VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

DETEKCE KYBERNETICKÝCH ÚTOKŮ V LOKÁLNÍCH SÍTÍCH
DETECTION OF CYBER ATTACKS IN LOCAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Libor Sasák
AUTHOR

VEDOUCÍ PRÁCE doc. Ing. Lukáš Malina, Ph.D.
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Libor Sasák ***ID:*** 203175
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Detekce kybernetických útoků v lokálních sítích

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s metodami a nástroji pro detekci a mitigaci kybernetických útoků v lokálních počítačových sítích. Popište existující i perspektivní metody detekce jako je heuristická analýza, sledování provozu, parsování komunikačních protokolů, sledování a vyhodnocení parametrů pomocí umělé inteligence atd. Dále se zaměřte na softwarové nástroje, jako jsou Snort, Suricata atp. V rámci práce vybrané nástroje otestujte na minipočítači a zhodnoťte jejich činnost a možnosti rozšíření o další moduly.
Cílem bakalářské práce bude návrh a realizace modulárního systému pro detekci kybernetických útoků v lokálních sítích, který využívá vybraný softwarový nástroj a vhodně jej rozšiřuje o perspektivní metody detekce. Dalším cílem je realizace řešení na jednodeskovém PC a testování funkčnosti řešení. Výstupem práce bude i vlastní aplikace s GUI, která vhodně zpracovává a zobrazuje jednotlivé logy a detekované útoky.

DOPORUČENÁ LITERATURA:

[1] PFLEEGER, Charles P.; PFLEEGER, Shari Lawrence. Analyzing computer security: a threat/vulnerability/countermeasure approach. Prentice Hall Professional, 2012.

[2] GARCIA-TEODORO, Pedro, et al. Anomaly-based network intrusion detection: Techniques, systems and challenges. computers & security, 2009, 28.1-2: 18-28.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: doc. Ing. Lukáš Malina, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT
Táto bakalárska práca sa zaoberá detekciou útokov v lokálnej sieti a využitím open source nástrojov na tento účel. V prvej kapitole je pojednávané o kybernetických útokoch ako takých vrátane príkladov konkrétnych útokov. Druhá kapitola sa všeobecne venuje systémom na detekciu prienikov a taktiež spomína a popisuje niektoré open source systémy. Tretia kapitola sa v skratke venuje všeobecnému deleniu metód detekcie útokov. Štvrtá kapitola uvádza a popisuje vybraný nástroj Suricata, ktorý je v piatej kapitole testovaný v detekcii rôznych útokov, počas čoho je sledované správanie a výstup tohto nástroja. V šiestej kapitol je uvedený nástroj ARPwatch a testovaný na detekciu útoku ARP spoofing. Siedma a ôsma kapitola sa zaoberajú návrhom a úspešnou realizáciou systému na detekciu útokov, ktorý zabezpečuje výstup vo forme logov vypovedajúci o škodlivom či podozrivom prenose na sieti. Deviata kapitola sa zaoberá návrhom a realizáciou aplikácie s grafickým užívateľským rozhraním, ktorá spomenuté logy prehľadne prezentuje a naviac umožňuje ďalšie operácie vrátanie základného ovládania nástrojov detekcie.

KĽÚČOVÉ SLOVÁ
ARPwatch, detekcia, IDS, kybernetický útok, lokálna sieť, open source, Raspberry Pi, spracovanie logov, Suricata

ABSTRACT
This bachelor thesis focuses on the detection of attacks in the local network and the use of open source tools for this purpose. The first chapter deals with cyber attacks and also describes some of them. The second chapter focuses primarily on intrusion detection systems in general and also mentions and describes some open source systems. The third chapter briefly deals with the general division of attack detection methods. The fourth chapter introduces and describes the selected tool Suricata, which is also tested in the fifth chapter in the detection of various attacks, during which the behaviour and output of this tool are tracked. In the sixth chapter, the ARPwatch tool is presented and tested for ARP spoofing attack detection. The seventh and eighth chapters deal with the design and successful implementation of an attack detection system that provides output in the form of logs indicating malicious or suspicious traffic on the network. The ninth chapter deals with the design and implementation of the application with a graphical user interface, which clearly presents the mentioned logs and also allows other operations, including the essential control of the detection tools.

KEYWORDS
ARPwatch, cyber attack, detection, IDS, local network, log processing, open source, Raspberry Pi, Suricata

SASÁK, Libor. *Detection of Cyber Attacks in Local Networks*. Brno, 2020, 68 s. Bakalárska práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: doc. Ing. Lukáš Malina, Ph.D.

Vysadené pomocou balíčku `thesis` verzie 4.00; `http://latex.feec.vutbr.cz`

VYHLÁSENIE

Vyhlasujem, že svoju bakalársku prácu na tému „Detection of Cyber Attacks in Local Networks“ som vypracoval samostatne pod vedením vedúceho bakalárskej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som týmto poďakoval vedúcemu bakalárskej práce pánovi doc. Ing. Lukášovi Malinovi, Ph.D. za odborné vedenie, konzultácie, trpezlivosť, podnetné návrhy a pripomienky k práci.

Obsah

Zoznam obrázkov

Úvod

Postupom času, nárastom počtu bezpečnostných incidentov a v neposlednom rade ich medializovaním, si spoločnosť postupne začala uvedomovať hrozby číhajúce či už na internete, v lokálnych sieťach alebo hrozby spojené s čímkoľvek čo je schopné digitálnej komunikácie, riadené nejakým operačným systémom či disponujúce úložiskom uchovávajúcim citlivé informácie.

V záujme človeka ja po prvé predchádzať týmto incidentom pomocou pasívneho zabezpečenia. Či už ide od zavedenie firewallu, riadenie prístupu do siete (NAC), využívanie virtuálnych privátnych sietí (VPN) či iné kroky na sťaženie prístupu k aktívam. Lenže samotné pasívne zabezpečenie často nestačí, história nám dosť jasne ukázala, že kde je prekážka, tam je aj spôsob. Čím teda chceme v rámci bezpečnosti disponovať, je aktívna detekcia útočníka a jeho činnosti.

Práca je zameraná na detekciu útokov v lokálnych sieťach. Túto detekciu umožňujú systémy na detekciu prienikov (IDS), ktorým je venovaný väčšinový podiel práce. Teoretická časť práce sa zaoberá všeobecným popisom týchto systémov, ich rozdelením podľa umiestnenia v sieti a metódami detekcie, ktoré spravidla využívajú.

Praktická časť práce sa zameriava jednak na testovanie nástroja Suricata IDS v kombinácii s jednodoskovým počítačom Raspberry Pi, neskôr je ale testovaný aj nástroj ARPwatch ako rozšírenie detekcie útokov k Suricate.

Ďalšie kapitoly zahŕňajú návrh a praktické overenie systému detekcie útokov pozostávajúce aj z týchto spomenutých nástrojov. Výstupom tohto systému sú logy obsahujúce záznamy o útokoch či podozrivých tokoch alebo dátach.

Posledná časť práce sa zaoberá návrhom a realizáciou aplikácie disponujúcej grafickým užívateľským rozhraním, ktorá spomenuté logy spracúva a vhodne prezentuje užívateľovi. Aplikácia naviac obsahuje užitočné ovládacie prvky pre spomínané nástroje.

1 Kybernetické útoky

Prvá časť práce sa zaoberá kybernetickými útokmi ako takými. Približuje nám a prekladá oficiálnu definíciu kybernetického útoku zo Slovníka Internetovej Bezpečnosti a predstavuje nám príklady najčastejších typov útokov v lokálnej sieti.

1.1 Definícia kybernetického útoku

Definícia kybernetického útoku podľa inštrukcie CNSS č. 4009
CNSS, 26. apríl 2010, preklad
„Útok prostredníctvom kybernetického priestoru, ktorý je zameraný na podnikové využívanie kybernetického priestoru za účelom narušenia, deaktivácie, zničenia alebo škodlivého riadenia výpočtového prostredia/infraštruktúry; alebo zničenia integrity údajov či ukradnutia kontrolovaných informácií.“

S predstavou o tom, čo je to kybernetický útok, je možné povedať, že útočníkom je osoba alebo proces, ktorý sa pokúša neoprávnene získať prístup k údajom, funkciám alebo iným obmedzeným oblastiam systému, spravidla s úmyslom uškodiť. V závislosti od okolností môžu byť počítačové útoky súčasťou kybernetického boja alebo počítačového terorizmu. Kybernetický útok môžu vykonávať suverénne štáty, organizácie, jednotlivci, skupiny či spoločnosť a taktiež môže pochádzať z anonymného zdroja.

Zoznam známych sieťových útokov:

- ARP spoofing
- Buffer overflow attack
- Cross-site scripting
- DHCP spoofing
- DHCP starvation
- DNS spoofing
- ICMP redirection
- IP fragmentation
- IP spoofing
- IRDP spoofing
- MAC flooding
- MAC spoofing
- Ping of Death
- Smurf attack
- STP manipulation
- SQL injection
- SYN flooding
- VLAN hopping

1.2 Popis vybraných útokov v lokálnej sieti

Z predchádzajúceho zoznamu útokov boli vybrané tie najznámejšie a sú bližšie popísané v rámci tejto podkapitoly.

