

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

Honeypot rendszer optimalizálása Biztonsági Műveleti Központban

Készítette
Balogh Ádám
Érsok Máté

Konzulens
Vörösné Dr. Bánáti-Baumann Anna

Külső konzulens
Szarvák Anikó

2022. július 3.

Tartalomjegyzék

1. Bevezetés	2
1.1. Igény a biztonságra	2
1.2. Veszélyek	2
1.3. Lehetőségek	2
2. Honeypotok	4
2.1. Általános bemutatás	4
2.2. Működésük	4
2.3. Hátrányai	6
3. Biztonsági Műveleti Központ	8
3.1. Biztonsági Műveleti Központ feladata és működése	8
3.2. SIEM rendszer	8
3.3. Szolgáltatások és riasztások	8
3.4. Biztonsági Műveleti Központ oldali követelmények	9
4. Tervezés	10
4.1. Környezet bemutatása	10
4.2. Követelmények felsorolása	10
4.3. Kiválasztás	11
4.3.1. Honeypotok listázása, legjobb öt kiválasztása	11
4.3.2. Végleges honeypot kiválasztása	14
4.4. <i>HoneyTrap</i> futtatott szolgáltatásainak kiválasztása	16
4.5. <i>T-Pot</i>	17
4.6. <i>Elastic Stack</i>	18
4.7. Honeypot által gyűjtött naplóállományok felhasználása	19
4.8. Szükséges kiegészítés	20
5. Implementáció	21
5.1. Docker	21
5.2. Függőségek	21
5.3. <i>Honeytrap</i>	21
5.3.1. Telepítés	21
5.3.2. Konfigurálás	22
5.3.3. Rejtett portscan detektálásának implementálása	23
5.3.4. Funkcionális tesztek	23
5.4. <i>T-Pot</i>	25
5.5. Naplózás konfigurálása	25
5.5.1. <i>HoneyTrap</i> naplózása	25
5.5.2. <i>T-Pot</i> naplózása	26
5.6. Honeypotok bekötése <i>Elasticsearch</i> -be	27
5.7. Honeypotra épülő szolgáltatások implementációja	29
6. Tesztelés	32
6.1. Módszer kiválasztása	32
6.2. Tesztelés megvalósítása	32
6.3. Eredmények értékelése	35

6.4. Automata szkennerek	36
6.5. Optimalizációs lehetőségek	37
7. Összegzés	39

Ábrák jegyzéke

1. Honeypotok helye az architektúrában	10
2. HoneyTrap beillesztése az architektúrába	17
3. Elastic Stack komponenseinek kapcsolata[36]	18
4. HoneyTrap konténer létrehozása	22
5. SYN letapogatás naplózása a 8022-es porton	23
6. Hiba, majd helyes bejelentkezés kiadott parancsokkal az SSH-Simulator szolgáltatáson	23
7. Bejelentkezési kísérlet az SSH-Authenticator szolgáltatásra	23
8. Bejelentkezés, majd kiadott parancs a Telnet szolgáltatáson	24
9. Bejelentkezés, majd kiadott parancs az FTP szolgáltatáson	24
10. Csatlakozás a TFTP szolgáltatáshoz	24
11. Csatlakozás a HTTPS szolgáltatáshoz	24
12. Kiadott parancsok a TELNET szolgáltatáson	24
13. T-Pot telepítése során a portok áthelyezése	25
14. Könyvtár felcsatolása konténerbe	26
15. Egy honeypot konfigurálható naplózási lehetőségei	26
16. Logstash konfiguráció minta[42]	28
17. Egy index bejegyzései a Kibana felületén	29
18. Az irányítópultokat alkotó panel készítése	30
19. Összesített irányítópult a Kibana felületén	30
20. A hallgatók előrehaladása a CTFd felületén szemléltetve	33
21. Összesítő irányítópult	34
22. Egyetlen honeypot, a Cowrie irányítópultja	34
23. A 15 leggyakrabban megcímzett port	35
24. Zászló megtalálásáig eltelt idő (percben)	36
25. Kérések időbeli eloszlása honeypot szolgáltatások és hívások között	36
26. Világtérkép a honeypotra érkező támadásokról	37

Táblázatok jegyzéke

1. Honeypotok osztályozása interáció szerint	5
2. Honeypotok értékelése a nyújtott szolgáltatások szerint	14
3. Honeypotok értékelése dokumentáció részletessége szerint	15
4. Honeypotok értékelése frissítések rendszeressége szerint	15
5. Honeypotok pontszámainak összegzése	16

Absztrakt

A negyedik ipari forradalom korszakában, a hétköznapijaink digitalizációjának rohamos terjedése közben, az előnyök kihasználása mellett sajnos a kockázatokra is fel kell készülnünk. Már nem az a kérdés, mikor törnek be a hálózatunkba, hanem az, hogy mikor vesszük észre – s mégsem teszünk ellene eleget. A kiberbűnözők olyan kifinomult és fejlett technikák alkalmazásával érik el céljaikat, amelyekkel szemben a hagyományos védelmi megoldások (tűzfalak, IDS/IPS, stb.) hatástalannak bizonyulnak, ezért a védelmi oldalon is egyrészt újszerű és automatizált megoldásokra van szükség, másrészt mérhetetlen mennyiségű adatra a kutatások, elemzések és a hatékonyabb védelmi mechanizmusok kifejlesztése érdekében. Célunk egy honeypot rendszer fejlesztése az egyetemi SOC (Security Operation Center) labor bővítéseként, mely mindkét elvárásnak egyszerre képes megfelelni. A honeypot-ok önmagukat sebezhetőnek mutatva keltik fel a behatolók figyelmét, majd amint bárki kapcsolatot létesít velük, egy riasztási lánc elindításával értesítik a hálózat üzemeltetőit a behatolás tényéről. A támadás passzív észlelése mellett a honeypotok feltartják a támadót, aki így nem a valós rendszerekkel fog foglalkozni, illetve az utólagos elemzésekhez is elengedhetetlen információkkal szolgál a támadó módszereiről. A projektünkben egy honeypot, illetve az Elastic Stack-en alapuló log és riasztás kezelő rendszer megtervezésével, implementálásával, konfigurálásával, illetve tesztelésével foglalkoztunk. A tesztelés során létrejövő adathalmaz vizsgálatával igyekeztünk feltárni a honeypotok optimalizálási lehetőségeit, valamint hatékonyságuk mérhetőségét. Kiválasztottuk, mely eseményeket érdemes folyamatosan megfigyelni, illetve riasztásokat kialakítani rájuk. Hogy megfelelő mennyiségű, valós támadásokat szimuláló információhoz jussunk, az egyetemen egy „zászlófoglaló” kihívást hirdettünk meg, és a szimulált honeypot szolgáltatások közé valós sérülékeny rendszert helyeztünk el. A beérkező információk felhasználásával azt igyekeztünk kideríteni, milyen módszerekkel próbáltak behatolni a támadók, illetve mennyire sikerült hátráltatni a lényeges információ kinyerését. Az ellenőrzést erre kialakított irányítópultokon keresztül tettük meg, amelyek a gyűjtött adatokat, gyorsan értelmezhető, diagramokon megjeleníthető információvá formálják.

1. Bevezetés

1.1. Igény a biztonságra

A 21. században, eddig soha nem látható tempóban láthatjuk kibontakozni a negyedik ipari forradalmat. A kiterjedt információs hálózatok már nem csak a nagyobb vállalatok privilégiumai, a folyamatosan csökkenő hardver árak és az egyszerűbben használható alkalmazások miatt ma már gyakori, hogy a kisebb pár fős vállalatok is nagyobb belső hálózatot üzemeltetnek. Ez a tendencia nem állt meg a vállalatoknál, az okos eszközök forradalmának köszönhetően IoT néven ma már rengeteg otthonba begyűrűzött. Az átlagos felhasználóknak fel sem tűnik, de nagyon könnyen, jelentős méretű hálózatot hozhatnak létre okos - hálózati csatlakozásra képes – eszközökből. Sajnos a nagyobb hálózatokkal és a figyelmetlen tervezéssel a biztonsági kockázat is nő, akaratlanul olyan kikapukat hagyhatunk, amelyeken bejutva a modern kor bűnözői könnyen anyagi vagy személyes károkat tudnak okozni. Dolgozatunkban egy lehetséges megoldást mutatunk be az illetéktelen behatolók felderítésére, tevékenységének nyomon követésére, lassítására, illetve az esetlegesen okozott károk csökkentésére.

1.2. Veszélyek

Gyakori, hogy cégek elhanyagolják, vagy nem kezelik kellő súllyal az információs rendszereken keresztül elkövetett bűncselekményeket, nem teszik meg a kellő óvintézkedéseket, ezáltal teret engedve a visszaéléseknek. Ez egyrészt főleg a kisebb vállalatoknál érthető, hiszen egy biztonságos informatikai rendszer kiépítése és felügyelete kimondottan költséges - nem hoz közvetlen, csak közvetett hasznot a cégnek - ugyanakkor felelőtlen, hiszen egy esetleges incidens sokkal nagyobb károkat okozhat. A Cisco adatai alapján 2018-ban egy közepes méretű vállalattal szembeni sikeres támadás a megkérdezett cégek 53%-ánál több, mint 500.000 dollár (akkori árfolyamon 140.000.000ft) kárt okozott[1]. Rengeteg támadási formáról hallhatunk, találhatunk információt, melyek legtöbb esetben károsíthatnak egy vállalatot vagy államigazgatási szervet, hátráltathatják a hatékony működésben. A támadási lehetőségek száma szinte végtelen, de vannak köztük látványosabban népszerűbbek. A DOS vagy DDOS – (distributed) denial of service, amely (elosztott) szolgáltatás megtagadást jelent. Ezen támadási forma a vállalat valamely szolgáltatását igyekszik oly mértékben leterhelni, hogy az szolgáltatás kieséshez vezessen. A malware-ek rengeteg formája okozhat további problémákat egy informatikai rendszeren. Ebbe csoportba tartozik a már régóta jelenlévő vírusok és férgek, de a napjainkban meglehetősen népszerű ransomware (zsarolóvírus) és spyware (kémprogram) megoldások is. Ha ez még nem lenne elég SQL injection (SQL injektálás) vagy épp a Cross-Site Scripting (weboldalak közötti szkriptelés) is kellőképpen nagy fenyegetést jelent a vállalatok számára. Az kiberbiztonsági szakemberek számára tehát adott a feladat, meg kell teremteni az ilyen és hasonló támadások elleni biztonsági eszközöket.

1.3. Lehetőségek

A fenyegetések elhárítására több egyszerű, illetve komplexebb megoldást is találhatunk. A lehetséges fenyegetések eredetét két részre lehet bontani: emberi eredetű, illetve tervezési/konfigurációs hibára. Emberi eredetű hibák lehetnek például a

felelőtlenül használt adathordozók, ismeretlen emailekben található csatolmányok/hivatkozások megnyitása, nem megbízható oldalak felkeresése. Ezeket a kockázatokat rendszeres biztonsági oktatásokkal, tűzfal szabályokkal, adathordozók korlátozásával, illetve vírusirtó szoftver használatával csökkenthetők. Tervezés és konfiguráció során számtalan hibát ejthetünk, mely az egész hálózat biztonságát veszélybe sodorhatja. A teljesség igénye nélkül ilyen lehet a nem megfelelő jelszóhasználat (root - toor), elavult kommunikációs kriptográfiai protokoll, érzékeny adatok nem vagy nem megfelelő titkosítása, belső hálózat nem megfelelő tagolása stb. Ezeket legegyszerűbben a tervezési fázisban, a „secure by design” elvét követve lehet kiküszöbölni. Mivel sosem leszünk képesek teljesen biztonságos hálózatot létrehozni, fontos, hogy a rendszerünket folyamatosan ellenőrzés alatt tartsuk és képesek legyünk észlelni, az illetéktelen behatolást. Figyelembe véve a tényt, hogy 2016-ban világszerte átlagosan 146 nap[2] telt el mire fény derül egy adatszivárgásra, a folyamatos passzív hálózat figyeléssel legalább olyan prioritással érdemes foglalkozni, mint a megelőzéssel. Ezek a rendszerek a hálózaton megfigyelhető anomáliákat kiértékelve jelzik az üzemeltetők felé az esetleges behatolást, ezáltal jobb esetben már a kezdetektől felhívhatják a figyelmünket egy esetleges betörésre. Ebbe a kategóriába tartoznak a honeypot rendszerek, a továbbiakban ezekkel fogunk foglalkozni.

2. Honeypotok

2.1. Általános bemutatás

A honeypotok működésük szempontjából az illetéktelen hálózati behatolást jelző rendszerek csoportjába tartoznak, lehetnek aktívak és passzívak is. A honeypotok magukat a hálózat szerves részének álcázva, egy létező szolgáltatást futtatnak vagy szimulálnak, amely valójában a rendszer többi komponensétől külön álló, izolált környezetben található. [3] Amint a támadó kapcsolatba lép a szerverrel, belesétál a „csapdába”, mivel a honeypot minden tevékenységét rögzíti, riasztást küld az üzemeltetőknek, vagy esetleges ellenintézkedéseket foganatosít. Megfelelő mennyiségű és minőségű információ esetén, egy későbbi jogi eljárásban ezeket az adatokat akár bizonyítékként is fel lehet használni. Ezeknél a rendszereknél lényeges, hogy a támadó felfigyeljen rájuk és minél nagyobb valószínűséggel létesítsen velük kapcsolatot. Ezt többféleképpen tudják elérni, az egyik lehetőség, hogy elavult sérülékeny technológiákat alkalmaznak. Ilyen lehet például egy Telnet protokoll használata, TLS helyett elavult SSL titkosítás használata, triviális jelszóhasználat, vagy bármilyen szolgáltatás sérülékeny verziójának a használata. A másik lehetőség a mennyiség növelése, több honeypot hálózatát honey network-nek nevezzük. Ilyet akkor érdemes kiépíteni, ha nagyobb, szerteágazóbb hálózatot szeretnénk felügyelni, ahol egy honeypot elveszne a tömegben.

2.2. Működésük

A honeypotokat kétféleképpen tudjuk csoportosítani, egyrészt az interakció szintjük, másrészt a környezetük szerint. [4] Az interakció szintje szerint háromféleképpen lehet osztályozni a honeypotokat. Ezek a kis, közepes és magas interakciójú honeypotok. Ez azt mutatja, hogy az általuk futtatott szimuláció mennyire komplex, mennyire tudja elhitetni a támadóval a rendszer valódiságát. [4] A különböző tulajdonságaik táblázatos formában:

	Alacsony interakció	Közepes interakció	Magas interakció
Jellemzésük	Egyszerű, szimulált információkat mutat, lényegi funkcionalitása nincs	Komplexebb szimulációt futtat, mint az alacsony inetrakciójú honeypot, de komolyabb beavatkozásoknak nem hagy teret	Nagyon komplex szimulációt vagy valós szolgáltatást futtat, tényleges funkcionalitással
Implementálásuk	Alacsony komplexitásuk miatt egyszerű	Mivel csak szimulált, nem feltétlenül valószerű szolgáltatásokat futtat, a telepítése bonyolultabb, mint az alacsony interakciójú honeypotoké, de egyszerűbb, mint a magasaké	Az ezek által futtatott szolgáltatások nagyon valóságghű szimulációk, vagy valós rendszerek, implementációjuk és konfigurációjuk hosszas folyamat
Gyűjthető információ mennyisége	Komoly funkcionalitás hiányában ezek a honeypotok csak minimális információt tudnak szerezni a támadóról, ilyen például az ip-címe	A közepes interakciójú honeypotok, a komplexebb szimulációnak köszönhetően több adatot tudnak szerezni a támadóról és a betörés módjáról, ilyen lehet egy jelszó feltörésének a módszere	Mivel itt a nyújtott szolgáltatások összetettek, sok lehetőséget nyújtanak a támadónak, ezáltal komplex képet alkothatunk a rendszeren belüli viselkedéséről
Példa	Alacsony interakciójú honeypotra példa egy szolgáltatás, ami a 25-ös nyitott porton smtp szolgáltatást jelez	Ilyen például egy ssh szimuláció, ahova ssh-n keresztül be lehet lépni, bekéri a jelszót és felhasználónevet, de beléptetés után minden parancsra „not found” hibát jelez	Magas interakciójú honeypot lehet például egy adatgyűjtéshez módosított Windows 2000

1. táblázat: Honeypotok osztályozása interakció szerint

A környezet szerint két féle honeypotot tudunk megkülönböztetni, ezek a production (termelési környezetben elhelyezett), illetve a kutatási célú honeypotok.

