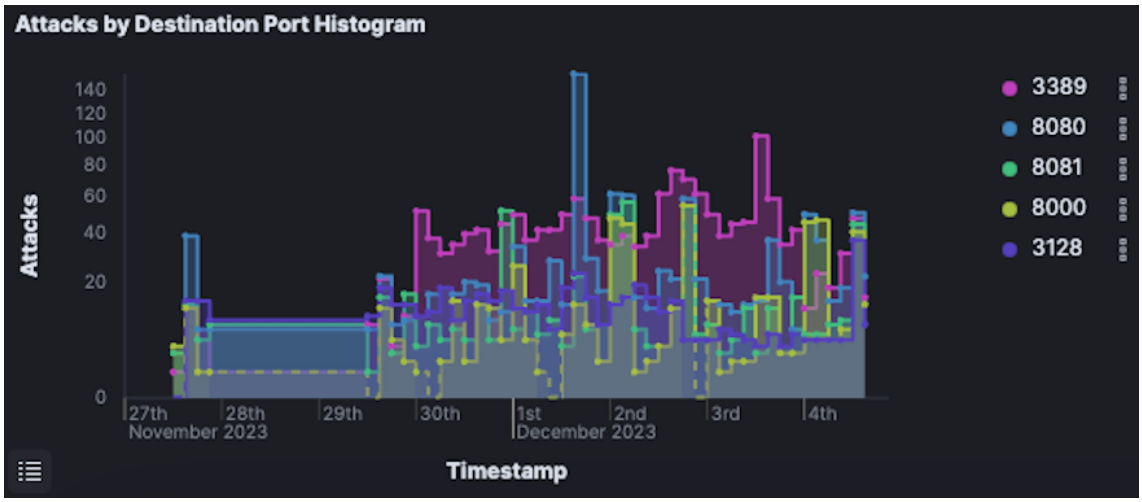
3.1.4 Honeytrap

Honeytrap je honeypotom, ktorý zaznamenal tretí najväčší počet útokov a to najmä z Belgicka, USA, Veľkej Británie, Holandska a Bulharska (zobrazuje **Obrázok 18**) – aj to aj napriek tomu, že niektoré z týchto krajín sa medzi prvých desať, v počte útokov, nedostali.

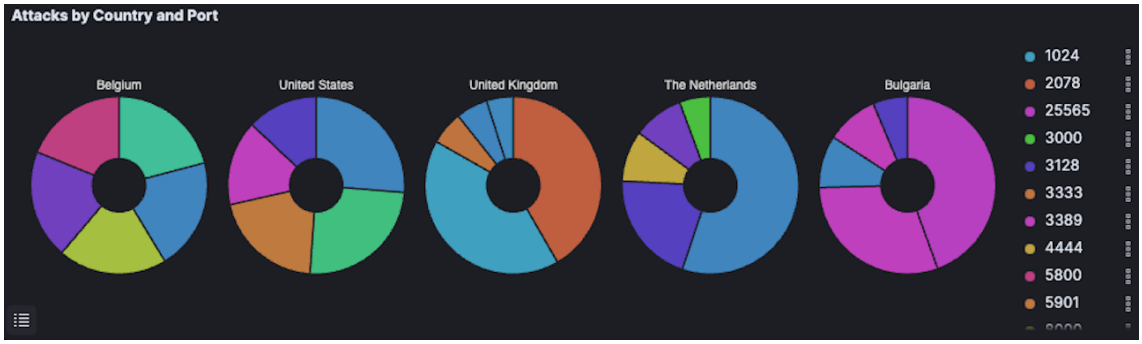


Obrázok 18 - Honeytrap - zdrojová krajina, Zdroj: vlastný

Porty na ktoré útočníci cielili najviac útokov, ako aj rozdelenie portov podľa krajín môžeme vidieť na **Obrázok 19** a **Obrázok 20.**



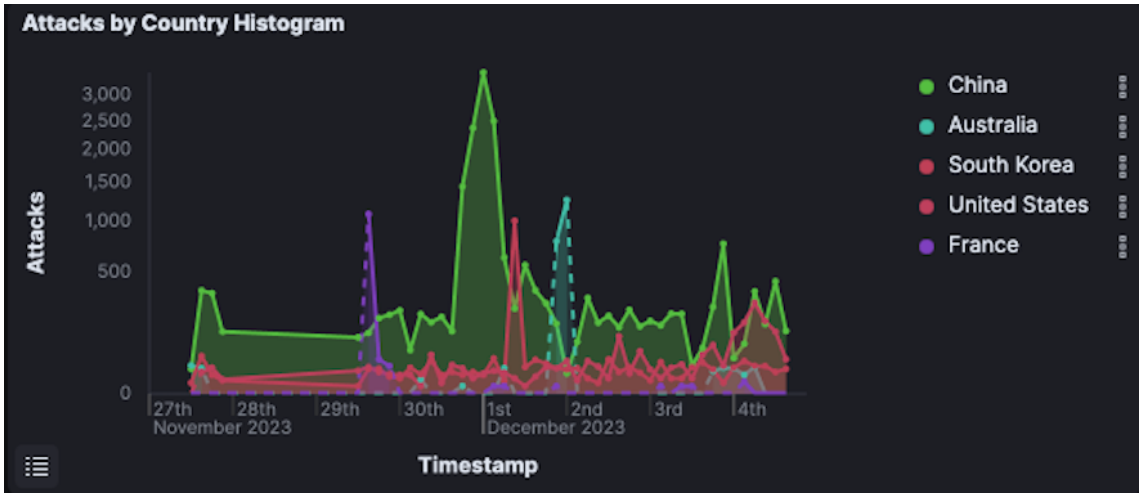
Obrázok 19 - Honeytrap - cieľový port, Zdroj: vlastný



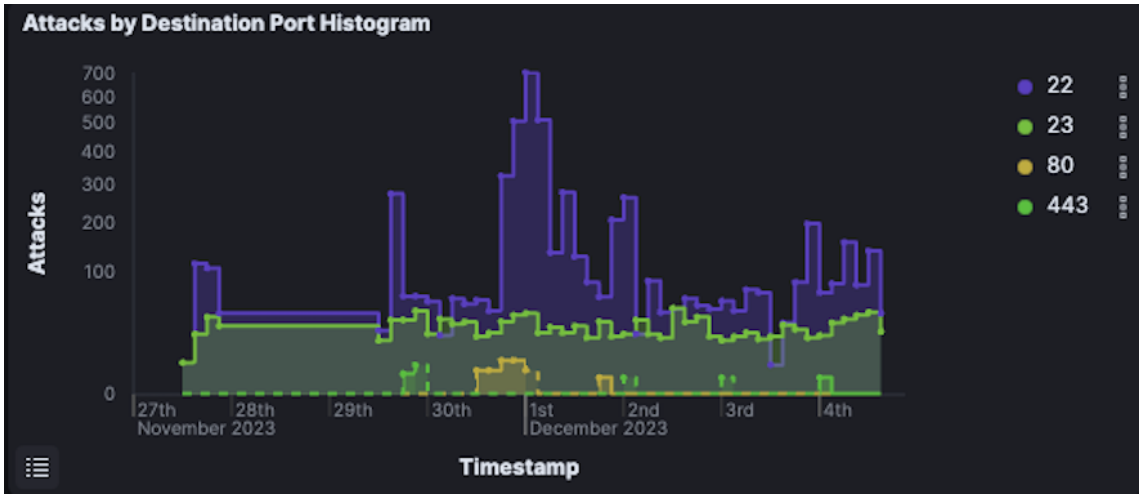
Obrázok 20 - Honeytrap - rozdelenie útokov medzi cieľové porty, Zdroj: vlastný

3.1.5 Cowrie

Štvrtý najväčší počet útokov zaznamenal Cowrie SSH a Telnet honeypot a to prevažne z Číny, Austrálie, Južnej Kórei, USA a Francúzska (zobrazuje **Obrázok 21**). Rozdelenie útokov podľa cieľových portov zobrazuje **Obrázok 22**.



Obrázok 21 - Cowrie - zdrojová krajina, Zdroj: vlastný



Obrázok 22 - Cowrie - cieľový port, Zdroj: vlastný

Keďže Cowrie predstavuje SSH a Telnet honeypot s funkcionalitou „falošného“ prihlásenia – teda nechá útočníka na náhodný pokus „uhádnuť“ správne meno a heslo, podarilo sa nám zozbierať aj najpoužívanejšie mená (zobrazuje **Obrázok 23**) a heslá (zobrazuje **Obrázok 24**), ktoré útočníci zadávali, ako aj najzadávanejšie príkazy po „úspešnom prihlásení“ (zobrazuje **Obrázok 25**).

Zadané užívateľské meno	Počet	Zadané užívateľské meno	Počet
root	2,807	mysql	11
admin	181	www	11
ubnt	53	git	10
user	50	dolphinscheduler	9
ubuntu	49	ftpuser	9
test	43	usr	9
guest	34	vagrant	9
oracle	34	centos	8
pi	33	demo	8
support	28	esuser	8
ftp	26	GET / HTTP/1.1	7
default	23	app	7
administrator	22	dbadmin	7
debian	21	ds	7
ansible	19	gitlab	7
john	18	mapr	7
nagios	17	nginx	7
postgres	17	sonar	7
hadoop	14	testuser	7
opc	13	tom	7
minima	12	tomcat	7
username	12	uftp	7
db2inst1	11	www-data	7
docker	11	Admin	6
es	11	elastic	6

Obrázok 23 - Cowrie - zadané používateľské mená, Zdroj: vlastný

Zadané heslo	Počet	Zadané heslo	Počet
123456	200	guest	13
admin	140	qwerty	13
root	85	Passw@rd	12
password	54	abc123	12
(empty)	52	kjashd123sadhj123dhs1SS	12
123	51	user	12
adminHW	47	1qaz2wsx	11
12345678	40	Test123	11
0	37	oracle	11
1234	35	test1234	11
1	34	ubuntu	11
12345	33	111111	10
ubnt	31	Admin1234	10
admin123	27	Test1234	10
eve	25	admin1234	10
default	23	db2inst1	10
raspberry	20	root123	10
test123	19	!@	9
pass	18	p@ssw0rd	9
1qaz@WSX	17	123456789	8
support	17	centos	8
Admin123	14	root1234	8
azerty	14	test	8
ansible	13	Root1234	7
changeme	13	1q2w3e4r	6

Obrázok 24 - Cowrie - zadané heslá, Zdroj: vlastný

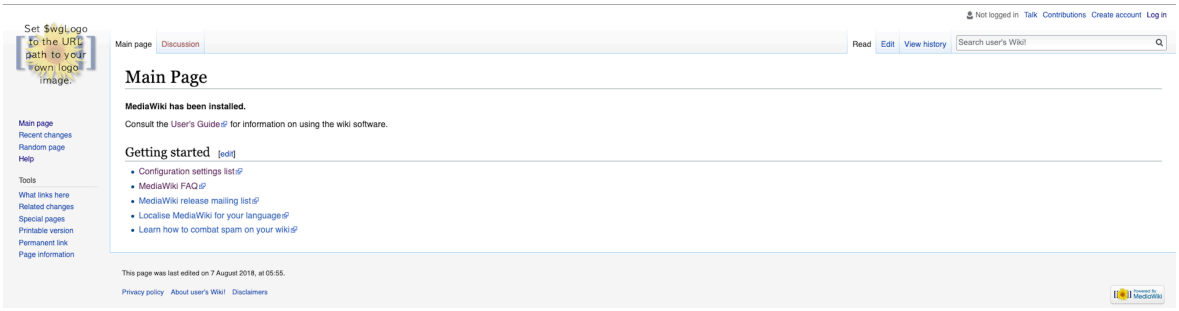
Príkaz	Počet
shell	96
system	88
while read i	71
enable	48
sh	48
uname -a	43
dd bs=52 count=1 if=.s || cat .s || while read i; do echo $i; done < .s	38
rm .s; exit	37
./oinasf	29
./oinasf; dd if=/proc/self/exe bs=22 count=1 || while read i; do echo $i; done < /proc/self/exe || cat /proc/self/exe;	29

Obrázok 25 - Cowrie - zadané príkazy, Zdroj: vlastný

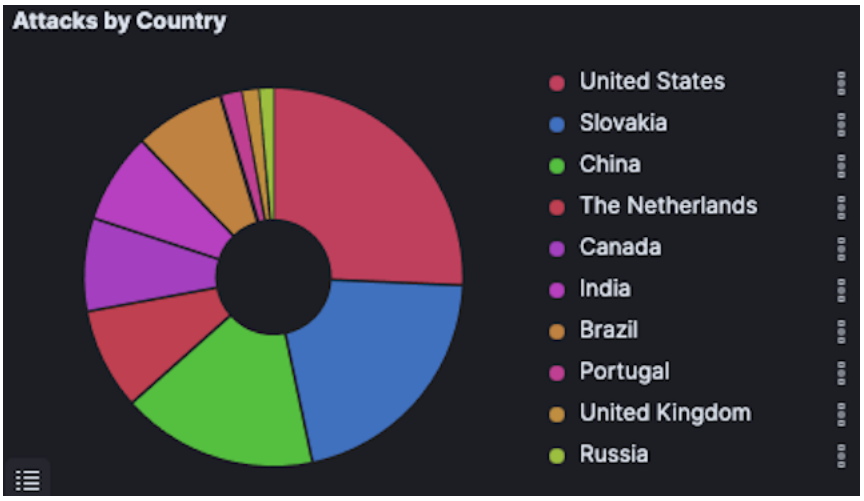
3.1.6 Tanner (Snare)

Tanner je webový honeypot, ktorý po každom štarte zobrazí falošnú webovú stránku z množiny predpripravených dizajnov stránok, ktoré má k dispozícií – jednu z nich zobrazuje **Obrázok 26**. V prípade Tanner honeypotu bol najväčší počet útokov zaznamenaný z USA, Číny, Holandska, Kanady a Indie (zobrazuje **Obrázok 27**) – Slovensko v tomto prípade opomíname keďže sa jednalo o našu vlastnú aktivitu. Rovnako

nebudeme uvádzať prehľad portov, keďže Tanner pracuje čisto na porte 80 a teda všetky útoky smerovali práve sem.



Obrázok 26 - Tanner - Webová stránka, Zdroj: vlastný

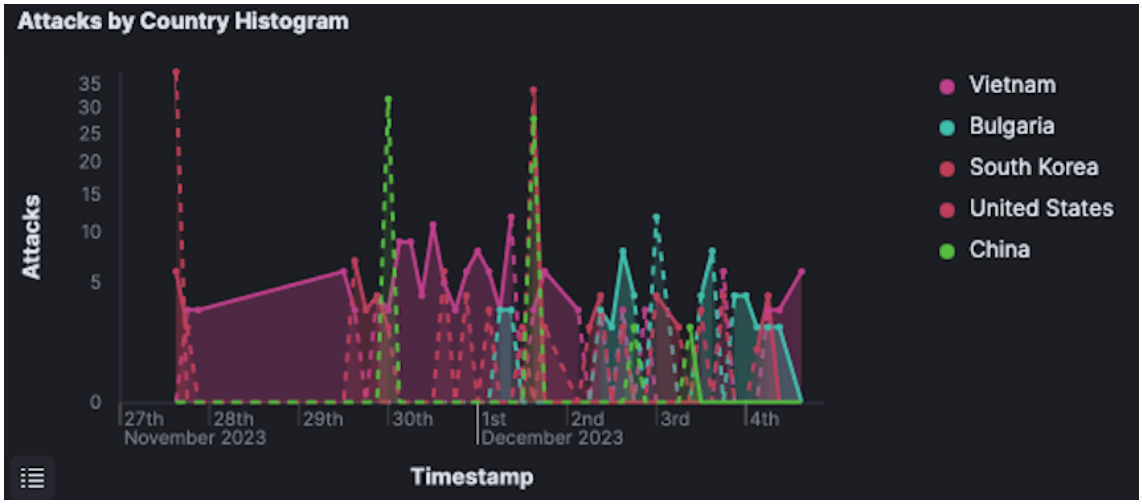


Obrázok 27 - Tanner - zdrojová krajina, Zdroj: vlastný

3.1.7 Adbhoney

Adbhoney je honeypot, simulujúci ADB (Android Debug Bridge) protokol, ktorého úlohou je udržiavať spojenie s virtuálnymi, ako aj reálnymi zariadeniami pripojenými k danému systému. Pre určený hlavne pre Android vývojárov, ktorým umožňuje na pripojené zariadenia odosielať príkazy, inštalovať software, sťahovať alebo ukladať dáta a pod. [30]

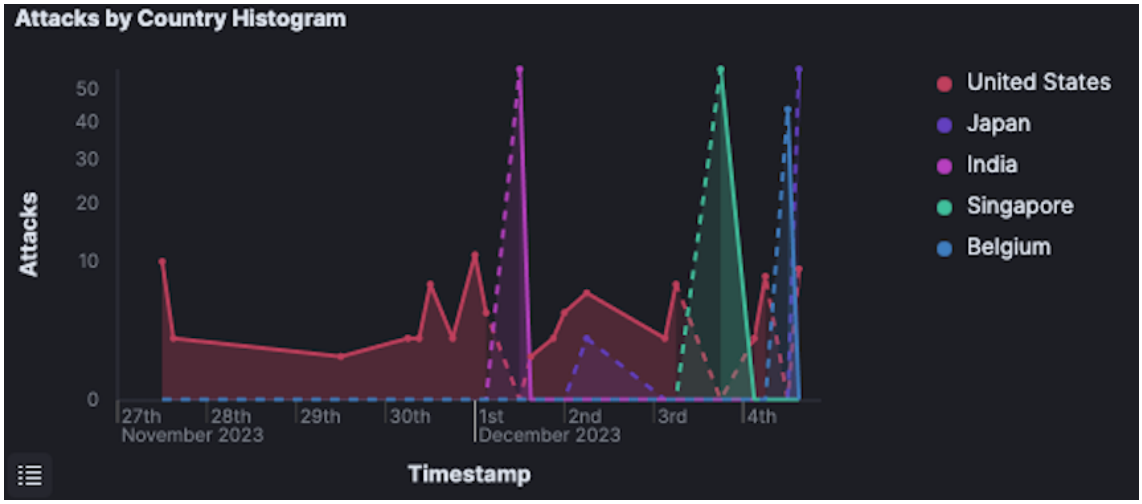
Adbhoney zaznamenal najväčší počet útokov z Vietnamu, Bulharska, Južnej Kórei, USA a Číny (zobrazuje **Obrázok 28**). Podobne ako v predchádzajúcom prípade honeypotu Tanner, Adbhoney pracuje len na jednom porte a to porte 5555 – všetky útoky preto pochopiteľne smerovali práve na tento port.



Obrázok 28 - Adbhoney - zdrojová krajina, Zdroj: vlastný

3.1.8 CiscoASA

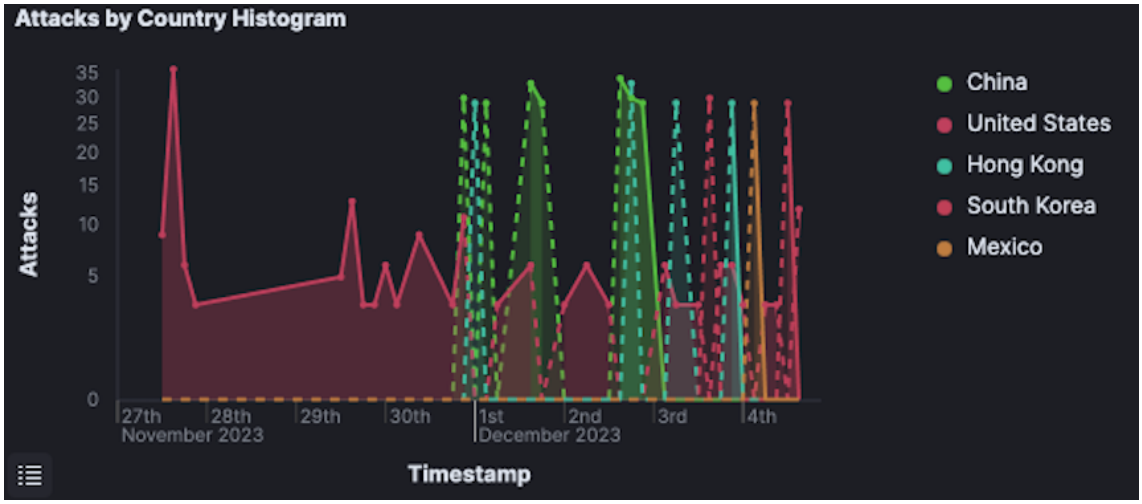
CiscoASA honeypot zaznamenal najväčší počet útokov z USA, Japonska, Indie, Singapuru a Belgicka. Ako môžeme vidieť na **Obrázok 29**, jednalo sa prevažne o ojedinelé, nárazové útoky s výnimkou USA, kde krivka poukazuje na viac stabilný a konštantný priebeh útokov.



Obrázok 29 - CiscoASA - zdrojová krajina, Zdroj: vlastný

3.1.9 Redishoneypot

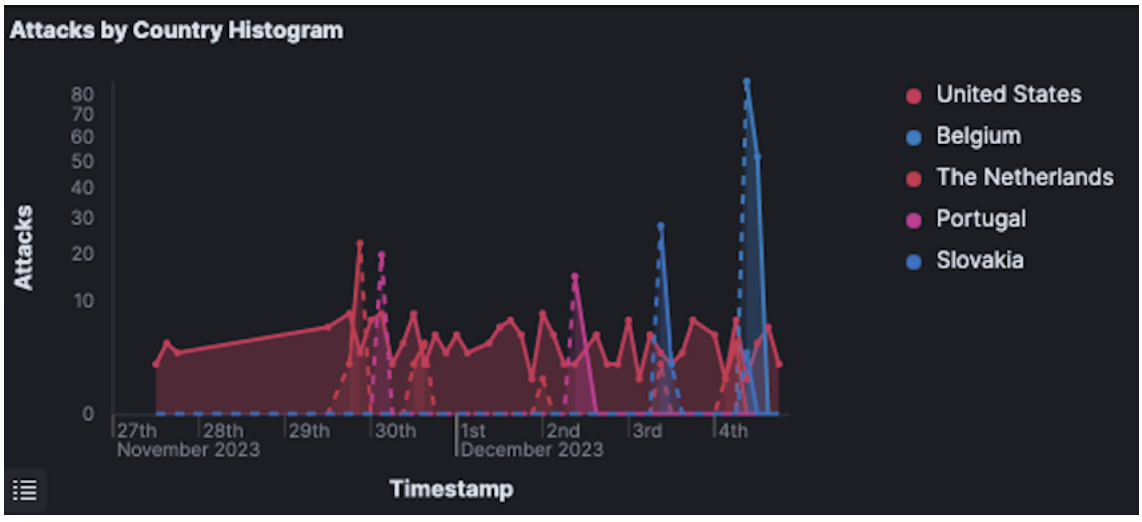
Redishoneypot fungujúci výhradne na porte 6379 zaznamenal ôsmy najväčší počet útokov a to najmä z Číny, USA, Hong-Kongu, Južnej Kórei a Mexika. Podobne ako v predchádzajúcom prípade môžeme na **Obrázok 30** vidieť, že sa v prevažnej väčšine jednalo o nárazové útoky s výnimkou USA, kde je krivka v priebehu času viac rovnomerná, teda aj počet útokov je rozdelený rovnomernejšie.



Obrázok 30 - Redishoneypot - zdrojová krajina, Zdroj: vlastný

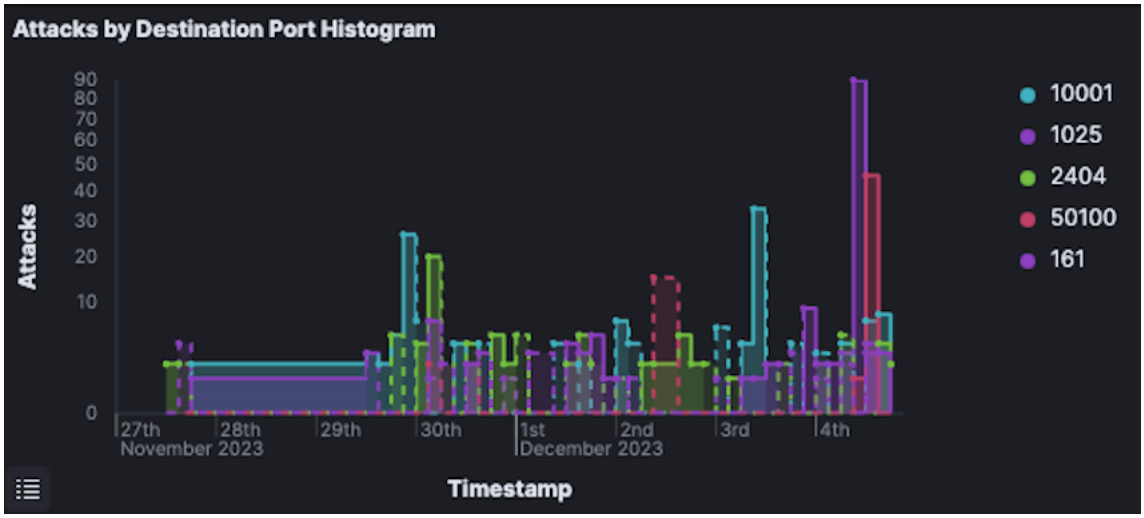
3.1.10 ConPot

ConPot je honeypot zameraný na veľmi špecifickú kategóriu industriálnych riadiacich systémov (z angl. *Industrial Control Systems*) aj napriek tomu zaznamenal deviaty najvyšší počet útokov, aj keď je nutné podotknúť, že v porovnaní napríklad s Ddospot honeypotom z **Podkapitoly 3.1.2** má výrazne menšie hodnoty. Avšak v týždňovom merítku nášho experimentu sa aj tak jednalo o hodnoty pozorovateľné, aj napriek tomu, že môžu byť mierne skreslené našim vlastným testovaním (Slovakia) – viz. **Obrázok 31**. Je ale viditeľné, že v našom prípade sa jednalo len o nárazové útoky, nie o konštantnú krivku.



Obrázok 31 - ConPot - zdrojová krajina, Zdroj: vlastný

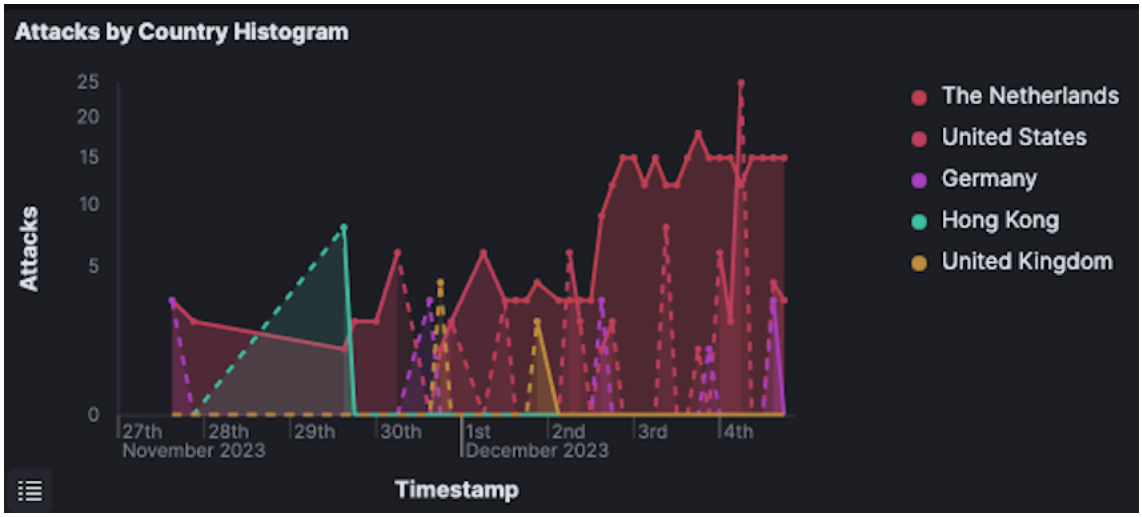
Rozdelenie útokov podľa portu môžeme vidieť nižšie na **Obrázok 32.**



Obrázok 32 - ConPot - cieľový port, Zdroj: vlastný

3.1.11 MailHoney

SMTP Honeypot MailHoney uzatvára desiatku honeypotov, na ktoré sa útočníci počas nášho pokusu zameriavali najviac. Jedná sa opäť o pomerne malé, ale nie zanedbateľné, množstvo útokov. Útoky na jediný port 25 prichádzali v tomto prípade najmä z Holandska, USA, Nemecka, Hong-Kongu a Veľkej Británie, ako môžeme vidieť na **Obrázok 33**.



Obrázok 33 - MailHoney - zdrojová krajina, Zdroj: vlastný

4 POUŽITÉ TECHNOLÓGIE

Skôr, než v praktickej časti prejdeme k všeobecnému popisu nami navrhovaného riešenia a neskôr k popisu samotnej implementácie projektu, jeho fungovaniu a testom, si v tejto kapitole v stručnosti definujeme technológie a iné zdroje, ktoré boli použité pri tvorbe práce. Opäť platí, že sa v žiadnom prípade nejedná o vyčerpávajúce definície, ale len o stručné priblíženie danej technológie či zdroja a jeho úlohy v našej práci.

4.1 VmWare Workstation PRO

Spoločnosť VmWare ponúka širokú škálu softwarových a cloudových riešení pre jednotlivcov, ako aj spoločnosti rôznej veľkosti. Pre našu prácu je podstatný ich software slúžiaci na virtualizáciu operačných systémov fungujúci na platforme Microsoft Windows. VmWare Workstation Pro je inštalovaný do zariadenia ako samostatný program, teda sa jedná o hypervisor typu 2 – viz. **Kapitola 1.9**. Pomocou tohto programu budeme v našej práci virtualizovať operačný systém Ubuntu 22.04.4 LTS. Takto docielime, že na jednom fyzickom zariadení môžeme naraz prevádzkovať niekoľko virtuálnych zariadení a simulovať tak našu infraštruktúru, ktorej návrh opíšeme v **Kapitole 5** nižšie. Zariadenia boli v testovacej infraštruktúre nastavené v tzv. „bridged“ móde, teda sa na sieti javili ako nezávislé zariadenia s vlastnou IP adresou. [31]

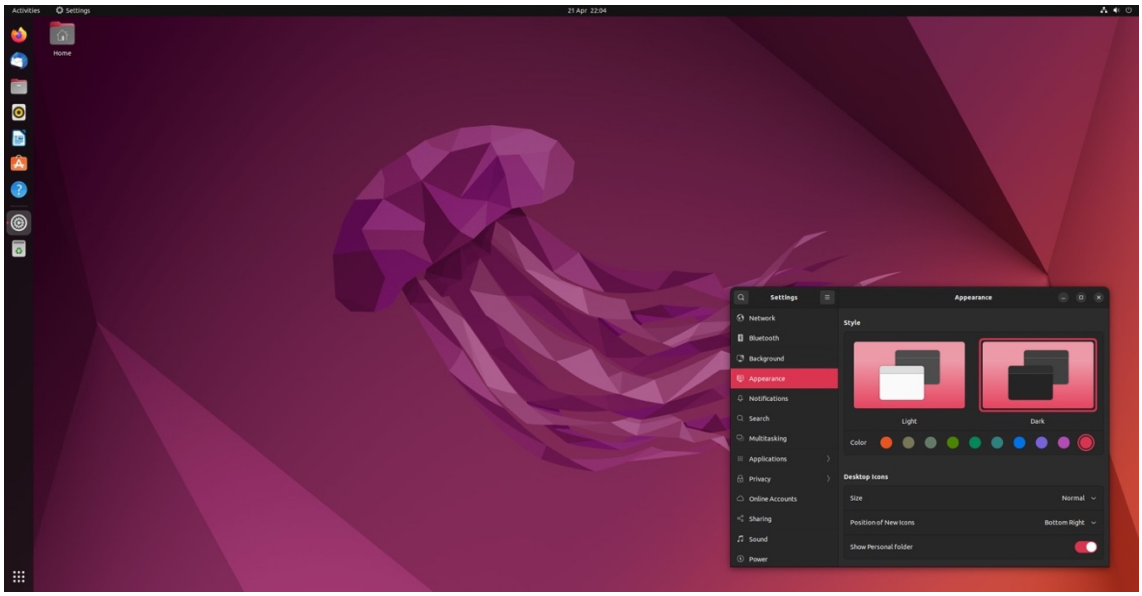
4.2 Linux

Linux patrí spolu s Windows a OS X (Mac OS) medzi najrozšírenejšie operačné systémy na svete. Zdieľa s nimi fakt, že sa jedná o plnohodnotný operačný systém, zahŕňajúci všetky komponenty od tzv. kernelu (jadra) až po grafické rozhranie, aplikácie atď. Rozdielom je, že sa jedná o open-source software. Na rozdiel od Windows alebo OS X ho teda svojim používateľom umožňuje používať zdarma (na akýkoľvek účel – teda aj komerčné využitie), študovať jeho zdrojový kód a upravovať ho podľa svojich potrieb, distribuovať jeho upravené, či neupravené kópie. Táto voľnosť, ktorú Linux poskytuje spôsobila, že vznikli tzv. linuxové distribúcie. Linuxová distribúcia je vlastne špecifická (upravená) verzia Linuxu, ktorá má často špecifické využitie a teda aj cieľovú skupinu používateľov. Medzi najznámejšie patria napr. Mint, Debian, Ubuntu, Solus, Fedora, Kali a mnohé iné. Väčšina distribúcií je dostupná zdarma – čo je prípad distribúcie, ktorú budeme používať my a ktorú si priblížme v **Podkapitole 4.2.1** nižšie, nájdeme však aj distribúcie platené. Rovnako ako

v prípade iných operačných systémov, nájdeme Linuxové distribúcie ako pre bežne používateľské zariadenia (tzv. desktopy), tak aj pre servery. [32]

4.2.1 Ubuntu

Ubuntu, od spoločnosti Canonical, patrí medzi jednu z najpopulárnejších Linuxových distribúcií na svete. Za toto vďačí hlavne svojej relatívnej jednoduchosti a užívateľskej prívetivosti, ako aj širokej škále ponúkaných služieb a aplikácií, aktívnemu vývojárskemu tímu a podpore. Ubuntu nájdeme ako v desktopovej, tak aj serverovej verzii. V našej práci budeme používať desktopovú verziu Ubuntu 22.04.4 LTS. [33]



Obrázok 34 - Ubuntu [33]

4.3 Skriptovacie jazyky

Skriptovacie jazyky sú vo svojej podstate programovacie jazyky, ktoré sú zväčša interpretované priamo za ich behu, teda si nevyžadujú kompiláciu – neplatí, ale že ju nemôžu aplikovať. V práci budeme používať najmä jazyk Bash, ktorý je natívnym skriptovacím jazykom pre Linux a jazyk Python, ktorý je v súčasnosti veľmi populárnym jazykom, vzhľadom k širokým možnostiam jeho použitia. Vzhľadom k tomu, že umožňuje ako skriptovanie, tak aj procedurálne či objektové programovanie. Problematika programovacích jazykov, ich prekladačov, typov, kompilácie a pod., nie je predmetom tejto práce a preto sa jej nebudeme širšie venovať. [34]

4.4 ELK Stack

ELK Stack (dnes tiež známy ako Elastic Stack) je kombináciou troch (prípadne štyroch, keďže sa často uvádza aj projekt Beats) populárnych open-source projektov od spoločnosti Elastic, slúžiacich na zber, agregáciu, ukladanie, prehľadávanie a vizualizáciu dát. Ako bolo spomenuté, jedná sa o veľmi populárny nástroj na management logov. Aj napriek tomu, že sa jedná o samostatné projekty, ktoré nutne nemusia byť využité spolu, ich popularita ako celku je dôkazom ich synergie. Výhodou je taktiež jednoduchosť ich vzájomnej integrácie a množstvo integrácií a plug-in-ov tretích strán. ELK teda predstavuje akronym pre tieto tri projekty – **E**lasticSearch, **L**ogstash a **K**ibana. V nasledujúcich podkapitolách bližšie definujeme načo každý z týchto projektov slúži, a teda aj jeho využitie v našej práci. Je nutné podotknúť, že bližšie technické špecifikácie (nastavenie, použité porty, nastavenie filtrov a pod.) budú rozobraté v ďalších častiach práce. Cieľom aktuálnej kapitoly, teda nie je podať presný popis implementácie ELK Stack-u v našej práci, ale čitateľa rýchlo oboznámiť so samotnou existenciou, všeobecnou funkcionalitou a účelom použitia ELK Stack-u. [35]