ARP spoofing

ARP spoofing je technika, ktorou útočník odosiela spoofované ARP správy do lokálnej siete. Cieľom útočníka je priradiť MAC adresu útočníka k IP adrese iného počítača (napríklad predvolenej brány), čo spôsobí, že sa všetok prenos určený tejto IP adrese namiesto toho pošle útočníkovi. ARP spoofing môže umožniť útočníkovi zachytiť dátové rámce v sieti, upraviť prenos alebo blokovať všetku komunikáciu. Tento útok sa často využíva ako príležitosť pre ďalšie útoky, napríklad útoky typu DoS, MITM alebo útok únosu spojenia (session hijacking attack).

Port stealing

Port stealing je druh útoku, pri ktorom útočník „kradne“ prenos iného počítača v sieti. Útočník je pripojený k inému portu prepínača, ktorý mu bráni sledovať komunikáciu ostatných. Útočník však môže do prepínača poslať falošné rámce, ktoré ako zdrojovú adresu obsahujú MAC adresu obete, čím sa vydáva za danú obeť a zmätie prepináč. Prepínač zvyčajne prenáša rámec do posledného portu, ktorý sa javil ako „vlastník“ tejto MAC adresy. Ak teda útočník posiela dostatočné množstvo rámcov, môže prijať rámec, ktorý bol pri tom určený obeti.

MAC flooding

MAC Flooding je technika používaná na ohrozenie bezpečnosti sieťových prepínačov. Útok funguje tak, že „vytlačí“ legitímny obsah tabuľky MAC z prepínača tým, že mu zahltí pamäť, a teoreticky ho degraduje na hub, čo spôsobí broadcast dát všetkými portami prepínača. Z tohoto dôvodu môžu byť zasielané citlivé informácie do častí siete, kam pôvodne neboli smerované.

DHCP spoofing

DHCP spoofing, ktorý spravidla nasleduje po DHCP starvation útoku a zriadení rogue DHCP servera, je útok, pri ktorom útočník distribuuje falošné adresy IP a ďalšie TCP/IP konfiguračné nastavenia ostatným klientom v sieti. Tieto konfiguračné nastavenia zahŕňajú aj IP adresu predvolenej brány a servera DNS. Útočník teda môže nahradiť pôvodné legitímne IP adresy predvolenej brány a DNS servera svojimi vlastnými adresami. Tento útok je využiteľný na odchytávanie citlivých údajov, na spustenie útoku typu MITM alebo na phishing pomocou rogue DNS servera.

DNS spoofing

DNS spoofing (alebo DNS cache poisoning) je útok, pri ktorom sa do vyrovnávacej pamäte prekladača DNS vkladajú poškodené údaje o systéme názvov domén, ktoré spôsobujú, že DNS server v konkrétnom prípade bude klientom vracať nesprávny DNS záznam, napr. podvrhnutú IP adresu.

SYN flood

SYN flood je typ útoku DoS, ktorý využíva časť three-way handshake-u protokolu TCP na zahltenie zdrojov obete, či už ide o klienta alebo server, a na jeho zneprístupnenie v rámci siete. Útočník v podstate zasiela TCP žiadosti o pripojenie rýchlejšie, ako ich dokáže cieľové zariadenie spracovať, čo spôsobí jeho nedostupnosť.

2 Systém na detekciu prienikov (IDS)

Systém na detekciu prienikov (IDS) je nástroj alebo softvér spolupracujúci so sieťou, v ktorej je umiestnený, na jej zabezpečení, a hlásiaci prípadný pokus o prienik do systému.
IDS môže vykonávať rôzne funkcie:

- monitorovanie používateľov a činnosti systému,
- auditovanie zraniteľností a nesprávnych konfigurácií systému,
- hodnotenie integrity kritického systému a dátových súborov,
- rozpoznávanie známych vzorov útokov v aktivite systému,
- dentifikácia neobvyklej aktivity štatistickou analýzou,
- spravovanie kontrolných záznamov a upozorňovanie používateľov na porušenie pravidiel alebo normálnej činnosti,
- opravovanie chýb konfigurácie systému,
- inštalácia a prevádzka „pascí“ na zaznamenávanie informácií o narušiteľoch.

Systémy na detekciu prienikov bežne nevykonávajú všetky tieto funkcie naraz. Systémy odvodené od týchto funkcií si však spomenieme neskôr.

2.1 Typy systémov na detekciu prienikov založené na ich pozícii v sieti

Systémy na detekciu prienikov môžu mať v rámci siete rôzne umiestnenia, podľa čoho sú väčšinou aj diferencované softvérovo či konfiguračne. Tieto systémy sa spravidla delia na systémy umiestnené tak, aby monitorovali všetok prenos na sieti, ktorú chcem monitorovať (network-based IDS), na systémy, ktoré monitorujú prenos na rozhraní jedného zariadenia (host-based IDS) a na komplexné systémy, ktoré pozostávajú z viacerých detekčných (pod)systémov umiestnených vo viacerých bodoch v sieti (distributed IDS).

2.1.1 Network-based IDS

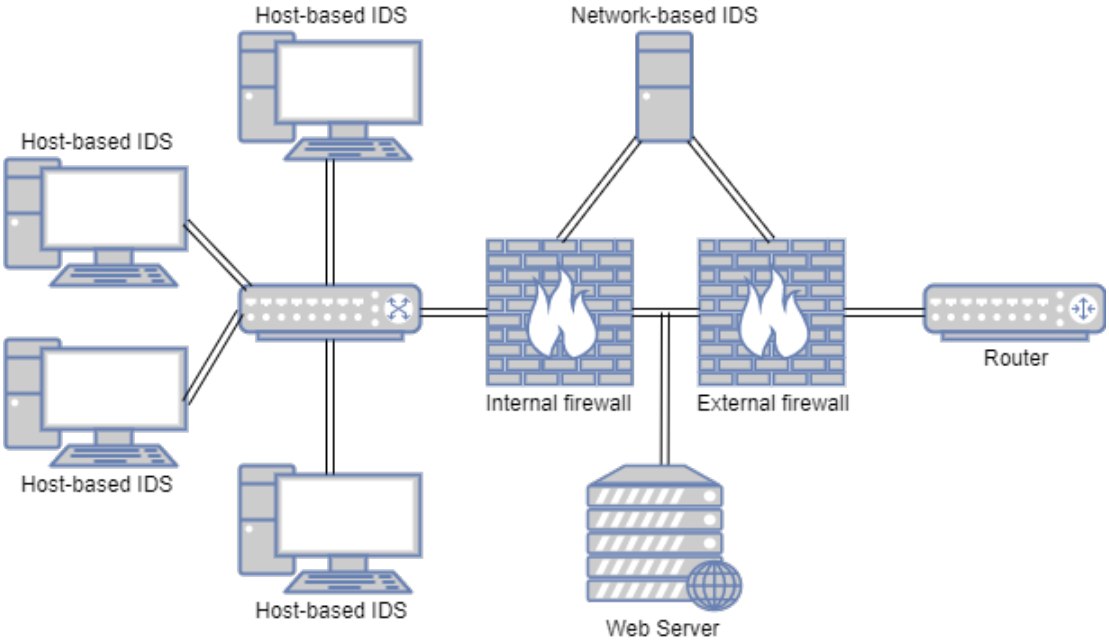
Network-based IDS býva strategicky umiestnený v bode alebo bodoch v rámci siete na monitorovanie prenosu do a zo všetkých zariadení v sieti. Po zistení útoku alebo po zistení neobvyklého správania pošle správcovi výstrahu. NIDS spravidla nekontroluje vždy všetok prichádzajúci a odchádzajúci prenos, nakoľko by v určitých prípadoch mohlo dôjsť k jeho zahlteniu. NIDS vieme rozdeliť na dva typy:

- on-line – monitoruje sieť v reálnom čase. Analyzuje ethernetové pakety a uplatňuje vopred definované pravidlá, aby rozhodol, či ide o útok alebo nie.

- off-line – zaoberá sa uloženými údajmi a necháva ich prejsť určitými detekčnými procesmi opäť za účelom rozhodovania o ich škodlivosti.

2.1.2 Host-based IDS

Host-based IDS sa spúšťa na jednotlivých klientoch alebo zariadeniach v sieti. HIDS monitoruje prichádzajúce a odchádzajúce pakety iba do a z jedného zariadenia a varuje používateľa alebo správcu, ak je zistená podozrivá aktivita. Takisto zvykne vytvárať snapshoty existujúcich systémových súborov a porovnávať ich s predchádzajúcimi snapshotmi. Ak boli kritické systémové súbory zmenené alebo odstránené, správcovi pošle výstrahu. HIDS sa zvykne používať na zariadeniach s kritickými funkciami, od ktorých sa neočakáva zmena ich konfigurácie. Príklad umiestnenia network-based a host-based IDS v sieti je možné vidieť na obr. 2.1.



Obr. 2.1: Príklad využitia host-based v kombinácii s network-based IDS v sieti.