Céljuk:

- A production honeypotok célja, hogy megvédjék a vállalat belső rendszereit az illetéktelen behatolóktól. Ehhez a mézesbödönt a többi szolgáltatással egyező hálózatba kell elhelyezni, ezzel növelve a behatolás leleplezésének az esélyét, illetve a támadó figyelmét elvonva a többi szolgáltatástól.
- Ezzel ellentétben a kutatási célú honeypotok célja, az információszerzés a különböző támadási módszerekről, motivációkról, hogy a későbbiekben a vállalat ezeket az információkat figyelembe véve tudja fejleszteni a védelmi képességeiket. [4] Annak érdekében, hogy több információt lehessen gyűjteni, ezeket honeypotokat általában a hálózaton kívülre, nyilvánosan elérhető helyre kell helyezni ezzel növelve a támadás esélyét.

Szolgáltatás:

- A production honeypotok általában csak szimulátorok, valós szolgáltatásokat nem futtatnak. A honeypot interakciója, a szimuláció komplexitásától függően változik.
- A kutatási célú honeypotok általában valós szolgáltatásokat nyújtanak, ezáltal rengeteg információt tudnak gyűjteni a támadókról. [4] Nem ritka, hogy az adatok melyeket ellopnak nyomon követhetőek, ezáltal még több információ kideríthető a támadás módjáról[5].

Gyűjthető információ mennyisége:

- Mivel a production honeypotok célja elsősorban nem az információgyűjtés, hanem a behatolás érzékelése egy színlelt szolgáltatással, ezért közepes mennyiségű adatot tudunk belőle gyűjteni. [4] Ez jelentheti a forrás IP címet, szimulált bejelentkezésnél a használt jelszót és felhasználónevet, illetve a szimulátoron belül kiadott parancsokat.
- A kutatási célú honeypotoknál maximális adat összegyűjtése a cél, amit egy valós szolgáltatáson keresztül lehet legkönnyebben elérni. A gyűjthető információ mennyisége, jelentős. [4]

2.3. Hátrányai

Számos előnyük mellett, a honeypotoknak vannak hátulütői. Általában minél komplexebb egy rendszer, annál könnyebb kiskaput találni rajta, ez a honeypotokra is jellemző.

Az interakció szintjének emelésével a támadók érdekesebbnek találhatják a honeypotot, ezáltal több idejüket pazarolják rá, de ez azt is jelenti, hogy nagyobb eséllyel tudják a saját céljukra használni azt. Emiatt a kockázat miatt érdemes megfontolni a magas interakciójú honeypotokat, hiszen nem előnyös a támadó kezébe adni egy támadási platformot. Ez a veszély csak a production honeypotoknál áll fent, a research

honeypotok esetében, mivel az a hálózaton kívül helyezkedik el, legfeljebb magában a rendszerben tud kárt tenni a támadó. [6]

Hátrány lehet a fals pozitív riasztás, ami abból eredhet, hogy a cég dolgozói hiányos, vagy semmilyen tájékoztatást nem kapnak a kiépült rendszerről, így figyelmetlenül kapcsolatot kezdeményeznek vele. Ennek az esélye helyes cégen belüli kommunikációval csökkenthető. A honeypotok további gyenge pontja, hogy nem feltétlenül talál rá a támadó. Nagyobb hálózatban, sok szolgáltatással körbe véve, a honeypot által nyújtott „kihagyhatatlan lehetőségek” elveszhetnek a többi információ tengerében. Ez a probléma kiküszöbölhető honeynet létrehozásával, ami két vagy több honeypot összességét jelenti. Erre a cégnek több erőforrást kell áldoznia, nehezíti az implementálást, konfigurálást, illetve a karbantartási folyamatokat.

Ha egy támadó tudomására jut, hogy a hálózatban honeypot rendszer működik, szándékosan félre tudja vezetni a biztonsági szakembereket. Ezt úgy tudja megtenni, hogy álcázott támadásokat indít a honeypot ellen, amíg mindenki a honeypot riasztására reagálva a nem létező támadót keresi, addig a valódi behatoló könnyedén hozzáférhet a valós rendszerekhez.

3. Biztonsági Műveleti Központ

3.1. Biztonsági Műveleti Központ feladata és működése

Egy Security Operation Center (továbbiakban SOC, Biztonsági Műveleti Központ) leginkább a NASA Küldetés Irányító Központjához hasonlítható, egy fal tele monitorokkal, amelyet egy egész teremnyi specialista figyel keresve a legapróbb jelet bármiféle támadásra. A valóság ennél egy kicsit evilágibb. Legegyszerűbben megfogalmazva a SOC egy gyakorlott szakértőkből álló csapat, akiknek rendelkezésére áll egy megfelelő eszközkészlet. Feladatuk szervezet teljes hálózati infrastruktúrájának monitorozása támadások és fenyegetések jeleit keresve, ezzel megteremve annak a lehetőséget, hogy szinte azonnal képesek legyenek reagálni a bekövetkezett eseményekre. Ahelyett, hogy passzívan riasztásokat várnának, vadászhatnak is a támadásokra, mellyel még inkább növelhetik a rendszer biztonságát.

3.2. SIEM rendszer

A SOC legfontosabb része a SIEM (Security Information and Event Management, Biztonsági Információ- és Eseménykezelés), amely egy központosított formában gyűjti és dolgozza fel az információs rendszer valamennyi eszközéről és szolgáltatásából érkező naplóbejegyzéseket. Az összegyűjtött adatokból értelmezhető információt hoz létre azáltal, hogy a hasonlókat, közös vonásokkal rendelkezőket egy kötegbe fogja össze. Ezen információt összeveti a rendelkezésére álló fenyegetési mintázatokkal és amennyiben talál egyezést, előre definiált szabályok alapján riasztást küld. A riasztás különböző csatornákon érkezhets az elemzőkhöz, akár egyszerű szöveges üzenet formájában, akár egy biztonsági irányítópult felületére. Kizárólag a riasztás célja fontos, ami pedig az, hogy felkeltse a szakemberek figyelmét a megszokottól eltérő működésre. [7]

3.3. Szolgáltatások és riasztások

Egy ilyen jelentős adatmennyiséget kezelő, megközelítőleg az infrastruktúrában minden eszköztől naplóadatot gyűjtő rendszerben a kialakítható további szolgáltatások listája végtelenül hosszú. Az eszközök, alkalmazások fontossága és a bekövetkezett események súlyossága alapján a szolgáltatás- és riasztásrendszert különféle kritikussági szinteken definiálhatjuk. Ezeket a biztonsági kategóriákat kulcsfontosságú precízen megszabni, annak érdekében, hogy kellően releváns tájékoztatást nyújtson az elemzők számára. Ez megtehető az értesítés gyakoriságának, a tájékoztató csatorna, valamint az informálni kívánt szakértői csoport jól megfontolt megválasztásával. Amennyiben egy jelentéktlenebb esetről van szó, meghatározhatunk például egy olyan szabályt, mely alapján az értesítés csak megszabott gyakorisággal (óránként egyszer, naponta egyszer) hívja fel a szakértők figyelmét. Ezzel szemben, hogyha egy rendkívül kritikus ügyet észlel, előnyös ezt azonnal a rendszert megfigyelői között, minél nagyobb körben elterjesztenie azért, hogy a kivizsgálás a lehető leghamarabb megkezdődhessen. Ezen riasztásoknál többféle kommunikációs csatorna is szóba jöhet. Történhet emailben, a szakemberek vállalati mobileszközeire egy pillanat alatt megérkező leküldéses értesítés (push notification) formájában, melyek vagy a szervezeti üzenetküldő platform elkülönített csatornáján, vagy akár egy erre célra létrehozott szolgáltatás segítségével.

Ezek mellett a SIEM rendszer irányítópultokba (dashboard) rendezve is felvilágosítja az elemzőket a történt eseményekről. Ebben a formában lehetőség nyílik többféle kimutatás konfigurálására, melyekre rápillantva azonnal észrevehető a szokásostól eltérő működés. [7][8]

3.4. Biztonsági Műveleti Központ oldali követelmények

A honeypot által gyűjtött információk felhasználása egy logkezelő rendszer segítségével kell történjen, amely centralizáltan gyűjti, tárolja, valamint rendszerezi a beérkezett naplódokumentumokat. Az egyetemi környezetben ezek az Elastic Stack alkalmazáskészlete által kerülnek feldolgozásra, ezáltal a honeypotnak valamilyen formában összekapcsolhatónak kell lennie a már meglévő rendszerrel.

A honeypot szolgáltatásáról/szolgáltatásairól az infrastruktúrára fenyegetést jelentő eseteket naplózni kell, és ezen naplózás szerint kritikussági szintekre bontva, definiálni riasztásokat, amelyek gyorsítják az elemzők munkáját. Ezen riasztások történhetnek emailen, de akár egy erre alkalmas csevegő platformon keresztül is. Az riasztási üzenetnek tartalmaznia kell, hogy a honeypot miatt generálódott, valamint, néhány fontosabb információt a kiváltó okról. Ez lehet például a megtámadott szolgáltatás neve, a támadó forrás címe, vagy éppen a támadó által bevitt adatok, parancsok.

A csapdaként működő alkalmazásoknak és szolgáltatásoknak illeszkednie kell a már meglévő környezetbe. Szükséges elkülöníteni a honeypotról érkező naplóbejegyzéseket egy külön csatorna vagy index létrehozásával, hogy ezen logok izolálva legyenek a további alkalmazások gyűjteményeitől.

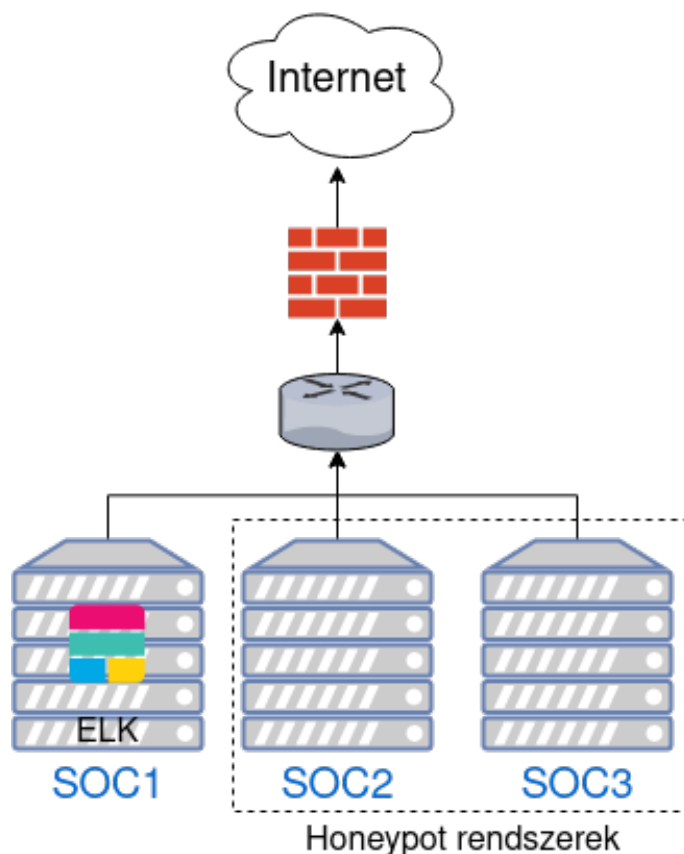
Fontos kidolgozni egy megoldást, mely segítségével a honeypot működése meghatározott gyakorisággal automatikusan tesztelhető anélkül, hogy fals pozitív riasztás keletkezne. Ezt érdemes egy honeypottal azonos környezetben üzemelő szerveren megvalósítani, ezáltal minimalizálni a teszt közben felmerülő hibalehetőségeket, amely forrása lehet egy téves tűzfalkonfiguráció, vagy akár egy hálózati kiesés.

Amennyiben lehetséges, célszerű a keletkezett adatokat vizualizálni, így gyorsítva az elemzők munkáját. A szemléltetés főbb pontjai, a felhasznált bejelentkezési adatok, bevitt parancsok, valamint a behatolások gyakorisága szolgáltatásokra lebontva, helyadatok felhasználásával a támadók IP címének térképen történő megjelenítése.

4. Tervezés

4.1. Környezet bemutatása

A honeypotot egy, az iskola SOC-nak fenntartott hálózatában szeretnénk implementálni, (lásd. 1 ábra). A SOC hálózat az internet felé tűzfalon keresztül kommunikál, a hálózatra ezen keresztül bejutható behatolók kiszűrése érdekében szükséges biztonság fokozása érdekében egy honeypotot fogunk elhelyezni ezen a hálózaton.



1. ábra. Honeypotok helye az architektúrában

4.2. Követelmények felsorolása

Mivel ma már nagyon sok és sokféle honeypot rendszer elérhető, szükséges egy követelményrendszer felállítása, hogy ezek közül ki tudjuk választani a számunkra legmegfelelőbbet. Az implementáláshoz az alábbi feltételeknek legjobban megfelelő honeypotot választjuk ki:

- **Nyílt forráskódú:** Mivel a mi célunknak egy egyszerűbb, könnyebben implementálható honeypot is megfelel, fölösleges külön pénzforrást szoftver és támogatás beszerzésére elkülöníteni, számos nyilvánosan elérhető megoldás található az interneten.
- **Dokumentáció:** Egy részletes, implementációra és konfigurációra egyaránt kitérő dokumentáció megléte fontos követelmény. Ez fontos lehet a rendszer

létrehozásánál, illetve a későbbi karbantartás sem követel majd hosszú utánajárást. Az ezt nem teljesítő, „kód dokumentálja önmagát” hozzáállással készült honeypotok nem alkalmasak

- Alacsony, közepes interakció: A honeypotnak képesnek kell lennie felkelteni a behatoló figyelmét és lekötni a figyelmét rövidebb időre miközben információt gyűjt róla. Fontos továbbá, hogy ne jelentsen biztonsági kockázatot a rendszer többi komponensére (lásd: 2.3). Ezt a két szempontot figyelembe véve, a legoptimálisabb egy alacsony, de inkább közepes interakciójú honeypot.
- Hálózat többi szolgáltatásától nem jelentősen eltérő: Fontos a támadó figyelmét felkelteni, de ha a honeypot a hálózaton futó rendszerektől nagyon eltér, az inkább gyanút kelthet a támadóban. Ilyen, hálózatba nem illeszkedő honeypotra példa egy android eszköz szimuláció. Elsősorban FTP, TFTP, HTTPS, SSH és Telnet szolgáltatásokra van szükségünk.
- További előny, ha a honeypot rendelkezik docker képállal, ezáltal egyszerűen el lehet helyezni azt egy konténerizált környezetben. Előnyeit a 5.1. fejezetben részletezzük.

4.3. Kiválasztás

A honeypot kiválasztását két lépésben hajtjuk végre. Az első fázisban különböző népszerű honeypotokat gyűjtünk össze, majd ezekből, a fent említett szempontok alapján az öt legjobbat választjuk ki. Ezeket dokumentáció, frissítés gyakorisága és a nyújtott szolgáltatások alkalmassága szerint pontozzuk. A legtöbb pontszámot elérő honeypot kerül később implementálásra.