4.4.1 Logstash

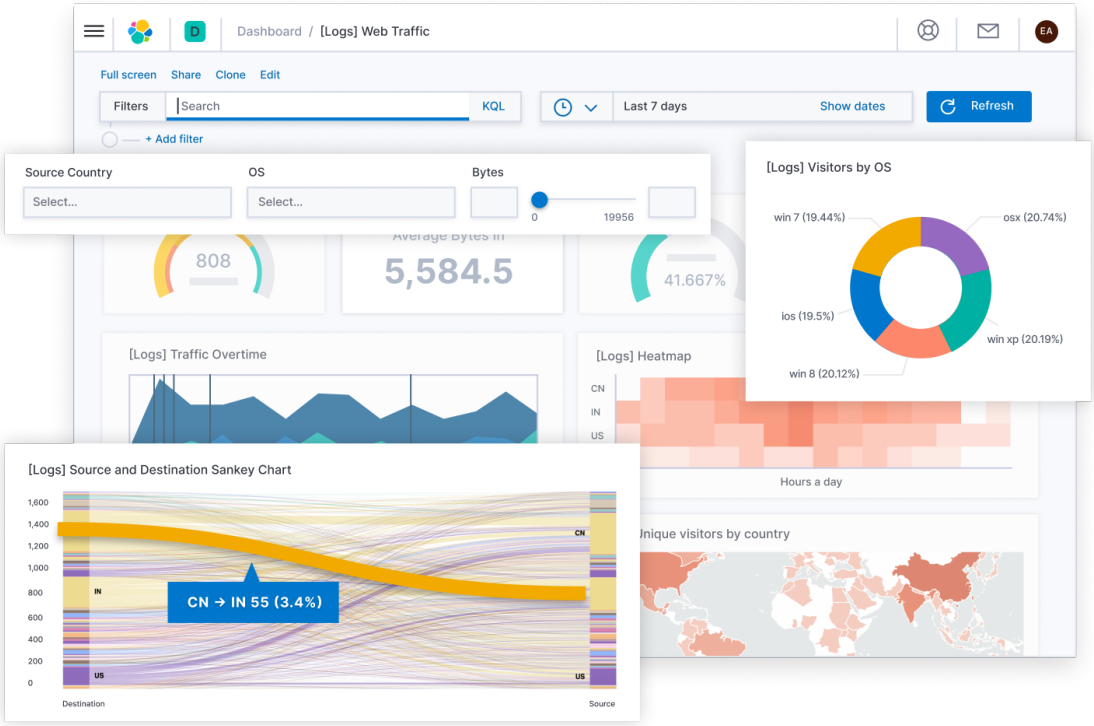
Aj napriek tomu, že Logstash je v ELK akronyme až druhý, začneme práve ním, vzhľadom k tomu, že sa jedná o nástroj slúžiaci na zber, agregáciu, konverziu a následný export dát (logov) – prevažne do nejakej inštancie ElasticSearch. Takto to platí aj v našom prípade, pričom okrem zberu logov samotným Logstash, sú mu logy doručené pomocou tzv. FileBeat protokolu – viz. **Podkapitola 4.4.4**. Logstash následne logy agreguje a upraví pomocou tzv. filtrov podľa potreby a exportuje logy do inštancie ElasticSearch. [35]

4.4.2 ElasticSearch

ElasticSearch je projekt slúžiaci na ukladanie a prehľadávanie dát. Jedná sa teda o vyhľadávací a analytický nástroj (databázový systém) založený na formáte JSON. [35]

4.4.3 Kibana

Kibana poskytuje užívateľské rozhranie, slúžiace na prehľadávanie, analýzu a vizualizáciu dát uložených v ElasticSearch, na ktorý je daná inštancia Kibana viazaná. Kibana teda umožňuje okrem „klasického“ prehľadávania ElasticSearch databázy aj vytváranie grafov, histogramov, máp a pod., v rôznej úrovni komplexnosti – viz. **Obrázok 35**. [35]



Obrázok 35 - Kibana - príklad rozhrania [35]

4.4.4 FileBeat

FileBeat je jeden z protokolov projektu Beats slúžiaci na monitorovanie, zber a prepravu logov – zvyčajne priamo do inštancie ElasticSearch, prípadne inštancie Logstash, pokiaľ si logy vyžadujú ďalšie spracovanie. [35]

4.5 NGINX

NGINX je proxy server, reverse-proxy server, mail proxy a webový server - pričom ponúka schopnosť rovnomerne rozložiť zaťaženie medzi viacero serverov, pôvodne vyvinutý Igorom Sysoevom. Ponúka teda pomerne širokú škálu funkcionalít, ktoré možno využívať samostatne, v nejakej kombinácií, či naraz. V našej práci budeme NGINX používať ako reverse-proxy pre našu inštanciu Kibana (viz. **Podkapitola 4.4.3**). [36]

4.5.1 Reverse-proxy

Povedali sme, že NGINX budeme využívať ako reverse-proxy, otázkou teda zostáva, čo to tá reverse-proxy vlastne je. Reverse-proxy je server, ktorý sa nachádza pred webovým (zdrojovým) serverom, na ktorý sa klient dotazuje a sprostredkováva komunikáciu medzi daným klientom a daným, zdrojovým serverom. Zabezpečuje, že žiadny klient

a zdrojový server spolu nekomunikujú priamo. Medzi hlavné výhody využitia reverse-proxy patrí [37]:

- *Rovnomerné rozloženie záťaže* (z angl. *„load balancing“*) – umožňuje rozložiť požiadavky klientov medzi viacero zdrojových serverov, tak aby jeden zdrojový server nebol preťažený a iný/iné boli nevyužité. V dnešnej dobe už nie je výnimkou, že veľké webové stránky, resp. ich servery obslúžia milióny klientov denne. Smerovať všetky tieto požiadavky na jeden server by bolo nemysliteľné a preto je takmer nutné zapojiť do infraštruktúry viacero zdrojových serverov a rovnomerne medzi ne rozkladať zaťaženie. Správna implementácia tejto techniky je o to dôležitejšia v prípade, že sa jedná o medzinárodnú spoločnosť so zdrojovými servermi rozmiestnenými na viacerých miestach po svete, kedy zvyčajne chceme smerovať požiadavky na (aktuálne najmenej vyťažený) geograficky najbližší zdrojový server.
- *Obrana proti útokom* – vzhľadom k tomu, že klient nikdy nekomunikuje priamo s daným serverom, nepozná ani jeho IP adresu a teda je daný zdrojový server lepšie chránený pred rôznymi útokmi, ako napr. DDOS. Útočník má možnosť zaútočiť len na samotnú reverse-proxy, ktorá býva na útoky lepšie pripravená.
- *Caching* – caching obsahu je technika, ktorú napr. prehliadače využívajú na dennom poriadku. V princípe sa jedná o uloženie dát, ktoré sme už raz načítali do pamäte, pre prípad, že ich budeme v relatívne krátkom čase potrebovať znova. Reverse-proxy servery sú teda schopné uložiť dáta, ktoré klienti od daného serveru požadujú a v prípade, že ďalší klient bude požadovať rovnaké dáta, je mu ich schopný poskytnúť samotný reverse-proxy server, bez nutnosti zaťažovať zdrojový server.
- *Šifrovanie* – reverse-proxy umožňuje šifrovať komunikáciu pomocou SSL (alebo TLS) priamo u seba, bez nutnosti šifrovania a dešifrovania na zdrojovom serveri, čím opäť umožňuje znižovať jeho zaťaženie.
- *Správa prístupu* – reverse-proxy taktiež umožňuje limitovať prístup k danému serveru len z vopred definovaných IP adries a/alebo s pomocou vopred definovaných prihlasovacích údajov – čo je aj náš prípad, resp. využitie (NGINX) reverse-proxy v našej práci.

4.6 WireGuard

„*WireGuard je extrémne rýchla, ale aj napriek tomu jednoduchá a moderná VPN, využívajúca najmodernejšiu kryptografiu.*“ [38]

Práve takto je WireGuard označený na oficiálnej stránke tohto projektu. Jedná sa teda o open-source VPN projekt, ktorého snahou je byť rýchlejší, jednoduchší, výkonnejší, chudší (rozumej menej riadkov kódu, ktoré sú potrebné na jeho fungovanie) a aj napriek tomu poskytovať rovnakú, ak nie lepšiu, funkcionalitu a bezpečnosť, než ponúkajú aktuálne najrozšírenejší konkurenti ako OpenVPN, či IPsec. V počiatkoch projektu bol mierený čisto na operačný systém Linux, dnes ho už nájdeme aj na operačnom systéme Microsoft Windows, OS X, BSC, iOS a Android. Jeho implementácie môžeme nájsť ako na desktopových, tak aj na serverových verziách týchto operačných systémov, pričom jeho vývoj je stále aktívny. Hlavnými benefitmi WireGuard projektu, pomocou ktorých sa snaží docieliť vyššie uvedené ciele sú [38]:

- *Jednoduchosť (použitia)* – WireGuard projekt sa snaží docieliť konfiguračnú a funkčnú jednoduchosť, pomocou konfigurácie a nasadeniu podobnému SSH. VPN pripojenie vzniká po jednoduchej výmene verejných kľúčov a o zvyšok sa transparentne postará priamo WireGuard. Odpadá teda nutnosť administrátorov starať sa o pripojenia a ich stav, či procesy bežiace na pozadí tzv. „*daemons*“.
- *Vysoká kryptografická úroveň* – WireGuard používa na svoje šifrovanie kryptografmi overené a často postkvantové šifrovacie protokoly, ako napríklad NPF (Noise Protokol Framework), Curve25519, ChaCha20, Poly1305, BLAKE2, SipHash24, HKDF.
- *Minimalizovanie možností útoku* – v porovnaní s obrovskou dĺžkou kódu iných projektov v oblasti VNP sa WireGuard snaží ísť cestou minimalizmu – čím menej, tým lepšie. Vďaka tomu sa minimalizuje množstvo potencionálnych rizík, ktoré je možné do kódu zaniesť a je jednoduchšie daný kód spravovať a testovať na prípadné zraniteľnosti.
- *Vysoká výkonnosť* – kombináciou extrémne rýchlych kryptografický postupov a priamej integrácie do Linuxového jadra WireGuard docielil vysokú výkonnosť, rýchlosť s minimalizáciou potrebných zdrojov. Vďaka tomu je vhodný aj do vstavaných, či miniatúrnych zariadení s obmedzeným výkonom.

- *Otvorený a pravidelne revidovaný* – vďaka tomu, že WireGuard projekt svoj zdrojový kód publikuje voľne, je možné ho podrobiť dobrovoľnej revízii zo strany obrovského množstva nadšencov, ako aj cielenej revízii zo strany akademických pracovníkov, či výskumníkov v oblasti kybernetickej bezpečnosti.

V našej práci WireGuard zabezpečuje bezpečný prenos zaznamenaných logov pomocou FileBeat z honeypotov do Logstash inštancie, kde sú ďalej spracovávané, ukladané a vizualizované.

4.7 Apache HTTP (Web) Server

Apache HTTP (Web) Server projekt je produktom snahy Apache Software Foundation a množstva dobrovoľníkov, ktorý do tohto projektu prispeli svojimi nápadmi, kódom alebo jeho úpravami, či tvorbou dokumentácie. Apache HTTP Server je teda voľne dostupný, open-source projekt, implementujúci – ako názov napovedá, webový server. V našej práci je Apache HTTP Server využívaný na hosting webového honeypotu. [39]

4.8 GeoLite2

GeoLite2 sú databázy slúžiace na geolokáciu na základe verejnej IP adresy. Jedná sa o voľne dostupnú verziu GeoIP2 databáz, pričom obe poskytuje spoločnosť MaxMind. MaxMind ako rozdiel medzi voľne dostupnou a platenou verziou udáva nižšiu presnosť GeoLite2 (v porovnaní s GeoIP2) súradníc, ktoré pri geolokácii GeoLite2 poskytne a vyššie riziko nesprávnej lokalizácie. GeoLite2 databázy sú aktualizované dvakrát týždenne, a to v utorok a piatok. GeoLite2 databázy je možné po stiahnutí využívať lokálne na danom zariadení, po vytvorení a zadaní (spárovaní) s licenčným kľúčom – čo je aj náš prípad. GeoLite2 databázy umožňujú vyhľadávanie na úrovni kontinentov, štátov, vyšších územných celkov, miest a následne súradníc. [40]

Teda pre fiktívny príklad IP adresy z oblasti Bratislavského hradu, by sme dostali niečo takéto: Europe, Slovakia, Bratislava Region, Bratislava, 48.14225894287251, 17.1001811558056. V tejto práci GeoLite2 využívame na približnú geolokáciu útočníkov na základe ich IP adries.

4.9 PostFix

V roku 1998 napísal Wietse Venema open-source mailový server PostFix, určený výhradne pre Unixové operačné systémy (v našom prípade Linux - Ubuntu), pod záštitou IBM

Research. PostFix sa odvtedy rozšíril medzi širšiu komunitu používateľov a je používaný a vyvíjaný aj dnes. [41]

V našom projekte PostFix využívame na odosielanie mailových notifikácií, pri novo vytvorenom pripojení útočníka na náš PyRDP honeypot (viz. **Kapitola 8**).

4.10 T-Pot

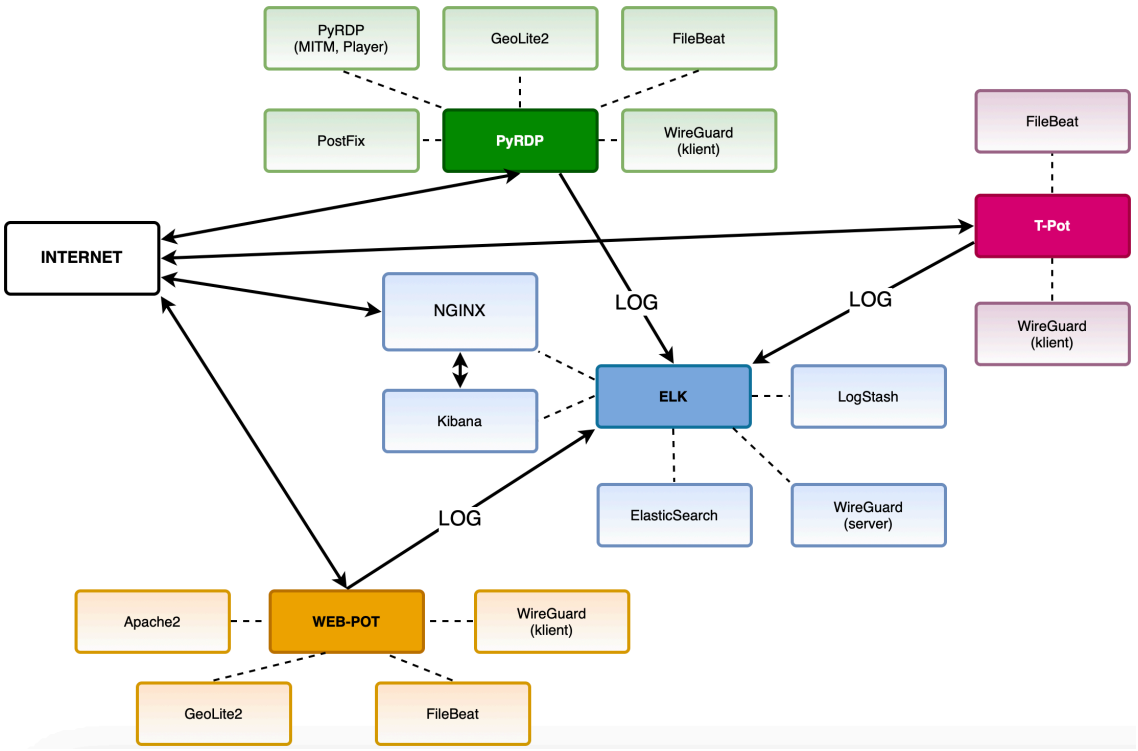
T-Pot je open-source honeypot projekt spoločnosti Deutsche Telekom Security GmbH. Samotná spoločnosť ho označuje ako all-in-one, distribuovanú a multi-architekturálnu (amd64, arm64) honeypot platformu, ktorá v sebe aktuálne, okrem iného, skrýva 22 honeypotov a vizualizácie pomocou ELK Stack inštancie - realizované pomocou docker kontajnerov. Niektoré z honeypotov sme uvideli už v **Kapitole 3**, kde sme tento projekt nasadili v cloudovom prostredí, na približne týždeň a sledovali záujem útočníkov. Bližšie si projekt po technickej stránke, spolu s našimi úpravami priblížime v **Kapitole 9**.

5 NAVRHOVANÉ RIEŠENIE

Predtým, než v tejto kapitole prejdeme na popis navrhovaného riešenia a v ďalších kapitolách na samotný popis implementácie je nutné podotknúť, že nebudeme popisovať jednotlivé kroky, resp. príkazy, pomocou ktorých sme k danej finálnej implementácii, či konfigurácii dospeli. Komponenty systému boli inštalované pomocou oficiálnej dokumentácie daného projektu a balíčkovacieho nástroja „apt“ operačného systému Ubuntu.

Navrhované riešenie zobrazuje **Obrázok 36** nižšie, ktorý zobrazuje návrh infraštruktúry, ako aj samotné VS (viz. **Podkapitola 1.9**) a súčasti týchto VS (označené prerušovanou čiarou), ktoré sú pre prácu relevantné. Ako teda môžeme vidieť infraštruktúru tvoria štyri hlavné súčasti (VS), ich vzájomné väzby a komponenty relevantné pre prácu. Granulárnejší opis každého VS (resp. jeho funkcionality), jeho komponent spolu s ich konfiguráciami a spôsobmi implementácie, či skriptami, ktoré boli pre daný VS vytvorené, poskytneme v nasledujúcich **Kapitolách 6**, **7**, **8**, a **9**. Každá z týchto kapitol sa venuje jednému z týchto VS. Aktuálna kapitola sa snaží poskytnúť čitateľovi prehľad o úlohe daného VS v navrhovanej infraštruktúre, jeho všeobecnej funkcionalite a väzbe s ostatnými systémami v rámci navrhovanej infraštruktúry.

Na úvod poznamenáme, že všetky VS, okrem T-Pot, sú založené na operačnom systéme Ubuntu (viz. **Podkapitola 4.2.1**) verzia 22.04.4 LTS. Plné čiary predstavujú sieťovú komunikáciu medzi VS alebo VS a internetom. Prerušované čiary predstavujú súčasti daného virtualizovaného operačného systému.



Obrázok 36 - Navrhované infraštruktúrne riešenie, Zdroj: vlastný

5.1 ELK

V „strede“ nami navrhovanej infraštruktúry (označené modrou farbou) sa nachádza VS pomenovaný **ELK** (viz. **Kapitola 6**), ktorý obsahuje lokálnu inštaláciu ELK stack (viz. **Podkapitola 4.4**), slúžiacu na zber, agregáciu, úpravu, ukladanie, prehľadávanie a vizualizáciu dát pomocou dashboardov. Ďalej tu nájdeme lokálnu inštaláciu WireGuard (viz. **Podkapitola 4.6**), ktorý plní úlohu servera, teda sa k nemu klienti pripájajú pomocou zabezpečeného VPN tunela a predom nakonfigurovaných kľúčových párov a rozhraní. Toto VPN pripojenie slúži na zabezpečenie prenosu logov získaných počas činnosti honeypotov, od honeypotov k ELK inštalácii na danom zariadení – ostatná komunikácia teda cez tento tunel neputuje. Poslednou, ale nie menej významnou komponentnou, je lokálna inštalácia NGINX reverse-proxy (viz. **Podkapitola 4.5**), ktorá nám slúži na zabezpečenie pripojenia ku Kibana webovému rozhraniu, pomocou predom definovaných prihlasovacích údajov.

5.2 PyRDP

Nad VS **ELK** sa nachádza VS pomenovaný **PyRDP** (označený zelenou, viz. **Kapitola 8**), ktorý obsahuje upravenú verziu projektu PyRDP od spoločnosti GoSecure (viz. [42]), spolu s vlastnými skriptami slúžiacimi na monitoring priečinka, kde PyRDP projekt ukladá

záznamy z útokov, tvorbu, parsovanie a ukladanie logov, či odosielanie notifikácií o novom pripojení pomocou lokálnej inštalácie PostFix (viz. **4.9**). PyRDP poskytuje MITM funkcionalitu, čiže sa MITM server pripojí na predom definované zariadenie, pričom sa na sieti prezentuje ako toto zariadenie. Útočník sa potom pomocou RDP protokolu pripojí práve na náš MITM server, v domnienke, že sa pripája na úplne iné zariadenie. PyRDP mu sprostredkuje pripojenie na dané zariadenie pomocou RDP, pričom zaznamenáva celý priebeh pripojenia, ktoré vieme dokonca sledovať live, prípadne si ho spätne prehrať pomocou PyRDP Player funkcionality. Ďalšou zaujímavou funkcionalitou PyRDP je spustenie v tzv. „crawl“ móde. Tento mód začne automaticky sťahovať obsah akéhokoľvek úložiska, ktoré útočník k vzdialenému zariadeniu, resp. k nášmu MITM serveru pripojí. Ďalej na tomto virtuálnom systéme nájdeme lokálne GeoLite2 databázy (viz. **Podkapitola 4.8**), slúžiace na geolokáciu útočníka podľa jeho IP adresy, lokálnu inštaláciu FileBeat protokolu slúžiacu na monitoring logov a ich odosielanie (označené hrubou čiarou s nápisom „LOG“ smerujúcou od PyRDP k ELK) do Logstash inštancie, nachádzajúcej sa na virtuálnom systéme **ELK** – z predchádzajúceho odseku - cez zabezpečený WireGuard VPN tunel, vytvorený pomocou lokálnej inštalácie WireGuard v podobe klienta.

5.3 WEB-POT

Pod VS **ELK** vidíme VS pomenovaný ako **WEB-POT** (označený oranžovou farbou, viz. **Kapitola 7**), ktorý obsahuje lokálnu inštaláciu Apache2 HTTP servera (viz. **Podkapitola 4.7**), na ktorom je umiestnená mierne upravená kópia zdrojového kódu Moodle portálu UTB a vlastný skript. Táto upravená kópia Moodle portálu sa vydáva za portál slúžiaci na administráciu Moodle. Zobrazuje teda prihlasovací formulár pre administrátorov Moodle a skript, ktorý simuluje overenie zadaných prihlasovacích údajov. Okrem toho skript zadané prihlasovacie údaje, spolu s IP adresou útočníka, jeho použitým webovým prehliadačom a ďalšími informáciami uloží do logov, ktoré sú monitorované a odosielané lokálnou inštaláciou protokolu FileBeat cez zabezpečený WireGuard tunel do virtuálneho systému ELK, na ďalšie spracovanie a vizualizáciu (označené hrubou čiarou s nápisom „LOG“ smerujúcou od WEB-POT k ELK). WireGuard je v tomto prípade, rovnako ako pri PyRDP virtuálnom systéme, v pozícii klienta.

5.4 T-Pot

Naľavo od virtuálneho stroja **ELK** sa nachádza VS pomenovaný **T-Pot** (označený ružovou farbou, viz. **Kapitola 9**). T-Pot VS predstavuje plnú (všetky honeypoty a ďalšie súčasti) inštanciu T-Pot projektu (viz. **Podkapitola 4.10** alebo adresa zdroja [24]). Aj napriek tomu, že projekt samotný obsahuje ELK Stack realizovaný pomocou Docker kontajnerov, v našej práci ho nebudem využívať a bude teda deaktivovaný. Namiesto toho budeme zber a odosielanie logov zabezpečovať pomocou dodatočnej lokálnej inštalácie FileBeat protokolu, ktorý bude dáta odosielať pomocou zabezpečeného VPN WireGuard tunela do inštancie Logstash na virtuálnom systéme ELK (označené hrubou čiarou s nápisom „LOG“ smerujúcou od T-Pot k ELK). Takto docielime jednotnosť v našej infraštruktúre, ako aj fakt, že zber logov, ich agregácia, úprava, ukladanie, prehľadávanie a vizualizácia bude prebiehať na jednom mieste. Logy teda budeme môcť prehľadávať na jednom centrálnom mieste, bez nutnosti „preskakovať“ cez viacero systémov.

6 ELK

Úlohu, ktorú v našej práci VS ELK zohráva, sme v stručnosti uvideli v **Podkapitole 5.1**. V tejto kapitole na tieto informácie naviažeme a detailnejšie rozoberieme funkcionalitu, komponenty a úpravy, ktoré sme v tomto virtuálnom systéme vykonali. Rovnako, ako vo všetkých ostatných častiach tejto práce, sa budeme zaoberať len súčasťami, ktoré sú pre prácu relevantné alebo ktoré sme do systému pridali, či nejako upravili pre potreby práce, navrhovaného riešenia, či cieľovej infraštruktúry. Nebudeme sa teda sústrediť na všetky komponenty, resp. funkcie, ktoré sú v dnešných operačných systémoch štandardom.

6.1 Implementácia

VS ELK sa nachádza v pomyselnom strede nami navrhovanej infraštruktúry a zodpovedá za niekoľko kľúčových úloh, ktoré korešpondujú s jeho komponentami, preto považujeme za rozumné sa naň pozrieť ako na prvý – keďže ostatné časti (systémy, honeypoty) sú na ňom, v istom slova zmysle, závislé. Medzi najpodstatnejšie komponenty daného systému, tak ako ich zobrazuje **Obrázok 36,** patria:

- ***ELK Stack*** – VS ELK obsahuje lokálnu inštaláciu ELK Stack, čo je čitateľovi v tomto bode pravdepodobne zrejmé – podľa názvu. Táto inštalácia pozostáva z troch, už definovaných, komponent (viz. **Podkapitoly 4.4.1** až **4.4.4**), ktoré plnia svoje špecifické úlohy. Vo všeobecnosti teda zodpovedá za zber, agregáciu, úpravu, ukladanie, prehľadávanie a vizualizáciu logov pomocou dashboardov z nami implementovaných honeypotov, ktoré rozoberieme v nasledujúcich kapitolách.
- ***WireGuard*** (viz. **Podkapitola 4.6**) – v tomto prípade v úlohe servera, ktorý prijíma od klientov (honeypotov) prostredníctvom VPN tunela logy, obsahujúce záznamy útokov na konkrétny honeypot. Je nutné podotknúť, WireGuard rozhrania sú navrhnuté tak, aby jediná komunikácia, ktorá putuje prostredníctvom tunela, bola práve komunikácia potrebná na bezpečný prenos logov. Ostatná komunikácia putuje „klasickou“ cestou.
- ***NGINX*** (viz. **Podkapitola 4.5**) – vzhľadom k tomu, že Kibana je v základe dostupná len na adrese *localhost* a porte *5601,* nie je možné sa k nej dostať bez toho, aby sa používateľ prihlásil priamo na daný VS (ELK). Toto samozrejme nie je žiadúce, bolo teda nutné nejakým spôsobom zabezpečiť prístup ku Kibane aj „z vonku“, teda mimo samotného systému ELK a pochopiteľne tento proces zabezpečiť. Vzhľadom

k tomu, že Kibana v základe neposkytuje overenie užívateľa, a teda ktokoľvek, kto zadá jej adresu je automaticky „prihlásený“. Riešením bola práve implementácia NGINX reverse-proxy (viz. **Podkapitola 4.5.1**) spolu s valídnym certifikátom (pre doménu *dphptest.fai.utb.cz*), poskytnutým Univerzitou Tomáše Bati, aby sme zabezpečili, že žiadny klient nekomunikuje priamo s našou implementáciou Kibana. Pred otvorením Kibana rozhrania teda NGINX vyžaduje overenie užívateľa pomocou predom definovanej kombinácie mena a hesla a že celá táto komunikácia je šifrovaná (HTTPS) – najmä vzhľadom k tomu, že implementuje autentizáciu užívateľa pomocou mena a hesla.

V nasledujúcich podkapitolách sa pozrieme na výsledky našej implementácie a ukážeme konfigurácie, ktorými sme ju docielili.

6.1.1 WireGuard

Konfigurácia nižšie predstavuje konfiguráciu WireGuard VPN tunela, pomocou ktorého sú do systému ELK dodávané logy získané z honeypotov – v tomto prípade troch (príklad každého z nami navrhovaných honeypotov – WEB-POT, PyRDP a T-Pot). Blok konfigurácie rozhrania servera sa začína kľúčovým slovom [Interface] a končí sa pred ďalším kľúčovým slovom [Peer], kde začína blok konfigurácie klienta, resp. honeypotu, využívajúceho daný tunel. Blok konfigurácie rozhrania servera teda obsahuje nasledujúce riadky:

- ***Address = 10.0.0.1/24*** – definícia IP adresy WireGuard rozhrania servera,
- ***SaveConfig = true*** – inštrukcia pre WireGuard o tom, že daná konfigurácia je perzistentná, teda ju chceme uložiť a používať aj medzi reštartmi celého systému, či WireGuard servera a taktiež, že chceme uložiť zmeny vykonané v konfigurácii aj za behu rozhrania,
- ***PostUP*** – definuje príkazy, ktoré sa majú vykonať pri spustení daného rozhrania,
    - ***iptables*** – nástroj operačného systému Linux slúžiaci na správu nastavenia firewall, ktorý umožňuje definovať pravidlá pre riadenie prichádzajúcej a odchádzajúcej sieťovej premávky,
    - ***-A FORWARD*** – pridáva záznam do reťazca FORWARD, ktorý definuje ako sa má systém chovať k sieťovej premávke, ktorá nie je určená priamo pre daný systém, ale má byť preposlaná ďalej,

- ***-i wg0*** – udáva rozhranie pre ktoré ma dané pravidlo platiť, v tomto prípade rozhranie *wg0*,
- ***-j ACCEPT;*** – udáva, že premávka, ktorá zodpovedá pravidlu má byť akceptovaná. Bodkočiarka symbolizuje koniec pravidla a začiatok ďalšieho pravidla.
- ***-t nat*** – hovorí iptables, aby použil tabuľku NAT, ktorá sa používa na zmenu zdrojovej alebo cieľovej adresy premávky pri jej prechode cez systém,
- ***-A POSTROUTING*** – pridáva pravidlo do reťazca POSTROUTING, ktorý sa používa na upravenie premávky po tom, čo už bola rozhodnutá jej cesta,
- ***-o ens33*** – určuje výstupné rozhranie, cez ktoré premávka odchádza, v tomto prípade ens33. To je fyzické alebo virtuálne sieťové rozhranie priradené k hostiteľovi, často pripojené k hlavnej sieti alebo internetu.
- ***-j MASQUERADE*** – nastavuje akciu MASQUERADE pre premávku, čo znamená, že zdrojová IP adresa odchádzajúcej premávky bude zmenená na IP adresu rozhrania ens33. To umožňuje viacerým zariadeniam za VPN používať jednu verejnú IP adresu na komunikáciu s internetom, čím sa zjednodušuje NAT a zabezpečuje, že odpovede na požiadavky od klientov prichádzajúce cez VPN sú správne smerované späť.

- ***PostDown*** – definuje príkazy, ktoré sa majú vykonať po vypnutí daného rozhrania,
- ***ListenPort*** – definuje port, na ktorom WireGuard server prijíma požiadavky na komunikáciu od klientov.

Konfigurácia ďalej obsahuje tri bloky príkazov začínajúce kľúčovým slovom [Peer]. Každý z týchto blokov potom obsahuje nasledujúce tri riadky:

- ***PublicKey*** – obsahuje verejný kľúč daného klienta, potrebný na dešifrovanie komunikácie, ktorú klient zašifroval svojím privátnym kľúčom a odoslal prostredníctvom tunela,
- ***AllowedIPs*** – obsahuje IP adresy WireGuard rozhrania, z ktorých je tento kľúč použiteľný, resp. teda IP adresy WireGuard rozhrania, z ktorých môže daný klient, s použitím daného kľúča, so serverom komunikovať. Ako môžeme vidieť, v našom prípade sa jedná o jednu konkrétnu IP adresu pre všetkých klientov,
- ***Endpoint*** – IP adresa konkrétneho klienta v danej sieti a port, na ktorom klient očakáva a odpovedá na prichádzajúce VPN spojenia. V našom prípade sa jedná o internú IP adresu, špecifickú pre našu testovaciu infraštruktúru.

```
[Interface]
Address = 10.0.0.1/24
SaveConfig = true
PostUp = iptables -A FORWARD -i wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o
ens33 -j MASQUERADE;
PostDown = iptables -D FORWARD -i wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -
o ens33 -j MASQUERADE;
ListenPort = 51820
PrivateKey = (Skryté pre potreby tlače)

[Peer]
PublicKey = w6JgAiSJNUvYdn68GJ0yz1ZnZzndeaAfJWyoWy0KbXQ=
AllowedIPs = 10.0.0.2/32
Endpoint = 192.168.100.34:43765

[Peer]
PublicKey = 8Bu3V6mydEHekNfJqxwkptEkkKZe6uDoKvxXNPRLDSE=
AllowedIPs = 10.0.0.3/32
Endpoint = 192.168.100.29:37710

[Peer]
PublicKey = Dy/Vm2bFGEyNzbmvY5nm9HnPRBnTFESMUAyJGDi8424=
AllowedIPs = 10.0.0.4/32
Endpoint = 192.168.100.45:54898
```

Po spustení a otestovaní bolo posledným krokom nastavenie automatického spustenia WireGuard servera, resp. rozhrania *wg0* pomocou príkazu „***sudo systemctl enable wg-quick@wg0***".

Konfigurácia sa v systéme ELK nachádza v ***/etc/wireguard/wg0.conf***

6.1.2 ELK Stack

Ako už nám je v tomto bode zrejmé, ELK stack pozostáva z troch častí, resp. projektov. Je teda pochopiteľné, že po inštalácii týchto projektov do systému sú automaticky vytvorené tri konfiguračné súbory – jeden pre každý z projektov. V prípade Elasticsearch a Kibana projektov, boli zmeny v konfiguráciách minimálne a väčšina konfiguračného súboru je automaticky generovaná pri inštalácií a/alebo neaktívna („#" na začiatku riadku označuje v programovacích jazykoch poznámku a teda daný riadok nie je vykonávaný). Preto v tejto podkapitole rozoberieme len konkrétne riadky, ktoré sme v konfigurácii zmenili, či „aktivovali". V prípade Logstash projektu je opak pravdou. Konfigurácia Logstash projektu je pomerne rozsiahla, a preto sa zameriame len na konkrétne modelové prípady. Kompletné

konfigurácie budú dostupné v prílohách tejto práce (viz. **PRÍLOHA P I: OBSAH CD**). V samotnom systéme ELK by sme ich našli tu:

- ***/etc/elasticsearch/elasticsearch.yml***
- ***/etc/kibana/kibana.yml***
- ***/etc/logstash/conf.d/logstash-sample.conf***

6.1.2.1 Elasticsearch

V prípade Elasticsearch konfigurácie nás bude zaujímať len sieťová časť konfigurácie (z angl. *Network*) nižšie a v nej taktiež len jeden konkrétny riadok:

- ***network.host: 10.0.0.1*** – špecifikujeme na akej IP adrese má byť Elasticsearch dostupný, port v tomto prípade nie je explicitne definovaný, ale ako môžeme vidieť, v konfigurácii nižšie sa hovorí, že pokiaľ port nedefinujeme, Elasticsearch bude používať predvolený port 9200. V tomto konkrétnom prípade sme Elasticsearch nastavili tak, aby bol dostupný na IP adrese 10.0.0.1 – čo je adresa WireGuard rozhrania z predchádzajúcej **Podkapitoly 6.1.1**. Takto sme docielili, že Elasticsearch bude dostupný len priamo zo systému ELK alebo pomocou zabezpečeného VPN rozhrania (tunela), tak ako sme ho definovali, v už spomínanej, predchádzajúcej podkapitole.

```
# ---------------------------------- Network -----------------------------------
#
# By default Elasticsearch is only accessible on localhost. Set a different
# address here to expose this node on the network:
#
network.host: 10.0.0.1
#
# By default Elasticsearch listens for HTTP traffic on the first free port it
# finds starting at 9200. Set a specific HTTP port here:
#
#http.port: 9200
#
# For more information, consult the network module documentation.
```

Po spustení a otestovaní bolo posledným krokom nastavenie automatického spustenia pomocou príkazu „***sudo systemctl enable elasticsearch***“.

6.1.2.2 Kibana

Podobne ako v prípade konfigurácie Elasticsearch sme v prípade konfigurácie Kibana taktiež museli vykonať minimum zmien. Konkrétne sa jednalo o zmenu na jednom riadku, týkajúcu sa nastavenia IP adresy pre elasticsearch, aby Kibana vedela na akej adrese sa Elasticsearch nachádza. Tento riadok zobrazuje časť konfigurácie nižšie:

- ***elasticsearch.hosts: ["http://10.0.0.1:9200"]*** – tento riadok teda hovorí, že Elasticsearch sa nachádza na IP adrese 10.0.0.1 a porte 9200 – presne tak, ako to môžeme vidieť v časti konfigurácie pre Elasticsearch vyššie. Hranaté zátvorky umožňujú definovanie viacerých inštancií Elasticsearch. Toto v našom prípade ale nie je relevantné.

```
# ================== System: Elasticsearch ==================
# The URLs of the Elasticsearch instances to use for all your queries.
elasticsearch.hosts: ["http://10.0.0.1:9200"]
```

Po spustení a otestovaní bolo posledným krokom nastavenie automatického spustenia pomocou príkazu „***sudo systemctl enable kibana***".

6.1.2.3 Logstash

Ako sme spomenuli v úvode tejto kapitoly, konfigurácia Logstash je pomerne rozsiahla a jej detailný popis by zabral veľkú časť práce a nemusel by byť zrovna efektívny, vzhľadom k tomu, že postupy sa často opakujú (pre rozdielne honeypoty), prípadne sa od seba líšia len veľmi málo. Konfigurácia je taktiež len upravenou verziou, originálnej konfigurácie T-Pot projektu, ktorá je dostupná na adrese zdroja [24]. Úprava tejto originálnej konfigurácie spočívala v častiach, ktoré sme pridali, kvôli rozšíreniu o ďalšie honeypoty (viz. **Kapitoly 7** a **8**) a v tom, že bolo nutné čiastočne pozmeniť syntax. Zmena syntaxy bola potrebná kvôli faktu, že T-Pot projekt predpokladá, že Logstash je inštalovaný na rovnakom zariadení ako samotné honeypoty a sám si logy pozbiera. V našom prípade, ako sme uviedli v **Kapitole 5**, na zber logov využívame protokol FileBeat (okrem logov z NGINX), ktorý následne logy zasiela cez zabezpečený tunel do lokálnej inštalácie Logstash na systéme ELK. V tejto kapitole preto len stručne definujeme najmä časti v konfigurácii, ktoré sme z nejakého dôvodu pozmenili alebo pridali

Hneď na začiatok konfigurácie sme pridali jednoduchý blok, ktorý je zobrazený nižšie. Tento odsek začína kľúčovým slovom *input* a definujeme v ňom dva vstupy. Prvý z nich definujeme ako *beats*, v našom prípade konkrétne FileBeat na porte 5044 – čo je štandardný

port pre Logstash projekt. Na tomto porte bude Logstash prijímať logy, ktoré mu budú zasielať FileBeat inštancie na honeypotoch. Druhý zo vstupov je vstup, ktorý sa nachádza priamo na zariadení, na ktorom sa nachádza aj Logstash. Logstash inštancii teda predávame priamo cestu k súboru, ktorý ma sledovať na zmeny. Taktiež pridávame premennú *type*, ktorá neskôr slúži na aplikovanie správneho filtra – bude vysvetlené ďalej v aktuálnej podkapitole.

```
input {
  beats {
    port => 5044
  }

  file {
    path => ["/var/log/nginx/access.log"]
    type => "nginx"
  }
}
```

Hneď pod týmto blokom vidíme väčší blok, slúžiaci pre honeypoty WEB-POT a PyRDP a aj honeypoty v projekte T-Pot – zobrazený nižšie. Táto časť začína blokom uzatvoreným v zložených zátvorkách, počínajúc kľúčovým slovom *filter*, ktorý sa v Logstash projekte používa na vytvorenie bloku, slúžiaceho na definovanie akým spôsobom má Logstash spracovať prichádzajúce dáta. Následne definujeme dve podmienky, ktoré porovnajú či sa premenná *type* (typ) zhoduje s hodnotou za „==“, a ak áno aplikuje blok *dissect*. Premenná *type* je definovaná v FileBeat konfigurácii na danom honeypote a odosielaná spolu s dátami. V kapitolách, v ktorých sa budeme venovať príslušným honeypotom, bude táto konfigurácia zobrazená. Dvojicu podmienok môžeme vidieť tu:

- ***if [type] == “webpot”***,
- ***if [type] == “pyrdp”***.