2.1.3 Distributed IDS

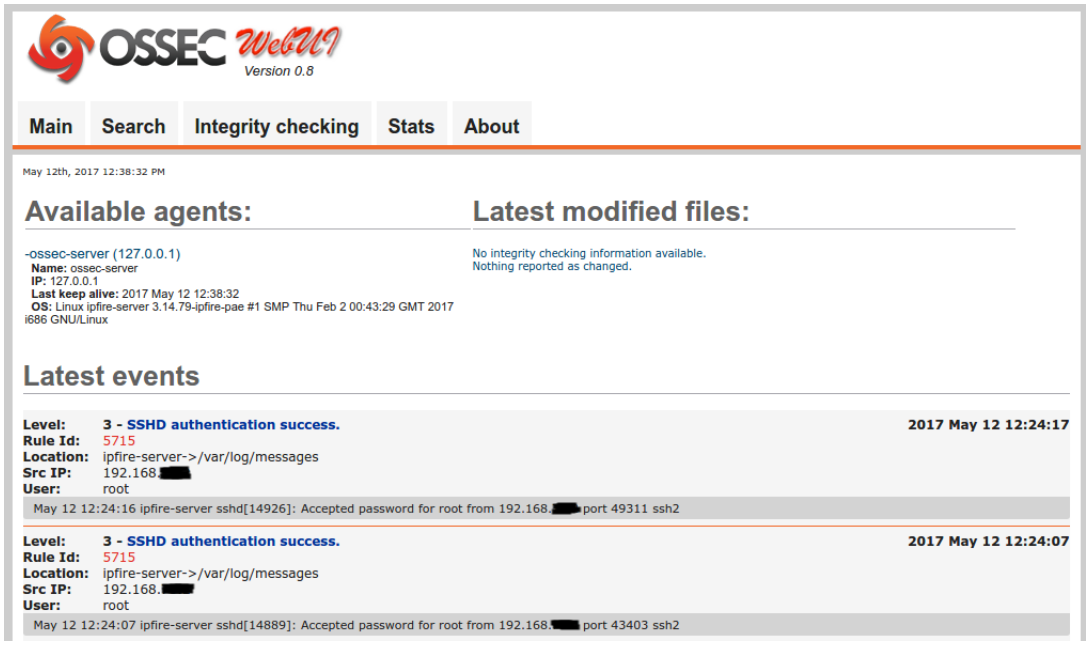
Distribuované IDS (DIDS) pozostáva z viacerých IDS v rozsiahlej sieti, pričom všetky komunikujú buď vzájomne alebo s centrálnym serverom, čo umožňuje pokročilé monitorovanie siete, analýzu incidentov a taktiež zabezpečuje presnejšie informácie o útokoch. Jednoduchšie povedané, DIDS umožňuje získať širší prehľad o tom, čo sa deje v sieti ako celku. Tento typ IDS je využívaný hlavne v sieťach väčších rozmerov a v tejto práci mu preto nebudem venovať viac pozornosti. Pre viac informácii o IDS, IPS a ich rozdelení viď [3], [4] a [5].

2.2 Bezplatné a open source systémy na detekciu prienikov

Na internete človek nájde veľké množstvo rôznych nástrojov schopných detegovať útoky na sieti, či útoky ohrozujúce samotné zariadenie. Dobre by teda bolo spomenúť tie z nich, ktoré sú bezplatné, overené a sithli si už získať lepšiu reputáciu v tomto odvetví softvéru.

OSSEC

OSSEC je HIDS vlastnený japonskou spoločnosťou Trend Micro. Na Unix-like systémoch sa predovšetkým zameriava na logy a konfiguračné súbory. Vytvára kontrolné súčty dôležitých súborov, pravidelne ich overuje a varuje používateľa, ak sa s nimi stane niečo neočakávané. Taktiež sleduje a zachytáva všetky „zvláštne“ pokusy o získanie root prístupu. V systéme Windows okrem iného dáva pozor aj na neoprávnené úpravy registra. Ukážka GUI systému na obr. 2.2.

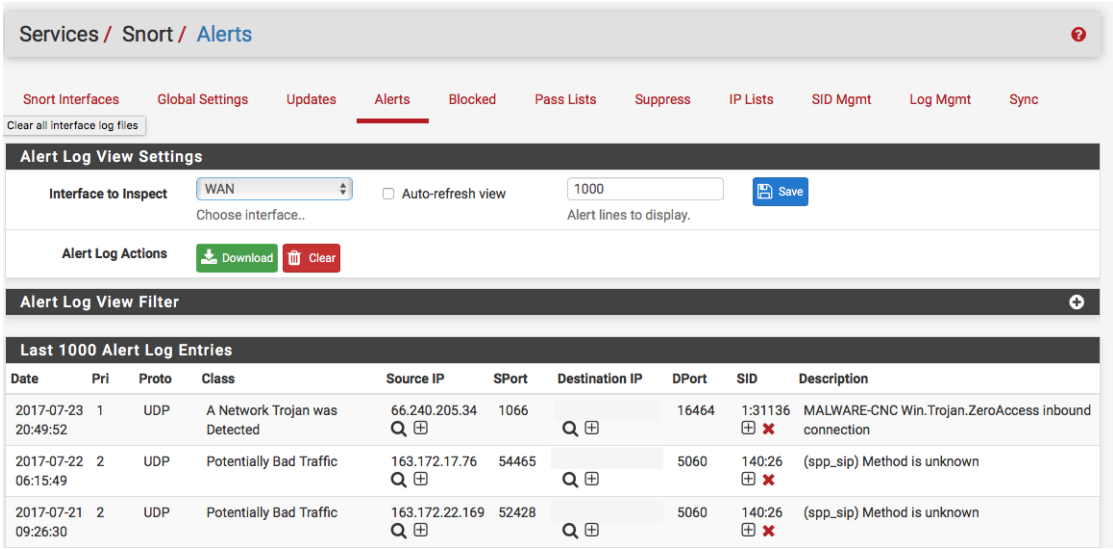


Obr. 2.2: OSSEC - GUI. Zdroj: [6].

Snort

Snort je NIDS udržiavaný spoločnosťou Cisco Systems. Patrí medzi najznámejšie IDS a je schopný bežať na operačných systémoch Windows, Linux a Unix a analyzovať prenos v reálnom čase. Snort disponuje troma módmi, módom paketového

sniffingu, paketového logovania a detekcie prieniku. Režim detekcie prieniku je založený na súbore pravidiel, ktoré si môžete sami vytvoriť alebo stiahnuť. Ukážka pfSense GUI na management Snortu na obr. 2.3.



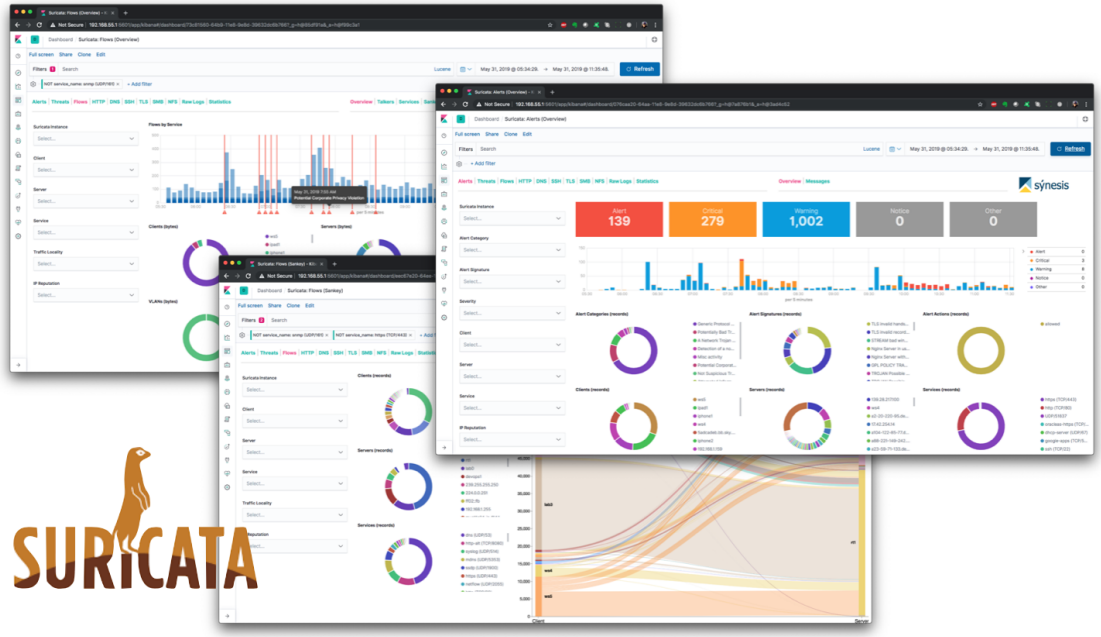
Obr. 2.3: Snort - pfSense GUI. Zdroj: [7].

Suricata

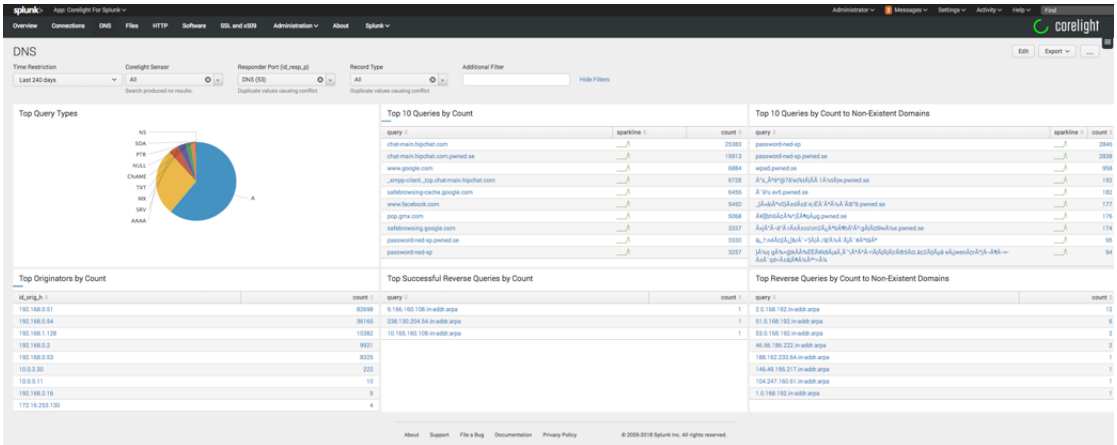
Suricata NIDS ako nadstavba a priamy konkurent Snortu poskytuje detekciu a prevenciu prienikov v reálnom čase a monitoruje zabezpečenie siete. Suricata je modernou alternatívou k Snortu s možnosťou behu na viacerých vláknach, akcelerácie na GPU a štatistickej detekcie anomálií viacerých modelov. Je kompatibilný s dátovou štruktúrou Snortu, vďaka čomu je v Suricate možné implementovať jeho politiku. Suricata narozdiel od Snortu naviac pracuje až po aplikačnú vrstvu. O tomto IDS si toho povieme viac za chvíľu, keďže bude hlavným predmetom praktickej časti časti práce. Ukážka Kibana dasboardu vizualizujúceho výstup Suricaty spracovaný enginom Elasticsearch na obr. 2.4.