4.3.1. Honeypotok listázása, legjobb öt kiválasztása

A honeypotokat a githubon található honeypot gyűjteményből állítottuk össze[9], itt nyújtott szolgáltatásonként kategorizálva szinte az összes ilyen szoftver megtalálható. Az alábbiakban a különböző kategóriák szerint válogattott, a github-on népszerű 20 honeypotot vizsgáljuk funkcióik, illetve a számunkra való megfelelésük szerint:

- **Ciscoasa**

Alacsony interakciójú honeypot, amely a Cisco által gyártott ASA (Adaptive Security Appliances) eszközök CVE-2018-0101 sérülékenységét tudja szimulálni, illetve észlelni, ha valaki kihasználja.

A hálózat többi elemétől nem tér el jelentősen, nem található hozzá dokumentáció, illetve csak alacsony interakcióra képes, nem megfelelő [10]

- **CitrixHoneypot**

Citrix Application Delivery Controller CVE-2019-19781-es sérülékenységét szimulálja, képes az ennek a kihasználására tett kísérleteket, illetve a bármilyen kapcsolat létrejöttét észlelni.

A hálózaton futó többi alkalmazástól jelentősen eltér, nem található hozzá dokumentáció, illetve csak alacsony interakcióra képes, nem megfelelő [11]

- **Conpot**

Ipari irányító rendszert szimuláló alacsony interakciójú honeypot.

A hálózaton futó többi alkalmazástól jelentősen eltér, nem található hozzá dokumentáció, illetve csak alacsony interakcióra képes, nem megfelelő [12]

- **Cowrie**

Nagyon népszerű, közepes és magas interakcióra képes SSH-t és Telnetet szimuláló honeypot. Állítható interakció szint, magas interakcióval SSH-proxyként megfigyelhető a támadó által folytatott kommunikáció, közepes interakcióval UNIX rendszer shell parancsait szimulálja.

A hálózaton futtatott alkalmazásokhoz illeszkedik, jól dokumentált, közepes és magas interakcióra is képes, megfelelő [13]

- **dicompot**

Egy teljesen funkcionális DICOM (Digital Imaging and Communications in Medicine) szerveret futtat.

A hálózaton futó többi alkalmazástól jelentősen eltér, kevés dokumentáció található, illetve csak magas interakcióra képes, nem megfelelő [14]

- **dionaea**

17 főleg alacsony interakciójú szolgáltatás futtatására képes honeypot keretrendszer, többek között SSH, FTP, TFTP, http.

A hálózaton futtatott alkalmazásokhoz illeszkedik, megfelelően dokumentált, alacsony és közepes interakcióra is képes, megfelelő [15]

- **elasticshoney**

Elasticsearch szolgáltatást szimuláló honeypot, ami képes különféle RCE sérülékenységek szimulálására, közepes – magas interakcióval.

A hálózaton futtatott alkalmazásokhoz illeszkedik, közepesen dokumentált, közepes és magas interakcióra képes, megfelelő [16]

- **ElasticPot**

Elasticsearch szolgáltatást szimuláló közepes interakciójú honeypot. GET PUT, POST, DELETE kérelmek kezelésére képes.

A hálózaton futtatott alkalmazásokhoz illeszkedik, nem dokumentált, közepes interakcióra képes, nem megfelelő [17]

- **glutton**

SSH és TCP proxyként működő magas interakciójú honeypot. A támadó proxyn keresztül végzett tevékenységét folyamatosan monitorozza.

A hálózaton futtatott alkalmazásokhoz illeszkedik, nem dokumentált, magas interakcióra képes, nem megfelelő [18]

- **heralding**

Bejelentkezésnél használt hitelesítő adatok rögzítésére kiélezett, alacsony-közepes interakciójú honeypot, 14 támogatott protokollal.

A hálózaton futtatott alkalmazásokhoz illeszkedik, nem dokumentált, alacsony-közepes interakcióra képes, nem megfelelő [19]

- **Honeyd**

A Honeyd egy daemon, aminek a segítségével egyszerűen tudunk a hálózatunkon

honeypotokat létrehozni és összekötni, így honeynetet kialakítva.

A kívánt célunk egy honeypot elhelyezése, nem honeynet kiépítése, ezért nem megfelelő [20]

- **HoneyPy**

Alacsony és közepes interakciójú szolgáltatásokat szimuláló honeypot keretrendszer, pluginok formájában adhatók meg a futtatni kívánt szolgáltatások.

A hálózaton futtatott alkalmazásokhoz illeszkedik, kielégítően dokumentált, közepes és magas interakciójú, megfelelő [21]

- **HoneySAP**

Alacsony interakciójú, főleg kutatási célú SAP szolgáltatást szimuláló honeypot.

A hálózaton futtatott alkalmazásokhoz nem illeszkedik, megfelelően dokumentált, alacsony interakciójú, nem megfelelő [22]

- **honeytrap**

Összesen 25 szolgáltatást futtatni képes honeypot keretrendszer. Interakciója szolgáltatástól függően változik, alacsony, közepes és magas is lehet. A szolgáltatások tartalmazzak elavult és még ma is használt protokollokat.

A hálózaton futtatott alkalmazásokhoz illeszkedik, részletesen dokumentált, szolgáltatásoktól függően alacsony, közepes és magas interakciójú is lehet, megfelelő [23]

- **ipphoney**

Egy IPP-t (Internet Printing Protocol) használó nyomtatót szimulál, közepes interakciós szinttel. A hálózaton futtatott alkalmazásokhoz nem illeszkedik, közepesen dokumentált, közepes interakciójú, nem megfelelő [24]

- **mailoney**

SMTP szolgáltatást szimuláló alacsony-közepes interakciójú honeypot. Képes belépéshez használt felhasználóneveket, jelszavakat, illetve az ide továbbított emaileket rögzíteni.

A hálózaton futtatott alkalmazásokhoz illeszkedik, nem megfelelően dokumentált, közepes-alacsony interakciójú, nem megfelelő [25]

- **RDPy**

Microsoft RDP-t (Remote Desktop Protocol) szimuláló közepes-magas interakciójú honeypot.

A hálózaton futtatott alkalmazásokhoz nem illeszkedik, nem megfelelően dokumentált, közepes-magas interakciójú, nem megfelelő [26]

- **SNARE**

A SNARE egy Nginx web alkalmazás szerver honeypot magas interakcióval, főleg kutatási célokra, hálózaton kívülre elhelyezve.

A hálózaton futtatott alkalmazásokhoz nem illeszkedik, megfelelően dokumentált, magas interakciójú, nem megfelelő [27]

- **dhp**

Dockert szimuláló alacsony interakciójú honeypot, GET version, GET ping és POST create image kérélmekre reagál, minden másra 500-as hibakódot ad vissza.

A hálózaton futtatott alkalmazásokhoz illeszkedik, közepesen dokumentált, alacsony interakciójú, nem megfelelő [28]

- **opencanary**

A Honeyd-hez hasonlóan ez is egy honeynetet hoz létre, illetve köt össze, SNMP (Simple Network Management Protocol) és RDP (Remote Desktop Protocol) szolgáltatásokat szimulálva.

Nem illeszkedik a hálózatba, illetve a kívánt célunk egy honeypot elhelyezése, nem honeynet kiépítése, ezért nem megfelelő [29]

4.3.2. Végleges honeypot kiválasztása

Az előző alfejezetben kiszűrtük azokat a honeypotokat amik nem felelnek meg az elvárásainknak, a megmaradt öt alkalmazást táblázatos formában részletesen mutatjuk be, illetve pontozzuk nyújtott szolgáltatások, dokumentáció, illetve frissítés rendszeressége, valamint a 4.2 fejezetben támasztott követelmények szerint. Mivel a fenti szempontok közül a nyújtott szolgáltatások a legfontosabbak, ezért az ott elérhető pontszámok duplán kerülnek súlyozásra, így ez adja majd a teljes értékelésünk 50%-át, míg a dokumentáció és rendszeres frissítés a pontok 25-25 százalékát teszi ki.

	Szolgáltatások	Pontszám
Cowrie	SSH, Telnet, közepes interakciós módban shell-t szimulálva.	14/20
honeytrap	Android Debug Bridge, Counterstrike, CWMP, DNS Proxy, DNS, Echo, Elasticsearch, EOS, Ethereum, FTP, HTTP Proxy, HTTP, HTTPS, IPP, LDAP, Memcached, Redis, SMTP, SSH-Authentication, SSH-Jail, SSH-Proxy, SSH-Simulator, Telnet, TFTP, VNC. Ennél a honeypotnál még a magas interakciójú szolgáltatásokat kiszűrve is rengeteg lehetőség áll rendelkezésünkre.	20/20
elasticshoney	HTTP-n keresztül elasticsearch szolgáltatást szimulál.	12/20
dionaea	Black hole, EPAMP, FTP, http, Memcached, Mirror, MongoDB, MQTT, MSSQL, NFQ, PPTP, Printer, VoIP, SMB, TFTP, UPnP. Habár sok szolgáltatást lehet rajta konfigurálni, ezek többsége nem releváns számunkra.	15/20
HoneyPy	DNS, Echo, Elasticsearch, HashCountRandom, Motd, NTP, Random, SÍP, SMTP, TFTP, Telnet, http.	16/20

2. táblázat: Honeypotok értékelése a nyújtott szolgáltatások szerint

	Dokumentáció	Pontszám
Cowrie	Részletesen dokumentáció, a honeypot minden területére kitér, verziókövetést is beleértve. [30]	10/10
honeytrap	Az implementálás és konfigurálás lépéseit részletesen leírja, a szolgáltatások működése használata és tesztelése kellően részletes. Habár néhány rész még kiegészítésre szorul, ezek nem befolyásolják a dokumentáció hasznosságát. [31]	9/10
elasticchoney	Az implementálásra, a konfigurálásra és a logolásra is kitér, de összességében nem túl részletes.	6/10
dionaea	Részletesen dokumentáció, a honeypot minden területére kitér, verziókövetést is beleértve. [32]	10/10
HoneyPy	Kellően részletes dokumentáció, az elérhető szolgáltatásokhoz tartozó információ nincs eléggé kifejtve. [33]	8/10

3. táblázat: Honeypotok értékelése dokumentáció részletessége szerint

	Frissítés	Pontszám
Cowrie	Rendszeres fejlesztés és javítás alatt, 2018-tól több mint 600 commit érkezett.	10/10
honeytrap	A rendszer teljesen működőképes, de továbbra is sokan járulnak hozzá hibajavítások, és új funkcionalitások hozzáadásával. 2018-tól több mint 200 commit érkezett.	8/10
elasticchoney	Már nem fejlesztik tovább, 2015 óta senki nem módosította.	2/10
dionaea	A rendszer teljesen működőképes, de továbbra is sokan járulnak hozzá hibajavítások, és új funkcionalitások hozzáadásával. 2018-tól több mint 200 commit érkezett.	8/10
HoneyPy	Már nincs aktív fejlesztés alatt, 2018-tól 84 commit érkezett, ennek nagyrésze hibajavítás.	6/10

4. táblázat: Honeypotok értékelése frissítések rendszeressége szerint

	Pontszám összesen
Cowrie	34/40
honeytrap	37/40
elasticsh	20/40
dionaea	33/40
HoneyPy	30/40

5. táblázat: Honeypotok pontszámainak összegzése

A fenti táblázatok alapján megállapítható, hogy az elasticsh 50%-ban, a HoneyPy 75%-ban a dionaea 82,5%-ban, a Cowrie 85%-ban, a honeytrap pedig 92,5%-ban felelt meg az általunk támasztott követelményeknek, így a továbbiakban a honeytrappal fogunk foglalkozni.

4.4. *HoneyTrap* futtatott szolgáltatásainak kiválasztása

A honeytrap tulajdonképpen egy keretrendszer, ami azt jelenti, hogy különböző szolgáltatások engedélyezésével és konfigurálásával mi tudjuk meghatározni, így teljes mértékben a saját igényeinkre szabhatjuk. A honeytrap keretein belül számtalan szolgáltatás közül válogathatunk (lásd: 4.3.2 fejezet Végleges honeypot kiválasztása), ezek közül számunkra főleg a követelményekben felsorolt, alacsony-közepes interakciójú protokollok relevánsak, így az alábbi szolgáltatások kerültek konfigurálásra:

SSH-Simulator: Ez egy SSH szerver szimulál, alapértelmezetten „root” felhasználónévvel és „password” jelszóval lehet belépni, de minden sikertelen próbálkozásnál is rögzíti a belépéshez használt adatokat. Sikeres belépésnél egy 16.04.1 LTS verziószámú Ubuntu-t szimulál, ahol az összes kiadott parancsot rögzíti, de mindegyikre „command not found” hibát ad vissza.//

SSH-Authenticator: Ez szintén egy SSH szerver szimulál, de ebbe nem lehet belépni, cserébe a belépési próbálkozáshoz használt összes adatot rögzíti. Hasznos lehet a jelszó feltöréséhez használt módszer megállapításához.

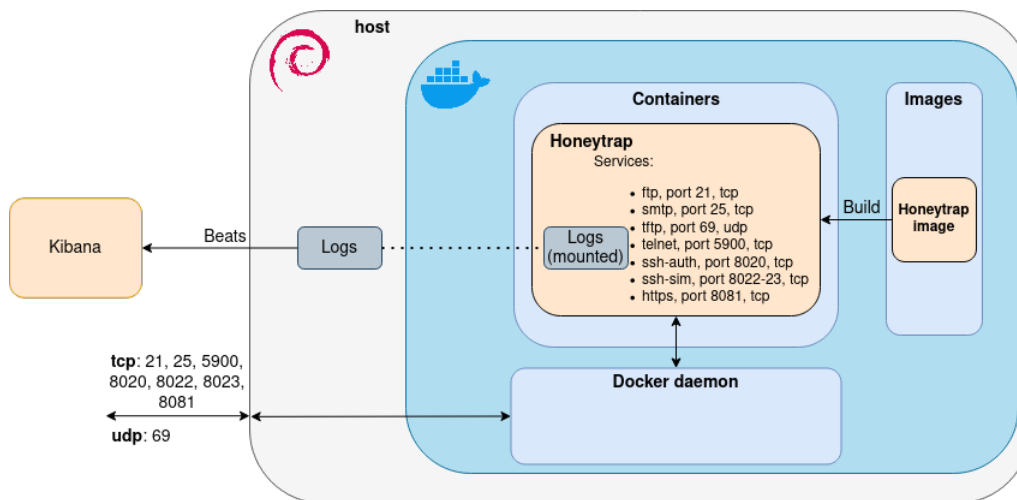
Telnet: Telnet szerver szimulál, minden belépéshez használt felhasználónevet és jelszót rögzít és elfogad. A rendszerbe belépve az összes kiadott parancsot rögzíti, de mindegyikre „command not found” hibát ad vissza.

FTP: Egy FTP szerver szimulál, belépés után az összes kiadott parancsot rögzíti és különféle 500-as (Internal Server Error) hibakóddal tér vissza.

SMTP: Csak egy SMTP szolgáltatást jelez a kiválasztott porton, a portra belépve bármilyen kiadott parancsot rögzít, és 500-as hibakóddal visszatér, majd megszakítja a kapcsolatot.

TFTP: UDP-n keresztül kommunikáló TFTP szerver szimulál, minden kiadott parancsot rögzít, file lekérésénél 256-os (file not found) hibakóddal tér vissza, minden másra „command not found” hibát ad.

HTTPS: Egyszerű, HTTPS kapcsolatot szimuláló szolgáltatás, amely a TLS kézfogás után időtúllépés miatt eldobja a kapcsolatot, miközben a támadás tényét, a támadó adataival együtt naplózza.