Ak sa premenná *type* zhoduje s jedným z definovaných typov, *dissect* bloky rozoberú prichádzajúce dáta, na základe definovaných reťazcov:

- *"%{timestamp}* | *Username: %{username}* | *Password: %{password}* | *IP: %{ip}* | *Agent: %{ua_string}* | *Referrer: %{referrer}* | *Country: %{country}* | *City: %{city}* | *Subdivision: %{subdivision}* | *Continent: %{continent}* | *Latitude: %{latitude}* | *Longitude: %{longitude}"* – v prípade **WEB-POT** honeypotu,
- *"message" => "%{timestamp}* | *Username: %{username}* | *Password: %{password}* | *ATTACKER_IP: %{ATTACKER_IP}* | *ATTACKER_HOST:*

%{ATTACKER_HOST} | Country: %{country} | City: %{city} | Subdivision: %{subdivision} | Continent: %{continent} | Latitude: %{latitude} | Longitude: %{longitude} | File: %{replay_file_name}" – v prípade **PyRDP** honeypotu.

Oba reťazce sú si v mnohom podobné, avšak obsahujú svoje špecifické časti, špeciálne pre potreby daného honeypotu. V prípade, že nie je možné prichádzajúce logy z nejakého dôvodu spracovať (napr. v prípade nesprávneho formátu resp. zloženia), blok nie je vykonaný a vkladáme *tag*, ktorý informuje o tom, že v tomto bloku nastala chyba:

- ***tag_on_failure => [_dissectfailureWEBPOT"]***,
- ***tag_on_failure => [_dissectfailurePYRDP"]***.

```
if [type] == "webpot" {
  # Apply specific filters for web-pot
  dissect {
    mapping => {
     "message" => "%{timestamp} | Username: %{username} | Password: %{password}
     | IP: %{ip} | Agent: %{ua_string} | Referrer: %{referrer} | Country:
     %{country} | City: %{city} | Subdivision: %{subdivision} | Continent:
     %{continent} | Latitude: %{latitude} | Longitude: %{longitude}"
    }
    tag_on_failure => ["_dissectfailureWEBPOT"]
  }
}
# Check if the event came from pyrdp
else if [type] == "pyrdp" {
  # Apply specific filters for pyrdp
  dissect {
    mapping => {
     "message" => "%{timestamp} | Username: %{username} | Password: %{password}
     | ATTACKER_IP: %{ATTACKER_IP} | ATTACKER_HOST: %{ATTACKER_HOST} | Country:
     %{country} | City: %{city} | Subdivision: %{subdivision} | Continent:
     %{continent} | Latitude: %{latitude} | Longitude: %{longitude} | File:
     %{replay_file_name}"
    }
    tag_on_failure => ["_dissectfailurePYRDP"]
  }
}
```

Podobný štýl spracovania (nevyužíva sa blok *dissect*, ale prevažne kombinácie blokov *date, mutate* a *rename*) opakujeme pre všetky honeypoty z projektu T-Pot, pričom sme ich konfiguráciu upravili len z hľadiska syntaxe, kvôli tomu, že v pôvodnej implementácii T-Pot si logy z honeypotov zbieral priamo Logstash, čo v našom prípade neplatí. Pomocou premennej *type* teda identifikujeme adekvátny filter pre daný honeypot, resp. jeho logy

a následne na ne tento filter aplikujeme, čím ich upravíme do nami požadovanej podoby. Výraznejšou zmenou si prešiel len blok spracúvajúci logy NGINX, kvôli tomu, že sme ho upravili pre potreby vlastnej implementácie NGINX reverse-proxy v našom projekte. Konfiguráciu môžeme vidieť nižšie, pričom sa opäť jedná o štandardný postup – pomocou premennej *type*, ktorú sme definovali v bloku vstupov (*input*), identifikujeme NGINX logy a aplikujeme náš filter. Pomocou bloku *grok* extrahujeme z logov pre nás relevantné dáta – IP adresa, z ktorej sa pristupovalo (*IP:**source_ip**)*, používateľské meno použité pri prihlásení (*DATA:**user_name***) a čas (*HTTPDATE:**nginx_timestamp***). V bloku *date* potom upravíme formát času pred odoslaním.

```
  if [type] == "nginx" {
    grok {
      match => { "message" => "%{IP:source_ip} - %{DATA:user_name}
\[%{HTTPDATE:nginx_timestamp}\]" }
    }
    date {
      match => [ "nginx_timestamp", "dd/MMM/yyyy:HH:mm:ss Z" ]
      target => "@timestamp"
    }
  }
```

Takmer pred koncom konfigurácie, po tom ako sme definovali všetky bloky slúžiace na spracovanie logov z daného honeypotu, nasleduje blok zabezpečujúci geolokáciu pre honeypoty z projektu T-Pot, vzhľadom k tomu, že geolokáciu pre logy z VS WEB-POT a VS PyRDP vykonávame priamo na danom systéme, pomocou lokálnej inštalácie GeoLite2 databáz (viz. **Podkapitoly 7.1.4** resp. **8.1.5**). Tento blok môžeme vidieť nižšie. Na začiatku bloku overíme, či premenná *src_ip* existuje (či sa v daných logoch nachádza), a ak áno, vykonáme na jej základe, pomocou troch vnorených blokov geolokáciu, vyhľadanie príslušného ASN a reputácie danej IP – ak je dostupná v súbore *iprep.yaml*.

```
# Add geo coordinates / ASN info / IP rep.
  if [src_ip] {
    geoip {
      cache_size => 10000
      source => "src_ip"
      target => "[source][geo]"
      default_database_type => "City"
    }
    geoip {
      cache_size => 10000
      source => "src_ip"
      target => "[source][asn]"
      default_database_type => "ASN"
```

```
    }
    translate {
      refresh_interval => 86400
      source => "src_ip"
      target => "ip_rep"
      dictionary_path => "/etc/listbot/iprep.yaml"
    }
  }
```

Na koniec konfigurácie následne pridávame blok začínajúci kľúčovým slovom *output* (zobrazený nižšie), v ktorom následne definujeme dva výstupy pre logy, po tom, čo prešli úpravou. Na začiatku bloku overíme pomocou premennej *type,* či sa jedná o NGINX logy, a ak áno, zašleme ich do Elasticsearch inštancie na adrese *http://localhost:9200* a indexu, ktorý je dynamicky vytvorený vo formáte *nginx-%{+YYYY.MM.dd}*. Napríklad pre dátum 23. Apríla 2024 by sme teda dostali index *nginx-2024.03.23* – ak takýto index už existuje, zašleme dáta do neho, ak nie, index je vytvorený a následne sú do neho uložené dáta. Podmienka *else* potom zahŕňa všetky ostatné logy, pričom postupujeme veľmi podobne. Logy zasielame do inštancie Elasticsearch, na adrese *http://localhost:9200,* do dynamicky vytvoreného indexu, vo formáte *{[@metadata][beat]}-{[@metadata][version]}-{+YYYY.MM.dd}*. Teda pre dátum 23. Apríla 2024 by sme dostali index *filebeat-8.12.1-2024.03.23* – kde *8.12.1* predstavuje verziu Filebeat protokolu. Rovnako platí, že ak takýto index už existuje, zašleme dáta do neho, ak nie, index je vytvorený a dáta sú do neho uložené následne.

```
output {
  if [type] == "nginx" {
    # Elasticsearch output block for Nginx logs
    elasticsearch {
      hosts => ["http://localhost:9200"]
      index => "nginx-%{+YYYY.MM.dd}"
    }
  } else {
    # Default Elasticsearch output block for other types of logs
    elasticsearch {
      hosts => ["http://localhost:9200"]
      index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"
    }
  }
}
```

Po spustení a otestovaní bolo posledným krokom nastavenie automatického spustenia pomocou príkazu „***sudo systemctl enable logstash***”.

6.1.3 NGINX

Nasledujúca konfigurácia nižšie zobrazuje konfiguračný súbor NGINX reverse-proxy, spolu s nastavením autentifikácie užívateľa pomocou predom definovaných prihlasovacích údajov a funkčným HTTPS protokolom – proces vytvorenia prihlasovacích údajov popíšeme neskôr (viz **Podkapitola 6.1.3.1**), hneď po tom, ako bližšie definujeme jednotlivé časti NGINX konfigurácie.

Samotná konfigurácia by sa dala rozdeliť na dva hlavné bloky, resp. definície serverov – začínajúce kľúčovým slovom *server* a uzatvorené v zložených zátvorkách. Prvý z týchto serverov je server slúžiaci na príjem nešifrovaného HTTP protokolu a obsahuje nasledujúce riadky:

- ***listen 80 default_server*** – tento riadok hovorí, že NGINX ma prijímať prichádzajúce požiadavky na porte 80 (štandardný HTTP port) zo všetkých IPv4 adries a že táto konfigurácia má byť použitá pre všetky požiadavky na tomto porte, bez ohľadu na užívateľom zadané URL, resp. doménu (server_name),
- ***listen [::]:80 default_server*** – verzia predchádzajúceho riadku pre IPv6 adresy,
- ***server_name dphptest.fai.utb.cz*** – definícia našej špecifickej domény (URL), pre ktoré ma byť daná konfigurácia použitá. Tento riadok by vo svojej podstate mohol byť vynechaný vzhľadom k tomu, že v prvom a druhom riadku sme definovali, že NGINX má túto konfiguráciu použiť pre všetky požiadavky na porte 80 (HTTP). Avšak pre lepšiu čitateľnosť sme sa rozhodli ho tu ponechať.
- ***location*** **/** - definuje blok uzatvorený v zložených zátvorkách, vzťahujúci sa na koreňovú cestu a všetky podcesty pre dané URL (dphptest.fai.utb.cz). Teda, že akákoľvek požiadavka prichádzajúca na tento port, bude ovplyvnená pravidlami definovanými v tomto bloku.
    - ***return 301 https://$host$request_uri*** – inštruuje NGINX, aby klienta presmeroval na HTTPS verziu daného URL (ak existuje) a vrátil mu stavový kód 301 (trvalo presunuté). *$host* obsahuje hostname, na ktorú pôvodná požiadavka putovala (doménu) a *$request_uri* potom celú cestu (query) pôvodnej požiadavky. Takto zabezpečíme, že akákoľvek požiadavka na nezabezpečenú HTTP verziu bude automaticky presmerovaná na zabezpečenú HTTPS verziu, a teda, že všetká komunikácia prebiehajúca medzi klientom a serverom bude šifrovaná.

Druhý blok opäť začína kľúčovým slovom server a je uzatvorený v zložených zátvorkách. V tomto prípade sa jedná o blok definujúci spôsob, akým má NGINX pristupovať k prichádzajúcej HTTPS komunikácii, ktorá mohla, ale nemusela byť presmerovaná z predchádzajúceho bloku. Blok obsahuje nasledujúce riadky:

- ***listen 443 ssl http2*** – tento riadok hovorí, že NGINX ma prijímať prichádzajúce požiadavky na porte 443 (štandardný HTTPS port) zo všetkých IPv4, s podporou SSL pre šifrovanie komunikácie s HTTP protokolom verzie dva,
- ***listen [::]:443 ssl http2*** – verzia predchádzajúceho riadku pre IPv6 adresy,
- ***server_name dphptest.fai.utb.cz*** – definícia našej špecifickej domény (URL) pre ktoré ma byť daná konfigurácia použitá,
- ***location*** **/** - opäť definuje blok uzatvorený v zložených zátvorkách, vzťahujúci sa na koreňovú cestu a všetky pod cesty pre dané URL (dphptest.fai.utb.cz). Teda, rovnako ako v predchádzajúcom prípade platí, že akákoľvek požiadavka prichádzajúca na tento port bude ovplyvnená pravidlami definovanými v tomto bloku.
    - ***auth_basic "Restricted Access"*** – zapína pre danú lokáciu (koreňovú lokáciu /) autentizáciu pomocou predom definovaného mena a hesla a zobrazí užívateľovi hlášku uvedenú v zátvorkách,
    - ***auth_basic_user_file /etc/nginx/.kibana*** - definuje súbor (a cestu k nemu), v ktorom má NGINX hľadať kombinácie mien a hesiel na autentizáciu užívateľov. V tomto prípade sa jedná o skrytý súbor (bodka pred menom súboru) s názvom kibana.
    - ***proxy_pass*** – uvádza kam má NGINX presmerovať požiadavky vedúce na danú lokáciu. V tomto prípade sa jedná o lokálnu inštanciu kibana na štandardnom porte 5601.
    - ***proxy_http_version 1.1*** – definícia verzie HTTP protokolu pre komunikáciu medzi NGINX a Kibanou,
    - ***proxy_set_header Upgrade $http_upgrade*** **a** ***proxy_set_header Connection 'upgrade'*** – umožňujú upgrade komunikácie na WebSockets,
    - ***Proxy_set_header Host $host*** – prepisuje hlavičku Host v požiadavke na hodnotu Host z pôvodnej požiadavky,
    - ***proxy_cache_bypass $http_upgrade*** – zaisťuje, že ak dôjde k upgrade spojenia na WebSockets, obchádza sa pôvodná cache spojenia.

Nasledujúce riadky sa zameriavajú na nastavenie SSL, ktoré zabezpečuje šifrovanú komunikáciu medzi serverom a klientom:

- ***ssl_certificate /home/utb/DPhoneyTest/dphptest.fai.utb.cz.pem*** – hovorí NGINX, kde je uložený certifikát pre SSL,
- ***ssl_certificate_key /home/utb/DPhoneyTest/dphptest.fai.utb.cz.pkey*** – hovorí NGINX, kde je uložený kľúč k danému certifikátu,
- ***ssl_session_timeout 1d*** – nastavuje limit platnosti jedného SSL pripojenia od klienta na 1 deň. To znamená, že pri opätovnom pripojení v rámci jedného dňa, klient nemusí opäť prechádzať autentizáciou a celkovým nadviazaním spojenia.
- ***ssl_session_cache shared:MozSSL:10m*** – nastavuje veľkosť cache, ktorá udržuje informácie o spojeniach na 10 megabajtov,
- ***ssl_session_tickets off*** – vypína používanie SSL Session Tickets, ktoré klientom umožňujú uchovávanie šifrovacích kľúčov pri opätovnom pripojení a slúži ako alternatíva k SSL Session Cache.

Ďalšia časť konfigurácie definuje nastavenie používania protokolov SSL, resp. TLS a obsahuje tieto riadky:

- ***ssl_protocols TLSv1.3*** – určuje, ktoré verzie protokolov SSL/TLS by mal server podporovať pri naviazaní šifrovaného spojenia. V tomto prípade je nastavenie obmedzené na TLSv1.3, ktorý je aktuálne najnovšou a najbezpečnejšou verziou protokolu.
- ***ssl_prefer_server_ciphers off*** – riadi, či by server mal vynútiť použitie svojej vlastnej preferencie šifier pri SSL/TLS handshaku, namiesto toho, aby nechal na klientovi, aby si vybral z ponúkaných možností. Toto nastavenie nie je v našom kontexte až také podstatné, vzhľadom k tomu, že v predchádzajúcom riadku sme jasne definovali verziu TLS protokolu, ktorú chceme používať. V prípade TLSv1.3 sú aktuálne všetky šifry považované za bezpečné, a teda môžeme nechať výber na klienta.

Nasledujúci odsek obsahuje len jeden riadok a to:

- ***add_header Strict-Transport-Security "max-age=63072000" always*** – pridáva do hlavičky odpovedí servera HSTS pravidlo, ktoré núti prehliadač na definovanú dobu (v tomto prípade 63072000 sekúnd) pristupovať k webu výlučne cez HTTPS.

Hneď za týmto riadkom sa nachádza odsek obsahujúci dva riadky týkajúce sa funkcie OCSP stapling:

- ***ssl stapling on*** – Táto direktíva zapína OCSP stapling. Server NGINX sa pokúsi získať a uchovávať aktuálne OCSP odpovede pre svoje certifikáty od autority, ktorá certifikát vydala (CA). Tieto odpovede potom poskytuje klientom počas TLS handshaku, čo umožňuje klientom overiť stav certifikátu, bez nutnosti kontaktovať OCSP server CA,
- ***ssl_stapling_verify on*** – nastavuje, že NGINX overí OCSP odpoveď od CA predtým, než ju priloží k handshaku. Toto zabezpečuje, že OCSP odpoveď je v danom čase valídna.

Predposledný a posledný odsek obsahujú po jednom riadku a to:

- ***ssl_trusted_certificate /home/utb/DPhoneyTest/chainECC2021.pem*** – hovorí NGINX, kde je uložený kompletný reťazec certifikátov, až k certifikátu koreňovému, teda ku konkrétnej CA. Súbor *chainECC2021.pem,* by mal teda obsahovať všetky certifikáty potrebné pre vytvorenie dôveryhodného reťazca, od serverového certifikátu až po koreňový CA certifikát.
- ***resolver 127.0.0.1*** – udáva DNS server, ktorý má NGINX používať, v tomto hovoríme, že NGINX má používať lokálny DNS server, čo je pre potreby našej testovacej infraštruktúry postačujúce.

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name dphptest.fai.utb.cz;

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name dphptest.fai.utb.cz;

    location / {
        auth_basic "Restricted Access";
        auth_basic_user_file /etc/nginx/.kibana;

        proxy_pass http://localhost:5601;
```

```
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }

    ssl_certificate /home/utb/DPhoneyTest/dphptest.fai.utb.cz.pem;
    ssl_certificate_key /home/utb/DPhoneyTest/dphptest.fai.utb.cz.pkey;
    ssl_session_timeout 1d;
    ssl_session_cache shared:MozSSL:10m;  # about 40000 sessions
    ssl_session_tickets off;

    # modern configuration
    ssl_protocols TLSv1.3;
    ssl_prefer_server_ciphers off;

    # HSTS (ngx_http_headers_module is required) (63072000 seconds)
    add_header Strict-Transport-Security "max-age=63072000" always;

    # OCSP stapling
    ssl_stapling on;
    ssl_stapling_verify on;

    # verify chain of trust of OCSP response using Root CA and Intermediate certs
    ssl_trusted_certificate /home/utb/DPhoneyTest/chainECC2021.pem;

    # replace with the IP address of your resolver
    resolver 127.0.0.1;
}
```

Konfigurácia sa v systéme ELK nachádza v ***/etc/nginx/sites-available/kibana*** a jej základ bol vygenerovaný pomocou nástroja na adrese zdroja [43], aby sme zabezpečili spoľahlivosť a bezpečnosť konfigurácie zodpovedajúcu dnešným štandardom. Podobne ako v predchádzajúcom prípade sme NGINX server nastavili na automatický štart pomocou príkazu operačného systému Ubuntu – *enable*.

6.1.3.1 APACHE2-UTILS

Apache2-utils je balíček nástrojov pre webové servery, dostupný aj pre operačný systém Ubuntu. Pre našu prácu sme využili jeden z nástrojov tohto balíčka, a to konkrétne nástroj *htpasswd.* Tento nástroj nám pomocou jednoduchého príkazu umožnil vytvoriť súbor (jednoduchú databázu), ktorý obsahuje prihlasovacie mená a k nim prislúchajúce heslá pre autentizáciu užívateľov do Kibana rozhrania pomocou NGINX reverse proxy. Tento súbor sme vytvorili pomocou nasledujúceho príkazu:

- ***htpasswd -c /etc/nginx/.kibana kibana_admin*** – príkaz teda pozostáva z 3 častí. V prvej definujeme, použitý nástroj *htpasswd* s príznakom *-c*, ktorým hovoríme, že chceme vytvoriť nový súbor pre našu pomyselnú databázu mien a hesiel. V druhej časti poskytujeme samotnú cestu k súboru, a teda aj jeho názov *.kibana*. Tento súbor sme už mohli vidieť v NGINX konfigurácii vyššie, v časti zaoberajúcej sa HTTPS konfiguráciou. V poslednej, tretej časti potom definujeme meno prvého užívateľa, ktorého chceme vytvoriť – *kibana_admin*. Po zadaní príkazu sme vyzvaní na zadanie hesla, ktoré chceme k danému používateľskému menu priradiť a samotný nástroj potom z tohto hesla vytvorí HASH a uloží ho do novo vytvoreného súboru ako prvý záznam vo formáte <používateľské_meno>:<HASH_hesla>, ako to zobrazuje **Obrázok 37**. Takto by sme mohli v prípade potreby vygenerovať nových užívateľov, resp. ich prihlasovacie údaje. V prípade vytvorenia nového používateľa príznak *-c* nevytvára nový súbor s jediným používateľom, ale pridá nového používateľa do už exitujúceho súboru.

```
1 kibana_admin:$apr1$DnLTGwa2$/W7UCXzDnuOHhfqxCwqar/
```

Obrázok 37 - Príklad uloženia prihlasovacích údajov, Zdroj: vlastný

7 WEB-POT

VS WEB-POT je prvým honeypotom, ktorý v tejto práci rozoberieme. Myšlienkou za jeho vznikom bolo vytvorenie webového honeypotu, ktorý by zbieral útočníkom zadané prihlasovacie údaje, ukladal ich do logov a odosielal tieto logy na ďalšie spracovanie. VS WEB-POT vo svojej finálnej podobe zbiera od útočníka o niečo viac informácií – jeho konečnú funkcionalitu a architektúru sme už načrtli v **Podkapitole 5.3**. V tejto kapitole sa na VS WEB-POT pozrieme bližšie a vysvetlíme jeho implementáciu. Rovnako ako v predchádzajúcej **Kapitole 6** sa budeme zaoberať len súčasťami, ktoré sú pre prácu relevantné alebo ktoré sme do systému pridali, či nejako upravili pre potreby práce, navrhovaného riešenia, či cieľovej infraštruktúry.

7.1 Implementácia

Komponentami virtuálneho systému, tak ako ich zobrazuje **Obrázok 36**, sú:

- ***Apache2*** – je asi jasné, že webový honeypot potrebuje na svoje fungovanie webový server. V našom prípade sme teda zvolili webový server Apache2, na ktorom udržujeme PHP kód pre falošné webové rozhrania, ako aj PHP skripty slúžiace na simuláciu procesu prihlasovania, zber informácií o útočníkovi a zadaných prihlasovacích údajoch, či geolokáciu.
- ***WireGuard*** – vo virtuálnom systéme ELK z predchádzajúcej kapitoly sme WireGuard konfigurovali do pozície servera. VS WEB-POT je v tomto prípade v pozícii klienta, ktorý pomocou zabezpečeného VPN tunela zasiela získané logy, resp. informácie o útočníkovi do virtuálneho systému ELK na ďalšie spracovanie a vizualizáciu. Opäť platí, že VPN tunel je nakonfigurovaný tak, aby jediná komunikácia (dáta), ktorá ním putuje, bola komunikácia spojená s prenosom logov z virtuálneho systému WEB-POT do virtuálneho systému ELK, resp. teda Logstash inštancie, ktorá je na ňom nainštalovaná a tieto dáta očakáva.
- ***FileBeat*** – v predchádzajúcom bode sme povedali, že pomocou zabezpečeného WireGuard VPN tunela zasielame logy z virtuálneho systému WEB-POT do virtuálneho systému ELK. Tieto logy treba teda pochopiteľne sledovať – či v nich nenastala nejaká zmena a v prípade, že áno, na túto zmenu zareagovať a odoslať novú časť logov - práve na toto nám slúži protokol FileBeat.

- ***GeoLite2*** – je, ako sme uviedli v **Podkapitole 4.8**, voľne dostupná verzia GeoIP2 databáz od spoločnosť MaxMind – stačí jednoduché prihlásenie na webových stránkach MaxMind a vygenerovanie licenčného kľúča pre voľne dostupnú verziu. V našom prípade sme potom zvolili lokálnu inštaláciu tejto databázy, resp. databáz na VS WEB-POT.

V nasledujúcich podkapitolách sa pozrieme na výsledky našej implementácie a ukážeme skripty a konfigurácie, ktorými sme ju docielili.

7.1.1 Apache2

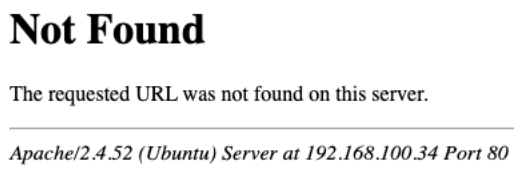
7.1.1.1 Inštalácia a základná konfigurácia

Inšpirácia pri tvorbe webového servera a jeho funkcionality bola inšpirovaná návodom na adrese zdroja [44]. Inštalácia balíka samotného servera (*apache2*) ako aj balíka *libapache2-mod-php*, ktorý webovému serveru umožňuje prácu s jazykom PHP (– jeho úloha v projekte bude objasnená neskôr v tejto kapitole), teda prebiehala pomocou balíčkovacieho systému *apt*. Po zadaní IP adresy virtuálneho systému WEB-POT do prehliadača sa nám zobrazila základná stránka informujúca používateľa o úspešnej inštalácii. Ďalším krokom bola inštalácia balíkov *a2enmod headers* a *libapache-mod-security2*. Prvý z balíkov nám umožňuje čítať hlavičky v komunikácii medzi klientmi (útočníkmi) a naším serverom, a druhý z balíkov predstavuje webový firewall poskytujúci dodatočnú ochranu pre náš webový server pomocou preddefinovaných pravidiel. Následne sme prešli k modifikácii konfiguračného súboru, ktorý apache2 pri inštalácii automaticky vytvoril a uložil v */etc/apache2/conf-available/security.conf* - upravená verzia je zobrazená nižšie a obsahuje tieto dôležité riadky:

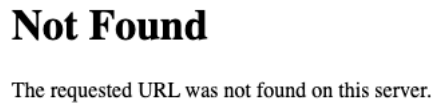
- ***ServerTokens Prod*** a ***ServerSignature off*** – tieto nastavenia zabraňujú odosielaniu informácií o serveri a hostiteľskom systéme v hlavičke HTTP odpovede. V základnom nastavení, napríklad pri zadaní adresy, ktorá na danom serveri neexistuje, odošle server chybovú správu, že daná adresa na serveri neexistuje a informácie o ňom samotnom (zobrazuje **Obrázok 38**). Po zmene nastavení odošle server len chybovú hlášku (zobrazuje **Obrázok** 39),
- ***TraceEnable off*** – vypína metódu TRACE,
- ***Header set X*-Content-Type-Options: "nosniff"** – zabraňuje prehliadaču interpretovať súbory ako iný typ, než je uvedené v HTTP hlavičkách,

- ***Header set X*-Frame-Options: "sameorigin"** – zabraňuje iným stránkam zahrnúť stránku z aktuálneho webu vo forme rámcov. Používa sa ako ochrana proti tzv. *clickjacking* útoku.

```
# ServerTokens
# This directive configures what you return as the Server HTTP response
# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of:  Full | OS | Minimal | Minor | Major | Prod
# where Full conveys the most information, and Prod the least.
#ServerTokens Minimal
ServerTokens Prod
#ServerTokens Full

# Optionally add a line containing the server version and virtual host
# name to server-generated pages (internal error documents, FTP directory
# listings, mod_status and mod_info output etc., but not CGI generated
# documents or custom error documents).
# Set to "EMail" to also include a mailto: link to the ServerAdmin.
# Set to one of:  On | Off | EMail
#ServerSignature Off
ServerSignature Off

# Allow TRACE method
#
# Set to "extended" to also reflect the request body (only for testing and
# diagnostic purposes).
#
# Set to one of:  On | Off | extended
TraceEnable Off
#TraceEnable On


# Setting this header will prevent MSIE from interpreting files as something
# else than declared by the content type in the HTTP headers.
# Requires mod_headers to be enabled.
#
Header set X-Content-Type-Options: "nosniff"

# Setting this header will prevent other sites from embedding pages from this
# site as frames. This defends against clickjacking attacks.
# Requires mod_headers to be enabled.
#
Header set X-Frame-Options: "sameorigin"
```



Obrázok 38 - Apache2 - plné informácie o serveri, Zdroj: vlastný



Obrázok 39 - Apache2 - bez informácii o serveri, Zdroj: vlastný

Nasledujúcim krokom bola konfigurácia samotného webového servera. Časť konfigurácie relevantná pre našu prácu a teda časť, ktorú sme editovali, je zobrazená nižšie, v systéme sa nachádza v */etc/apache2/apache2.conf* (pre celú konfiguráciu viz. **PRÍLOHA P I: OBSAH CD**). Konfigurácia obsahuje tri bloky ohraničené kľúčovými slovami *Directory* za ktorými nasleduje cesta, na ktorú sa daný blok vzťahuje. Prvý blok teda obsahuje tieto riadky:

- ***<Directory />*** - začiatok bloku vzťahujúceho sa na koreňový adresár („/“),
    - ***Options None*** – direktíva popisujúca, či je povolené použitie dodatočných možností (z angl. *options*) pre koreňový adresár. V tomto prípade zakazujeme použitie akejkoľvek dodatočnej možnosti pre koreňový adresár, ako je napríklad možnosť *FollowSymLinks*, ktorá je v základe povolená.
    - ***AllowOverride None*** – direktíva popisujúca, či je povolené editovanie konfigurácie pomocou *.htaccess* súboru. V tomto prípade zakazujeme prepísanie konfigurácie pomocou *.htaccess* súboru.
    - ***Require all denied*** – direktíva popisujúca, či je povolený prístup ku zdrojom (súborom) v koreňovom adresári pre dané IP adresy či používateľov. V tomto prípade zakazujeme prístup ku všetkým zdrojom v koreňovom adresári pre všetkých používateľov a všetky IP adresy.
- ***</Directory>*** - koniec bloku vzťahujúceho sa na koreňový adresár.

Druhý blok kódu je zhodný s aktuálnym blokom kódu, rozdielom je len cieľ (adresár) na ktorý sa daná konfigurácia vzťahuje. Blok teda obsahuje tieto riadky:

- ***<Directory /usr/share>*** - začiatok bloku vzťahujúceho sa na adresár („/usr/share“),
    - ***Options None*** – zhodné s prvým blokom kódu, pre adresár /usr/share

    - ***AllowOverride None*** – zhodné s prvým blokom kódu, pre adresár /usr/share
    - ***Require all denied*** – zhodné s prvým blokom kódu, pre adresár /usr/share
- ***</Directory>*** - koniec bloku vzťahujúceho sa na adresár /usr/share.

Tretí blok kódu je opäť v mnohom podobný predchádzajúcim dvom blokom, ale pridáva niekoľko podstatných riadkov. Tretí blok kódu teda obsahuje tieto riadky:

- ***<Directory /var/www>*** - začiatok bloku vzťahujúceho sa na adresár („/var/www“),
    - ***<LimitExcept GET POST>*** - začiatok bloku udávajúceho pravidlá pre metódy HTTP požiadaviek, ktoré nie sú GET alebo POST,
        - ***Order deny, allow*** – direktíva popisujúca v akom poradí sa majú vykonávať pravidlá obmedzujúce prístup. Najskôr teda vykonávame direktívy zakazujúce prístup (ako napr. direktíva na nasledujúcom riadku) a následne by sme vykonávali direktívy povoľujúce prístup (*allow*), ak by sme nejaké definovali.
        - ***Deny from all*** – direktíva explicitne zakazujúca prístup ku všetkým metódam, ktoré nie sú GET alebo POST,
    - ***</LimitExcept>*** - koniec bloku udávajúceho pravidlá pre metódy HTTP požiadaviek,
    - ***Options None*** – zhodné s prvým a druhým blokom kódu, pre adresár /var/www
    - ***AllowOverride None*** – zhodné s prvým a druhým blokom kódu, pre adresár /var/www
    - ***Require all granted*** – direktíva popisujúca, či je povolený prístup ku zdrojom (súborom) v koreňovom adresári pre dané IP adresy či používateľov. V tomto prípade povoľujeme prístup ku všetkým zdrojom v koreňovom adresári pre všetkých používateľov a všetky IP adresy.
- ***</Directory>*** - koniec bloku vzťahujúceho sa na koreňový adresár.

```
<Directory />
    Options None
    AllowOverride None
    Require all denied
</Directory>

<Directory /usr/share>
    Options None
    AllowOverride None
    Require all denied
```

```
</Directory>

<Directory /var/www/>
    <LimitExcept GET POST>
        Order deny,allow
        Deny from all
    </LimitExcept>
    Options None
    AllowOverride None
    Require all granted
</Directory>
```

V nesposlednom rade sme upravili súbor nachádzajúci sa v */etc/apache2/sites-avalaible/000-default.conf*. V tomto súbore sme definovali na akých URL chceme zobrazovať naše webové stránky, resp. honeypoty. Celá konfigurácia pozostáva z troch takmer identických blokov – každý pre jeden z našich webových stránok, ktoré bližšie definujeme v nasledujúcej podkapitole. Nebudeme teda zobrazovať každý blok samostatne, ale princíp vysvetlíme len na jednom z nich, pričom uvedieme aká hodnota by sa v danej časti nachádzala v prípade daného honeypotu, resp. teda aká hodnota sa nachádza v ďalších blokoch. Celú konfiguráciu je možné nájsť v **PRÍLOHA P I: OBSAH CD**. Jeden z blokov zobrazuje konfigurácia nižšie a obsahuje tieto riadky:

- ***<VirtualHost *:80>*** - začiatok bloku konfigurácie virtuálneho webového servera, zhodný pre všetky bloky,
- ***ServerName*** **–** direktíva udávajúca názov virtuálneho webového servera, resp. teda URL, ktorú budeme do prehliadača zadávať. Hodnota za touto direktívou sa líši v závislosti na danom bloku.
    - V tomto prípade (úvodnej stránky) je daná URL „***test.local***“, v prípade *devuseonly* honeypotu by bola „***devuseonlytest.local***“ a v prípade *moodle* honeypotu potom „***moodletest.local***“
- ***DocumentRoot*** – direktíva udávajúca cestu k súborom, ktoré ma Apache po zadaní danej URL odoslať prehliadaču. Hodnota za touto direktívou sa líši v závislosti na danom bloku.
    - V tomto prípade (úvodnej stránky) je daná cesta „***/var/www/html***“, v prípade *devuseonly* honeypotu by bola „***/var/www/html/devuseonly***“ a v prípade *moodle* honeypotu potom „***/var/www/html/moodle***“

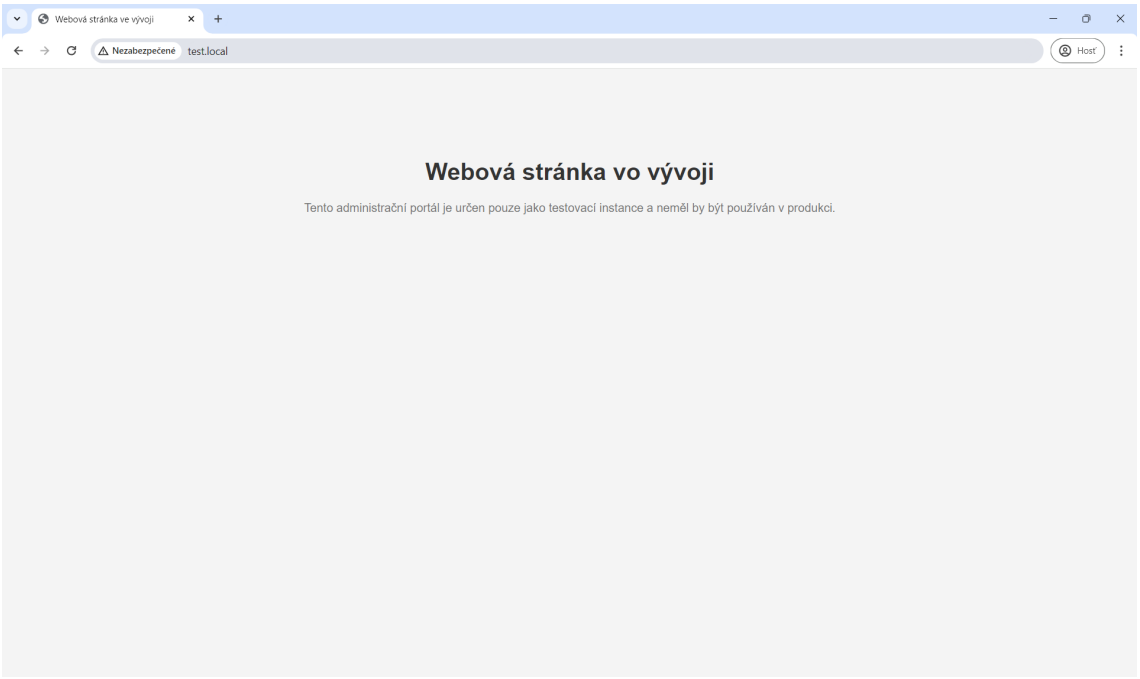
- ***ErrorLog ${APACHE_LOG_DIR}/error.log*** – automaticky generovaný riadok zhodný pre všetky bloky, udávajúci, kde sa nachádza logovací súbor chýb nášho Apache2 webového servera,
- ***CustomLog ${APACHE_LOG_DIR}/access.log combined*** – automaticky generovaný riadok zhodný pre všetky bloky, udávajúci, kde sa nachádza logovací súbor prístupu k nášmu Apache2 webovému serveru,
- ***</VirtualHost>*** - koniec bloku konfigurácie virtuálneho webového servera, zhodný pre všetky bloky.