Zeek

Zeek HIDS môže bežať na FreeBSD, Linuxe a Mac OS a vykonávať dve operácie, záznam prenosu a jeho analýza. Zeek sa, rovnako ako Suricata, líši od Snortu tým, že pracuje aj na aplikačnej vrstve, čo umožňuje sledovať množstvo ďalších služieb, ako napríklad HTTP, DNS, SNMP a FTP. Ukážka Splunk dashboardu zo spracovaného výstupu Zeeku na obr. 2.5.



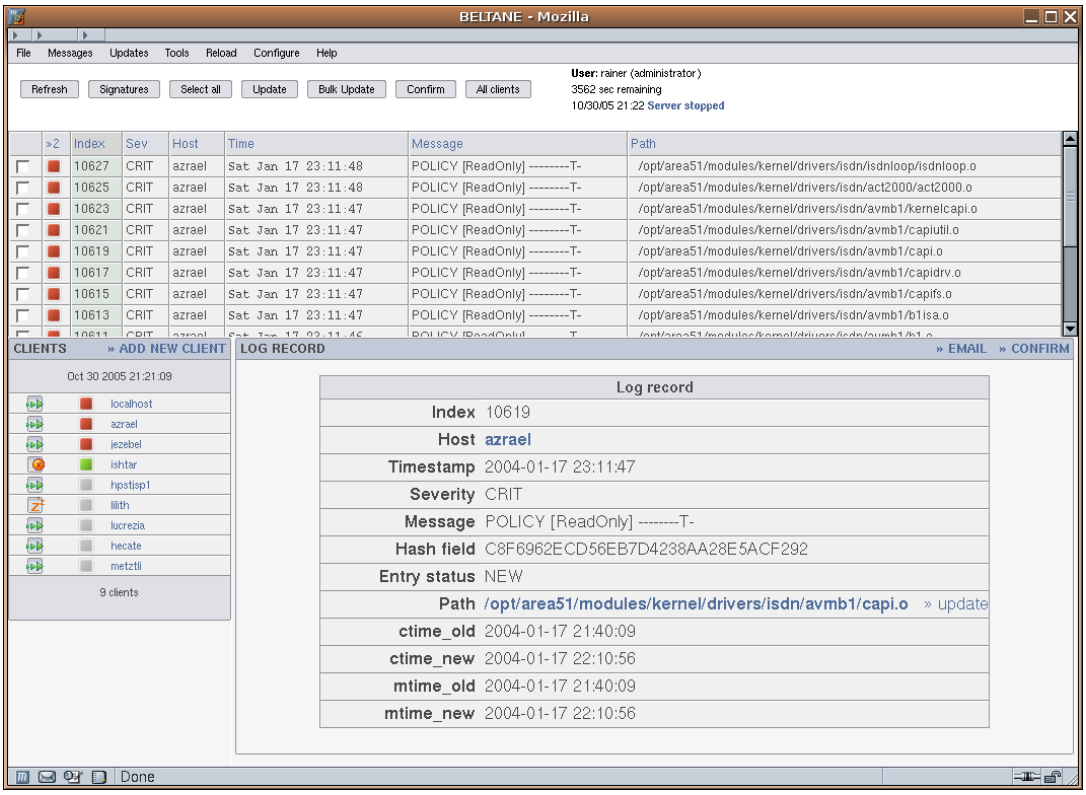
Obr. 2.4: Suricata - Kibana dashboard. Zdroj: [8].



Obr. 2.5: Zeek - Splunk dashboard. Zdroj: [9].

Samhain

Samhain HIDS poskytuje kontrolu integrity súborov a monitorovanie/analýzu logov. Samahin okrem toho monitoruje porty a deteguje rootkity, rogue spustiteľné súbory SUID (Set user ID) a skryté procesy. Tento nástroj bol navrhnutý na monitorovanie viacerých systémov s rôznymi operačnými systémami s centralizovaným logovaním a údržbou, dá sa však použiť aj ako samostatná aplikácia na jednom počítači. Samhain môže bežať na Linuxe a všetkých POSIX/UNIX systémoch. Ukážka Beltane GUI na management Samhainu na obr. 2.6.



Obr. 2.6: Samhain - Beltane GUI. Zdroj: [10].

Fail2ban

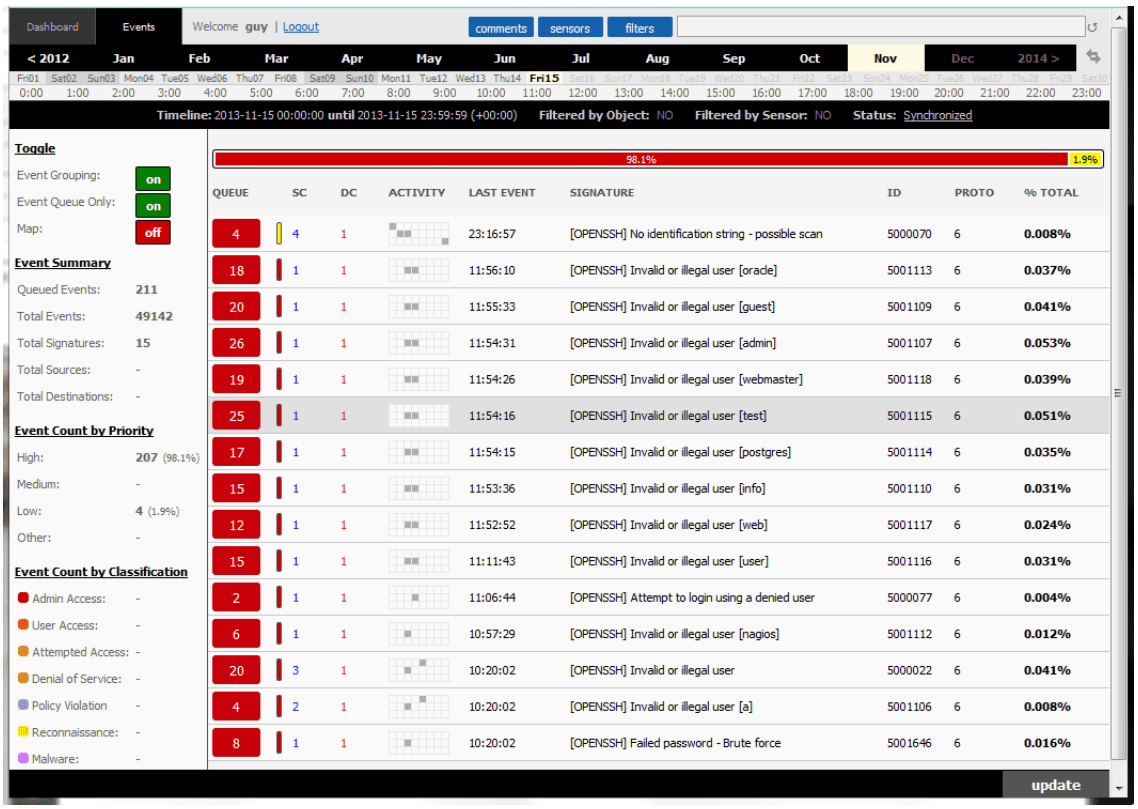
Fail2ban je HIDS, ktorá však disponuje aj pár prevenčnými funkciami. Dokáže monitorovať logy a tým detegovať podozrivé udalosti s následnou aktualizáciou pravidiel lokálneho firewallu (napr. IPtables alebo Netfilter) aby zablokoval IP adresu zdroja. Jedná sa o predvolenú funkciu, avšak je možná aj konfigurácia iných akcií po detegovaní hrozby. Naviac disponuje vstavanými filtrami pre služby ako Apache, Courrier, SSH, FTP, Postfix a iné. Ukážka z výstupu Fail2ban na obr. 2.7.