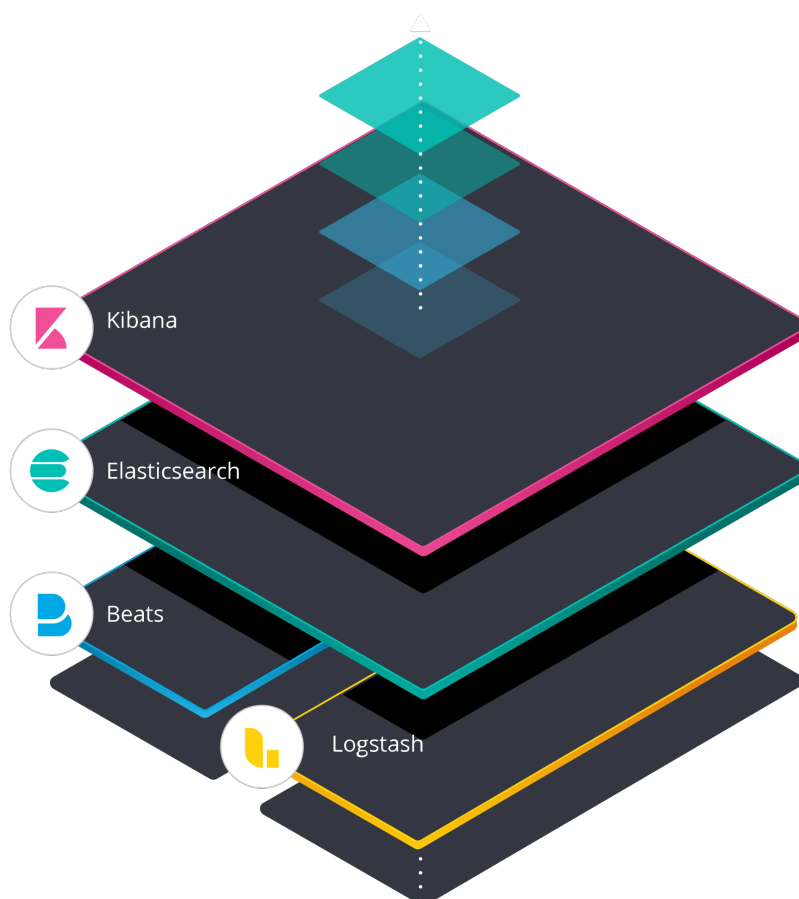
2. ábra. HoneyTrap beillesztése az architektúrába

4.5. T-Pot

Az általunk támasztott követelményeknek megfelelő honeypot kiválasztása mellett fontosnak tartjuk, hogy egy kutatási célú honeypotot is telepítsünk. Habár ez a támadások kivédésében nem feltétlenül a legpraktikusabb, oktatás és kutatás céljából nagyon hasznos eszköz. A kutatási honeypotként a T-Potot választottuk ki. [34] Ez az egyik legnépszerűbb ilyen jellegű nyílt forráskódú szoftvercsomag, melynek a fejlesztésében többek közt a Telekom is részt vett. Ez egy Debian operációs rendszeren futó Docker alapokon működő csomag melyet hat féleképpen lehet telepíteni. Minden telepítési mód más funkció ellátására hivatott, választható orvosi, vagy akár ipari honeypotok telepítése is. Számunkra a „Standard” telepítési mód a legmegfelelőbb, ez a mód a legnépszerűbb honeypotok implementálása mellett, számos hasznos eszközt telepít (cockpit, fatt, p0f & suricata, stb.), ezzel segítve a további hasznos adatok gyűjtését. Ilyen hasznos eszköz például a *Suricata*, amely egy hálózati forgalmat monitorozó IDS, segítségével bármilyen, a rendszert érintő forgalmat figyelni tudunk. Ez főleg a honeypotok portjait nem érintő forgalom elemzésekor hasznos, így egy sokkal átfogóbb képet kaphatunk a támadás menetéről és módszereiről. A T-Pot által létrehozott logkezelő felületen (ld. 4.6 *Elastic Stack*) elkészített infografikákat tartalmaz, melyek látványosan, jól nyomonkövethető módon szemléltetik a beérkező adatokat. Ilyen például leggyakrabban támadott portok, vagy éppen honeypotok listázása. Ezekre tovább szűrve, a célpontra specifikus adatokat - például támadási mintázatokat - hatékonyan tudjuk elemezni. [35]

4.6. Elastic Stack

A Biztonsági Műveleti Központban a naplóadatok központosított kezelésére előzetesen az *Elastic Stack* [36] (ELK Stack) került megvalósításra. Ez egy nyílt forráskódú elosztott rendszer, amely flexibilitása és skálázhatósága miatt alkalmazható szinte bármely informatikai rendszer adatainak centralizált gyűjtésére. Ez rendszer felel az naplóállományok gyűjtése mellett a rendszerezéséért, illetve tárolásáért. Három alapvető komponens alkotja az Elasticsearch, a Logstash valamint a Kibana.



3. ábra. Elastic Stack komponenseinek kapcsolata[36]

Az *Elasticsearch* egy kereső és elemző motor, amely az ELK leglényegesebb, kiadását tekintve legidősebb komponense. Legyen szó struktúrált vagy struktúrátlan szövegről, numerikus vagy térinformatikai adatokról az Elasticsearch képes keresést és elemzést végezni közel valós időben. A begyűjtött bejegyzéseket indexekben tárolja, amely további szűrési opciókat tesz lehetővé. Ezen indexek a felhasználási cél alapján definiálhatóak, egyszerű kezelésüket minták létrehozásával segíthetjük. Ezen mintákban definiálható, hogy milyen mezőkre bontsa fel a gyűjtött naplóadatokat, hogyan tárolja és archiválja. A további komponensek stabil és hatékony működését az Elasticsearch elengedhetetlen működése teszi lehetővé.

Az Elasticsearch munkáját segíti egy Logstash nevezetű adatgyűjtő motor. Három alapvető műveletet valósít meg, begyűjt, módosít és továbbít. Begyűjtés során az input lehet szöveges file, syslog vagy *beat*-ek amelyek folyamatok eseményeit tartalmazzák és a

Beat modul küldi őket. A módosításnál szöveges feldolgozás, mező létrehozás, átnevezés vagy kihagyás valósul meg, mely segítségével kívánt formára alakíthatóak a különböző rendszerekben keletkezett logadatok. Az utolsó folyamat, a továbbítás, ezen rész végzi az adatok megfelelő cél felé való küldését. A cél lehet a már említett Elasticsearch, egy fájl amelyben a módosított adatok írásra kerülnek, vagy akár más naplóadatkezelő megoldás is.

Végfelhasználói szempontból az ELK leglátványosabb alkotóeleme a Kibana. Feladata rendkívül sokszínű, felhasználói interfészt biztosít a teljes ELK rendszerhez. Segítségével a gyűjtött adatok alakot ölthetnek, átlátható és könnyen szűrhető formában jeleníthetők meg. A bejegyzések szabályok alapján való összekapcsolásával további grafikonokat, kimutatásokat készíthetünk, amely segít a rendszer történéseinek megértésében. Irányítópultok létrehozásával felgyorsíthatjuk az elemzést azáltal, hogy így egy szempillantás alatt észrevehető normál működéstől való eltérés, legyen szó akár valamilyen rendszerhibáról, akár egy biztonsági incidensről. Ezen felül itt végezhetjük szinte a teljes Elastic konfigurálását és menedzselését.

Az egyes végpontokon az Elasticsearch és a Logstash folyamatait megelőzve a Filebeat segíthet az adatok előkészítésében. Ez egy meglehetősen alacsony erőforrásigényű kliensalkalmazás, amely a keletkezett események és adatok aggregálását végzi, majd ezt követően küldi tovább a Logstashnek, vagy az Elasticsearch egy meghatározott indexe felé. A forrás a célhoz hasonlóan igény szerint testreszabható, megadható hol keresse a naplófájlokat vagy éppen milyen szabály mentén szelektálja azokat. A Filebeat lehetőséget biztosít arra, hogy az informatika rendszerünk bármely eszközéről gyűjthessünk adatokat, különösebb erőforrásigény nélkül. [36]

4.7. Honeypot által gyűjtött naplóállományok felhasználása

Egy informatikai rendszerben meglehetősen sok adat keletkezik, már az eszközök alapállapotú futása közben is, ezért egy honeypot esetén különösen fontos megválasztani a felderítés szempontjából fontos naplóállományok csoportját. Egy ilyen csapda esetében érdemes azokkal a jelzésekkel foglalkozni, amelyek egy interakció során jönnek létre, mert ezek nyújthatnak információt a támadóról. Természetesen egyéb naplóadatokat is tulajdoníthatunk jelentőséget, például egy egyszerű „Igen, életben vagyok” jellegű bejegyzésen keresztül láthatjuk, hogy a rendszerünk továbbra is működik és várja a beérkező kéréseket.

Az adatok a konténerizált rendszerbe egy a gazdagépről felcsatolt könyvtárban kerülnek tárolásra, így az adatok lokálisan is folytatódhatnak lehet abban az esetben is ha a konténert valamilyen hiba folytán újra kell telepíteni. Innen a naplóadatokat elszállításáért egy *Filebeat* [36] nevű logszállító alkalmazás felel, amely a definiált elérési útvonalakról gyűjti a bejegyzéseket, és az Elasticsearch megfelelő indexeibe juttatja el azokat. Eközben megtörténik a fájlokban lévő adatok meghatározott szabály alapján történő mezőkre bontása. Ez az adathalmaz már könnyen kereshető és szűrhető, mely lehetőséget ad arra, hogy a Kibana felületén irányítópultokba rendezve megjelenítsük, további folyamatokban automatizálva használjuk, kialakíthassunk egy szolgáltatási- és riasztási rendszert.

4.8. Szükséges kiegészítés

A legtöbb honeypot, köztük a honeytrap sem képes a rejtett port letapogatások észlelésére. Ez főleg azért probléma, mert a támadók eszköztárában ez egy nagyon gyakori felderítési módszer. Port letapogatás során a támadó azt próbálja kideríteni, hogy a céleszköz milyen portokat tart nyitva kommunikáció céljára, illetve azokon a portokon milyen szolgáltatást futtat. Ezt általában úgy éri el, hogy célzott csomagokat küld egy portra, és az arra adott válaszból próbálja kikövetkeztetni a számára releváns információt. A letapogatás észlelhetőségének csökkentésére különböző technikák jöttek létre, ilyen például a TCP SYN letapogatás, amely során a támadó nem hagyja, hogy kiépüljön a TCP kapcsolat, a háromlépéses TCP kézfogás befejezésére sosem kerül sor. Egy ilyen felderítés kiválóan alkalmas a portok állapotának kiderítésére és mivel a kapcsolat sosem épül fel teljesen, azt csak az operációs rendszer szintjén lehet észlelni. Hogy ezeket a próbálkozásokat is rögzíteni lehessen, operációs rendszer szinten naplózzuk ezeket iptables szabályok létrehozásával. Ennek köszönhetően bármilyen, a honeytrap portjaihoz történő letapogatási kísérlet is észlelhető lesz.

5. Implementáció

5.1. Docker

A honeytrapet docker alapokon implementáljuk. Ez egy széleskörben elterjedt virtualizációs megoldás. Működését tekintve, a docker konténerizált szolgáltatásokat futtat, amikhez a docker daemon keresztül lehet hozzáférni. Ez kezeli a kliens és a konténerek közti API kérélmeket és magukat a konténereket, a docker képfájlokat melyek új konténerek létrehozásához szükségesek, illetve a konténereket összekötő hálózatokat. Számunkra konténerekben futtatott alkalmazások előnye, hogy egy elszigetelt környezetet jelentenek, amelyen keresztül a támadó nem fér hozzá más alkalmazásokhoz, fájlokhoz. További előnye, hogy moduláris, az így futtatott alkalmazások jól elkülöníthetőek, egyszerűen indíthatóak és leállíthatóak, az erőforrásokat pedig hatékonyan osztja szét köztük. Konténereket docker képfájlokkal lehet létrehozni, ezeket vagy a helyi adatbázisban tárolja, vagy ha ott nem talál egy adott képfájlt, azt a hálózatról próbálja letölteni. Hasznos tulajdonsága, hogy támogatja a legelterjedtebb operációs rendszereket: macOS, windows, CentOS, Debian, Fedora, Raspbian, Ubuntu[37] és ezek különbségeit a működési elvéből adódóan elfedi a futó szolgáltatások elől.

5.2. Függőségek

A kiválasztott eszközök telepítéséhez a dockeren kívül több más eszközre is szükségünk lesz, ezeket az alábbiakban fogjuk részletesen bemutatni.

net-tools: Ez egy hálózati statisztikai eszköz, a segítségével könnyen elemezhetjük az eszköz kapcsolatait, nyitott portjait, a használt protokollokat, illetve azt, hogy azokat mely alkalmazások használják.[38]

firewalld: A firewalld egy linux rendszerekhez fejlesztett tűzfalkezelő eszköz. Nagyban megkönnyíti a hálózati szabályok, tűzfal zónák kialakítását. [39]

iptables: Hálózati csomagok szabályozására fejlesztett program. Segítségével a hálózati kapcsolat figyelembevételével tudjuk kezelni a hálózati csomagok áramlását.

docker: Afentiek mellett a 5.1-es fejezetben részletezett dockerre is szükségünk lesz, a honeytrap és T-Pot futtatásához.

5.3. Honeytrap

5.3.1. Telepítés

A honeytrap implemenálása egy debian 10 operációs rendszert futtató eszközön fogjuk végrehajtani. Első lépésként szükséges a docker telepítése, ezt a dokumentációjában található telepítési segédlet alapján könnyen végre lehet hajtani. [31] A telepítés sikerességét a beépített „hello world” képfájl futtatásával tudjuk ellenőrizni. Mielőtt létrehozzuk a honeytrapet futtató konténert a honeytrap képfájlja alapján, fontos előre tudnunk milyen paraméterekkel kell futtatni a szolgáltatást. Ez azért fontos, mert a már egyszer létrehozott konténert sokkal nehezebb utólag konfigurálni, mintha azt első fordításkor tennénk meg.

Jelen esetben két fontos paramétert kell eldöntenünk:

1. A honeytrap konténerben létrehozott logokat valahogy el kell juttatni egy réteggel feljebb, a debian fájlrendszerébe. Szerencsére a dockerben lehetőség van arra, hogy egy, a konténerben található mappát a host rendszer fájlrendszerébe fel lehessen csatolni, annak minden tartalmával együtt.
2. Második lépésként fontos előre tudni, hogy milyen portokra szeretnénk beállítani a honeytrap szolgáltatásait, illetve azok milyen szállítási rétegbeli protokollt használnak, esetünkben ez a 4.4 fejezetben részletezettek szerint fog történni. Ez azért fontos, mert a konténer fordításakor kell megadni, hogy mely portokon továbbítsa a docker daemon az információt a szolgáltatások számára.