```
<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    ServerName test.local

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
</VirtualHost>
```

Podotýkame, že fungovanie týchto URL v testovacej prevádzke je podmienené úpravou DNS súboru zariadenia pomocou, ktorého na dané URL pristupujeme. V našom prípade teda hostiteľského zariadenia, na ktorom sa virtuálne systémy nachádzajú. DNS záznam teda nie je upravený priamo na daných VS.

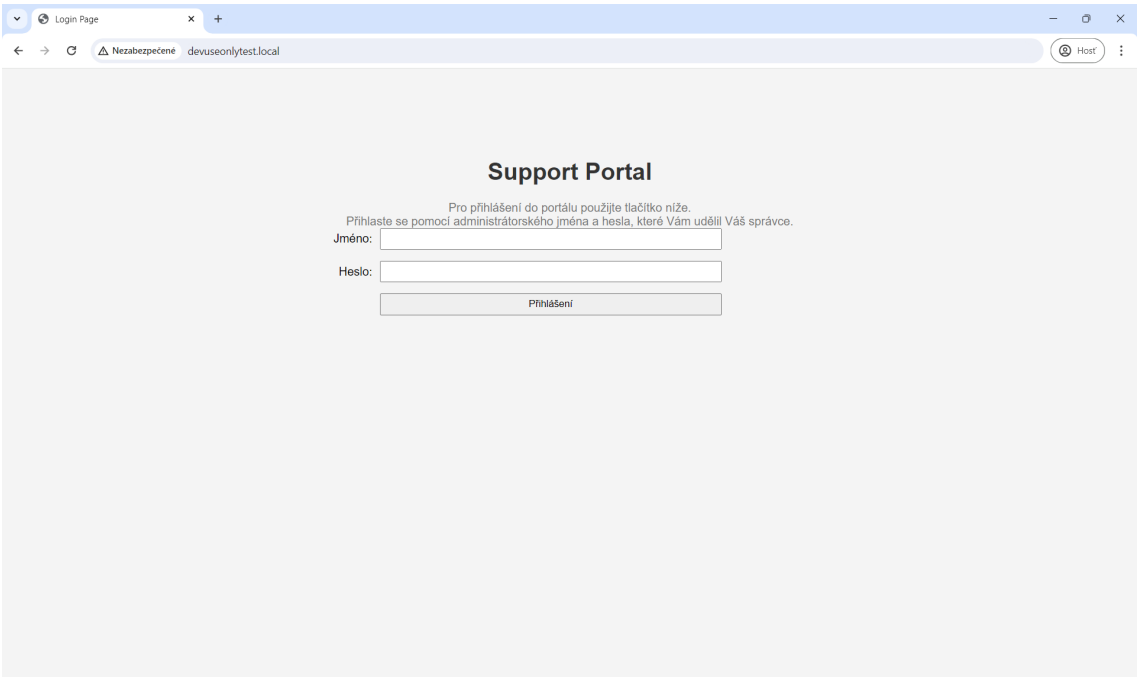
7.1.1.2 Súbory webového servera

V tomto bode bol náš webový server funkčný a prešiel si základnou konfiguráciou. Ďalším krokom bolo pridanie samotných webových stránok. Pokračovali sme teda podľa návrhu na adrese zdroja [44] a do práce pridali prvý HTML kód, slúžiaci ako „úvodná stránka“ informujúca používateľa, že daný portál je vo vývoji a neslúži na produkčné účely. HTML kód stránky je čisto statický a neobsahuje žiadnu honeypot funkcionalitu. Kód pre túto úvodnú stránku je možné nájsť v **PRÍLOHA P I: OBSAH CD**, v systéme sa nachádza v adresári */var/www/html* a jej vzhľad zobrazuje **Obrázok 40**.

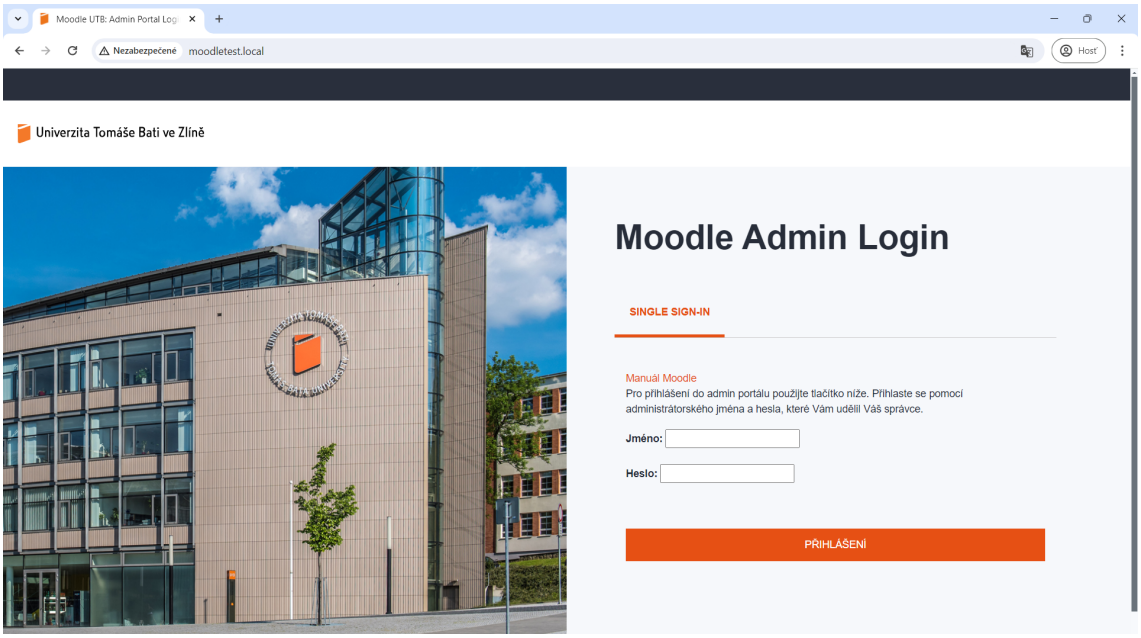


Obrázok 40 - WEB-POT - úvodná stránka, Zdroj: vlastný

Nasledujúcim krokom bolo pridanie samotných webových stránok, resp. teda honeypotov. V adresári */var/www/html* sme vytvorili dva priečinky devuseonly a moodle. Každý z týchto priečinkov obsahuje jeden PHP kód predstavujúci stránku samotnú a PHP skript nazvaný ***post.php*** slúžiaci na spracovanie (zber) vstupov od útočníkov, ich prvotné spracovanie a uloženie do logov. PHP kódy predstavujúce dané webové stránky sú dostupné v **PRÍLOHA P I: OBSAH CD**. V prípade portálu *moodle* sa jedná o upravenú kópiu zdrojového kódu originálneho portálu Moodle UTB. Portál *devuseonly* predstavuje falošný portál slúžiaci pre prihlásenie vývojárov (viz. **Obrázok 41**) a portál *moodle* predstavuje falošné administrátorské rozhranie Moodle UTB (viz. **Obrázok 42**).



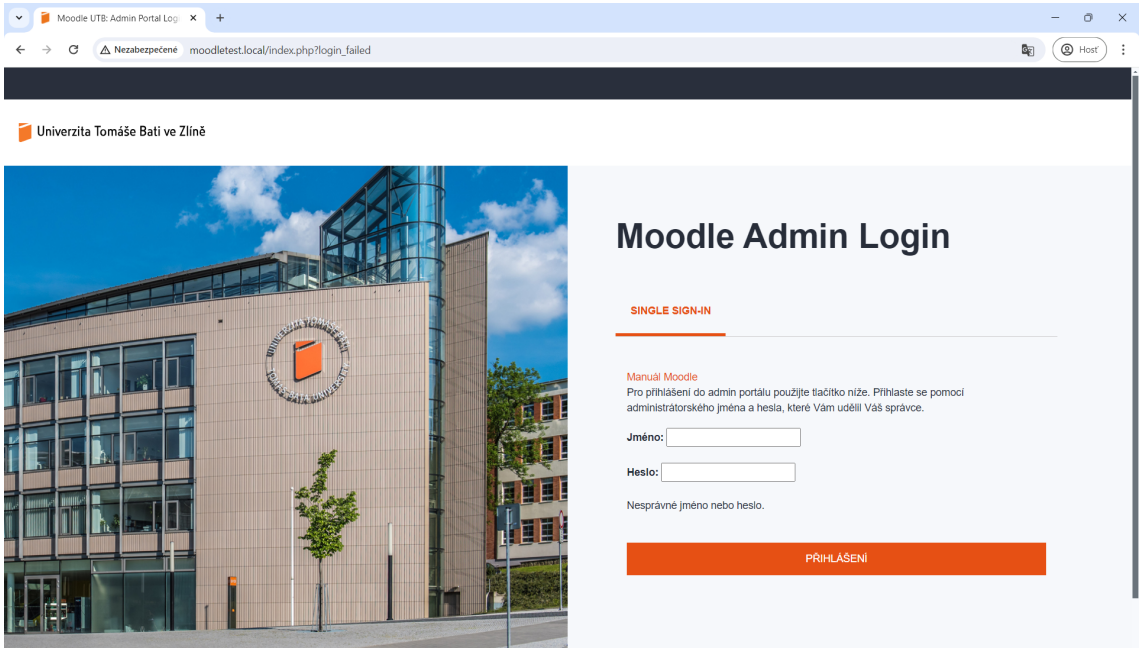
Obrázok 41 - WEB-POT - Devuseonly portál, Zdroj: vlastný



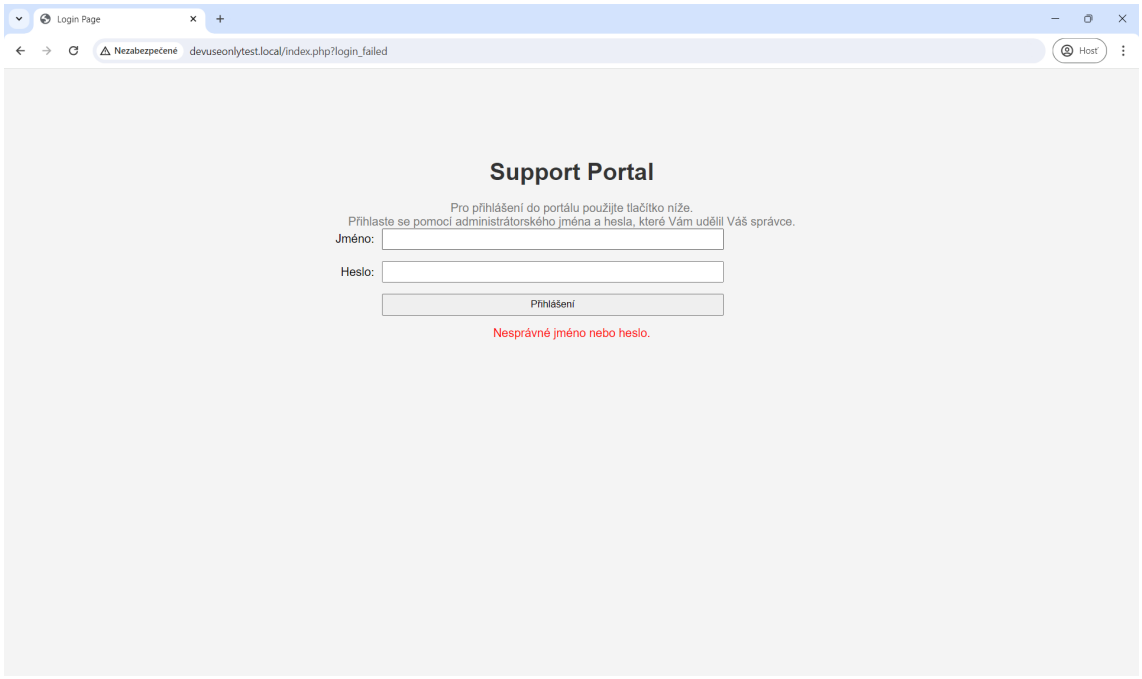
Obrázok 42 - WEB-POT - Falošný Moodle portál, Zdroj: vlastný

Diskutabilne najpodstatnejšou časťou oboch portálov je prihlasovací formulár, ktorého hlavnou úlohou je prijatie prihlasovacích údajov od užívateľa a predanie týchto údajov *post.php* skriptu, ktorého funkcionalitu rozoberieme v ďalšom odseku a zobrazenie chybovej hlášky o tom, že užívateľ zadal nesprávnu kombináciu mena a hesla – pričom,

samozrejme, správna kombinácia neexistuje (viz. **Obrázok 43** a **Obrázok 44**). Kód *post.php* skriptu je pre oba portály totožný.



Obrázok 43 - WEB-POT - Chybová hláška Moodle portálu, Zdroj: vlastný



Obrázok 44 - WEB-POT - Chybová hláška Devuseonly portálu, Zdroj: vlastný

7.1.1.3 Skript – post.php

Poslednou časťou, ktorá v tomto bode v práci našim webovým honeypotom chýba je samotný *post.php* skript, ktorý si v nasledujúcich odsekoch rozoberieme celý v sekvenčnom poradí (pre celú konfiguráciu viz. **PRÍLOHA P I: OBSAH CD**). Prvá časť je zobrazená nižšie a jej úlohou je pozastaviť vykonávanie skriptu po dobu dvoch sekúnd, aby sme útočníkovi simulovali overovanie zadaného mena a hesla voči databáze hesiel, ktorá v skutočnosti pochopiteľne neexistuje. Následne do premennej ***ip*** uložíme hodnotu ***$_SERVER['REMOTE_ADDR']***, a ak je ***$_SERVER['HTTP_X_FORWARDED_FOR']*** alebo ***$_SERVER['HTTP_CLIENT_IP']*** dostupná a validna, prepíšeme a uložíme príslušnú hodnotu. Takto by sme s istotou mali zaistiť IP Adresu klienta, resp. útočníka.

```
// Sleep for two seconds to simulate validation
sleep(2);

// Capture client info
$ip = $_SERVER['REMOTE_ADDR'];
if (isset($_SERVER['HTTP_X_FORWARDED_FOR']) &&
filter_var($_SERVER['HTTP_X_FORWARDED_FOR'], FILTER_VALIDATE_IP)) {
    $ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
} elseif (isset($_SERVER['HTTP_CLIENT_IP']) &&
filter_var($_SERVER['HTTP_CLIENT_IP'], FILTER_VALIDATE_IP)) {
    $ip = $_SERVER['HTTP_CLIENT_IP'];
}
```

Ďalší blok kódu predstavuje funkciu, ktorá je použitá ďalej v kóde a jej úlohou je spracovať výstup z geolokačného príkazu a upraviť ho do nami žiadanej podoby. Pre predstavu GeoLite2 databáza by nám vrátila pre krajinu výstup vo formáte:

- "United States" <utf8_string>

Nami požadovaný výstup vyzerá nasledovne:

- United States

Buď blok *if* vyhľadá text medzi prvou a druhou úvodzovkou – čo je vlastne nami požadovaný výstup. V prípade zemepisnej dĺžky a šírky ale vracia GeoLite2 príkaz hodnotu bez úvodzoviek – pre náš príklad s krajinou by sme teda dostali:

- United States <utf8_string>

Alebo blok *else* rozdelí výstup na dve časti – pred a za medzerou, teda dostaneme:

- United States

- <utf8_string>

A vráti prvú (index nula) časť, čo je opäť nami požadovaný výstup.

```
function extractGeoIPValue($lookupOutput) {
    // Check if the output contains a quoted string
    if (strpos($lookupOutput, '"') !== false) {
        // Find the position of the first double quote
        $firstQuotePos = strpos($lookupOutput, '"');

        // Find the position of the second double quote
        $secondQuotePos = strpos($lookupOutput, '"', $firstQuotePos + 1);

        // Extract the substring between the quotes
        return substr($lookupOutput, $firstQuotePos + 1, $secondQuotePos -
$firstQuotePos - 1);
    } else {
        // For non-quoted output (like numeric values), extract the first word
        // This assumes the value is the first word in the line
        $outputParts = explode(" ", trim($lookupOutput));
        return $outputParts[0];
    }
}
```

Nasledujúce riadky vykonávajú samotné geolokačné vyhľadávanie (v poradí: krajina, mesto, vyšší územný celok, kontinent, zemepisná šírka a dĺžka) pomocou príkazov a ukladajú výsledky príkazov do príslušných premenných so zakončením ***RAW*.** Následne na každú hodnotu označenú ***RAW*** aplikujú na vyššie definovanú funkciu, pričom výslednú (finálnu) hodnotu ukladajú do príslušnej rovnomennej premennej – bez označenia ***RAW***.

```
// GeoIP Lookup
$countryLookupRAW = shell_exec("mmdblookup --file /usr/share/GeoIP/GeoLite2-
Country.mmdb --ip $ip_geo country names en");
$countryLookup = extractGeoIPValue($countryLookupRAW);

$cityLookupRAW = shell_exec("mmdblookup --file /usr/share/GeoIP/GeoLite2-
City.mmdb --ip $ip_geo city names en");
$cityLookup = extractGeoIPValue($cityLookupRAW);

$subdivisionLookupRAW = shell_exec("mmdblookup --file /usr/share/GeoIP/GeoLite2-
City.mmdb --ip $ip_geo subdivisions 0 names en");
$subdivisionLookup = extractGeoIPValue($subdivisionLookupRAW);

$continentLookupRAW = shell_exec("mmdblookup --file /usr/share/GeoIP/GeoLite2-
City.mmdb --ip $ip_geo continent names en");
$continentLookup = extractGeoIPValue($continentLookupRAW);

$latitudeLookupRAW = shell_exec("mmdblookup --file /usr/share/GeoIP/GeoLite2-
City.mmdb --ip $ip_geo location latitude");
```

```
$latitudeLookup = extractGeoIPValue($latitudeLookupRAW);

$longitudeLookupRAW = shell_exec("mmdblookup --file /usr/share/GeoIP/GeoLite2-
City.mmdb --ip $ip_geo location longitude");
$longitudeLookup = extractGeoIPValue($longitudeLookupRAW);
```

Ďalej pomocou riadku nižšie zaznamenáme a uložíme URL na ktorú útočník útočil:

```
// Capture the referring URL
$referrer = isset($_SERVER['HTTP_REFERER']) ? $_SERVER['HTTP_REFERER'] : 'No
referrer';
```

Následne skript zaznamená a uloží typ a verziu webového prehliadača útočníka (premenná ***$user_agent***) a prihlasovacie údaje, ktoré útočník zadal (premenné ***$username*** a ***$password***). Pričom limituje dĺžky záznamov na 500 resp. 100 znakov.

```
// Capture form data
$user_agent = substr($_SERVER['HTTP_USER_AGENT'], 0, 500);
$username = substr($_POST["username"] ?? '', 0, 100);
$password = substr($_POST["password"] ?? '', 0, 100);
```

Pokračujeme pomocou overenia, že hodnoty pre meno (***$username***) a heslo nie sú prázdne (***$password***) – čo nepredpokladáme, vzhľadom k tomu, že sú v HTML kóde uvedené ako povinné polia, ale, aj napriek tomu, overujeme. V prípade, že ani meno, ani heslo nie je prázdne, uložíme do premennej ***$timestamp*** aktuálny dátum a čas a do premennej ***$entry*** riadok pozostávajúci z hodnôt, ktoré sme doteraz získali a ktorý chceme zapísať do logovacieho súboru. Následne načítame cestu k logovaciemu súboru do premennej ***$entry*** (súbor sa teda nachádza v */var/www/logons.txt*) a uložíme do neho predpripravený riadok z premennej ***$entry*.**

```
// Check if both username and password are not empty
if (!empty($username) || !empty($password)) {
    // Prepare the log entry
    $timestamp = date('Y-m-d H:i:s');
    $entry = "$timestamp | Username: $username | Password: $password | IP: $ip |
Agent: $user_agent | Referrer: $referrer | Country: $countryLookup | City:
$cityLookup | Subdivision: $subdivisionLookup | Continent: $continentLookup |
Latitude: $latitudeLookup | Longitude: $longitudeLookup\r\n";

    // Append to the entry to the log file
    $file = '/var/www/logons.txt';
    file_put_contents($file, $entry, FILE_APPEND);
}
```

Nasleduje presmerovanie používateľa späť na prihlasovací portál s pozmeneným URL. Kód zabezpečujúci presmerovanie vyzerá nasledovne:

```
// Construct the redirect URL dynamically
$redirect_url = 'http://' . $_SERVER['HTTP_HOST'] . '/index.php?login_failed';

// Redirect the client to the constructed URL
header("Location: $redirect_url");
exit;
?>
```

Princípom je presmerovanie útočníka späť na prihlasovací portál s upravenou hodnotou URL - s pridanou premennou ***login_failed***. Pomocou ***$_SERVER['HTTP_HOST']*** získame aktuálnu URL, na ktorú útočník pristupoval a pridáme k nej „***/index.php?login_failed***“, čím získame finálnu URL na presmerovanie. Túto premennú potom blok kódu zobrazený nižšie, v PHP kóde samotných portálov, rozozná a na základe jej prítomnosti zobrazí chybovú hlášku.

```
if (isset($_GET['login_failed'])) {
    echo '<div class="error-message">Nesprávné jméno nebo heslo.</div>';
}
```

7.1.2 WireGuard

Ako môžeme vidieť na konfigurácii nižšie, konfigurácia klienta je veľmi podobná serverovej konfigurácii z **Podkapitoly 6.1.1**. Blok konfigurácie rozhrania klienta sa opäť začína kľúčovým slovom [Interface] a končí sa pred ďalším kľúčovým slovom [Peer], kde začína blok konfigurácie servera, ku ktorému sa bude klient pomocou VPN tunela pripájať – v našom prípade sa teda jedná o IP adresu WireGuard rozhrania virtuálneho systému ELK. Blok konfigurácie rozhrania klienta teda obsahuje tieto riadky:

- ***Address = 10.0.0.2/24*** – definícia IP adresy WireGuard rozhrania klienta,
- ***SaveConfig = true*** – inštrukcia pre WireGuard o tom, že daná konfigurácia je perzistentná, teda ju chceme uložiť a používať aj medzi reštartmi celého systému, či WireGuard servera a taktiež, že chceme uložiť zmeny vykonané v konfigurácii aj za behu rozhrania,
- ***ListenPort*** – definuje port, na ktorom WireGuard klient prijíma požiadavky na komunikáciu od servera, prípadne iných klientov. V našom prípade nie, keďže jediné nakonfigurované spojenie, resp. [Peer] je práve náš WireGuard server na virtuálnom systéme ELK,

- ***FwMark = 0xca6c*** – automaticky generované firewall označenie (z angl. *Firewall Mark*) paketu, slúžiace na identifikovanie konkrétnych paketov a smerovanie,
- ***PrivateKey*** – privátny kľúč daného klienta, slúžiaci na zašifrovanie odchádzajúcej komunikácie, resp. logov.

Konfigurácia ďalej obsahuje blok začínajúci kľúčovým slovom [Peer], kde definujeme s kým bude náš klient komunikovať – ako bolo uvedené, v našom prípade sa jedná o VS ELK, resp. Logstash inštanciu na tomto systéme. Tento blok pozostáva z nasledujúcich riadkov:

- ***PublicKey = TMcCr2snEILgx06a5Cy/6D0+Jz2uH0DT/PsQ/iC+zA8=*** – obsahuje verejný kľúč pre WireGuard rozhranie virtuálneho systému ELK, potrebný na dešifrovanie komunikácie, ktorú server zašifroval svojím privátnym kľúčom a odoslal prostredníctvom tunela,
- ***AllowedIPs = 10.0.0.1/32*** – IP adresa WireGuard rozhrania na komunikáciu, pomocou ktorého je tento kľúč použiteľný. Logicky sa teda jedná o adresu WireGuard servera na virtuálnom systéme ELK.
- ***Endpoint*** – IP adresa virtuálneho systému ELK v danej sieti a port, na ktorom očakáva a odpovedá na prichádzajúce VPN spojenia. V našom prípade sa opäť jedná o internú IP adresu, špecifickú pre našu testovaciu infraštruktúru.
- ***PersistentKeepalive = 30*** – predstavuje interval v sekundách, po ktorých bude klient na danom rozhraní automaticky odosielať tzv. „keepalive“ pakety, aby udržal perzistentné spojenie so serverom, aj keď cez VPN tunel neprechádzajú žiadne iné dáta.

```
[Interface]
Address = 10.0.0.2/24
SaveConfig = true
ListenPort = 43765
FwMark = 0xca6c
PrivateKey = (Skryté pre potreby tlače)

[Peer]
PublicKey = TMcCr2snEILgx06a5Cy/6D0+Jz2uH0DT/PsQ/iC+zA8=
AllowedIPs = 10.0.0.1/32
Endpoint = 192.168.100.43:51820
PersistentKeepalive = 30
```

Po spustení a otestovaní bolo posledným krokom nastavenie automatického spustenia WireGuard rozhrania *wg0* pomocou príkazu „***sudo systemctl enable wg-quick@wg0***“.

Samotný konfiguračný súbor sa vo virtuálnom systéme WEB-POT nachádza v ***/etc/wireguard/wg0.conf***

7.1.3 FileBeat

Vzhľadom k tomu, že FileBeat je konfiguračne pomerne jednoduchý protokol a veľkú časť konfigurácie si generuje sám, pričom od používateľa vyžaduje len zopár konfiguračných zmien, zameriame sa práve na tieto zmeny. Pre celú konfiguráciu viz. **PRÍLOHA P I: OBSAH CD**. Veľmi dôležitou časťou konfigurácie FileBeat je časť vstupov (z angl. *Inputs*), v ktorej konfigurujeme vstupy, resp. súbory logov, ktoré má FileBeat sledovať na prípadné zmeny a odosielať – viac v časti výstupov (z angl. *Output*). Časť vstupov zobrazuje konfigurácie nižšie a pozostáva teda z týchto riadkov:

- ***filebeat.inputs:*** - označuje kde sa začína sekcia definície vstupov,
- ***- type: filestream*** - určuje typ vstupu, v tomto prípade sa jedná o filestream, ktorý slúži na zbieranie logov zo súborov,
- ***id: logons -*** jedinečný identifikátor tohto vstupu medzi všetkými ostatnými vstupmi. Identifikátor je vyžadovaný pre každý vstup, v našom prípade máme vstup len jeden, takže sa vyžaduje len jeden jedinečný identifikátor.
- ***enabled: true –*** hodnota určujúca, či má byť táto konfigurácia vnímaná ako aktívna a má byť použitá. Zmena hodnoty z true na false, by teda logicky spôsobila „deaktiváciu“ konfigurácie tohto vstupu a FileBeat by ju ignoroval.
- ***paths:*** - definuje začiatok odseku v konfigurácii, v ktorom udávame, kde sa nachádzajú súbory (logy), ktoré má FileBeat sledovať,
    - ***– /var/www/logons.txt*** – cesta k súboru, ktorý má FileBeat sledovať. Jedná sa teda o textový (.txt) súbor s názvom logons, nachádzajúci sa v priečinku /var/www
- ***fields:*** - definuje začiatok odseku slúžiaceho na pridanie dodatočných polí (z angl. *filed*) s informáciami. Využívajú sa teda najmä na pridávanie metadát, kategorizáciu, jednoznačnú identifikáciu a podobne,
    - ***type:*** “webpot” – definícia poľa s informáciami vo formáte <kľúč>:<hodnota>. Hovoríme teda, že chceme pridať pole s názvom *type*, s hodnotou “*webpot*” (pochopiteľne bez úvodzoviek). Toto pole slúži Logstash inštancii na virtuálnom systéme ELK pre jednoznačnú identifikáciu logov a ich adekvátne spracovanie, resp. aplikovanie príslušného filtra.

Podmienené aplikovanie filtrov na základe premennej *type* sme už spomínali v **Podkapitole 6.1.2.3**, kde sme vysvetľovali konfiguráciu Logstash.

- ***fields_under_root: true*** – hovoríme, že premenná *type* má byť vložená priamo do koreňa každého odosielaného logu, aby k nej bolo možné pristupovať priamo, bez nutnosti definovať dlhšiu cestu (tzv. zanorenie).

```
# ============================== Filebeat inputs ===============================

filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input specific configurations.

# filestream is an input for collecting log messages from files.
- type: filestream

  # Unique ID among all inputs, an ID is required.
  id: logons

  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /var/www/logons.txt
    #- c:\programdata\elasticsearch\logs\*

  fields:
    type: "webpot"
  fields_under_root: true
```

Sekcia výstupov pozostáva z dvoch pomyselných častí, z ktorých je nutné si vybrať na základe toho, ako plánujeme FileBeat protokol používať, resp. teda kam budeme logy odosielať. V našom prípade budeme logy odosielať do Logstash inštancie na virtuálnom systéme ELK, a teda volíme časť konfigurácie označenú ako „*Logstash Output*“, ktorú môžeme vidieť nižšie. V tejto časti musíme vykonať len veľmi jednoduchú konfiguračnú zmenu na dvoch riadkoch:

- ***output.logstash:*** - definujeme začiatok odseku konfigurácie výstupu do danej inštancie Logstash,
  - ***hosts: [“10.0.0.1:5044”]*** – definujeme na akej IP adrese sa daná inštancia Logstash nachádza a na akom porte očakáva prichádzajúci komunikáciu,

resp. logy. Ako môžeme vidieť, v našom prípade sa teda jedná o adresu serverového WireGuard rozhrania virtuálneho systému ELK a predvolený port projektu Logstash. Odosielanie logov teda prebieha pomocou zabezpečeného VPN tunela, ako sme to už niekoľkokrát uvideli v predchádzajúcich kapitolách.

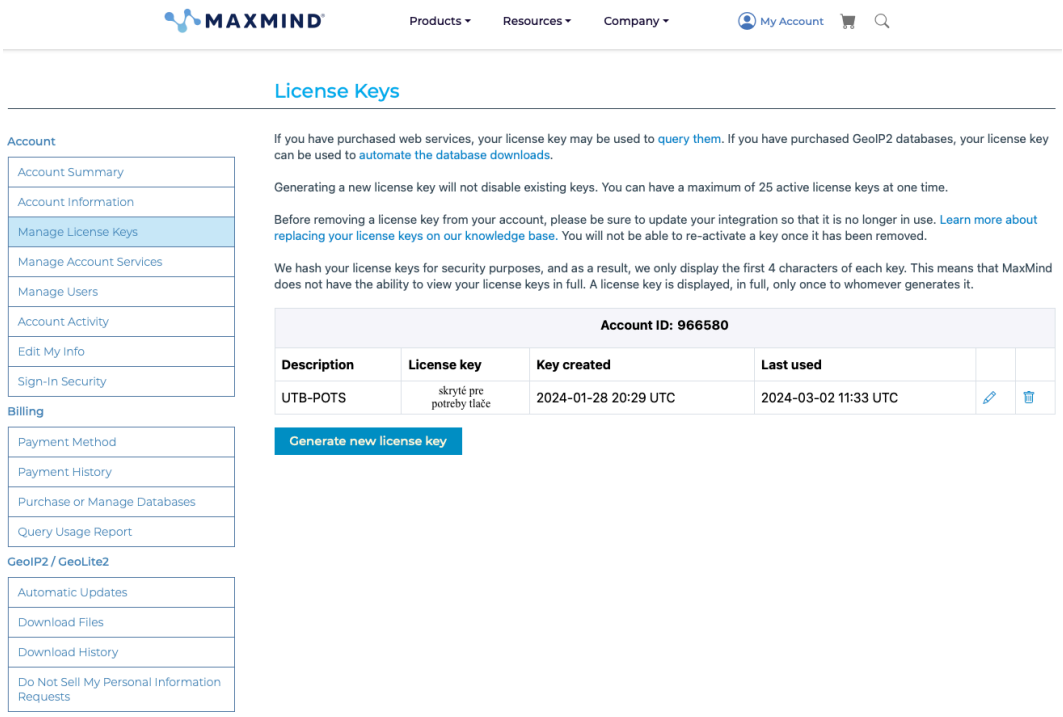
```
# ------------------------------ Logstash Output -------------------------------
output.logstash:
  # The Logstash hosts
  hosts: ["10.0.0.1:5044"]
```

Po spustení a otestovaní bolo posledným krokom nastavenie automatického spustenia pomocou príkazu „***sudo systemctl enable filebeat***".

7.1.4 GeoLite2

Predtým, než sme mohli použiť GeoLite2 databázy v našom *post.php* skripte, z **Podkapitoly 7.1.1**, sme museli vykonať niekoľko krokov:

1. Prvým krokom bola registrácia a vygenerovanie licenčného kľúča na stránkach MaxMind – viz. **Obrázok 45** nižšie. Hodnota licenčného kľúča a ID účtu (*AccountID*) je potrebná v ďalších krokoch, a preto je potrebné si ju zaznamenať.



Obrázok 45 - GeoLite2 generovanie licenčného kľúča, Zdroj: vlastný

2. Po vygenerovaní licenčného kľúča a inštalácii balíka do operačného systému Ubuntu je potrebné upraviť automaticky generovaný konfiguračný súbor pre GeoLite2, ktorý sa nachádza v ***/etc/GeoIP.conf*** – viz. konfigurácie nižšie. Úprava spočíva vo svojej podstate len v zadaní ID účtu, ku ktorému je licenčný kľúč viazaný a ID samotného, preto konfigurácia nižšie zobrazuje len túto časť. Pre celú konfiguráciu viz. **PRÍLOHA P I: OBSAH CD**. Nami editovaná časť konfigurácie teda pozostáva z dvoch riadkov *AccountID* – hodnota ID pre daný účet a *LicenseKey* – hodnota licenčného kľúča.

```
# Replace YOUR_ACCOUNT_ID_HERE and YOUR_LICENSE_KEY_HERE with an
activeaccount
# ID and license key combination associated with your MaxMind account.
These
# are available from https://www.maxmind.com/en/my_license_key.
AccountID 966580
LicenseKey (Skryté pre potreby tlače)
```

3. V neposlednom rade bolo potrebné do systému doinštalovať aj balík ***mmdb-bin***, ktorý systému umožňuje správne čítať a používať GeoLite2 databázové súbory pomocou príkazov, tak ako to robíme v už spomínanom skripte *post.php* z **Podkapitoly 7.1.1**.

7.1.4.1 Automatická aktualizácia GeoLite2 databáz

Aby sme zabezpečili automatickú aktualizáciu GeoLite2 databáz, vytvorili sme jednoduchý skript s názvom ***run_geoipupdate*** nachádzajúci sa v ***/home/utb***. Obsah tohto skriptu je zobrazený nižšie a obsahuje tieto riadky:

- ***GEOIP_UPDATE_CMD=“/usr/bin/geoipupdate“*** – tento riadok do premennej *GEOIP_UPDATE_CMD* ukladá cestu k aktualizačnému skriptu GeoLite2,
- ***$GEOIP_UPDATE_CMD*** – tento riadok spúšťa skript uložený v danej premennej, resp. teda spúšťa aktualizáciu GeoLite2 databáz.

Nasledujúce zakomentované riadky (riadky začínajúce znakom „#”) predstavujúce blok *if,* slúžili v procese testovania na overenie úspešného priebehu aktualizácie, keďže presmerujú návratovú hodnotu posledného vykonaného príkazu na štandardný výstup. Ak je hodnota rovná 0, príkaz prebehol úspešne, ak nie je rovná 0, v procese aktualizácie nastala chyba. Tento blok kódu sme v skripte ponechali pre budúce použitie.

```
#!/bin/bash
```

```
# Path to the geoipupdate command
GEOIP_UPDATE_CMD="/usr/bin/geoipupdate"

# Run the geoipupdate command
$GEOIP_UPDATE_CMD

# Check if geoipupdate command was successful
#if [[ $? -eq 0 ]]; then
#    echo "GeoIP2 database updated successfully."
#else
#    echo "Failed to update GeoIP2 database."
#fi
```

Posledným krokom bolo nastavenie periodického spustenia aktualizačného skriptu. Tento stav sme docielili pridaním záznamu (riadku) do tzv. *cron table* správcu:

- ***0 3 * * 7 /home/utb/run_geoipupdate >> /home/utb/Desktop/update_log.txt 2>&1*** – riadok môžeme pre lepšie pochopenie rozdeliť na niekoľko častí:
  - ***0 3 * * 7*** – predstavuje definíciu toho, kedy má systém vykonať to, čo užívateľ definoval na riadku ďalej. V tomto prípade teda hovoríme, že chceme daný riadok vykonať nultú minútu – 0, tretiu hodinu – 3, ktorýkoľvek deň v mesiaci - *, ktorýkoľvek mesiac - * a siedmy deň v týždni – 7. Príkaz sa teda spustí každú nedeľu o 3 ráno.
  - ***/home/utb/run_geoipupdate*** – definujeme skript na vykonanie,
  - **>>** – presmerovanie výstupu skriptu,
  - ***/home/utb/Desktop/update_log.txt*** – cieľ presmerovania (textový súbor, vytvorený na ploche)
  - ***2>&1*** – presmerovanie chybových hlášok na štandardný výstup.

V prípade, že testovací *if* blok spomínaný vyššie nie je odkomentovaný, do súboru sa neuloží nič, v prípade, že je odkomentovaný, uložíme do neho záznam tak, ako to zobrazuje **Obrázok 46**.