Sagan

Sagan je skôr systém na analýzu logov ako skutočné IDS, má však jeho určité funkcie. Sagan môže byť využitý na monitorovanie miestnych logov, ale taktiež môže spolupracovať aj s inými nástrojmi. Napríklad dokáže analyzovať logy Snortu a v podstate ho tým rozšíriť o funkcie HIDS. Rovnako je kompatibilný aj so Suricatou. Ukážka GUI systému na obr. 2.8.

```
cyril@earth:~
File  Edit  View  Terminal  Tabs  Help
02/18/05 23:03:57.936 INFO Fail2Ban v0.3.0-CVS is running
02/18/05 23:03:59.065 INFO Ban 62.94.10.80
02/18/05 23:08:19.221 INFO Restoring iptables...
02/18/05 23:08:19.224 INFO Unban 62.94.10.80
02/18/05 23:08:19.238 INFO Exiting...
02/18/05 23:12:16.017 INFO Fail2Ban v0.3.0-CVS is running
02/18/05 23:38:43.211 INFO Restoring iptables...
02/18/05 23:38:43.213 INFO Exiting...
02/18/05 23:45:11.090 INFO Fail2Ban v0.3.0-CVS is running
02/19/05 12:01:32.866 INFO Ban 66.139.75.25
02/19/05 12:11:33.871 INFO Unban 66.139.75.25
02/19/05 12:29:45.734 INFO Ban 66.139.75.25
02/19/05 12:39:46.656 INFO Unban 66.139.75.25
02/19/05 22:12:27.553 INFO Ban 69.10.152.217
02/19/05 22:22:28.475 INFO Unban 69.10.152.217
02/20/05 03:40:57.450 INFO Ban 219.147.179.93
02/20/05 03:50:58.408 INFO Unban 219.147.179.93
02/20/05 06:37:21.951 INFO Ban 211.43.24.222
02/20/05 06:47:22.872 INFO Unban 211.43.24.222
02/20/05 11:35:54.693 INFO Restoring iptables...
02/20/05 11:35:54.696 INFO Exiting...
02/20/05 12:12:17.195 WARNING Fail2Ban v0.3.0-CVS is running
earth root #
0 cyril  1 root  2 fail2ban                earth: 0.00 0.03 0.00 20/02 14:44
```

Obr. 2.7: Fail2ban - ukážka výstupu. Zdroj: [11].



Obr. 2.8: Sagan - GUI. Zdroj: [12].

3 Metódy detekcie kybernetických útokov

Táto kapitola sa zaoberá detekčnými metódami, ktoré systémy na detekciu prienikov využívajú, a ich rozdelením podľa toho, akým spôsobom a technikami hrozby detegujú.

3.1 Metóda sledovania príznakov

Najbežnejšou metódou detekcie útokov v lokálnych sieťach je metóda sledovania príznakov. Inými slovami sa jedná o vyhľadávanie príznakov/signatúr v sieťovom prenose. To znamená, že systémy na detekciu prienikov založené na signatúrach (ang. siganture-based IDS) fungujú takmer rovnako ako antivírusový softvér pri vyhľadávaní známej škodlivej aktivity či kódu.

Avšak zatiaľ čo signature-based IDS je veľmi efektívny pri detegovaní známych útokov, tak rovnako, ako antivírusový program, závisí od pravidelných aktualizácií databázy signatúr, aby zostal „informovaný“ o množstve rozličných techník útočníkov. Inými slovami, signature-based IDS je rovnako dobrý ako je dobrá jeho databáza signatúr.

Z dôvodu obmedzenia databázou máme okamžite ďalšie dva problémy na svete. Po prvé, je ľahké oklamať riešenia založené na sledovaní signatúr zmenou formy vykonania daného útoku. Týmto spôsobom útočník jednoducho obíde databázu signatúr uloženej v IDS a má ideálnu príležitosť na získanie prístupu k sieti.

Po druhé, čím komplexnejšia je databáza signatúr, tým vyššie sú nároky na CPU, nakoľko je systém zaťažený analýzou každej jednej signatúry. To nevyhnutne znamená, že pakety nad maximálnou šírkou pásma môžu byť zahodené. Z tohto dôvodu môže byť potrebné informačné kanály rozdeliť a po analýze znova skombinovať, čím sa zvýši zložitosť a pravdaže aj náklady. Okrem toho treba brať do úvahy fakt, že čím väčší je počet hľadaných signatúr, tým vyššia je pravdepodobnosť výskytu falošne pozitívnych nálezov.

Útočníci, ktorý sú si vedomí takéhoto IDS v sieti, sa budú snažiť svoj útok pred ním skrývať. To im značne uľahčuje zložitosť interakcií jednotlivých aplikácií a v neposlednom rade aj podpora Unicode. Unicode, ktorý okrem iného umožňuje zastúpenie každého znaku z každého jazyka, je štandardom rôznych programovacích jazykov. Útočník môže vo svojom útoku použiť znaky, ktoré danému IDS môžu chýbať, a kvôli tomu signatúra nebude nájdená.

Vzhľadom na tendenciu hackerov neustále testovať a experimentovať je len otázkou času, že niekto objaví cestu aj okolo najnáročnejších signature-based IDS.

3.2 Metóda detekcie anomálií

Na rozdiel od detekčných systémov založených na signatúrach anomaly-based IDS hľadá druhy neznámych útokov, ktoré sú pre signature-based IDS ťažko odhaliteľné. Klasifikácia prenosu ako normálny či abnormálny je založená skôr na heuristike a pravidlách, ako na vzoroch či signatúrach, pričom sa IDS snaží nájsť akýkoľvek typ zneužitia odkláňajúceho sa od normálnej prevádzky systému.

Anomaly-based IDS môže napríklad využívať strojové učenie na porovnávanie modelov dôveryhodného správania sa v sieti s novým správaním. Vo výsledku systém anomálie či podozrivé správanie označí. Často sa pritom však stáva, že vopred neznáme, ale pritom legitímne správanie je taktiež označené.

Detekcia anomálií na báze strojového učenia pozostáva z dvoch fáz, z fázy tréningu, kedy sa vytvára profil normálneho správania, a z fázy testovania, v ktorej sa porovnáva súčasný prenos s profilom vytvoreným vo fázy tréningu.

Iná technika zase pozostáva z definovania toho, ako má bežne využitie systému vyzerať, použitím prísneho matematického modelu a označením akejkoľvek odchýlky od neho ako útok. Táto metóda je známa ako prísna detekcia anomálií. Medzi ďalšie techniky využívané na detekciu anomálií patria metódy ťaženia dát, metódy založené na gramatike ale aj Artificial Immune Systems (umelé imunitné systémy).

3.3 Stavová analýza protokolov

Stavová analýza protokolov je proces porovnávania vopred určených profilov bežnej aktivity protokolu s pozorovanou udalosťou, pričom sú hľadané výchylky.

Stavová analýza protokolov využíva informácie o spojeniach medzi klientami a porovnáva ich so záznamami v tabuľke stavov. Tabuľka stavov uchováva záznam o spojení medzi počítačmi, ktorý obsahuje zdrojovú a cieľovú IP adresu a port a využívané protokoly. Z dôvodu uchovania každého stavu každého protokolu pri každom jednom spojení dokáže byť táto metóda skutočne náročná na zdroje zariadenia.

Stavová analýza protokolov môže okrem iného identifikovať neočakávané sekvencie príkazov, ako napríklad opakované zadávanie rovnakého príkazu alebo zadávanie príkazu bez toho, aby bol vopred zadaný príkaz, po ktorom mal nasledovať, alebo od ktorého je závislý.

Na rozdiel od detekcie anomálií, ktorá využíva profily špecifické pre hostiteľa alebo sieť, sa stavová analýza protokolu spolieha na univerzálne profily, ktoré určujú, ako by sa konkrétne protokoly mali a nemali využívať. Pre viac informácií o metódach detekcie útokov viď [13], [14], [15] a [17].

3.4 Detekcia shell kódu pomocou umelej inteligencie

Umelé neurónové siete (ANN), ako forma algoritmu strojového učenia inšpirovaná správaním biologických neurónov v centrálnej nervovej sústave, bola vo svojej histórii využitá v niekoľkých oblastiach kybernetickej bezpečnosti, vrátane analýzy chýb v dizajne softvéru, detekcie počítačových vírusov či viacerých typov sieťových útokov. Už v dnešnej dobe sú na trhu prístupné AI systémy, ktoré dokážu ako reakciu na útok vytvárať honeypoty či návnady pre útočníkov.

Úspech a záujem o AI v oblasti kybernetickej bezpečnosti naznačuje aj rýchlo rastúca hodnota trhu, ktorá ešte v roku 2016 predstavovala hodnotu 1 miliardy amerických dolárov, pričom už v roku 2019 to bolo 8,8 miliard a do roku 2026 je očakávaná hodnota až 38,2 miliardy dolárov.