Érdeemes, de nem kötelező a konténernek nevet választani, ha ezt kihagyjuk, a docker véletlenszerűen fogja őket elnevezni, ami nem feltétlenül a legátláthatóbb. Amint ezek véglegesítésre kerültek, a szükséges konténert egy paranccsal létre lehet hozni.

```
root@debian:~# sudo docker run -p 8022:8022 -p 5900:5900 -p 21:21 -p 8080:8080 -p 26:26
-p 8020:8020 -p 69:69/udp -p 8023:8023 -v /var/log/honeylog:/var/log/honeylog --name w
innie_the_pooh honeytrap/honeytrap:latest
Unable to find image 'honeytrap/honeytrap:latest' locally
latest: Pulling from honeytrap/honeytrap
188c0c94c7c5: Pull complete
elc826e79407: Pull complete
d58a33121ace: Pull complete
048810fe0acf: Pull complete
1a5bdcad466a: Pull complete
0f258b2cadc9: Pull complete
Digest: sha256:6a8a9f7fc31476b2f405d86d787a4b157af95739548839e4294cd70c12473caf
Status: Downloaded newer image for honeytrap/honeytrap:latest
2021/04/19 16:44:20 Using config file /config/config.toml
Honeytrap starting (clur6p4kval000b55ts0)...
Version: DEVELOPMENT.GOGET (DEVELOPMENT.)
16:44:20.153 honeytrap/server ► DEBU 002 Using datadir: /data
16:44:20.153 services ► DEBU 003 Could not load private-key, generating one.
16:44:20.368 honeytrap/server ► INFO 004 Configured service ssh-simulator (ssh-simulat
or)
16:44:20.368 honeytrap/server ► INFO 005 Configured service telnet (telnet)
16:44:20.368 honeytrap/server ► INFO 006 Configured port tcp/:8022
16:44:20.368 honeytrap/server ► INFO 007 Configured port tcp/:8023
16:44:20.368 honeytrap/server ► INFO 008 Configured port tcp/:5900
16:44:20.368 honeytrap/server ► WARN 009 Unrecognized keys in configuration: [service.t
elnet.image service.telnet.server-name]
16:44:20.369 listeners/socket ► INFO 00a Listener started: tcp/:8022
16:44:20.369 listeners/socket ► INFO 00b Listener started: tcp/:8023
16:44:20.369 listeners/socket ► INFO 00c Listener started: tcp/:5900
```

4. ábra. HoneyTrap konténer létrehozása

5.3.2. Konfigurálás

A konfigurálás két lépésből áll, először a honeytrapen kell beállítani, hogy pontosan milyen szolgáltatásokat milyen paraméterekkel futtasson, majd a host gépen kell engedélyezni, hogy ki és beengedje a forgalmat a honeytrap által használt portokon. A honeytrap konfigurálásához dockeren keresztül be kell lépni a konténeren belülre. Ekkor a konténeren belül elindul a konténerhez tartozó rendszerhéj (system shell), melyen keresztül beülről lehetséges a konténer módosítása. A parancs kiadása után, a honeytrap dokumentációja által megadott [23] konfigurációs fájlban tudjuk beállítani a szolgáltatásokat. A honeytrap toml formátumú konfigurációs fájlt vár el, ez alapján, illetve a dokumentációjában található séma szerint a nekünk szükséges konfiguráció a

következőképpen néz ki. Fontos kiemelni, hogy a naplófájl elérési útjaként meg kell adni a tükrözött mappát, hogy később a konténeren kívülről is hozzá lehessen férni. A második lépésben, ha a docker automatikusan nem nyitotta ki a használt portokat, engedélyezni kell a host gépen a használt portok megnyitását. Ezt a firewalld segítségével könnyen megtehető.

5.3.3. Rejtett portscan detektálásának implementálása

A rejtett portletapogatások észleléséhez, mindegyik honeytrap által használt portra érkező csomagot logolni fogunk a rendszer logokba. Ehhez az iptables segítségével az INPUT lánc elejére helyezzük a logolást végző utasításokat, ami „connection_to_honeypot” felirattal naplóz minden releváns portra beérkező csomagot.

```
Apr 22 11:08:01 debian kernel: [ 348.950588] connection_to_honeypotIN=lo OUT= MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00 SRC=127.0.0.1 DST=127.0.0.1 LEN=44 TOS=0x00 PREC=0x00 TTL=48 ID=2517 PROTO=TCP SPT=55317 DPT=8022 WINDOW=1024 RES=0x00 SYN URG=0
```

5. ábra. SYN letapogatás naplózása a 8022-es porton

5.3.4. Funkcionális tesztek

A Honeytrap szolgáltatásainak funkcionális teszteléséhez az összes szolgáltatás felé egy kérést indítunk. Amennyiben ezek a kérések rögzítésre kerülnek, a teszt sikeresnek tekinthető.

SSH-Simulator: Ennél a szolgáltatásnál a belépéshez használt adatok, illetve a konfiguráció során megadott felhasználónév - jelszó párossal történő belépés után a szimulált rendszeren kiadott parancsok kerülnek rögzítésre. Ezt egy egyszerű *"ssh -p 8022 felhasználónév@IP cím"* parancssal tesztelhetjük:

```
{ "category": "ssh", "date": "2021-11-10T21:54:55.928295649Z", "destination-ip": "172.17.0.2", "destination-port": 8022, "sensor": "services", "source-ip": "145.236.145.97", "source-port": 45011, "ssh.password": "TesztJelszó", "ssh.sessionid": "c663vmo7hoo000fiatb0", "ssh.username": "TesztNév", "token": "c6608cg7hoo000das1bg", "type": "password-authentication" }, { "category": "heartbeat", "date": "2021-11-10T21:55:08.104158071Z", "sensor": "honeytrap", "sequence": 500, "token": "c6608cg7hoo000das1bg", "type": "info" }, { "category": "heartbeat", "date": "2021-11-10T21:55:38.103842199Z", "sensor": "honeytrap", "sequence": 501, "token": "c6608cg7hoo000das1bg", "type": "info" }, { "category": "ssh", "date": "2021-11-10T21:55:43.568766241Z", "destination-ip": "172.17.0.2", "destination-port": 8022, "sensor": "services", "source-ip": "145.236.145.97", "source-port": 45106, "ssh.password": "password", "ssh.sessionid": "c663vmo7hoo000fiatbg", "ssh.username": "root", "token": "c6608cg7hoo000das1bg", "type": "password-authentication" }, { "category": "ssh", "date": "2021-11-10T21:55:43.608240903Z", "destination-ip": "172.17.0.2", "destination-port": 8022, "sensor": "services", "source-ip": "145.236.145.97", "source-port": 45106, "ssh.payload": "AAAA0n02X0t1I1mVbG9yAAAAeAAAA4AAAKAAAB4AAAAuBAAAlgIAAACMAAA=", "ssh.request-type": "pty-req", "ssh.sessionid": "c663vmo7hoo000fiatbg", "token": "c6608cg7hoo000das1bg", "type": "ssh-request" }, { "category": "ssh", "date": "2021-11-10T21:55:43.608402364Z", "destination-ip": "172.17.0.2", "destination-port": 8022, "sensor": "services", "source-ip": "145.236.145.97", "source-port": 45106, "ssh.payload": "", "ssh.request-type": "shell", "ssh.sessionid": "c663vmo7hoo000fiatbg", "token": "c6608cg7hoo000das1bg", "type": "ssh-request" }, { "category": "ssh", "date": "2021-11-10T21:55:52.001162668Z", "destination-ip": "172.17.0.2", "destination-port": 8022, "sensor": "services", "source-ip": "145.236.145.97", "source-port": 45106, "ssh.command": "TesztCommand", "ssh.sessionid": "c663vmo7hoo000fiatbg", "token": "c6608cg7hoo000das1bg", "type": "ssh-channel" }
```

6. ábra. Hibás, majd helyes bejelentkezés kiadott parancsokkal az SSH-Simulator szolgáltatáson

SSH-Authenticator: Az előzőhöz hasonlóan ez is egy SSH protokollon alapuló szolgáltatás, de itt nincs szimulált környezet, csak a hitelesítő adatok kerülnek naplózásra. Az előbbihez hasonlóan, ezt a *"ssh -p 8020 felhasználónév@IP cím"* parancssal teszteljük:

```
{ "category": "ssh", "date": "2021-11-10T22:02:56.522774388Z", "destination-ip": "172.17.0.2", "destination-port": 8020, "sensor": "services", "source-ip": "145.236.145.97", "source-port": 52747, "ssh.password": "TesztJelszó", "ssh.username": "TesztNév", "token": "c6608cg7hoo000das1bg", "type": "password-authentication" }
```

7. ábra. Bejelentkezési kísérlet az SSH-Authenticator szolgáltatásra

Telnet: Telnet hozzáférést szimulál, bármilyen hitelesítő adattal tovább engedi a támadót, azokat rögzíti. A szimulált környezetben minden kiadott parancsot rögzít.

Ennek teszteléséhez a "telnet IP cím 5900" parancsot adtuk ki:

```
{ "category": "telnet", "date": "2021-11-10T22:05:45.851541213Z", "destination-ip": "172.17.0.2", "destination-port": 5900, "sensor": "services", "source-ip": "172.17.0.1", "source-port": 46494, "telnet.sessionid": "c6644e87hoo000fiatc0", "token": "c6608cg7hoo000das1bg", "type": "connect" }
{ "category": "telnet", "date": "2021-11-10T22:05:56.903151367Z", "destination-ip": "172.17.0.2", "destination-port": 5900, "sensor": "services", "source-ip": "172.17.0.1", "source-port": 46494, "telnet.password": "TesztJelszó", "telnet.sessionid": "c6644e87hoo000fiatc0", "telnet.username": "TesztNév", "token": "c6608cg7hoo000das1bg", "type": "password-authentication" }
{ "category": "telnet", "date": "2021-11-10T22:06:02.951189961Z", "destination-ip": "172.17.0.2", "destination-port": 5900, "sensor": "services", "source-ip": "172.17.0.1", "source-port": 46494, "telnet.command": "TesztCommand", "telnet.sessionid": "c6644e87hoo000fiatc0", "token": "c6608cg7hoo000das1bg", "type": "session" }
```

8. ábra. Bejelentkezés, majd kiadott parancs a Telnet szolgáltatáson

FTP: A telnethez hasonlóan itt is a bejelentkezéshez használt, valamint a szimulált környezetben kiadott parancsok kerülnek rögzítésre. Teszteléshez az "ftp IP cím" parancsot használtuk:

```
{ "category": "ftp", "date": "2021-11-10T22:09:18.831574394Z", "destination-ip": "172.17.0.2", "destination-port": 21, "ftp.command": "OPTS UTF8 ON", "ftp.sessionid": "3cd06aa4b522f4ac6ace", "sensor": "services", "source-ip": "80.99.9.174", "source-port": 41562, "token": "c6608cg7hoo000das1bg" }
{ "category": "ftp", "date": "2021-11-10T22:09:31.362057009Z", "destination-ip": "172.17.0.2", "destination-port": 21, "ftp.command": "USER TesztCommand", "ftp.sessionid": "3afbe4a88f4b716822c7", "sensor": "services", "source-ip": "80.99.9.174", "source-port": 41576, "token": "c6608cg7hoo000das1bg" }
{ "category": "heartbeats", "date": "2021-11-10T22:09:38.103549551Z", "sensor": "honeytrap", "sequence": 529, "token": "c6608cg7hoo000das1bg", "type": "info" }
{ "category": "ftp", "date": "2021-11-10T22:10:06.709015737Z", "destination-ip": "172.17.0.2", "destination-port": 21, "ftp.command": "OPTS UTF8 ON", "ftp.sessionid": "0e54c082d05ef098adf6", "sensor": "services", "source-ip": "167.248.133.60", "source-port": 50080, "token": "c6608cg7hoo000das1bg" }
{ "category": "heartbeats", "date": "2021-11-10T22:10:08.104261017Z", "sensor": "honeytrap", "sequence": 530, "token": "c6608cg7hoo000das1bg", "type": "info" }
{ "category": "ftp", "date": "2021-11-10T22:10:11.730774672Z", "destination-ip": "172.17.0.2", "destination-port": 21, "ftp.command": "PASS TesztJelszó", "ftp.sessionid": "6d8284509ea264589647", "sensor": "services", "source-ip": "80.99.9.174", "source-port": 41556, "token": "c6608cg7hoo000das1bg" }
{ "category": "ftp", "date": "2021-11-10T22:10:17.035987534Z", "destination-ip": "172.17.0.2", "destination-port": 21, "ftp.command": "PASSTesztJelszó", "ftp.sessionid": "0f03f2de20c27448b6df", "sensor": "services", "source-ip": "145.236.145.97", "source-port": 54183, "token": "c6608cg7hoo000das1bg", "type": "session" }
```

9. ábra. Bejelentkezés, majd kiadott parancs az FTP szolgáltatáson

TFTP: A TFTP egy egyszerű szolgáltatás csak azt jelzi hogy valaki csatlakozott hozzá. Ennek teszteléséhez a netcat programmal a "nc -vu IP cím 69" parancsot adjuk ki

```
22:36:48.112 honeytrap/server @ DEBU 0cd Accepted connection for 145.236.145.97:55674 => [::]:69
22:36:48.112 honeytrap/server @ DEBU 0ce Handling connection for 145.236.145.97:55674 => [::]:69 tftp(tftp)
22:36:48.112 honeytrap/server @ DEBU 0cf Disconnected connection for 145.236.145.97:55674 => [::]:69
```

10. ábra. Csatlakozás a TFTP szolgáltatáshoz

HTTPS: Mivel ez a szolgáltatás csak egy időtűllépésre futó TLS kézfogást szimulál, csak a támadás tényét rögzíti, a támadó IP címével. A tesztelést egy curl parancssal hajtjuk végre, "curl IP cím:8080":

```
{ "category": "https", "date": "2021-11-10T22:22:34.2153374Z", "destination-ip": "172.17.0.2", "destination-port": 8080, "https.ja3-digest": "", "https.server-name": "", "sensor": "services", "source-ip": "145.236.145.97", "source-port": 55520, "token": "c6608cg7hoo000das1bg", "type": "handshake-failed" }
{ "category": "heartbeats", "date": "2021-11-10T22:22:38.103782653Z", "sensor": "honeytrap", "sequence": 555, "token": "c6608cg7hoo000das1bg", "type": "info" }
isconnected connection for 145.236.145.97:55520 => 172.17.0.2:8080
```

11. ábra. Csatlakozás a HTTPS szolgáltatáshoz

SMTP: Az SMTP port a csatlakozás után kiadott parancsokat rögzíti. A tesztelést a "nc -v IP cím 26" parancssal végezzük el:

```
{ "category": "smtp", "date": "2021-11-10T22:32:28.353126414Z", "destination-ip": "172.17.0.2", "destination-port": 26, "sensor": "services", "smtp.line": "TesztCommand", "source-ip": "145.236.145.97", "source-port": 50141, "token": "c6608cg7hoo000das1bg", "type": "input" }
```

12. ábra. Kiadott parancsok a TELNET szolgáltatáson

5.4. T-Pot

Mivel a T-Pot egy teljes rendszer különféle egymással összhangban működő szolgáltatásokból összeállítva, a fejlesztők létrehoztak egy Debian képfájlba ágyazott telepítőt. Esetünkben ez a legpraktikusabb, első lépésként egy Debian rendszert telepít, majd első indításkor elkezd a T-Pot implementálását. Ennél 6 telepítési mód közül választhatunk (Standard, Sensor, Industrial, Collector, NextGen, Medical), amikből a 4.5-ös fejezetben részletezett okok miatt a „Standard” telepítési módot fogjuk választani. Konfigurálásra ez esetben nincs szükség, a telepítő automatikusan konfigurálja a telepített szolgáltatásokat. A T-Pot, a telepítés során a támadók megtévesztése érdekében az alapértelmezett 22-es SSH port mögé is honeypot szolgáltatást helyez, a rendszert a 64295-as porton SSH-n, 64297-es porton grafikus felületen keresztül lehet elérni a telepítés során megadott felhasználónév – jelszó párossal. Ezeket a portokat az autentikációt a Fail2Ban szoftver segítségével biztosítja, amely nyers erő elven alapuló támadásokkal szemben biztosít védelmet. Az egyes szolgáltatások szabadon konfigurálhatók, külön-külön engedélyezhetők, valamint lehetőségünk van további alkalmazások vagy honeypotok hozzáillesztésére is.

```
#####
### T-Pot Installer for Debian (Stable) ###
#####

### Checking for active services.

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       User        Inode      PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      0           19037       521/sshd
tcp        0      0 127.0.0.1:631            0.0.0.0:*               LISTEN      0           1062558     7641/cupsd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      0           69470       9186/exim4
tcp6       0      0 :::22                   :::*                    LISTEN      0           19039       521/sshd
tcp6       0      0 :::631                   :::*                    LISTEN      0           1062557     7641/cupsd
tcp6       0      0 :::25                    :::*                    LISTEN      0           69471       9186/exim4
tcp6       0      0 :::80                    :::*                    LISTEN      0           68279       7576/apache2
udp        0      0 0.0.0.0:631             0.0.0.0:*               0           1062565     7643/cups-browsed
udp        0      0 0.0.0.0:5353            0.0.0.0:*               112         17844       394/avahi-daemon: r
udp        0      0 0.0.0.0:47251           0.0.0.0:*               112         17846       394/avahi-daemon: r
udp6       0      0 :::45982                :::*                    112         17847       394/avahi-daemon: r
udp6       0      0 :::5353                  :::*                    112         17845       394/avahi-daemon: r

### Please review your running services.
### We will take care of SSH (22), but other services i.e. FTP (21), TELNET (23), SMTP (25), HTTP (80), HTTPS (443), etc.
### might collide with T-Pot's honeypots and prevent T-Pot from starting successfully.

Continue [y/n]? █
```