```
1 GeoIP2 database updated successfully.
```

Obrázok 46 - GeoLite2 úspešná aktualizácia, Zdroj: vlastný

8 PYRDP

VS PyRDP je druhým honeypotom, na ktorý sa v tejto práci pozrieme. Ako napovedá už samotný názov, jedná sa o honeypot zameraný na protokol vzdialeného prístupu - RDP (viz. **Podkapitola 1.6.3**). Finálna implementácia teda pozostáva z niekoľkých častí, ktoré v tejto kapitole rozoberieme. PyRDP zdieľa niektoré spoločné súčasti s VS WEB-POT, ktorý sme rozoberali v predchádzajúcej **Kapitole 7**. V prípade zhodných implementácii niektorých súčastí sa na ne preto budeme odkazovať. Rovnako sa budeme zaoberať len súčasťami, ktoré sú pre prácu relevantné alebo ktoré sme do systému pridali, či nejako upravili pre potreby práce, navrhovaného riešenia, či cieľovej infraštruktúry.

8.1 Implementácia

VS PyRDP, tak ako ho ilustruje **Obrázok 36** a tak ako sme ho v skratke predstavili v **Podkapitole 5.2**, pozostáva z týchto komponent:

- ***pyrdp projekt*** – VS PyRDP implementuje rovnomenný projekt *pyrdp* od spoločnosti GoSecure (viz. adresa zdroja [42]), ktorý vo svojej podstate umožňuje obrancom vykonať MITM (Man-in-the-middle) útok proti útočníkom. Podstatou je nasmerovanie pyrdp servera na zariadenie, na ktoré je možné sa pripojiť pomocou RDP protokolu. Pyrdp server sa potom na sieti javí ako toto zariadenie a po pripojení útočníka na toto zariadenie naozaj presmeruje. Avšak vzhľadom k tomu, že útočník sa vlastne nepripája k cieľovému zariadeniu, ale k nášmu pyrdp serveru, sme schopní zachytávať veľkú časť priebehu útočníkovej činnosti – zadané prihlasovacie meno a heslo, stisnuté klávesy, súradnice kam útočník klikal. V prípade, že útočník k zariadeniu pripojí aj nejaké ďalšie pamäťové médium, začne pyrdp (ak je spustený v tzv. *crawl* móde) okamžite prechádzať a sťahovať obsah tohto média na náš pyrdp server. Po ukončení pripojenia ukladá pyrdp súbor obsahujúci záznam daného pripojenia, ktorý je možné prechádzať v textovej podobe, ale aj vo forme vizuálneho prehrateľného záznamu pomocou tzv. *pyrdp player. Pyrdp player* taktiež umožňuje vizuálne sledovať aktuálne prebiehajúce pripojenia a v originálnej implementácii aj prebrať kontrolu nad daným pripojením. Možnosť prebrať kontrolu nad pripojením útočníka je ofenzívnou funkcionalitou projektu, ktorú pre naše potreby v práci opomíname.

- ***PostFix*** – ako vieme z **Podkapitoly 4.9**, *PostFix* predstavuje open-source mailový server. V našej práci ho využívame na odosielanie mailových notifikácií so základnými informáciami pri novo vzniknutom pripojení na náš pyrdp server.
- ***WireGuard*** - vo virtuálnom systéme WEB-POT z predchádzajúcej kapitoly sme WireGuard konfigurovali do pozície klienta, ktorý zabezpečený WireGuard VPN tunel využíva na odosielanie logov o útokoch do virtuálneho systému ELK, resp. teda Logstash inštancie, ktorá sa na ňom nachádza. V prípade virtuálneho systému platí presne to isté, spolu s faktom, že konfigurácia VPN tunela je nastavená tak, aby jediná komunikácia (dáta), ktorá nim putuje bola komunikácia spojená s prenosom logov z virtuálneho systému PyRDP do virtuálneho systému ELK.
- ***FileBeat*** – rovnako ako v prípade virtuálneho systému WEB-POT z predchádzajúcej kapitoly FileBeat využívame na sledovanie logov, resp. ich zmeny a zasielanie týchto inkrementov do virtuálneho systému ELK na ďalšie spracovanie a vizualizáciu.
- ***GeoLite2*** – rovnako ako v prípade virtuálneho systému WEB-POT z predchádzajúcej kapitoly využívame lokálnu inštaláciu voľne dostupnej verzie GeoIP2 databáz od spoločnosť MaxMind – pričom proces prihlásenia na webových stránkach MaxMind a vygenerovania licenčného kľúča pre voľne dostupnú verziu sme už absolvovali a využívame už existujúcu kombináciu ID účtu a licenčného kľúča.

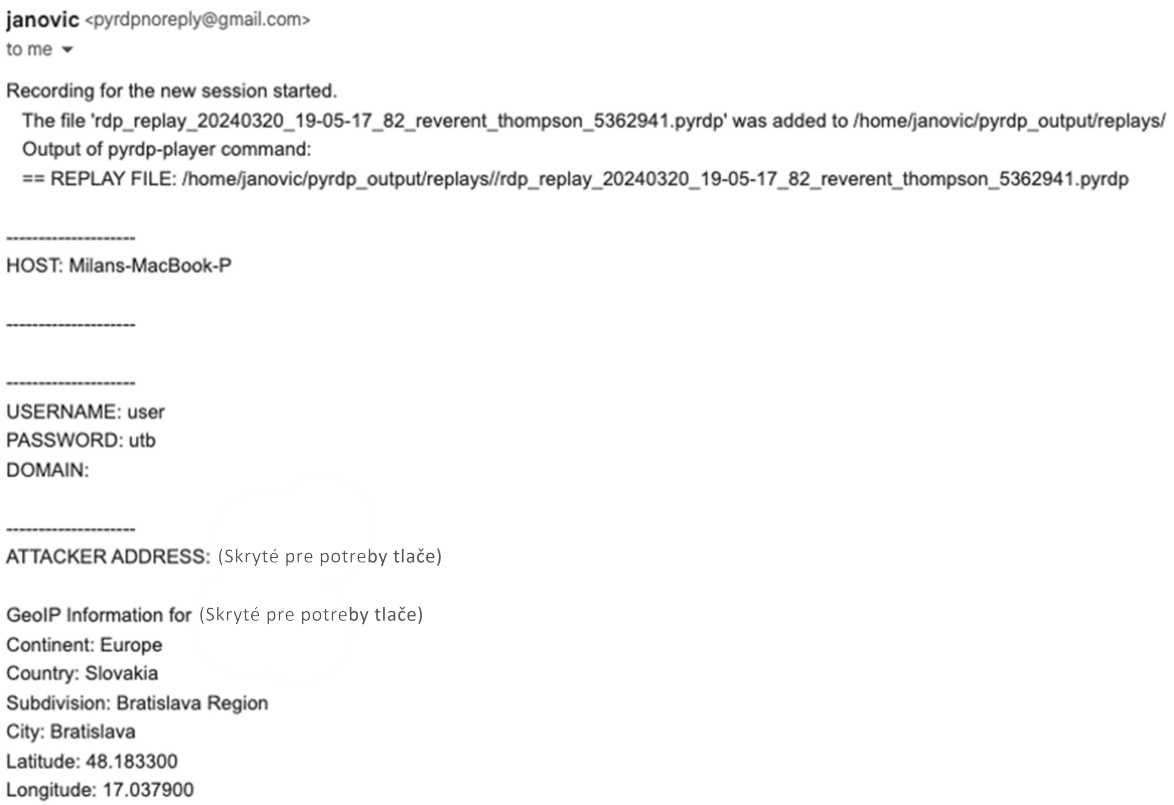
V nasledujúcich podkapitolách sa pozrieme na výsledky našej implementácie a ukážeme skripty a konfigurácie, ktorými sme ju docielili.

8.1.1 PostFix

PostFix mailový server v našej práci slúži na zasielanie mailových notifikácií, pri novo vytvorenom pripojení na náš pyrdp honeypot – viz. **Obrázok 47**. Ako môžeme vidieť zasielanie mailov je v práci na testovacie účely realizované pomocou emailovej adresy, ktorú sme vytvorili čisto na tento účel – ***pyrdpnoreply@gmail.com***, pričom mailové notifikácie zasielame na tú istú mailovú adresu, teda sami-sebe. Mailová notifikácia obsahuje niekoľko kľúčových informácií, ktoré je možné získať takmer okamžite po pripojení útočníka k nášmu pyrdp honeypotu. Medzi tieto informácie patrí:

- ***Názov a cesta k súboru so záznamom***
    - */home/janovic/pyrdp_output/replays/* - cesta do súboru obsahujúcom záznamy z pripojení,

  - */rdp_replay_20240320_19-05-17_82_reverent_thompson_5362941.pyrdp* – názov konkrétneho súboru so záznamom daného pripojenia, ku ktorému sa daný mail vzťahuje,
- ***HOST: Milans-MacBook-P*** – názov zariadenia útočníka,
- ***USERNAME: user –*** útočníkom zadané prihlasovacie meno, resp. názov účtu, ktoré zadal pred nadviazaním spojenia. Pokiaľ túto hodnotu útočník zadá až po nadviazaní spojenia, táto hodnota bude prázdna.
- ***PASSWORD: utb –*** útočníkom zadané prihlasovacie heslo, ktoré zadal pred nadviazaním spojenia. Pokiaľ túto hodnotu útočník zadá až po nadviazaní spojenia, táto hodnota bude prázdna.
- ***DOMAIN:*** - doména zariadenia ku ktorému sa útočník prihlasuje.
- ***ATTACKER ADDRESS: 178.143.46.11 –*** verejná IP adresa útočníka
- ***Geolokačné informácie –*** získané pomocou GeoLite2 databáz (viz. **Podkapitola 8.1.5**)
  - *Continent: Europe* – Kontinent (v príklade Európa)
  - *Country: Slovakia* **–** Krajina (v príklade Slovensko)
  - *Subdivision: Bratislava Region* – Kraj (v príklade Bratislavský)
  - *City: Bratislava* **–** Mesto (v príklade Bratislava)
  - *Latitude: 48.183300* – Zemepisná šírka (v príklade 48.183300)
  - *Longitude: 17.037900* – Zemepisná dĺžka (v príklade 17.037900)



Obrázok 47 - PyRDP emailová notifikácia, Zdroj: vlastný

8.1.1.1 Konfigurácia

Samotná konfigurácia po inštalácii pomocou balíčkovacieho systému *apt* operačného systému Ubuntu vyžaduje niekoľko krokov. Prvý krokom je konfigurácia tzv. SASL (Simple Authentication and Security Layer), ktorá nám umožní zasielanie mailov pomocou predom generovaného kľúča pre náš emailový účet – proces tvorby daného kľúča bude objasnený v nasledujúcej **Podkapitole 8.1.1.2.**. Najskôr sme vytvorili súbor *sasl_passwd* v */etc/postfix/sasl* a pridali sme do neho záznam v tomto formáte:

```
[smtp.gmail.com]:587 pyrdpnoreply@gmail.com:<kľúč>
```

Štruktúra záznamu je pomerne jednoduchá:

- ***[smtp.gmail.com]*:587** – definujeme adresu a port SMTP servera, ku ktorému sa kombinácia emailovej adresy a kľúča, ktorá nasleduje po medzere, vzťahuje – oddeľujeme pomocou dvojbodky,
- ***pyrdpnoreply@gmail.com:<klúč>*** - mailová adresa a kľúč, ktorý sme k nej vygenerovali – oddeľujeme pomocou dvojbodky (hodnota kľúča je skrytá pre potreby tlače).

Následne použijeme príkaz *postmap* na náš *sasl_passwd* súbor. Tento príkaz vytvorí v rovnakej zložke rovnomenný súbor s príponou *.db*. Aby sme oba súbory lepšie ochránili, zmeníme vlastníka a práva k daným súborom pomocou príkazov:

- ***sudo chown root:root /etc/postfix/sasl/sasl_passwd /etc/postfix/sasl/sasl_passwd.db*** – definujeme, že vlastníkom oboch súborov je používateľ *root* a rovnomenná skupina,
- ***sudo chmod 0600 /etc/postfix/sasl/sasl_passwd /etc/postfix/sasl/sasl_passwd.****db* – definujeme, že len vlastník má právo na čítanie a zápis do súboru.

Následne je potrebné pozmeniť konfiguráciu samotného PostFix mailového servera nachádzajúcu sa v */etc/postfix/main.cf*, ktorá je z veľkej časti generovaná automaticky pri inštalácii. Nebudeme preto objasňovať celú konfiguráciu, ale len časti, ktoré sme po potreby práce upravili a/alebo pridali. Celá konfigurácia je dostupná v **PRÍLOHA P I: OBSAH CD**. V konfigurácii najskôr pozmeníme nasledujúci riadok, ktorým za direktívou ***relayhost*** definujeme adresu a port SMTP servera, ktorý má PostFix používať na odosielanie mailov – oddeľujeme pomocou dvojbodky :

```
relayhost = [smtp.gmail.com]:587
```

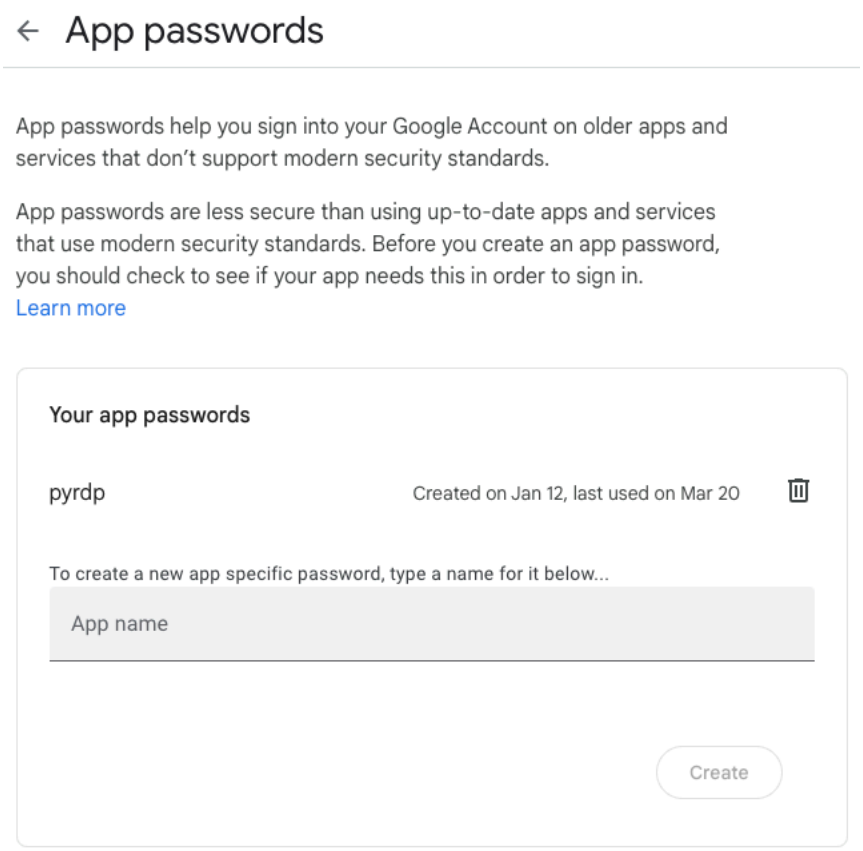
Následne, na koniec konfigurácie pridáme nasledujúce riadky, ktoré umožňujú autentizáciu pomocou SASL:

```
# Enable SASL authentication
smtp_sasl_auth_enable = yes
smtp_sasl_security_options = noanonymous
smtp_sasl_password_maps = hash:/etc/postfix/sasl/sasl_passwd
smtp_tls_security_level = encrypt
smtp_tls_CAfile = /etc/ssl/certs/ca-certificates.crt
```

Prvý riadok konfigurácie umožňuje použitie SASL autentizácie, druhý riadok potom zakazuje anonymnú autentizáciu – teda autentizáciu vždy vyžadujeme. Tretí riadok udáva cestu k súboru s kombináciami mien a hesiel, ktoré sa na autentizáciu majú použiť. Štvrtý riadok definuje, že odchádzajúce spojenie má byť šifrované a posledný, piaty riadok udáva cestu k adresám certifikačných autorít, ktoré operačný systém považuje za dôveryhodné a ktoré má PostFix použiť.

8.1.1.2 Gmail App Key

Gmail App Password, slúžiaci ako kľúč pre odosielanie mailov, bez zadania hesla k účtu samotnému, je možné vygenerovať v administrátorskom rozhraní daného mailového konta v časti *App password* – viz. **Obrázok 48**. Daný kľúč sa používateľovi zobrazuje len raz, po jeho vytvorení a je teda nutné si ho zaznamenať.



Obrázok 48 - PyRDP - App password, Zdroj: vlastný

8.1.2 Pyrdp projekt

8.1.2.1 Inštalácia a použitie

Inštalácia PyRDP projektu bola realizovaná tak, ako to uvádzajú autori na adrese zdroja [42], v sekcii *Installing*, pričom sa jedná o lokálnu inštaláciu (nie docker kontajner) pomocou nástroja *pipx*. Ako sme už uviedli v úvode **Podkapitoly 8.1** v odseku ***pyrdp projekt***, naša práca opomína ofenzívnu funkcionalitu originálnej implementácie, vzhľadom k tomu, že nie je pre potreby našej práce relevantná. Po inštalácii je možné projekt spustiť pomocou nasledujúceho príkazu:

- ***pyrdp-mitm –crawl 192.168.100.31***

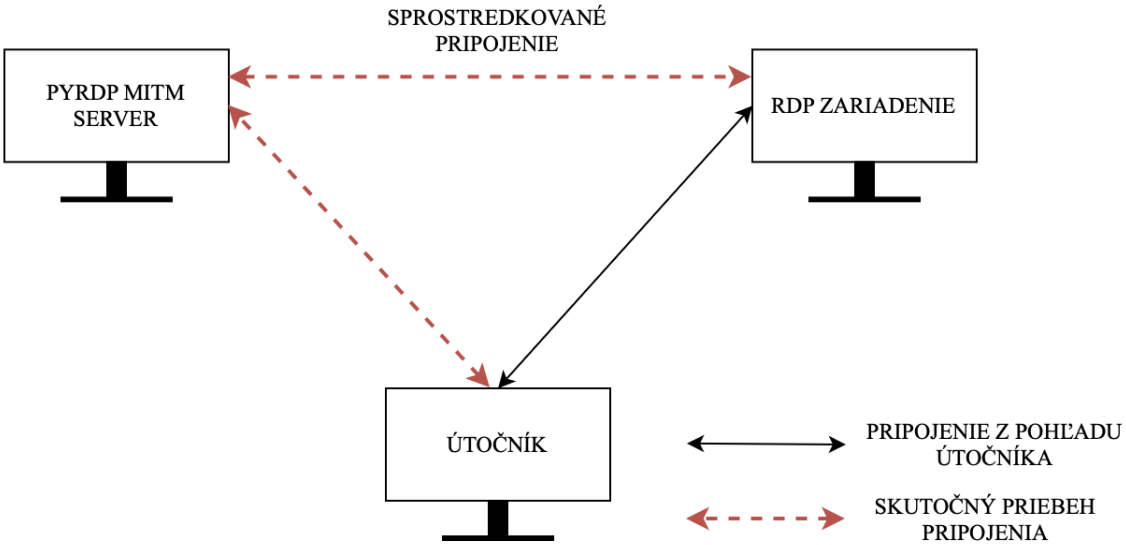
respektíve teda:

- ***pyrdp-mitm –crawl 192.168.100.31 -i 127.0.0.0***

Štruktúra príkazov je potom nasledovná:

- ***pyrdp-mitm*** – udávame akú funkcionalitu chceme spustiť, v tomto prípade spúšťame *pyrdp-mitm,* teda pyrdp MITM server na danom zariadení,
- ***--crawl*** – príznak, ktorým pyrdp serveru hovoríme, že ak k nemu útočník pripojí akékoľvek externé pamäťové médium, má pyrdp server začať prechádzať a sťahovať jeho obsah,
- ***192.168.100.31*** – IP adresa zariadenia s možnosťou pripojenia pomocou RDP protokolu, na ktoré sa má pyrdp server pripojiť, resp. za ktoré sa má na sieti vydávať,
- ***-i 127.0.0.0*** – príznak spolu s adresou *pyrdp-player* funkcionality v prípade, že ju chceme spustiť súbežne s pyrdp serverom. Toto je nie vyžadované, ak nepredpokladáme, že budeme chcieť čakať na pripojenie útočníka a sledovať jeho aktivitu v reálnom čase.

Pyrdp MITM server sa v tomto bode pripojí na definované zariadenie a začne sa zaň na sieti vydávať. Útočník sa potom v domnienke, že sa pripája na Windows zariadenie s otvoreným RDP protokolom vlastne pripája na náš pyrdp server, ktorý mu dané pripojenie sprostredkuje. Útočník sa teda v konečnom dôsledku naozaj na dané zariadenie pripojí, ale náš pyrdp server stojí v pomyselnom strede medzi útočníkom a daným zariadením (MITM) a je teda schopný zachytávať celý priebeh komunikácie. Tento stav zobrazuje jednoduchá schéma na **Obrázok 49** nižšie.



Obrázok 49 - Pyrdp MITM server - schéma pripojenia, Zdroj: vlastný

8.1.2.2 Skript - watcher.sh

Ďalším krokom bolo zabezpečenie ukladania relevantných informácií do logov a ich odosielanie pomocou mailovej notifikácie (viz. **Podkapitola 8.1.1**). Pyrdp server po spustení zobrazuje logy priamo do konzolového okna, z ktorého bol spustený a taktiež ich ukladá do */home/janovic/pyrdp_output/logs/mitm.log* – logy v tomto súbore sú nanešťastie, ale rozsiahle a obsahujú veľké množstvo informácií, ktoré nie sú pre potreby práce relevantné, napr. majú len informačný charakter a pod. Preto sme vytvorili skript s názvom ***watcher.sh***, ktorý je v systéme uložený v ***/home/janovic/pyrdp_output*** a jeho obsah (zakomentované časti začínajúce príkazom ***echo*** slúžia na výpis hodnôt počas testovania skriptu, v skripte sme ich ponechali pre potreby budúcich úprav, ale nebudeme ich v práci objasňovať – vzhľadom k tomu, že nepredstavujú funkcionalitu, ktorá je pre funkčnosť práce relevantná) si teraz rozoberieme po častiach, v sekvenčnom poradí. Prvý blok slúžiaci na definíciu premenných, ktoré budeme potrebovať v ďalších častiach skriptu, je zobrazený nižšie a obsahuje tieto riadky:

- ***DIRECTORY_TO_MONITOR="/home/janovic/pyrdp_output/replays"***– definujeme premennú *DIRECTORY_TO_MONITOR* s hodnotou */home/janovic/pyrdp_output/replays*. Táto premenná v sebe teda drží hodnotu, resp. cestu k súboru, ktorý budeme chcieť pomocou nášho skriptu monitorovať.

- ***EMAIL="pyrdpnoreply@gmail.com"*** – definujeme premennú *EMAIL* s hodnotou *pyrdpnoreply@gmail.com*. Táto premenná v sebe teda drží mailovú adresu, na ktorú budeme chcieť pomocou nášho skriptu odosielať notifikácie o nových pripojeniach.
- ***LOG_FILE="/home/janovic/pyrdp_output/logs/mitm.log"*** – definujeme premennú *LOG_FILE* s hodnotou /home/janovic/pyrdp_output/logs/mitm.log. Táto premenná v sebe teda drží hodnotu, resp. cestu k súboru, do ktorého pyrdp server ukladá logy a z ktorého budeme chcieť pomocou nášho skriptu extrahovať relevantné informácie.

```
DIRECTORY_TO_MONITOR="/home/janovic/pyrdp_output/replays"
EMAIL="pyrdpnoreply@gmail.com"
LOG_FILE="/home/janovic/pyrdp_output/logs/mitm.log"
```

Nasleduje pomerne dlhý blok, ktorý sa začína príkazom ***inotifywait*** (resp. ***do while***) a končí sa kľúčovým slovom ***done***, ako môžeme vidieť nižšie:

```
inotifywait -m -e create "$DIRECTORY_TO_MONITOR" | while read path action file;
do
        (zvyšok kódu)

done
```

Štruktúra zjednodušeného bloku je teda nasledujúca:

- ***inotifywait*** – nástroj, ktorý sleduje zmeny v súborovom systéme a generuje udalosti, keď sa tieto zmeny vyskytnú. Používa sa na monitorovanie zmenených súborov a priečinkov.
    - ***-m*** – príznak, ktorým hovoríme, že chceme adresár (ktorý definujeme neskôr) sledovať nepretržite,
    - ***-e create*** – príznak spolu s voľbou udalosti, ktorú chceme sledovať, teda udalosť vytvorenia nového súboru v danom adresári,
    - ***"$DIRECTORY_TO_MONITOR"*** – premenná obsahujúca cestu k súboru, ktorý chceme monitorovať,
    - | - presmerovanie výstupu aktuálneho príkazu ďalšiemu príkazu,
    - ***while read path action file*** – cyklus *while*, ktorý číta riadky výstupu prvého príkazu (*inotifywait*) a ukladá hodnoty do premenných ***path*** (cesta k súboru, pre ktorý nastala udalosť), ***action*** (typ udalosti, ktorý nastal), ***file*** (názov súboru, pre ktorý nastala udalosť),
    - ***do*** – začiatok vykonávania cyklu *while*,

- ***(zvyšok kódu)*** – samotný kód, ktorý v cykle *while* chceme vykonávať – rozoberieme neskôr v aktuálnej podkapitole,
- ***done*** – koniec cyklu *while* a v našom prípade aj cyklu samotného.

V predchádzajúcom bloku sme vynechali pomerne veľkú časť kódu, ktorý sa nachádza vo vnútri cyklu *while* z predchádzajúceho odseku a ktorý sme označili ako (*zvyšok kódu*). Teraz sa pozrieme práve na túto časť. Stále platí, že kód budeme rozoberať po častiach v sekvenčnom poradí. Je taktiež nutné podotknúť, že v skripte vychádzame z predchádzajúcej znalosti štruktúry pyrdp logov. Prvú časť zobrazuje konfigurácia nižšie a pozostáva z týchto riadkov:

- ***sleep 10*** – hovoríme, že skript má 10 sekúnd počkať pred tým, než bude pokračovať, tento riadok zabezpečuje, že v pyrdp logoch sú už zapísané všetky dáta, ktoré budeme potrebovať,
- ***PYRDP_PUTPUT=$(/home/janovic/.local/bin/pyrdp-player "$path/$file" –headless | head -13)*** – do premennej *PYRDP_OUTPUT* ukladáme hodnotu, ktorú nám *pyrdp-player* vráti z novo vzniknutého súboru daného pripojenia v tzv. *headless* móde – ten nám umožňuje spustiť *pyrdp-player* nad daným súborom bez užívateľského rozhrania, teda dostávame výstup vo forme textového výpisu.
- ***ATTACKER_ADDRESS=$(cat "$LOG_FILE" | grep pyrdp.mitm.connections.security | tail -n1 | awk -F "clientAddress = '" '{print $2}' | awk -F "\\'" '{print $1}' | sed 's/\\x00//')*** – do premennej *ATTACKER_ADDRESS* ukladáme extrahovanú a „očistenú“ IP adresu útočníka z logov.

```
# Wait for 10 seconds
sleep 10

# Run the pyrdp-player command and capture its output
#echo "Running pyrdp-player on the file $file..."
PYRDP_OUTPUT=$(/home/janovic/.local/bin/pyrdp-player "$path/$file" --headless |
head -13)
#echo "pyrdp-player output captured."

# Extract the latest client address from the mitm.log
#echo "Extracting client address from the log..."
ATTACKER_ADDRESS=$(cat "$LOG_FILE" | grep pyrdp.mitm.connections.security | tail
-n1 | awk -F "clientAddress = '" '{print $2}' | awk -F "\\'" '{print $1}' | sed
's/\\x00//')
#echo "Client address extracted: $ATTACKER_ADDRESS"
```

Nasledujúcim blokom je časť zabezpečujúca geoiplokáciu na základe IP adresy útočníka, ktorú sme extrahovali v predchádzajúcom bloku. V úvode sa ubezpečíme, že premenná *ATTACKER_ADDRESS* obsahuje hodnotu, resp. nie je prázdna a následne na nej vykonávame pomocou lokálnej inštalácie GeoLite2 databáz geolokáciu a extrahovanie relevantnej hodnoty, podobne ako sme to demonštrovali v **Podkapitole 7.1.1.2** v časti, ktorá sa venovala geolokácii.

```
# Run mmdblookup commands on the attacker's address and capture the results
if [[ ! -z "$ATTACKER_ADDRESS" ]]; then
    #echo "Running mmdblookup commands on the Attacker address:
$ATTACKER_ADDRESS"
    COUNTRY=$(mmdblookup --file /usr/share/GeoIP/GeoLite2-Country.mmdb --ip
$ATTACKER_ADDRESS country names en | awk -F '"' 'NF>1{print $2}')

    CITY=$(mmdblookup --file /usr/share/GeoIP/GeoLite2-City.mmdb --ip
$ATTACKER_ADDRESS city names en | awk -F '"' 'NF>1{print $2}')

    SUBDIVISION=$(mmdblookup --file /usr/share/GeoIP/GeoLite2-City.mmdb --ip
$ATTACKER_ADDRESS subdivisions 0 names en | awk -F '"' 'NF>1{print $2}')

    CONTINENT=$(mmdblookup --file /usr/share/GeoIP/GeoLite2-City.mmdb --ip
$ATTACKER_ADDRESS continent names en | awk -F '"' 'NF>1{print $2}')

    LATITUDE=$(mmdblookup --file /usr/share/GeoIP/GeoLite2-City.mmdb --ip
$ATTACKER_ADDRESS location latitude | awk 'NF>0{print $1}')

    LONGITUDE=$(mmdblookup --file /usr/share/GeoIP/GeoLite2-City.mmdb --ip
$ATTACKER_ADDRESS location longitude | awk 'NF>0{print $1}')
fi
```

Pokračujeme časťou zobrazenou nižšie, ktorá z doteraz získaných údajov zostavuje do premennej *EMAIL_CONTENT* správu, ktorú budeme odosielať prostredníctvom mailovej notifikácie a ktorú môže čitateľ vidieť na **Obrázok 47**:

```
# Prepare the email content with mmdblookup results
EMAIL_CONTENT="Recording for the new session started.
The file '$file' was added to $path
Output of pyrdp-player command:
$PYRDP_OUTPUT
ATTACKER ADDRESS: $ATTACKER_ADDRESS

GeoIP Information for $ATTACKER_ADDRESS:
Continent: $CONTINENT
Country: $COUNTRY
Subdivision: $SUBDIVISION
City: $CITY
Latitude: $LATITUDE
```

```
Longitude: $LONGITUDE"
```

Ďalšou časťou je časť, ktorej úlohou je extrakcia používateľského mena (*USERNAME*) a hesla (*PASSWORD*) zadaného útočníkom, domény (*DOMAIN*) a názvu zariadenia, z ktorého sa útočník pripájal (*ATTACKER_HOST*) z pyrdp logov, podobne ako sme to vykonali v prípade IP adresy útočníka (*ATTACKER_ADDRESS*).

```
# Extracting each required value
USERNAME=$(cat "$LOG_FILE" | grep pyrdp.mitm.connections.security | tail -n1 |
awk -F "username = '" '{print $2}' | awk -F "'," '{print $1}' | sed 's/\\x00//')

PASSWORD=$(cat "$LOG_FILE" | grep pyrdp.mitm.connections.security | tail -n1 |
awk -F "password = '" '{print $2}' | awk -F "'," '{print $1}' | sed 's/\\x00//')

DOMAIN=$(cat "$LOG_FILE" | grep pyrdp.mitm.connections.security | tail -n1 | awk
-F "domain = '" '{print $2}' | awk -F "'," '{print $1}' | sed 's/\\x00//')

ATTACKER_HOST=$(cat "$LOG_FILE" | grep -oP 'Client hostname \K[^\x00]+' | tail -
n1)
```

Po získaní týchto informácií uložíme aktuálny čas do premennej *TIMESTAMP* a z doteraz získaných dát zostavíme záznam do logov, ktorý uložíme do premennej *LOG_ENTRY*, tak ako to zobrazuje nasledujúca časť nižšie.

```
# Get the current datetime
TIMESTAMP=$(date +"%Y-%m-%d %H:%M:%S")  # Format: YYYY-MM-DD HH:MM:SS

# Format the log entry with the current datetime and other information
LOG_ENTRY="$TIMESTAMP | Username: $USERNAME | Password: $PASSWORD | ATTACKER_IP:
$ATTACKER_ADDRESS | ATTACKER_HOST: $ATTACKER_HOST | Country: $COUNTRY | City:
$CITY | Subdivision: $SUBDIVISION | Continent: $CONTINENT | Latitude: $LATITUDE |
Longitude: $LONGITUDE | File: $file"
```

V tomto bode sa blížime ku koncu skriptu. Vzhľadom k tomu, že sme zostavili obsah mailovej notifikácie, ktorú chceme odoslať, ako aj záznam, ktorý chceme do logov uložiť, je posledným krokom samotné odoslanie mailu a uloženie do logov. Toto zabezpečuje časť zobrazená nižšie. Pred odoslaním a uložením záznamu sa presvedčíme, že súbor, ktorý pribudol do nami sledovaného priečinka má príponu *.pyrdp*. Teda sa jedná o záznam z pripojenia. Toto je nutné vzhľadom k tomu, že do tohto súboru ukladáme aj upravenú verziu záznamu, ktorú vytvárame pomocou skriptu (***strip.py***), ktorý budeme objasňovať dalej v tejto kapitole. Následne pomocou PostFix servera odošleme predpripravený obsah mailu (*EMAIL_CONTENT*) na adresu uloženú v premennej *EMAIL* a predpripravený záznam

(*LOG_ENTRY*) uložíme do logovacieho súboru s názvom ***filebeat_logs.txt***, ktorý sa nachádza v ***/home/janovic/pyrdp_output***.

```
if [[ "${file##*.}" == "pyrdp" ]]; then
    echo "$EMAIL_CONTENT" | mail -s "PyRDP New Session Established - Directory
Change Detected" "$EMAIL"
    # Append to the log file
    echo "$LOG_ENTRY" >> "/home/janovic/pyrdp_output/filebeat_logs.txt"
fi
```

8.1.2.3 Skript - run_strip.sh

Ako sme už naznačili ďalším skriptom, ktorý sme pre potreby našej práce vytvorili je skript s názvom ***strip.py*** uložený v ***/home/janovic/pyrdp_output/replays*** – ktorý zabezpečuje úpravu uloženého záznamu do prehľadnejšej a rýchlejšie čitateľnej podoby. Predtým, ale musíme čitateľovi priblížiť pomerne jednoduchý skript s názvom ***run_strip.sh***, ktorý sa nachádza v ***/home/janovic/pyrdp_output*** a zabezpečuje spustenie ***strip.py*** skriptu v požadovanom momente. Skript ***run_strip.sh*** je zobrazený nižšie a už na pohľad má veľa spoločného so skriptom ***watcher.sh***, ktorý sme v tejto kapitole detailne rozobrali. V skripte v úvode definujeme dve premenné *DIRECTORY_TO_MONITOR* a *STRIP_SCRIPT* – prvá obsahuje cestu k adresáru, ktorý budeme pomocou skriptu sledovať na zmeny a druhá obsahuje cestu k *strip.py* skriptu, ktorý budeme pomocou aktuálneho skriptu spúšťať. Následne pomocou *inotifywait* sledujeme daný adresár, rovnako ako v prípade skriptu *watcher.sh*, preto tento postup nebudeme znova objasňovať. Pomocou podmienky *if* potom sledujeme, či do priečinka pribudol súbor s príponou *.pyrdp* – teda súbor obsahujúci záznam z pripojenia. Ak áno, tak v cykle *while* overujeme, či súbor používa nejaký používateľ, ak áno, čakáme 10 sekúnd a opakujeme. Toto vykonávame z dôvodu, že útočník môže byť stále pripojený a teda je do súboru stále zapisované, v tomto prípade nechceme súbor upravovať. V prípade, že daný súbor už nepoužíva žiadny používateľ môžeme tento súbor začať upravovať pomocou skriptu ***strip.py***.