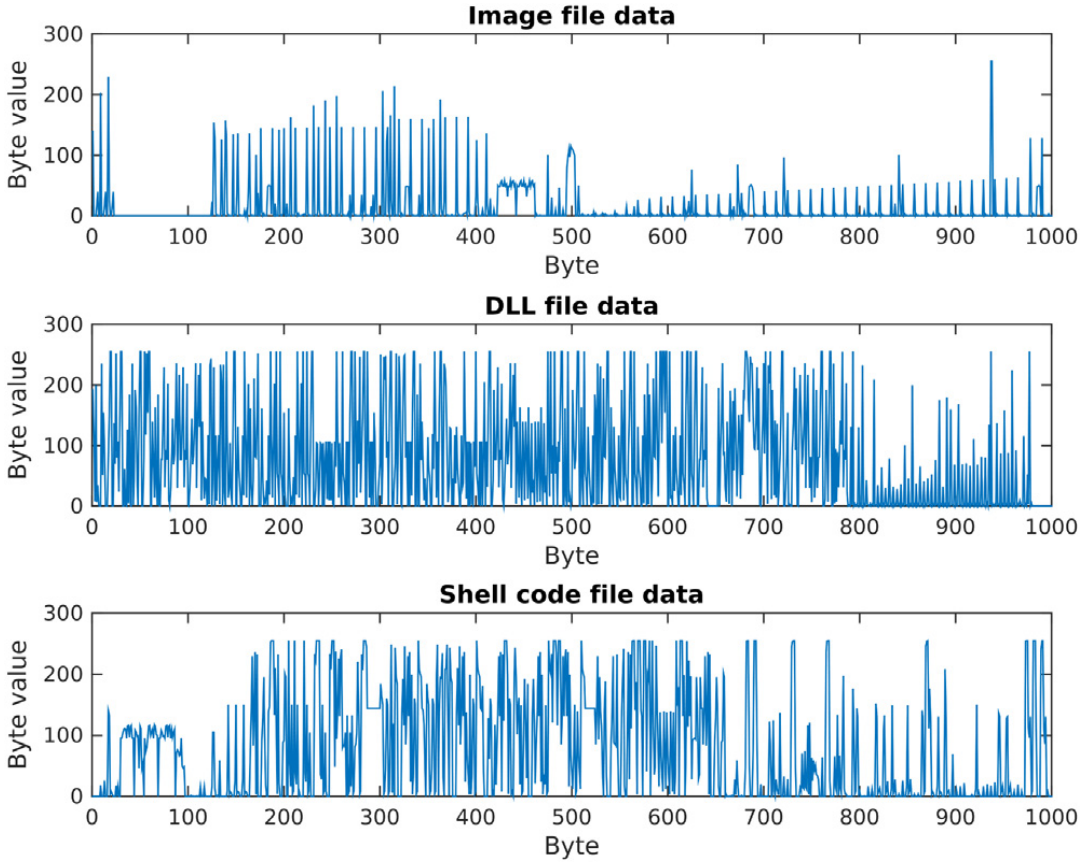
Ešte donedávna však ANN neboli považované za možné riešenie v problematike detekcie shell kódu, ktorý je popri väčšine útokov na získanie prístupu k systému hlavným nástrojom na spustenie škodlivého procesu či operácie. Detekcia shell kódu bola pre IDS vždy výzvou z dôvodu toho, že je shell kód obvykle písaný v strojovom kóde, má vzhľadom k rušnému prenosu na sieti miniatúrnu veľkosť a exploity, ktoré ho využívajú, majú často efektívne mätúcu povahu. Naviac je samotný shell kód často nerozpoznateľných od mnoho iných druhov sieťového prenosu.

Podnetom na vývoj tejto formy detekcie bol fakt, že bežne nástroje na detekciu ako je Snort či Suricata, v tejto oblasti produkujú veľké množstvo falošne pozitívnych nálezov, ktoré vyvolávajú spravidla binárne a obrazové súbory (napr. obrazové kópie systémov), či veľmi frekventovane aj DLL súbory zo služby Windows Update.

Riešenie na detekciu shell kódu, ktoré v dnešnej dobe vďaka univerzitám Sheffield Hallam University a De Montfort University zo Spojeného kráľovstva existuje, bolo schopné zo vzorku 400 tisíc náhodných súborov, ktoré sa na sieti objavili, správne označiť vzhľadom k tomu, či obsahujú shell kód, až 98,2% z nich. Súbory zároveň neboli počas učenia použité ani na trénovanie ani na krížovú validáciu.

Dáta na úrovni bajtov zaznamenaných počas prenosu boli prevedené do celočíselnej formy vhodnej ako vstup do ANN. Zaujímavý bol pokus o vizualizáciu rôznych sád tisícich po sebe idúcich bajtov z týchto dát, ktoré pre rôzne typy súborov ukazovali výrazne rôzne vzorce, aj napriek tomu, že sa vyskytovali značné rozdiely medzi súbormi rovnakého typu. Porovnanie vzorcov dát z obrazových súborov, DLL súborov a súborov obsahujúcich shell kód je možné vidieť na obr. 3.1. Pre viac informácií o tejto problematike viď [18].

Nástupom umelej inteligencie v oblasti kybernetickej bezpečnosti však rovnako, ako výrazne zlepšenie postupov, prichádzajú aj podnety na nové druhy útokov mierených priamo na AI aplikácie zodpovedné za detekciu. Tento fakt môže do budúcna predstavovať pre systémy vážne bezpečnostné hrozby (viď [19]).



Obr. 3.1: ANN - porovnanie vzorcov dát. Zdroj: [18].

4 Suricata IDS

Suricata je bezplatný open source nástroj na detekciu sieťových hrozieb schopný detekcie prienikov v reálnom čase (IDS), ako aj on-line prevencie prienikov (IPS), monitorovania sieťovej bezpečnosti (NSM) či off-line spracovania pcap súborov. Suricata kontroluje sieťový prenos pomocou celej škály rôznych pravidiel a signatúr a taktiež podporuje Lua skripty na detekciu komplexných hrozieb.

Podporuje dekódovanie paketov protokolov:

- IPv4, IPv6, TCP, UDP, SCTP, ICMPv4, ICMPv6, GRE
- Ethernet, PPP, PPPoE, Raw, SLL, VLAN, QINQ, MPLS, ERSPAN, VXLAN

Na aplikačnej vrstve podporuje dekódovanie:

- HTTP, SSL, TLS, SMB, DCERPC, SMTP, FTP, SSH, DNS, Modbus, ENIP/CIP, DNP3, NFS, NTP, DHCP, TFTP, KRB5, IKEv2, SIP, SNMP, RDP
- protokoly vyvinuté v jazyku Rust

4.1 Možnosti konfigurácie

Ako prvé je dobré spomenúť hlavný konfiguračný súbor *suricata.yaml* písaný vo formáte YAML (YAML Ain't Markup Language), vďaka ktorému je konfigurácia pre človeka ľahko čitateľná. Predvolene sa nachádza v priečinku */etc/suricata/* a jednotlivé časti sú v ňom zrozumiteľne okomentované. Suricata kvôli prehľadnosti podporuje rozdelenie tohoto súbor do viacerých konfiguračných súbor, ktoré je možné do hlavného súboru popridávať pomocou kľúčových slov *include* a *!include*.

V hlavnom konfiguračnom súbore sa nachádzajú nastavenia, ktoré umožňujú „povedať" niečo o sieti, ktorej prenos nástroj monitoruje, nastaviť zaznamenávanie štatistických údajov, výstupy monitorovania a činnosti Suricaty, parsery aplikačnej vrstvy a upravovať pokročilé nastavenia pre iné prvky nástroja.

Časťou súboru *suricata.yaml* zaujímavou pre túto prácu je *outputs: => - eve-log: => types: => - anomaly:*, ktorá spomína záznamy logov z detekcie anomálií. Hovorí o detekcii špecifických neočakávaných podmienok, objavujúcich sa buď pri dekódovaní paketov, pri sledovaní toku (stream) či pri sledovaní aplikačnej vrstve. Spomínaná detekcia anomálií v tomto súbore podľa dostupných príkladov anomálií zahŕňa aj stavovú analýzu protokolov.

Jeden z komentárov informuje o tom, že zaznamenávanie anomálií je predvolene vypnuté, avšak v našom prípade to zjavne pravda nie je.

```
- anomaly:
    # Anomaly log records describe unexpected conditions such
    # as truncated packets, packets with invalid IP/UDP/TCP
    # length values, and other events that render the packet
    # invalid for further processing or describe unexpected
    # behavior on an established stream. Networks which
    # experience high occurrences of anomalies may experience
    # packet processing degradation.
    #
    # Anomalies are reported for the following:
    # 1. Decode: Values and conditions that are detected while
    # decoding individual packets. This includes invalid or
    # unexpected values for low-level protocol lengths as well
    # as stream related events (TCP 3-way handshake issues,
    # unexpected sequence number, etc).
    # 2. Stream: This includes stream related events (TCP
    # 3-way handshake issues, unexpected sequence number,
    # etc).
    # 3. Application layer: These denote application layer
    # specific conditions that are unexpected, invalid or are
    # unexpected given the application monitoring state.
    #
    # By default, anomaly logging is disabled. When anomaly
    # logging is enabled, applayer anomaly reporting is
    # enabled.
    enabled: yes
    #
    # Choose one or more types of anomaly logging and whether to enable
    # logging of the packet header for packet anomalies.
    types:
      # decode: no
      # stream: no
      # applayer: yes
    #packethdr: no
```

Obr. 4.1: Ukážka suricata.yaml.

4.2 Pravidlá

Pravidlá, v Suricate nazývané aj signatúry (signatures), hrajú veľmi významnú rolu v činnosti Suricaty. Práve na základe nich sa Suricata rozhoduje, čo s každým jedným spracovávaným paketom urobí. Paket sa podrobuje pravidlám postupne, rovnako ako tomu býva pri pravidlách vo firewalle. Avšak s tým rozdielom, že im je vzhľadom na ich akciu prideľovaná priorita, v dôsledku čoho sú spravidla pakety najprv podrobené pravidlám s akciou *pass*, potom *drop*, ďalej *reject* a nakoniec *alert*. Toto poradie je však možné zmeniť v konfiguračnom súbore pod kľúčovým slovom *action-order*.

Pravidlá sa nachádzajú v súbore */var/lib/suricata/rules/suricata.rules*, v ktorom môžme nájsť okrem iného aj jednoduché pravidlá, ktoré určujú, čo je treba vykonať

v prípade detegovanej anomálie. Jedno z týchto pravidiel je napríklad:

alert tcp any any -> any any (msg:"SURICATA STREAM TIMEWAIT ACK with wrong seq"; stream-event:timewait_ack_wrong_seq; classtype:protocol-command-decode; sid:2210042; rev:2;)

Konkrétne toto pravidlo určuje, aká akcia sa má vykonať, ak je v toku detegovaná anomálna udalosť s označením *timewait_ack_wrong_seq*.