13. ábra. T-Pot telepítése során a portok áthelyezése

5.5. Naplózás konfigurálása

5.5.1. HoneyTrap naplózása

A HoneyTrap naplóadatai egyetlen fájlba íródnak, amely a konténerbe egy, a gazdagépen lévő mappa felcsatolásával történik. Naplózás során előztes szűrés konfigurálható, így az egyes bejegyzések prioritási szint szerint különböző formában is írásra kerülhetnek. Definiálható fájl vagy konzolra való kiíratás kimenetként, illetve a prioritás debug, info, notice, warning, error és critical szintekben van megállapítva.[40] Konfigurálható továbbá, hogy a honeypot szolgáltatások mellett mely rendszer eseményekről keletkezzen bejegyzés. Esetünkben az interekciós logok mellett egy „heartbeat” is megjelenik meghatározott rendszerességgel, amely azt a célt szolgálja, hogy információval rendelkezessünk arról, hogy a HoneyTrap szolgáltatásai működnek.

```
$ docker run -v /var/log/honeytrap:/var/log/honeytrap
```

14. ábra. Könyvtár felcsatolása konténerbe

5.5.2. T-Pot naplózása

A T-Pot minden egyes külön konténerben futó honeypotja az HoneyTraphez hasonló példát követ naplóadatok tárolását tekintve. A hostról felcsatolt könyvtárba írják a szükséges adatokat. Ezen elérési utak telepítéskor módosíthatóak, vagy az egyes docker képfájlok újrabuildelése esetén tehető meg. Az egyes honeypotok naplóadat generálási szempontból meglehetősen eltérőek, de általánosságban elmondható, hogy mindegyik esetében lehetőségünk van a célkönyvtárat, formátumot és prioritási szintet definiálni, melyeket egy konfigurációs fájlban tehetünk meg. Ezen fájlok a T-Pot működése során módosíthatóak, a beállítások a konténer újraindításával lépnek életbe.

```
[honeypot]
log_path = log

[output_jsonlog]
enabled = true
logfile = log/ipphoney.json
epoch_timestamp = false

[output_textlog]
enabled = false
logfile = log/ipphoney-textlog.log
format = text

[output_crashreporter]
enabled = false
debug = false
```

15. ábra. Egy honeypot konfigurálható naplózási lehetőségei

5.6. Honeypotok bekötése *Elasticsearch*-be

A naplóadatok honeypotról történő elszállításaért a Filebeat felel, melynek konfigurálását három részre oszthatjuk: input, szűrés és output. Az input során definiáltuk, a honeypotok által gyűjtött logfájlok elérési útját. A Filebeat ezeket a forrásokat figyeli és változás esetén elküldi az új bejegyzéseket. A szűrésben a megvalósítható az adatok csoportokba bontása, amelyet egy, a honeypot nevének megfelelő típus megadásával tettünk meg. Végezetül az outputban kerül beállításra, hogy pontosan hova kerüljön továbbításra az összegyűjtött és felcímkézett adat. Lehetőségünk van közvetlenül az Elasticsearchbe juttatni, vagy további feldolgozásra a Logstashbe küldeni. Utóbbi mellett döntöttünk, mivel voltak olyan bejegyzések, amelyeket a Filebeat nem tudott különálló mezőkre bontani.

A Logstash a Filebeathez képest sokkal intenzívebb, precízebb szűrésre és feldolgozásra képes, szerkezetünk viszont egészen hasonló. A Logstash is meglehetősen sok forrást típust különböztet meg, amelyből - az előbbiekben leírtak alapján kikövetkeztethető módon - a Filebeat által biztosított *beat* típust alkalmaztuk. A forrás variáns megadása után, a korábban hozzárendelt típus alapján további szűrést végezhetünk az adathalmazon. Meghatározhatjuk, hogy a már független mezőkben szereplő adat milyen formátumot kövessen (pl. időbélyeg) vagy esetleg módosíthatjuk a mezők nevét, hozzáfűzhetünk tartalmukhoz. Ahogyan esetünkben is, többsoros bejegyzések keletkezése során, úgynevezett *grok* mintát kell létrehoznunk, amely alapján a Logstash elvégezheti a mezőkre bontást. A minta leköveti a bejegyzés szerkezetét és feldarabolja a karakterláncot, illetve ezen részekből új mezőket hoz létre. A mintának tartalmaznia kell az egyes részekhez tartozó új mezők nevét, a benne tárolt adat típusát[41]. A feldolgozott, teljes egészében cellákra bontott adat az Elasticsearchbe kerül.

```

input {
  file {
    path => "/tmp/access_log"
    start_position => "beginning"
  }
}

filter {
  if [path] =~ "access" {
    mutate { replace => { "type" => "apache_access" } }
    grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }
  }
}

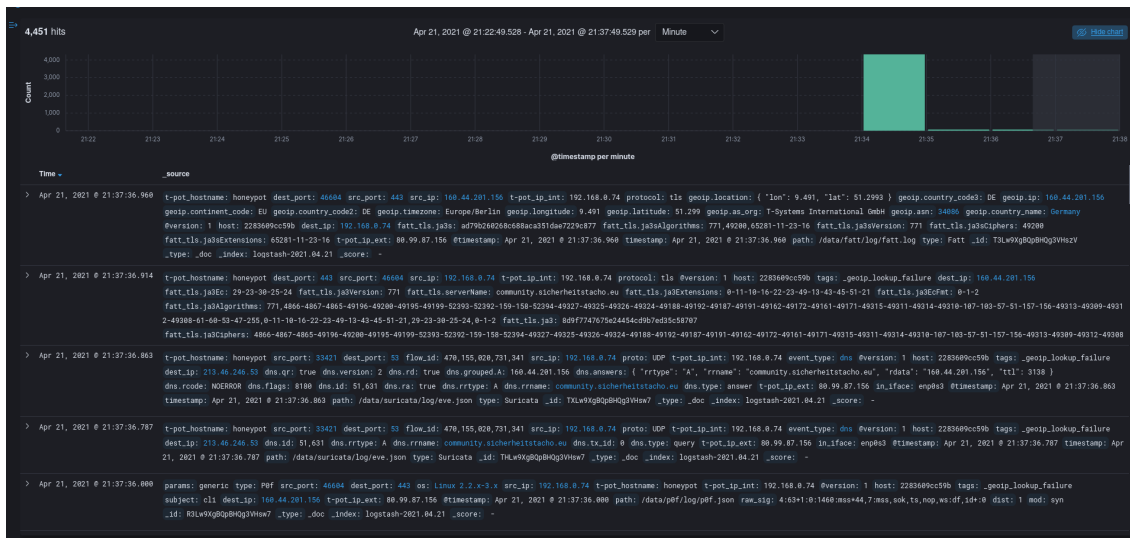
date {
  match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
}

output {
  elasticsearch {
    hosts => ["localhost:9200"]
  }
}

```

16. ábra. Logstash konfiguráció minta[42]

Az átláthatóságot és hatékony kereshetőség érdekében az Elasticsearchben meg kell határozni, hogy a beérkező rekordok melyik indexhez kerüljenek hozzárendelésre. Az indexeket sablonok létrehozásával könnyen kezelhetjük, ezek definiálják mi szerepelhet mezőikben. Honeypot adatait - hasonló formátumuk és egyező célfelhasználásuk miatt - egyetlen indexhez rendeltük, melyek így már egységesen szűrhetőek rekordjaik alapján, valamint konzisztens alapot biztosítanak a rájuk épülő szolgáltatásoknak.



17. ábra. Egy index bejegyzései a Kibana felületén

5.7. Honeypotra épülő szolgáltatások implementációja

Az adatok vizualizálását a Kibana végzi, mely lehetőségeket nyújt az Elasticsearchben rendszerezett adatok irányítópultokba való rendezésére. A T-Pot rendszere alapértelmezetten biztosít ilyen irányítópultok készítéséhez való objektumokat, amelyeket importáltunk a Kibanába. Ezt természetesen igény szerint testreszabhatjuk, valamint az indexelt adatokra építve teljesen új is készíthető.

A irányítópultok panelekből épülnek fel, amelyek egy kettő vagy több mező adattartalmának összerendelésével hozható. Ez a kimutatási felület sokféle formát ölthet, támogatott a különféle oszlop-, kör- és vonaldiagramok készítése, továbbá adatainkat automatikusan táblázatba rendezheti vagy akár - geolokációs adatok esetén - elhelyezheti egy térképen. Amennyiben nem vagyunk elégedettek a Kibana alapértelmezéseivel, manuálisan elvégezhetjük a panelek finomhangolását, szűrést végezhetünk, állíthatjuk a diagrammok osztását és természetesen a színezetét.



18. ábra. Az irányítópultokat alkotó panel készítése

A panelek elrendezése igények alapján tetszőlegesen állítható, az Elasticsearch bármely indexének rekordjai felhasználhatók egy irányítópultban. Honeypot rendszerünk esetében több ilyen nézetet is létrehoztunk, amelyek eltérő szempontból mutatják az összegyűjtött adatot. A nézetek között szerepel olyan, amely szemlélteti a teljes honeypot rendszer hozzávetőlegesen minden adatát (lásd 19. ábra), ennek célja, hogy pillantás alatt észrevehető legyen bármely alrendszerért támadás, illetve egy összesítet képet ad az átlag támadások formájáról, esetleg helyéről.



19. ábra. Összesített irányítópult a Kibana felületén

A támadók technikáinak részletes elemzéséhez egy sokkal inkább szűrt

adathalmazból érdemes kimutatásokat készíteni. A szűrés alapja, lehet egyetlen honypotra vagy akár egy konkrét szolgáltatásra vonatkozó, így sokkal pontosabb képet kaphatunk arról mivel is próbálkozott a betokodó. Ez az egyes honeypotoknál a szolgáltatások különbsége miatt meglehetősen eltérő eredményt hozhat,de egy erre megfelelő példa a 22. ábrán tekinthető meg a következő fejezetben.

6. Tesztelés

6.1. Módszer kiválasztása

Az elkészült rendszerünk tesztelésére többféle módszer közül is választhatunk. Talán legidőigényesebb, viszont annál kevésbé hatékony módszer, ha mi magunk állunk neki kidolgozni a rengeteg támadási formát és ezeket manuálisan végrehajtjuk. Egy optimálisabb esetben bevethetünk egy automatikus eljárást a támadások végrehajtására. Amennyiben az internet felé is elérhetővé tesszük, biztosan megtalálják automata szkennerek, ez is mutathat egyfajta képet a fenyegető veszélyekről, ezzel kapcsolatos tapasztalatainkat a 6.4 fejezetben részletezzük.

Azonban mi ennél egy átfogóbb és életszerűbb eredményeket hozó tesztelést szerettünk volna végrehajtani. Arra voltunk kíváncsiak, hogy mennyire hatékony a honeypot rendszerünk különböző valós támadókkal szemben, akik különféle metodikával támadnak, más sérülékeny szolgáltatást helyeznek előtérbe és egymástól eltérő idő után próbálkoznak inkább egy másik szolgáltatással vagy módszerrel. Ezen adatok segítségével szerettünk volna megbecsülni, hogy egy SOC-ban „védekező” csapatnak mennyi idő nyerhető az első honeypotra érkező kéréstől számítva. Az eredmények alapján további optimalizációs lehetőséget adhat, hogy látjuk mely honeypot-ok a legnépszerűbbek, melyek kötötték le leginkább a támadók figyelmét.

A módszerünk az volt, hogy a honeypot rendszerbe egy „zászlófoglaló” (*capture the flag*) játékot implementáltunk és ezt versenyként meghirdettük egyetemi hallgatók számára. Így kellően eltérő felderítési és támadási technikákat figyelhettünk meg és még tanulási, gyakorlási lehetőséget is biztosítottunk hallgatótársaink számára.

6.2. Tesztelés megvalósítása

A verseny, illetve ezzel együtt az eredményeket szolgáltató tesztkörnyezet a *Tpot* honeypot rendszerből, valamint három kihívás szolgáltatásból áll. A *Honeytrap* tesztből való kihagyása mellett azért döntöttünk, mivel a *Tpot* is rendelkezik hasonló szolgáltatásokkal, illetve kutatási honeypot lévén nagyobb mennyiségű információval tud szolgálni.

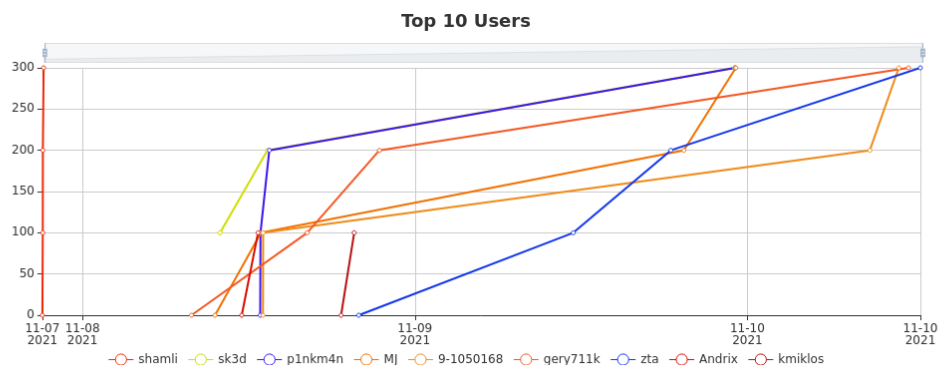
A kihívásokat - amelyeket a honeypotokhoz hasonlóan - szintén egy-egy docker konténerben futtatunk, három széles körben használt szolgáltatás, az ssh, ftp és http sérülékeny konfigurálásával hoztuk létre. Mindhárom esetben elrejtettünk egy „zászlót” (*flag*), amely a hallgatók számára meghirdetett játék alapját adja. Ezen kihívásokat a honeypot konténerek közé rejtettük, így a rájuk váró feladat nem csupán annyi volt, hogy valamilyen módon kinyerjék a zászlót egy meghatározott szolgáltatástól. Az igazi megmérettetés az volt, hogy hogyan találják meg ezeket az alkalmazásokat a sok honeypot között.

A tesztrendszer elsődleges tervek szerint kizárólag az egyetemen elérhető formában került telepítésre és így is kezdődött meg a tesztelés. Azonban a hibrid oktatás miatti távollét és több lelkes érdeklődő kérésének eleget téve egy későbbi szakaszban felhő alapú szerverre is felmásoltuk az egyetemen működő környezetet. Ebben, a már párhuzamos működésben egyrészt több hallgatót be tudtunk vonni, mivel otthonról is elérték. Másrészt automata szkennerekről is sikerült információt gyűjtenünk, amely jó alapot szolgálhat egy jövőbeli kutatáshoz.