```
#!/bin/bash
DIRECTORY_TO_MONITOR="/home/janovic/pyrdp_output/replays"
STRIP_SCRIPT="/home/janovic/pyrdp_output/replays/strip.py"

# Monitor the directory for file creation
inotifywait -m -e create "$DIRECTORY_TO_MONITOR" | while read path action file;
do
    # Check if the file has a .pyrdp extension
    if [[ "${file##*.}" == "pyrdp" ]]; then
        # Wait until the file is not being accessed by any other process
```

```
        while fuser "$path/$file" &>/dev/null; do
            sleep 10
        done

        # Run the strip.py script on the file
        /usr/bin/python3 "$STRIP_SCRIPT" "$path/$file"
    fi
done
```

8.1.2.4 Skript - strip.py

Týmto sa teda dostávame k samotnému ***strip.py*** skriptu, ktorého úlohou je upraviť textový výstup (výstup z *pyrdp-player*) z konkrétneho pripojenia do lepšie čitateľnej a prehľadnejšej podoby. Neupravený výstup z pyrdp obsahuje veľké množstvo informácií, ktoré sú uzatvorené v <> zátvorkách. V skripte ich označujeme ako "tag-y" a vyzerajú napríklad nasledovne:

- <Resolution: 1920x1200>
- <Tab released> u
- <Shift released> t
- <Control released> b
- <Click (Left) @ (1115, 763)>
- <Smart card mapped: SCARD>
- <Printer mapped: PRN5>
- <Windows released>

Tieto tagy teda nesú dodatočné informácie a taktiež sú do logov pridávané po každom uvoľnení, či stlačení tlačidla Shift, Control, Windows, Tab atď., či dokonca kliknutí na obrazovku. Z povahy používania bežného PC nám je jasné, že týmto spôsobom logy narastajú veľmi rýchlym tempom a bolo by veľmi ťažké ich neskôr prechádzať a selektovať z nich len podstatné informácie. Práve preto sme do práce pridali *strip.py*, ktorého funkcionalitu si teraz podrobnejšie, po častiach, rozoberieme, pričom budeme ignorovať import knižníc do skriptu. Skript teda pozostáva troch funkcií :

- ***run_pyrdp_player*** – prijíma cestu k danému záznamu a spúšťa nad ním *pyrdp-player*, pričom jeho výstup v textovej podobne vracia,
- ***condense_and_filter_lines*** – prijíma textový výstup z *pyrdp-player*, generovaný predchádzajúcou funkciou a vykonáva potrebné úpravy,

- ***main*** – spúšťa obe funkcie v definovanom poradí.

Funkcia ***run_pyrdp_player*** je zobrazená nižšie a obsahuje tieto riadky:

- ***def run_pyrdp_player(file_path):*** – definujeme funkciu *run_pyrdp_plazer* s jedným vstupným parametrom *file_name*,
    - ***txt_file = f"{file_path}.txt"*** – pripravíme si názov výstupného súboru pomocou názvu originálneho súboru a prípony *.txt*,
    - ***cmd = ["/home/janovic/.local/bin/pyrdp-player", file_path, "--headless"]*** – do premennej *cmd* si uložíme celý príkaz, ktorý budeme chciet vykonať, v tomto prípade teda spúšťame *pyrdp-player* nad daným záznamom (názov záznamu je v premennej *file_name*) v *headless* móde,
    - ***with open(txt_file, "w") as output_file:*** – otvoríme predpripravený súbor s možnosťou zápisu,
        - ***subprocess.run(cmd, stdout=output_file, stderr=subprocess.PIPE, check=True)*** – vytvoríme proces, ktorý spustí predpripravený príkaz (premenná *cmd*), jeho výstup uloží do predpipraveného súboru (premenná *output_file*), presmeruje chybový výstup a overí či chyba nastala alebo nie,
    - ***r*eturn txt_file** – vraciame súbor v textovej podobe.

```
def run_pyrdp_player(file_path):
    txt_file = f"{file_path}.txt"
    cmd = ["/home/janovic/.local/bin/pyrdp-player", file_path, "--headless"]
    with open(txt_file, "w") as output_file:
        subprocess.run(cmd, stdout=output_file, stderr=subprocess.PIPE,
check=True)
    return txt_file
```

Funkcia ***condense_and_filter_lines*** je následne zobrazená nižšie a obsahuje tieto riadky:

- ***def condense_and_filter_lines(file_path):*** – definujeme funkciu *condense_and_filter_lines* s jedným vstupným parametrom *file_path*,
    - ***with open(file_path, 'r') as file:*** – otvoríme predpripravený súbor s možnosťou čítania,
    - ***tag_pattern = re.compile(r"^<((?!Control|Windows).)*?>\s*$")*** – vytvoríme vzor, ktorý zahŕňa všetky riadky, ktoré obsahujú <> zátvorky – tagy, nejedná sa o tagy *Control* a *Windows* a neobsahujú za zátvorkami text.

- ***condensed_lines = []*** – inicializujeme pole, kam budeme ukladať upravené riadky,
- ***prev_line = None*** – inicializujeme premennú *prev_line*, ktorá udržuje hodnotu prechádzajúceho riadka na *None*,
- ***count = 1*** – nastavíme počítadlo na hodnotu *1*,
- ***for line in lines*** – v cykle *for* prechádzame každý riadok záznamu,
  - ***stripped_line = line.strip()*** – do premennej *stripped_line* uložíme hodnotu aktuálneho riadka, bez medzier za a pred riadkom,
  - ***if prev_line is not None and stripped_line == prev_line and not tag_pattern.match(stripped_line):*** - overíme či hodnota *prev_line* nie je *None* a hodnota *stripped_line* je zhodná s hodnotou prechádzajúceho riadka (*prev_line*) a aktuálny riadok sa nezhoduje s vyššie definovaným vzorom. V tomto prípade navýšime počítadlo o 1 a nahradíme riadok vo výstupnom súbore tým istým riadkom s navýšenou hodnotou počítadla. Teda pre tri za sebou idúce riadky <TAG> by sme dostali <TAG> x3. Podmienka overujúca, či sa riadok nezhoduje so vzorom zabezpečí, že do upraveného záznamu zapisujeme len relevantné tagy (*Windows* a *Control*) a tagy, ktoré za sebou obsahujú nejaký text a to preto, že pokiaľ útočník píše na klávesnici, *pyrdp* tento text pridáva k poslednému tagu.
  - ***else*** – ak aktuálny riadok nie je zhodný s predchádzajúcim riadkom resetujeme počítadlo na 1. Pomocou *if* overíme, či sa hodnota aktuálneho riadku nezhoduje s definovaným vzorom a ak nie, tak ju pridáme do výstupného súboru. Do premennej *prev_line*, ktorá udržuje hodnotu prechádzajúceho riadku potom uložíme hodnotu aktuálneho riadku len v tom prípade, že sa nezhoduje s definovaným vzorom – teda predstavuje relevantnú hodnotu, inak ukladáme prázdny reťazec.
- ***with open(file_path, 'w') as file:*** - súbor obsahujúci textový záznam z daného pripojenia znova otvoríme s možnosťou zápisu,
  - ***file.writelines(condensed_lines)*** – prepíšeme súbor pomocou upravenej verzie uloženej v premennej *condensed_lines*.

Poslednou funkciou je funkcia ***main*** zobrazená nižšie, ktorá obsahuje tieto riadky:

- ***if __name__ == "__main__"*** – Táto podmienka kontroluje, či je skript spúšťaný priamo interpretom jazyka Python,
    - ***if len(sys.argv) != 2:*** - overíme, či bol skriptu pri jeho spustení predaný presne jeden argument, v prípade, že nie, vypíšeme chybovú hlášku a ukončíme proces,
    - ***file_path = sys.argv[1]*** – extrahujeme názov (cestu k) záznamu,
    - ***try*** – „vyskúšame“ spustenie oboch funkcií s extrahovaným názvom záznamu. Najskôr teda názov záznamu predáme funkcii *run_pyrdp_player*, ktorá nám z neho vytvorí textový výstup, ktorý uložíme do premennej *file_as_text*. Tú potom predáme funkcii *condense_and_filter_lines*, ktorá ho spracuje a uloží.
    - ***except*** – v prípade, že vykonávanie v nejakom bode zlyhá, chybu zachytíme, uložíme a ukončíme proces.

```
if __name__ == "__main__":

    if len(sys.argv) != 2:
        print("Usage: python script.py <file_path>")
        sys.exit(1)

    file_path = sys.argv[1]

    try:
        file_as_text = run_pyrdp_player(file_path)
        condense_and_filter_lines(file_as_text)
    except subprocess.CalledProcessError as e:
        print(f"Error running pyrdp-player: {e}")
        sys.exit(1)
```

8.1.3 WireGuard

Ako môžeme vidieť na konfigurácii nižšie, konfigurácia klienta je takmer zhodná s konfiguráciou virtuálneho systému WEB-POT z **Podkapitoly 7.1.2**. Blok konfigurácie rozhrania klienta teda opäť začína kľúčovým slovom [Interface] a končí sa pred ďalším kľúčovým slovom [Peer], kde začína blok konfigurácie servera, ku ktorému sa bude klient pomocou VPN tunela pripájať (táto časť je zhodná s konfiguráciou virtuálneho systému WEB-POT vzhľadom k tomu, že oba virtuálne systémy sa pripájajú k rovnakému

WireGuard serveru) – v našom prípade sa teda jedná o IP adresu WireGuard rozhrania virtuálneho systému ELK. Blok konfigurácie rozhrania klienta teda obsahuje tieto riadky:

- ***Address = 10.0.0.3/24*** – definícia IP adresy WireGuard rozhrania klienta,
- ***SaveConfig = true*** – inštrukcia pre WireGuard o tom, že daná konfigurácia je perzistentná, teda ju chceme uložiť a používať aj medzi reštartmi celého systému, či WireGuard servera a taktiež, že chceme uložiť zmeny vykonané v konfigurácii aj za behu rozhrania,
- ***ListenPort*** – definuje port, na ktorom WireGuard klient prijíma požiadavky na komunikáciu od servera prípadne iných klientov. V našom prípade nie, keďže jediné nakonfigurované spojenie, resp. [Peer] je práve náš WireGuard server na virtuálnom systéme ELK,
- ***FwMark = 0xca6c*** – automaticky generované firewall označenie (z angl. *Firewall Mark*) paketu slúžiace na identifikovanie konkrétnych paketov a smerovanie,
- ***PrivateKey*** – privátny kľúč daného klienta, slúžiaci na zašifrovanie odchádzajúcej komunikácie, resp. logov.

Konfigurácia ďalej obsahuje blok začínajúci kľúčovým slovom [Peer], kde definujeme s kým bude náš klient komunikovať – ako bolo uvedené v našom prípade sa jedná o VS ELK, resp. Logstash inštanciu na tomto systéme. Tento blok pozostáva z nasledujúcich riadkov:

- ***PublicKey = TMcCr2snEILgx06a5Cy/6D0+Jz2uH0DT/PsQ/iC+zA8=*** – obsahuje verejný kľúč pre WireGuard rozhranie virtuálneho systému ELK, potrebný na dešifrovanie komunikácie, ktorú server zašifroval svojím privátnym kľúčom a odoslal prostredníctvom tunela,
- ***AllowedIPs = 10.0.0.1/32*** – IP adresa WireGuard rozhrania na komunikáciu, pomocou ktorého je tento kľúč použiteľný. Logicky sa teda jedná o adresu WireGuard servera na virtuálnom systéme ELK.
- ***Endpoint*** – IP adresa virtuálneho systému ELK v danej sieti a port, na ktorom očakáva a odpovedá na prichádzajúce VPN spojenia. V našom prípade sa opäť jedná o internú IP adresu, špecifickú pre našu testovaciu infraštruktúru.
- ***PersistentKeepalive = 30 –*** predstavuje interval v sekundách, po ktorých bude klient na danom rozhraní automaticky odosielať tzv. „keepalive“ pakety, aby udržal

perzistentné spojenie so serverom, aj keď cez VPN tunel neprechádzajú žiadne iné dáta.

```
[Interface]
Address = 10.0.0.3/24
SaveConfig = true
ListenPort = 37710
FwMark = 0xca6c
PrivateKey = (Skryté pre potreby tlače)

[Peer]
PublicKey = TMcCr2snEILgx06a5Cy/6D0+Jz2uH0DT/PsQ/iC+zA8=
AllowedIPs = 10.0.0.1/32
Endpoint = 192.168.100.43:51820
PersistentKeepalive = 30
```

Po spustení a otestovaní bolo posledným krokom nastavenie automatického spustenia WireGuard rozhrania *wg0* pomocou príkazu „***sudo systemctl enable wg-quick@wg0***". Samotný konfiguračný súbor sa vo virtuálnom systéme PyRDP nachádza v ***/etc/wireguard/wg0.conf***

8.1.4 FileBeat

Ako už vieme z predchádzajúcej kapitoly, FileBeat je konfiguračne pomerne jednoduchý protokol a veľkú časť konfigurácie si generuje sám, pričom od používateľa vyžaduje len zopár konfiguračných zmien. V tejto kapitole sa teda opäť zameriame len na tieto zmeny, celú konfiguráciu nájdeme v **PRÍLOHA P I: OBSAH CD**. Veľmi dôležitou časťou konfigurácie FileBeat je časť vstupov (z angl. *Inputs*), v ktorej konfigurujeme vstupy, resp. súbory logov, ktoré má FileBeat sledovať na prípadné zmeny a odosielať – viac v časti výstupov (z angl. *Output*). Časť vstupov zobrazuje konfigurácie nižšie a pozostáva teda z týchto riadkov:

- ***filebeat.inputs:*** - označuje kde sa začína sekcia definície vstupov,
- ***- type: filestream*** - určuje typ vstupu, v tomto prípade sa jedná o filestream, ktorý slúži na zbieranie logov zo súborov,
- ***id: pyrdp-logs*** **-** jedinečný identifikátor tohto vstupu medzi všetkými ostatnými vstupmi. Identifikátor je vyžadovaný pre každý vstup, v našom prípade máme vstup len jeden, takže sa vyžaduje len jeden jedinečný identifikátor.

- ***enabled: true* –** hodnota určujúca, či má byť táto konfigurácia vnímaná ako aktívna a má byť použitá. Zmena hodnoty z true na false, by teda logicky spôsobila „deaktiváciu“ konfigurácie tohto vstupu a FileBeat by ju ignoroval.
- ***paths:*** - definuje začiatok odseku v konfigurácii, v ktorom udávame kde sa nachádzajú súbory (logy), ktoré má FileBeat sledovať,
    - **– *~~/~~home/janovic/pyrdp_output/filebeat_logs.txt*** – cesta k súboru, ktorý má FileBeat sledovať. Jedná sa teda o textový (.txt) súbor s názvom *filebeat_logs* nachádzajúci sa v priečinku */home/janovic/pyrdp_output*
- ***fields:*** - definuje začiatok odseku slúžiaceho na pridanie dodatočných polí (z angl. *filed*) s informáciami. Využívajú sa teda najmä na pridávanie metadát, kategorizáciu, jednoznačnú identifikáciu a podobne,
    - ***type:*** “pyrdp” – definícia poľa s informáciami vo formáte <kľuč>:<hodnota>. Hovoríme teda, že chceme pridať pole s názvom *type* s hodnotou “pyrdp” (pochopiteľne bez úvodzoviek). Toto pole slúži Logstash inštancii na virtuálnom systéme ELK na jednoznačnú identifikáciu logov a ich adekvátne spracovanie, resp. aplikovanie príslušného filtra. Podmienené aplikovanie filtrov na základe premennej *type* sme už spomínali v **Podkapitole 6.1.2.3**, kde sme vysvetľovali konfiguráciu Logstash.
    - ***fields_under_root: true*** – hovoríme, že premenná *type* má byť vložená priamo do koreňa každého odosielaného logu, aby k nej bolo možné pristupovať priamo, bez nutnosti definovať dlhšiu cestu (tzv. zanorenie).

```
# ============================== Filebeat inputs ===============================

filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input-specific configurations.

# filestream is an input for collecting log messages from files.
- type: filestream

  # Unique ID among all inputs, an ID is required.
  id: pyrdp-logs

  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based paths.
```

```
  paths:
    - /home/janovic/pyrdp_output/filebeat_logs.txt
    #- c:\programdata\elasticsearch\logs\*

  fields:
    type: "pyrdp"
  fields_under_root: true
```

Sekcia výstupov pozostáva z dvoch pomyselných častí, z ktorých ju nutné si vybrať na základe toho, ako plánuje FileBeat protokol používať, resp. teda kam budeme logy odosielať. V našom prípade budeme logy odosielať do Logstash inštancie na virtuálnom systéme ELK a teda volíme časť konfigurácie označenú ako „*Logstash Output*“, ktorú môžeme vidieť nižšie. V tejto časti musíme vykonať len veľmi jednoduchú konfiguračnú zmenu na dvoch riadkoch:

- ***output.logstash:*** - definujeme začiatok odseku konfigurácie výstupu do danej inštancie Logstash,
    - ***hosts: [“10.0.0.1:5044”]*** – definujeme na akej IP adrese sa daná inštancia Logstash nachádza a na akom porte očakáva prichádzajúci komunikáciu, resp. logy. Ako môžeme vidieť v našom prípade sa teda jedná o adresu serverového WireGuard rozhrania virtuálneho systému ELK a predvolený port projektu Logstash. Odosielanie logov teda prebieha pomocou zabezpečeného VPN tunela, ako sme to už niekoľkokrát uvideli v predchádzajúcich kapitolách.

```
# ------------------------------ Logstash Output -------------------------------
output.logstash:
  # The Logstash hosts
  hosts: ["10.0.0.1:5044"]
```

Po spustení a otestovaní bolo posledným krokom nastavenie automatického spustenia pomocou príkazu „***sudo systemctl enable filebeat***”.

8.1.5 GeoLite2

Celý proces inštalácie GeoLite2 databáz, ich konfiguračné súbory, ako aj celý proces automatickej aktualizácie je zhodný s postupom pre VS WEB-POT, teda sme ho už popísali v **Podkapitole 7.1.4**. Nebudeme ho teda znova popisovať v tejto kapitole. Jediným rozdielom je, že v prípade PyRDP odpadá nutnosť registrácie na stránkach MaxMind a generovania licenčného kľúča, keďže tento proces sme už absolvovali pre WEB-POT.

Druhým drobným rozdielom je rozdiel v ceste ku skriptu automatickej aktualizácie a logovaciemu súboru vzhľadom k tomu, že vo virtuálnom systéme PyRDP je iné meno užívateľa.

Pre WEB-POT sme uvideli zápis v cron tabuľke správcu ako:

- ***0 3 * * 7 /home/utb/run_geoipupdate >> /home/utb/Desktop/update_log.txt 2>&1***

Pre PyRDP potom vyzerá nasledovne:

- ***0 3 * * 7 /home/janovic/run_geoipupdate>> /home/janovic/Desktop/update_log.txt 2>&1***

9 T-POT

VS *DP_Janovic_tpot* predstavuje, tak ako sme v stručnosti uviedli v **Podkapitole 5.4,** plnú (všetky honeypoty a ďalšie súčasti) inštanciu T-Pot projektu (viz. **Podkapitola 4.10** alebo [24]), pričom sme projekt upravili pre naše potreby. Projekt teda obsahuje všetky súčasti avšak nie všetky z nich sú pre našu prácu relevantné a teda niektoré sú v projekte deaktivované.

9.1 Implementácia

VS DP_Janovic_tpot tak, ako ho ilustruje **Obrázok 36**, pozostáva z týchto komponent:

- ***T-Pot projekt*** – sme už v skratke priblížili v **Podkapitolách 4.10**, **5.4** ako aj v **Kapitole 3**, kde sme tento projekt na približne týždeň nasadili v cloudovom prostredí a sledovali chovanie útočníkov. Jedná sa teda o all-in-one honeypot platformu od spoločnosti Telekom, ktorá v sebe v aktuálnej verzii zahŕňa, okrem iného, 22 honeypotov, ELK inštanciu, NGINX reverse-proxy, ako aj prostriedky na monitorovanie sieťovej prevádzky. Ako sme už uvideli, nie všetky z týchto súčastí sme v práci využili a preto sme ich deaktivovali – viac v **Podkapitole 9.1.3.**
- ***WireGuard*** – rovnako ako v prípade VS WEB-POT a PyRDP z predchádzajúcich kapitol sme WireGuard konfigurovali do pozície klienta, ktorý zabezpečený WireGuard VPN tunel využíva na odosielanie logov o útokoch do virtuálneho systému ELK, resp. teda Logstash inštancie, ktorá sa na ňom nachádza. Opäť platí, že konfigurácia VPN tunela je nastavená tak, aby jediná komunikácia (dáta), ktorá nim putuje bola komunikácia spojená s prenosom logov z virtuálneho systému DP_Janovic_tpot do virtuálneho systému ELK.
- ***FileBeat*** – rovnako ako v prípade VS WEB-POT a PyRDP z predchádzajúcich kapitol FileBeat využívame na sledovanie logov, resp. ich zmeny a zasielanie týchto inkrementov do virtuálneho systému ELK na ďalšie spracovanie a vizualizáciu.

V nasledujúcich podkapitolách sa pozrieme na výsledky našej implementácie a ukážeme skripty a konfigurácie, ktorými sme ju docielili.

9.1.1 WireGuard

Ako môžeme vidieť na konfigurácii nižšie, konfigurácia klienta je takmer zhodná s konfiguráciou virtuálneho systému WEB-POT a PyRDP z **Podkapitoly 7.1.2**, respektíve

8.1.3. Blok konfigurácie rozhrania klienta teda opäť začína kľúčovým slovom [Interface] a končí sa pred ďalším kľúčovým slovom [Peer], kde začína blok konfigurácie servera ku ktorému sa bude klient pomocou VPN tunela pripájať (táto časť je zhodná s konfiguráciou VS WEB-POT a VS PyRDP vzhľadom k tomu, že tieto virtuálne systémy sa pripájajú k rovnakému WireGuard serveru) – v našom prípade sa teda jedná o IP adresu WireGuard rozhrania virtuálneho systému ELK. Blok konfigurácie rozhrania klienta teda obsahuje tieto riadky:

- ***Address = 10.0.0.4/24*** – definícia IP adresy WireGuard rozhrania klienta,
- ***SaveConfig = true*** – inštrukcia pre WireGuard o tom, že daná konfigurácia je perzistentná, teda ju chceme uložiť a používať aj medzi reštartmi celého systému, či WireGuard servera a taktiež, že chceme uložiť zmeny vykonané v konfigurácii aj za behu rozhrania,
- ***ListenPort*** – definuje port, na ktorom WireGuard klient prijíma požiadavky na komunikáciu od servera, pripadne iných klientov. V našom prípade nie, keďže jediné nakonfigurované spojenie, resp. [Peer] je práve náš WireGuard server na virtuálnom systéme ELK,
- ***FwMark = 0xca6c*** – automaticky generované firewall označenie (z angl. *Firewall Mark*) paketu slúžiace na identifikovanie konkrétnych paketov a smerovanie,
- ***PrivateKey*** – privátny kľúč daného klienta, slúžiaci na zašifrovanie odchádzajúcej komunikácie, resp. logov.

Konfigurácia ďalej obsahuje blok začínajúci kľúčovým slovom [Peer], kde definujeme s kým bude náš klient komunikovať – ako bolo uvedené v našom prípade sa jedná o VS ELK, resp. Logstash inštanciu na tomto systéme. Tento blok pozostáva z nasledujúcich riadkov:

- ***PublicKey = TMcCr2snEILgx06a5Cy/6D0+Jz2uH0DT/PsQ/iC+zA8=*** – obsahuje verejný kľúč pre WireGuard rozhranie virtuálneho systému ELK, potrebný na dešifrovanie komunikácie, ktorú server zašifroval svojím privátnym kľúčom a odoslal prostredníctvom tunela,
- ***AllowedIPs = 10.0.0.1/32*** – IP adresa WireGuard rozhrania na komunikáciu, pomocou ktorého je tento kľúč použiteľný. Logicky sa teda jedná o adresu WireGuard servera na virtuálnom systéme ELK.

- ***Endpoint*** – IP adresa virtuálneho systému ELK v danej sieti a port, na ktorom očakáva a odpovedá na prichádzajúce VPN spojenia. V našom prípade sa opäť jedná o internú IP adresu, špecifickú pre našu testovaciu infraštruktúru.
- ***PersistentKeepalive = 30*** **–** predstavuje interval v sekundách, po ktorých bude klient na danom rozhraní automaticky odosielať tzv. „keepalive“ pakety, aby udržal perzistentné spojenie so serverom, aj keď cez VPN tunel neprechádzajú žiadne iné dáta.

```
[Interface]
Address = 10.0.0.4/24
SaveConfig = true
ListenPort = 54898
FwMark = 0xca6c
PrivateKey = (Skryté pre potreby tlače)

[Peer]
PublicKey = TMcCr2snEILgx06a5Cy/6D0+Jz2uH0DT/PsQ/iC+zA8=
AllowedIPs = 10.0.0.1/32
Endpoint = 192.168.100.43:51820
PersistentKeepalive = 30
```

9.1.2 FileBeat

Ako už vieme z príkladov z predchádzajúcich kapitol, FileBeat je konfiguračne pomerne jednoduchý protokol a veľkú časť konfigurácie si generuje sám, pričom od používateľa vyžaduje len zopár konfiguračných zmien. V tejto kapitole sa teda opäť zameriame len na tieto zmeny, celú konfiguráciu nájdeme v **PRÍLOHA P I: OBSAH CD**. Veľmi dôležitou časťou konfigurácie FileBeat je časť vstupov (z angl. *Inputs*), v ktorej konfigurujeme vstupy, resp. súbory logov, ktoré má FileBeat sledovať na prípadné zmeny a odosielať – viac v časti výstupov (z angl. *Output*). Časť vstupov je v tomto prípade zložitejšia, než v prípade VS WEB-POT a PyRDP, vzhľadom k tomu, že definujeme väčšie množstvo vstupov (26) – pre väčšie množstvo honeypotov a iných súčastí systému (Fatt, Suricata, p0f, Adbhoney, Ciscoasa, CitrixHoneyPot, ConPot, Cowrie, Dionaea, Dicompot, Ddospot, Elasticpot, Endlessh, Glutton, Hellpot, Heralding, Honeypots, Honeytrap, Ipphoney, Log4pot, MailHoney, Medpot, Redishoneypot, Sentrypeer, NGINX, Tanner). Vstupy sú si v mnohom podobné, a preto ich nebudeme rozoberať všetky – líšia sa najmä v parametroch jedinečných pre daný vstup ako je ID a cesta k logom a použitím, resp. nepoužitím parserov. Príklad jedného zo vstupov zobrazuje konfigurácia nižšie a pozostáva teda z týchto riadkov:

- ***filebeat.inputs:*** - označuje, kde sa začína sekcia definície vstupov,

- ***- type: filestream*** - určuje typ vstupu, v tomto prípade sa jedná o filestream, ktorý slúži na zbieranie logov zo súborov,
- ***id: fatt -*** jedinečný identifikátor tohto vstupu medzi všetkými ostatnými vstupmi. Identifikátor je vyžadovaný pre každý vstup, v našom prípade máme vstup len jeden, takže sa vyžaduje len jeden jedinečný identifikátor.
- ***enabled: true –*** hodnota určujúca, či má byť táto konfigurácia vnímaná ako aktívna a má byť použitá. Zmena hodnoty z true na false, by teda logicky spôsobila „deaktiváciu" konfigurácie tohto vstupu a FileBeat by ju ignoroval.
- ***paths: –*** definuje začiatok odseku v konfigurácii, v ktorom udávame kde sa nachádzajú súbory (logy), ktoré má FileBeat sledovať,
  - ***– /data/fatt/log/fatt.log*** – cesta k súboru, ktorý má FileBeat sledovať. Jedná sa teda o súbor s názvom *fat.log* nachádzajúci sa v priečinku */data/fatt/log*
- ***parsers: –*** definuje začiatok odseku konfigurácie, v ktorom udávame zoznam parserov logov,
  - ***– ndjson: –*** definujeme parser pre logy vo formáte NDJSON,
  - ***keys_under_root: true –*** hovoríme, že kľúče (z angl. *keys*) JSON objektu logov by mali byť uložené priamo do koreňa odosielaného logu,
  - ***add_error_key: true –*** hovoríme, že v prípade, že nevie parser nejaký log alebo jeho časť spracovať, má do odosielanej správy vložiť informáciu o chybe,
- ***fields:*** - definuje začiatok odseku slúžiaceho na pridanie dodatočných polí (z angl. *filed*) s informáciami. Využívajú sa teda najmä na pridávanie metadát, kategorizáciu, jednoznačnú identifikáciu a podobne.
  - ***type:*** "Fatt" – definícia poľa s informáciami vo formáte <kľuč>:<hodnota>. Hovoríme teda, že chceme pridať pole s názvom *type* s hodnotou "Fatt" (pochopiteľne bez úvodzoviek). Toto pole slúži Logstash inštancii na virtuálnom systéme ELK na jednoznačnú identifikáciu logov a ich adekvátne spracovanie, resp. aplikovanie príslušného filtra. Podmienené aplikovanie filtrov na základe premennej *type* sme už spomínali v **Podkapitole 6.1.2.3**, kde sme vysvetľovali konfiguráciu Logstash.
  - ***fields_under_root: true*** – hovoríme, že premenná *type* má byť vložená priamo do koreňa každého odosielaného logu, aby k nej bolo možné pristupovať priamo, bez nutnosti definovať dlhšiu cestu (tzv. zanorenie).

```
# ============================== Filebeat inputs ===============================

filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input-specific configurations.

# filestream is an input for collecting log messages from files.
- type: filestream

  # Unique ID among all inputs, an ID is required.
  id: fatt

  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /data/fatt/log/fatt.log
  parsers:
    - ndjson:
        keys_under_root: true
        add_error_key: true
  fields:
    type: "Fatt"
  fields_under_root: true
```

Sekcia výstupov pozostáva z dvoch pomyselných častí, z ktorých je nutné si vybrať na základe toho, ako plánujeme FileBeat protokol používať, resp. teda kam budeme logy odosielať. V našom prípade budeme logy odosielať do Logstash inštancie na virtuálnom systéme ELK, a teda volíme časť konfigurácie označenú ako „*Logstash Output*“, ktorú môžeme vidieť nižšie. V tejto časti musíme vykonať len veľmi jednoduchú konfiguračnú zmenu na dvoch riadkoch:

- ***output.logstash:*** - definujeme začiatok odseku konfigurácie výstupu do danej inštancie Logstash,
    - ***hosts: [“10.0.0.1:5044”]*** – definujeme na akej IP adrese sa daná inštancia Logstash nachádza a na akom porte očakáva prichádzajúci komunikáciu, resp. logy. Ako môžeme vidieť v našom prípade sa teda jedná o adresu serverového WireGuard rozhrania virtuálneho systému ELK a predvolený port projektu Logstash. Odosielanie logov teda prebieha pomocou zabezpečeného VPN tunela, ako sme to už niekoľkokrát uvideli v predchádzajúcich kapitolách.