Každé pravidlo sa skladá z akcie, záhlavia a možností.

Akcia

Akcia (v príklade pravidla označená červenou farbou, „alert") priamo určuje, čo má byť po splnení podmienok s paketom vykonané. K dispozícii sú 4 rôzne akcie, *pass*, *drop*, *reject* a *alert*.

pass

Zastaví skenovanie paketu a premiestni ho na koniec pravidiel (preskočí všetky nasledujúce pravidlá). V dôsledku toho sa s paketom Suricata ďalej nezapodieva a nijak na neho neupozorňuje.

drop

Zahodí paket, nikam ďalej ho nezasiela a vyvolá *alert*. Cieľová stanica nie je o osude paketu informovaná. Toto pravidlo sa týka iba Suricaty v on-line IPS móde.

reject

Aktívne zamietne paket a informuje odosielateľa aj cieľovú stanicu. V prípade TCP protokoul zašle RST paket, pre všetky ostatné protokoly zase ICMP-error paket. Taktiež vyvolá *alert* a v prípade on-line IPS módu paket zahodí a nikam ďalej ho nezasiela.

alert

Vygeneruje záznam o udalosti disponujúci metadátami podľa v pravidle zadanej konfigurácie.

Záhlavie

Záhlavie (v príklade pravidla označené zelenou farbou, „tcp any any -> any any") určuje pre aký protokol, IP adresy, porty a smer prenosu sa bude pravidlo aplikovať.

Časti záhlavia sú oddelené medzerami. Prvá časť záhlavia definuje protokol. Druhá časť definuje zdrojové adresy a tretia zdrojové porty. Štvrtá časť definuje smer prenosu, pričom sú dve možnosti, jednosmerný „->” a obojsmerný „<>” prenos. Piata časť definuje cieľové adresy a šiesta cieľové porty.

Možnosti

Možnosti (v príklade pravidla označené modrou farbou, „(msg:"SUR ... ; rev:2;)”) slúžia na špecifikáciu pravidla. Možnosti sú ohraničené okrúhlymi zátvorkami. Niektoré z nich majú nastavenia, ktoré nasledujú za dvojbodkou, ostatné sú jednoducho tvorené z jedného kľúčového slova. Záleží na poradí pravidiel, pričom zmena poradia by zmenila význam celého pravidla. Je definované množstvo kľúčových slov, napr. pre pakety jednotlivých protokolov či rôzne úkony. Kľúčové slová s popisom je možné nájsť na adrese `https://suricata.readthedocs.io/en/suricata-5.0.0/rules/`, je ich však príliš mnoho na to, aby boli súčasťou obsahu tejto práce.

4.3 Výstupy a logy

Hlavný výstupom Suricaty je log */var/log/suricata/eve.json*, do ktorého sú vo formáte JSON zapisované výstrahy, anomálie, metadáta, informácie o súboroch a špecifické protokolové záznamy.

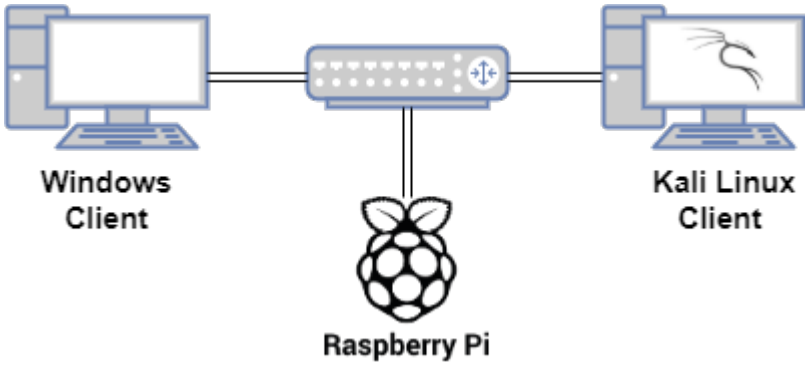
Suricata je taktiež schopná varovať prostredníctvom syslogu, čo je veľmi užitočná funkcia pre centrálny zber logov, dodržiavanie stanovených predpisov v systéme či pre reportovanie do SIEM (Security Information and Event Management).

Okrem iného Suricata podporuje detailnejší výstup o špecifickom sieťovom prenose pomocou prídavných Lua skriptov tým, že ponúka širokú škálu Lua funkcií. Taktiež je pre používateľa možné definovať si vlastný spôsob a formát zaznamenávania prenosu HTTP a TLS protokolov. Pre viac informácií o možnostiach Suricaty viď [22].

5 Testovanie nástroja Suricata

5.1 Príprava prostredia na testovanie

Počas testovania nástroja Suricata bude tento nástroj spustený na jednodoskovom počítači Raspberry Pi, ktorý musíme najprv spojazdniť. RPi bude zapojené do plne funkčného MiktoTik routera s prístupom na internet, ktorý budeme dodatočne konfigurovať. Budeme využívať ešte dvoch klientov, jedného s OS Windows (ako obeť), druhého s OS Kali Linux (ako útočník). Schému zapojenia zariadení je možné vidieť na obr. 5.1.



Obr. 5.1: Zapojenie zariadení v sieti.

5.1.1 Relevantné špecifikácie použitých zariadení

MikroTik Router hAP ac2 (RBD52G-5HacD2HnD-TC)

- 5 Gigabit Ethernet portov
- 802.11a/b/g/n/ac, 2,4/5GHz

Raspberry Pi 2 Model B V1.1

- 900 MHz ARM Cortex-A7 (quad-core)
- 1 GB LPDDR2 SDRAM
- 100 Base Ethernet

Raspberry Pi 4 Model B

- 1,5 GHz Broadcom BCM2711, Cortex-A72 (ARM v8) 64-bit SoC (quad-core)
- 4 GB LPDDR4-3200 SDRAM
- Gigabit Ethernet

Kali Linux klient

- 4,6 GHz Intel Core i7-8700 (prístup k 3 zo 6 jadier)
- 8 GB DDR4 SDRAM
- Gigabit Ethernet

Pre viac informácií o hardvérových špecifikáciách zariadení viď [23], [24] a [25].

5.1.2 Inštalácia operačného systému Raspbian

Náš prvý krok bude inštalácia Raspbianu (operačného systému určeného pre Raspberry Pi). Z oficiálnej stránky si stiahneme jeho obraz, konkrétne „Raspbian Buster with desktop and recommended software“ a po rozbalení ho pomocou odporúčaného nástroja balenaEtcher zapíšeme na 16 GB SD kartu.

Po tomto kroku SD kartu vložíme do RPi, pripojíme myš a klávesnicu pomocou USB a obrazovku cez HDMI rozhranie zariadenia. Cez ethernet rozhranie RPi zapojíme do routera a nakoniec zapojíme napájanie. Tým RPi začne svoje prvé bootovanie. Prejdeme prvotnými nastaveniami zariadenia a RPi je pripravené na používanie. Systém ešte pred prvou prácou s ním aktualizujeme.

5.1.3 Inštalácia nástroja Suricata

Ďalej prejdeme na inštaláciu momentálne najnovšej verzie Suricaty 5.0.0. Postup inštalácie bude nasledovný:

1. Nainštalujeme všetky balíky potrebné pre správny chod a inštaláciu Suricaty.

   $ *sudo -i*
   $ *apt-get -y install libpcre3 libpcre3-dbg libpcre3-dev build-essential autoconf automake libtool libpcap-dev libnet1-dev libyaml-0-2 libyaml-dev zlib1g zlib1g-dev libmagic-dev libcap-ng-dev libjansson-dev pkg-config*

2. Stiahneme archív so Suricatou, rozbalíme ju, skompilujeme a nainštalujeme. Využijeme pri tom príkaz na automatickú konfiguráciu, ktorý vytvorí všetky potrebné adresáre a konfiguračný súbor suricata.yaml. Okrem toho stiahne a nastaví najnovšiu sadu pravidiel z „Emerging Threats“ dostupnú pre Suricatu.

   $ *wget http://www.openinfosecfoundation.org/download/suricata-5.0.0.tar.gz*
   $ *tar -xvzf suricata-5.0.0.tar.gz*
   $ *cd suricata-5.0.0*
   $ *./configure --libdir=/usr/lib64 --prefix=/usr --sysconfdir=/etc --localstatedir=/var --enable-nfqueue --enable-lua*
   $ *make install-full*

Ako už bolo spomenuté, pravidlá, podľa ktorých bude Suricata detegovať hrozby a upozorňovať na ne, sa nastavili automaticky a o každej jednej sade pravidiel, ktorá bola zapísaná do súboru pravidiel *suricata.rules*, sme boli informovaní (viď obr. 5.2). Suricata predvolene využíva iba pravidlá zapísané v tomto súbore.

```
<Info> -- Loading distribution rule file /usr/share/suricata/rules/app-layer-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/decoder-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/dhcp-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/dnp3-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/dns-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/files.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/http-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/ipsec-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/kerberos-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/modbus-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/nfs-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/ntp-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/smb-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/smtp-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/stream-events.rules
<Info> -- Loading distribution rule file /usr/share/suricata/rules/tls-events.rules
```

Obr. 5.2: Sady pravidiel zapísané do súboru suricata.rules.