A támadások megfigyelésére a honeypotok naplói mellett, felhasználtuk a *Tpot*-ban

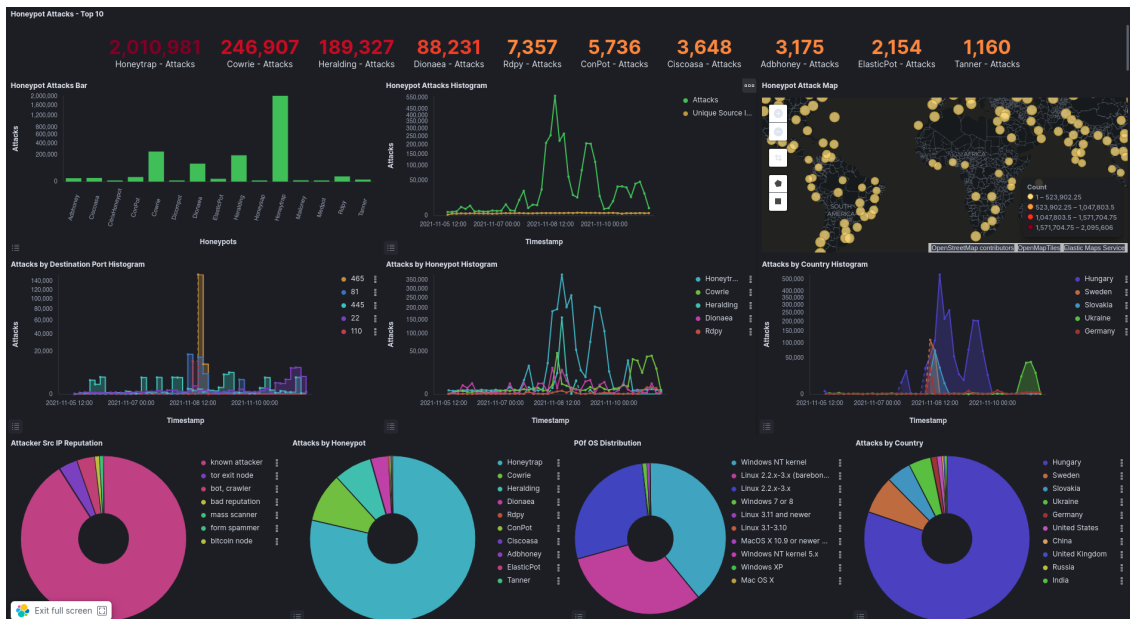
elérhető *Suricata* IDS rendszert is. Ezt azért éreztük szükségesnek, mivel így nem csak az egyes honeypotok konténereibe érkezett kéréseket tudtuk tanulmányozni, hanem az összeset, ami a rendszerre érkezett, továbbá így egyszerűen elemezhetünk a kihívásokat, amelyeknél problémát okozott a naplódatok továbbítása.[35]

A verseny menedzselését a CTFd nevezetű nyílt forráskódú capture the flag keretrendszer, amely egy webes felületet biztosít egy ilyen esemény szervezésére. Ezen a felületen regisztráltak a hallgatók és figyeltük előrehaladásukat a verseny során az egyes zászlók validálásával. A teszt két hétig tartott 20 résztvevővel. Kizárólag a honeypotokra több millió kérés érkezett, amely megfelelő nagyságú adathalmazt biztosított az elemzéshez.



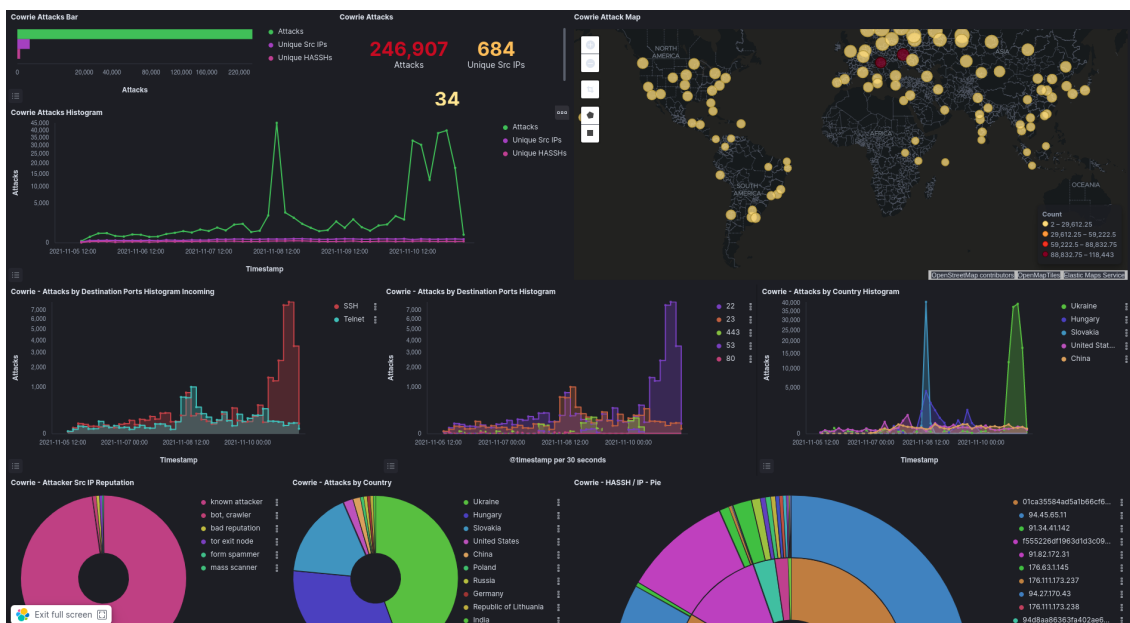
20. ábra. A hallgatók előrehaladása a CTFd felületén szemléltetve

A beérkezett adatok vizsgálatát a Kibana felületének irányítópultjai, az egyes résztvevőkre vonatkozó lekérdezések, valamint az előbbiekben említett CTFd platform összevetésével végeztük el. A Tpot összeítő dashboardja segítségével átfogó képet kaphattunk arról, hogyan alakul a kérések megoszlása az összes vagy akár egyetlen forrás címre szűrve időegységekre lebontva. Összegyűjti, hogy melyek a leggyakrabban megcímzett honeypot szolgáltatások, IP címek geolokációja alapján a világ mely pontjáról lettek detektálva támadások, valamint milyen autentikációs paraméterekkel történt próbálkozás.



21. ábra. Összesítő irányítópult

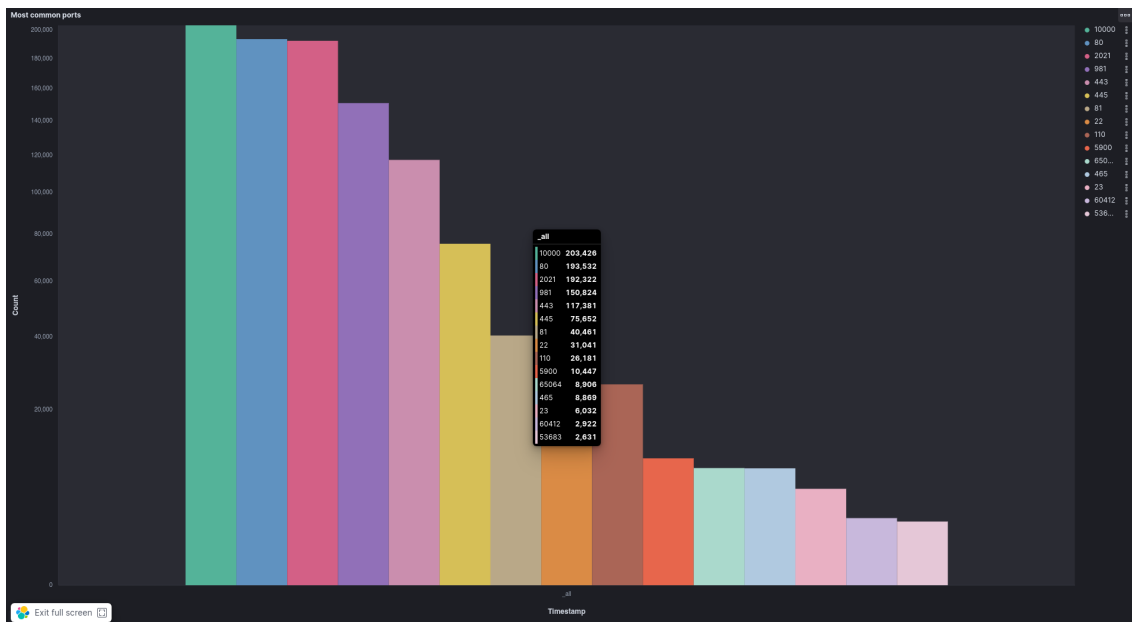
Az egyes honeypotokra vonatkozó felület alapján analizálhattuk, hogy a szolgáltatásoktól függően milyen támadási módszerek kerültek alkalmazásra. Az egyes kérésekre vonatkozó időbeli eloszlás megmutatta, hogy mennyi időt töltöttek a hallgatók a szolgáltatások felderítésével, mennyi időre volt szükségük az egyes kihívások megtalálásához. Ennek segítségével tudtunk megbecsülni mennyi idő nyerhető a védekező oldal számára.



22. ábra. Egyetlen honeypot, a Cowrie irányítópultja

Az alapértelmezetten elérhető vizualizációs panelek mellett egyes esetekben általunk létrehozottakat is használtunk. Többnyire akkor volt ilyenre szükség, ha nem kizárólag honeypotra vonatkozó, hanem a verseny egészét érintő adatot szerettünk volna

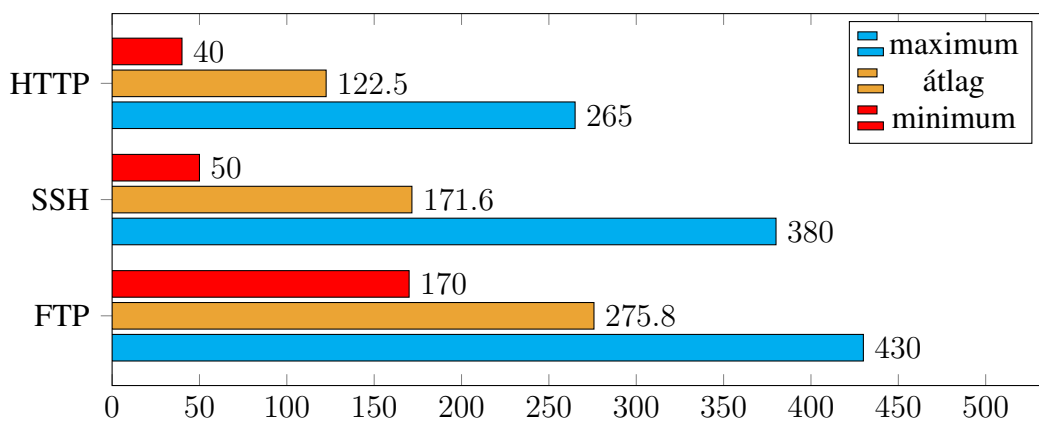
szemléltetni. Ilyen például, hogy az alábbi 23. ábra, amely tartalmazza a kihívások, valamint azon portok listáját, amelyen nem működött szolgáltatás, azonban érkezett rá kérés.



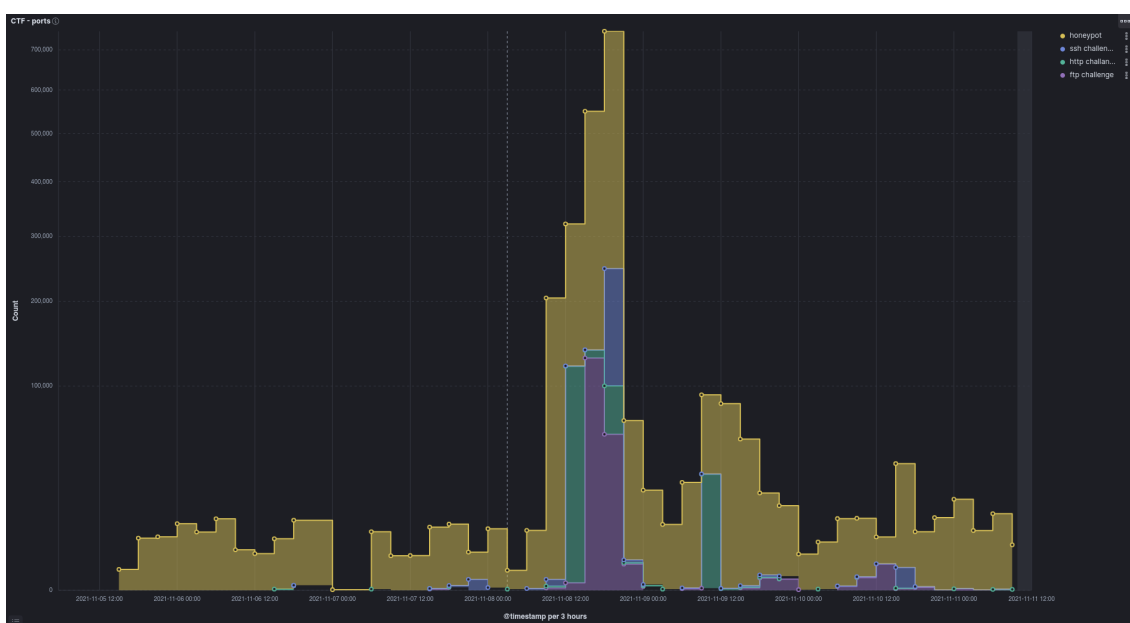
23. ábra. A 15 leggyakrabban megcímzett port

6.3. Eredmények értékelése

Az eredmények értékelése során az egyik legfontosabb szempont, hogy mennyire hatékonyan sikerült elterelni a támadók figyelmét a lényeges szolgáltatásokról, ez esetben a zászlókról. Ezt a különböző szolgáltatásokkal eltöltött idő mérésével tudjuk legpontosabban meghatározni. Ez alapján a honeypotok kifejezetten hatékonyak mondhatók, átlagosan a behatoló támadással eltöltött idejének csupán tíz százalékában foglalkozott a kihívásokat futtató szolgáltatásokkal, ez egyben azt is jelenti, hogy a támadás teljes ideje közel tízszeresére növekedett. Ez már önmagában elkönnyelhető sikerként, hiszen egy éles helyzetben a honeypot a támadás észlelésén túl értékes percek, órákat nyerne a SOC üzemeltetőinek. A nyerhető perceket szemlélteti a 24. ábra, amely szemlélteti, hogy hány perc kellett a versenyzőknek egy zászló megtalálásához. A honeypotok további célja, a támadás módjának megismerése. Erről szintén rengeteg adatot tudtunk kinyerni. A legnépszerűbb támadás/felderítési technika a portok letapogatása volt, de ez nem meglepő, hiszen ez a legerjedtebb módja a felderítésnek. További gyakori támadás volt a bruteforce, amellyel rengetegen kiderítették a szándékosan gyenge felhasználónév - jelszó párosokat. Már kevésbé gyakran, de többször észleltünk hallgatók által használt automatizált scripteket, amely pár másodperc alatt végigpróbált több száz, az adott szolgáltatásra jellemző sérülékenységet. További érdekes megfigyelés volt az UDP protokoll mellőzése, a forgalom csupán 0.2 százalékát tette ki. Ez egyben azt is jelenti, hogy az UDP protokollt használó szolgáltatásokat a támadók többsége fel sem fedezte. Ez egyrészt érthető, hiszen a legismertebb szolgáltatások TCP-n keresztül üzemelnek, másrészt furcsa, hogy a legtöbb támadó még csak egy udp portletapogatást sem indított, emiatt több szolgáltatásról nem is tudott.



24. ábra. Zászló megtalálásáig eltelt idő (percben)



25. ábra. Kérések időbeli eloszlása honeypot szolgáltatások és kihívások között

6.4. Automata szkennerek

Az internetet fűrkésző automatikus szkennereknek hála a világ számos pontjáról detektáltunk felderítési és/vagy támadási mechanizmust. Ezekről tapasztalataink alapján általánosságban elmondható, hogy egy támadás során meglehetősen hasznosak lehetnek a egyes rendszerek feltérképezésére, illetve a kezdeti információgyűjtésre. Többnyire egy jól megírt szkript formájában működhetnek, amelyek specifikus szolgáltatásokat keresnek, illetve meghatározott támadási formákat vetnek be ezek feltörésére, azonban nem fedik le egy tényleges támadó képességeit. Egy megfelelően konfigurált hálózatra nem jelentenek veszélyt, azonban hiányosságok esetén könnyen megtalálhatják annak gyengepontjait. Ezen adatok részletesebb értékelése a jövőben megvalósítandó terveink között szerepel.