```
# ------------------------------ Logstash Output -------------------------------
output.logstash:
  # The Logstash hosts
  hosts: ["10.0.0.1:5044"]
```

Po spustení a otestovaní bolo posledným krokom nastavenie automatického spustenia pomocou príkazu „***sudo systemctl enable filebeat***".

9.1.3 T-Pot projekt

T-Pot projekt má veľmi podrobnú dokumentáciu (obsahujúcu postupy inštalácie, základného nastavenia, definíciu komponent a portov a iné) na adrese zdroja [24], preto v tejto podkapitole nebudeme túto dokumentáciu podrobne prepisovať, ale zameriame sa na, pre nás, najpodstatnejšie časti a zmeny, ktoré sme vykonali. T-Pot projekt teda podľa spomínanej dokumentácie obsahuje docker kontajnery pre 22 honeypotov:

1. ***Adbhoney***: Honeypot navrhnutý na detekciu pokusov o zneužitie zraniteľnosti Android Debug Bridge (ADB),
2. ***CiscoASA***: Špeciálne navrhnutý na emuláciu zariadení Cisco Adaptive Security Appliance (ASA),
3. ***CitrixHoneyPot***: Honeypot zameraný na emuláciu služieb Citrix,
4. ***ConPot***: Open-source ICS/SCADA honeypot, ktorý simuluje priemyselné riadiace systémy,
5. ***Cowrie***: Honeypot pre SSH a Telnet,
6. ***Ddospot***: Honeypot navrhnutý na prilákanie DDoS útokov,
7. ***Dicompot***: Honeypot zameraný na detekciu útokov zameraných na služby DICOM (Digital Imaging and Communications in Medicine),
8. ***Dionaea***: Honeypot navrhnutý na zachytávanie "shellcode" útokov,
9. ***Elasticpot***: Honeypot navrhnutý na detekciu pokusov o zneužitie zraniteľnosti Elasticsearch,
10. ***Endlessh***: SSH honeypot, ktorý dokáže prijať prichádzajúce SSH pripojenia, ale neumožňuje autentifikáciu, čím plytvá zdrojmi a časom útočníkov,
11. ***Glutton***: Sieťový honeypot navrhnutý na zachytávanie útokov zameraných na webové služby, najmä na systémy na správu obsahu (CMS) ako WordPress,
12. ***Heralding***: Honeypot zameraný na detekciu útokov a zber zadaných prihlasovacích údajov pre rôzne protokoly a služby (FTP, Telnet, SSH, HTTP, HTTPS, POP3, POP3s, IMAP, IMAPs, SMTP, VNC, Postgresql a Socks5),

13. ***Hellpot***: Konečný honeypot, ktorý HTTP botom nerešpektujúcich *robots.txt* donekonečna zasiela dáta a simuluje skutočnú webovú stránku,
14. ***Honeypots***: Projekt, ktorý v sebe skrýva 30 honeypotov s rôznou úrovňou interakcie navrhnutých na monitorovanie sieťovej prevádzky, aktivity botov a zberu prihlasovacích mien a hesiel,
15. ***Honeytrap***: Honeypot navrhnutý na monitorovanie útokov proti TPC alebo UDP službám,
16. ***Ipphoney***: Honeypot emulujúci tlačiareň podporujúcu IPP (Internet Printing Protocol),
17. ***Log4pot***: Honeypot navrhnutý na detekciu zameraných na log4j,
18. ***MailHoney***: Honeypot zameraný na detekciu útokov zameraných na emailové služby, ako je SMTP,
19. ***Medpot***: Honeypot zameraný na detekciu útokov zameraných na zdravotnícke zariadenia,
20. ***Redishoneypot***: Honeypot navrhnutý na detekciu útokov zameraných na Redis,
21. ***Sentrypeer***: Honeypot navrhnutý na ochranu SIP serverov,
22. ***Tanner***: Honeypot navrhnutý na emuláciu správania sa zraniteľnej webovej aplikácie,
    - ***Snare***: Senzor navrhnutý na detekciu útokov na webové stránky,

Dokumentácia taktiež uvádza niekoľko nástrojov:

- ***Cockpit*** – slúži na ľahkú a bezpečnú webovú správu a webový terminál,
- ***Cyberchef*** – webová aplikácia pre šifrovanie, kódovanie, kompresiu a analýzu dát,
- ***ELK*** – slúži na vizualizáciu zachytených útokov,
- ***Elasticvue*** – webové rozhranie na prezeranie a interakciu s Elasticsearch,
- ***Fatt*** – skript založený na pyshark pre extrakciu metadát siete a odtlačkov z pcap súborov a živého sieťového prenosu,
- ***T-Pot-Attack-Map*** – animovaná útočná mapa pre T-Pot,
- ***P0f*** – nástroj pre pasívne monitorovanie sieťovej prevádzky,
- ***Spiderfoot*** – nástroj pre zbieranie informácií OSINT (Open-source Intelligence),
- ***Suricata*** – sieťový monitorovací nástroj.

Aj napriek tomu, že T-Pot projekt obsahuje všetky honeypoty a nástroje uvedené vyššie, už v základnej konfigurácii nefungujú všetky simultánne, vzhľadom k tomu, že niektoré

honeypoty zdieľajú funkcionalitu, napr. ako v prípade *Cowrie* a *Endlessh* alebo sa jedná o celý honeypot projekt, ako napr. *Honeypots*. T-Pot teda umožňuje definovať, ktoré z honeypotov chceme použiť pomocou súboru *tpot.yml*, ktorý sa nachádza v */opt/tpot/etc*. Tvorcovia T-Pot projektu taktiež v */opt/tpot/etc/compose* vytvorili niekoľko predpripravených príkladov konfigurácií, ktorými sa môže používateľ inšpirovať a adaptovať ich pre svoje potreby alebo ich použiť tak, ako sú vytvorené. Každá z týchto predpripravených konfigurácií obsahuje definíciu konkrétnych honeypotov a služieb špecifických pre dané použitie (potreby, odvetvie), či veľkosť (množstvo honeypotov a služieb). Nájdeme tu konfigurácie ako: *collector.yml, hive_sensor.yml, hive.yml, industrial.yml, log4j.yml, medical.yml, mini.yml, nextgen.yml, sensor.yml, standard.yml, tarpit.yml.* O konkrétnom použití vypovedajú už samotné názvy týchto konfiguračných súborov, pričom v našom prípade používame upravenú verziu štandardnej konfigurácie (*standard.yml*). V prípade, že chce daný používateľ použiť inú konfiguráciu je potrebné jej obsah skopírovať do súboru *tpot.yml*, upraviť – ak si to tak používateľ vyžaduje a reštartovať tpot službu pomocou príkazu:

- ***(sudo) systemctl restart tpot***

Celý súbor *tpot.yml* je dostupný v **PRÍLOHA P I: OBSAH CD** a dal by sa rozdeliť do štyroch pomyselných častí:

- ***Definícia sietí pre docker kontajnery,***
- ***Honeypoty:***
    - Adbhoney,
    - Ciscoasa,
    - CitrixHoneyPot,
    - ConPot,
    - Cowrie,
    - Ddospot,
    - Dicompot,
    - Dionaea,
    - Elasticpot,
    - Heralding,
    - Honeytrap,
    - Ipphoney,

  - MailHoney,
  - Medpot,
  - Redishoneypot,
  - Sentrypeer,
  - Tanner+Snare,

(V tejto konfigurácii teda nie sú zahrnuté honeypoty – *Endlessh, Glutton, Heralding, Hellpot, Honeypots a Log4pot*)

- ***NSM (Network Security Monitoring) nástroje:***
  - Fatt,
  - P0f,
  - Suricata.
- ***Iné nástroje:***
  - Elasticsearch,
  - Kibana,
  - Logstash,
  - Map Redis,
  - Map Data,
  - NGINX,
  - Spiderfoot,
  - Ewsposter.

Naša úprava tohto súboru spočívala v zakomentovaní (deaktivácii) nástrojov:

- ***Elasticsearch, Kibana, Logstash a NGINX*** – vzhľadom k tomu, že vo virtuálnom systéme ELK (viz. **Kapitola 6**) využívame vlastnú inštanciu týchto služieb,
- ***Map Redis a Map Data*** – vzhľadom k tomu, že vizualizáciu pomocou máp v našej práci opomíname,
- ***Spiderfoot –*** vzhľadom k tomu, že danú službu neplánujeme používať pre potreby práce.

10 TESTOVANIE

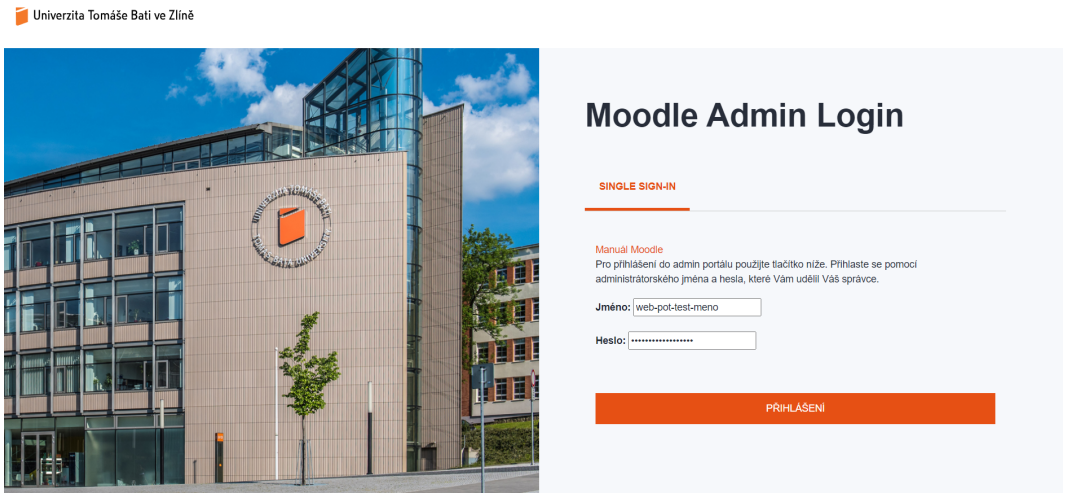
V nasledujúcej kapitole budeme v jednoduchosti demonštrovať priebeh útoku na naše honeypoty a to ako z pohľadu útočníka, tak aj z pohľadu honeypotu samotného. Vykonáme teda útok na daný honeypot a budeme demonštrovať proces, ktorý tento útok spustí.

10.1 WEB-POT

Nasledujúce podkapitoly ilustrujú priebeh útoku na honeypot (VS) WEB-POT ako z pohľadu útočníka, tak z pohľadu honeypotu samotného.

10.1.1 Z pohľadu útočníka

1. Prejdeme na adresu nášho webového honeypotu (falošného administrátorského rozhrania moodle) – **moodletest.local** (viz. **Obrázok 42**).
2. Zadáme prihlasovacie meno (*web-pot-test-meno*) a heslo (*web-pot-test-heslo*), odošleme a počkáme interval, počas ktorého web simuluje prihlasovanie – viz. **Obrázok 50**.



Obrázok 50 - Testovanie - WEB-POT - pokus o prihlásenie, Zdroj: vlastný

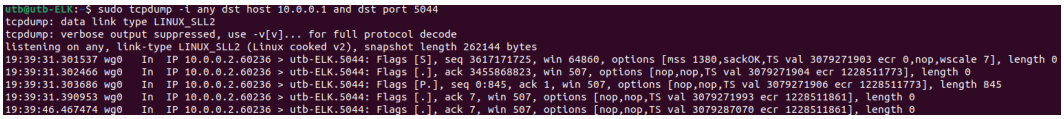
3. Zobrazí sa nám informácia o tom, že sme zadali nesprávne prihlasovacie meno a heslo (viz. **Obrázok 43**).
4. Vykonáme opakované pokusy o prihlásenie, pričom k prihlasovacím menám a heslám pridávame postupne inkrementované čísla (1,2,3 atď.).

10.1.2 Z pohľadu honeypotu

1. Po zadaní prihlasovacích údajov útočníkom náš *post.php* skript extrahoval IP adresu útočníka, vykonal nad ňou geolokáciu a spolu so zadanými prihlasovacími informáciami ju uložil do pripraveného logovacieho súboru – viz. **Podkapitola 7.1.1.3** a **Obrázok 51**. Logy zobrazujú lokálnu ip adresu, pre potreby geolokácie bola na testovanie adresa neskôr ručne zmenená.

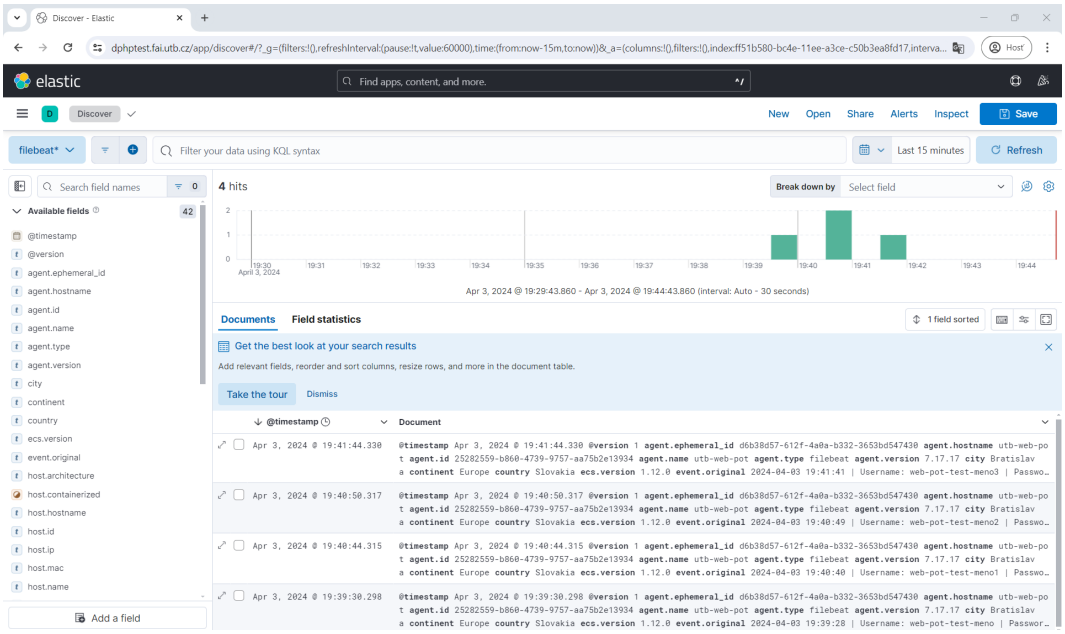
Obrázok 51 - Testovanie - WEB-POT - Log, Zdroj: vlastný

2. Lokálna inštalácia Filebeat protokolu (viz. **Podkapitola 7.1.3**) zaznamená, že do logov pribudol nový záznam, ktorý odošle do Logstash inštancie na virtuálnom systéme ELK, cez zabezpečený WireGuard tunel – viz. **Obrázok 52** - Testovanie - WEB-POT - WireGuard, Zdroj: vlastný.

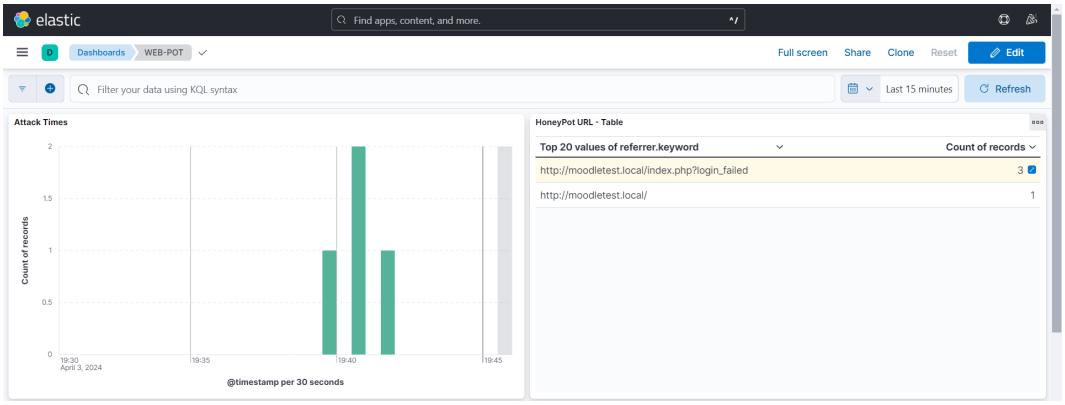


Obrázok 52 - Testovanie - WEB-POT - WireGuard, Zdroj: vlastný

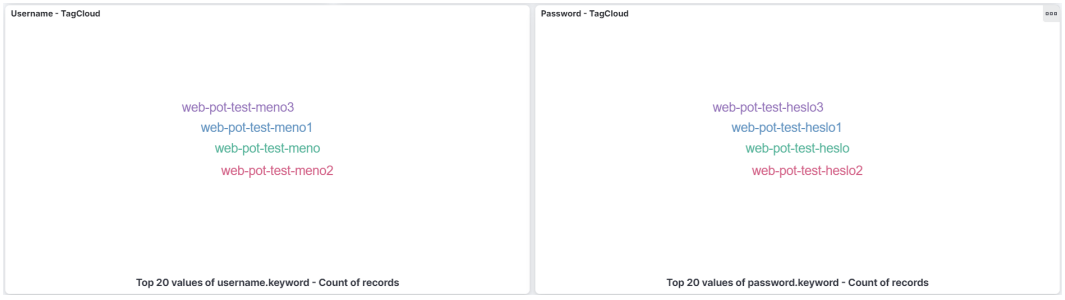
3. Logstash inštancia na virtuálnom systéme ELK, rozozná pomocou premennej *type*, že sa jedná o logy z virtuálneho systému (honeypotu) WEB-POT a aplikuje príslušný filter (viz. **Podkapitola 6.1.2.3**).
4. Logy sú po spracovaní odoslané do Elasticsearch inštancie na rovnakom virtuálnom systéme (ELK).
5. Postup sa opakuje pre všetky pokusy o prihlásenie, ktoré sme vykonali.
6. Logy následne môžeme vidieť v Kibana rozhraní *Discover*, ako aj v predpripravenom dashboarde *WEB-POT* – viz. **Obrázok 53, Obrázok 54** a **Obrázok 55**.



Obrázok 53 - Testovanie - WEB-POT - Kibana Discover, Zdroj: vlastný



Obrázok 54 - Testovanie - WEB-POT - Kibana Dashboard, Zdroj: vlastný



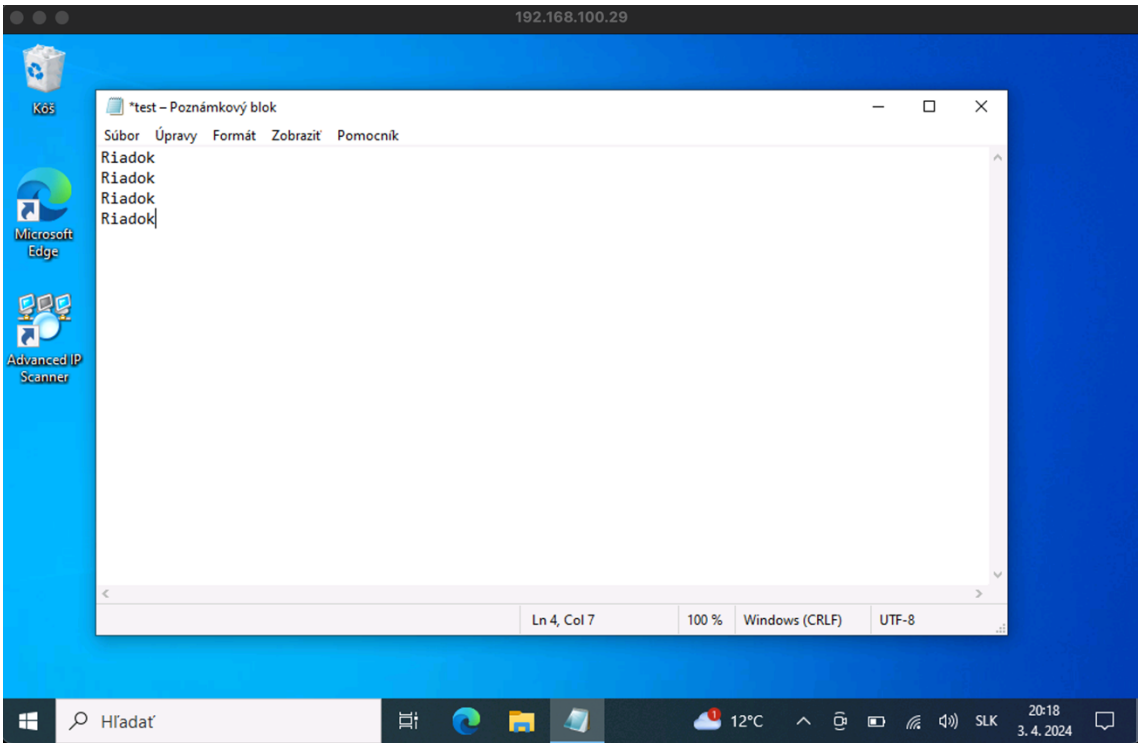
Obrázok 55 - Testovanie - WEB-POT - Kibana Dashboard 2, Zdroj: vlastný

10.2 PyRDP

Nasledujúce podkapitoly ilustrujú priebeh útoku na honeypot (VS) PyRDP ako z pohľadu útočníka, tak z pohľadu honeypotu samotného.

10.2.1 Z pohľadu útočníka

1. Identifikujeme zariadenie, na ktoré budeme útočiť. Skutočný útočník by pravdepodobne vykonal nejakú formu skenovania, pomocou ktorej by dané zariadenie odhalil (napr. nmap) v našom prípade na testovanie IP adresu zariadenia poznáme.
2. Vykonáme RDP pripojenie na naše zraniteľné zariadenie, resp. teda na pyrdp MITM server – viz. **Obrázok 49**. Pripájame sa zo zariadenia s operačným systémom MacOS a teda zadáme prihlasovacie meno a heslo, ktoré je nám v tomto prípade známe, aby sme mohli simulovať priebeh pripojenia.
3. Vytvoríme textový súbor s názvom *test.txt* a pridáme do neho niekoľko riadkov.



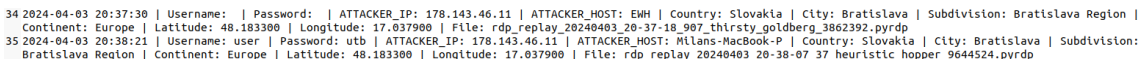
Obrázok 56 - Testovanie - PyRDP - Pripojenie, Zdroj: vlastný

4. Uzavrieme pripojenie.
5. Postup opakujeme aj z iného zariadenia s pripojeným prenosným médiom.

10.2.2 Z pohľadu honeypotu

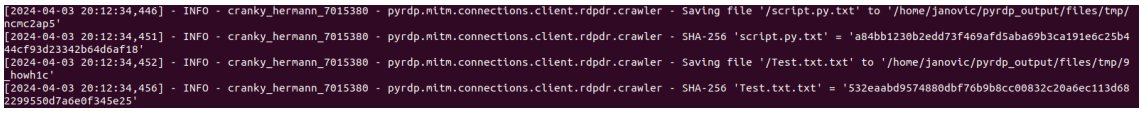
1. Spustíme pyrdp mitm server, aby útočník mohol naviazať spojenie – viz. **8.1.2.1**.
2. Útočník naviaže spojenie a pyrdp vytvorí v zložke */home/janovic/pyrdp_output/replays* súbor obsahujúci informácie o novom pripojení. Skript *watcher.sh* následne extrahuje potrebné informácie, zostaví záznam,

ktorý sa bude ukladať do logov a obsah mailu na odoslanie. Následne mail (viz. **Podkapitola 8.1.1** a **Obrázok 47** - PyRDP emailová notifikácia, Zdroj: vlastný) odošle a záznam uloží do predpripraveného logovacieho súboru – viz. **Podkapitola 8.1.2.2**.



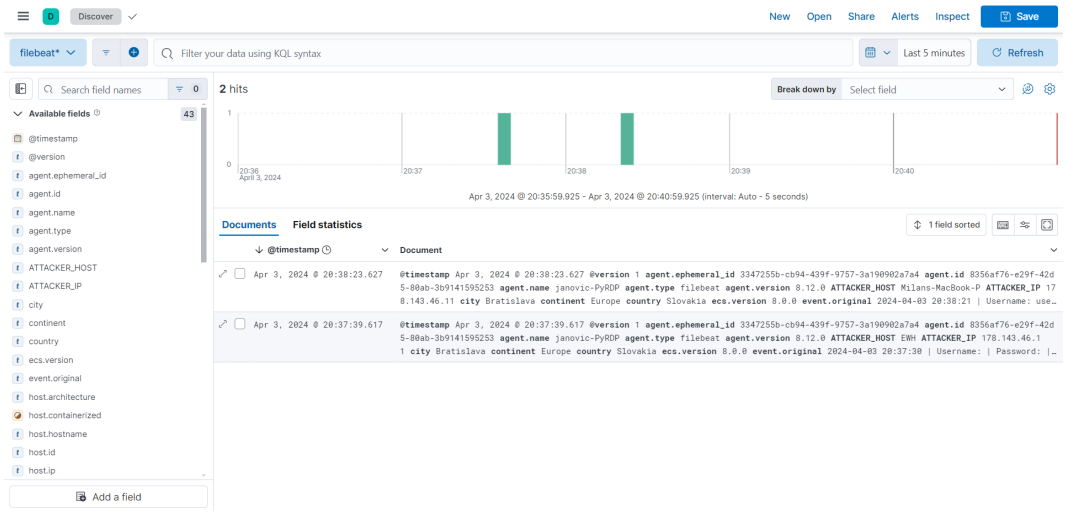
Obrázok 57 - Testovanie - PyRDP - Log, Zdroj: vlastný

2.1. V prípade, že útočník pripojí k honeypotu externé pamäťové médiu pyrdp server automaticky stiahne a uloží jeho obsah (predpokladáme spustenie v *crawl* móde). V našom prípade sa na pripojenom médiu nachádzali dva testovacie súbory *script.py.txt* a *test.txt.txt* – viz. **Obrázok 58**. Po spracovaní sú súbory dostupné v *home/janovic/pyrdp_output/files* pod HASH hodnotami, ktoré sú pridelené pyrdp serverom a viditeľné nižšie.



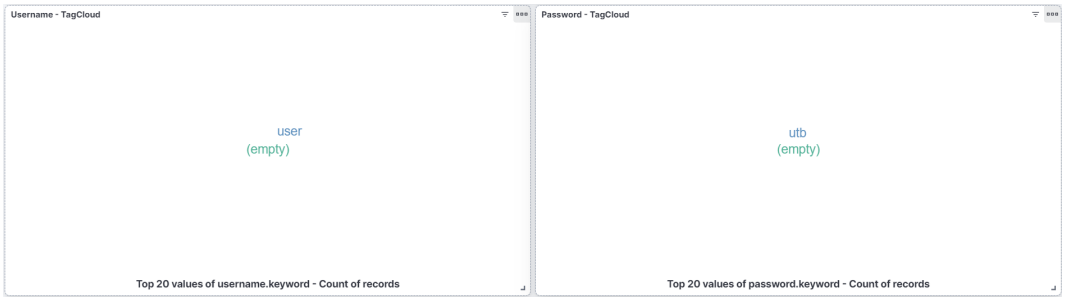
Obrázok 58 - Testovanie - PyRDP - Crawl, Zdroj: vlastný

3. Lokálna inštalácia Filebeat protokolu (viz. **Podkapitola 8.1.4**) zaznamená, že do logov pribudol nový záznam, ktorý odošle do Logstash inštancie na virtuálnom systéme ELK, cez zabezpečený WireGuard tunel, rovnako ako v predchádzajúcom prípade.
4. Logy sú po spracovaní odoslané do Elasticsearch inštancie na rovnakom virtuálnom systéme (ELK).
5. Postup sa opakuje pre všetky RDP pripojenia.
6. Logy následne môžeme vidieť v Kibana rozhraní *Discover*, ako aj v predpripravenom dashboarde *PyRDP* – viz. **Obrázok 60**, **Obrázok 61** a **Obrázok 61**.



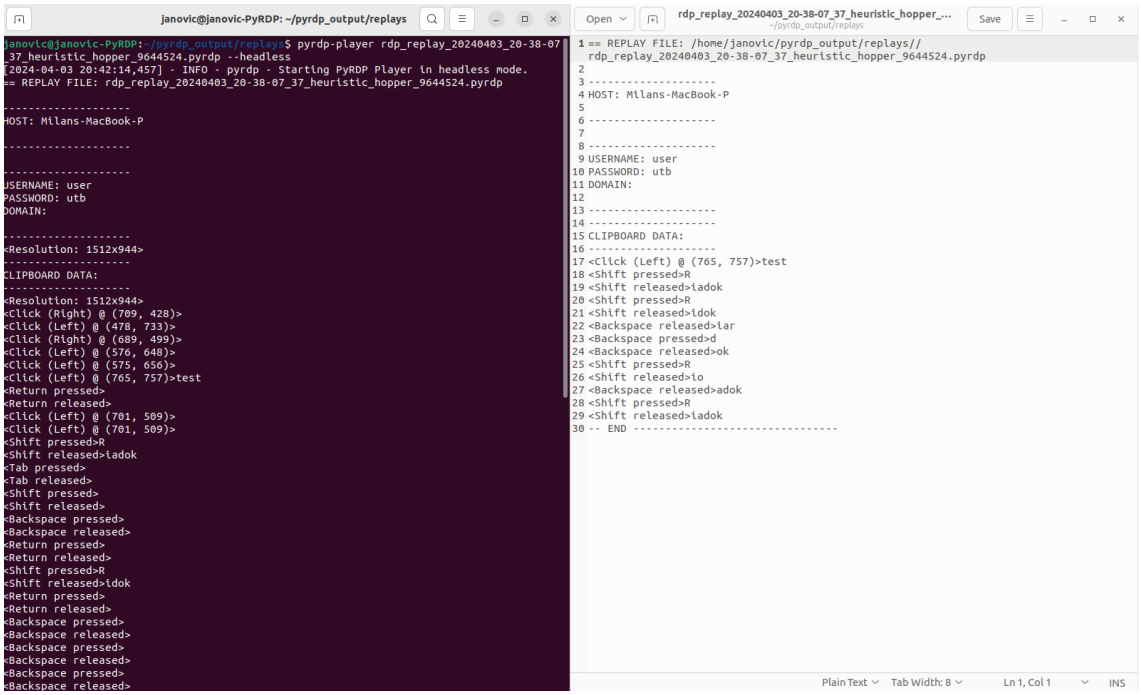
Obrázok 59 - Testovanie - PyRDP - Kibana Discover, Zdroj: vlastný

Obrázok 60 - Testovanie - PyRDP - Kibana Dashboard, Zdroj: vlastný



Obrázok 61 - Testovanie - PyRDP - Kibana Dashboard 2, Zdroj: vlastný

Rozdiel medzi logmi, ktoré získame pomocou nástroja *pyrdp-player* v tzv. *headless* móde a logmi spracovanými pomocou nášho *strip.py* skriptu (viz. **8.1.2.4**) zobrazuje **Obrázok 62** nižšie. V prípade, že by sme chceli prehrať záznam z daného pripojenia, použili by sme *pyrdp-player* **bez** *headless* módu.



Obrázok 62 - Testovanie - PyRDP - Spracované logy - Strip.py, Zdroj: vlastný

10.3 T-Pot

T-Pot projekt obsahuje v konfigurácii (*standard.yml,* resp. *tpot.yml*), tak ako sme ju popísali v **Podkapitole 9.1.3**, spolu 16 aktívnych honeypotov. V aktuálnej kapitole budeme, rovnako ako v predchádzajúcich dvoch podkapitolách, demonštrovať útok na vybrané honeypoty, a teda základy ich funkcionality. Logy uvádzané v nasledujúcich podkapitolách sa vo virtuálnom systéme *DP_Janovic_tpot* nachádzajú v priečinkoch, ktoré nasledujú jednoduchú schému */data/<nazov_honeypotu>/log/<subor_logov>.typ* teda napríklad pre honeypot *tanner* by sme dostali */data/tanner/log/tanner_report.json*

10.3.1 Ddospot

10.3.1.1 Z pohľadu útočníka

1. Zvolíme typ honeypotu, na ktorý budeme útočiť z Ddospot projektu – v našom prípade DNSPot na porte 53.
2. Honeypotu zašleme niekoľko predpripravených DNS dotazov. Skutočný útočník by pochopiteľne zasielal dotazy opakovane.

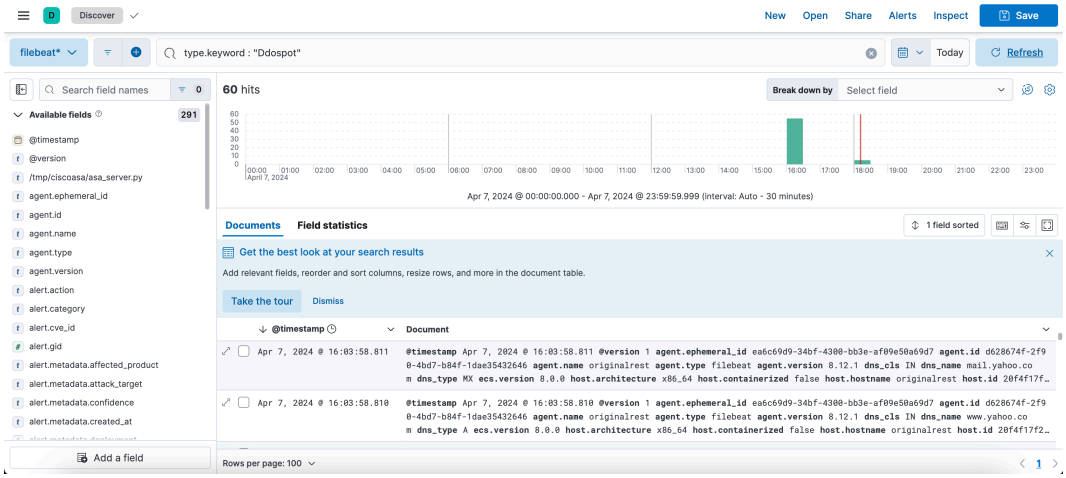
10.3.1.2 Z pohľadu honeypotu

1. Honeypot obdrží DNS dotazy od útočníka a zapíše ich do logov – viz. **Obrázok 63**.