26. ábra. Világtérkép a honeypotra érkező támadásokról

6.5. Optimalizációs lehetőségek

A honeypotok optimalizációjához szükséges, hogy számszerűsíthető mérőszámokkal tudjuk értékelni a honeypot hatékonyságát. Ehhez hat fő pontot tartottunk relevánsnak, a komplexitást, a hatékonyságot, a megtévesztési képességet, a minőséget, a támadó profilozási képességet és a biztonságot. A honeypotok teljesítményének a mérését két részre lehet osztani, a teljes hálózat hatékonyságára, valamint az egyedi honeypotok eredményességére. A teljes hálózat eredményességének a mérése viszonylag kézenfekvő, a támadók honeypotokkal eltöltött idejét a támadással eltöltött teljes idő arányában tekintve, vagy a honeypotokra érkező requesteket az összes requesthez képest jó képet kaphatunk arról, hogy mennyire értük el a célunkat, vagyis mennyire tereltük el a támadók figyelmét a valós rendszereinkről. Habár a honeypotok egyéni teljesítményének mérése sokkal nehezebb feladat, hiszen ezek jellemzően interakciós szintben, komplexitásban, nyújtott szolgáltatásokban, stb. nagyon eltérőek, megpróbáltunk néhány univerzális szempontot létrehozni ami alapján becsülhető a teljesítményük.

Egy honeypot komplexitásának a mérését az általa futtatott szolgáltatások, illetve a futtató környezet alapján lehet besorolni. Minél összetettebb, több szolgáltatást valóshoz hasonlóan szimulálni képes, annál jobban képes megtévesztetni a támadókat. A komplexitást közvetetten a honeypoton eltöltött idővel is lehet mérni, a komplexebb rendszerekkel várhatóan több időt tölt a támadó mint az egyszerűbbekkel. Egy nem kellően komplex honeypoton a támadó elhanyagolható időt tölt, ezesetben szükséges lehet a rendszer továbbfejlesztése, optimalizálása komplexitás terén.

A megtévesztési képesség azt mutatja, mennyire képes a honeypot a támadó megtévesztésére, az mennyi időt pazarol a honeypoton amíg rájön, hogy az nem egy valós szolgáltatás. Ez jelentősen összefügg a komplexitással, egy összetettebb honeypot várhatóan tovább leköti a támadó figyelmét, de fontos a támadó képességeit is figyelembe venni. Egy gyengébb képességű támadó várhatóan több időt tölt a honeypoton, tapasztalat hiányában valószínűleg csak később veszi észre hogy hamis

rendszert akar feltörni.

A támadó profilozási képességet mérésének célja az, hogy mennyire tudjuk a támadó tevékenységét részletesen nyomonkövetni. Ehhez részletes naplózásra van szükségünk a honeypotokon, amelyek segítségével minél részletesebben meg tudjuk határozni a támadásuk jellemzőit. Ezt legegyszerűbben a "Detection Maturity Level" [43] szintjei szerint lehet mérni, minél részletesebb adatgyűjtést valósít meg a honeypot, annál több szintet tudunk megállapítani. Amíg azonban egy önálló honeypot remekül képes fényt deríteni az alacsonyabb szintű információkra, a magasabb szinteken már a hálózat egészét átfogó eszközökre van szükségünk. Ilyen lehet például egy honeynetwork-ön üzemelő Suricata[35], amellyel a forgalmat hatékonyan tudjuk elemezni.

A production célú honeypotok egyik legfontosabb aspektusa a biztonság, egy nem kellő figyelemmel létrehozott honeypot nagyobb kárt tud okozni mint amennyit használ. Fontos annak a megakadályozása, hogy a támadó jump hostként használja a honeypotot további támadásokhoz. Ez a támadás felderítésének a nehezítése mellett lehetővé teheti, hogy a támadó olyan eszközöket érjen el amikhez egyébként nem tudna kapcsolódni. Ez megfelelő konfigurációval és kellő elővigyázatossággal megelőzhető.

Érdemes konfigurálni, hogy a honeypot csak korlátozottan férjen hozzá az erőforrásokhoz, ennek hiányában a támadó az erőforrások kimerítésére irányuló támadással könnyen megbéníthatja az egész eszközt.

Fontos továbbá, hogy a honeypot egy szeparált környezetben fusson. Ennek hiányában előfordulhat, hogy egy hibás konfiguráció vagy biztonsági rés esetén a támadó átveheti a host rendszer feletti irányítást, ez az első pontban részletezett veszélyekkel járhat.

Amennyiben a támadónak szándékosan egy valós operációs rendszerhez adunk hozzáférést és nem csak egy OS-t szimuláló környezethez, fontos a körültekintő jogosultságbeállítás. Hibás korlátozások hiányában a támadó olyan erőforrásokhoz juthat, amellyel teljes mértékben átveheti a rendszer fölötti irányítást.

Végül fontos a honeypot komplexitásának a figyelembe vétele, ennek növekedésével folyamatosan nő a számunkra ismeretlen, de kiaknázzható biztonsági rések kockázata, melyeket kihasználva a támadó további rendszerekhez férhet hozzá.

A fent említett szempontokat megvizsgálva behatárolható, hogy mennyire biztonságos egy honeypot. A megvalósítás előtt ezeket az eszközöket mindenképp érdemes megvizsgálni a fent részletezettek szerint.

Sajnos ezeket a szempontokat nagyon nehéz számszerűsíteni, hiszen a támadó tudása és képessége is szubjektív, nagyban függ az aktuális rendszerektől a rendelkezésre álló eszközöktől, stb. Ez egy nem elhanyagolható jövőbeli kutatási terület, amellyel nagyban pontosíthatjuk a honeypotok teljesítményének a mérését és könnyebben meghatározhatjuk az optimalizálandó területeket.

7. Összegzés

A honeypotok minden hátrányukkal együtt, nagyon hasznos eszközök, mind kutatási, mind biztonsági szempontból. Úgy gondoljuk, hogy az ilyen, részben a támadók emberi tulajdonságait kihasználó passzív eszközöknek egy jól konfigurált logkezelő és riasztási rendszerrel támogatva létjogosultsága van a mai IT rendszerekben. Természetesen gondos tervezés és kutatómunka szükséges az igényeinknek megfelelő honeypot megtalálásához, illetve a kockázatok minimalizálásához, de ez megéri hiszen, minimális karbantartást igényelnek. Az alacsony karbantartási igények mellett, lassan válnak elavulttá, ezt tökéletesen mutatja például a tény, hogy az SMTP protokoll 1982 óta létezik és majdnem 40 év után még mindig nagyon sok helyen támogatott. További fejlesztések szempontjából hatalmas potenciállal rendelkezik, a projekt, kialakítható riasztási rendszer, emailen vagy egyéb kommunikációs platformon keresztül, esetleg megoldható, hogy egy észlelt támadás során automatikus ellenintézkedéseket tegyen, ez lehet egy kapcsolódó belső eszköz automatikus leválasztása a hálózatról, vagy hálózaton kívüli esetén fekete listához adása a tűzfalon. Adminisztratív irányban érdekes lehet egy riasztáskor életbe lépő eljárásrend kialakítása, amely alapján éles helyzetben a biztonsági szakemberek gyorsan ellenlépéseket tudnak tenni.

A kitűzött célunkat elértük, egy általános, és egy kutatási célú honeypot implementálását, konfigurálását, bekötését és tesztelését sikeresen befejeztük. A kutatásunk során, kifejezetten a versennyel való tesztelés alatt rengeteg tanulságot szűrtünk le. Sikerült bizonyítanunk, hogy a honeypotok valóban hatékony eszközök a támadások lassítására, illetve a támadások során történő adatgyűjtésre, ezáltal lehetőséget biztosítva az időben történő reagálásra és a keletkező adatok elemzésére. Természetesen mindig van lehetőség fejlődésre, a honeypotok osztályozásával és éles szolgáltatásokhoz való összeválogatásával még hatékonyabban tudnánk védekezni a támadások ellen. További céljaink között szerepel a versennyel való tesztelés kiterjesztése, illetve a kihívások számának és sokszínűségének növelése. Úgy gondoljuk, a verseny kiterjesztésével és több versenyző megszólításával és az általuk használt módszerek részletes elemzésével a jövőben további sok érdekes, számunkra eddig ismeretlen tényre deríthetünk fényt.

Hivatkozások

- [1] Cisco. *Cisco Cybersecurity Report: Special Edition SMB*. 2018. URL: https://www.cisco.com/c/dam/global/hr_hr/solutions/small-business/pdf/small-mighty-threat.pdf (elérés dátuma 2021. 04. 21.).
- [2] Mandiant Solutions. *SPECIAL REPORT / M-TRENDS EMEA EDITION 2016*. Techn. jel. 2016.
- [3] Eva Chovancová és Norbert Ádám. „A Clustered Hybrid Honeypot Architecture”. *Acta Polytechnica Hungarica* 16.10 (2019).
- [4] Bharti Nagpal és tsai. „Catch: Comparison and analysis of tools covering honeypots”. *2015 International Conference on Advances in Computer Engineering and Applications*. IEEE. 2015, 783–786. old.
- [5] Howard F Lipson. *Tracking and tracing cyber-attacks: Technical challenges and global policy issues*. Techn. jel. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2002.
- [6] Lukáš Zobal, Dušan Kolář és Radek Fujdiak. „Current State of Honeypots and Deception Strategies in Cybersecurity”. *2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE. 2019, 1–9. old.
- [7] Moukafih Nabil és tsai. „SIEM selection criteria for an efficient contextual security”. *2017 International Symposium on Networks, Computers and Communications (ISNCC)*. IEEE. 2017, 1–6. old.
- [8] Rasmi-Vlad Mahmoud és Jens Myrup Pedersen. „Deploying a University Honeypot: A case study”. *CEUR Workshop Proceedings*. 2443. köt. CEUR Workshop Proceedings. 2019, 27–38. old.
- [9] parallax. *awesome-honeypots*. 2021. URL: <https://github.com/parallax/awesome-honeypots> (elérés dátuma 2021. 04. 22.).
- [10] cymmetria. *Ciscoasa honeypot*. 2021. URL: https://github.com/cymmetria/ciscoasa_honeypot (elérés dátuma 2021. 04. 22.).
- [11] MalwareTech. *CitrixHoneypot*. 2021. URL: <https://github.com/MalwareTech/CitrixHoneypot> (elérés dátuma 2021. 04. 22.).
- [12] mushorg. *Conpot honeypot*. 2021. URL: <https://github.com/mushorg/conpot> (elérés dátuma 2021. 04. 22.).
- [13] Michel Oosterhof Revision. *Cowrie SSH/Telnet Honeypot*. 2021. URL: <https://github.com/cowrie/cowrie> (elérés dátuma 2021. 04. 18.).
- [14] nsmfoo. *Dicompot honeypot*. 2021. URL: <https://github.com/nsmfoo/dicompot> (elérés dátuma 2021. 04. 22.).
- [15] DinoTools. *dionaea honeypot*. 2021. URL: <https://github.com/DinoTools/dionaea> (elérés dátuma 2021. 04. 22.).

- [16] jordan-wright. *elastic_honey_honeypot*. 2021. URL: https://github.com/jordan-wright/elastic_honey_honeypot (elérés dátuma 2021. 04. 22.).
- [17] terry-pan-dev. *ElasticPot_honeypot*. 2021. URL: <https://github.com/terry-pan-dev/ElasticPot> (elérés dátuma 2021. 04. 22.).
- [18] mushorg. *glutton_honeypot*. 2021. URL: <https://github.com/mushorg/glutton> (elérés dátuma 2021. 04. 22.).
- [19] johnnykv. *heralding_honeypot*. 2021. URL: <https://github.com/johnnykv/heralding> (elérés dátuma 2021. 04. 22.).
- [20] provos. *honeyd_honeypot*. 2021. URL: <https://github.com/provos/honeyd/tree/master/honeyd> (elérés dátuma 2021. 04. 22.).
- [21] foospidy. *HoneyPy_honeypot*. 2021. URL: <https://github.com/foospidy/HoneyPy> (elérés dátuma 2021. 04. 22.).
- [22] SecureAuthCorp. *HoneySAP_honeypot*. 2021. URL: <https://github.com/SecureAuthCorp/HoneySAP> (elérés dátuma 2021. 04. 22.).
- [23] *HoneyTrap Documentation*. 2021. URL: <https://docs.honeytrap.io/> (elérés dátuma 2021. 04. 18.).
- [24] bontchev. *ipphoney_honeypot*. 2021. URL: <https://github.com/bontchev/ipphoney> (elérés dátuma 2021. 04. 22.).
- [25] awhitehatter. *mailoney_honeypot*. 2021. URL: <https://github.com/awhitehatter/mailoney> (elérés dátuma 2021. 04. 22.).
- [26] citronneur. *rdpy_honeypot*. 2021. URL: <https://github.com/citronneur/rdpy> (elérés dátuma 2021. 04. 22.).
- [27] mushorg. *snare_honeypot*. 2021. URL: <https://github.com/mushorg/snare> (elérés dátuma 2021. 04. 22.).
- [28] ciscocsirt. *dhp_honeypot*. 2021. URL: <https://github.com/ciscocsirt/dhp> (elérés dátuma 2021. 04. 22.).
- [29] thinkst. *opencanary_honeypot*. 2021. URL: <https://github.com/thinkst/opencanary> (elérés dátuma 2021. 04. 22.).
- [30] Cowrie. *Cowrie_honeypot_hivatalos_oldala*. 2021. URL: <https://cowrie.readthedocs.io/en/latest/index.html> (elérés dátuma 2021. 04. 22.).
- [31] honeytrap. *honeytrap_honeypot*. 2021. URL: <https://docs.honeytrap.io/> (elérés dátuma 2021. 04. 22.).
- [32] DinoTools. *dionaea_honeypot_hivatalos_oldala*. 2021. URL: <https://dionaea.readthedocs.io/en/latest/> (elérés dátuma 2021. 04. 22.).

- [33] foospidy. *honeypy honeypot hivatalos oldala*. 2021. URL: <https://honeypy.readthedocs.io/en/latest/> (elérés dátuma 2021. 04. 22.).
- [34] telekom-security. *T-Pot honeypot*. 2021. URL: <https://github.com/telekom-security/tpotce> (elérés dátuma 2021. 04. 22.).
- [35] OISF. *Suricata*. 2021. URL: <https://suricata.io/> (elérés dátuma 2021. 11. 02.).
- [36] Elasticsearch B.V. *Elastic Stack*. 2021. URL: <https://www.elastic.co/> (elérés dátuma 2021. 04. 18.).
- [37] Docker. *honeypy honeypot hivatalos oldala*. 2021. URL: <https://docs.docker.com/engine/install/> (elérés dátuma 2021. 04. 22.).
- [38] eckes. *net-tools weboldala*. 2021. URL: <https://net-tools.sourceforge.io/> (elérés dátuma 2021. 04. 22.).
- [39] community. *firewalld weboldala*. 2021. URL: <https://firewalld.org/> (elérés dátuma 2021. 04. 22.).
- [40] community. *Golang logging library*. 2021. URL: <https://github.com/op/go-logging#example> (elérés dátuma 2021. 04. 22.).
- [41] StreamSets. *Grok Patterns*. 2021. URL: https://streamsets.com/documentation/datacollector/latest/help/datacollector/UserGuide/Apx-GrokPatterns/GrokPatterns_title.html (elérés dátuma 2021. 04. 18.).
- [42] Elasticsearch B.V. *Logstash configuration examples*. 2021. URL: <https://www.elastic.co/guide/en/logstash/current/config-examples.html> (elérés dátuma 2021. 04. 18.).
- [43] Ryan Stillions. „The DML model”. *Retrieved August 7 (2014)*, 2021. old.