```
{"src_ip": "192.168.100.36", "src_port": 47057, "opcode": 0, "dns_name": "google.com", "dns_type": "A", "dns_cls": "IN", "time": "2024-04-07 16:03:58.795497"}
New attack started for quartet ('192.168.100.36', 'www.google.com', 'A', 'IN')
{"src_ip": "192.168.100.36", "src_port": 47057, "opcode": 0, "dns_name": "www.google.com", "dns_type": "A", "dns_cls": "IN", "time": "2024-04-07 16:03:58.799785"}
{"src_ip": "192.168.100.36", "src_port": 47057, "opcode": 0, "dns_name": "mail.google.com", "dns_type": "MX", "dns_cls": "IN", "time": "2024-04-07 16:03:58.801952"}
{"src_ip": "192.168.100.36", "src_port": 47057, "opcode": 0, "dns_name": "facebook.com", "dns_type": "A", "dns_cls": "IN", "time": "2024-04-07 16:03:58.805092"}
{"src_ip": "192.168.100.36", "src_port": 47057, "opcode": 0, "dns_name": "www.facebook.com", "dns_type": "A", "dns_cls": "IN", "time": "2024-04-07 16:03:58.806466"}
{"src_ip": "192.168.100.36", "src_port": 47057, "opcode": 0, "dns_name": "mail.facebook.com", "dns_type": "MX", "dns_cls": "IN", "time": "2024-04-07 16:03:58.807294"}
{"src_ip": "192.168.100.36", "src_port": 47057, "opcode": 0, "dns_name": "yahoo.com", "dns_type": "A", "dns_cls": "IN", "time": "2024-04-07 16:03:58.807891"}
{"src_ip": "192.168.100.36", "src_port": 47057, "opcode": 0, "dns_name": "www.yahoo.com", "dns_type": "A", "dns_cls": "IN", "time": "2024-04-07 16:03:58.810563"}
{"src_ip": "192.168.100.36", "src_port": 47057, "opcode": 0, "dns_name": "mail.yahoo.com", "dns_type": "MX", "dns_cls": "IN", "time": "2024-04-07 16:03:58.811082"}
```


Obrázok 63 - Testovanie - T-Pot - Ddospot - Logy, Zdroj: vlastný

2. Logy sú následne prevzaté pomocou FileBeat protokolu a zaslané do Logstash inštancie na virtuálnom systéme ELK (viz. **Podkapitola 6.1.2.3**) – prostredníctvom zabezpečeného WireGuard tunela, spracované a uložené do Elasticsearch. Rovnako ako v prechádzajúcich dvoch prípadoch VS WEB-POT (viz. **Podkapitola 10.1**) a PyRDP (viz. **Podkapitola 10.2**), resp. honeypotov, ktoré implementujú.
3. Logy následne môžeme vidieť v Kibana rozhraní *Discover* (viz. **Obrázok 64**), ako aj v predpripravenom dashboarde *Ddospot*.

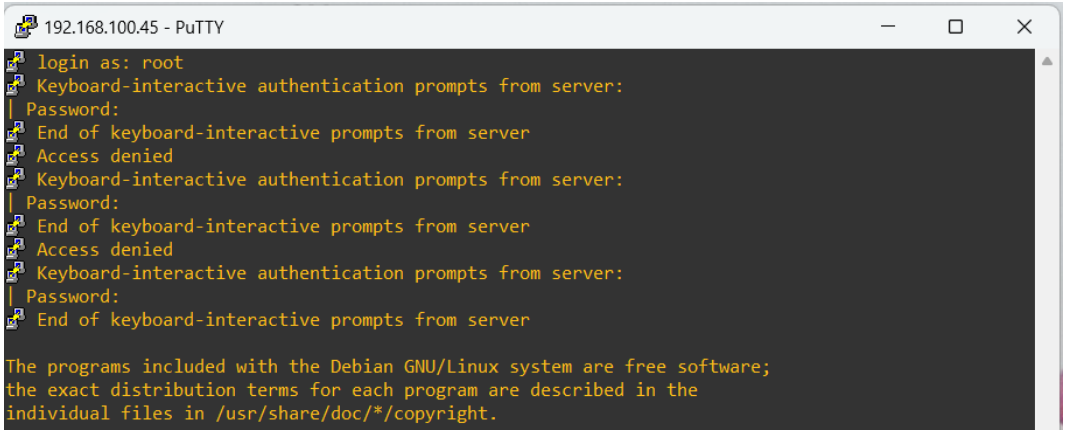


Obrázok 64 - Testovanie - T-Pot - Ddospot – Kibana Discover, Zdroj: vlastný

10.3.2 Cowrie

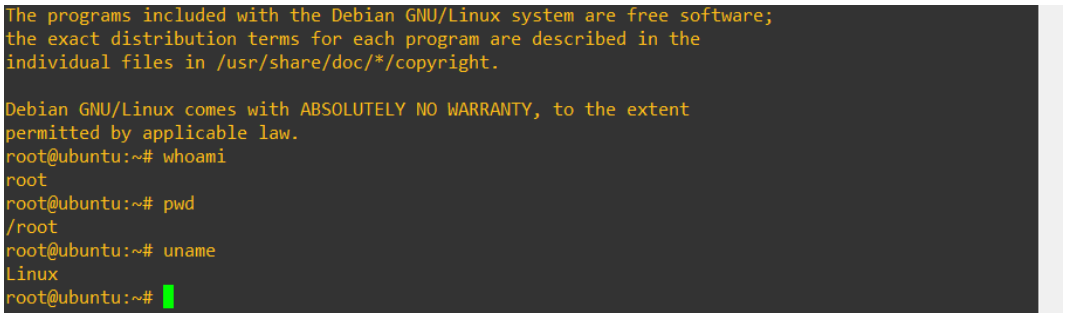
10.3.2.1 Z pohľadu útočníka

1. Identifikujeme zariadenie, na ktoré budeme útočiť. Skutočný útočník by pravdepodobne vykonal nejakú formu skenovania, pomocou ktorej by dané zariadenie odhalil (napr. nmap). V našom prípade na testovanie IP adresu zariadenia poznáme.
2. Pripojíme sa na jeden z portov, ktoré Cowrie honeypot poskytuje – v našom prípade port 22. Zadáme niekoľko hesiel, vzhľadom k tomu, že Cowrie imituje validitu prihlásenia a sme prihlásení – viz. **Obrázok 65** - Testovanie - T-Pot - Cowrie - Pripojenie, Zdroj: vlastný



Obrázok 65 - Testovanie - T-Pot - Cowrie - Pripojenie, Zdroj: vlastný

3. Následne zadáme zopár základných príkazov na ilustráciu – viz. **Obrázok 66**.

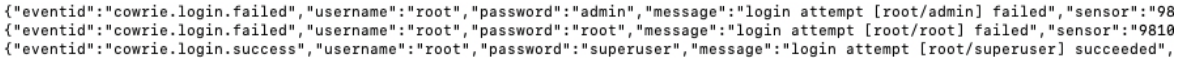


Obrázok 66 - Testovanie - T-Pot - Cowrie - Prihlásenie, Zdroj: vlastný

4. Uzavrieme pripojenie.

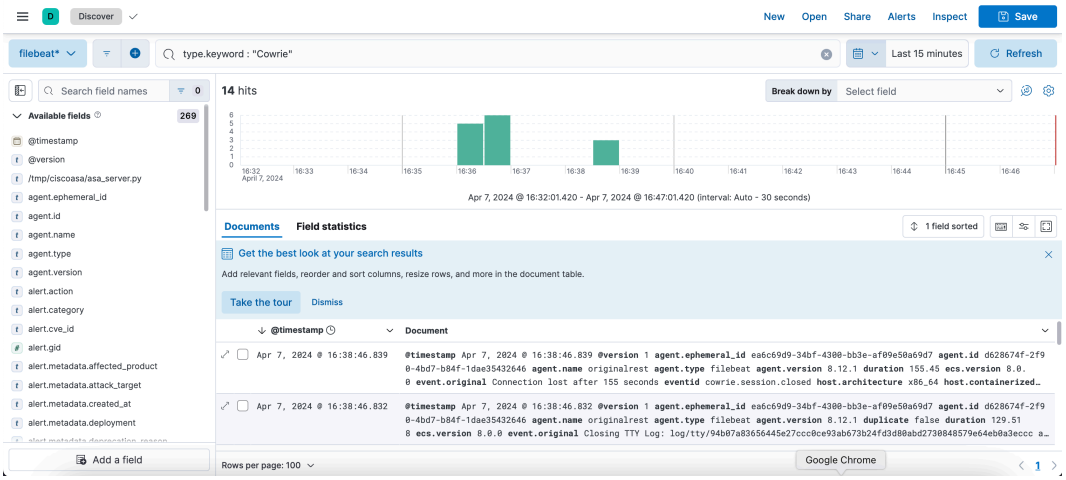
10.3.2.2 Z pohľadu honeypotu

1. Po pripojení útočníka Cowrie začne logovať pripojenie a zadané prihlasovacie údaje – viz. **Obrázok 67**.

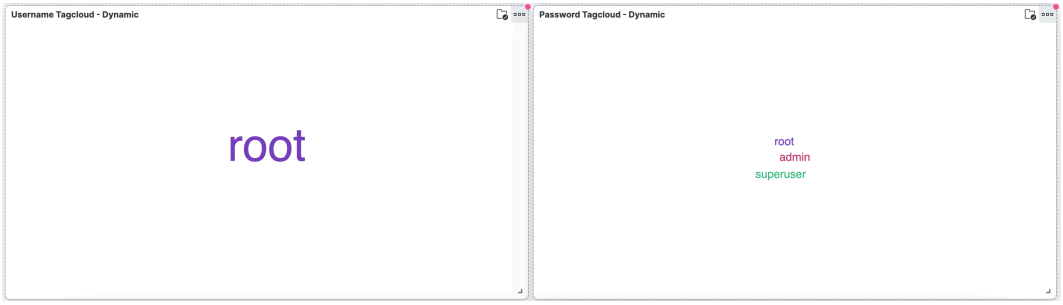


Obrázok 67 - Testovanie - T-Pot - Cowrie - Logy, Zdroj: vlastný

2. Logy sú následne prevzaté pomocou FileBeat protokolu a zaslané do Logstash inštancie na virtuálnom systéme ELK (viz. **Podkapitola 6.1.2.3**) – prostredníctvom zabezpečeného WireGuard tunela, spracované a uložené do Elasticsearch. Rovnako ako v prechádzajúcich prípadoch.
3. Logy následne môžeme vidieť v Kibana rozhraní *Discover*, ako aj v predpripravenom dashboarde *PyRDP* – viz. **Obrázok 68** - Testovanie - T-Pot - Cowrie - Kibana Discover, Zdroj: vlastný a **Obrázok 69**.



Obrázok 68 - Testovanie - T-Pot - Cowrie - Kibana Discover, Zdroj: vlastný

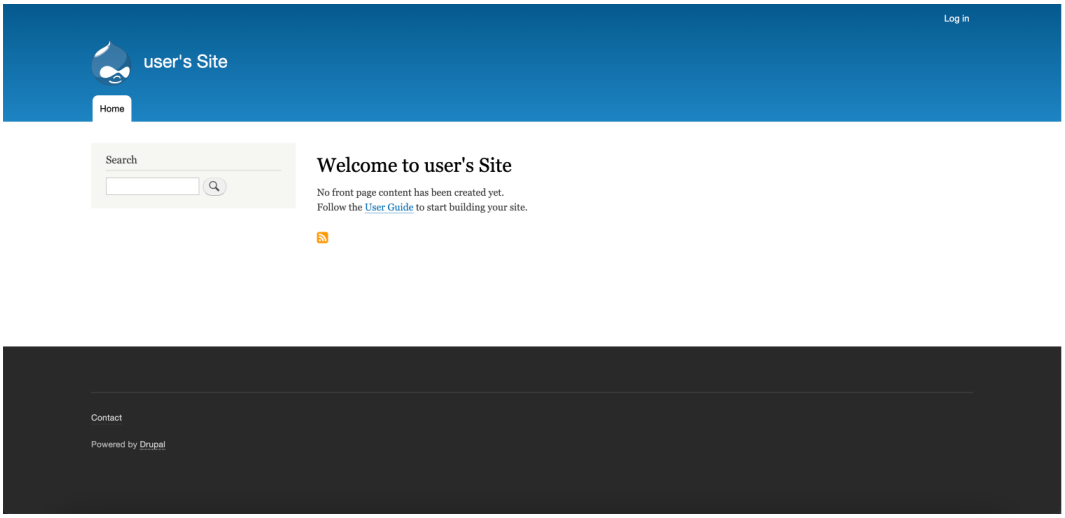


Obrázok 69 - Testovanie - T-Pot - Cowrie - Kibana Dashboard, Zdroj: vlastný

10.3.3 Tanner

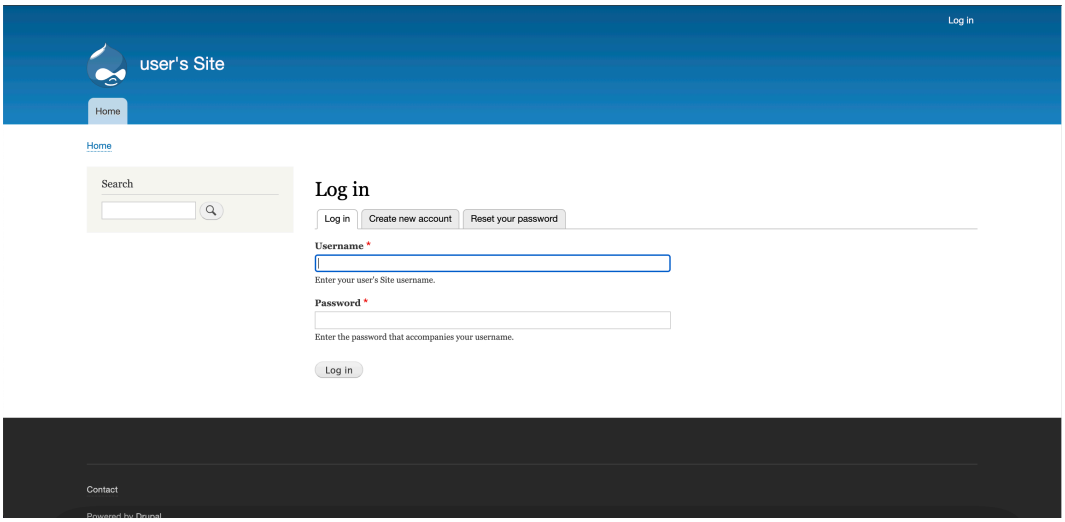
10.3.3.1 Z pohľadu útočníka

1. Prejdeme na adresu Tanner webového honeypotu (viz. **Obrázok 70**). Tanner honeypot môže po každom štarte vygenerovať inú webovú stránku, takže sa môže, ale nemusí zhodovať s nami zobrazenou.



Obrázok 70 - Testovanie - T-Pot - Tanner, Zdroj: vlastný

2. V pravom hornom rohu klikneme na "Log in", zadáme prihlasovacie meno (*tanner-test-meno*) a heslo (*tanner-test-heslo*) a odošleme. Formulár sa vyprázdni, ale prihlásení nie sme (viz. **Obrázok 71**).



Obrázok 71 - Testovanie - T-Pot - Tanner - Prihlásenie, Zdroj: vlastný

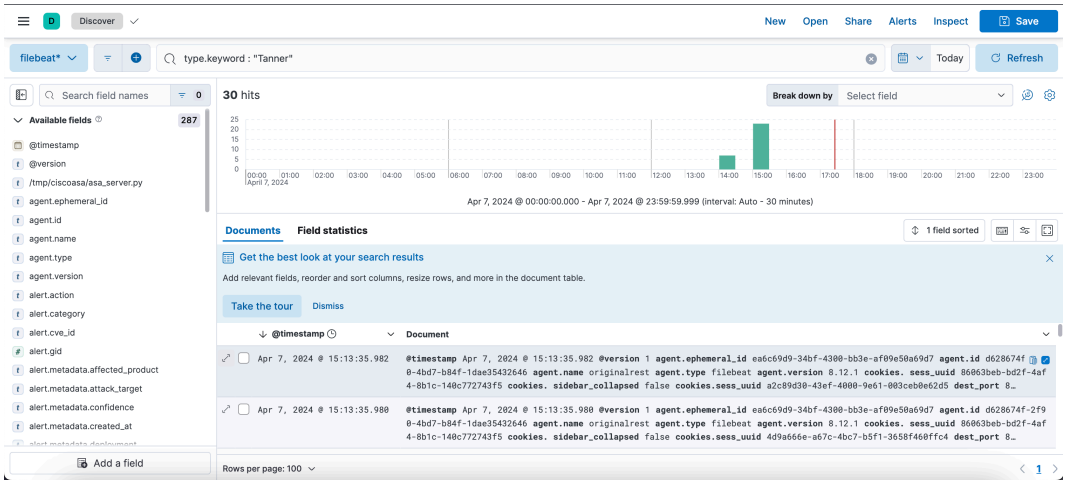
10.3.3.2 Z pohľadu honeypotu

1. Po pripojení útočníka začne Tanner logovať prístup, resp. GET požiadavky klienta útočníka.
2. Rovnako zaznamená prístup na prihlasovací formulár a zadané prihlasovacie meno a heslo, ktoré uloží do logov – viz **Obrázok 72**.

{"method": "POST", "path": "/user/login", "headers": {"host": "192.168.100.45", "connection": "keep-alive", "content-length": "141", "cache-control": "max-age=0", "upgrade-insecure-requests": "1", "origin": "http://192.168.100.45", "content-type": "application/x-www-form-urlencoded", "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36", "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7", "referer": "http://192.168.100.45/user/login", "accept-encoding": "gzip, deflate", "accept-language": "sk-SK,sk;q=0.9,cs;q=0.8,en-US;q=0.7,en;q=0.6", "cookie": "sess_uuid=86063beb-bd2f-4af4-8b1c-140c772743f5; sidebar_collapsed=false; sess_uuid=86063beb-bd2f-4af4-8b1c-140c772743f5"}, "uuid": "d3d2e650-9b5b-439e-9ed3-ca545373d77f", "peer": {"ip": "192.168.100.5", "port": 57780}, "status": 200, "cookies": {"sess_uuid": "86063beb-bd2f-4af4-8b1c-140c772743f5", " sidebar_collapsed": "false", " sess_uuid": "86063beb-bd2f-4af4-8b1c-140c772743f5"}, "post_data": {"name": "tanner-test-meno", "pass": "tanner-test-heslo", "form_build_id": "form-M2r5d8Vq_yCtNdWLTBoFt8672T928cd-5O1aKeNmS-U", "form_id": "user_login_form", "op": "Log in"}, "response_msg": {"version": "0.6.0", "response": {"message": {"detection": {"name": "unknown", "order": 0, "type": 1, "version": "0.6.0"}, "sess_uuid": "7adb716c-56f4-4418-a51f-3703e991cabb"}}}, "timestamp": "2024-04-07T15:13:35.882940"}

Obrázok 72 - Testovanie - T-Pot - Tanner - Logy, Zdroj: vlastný

3. Logy sú následne prevzaté pomocou FileBeat protokolu a zaslané do Logstash inštancie na virtuálnom systéme ELK (viz. **Podkapitola 6.1.2.3**) – prostredníctvom zabezpečeného WireGuard tunela, spracované a uložené do Elasticsearch. Rovnako ako v prechádzajúcich prípadoch.
4. Logy následne môžeme vidieť v Kibana rozhraní *Discover*, ako aj v predpripravenom dashboarde *PyRDP* – viz. **Obrázok 73**.



Obrázok 73 - Testovanie - T-Pot - Tanner - Kibana Discover, Zdroj: vlastný

10.3.4 Hail Mary

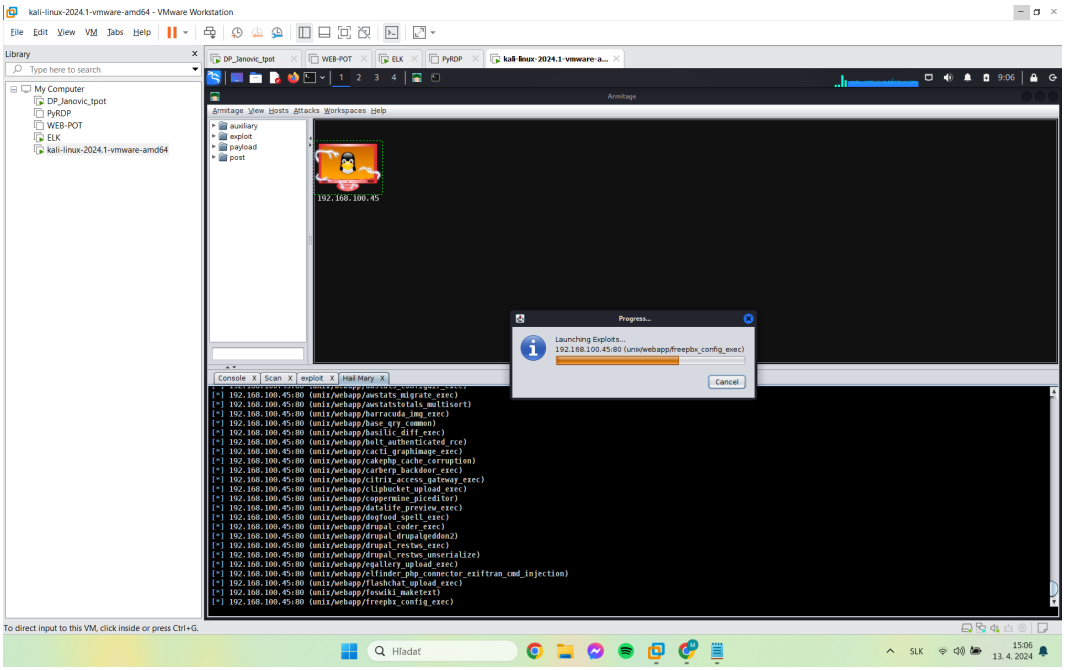
Hail Mary útok je typ útoku, kedy sa nesnažíme potichu a precízne zistiť zraniteľnosť daného systému za účelom jeho napadnutia, ale jednoducho použijeme všetky dostupné zraniteľnosti (z angl. *exploit*) s vierou, že niektorá (alebo viaceré) z nich bude účinná. V našom prípade na tento účel využijeme framework od spoločnosti Rapid7, tzv. *metasploit*, ktorý sa používa na penetračné testovanie, spolu s jeho grafickou nadstavbou, tzv. *Armitage* na operačnom systéme *Kali Linux*.

10.3.4.1 Z pohľadu útočníka

1. Po inicializácii metasploit frameworku, databázy zraniteľnosti a zapnutí Armitage, doň zadáme IP adresu zariadenia, na ktoré budeme útočiť – v našom prípade lokálna

IP adresa nášho T-Pot projektu, teda 192.168.100.45. Opäť platí, že reálny útočník by IP adresu zariadenia musel najskôr získať, napr. pomocou skenovania siete a pod.

2. Následne v Armitage zvolíme Hail Mary útok a potvrdíme jeho spustenie na zvolený cieľ. Armitage, resp. metasploit framework teda na zariadenie zaútočí všetkými zraniteľnosťami, ktoré sa v jeho databáze nachádzajú – viz. **Obrázok 74**.

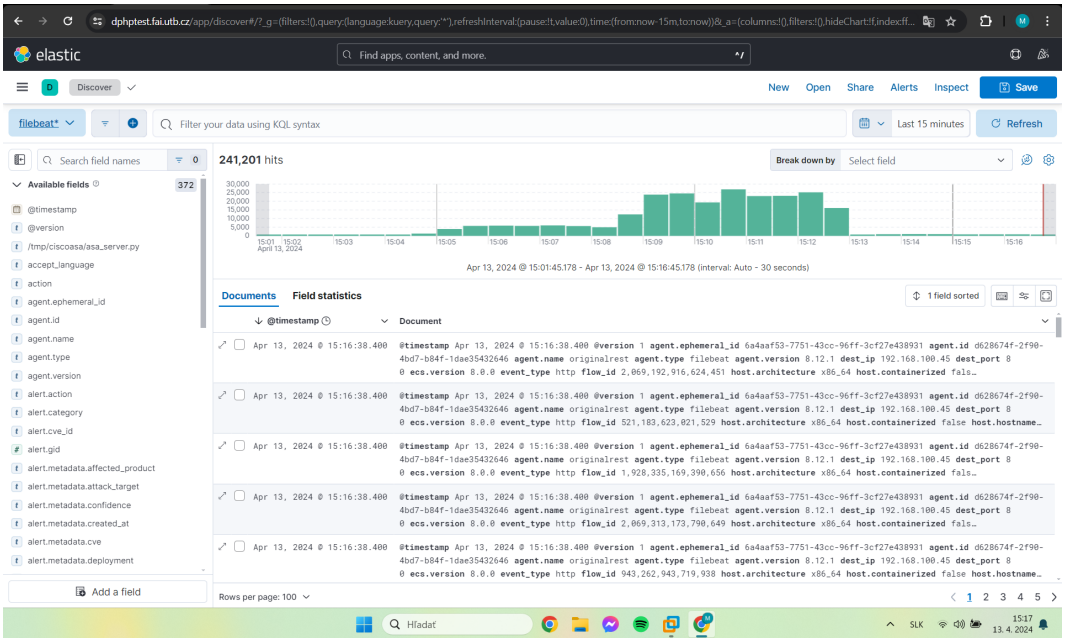


Obrázok 74 - Testovanie - T-Pot - Hail Mary - Armitage, Zdroj: vlastný

3. Reálny útočník by po skončení útoku pravdepodobne pokračoval v útoku pomocou zraniteľností, ktoré boli úspešné pri napadnutí systému.

10.3.4.2 Z pohľadu honeypotu

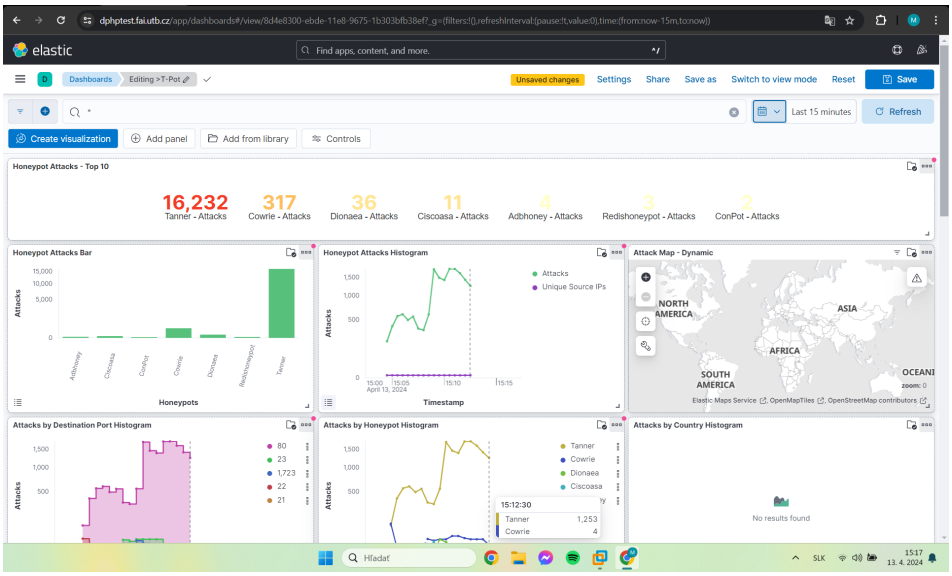
1. Po skončení útoku môžeme pozorovať v rozhraní Kibana *Discover* enormný nárast aktivity, resp. počtu eventov (viz. **Obrázok 75**). Tento počet sa ale nezhoduje s reálnym počtom útokov na honeypoty, čo ukážeme v ďalších bodoch, vzhľadom k tomu, že sa sem radia aj záznamy nástrojov monitorujúcich sieťovú prevádzku, ktoré pri takto masívnom útoku zaznamenajú veľké množstvo škodlivej komunikácie atď.



Obrázok 75 - Testovanie - T-Pot - Hail Mary - Kibana Discover, Zdroj: vlastný

2. Počet útokov, ktoré zaznamenali jednotlivé honeypoty potom zobrazuje predpripravený *Dashboard T-Pot*.

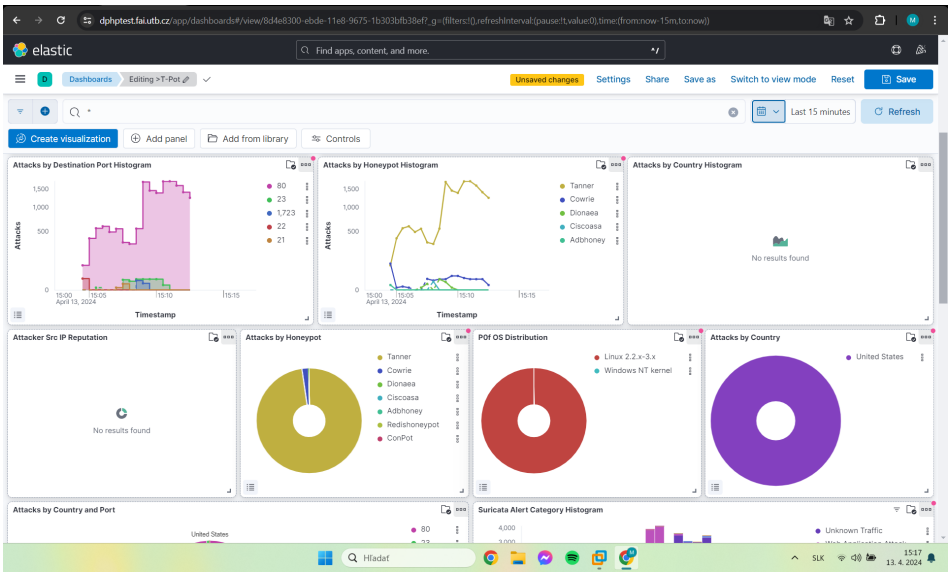
2.1. **Obrázok 76** zobrazuje najmä rozdelenie útokov medzi jednotlivé honeypoty, histogramov útokov v čase – kde môžeme vidieť nárast v čase keď sme Hail Mary útok spustili.



Obrázok 76 - Testovanie - T-Pot - Hail Mary - Kibana Dashboard, Zdroj: vlastný

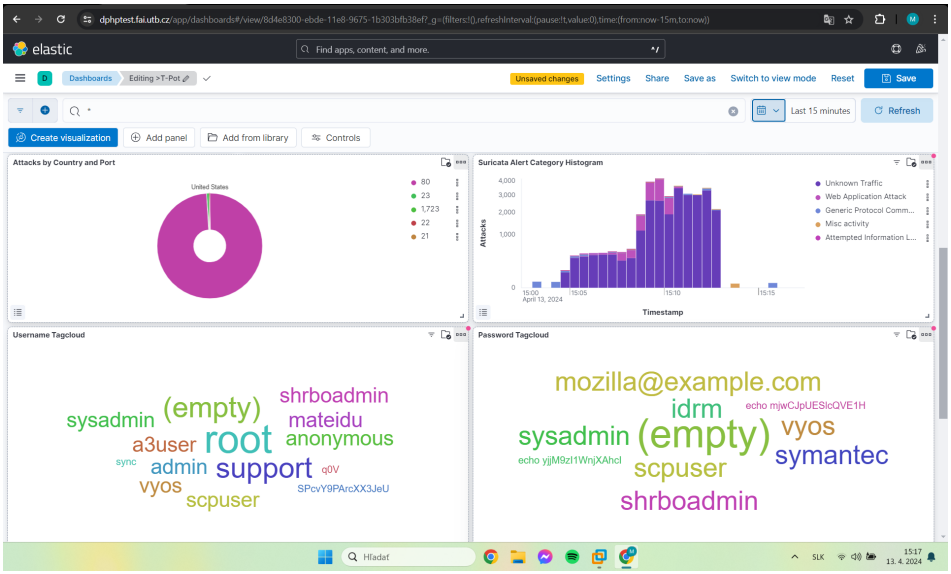
2.2. **Obrázok 77** zobrazuje najmä rozdelenie útokov medzi jednotlivé porty, operačné systémy, z ktorý sa útočilo a krajinu, z ktorej sa útočilo (v testovacej infraštruktúre sme použili ručne zadanú IP adresu s geolokáciou

v USA, vzhľadom k tomu, že lokálne IP adresy nie je, z pochopiteľných dôvodov, možné lokalizovať).



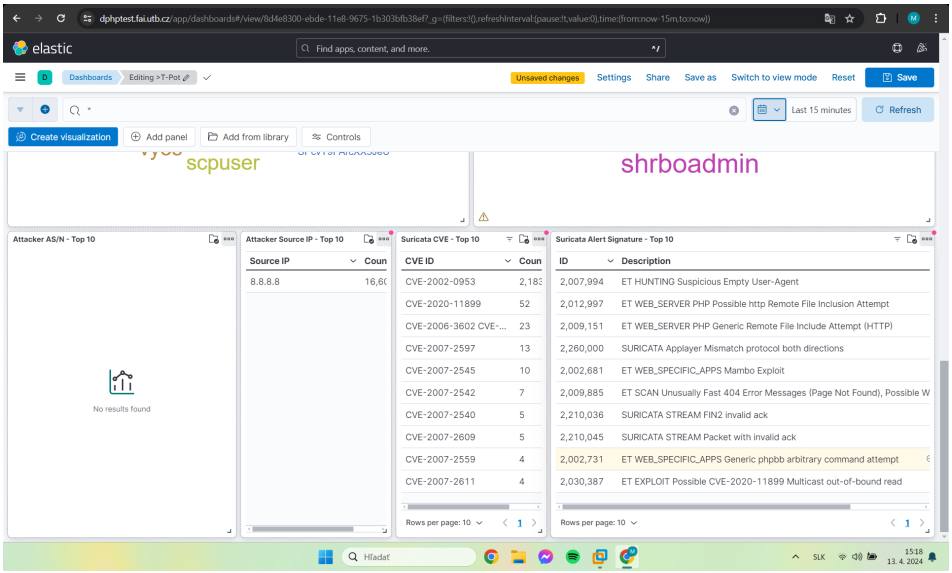
Obrázok 77 - Testovanie - T-Pot - Hail Mary-Kibana Dashboard 2, Zdroj: vlastný

2.3. **Obrázok 78** zobrazuje najmä používateľské mená a heslá, ktoré pri útokoch Armitage použil v snahe napadnúť systém.



Obrázok 78 - Testovanie - T-Pot - Hail Mary-Kibana Dashboard 3, Zdroj: vlastný

2.4. **Obrázok 79** zobrazuje najmä CVE a Suricata signatúry, ktoré boli pri útokoch zistené. V neposledom rade môžeme vidieť aj ilustračnú IP adresu – 8.8.8.8, ktorá ale v žiadnom prípade **nie je** adresou útočníka. Adresa bola použitá, pretože je to dobre a verejne známa IP adresa (DNS Server spoločnosti Google) a je na nej zaručená presnosť geolokácie.



Obrázok 79 - Testovanie - T-Pot - Hail Mary-Kibana Dashboard 4, Zdroj: vlastný

11 ĎALŠÍ VÝVOJ A ÚPRAVY

Rovnako ako v prípade akejkoľvek aplikácie, služby, či projektu, ani náš projekt nie je v žiadnom prípade dokonalý a jeho vývoj by mohol pokračovať po niekoľko stoviek hodín. Okrem všade prítomnej potreby optimalizácie slúžiacej na zrýchlenie projektu a zefektívnenia využívania zdrojov, táto kapitola uvádza niekoľko návrhov na rozšírenie funkcionality projektu a iné úpravy:

1. Sfunkčnenie služieb máp z pôvodného projektu T-Pot – tento krok si vyžaduje úpravu Logstash kódu na virtuálnom systéme ELK. Vzhľadom k tomu, že originálna implementácia projektu T-Pot sa spolieha na fakt, že Logstash inštancia sa nachádza na rovnakom zariadení ako honeypoty.
2. Implementácia služieb máp aj pre webový a RDP honeypot po ich sfunkčnení.
3. Prechod z desktopový verzií operačného systému Ubuntu na serverovú verziu.
4. Pridanie viacerých falošných webový stránok, resp. webových honeypotov.
5. „Fine Tunnig“ T-Pot projektu ako takého, tak aj pre potreby danej infraštruktúry.

ZÁVER

Cieľom tejto diplomovej práce, tak ako to ostatne uvádzajú aj body zadania, bolo uviesť čitateľa do problematiky honeypotov a honeynetov. Zvoliť vhodný honeypot na implementáciu do testovacej infraštruktúry, navrhnúť a implementovať samotný honeypot a v neposledom rade ho otestovať, resp. jeho reakciu na útoky.

V teoretickej časti sme preto čitateľa uviedli do problematiky honeypotov (viz. **Kapitola 2**), a objasnili niektoré základné pojmy z oblasti IT (viz **Kapitola 1**), ktoré mohli byť pre čitateľa užitočné na jednoduchšie a rýchlejšie pochopenie práce.

V praktickej časti práce sme v prvom rade prezentovali náš úvodný pokus, v ktorom sme v verejnom cloudovom prostredí implementovali T-Pot projekt – bez akýchkoľvek zmien a sledovali sme záujem a chovanie útočníkov (viz. **Kapitola 3**). Následne sme čitateľovi priblížili technológie, ktoré sme v práci použili na implementáciu samotných honeypotov, zber logov, zabezpečenie komunikácie atď. – viz. **Kapitola 4**. Ďalej sme zatiaľ nadobudnuté poznatky zhrnuli a predstavili nami navrhované riešenie (viz. **Kapitola 5**). V tomto bode sa dostávame k implementácii samotných honeypotov. V nasledujúcich kapitolách sme čitateľovi, čo možno najpodrobnejšie, avšak so snahou nezaoberať sa zbytočnými detailmi, priblížili implementáciu troch honeypotov, resp. dvoch honeypotov a jedného honeypot projektu, a jedného systému slúžiaceho na príjem, uloženie, spracovanie a vizualizáciu logov. V **Kapitole 6** sme teda najskôr čitateľovi objasnili implementáciu systému slúžiaceho na príjem, uloženie, spracovanie a vizualizáciu logov – pomocou tzv. ELK stack. Následne sme v **Kapitole 7** objasnili implementáciu nášho webového honeypotu a v **Kapitole 8** implementáciu RDP honeypotu. V **Kapitole 9** sme potom demonštrovali implementáciu upravenej verzie T-Pot honeypot projektu.

Kapitola 10 potom čitateľovi v stručnosti priblížila odozvu honeypotov na útoky. Náš webový honeypot a RDP honeypot sa v tejto kapitole dočkali väčšej pozornosti vzhľadom k tomu, že sa jednalo o našu implementáciu, nie „out-of-the-box“ riešenie, ako v prípade honeypot projektu T-Pot – kde zohrávala úlohu aj jeho komplexnosť.

V neposlednom rade **Kapitola 11** uvádza niektoré návrhy a možnosti úprav na budúci vývoj projektu.

ZOZNAM POUŽITEJ LITERATÚRY

[1] OLEJÁR, Daniel, Jozef ANDRAŠKO, Lenka GONDOLA, et al. UNIVERZITA KOMENSKÉHO V BRATISLAVE. *Základy kybernetickej a informačnej bezpečnosti*. 2020.

[2] ČANDÍK, Marek. *Základy informační bezpečnosti*. Zlín: Univerzita Tomáše Bati ve Zlíně, 2004. ISBN 80-731-8218-1.

[3] What is Cyber Security? *Kaspersky Lab* [online]. 2024 [cit. 2024-02-23]. Dostupné z: https://www.kaspersky.com/resource-center/definitions/what-is-cyber-security

[4] KOLOUCH, Jan. *CyberCrime* [online]. Praha: Edice CZ.NIC, 2016 [cit. 2024-02-23]. ISBN 978-80-88168-15-4. Dostupné z: https://www.bookport.cz/e-kniha/cybercrime-1696887/

[5] KOLOUCH, Jan a Pavel BAŠTA. *CyberSecurity*. Praha: Edice CZ.NIC, 2019. ISBN 978-80-88168-34-8.

[6] VONDRUŠKA, Pavel. *Kryptologie, šifrování a tajná písma*. Praha: Albatros, 2006. ISBN 8000018888.

[7] PAAR, Christof a Jan PELZL. *Understanding cryptography a textbook for students and practitioners*. Berlin: Springer, 2010. ISBN 978-3-642-04101-3.

[8] OXFORD UNIVERSITY PRESS. Network. *Oxford University Press* [online]. 2024 [cit. 2024-02-24]. Dostupné z: https://www.oxfordlearnersdictionaries.com/definition/english/network_1

[9] COMPTIA, INC. What Is a Network Protocol, and How Does It Work? *CompTIA* [online]. 2024 [cit. 2024-02-25]. Dostupné z: https://www.comptia.org/content/guides/what-is-a-network-protocol

[10] STALLINGS, William a Lawrie BROWN. *Computer security : principles and practice*. 4. vydanie. Pearson, 2020. ISBN 978-93-534-3886-9.

[11] FIELDING, R., J. GETTYS, J. MOGUL, H. FRYSTIK, L. MASINTER, P. LEACH a T. BERNERS-LEE. RFC 2616. *Datatracker.ietf.org* [online]. 1999 [cit. 2024-02-25]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc2616

[12] RESCORLA, E. RFC 2818. *Datatracker.ietf.org* [online]. 2000 [cit. 2024-02-25]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc2818

[13] LAITH ACADEMY. HTTPS, SSL, TLS & Certificate Authority Explained. *Youtube* [online]. 2023 [cit. 2024-02-25]. Dostupné z: https://www.youtube.com/watch?v=EnY6fSng3Ew

[14] MICROSOFT. Understanding the Remote Desktop Protocol (RDP). *Learn.microsoft.com* [online]. 2023 [cit. 2024-02-25]. Dostupné z: https://learn.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol

[15] SOLARWINDS. What Is Remote Desktop Protocol? *SolarWinds Worldwide, LLC* [online]. 2024 [cit. 2024-02-25]. Dostupné z: https://www.solarwinds.com/resources/it-glossary/remote-desktop-protocol

[16] VOJTĚŠEK, Jiří. *Internet a jeho služby* [online]. Zlín: Univerzita Tomáše Bati ve Zlíně, 2012 [cit. 2024-02-25]. ISBN 9788074542176. Dostupné z: https://digilib.k.utb.cz/handle/10563/18588

[17] IANA. Service Name and Transport Protocol Port Number Registry. *Iana.org* [online]. 2024 [cit. 2024-02-25]. Dostupné z: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[18] IBM. What are virtual machines (VMs)? *IBM.com* [online]. 2024 [cit. 2024-02-26]. Dostupné z: https://www.ibm.com/topics/virtual-machines

[19] GRIMES, Roger A. *Hacking the Hacker* [online]. Indianapolis: John Wiley & Sons, 2017 [cit. 2024-02-26]. ISBN 978-1-119-39623-9. Dostupné z: https://onlinelibrary-wiley-com.proxy.k.utb.cz/doi/book/10.1002/9781119396260

[20] ADARSH, S. a Jain KURUNANDAN. *Capturing Attacker Identity with Biteback Honeypot* [online]. International Conference on System, Computation, Automation and Networking (ICSCAN), 2021 [cit. 2024-04-25]. Dostupné z: doi:10.1109/ICSCAN53069.2021.9526371

[21] BUZZIO-GARCIA, Jorge. *Creation of a High-Interaction Honeypot System based-on Docker containers.* [online]. Fifth World Conference on Smart Trends in Systems Security and Sustainability, 2021 [cit. 2024-02-26]. Dostupné z: doi:10.1109/WorldS451998.2021.9514022

[22] WANG, Ping a Hubert D'CRUZE. *Honeypots and knowledge for network defence* [online]. 2021 [cit. 2024-02-26]. Dostupné z: doi:10.48009/3_iis_2021_259-272

[23] OKTA. What Is a Honeynet? Definition, Usage & the Honeynet Project. *Okta.com* [online]. 2022 [cit. 2024-02-27]. Dostupné z: https://www.okta.com/identity-101/honeynet/

[24] DEUTSCHE TELEKOM SECURITY GMBH. Tpotce. DEUTSCHE TELEKOM SECURITY GMBH. *T-Pot - The All In One Multi Honeypot Platform* [online]. 2022 [cit. 2024-03-02]. Dostupné z: https://github.com/telekom-security/tpotce

[25] ARIN. Autonomous System Numbers. *Arin.net* [online]. 2024 [cit. 2024-03-01]. Dostupné z: https://www.arin.net/resources/guide/asn/

[26] OISF. Suricata. *Suricata.io* [online]. 2024 [cit. 2024-03-01]. Dostupné z: https://suricata.io

[27] MITRE. CVE-2020-11899. *Cve.mitre.org* [online]. 2020 [cit. 2024-03-01]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-11899

[28] MITRE. *CVE-2019-12263* [online]. 2019 [cit. 2024-03-01]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12263

[29] MITRE. *CVE-2020-11910* [online]. 2020 [cit. 2024-03-01]. Dostupné z: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-11910

[30] HUUUCK. *ADBHoney* [online]. 2018 [cit. 2024-03-02]. Dostupné z: https://github.com/huuck/ADBHoney

[31] BROADCOM. VMware Solutions. *Vmware.com* [online]. 2024 [cit. 2024-03-03]. Dostupné z: https://www.vmware.com/solutions.html

[32] THE LINUX FOUNDATION. What Is Linux? *Linux.com* [online]. 2024 [cit. 2024-03-04]. Dostupné z: https://www.linux.com/what-is-linux/

[33] CANONICAL. Ubuntu for desktops. *Ubuntu.com* [online]. 2024 [cit. 2024-03-04]. Dostupné z: https://ubuntu.com/desktop

[34] NAMANKEDI. Introduction to Scripting Languages. *GeeksForGeeks* [online]. c2024 [cit. 2024-03-04]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-scripting-languages/

[35] ELASTIC. Elastic Stack. *Elastic.co* [online]. c2024 [cit. 2024-03-04]. Dostupné z: https://www.elastic.co/elastic-stack

[36] Nginx. *Nginx.org* [online]. c2024 [cit. 2024-03-04]. Dostupné z: https://nginx.org/en/

[37] CLOUDFLARE. What is a reverse proxy? | Proxy servers explained. *Cloudflare.com* [online]. c2024 [cit. 2024-03-04]. Dostupné z: https://www.cloudflare.com/en-gb/learning/cdn/glossary/reverse-proxy/

[38] WIREGUARD. Conceptual Overview. *WireGuard.com* [online]. C2015-2022 [cit. 2024-03-05]. Dostupné z: https://www.wireguard.com/#conceptual-overview

[39] APACHE SOFTWARE FOUNDATION. About the Apache HTTP Server Project. *Apache HTTP Server Project* [online]. c1997-2023 [cit. 2024-03-05]. Dostupné z: https://httpd.apache.org/ABOUT_APACHE.html

[40] MAXMIND. GeoLite2 Free Geolocation Data. *Dev.maxmind.com* [online]. c2024 [cit. 2024-03-05]. Dostupné z: https://dev.maxmind.com/geoip/geolite2-free-geolocation-data

[41] The Postfix Home Page. *Postfix.org* [online]. - [cit. 2024-03-05]. Dostupné z: https://www.postfix.org

[42] GOSECURE. Pyrdp. *GitHub.com* [online]. c2024 [cit. 2024-03-06]. Dostupné z: https://github.com/GoSecure/pyrdp

[43] MOZILLA. SSL Configuration Generator. *Ssl-config.mozila.org* [online]. c2024 [cit. 2024-03-12]. Dostupné z: https://ssl-config.mozilla.org/#server=nginx&version=1.17.7&config=modern&openssl=1.1.1k&guideline=5.7

[44] OFFSEC. Intro To Honeypots. , Tristram. *Offsec.com* [online]. c2024 [cit. 2024-03-17]. Dostupné z: https://www.offsec.com/offsec/intro-to-honeypots/

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

IP	internet protocol
HTTP	hypertext transfer protocol
HTTPS	hypertext transfer protocol secured
SSL	secure sockets layer
TLS	transport layer security
URL	uniform resource locator
DNS	domain name system
VS	virtuálny systém
CA	certification authority
OCSP	online certificate status protocol
PHP	PHP: hypertext preprocessor

ZOZNAM OBRÁZKOV

ZOZNAM PRÍLOH

PRÍLOHA P I: OBSAH CD

Štruktúra priloženého CD:

- Adresár **Configuration files and scripts -** obsahuje konfiguračné súbory a skripty, ktoré boli použité pri tvorbe diplomovej práce, resp. honeypotov:
  - **DP_Janovic_tpot** – obsahuje konfiguračné súbory a skripty pre rovnomenný VS:
    - Filebeat – konfiguračný súbor projektu FileBeat,
    - WireGuard – konfiguračný súbor WireGuard VPN tunela,
    - Others – tpot.yml súbor.
  - **ELK** – obsahuje konfiguračné súbory a skripty pre rovnomenný VS:
    - ELK Stack – konfiguračné súbory pre elasticsearch, kibana a logstash,
    - NGINX – konfiguračný súbor NGINX reverse proxy,
    - WireGuard – konfiguračný súbor WireGuard VPN tunela,
    - Others – cve.yml a iprep.yml súbory.
  - **PyRDP** – obsahuje konfiguračné súbory a skripty pre rovnomenný VS:
    - FileBeat – konfiguračný súbor projektu FileBeat,
    - GeoLite2 – konfiguračný súbor GeoLite2 geolokačných databáz,
    - PostFix – konfiguračný súbor mailového servera PostFix,
    - Scripts – run_geoipupdate, run_strip, strip a watcher skripty,
    - WireGuard – konfiguračný súbor WireGuard VPN tunela.
  - **WEB-POT** – obsahuje konfiguračné súbory a skripty rovnomenný VS:
    - Apache2 – konfiguračné súbory webového servera Apache2,
    - FileBeat – konfiguračný súbor projektu FileBeat,
    - GeoLite2 – konfiguračný súbor GeoLite2 geolokačných databáz,
    - Scripts – run_geoipupdate skript,
    - WebPages – HTML a PHP pre webové stránky WEB-POT honeypotu,
    - WireGuard – konfiguračný súbor WireGuard VPN tunela.
- Adresár **Fulltext -** obsahuje text diplomovej práce vo formáte PDF.