Taktiež sme boli informovaní, že pravidlá môžu byť aktualizované a spravované pomocou nástroja *„suricata-update“*.

5.1.4 Prvé spustenie nástroja Suricata

Suricatu spustíme príkazom:

$ sudo suricata -c /etc/suricata/suricata.yaml -i eth0

Po spustení sme sa stretli s varovaním:

<Warning> - [ERRCODE: SC_ERR_INVALID_ARGUMENT(13)]

- eve-log dns version not found, forcing it to version 2

Toto varovanie môžeme buď ignorovať, alebo mu predísť tak, že v konfiguračnom súbore */etc/suricata/suricata.yaml* odkomentujeme *outputs: => - eve-log: => types: => - dns: =>* ***version: 2***. Tým sme iba explicitne nastavili DNS logy na verziu 2, čo je zároveň predvolená verzia v Suricate 5.0.0. DNS logy sú predvolene zapisované do súboru eve.json a predstavujú záznamy ďalej nejdeme zaoberať. Úspešné spustenie Suricaty bez varovaní a chýb je možné vidieť na obr. 5.3.

```
pi@raspberrypi:~ $ sudo suricata -c /etc/suricata/suricata.yaml -i eth0
15/12/2019 -- 13:50:55 - <Notice> - This is Suricata version 5.0.0 RELE
ASE running in SYSTEM mode
15/12/2019 -- 13:51:17 - <Notice> - all 4 packet processing threads, 4
management threads initialized, engine started.
```

Obr. 5.3: Suricata oznámila úspešné spustenie.

5.1.5 Dodatočná konfigurácia routeru

Suricatu sme spustili v móde, kedy sleduje prenos na rozhraní *eth0*. To nám však neumožní monitorovanie prenosu, ktorý nie je smerovaný do alebo z RPi. Ale práve tento prenos je v našom záujme sledovať.

Odporúčanou metódou ako sprístupniť prenos, ktorý chceme nechať Suricatou monitorovať, je port mirroring. Na RPi teda budeme chcieť zasielať kópiu všetkého prenosu, čo cez daný port/porty prechádza. Náš router (presnejšie jeho vstavaný switch) však podporuje port mirroring iba z jedného konkrétneho portu na jeden cieľový port. To by nám však malo na testovacie účely zatiaľ stačiť, nakoľko presne vieme, cez ktorý port bude útočník pripojený.

5.2 Test úspešnosti detekcie na RPi 2 a RPi 4

V prvom teste nás bude zaujímať to, ako si Suricata na RPi 2 a RPi 4 poradí s detekciou počas zahlteného prenosu. Doposiaľ nás podľa logov upozorňuje len na potencionálne porušenie stanovenej politiky, konkrétne detegovaný Kali Linux hostname zaznamenaný v pakete DHCP požiadavku. Tým nám v podstate Suricata oznamuje, že je „asi“ do siete pripojený klient s Kali Linuxom (potenciálna hrozba). Poďme však zistiť, či Suricata tento podozrivý DHCP Request deteguje počas zapnutého hping3 floodu miereného na RPi.

5.2.1 Príprava testu

Momentálne máme na RPi otvorené dva terminály so zadanými príkazmi:

1. *$ sudo tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'*
2. *$ sudo tail -f /var/log/suricata/eve.json | grep DHCP*

V prvom príkaze používame nástroj **jq**, ktorý nám zaistí lepšiu čitateľnosť eve.json logu v príkazovom riadku. Druhý príkaz nám pomôže vyhľadať akékoľvek nové záznamy v logu obsahujúce výraz „DHCP”. Bude slúžiť iba na indikovanie úspešnosti detekcie.

Začneme tým, že zmeníme hostname Windows klienta na „kali“ (zmena hostname je jednoduchšia, ako vytvorenie nového Kali Linux klienta). Suricata by mala taktiež upozorniť aj na DHCP Request od tohto klienta, nezáleží na tom, že sa nejedná o Linux.

Po reštarte Windowsu nás Suricata na tohto klienta upozornila. V oboch oknách terminálu nám vyhodilo záznam z *eve.json* obsahujúci okrem iného aj informácie, ktoré je možné vidieť na obrázku 5.4.

```
"signature": "ET POLICY Possible Kali Linux hostname in DHCP Request Packet",
"category": "Potential Corporate Privacy Violation",
```

Obr. 5.4: Časť upozornenia na Kali Linux hostname.

Na Kali spustíme príkaz (*192.168.88.254 je IP adresa RPi)*:

$ hping3 -S -P -U --flood -V --rand-source 192.168.88.254

Pomocou tohto príkazu zahltíme RPi falošným prenosom. Jedná sa o SYN flood útok s pridanými flagmi PSH a URG.

Okno terminálu s druhým príkazom ostáva nezmenené, avšak okno terminálu s prvým príkazom začalo nepretržite vyhadzovať výstrahy so signatúrou a kategóriou, ktoré je možné vidieť na obr. 5.5.

```
"signature": "ET DROP Spamhaus DROP Listed Traffic Inbound group 13",
"category": "Misc Attack",
```

Obr. 5.5: Časť jednej z opakujúcich sa výstrah.

5.2.2 Analýza pravidla s detegovanou signatúrou

Podľa signature id, ktoré bolo uvedené vo výstrahe, môžeme vo */var/lib/suricata/rules/suricata.rules* nájsť pravidlo, ktoré podnietilo danú výstrahu. Podľa tohto pravidla Suricata upozorňuje na prenos z adries pochádzajúcich z predom uvedeného zoznamu rozsahov IP adries. Adresu si zapamätá a najbližšiu hodinu (3600 sekúnd) už na danú adresu neupozorní, aby sa predišlo zahlteniu pamäte či iných zdrojov. Tieto výstrahy však vyhadzuje len z dôvodu, že sme pri spustení hpingu použili parameter *--rand-source*, ktorý náhodne určuje IP adresu zdroja každého paketu, a tým pravidelne zvolí adresu vyhovujúcu podmienke daného pravidla Suricaty.

5.2.3 Testovanie pomocou batch scriptu

Ďalej si na Windows klientovi vypýtame novú IP adresu príkazom *ipconfig /renew*. DHCP request zaslaný síce bol, dostali sme aj odpoveď, akurát Suricata na danú skutočnosť zjavne nereagovala, nakoľko v logu nebola zaznamenaná žiadna nová zmienka o „DHCP". Suricate sa podarilo detegovať signatúru až na štvrtý pokus.

Na zvýšenie poštu pokusov pre lepšiu štatistiku si vytvoríme jednoduchý batch script, ktorý 1000-krát po sebe zavolá príkaz *ipconfig /renew* (viď obr. 5.6). Skúšobné testy vykonané mimo záznam ukázali, že viac zaslaných DHCP požiadavkov by znamenalo dlhšiu dobu ich zasielania, počas ktorej by sa zariadenia stihli až príliš veľmi prehriať, čo by nám nepredvídateľne ovplyvnilo výsledky testov.

So scriptom vykonáme sériu niekoľkých krátkych testov, ktorých výsledok nám po spriemerovaní vypovie o niečo viac o úspešnosti detekcie v takomto rušnom prenose. Záleží nám na rovnakých podmienkach pre každý jeden test, aby výsledky neboli príliš skreslené.

```
@echo off
set loopcount=1000
:loop
ipconfig /renew
set /a loopcount=loopcount-1
if %loopcount%==0 goto exitloop
goto loop
:exitloop
pause
```

Obr. 5.6: Batch script volajúci „ipconfig /renew“.

Riadky vypovedajúce o úspešnej detekcii tentokrát budem zapisovať kvôli prehľadnosti do súboru *test* pomocou príkazu:

$ sudo tail -f /var/log/suricata/eve.json | grep DHCP > /home/pi/Desktop/test

Najprv s RPi 4 vykonáme skúšobný test bez hpingu. Počas tohto testu bolo detegovaných 996 z 1000 podozrivých DHCP request paketov, čo predstavuje 99,6%-nú úspešnosť. Následne budeme testovať so spusteným hpingom RPi 4, potom RPi 2. Po každom jednom z testov si zapíšeme číslo posledného riadku v súbore.

5.2.4 Výsledky testov

Výsledkom testov na RPi 4 je množina zapísaných čísel {292, 272 , 292, 359, 369}, ktorých priemer je po zaokrúhlení rovný **317**. Toto číslo nám vzhľadom k 1000 pokusom hovorí o **31,7%**-nej úspešnosti pri danom zaťažení systému spustenými aplikáciami a službami v pozadí.

Výsledok testov na RPi 2 bol zásadne horší. Už pri prvom teste bol spozorovaný značný pokles úspešnosti detekcie, výsledok testov bol {113, 98, 133, 77, 92}, ich priemer dosahoval hodnotu **103**, jednalo sa teda o **10,3%**-nú úspešnosť.

Z dôvodu značne vyššieho výkonu (dokázaného aj predchádzajúcimi testami) budeme pri ďalšom testovaní používať výhradne RPi 4.

5.3 Detekcia útoku SYN flood

V rámci predchádzajúceho testu sme použili nástroj hping3 a využili ho na zahltenie prenosu útokom SYN flood. Suricata však na tento útok nijak upozornila. SYN flood ako DoS či DDoS útok vie narobiť v sieti značnú škodu, v niektorých prípadoch jednoducho dokáže odstaviť router. Ďalej sa teda pokúsime otestovať detekčné schopnosti Suricaty pri pokuse o tento útok za rôznych podmienok a prípadne analyzujeme pravidlá, vďaka ktorým bude detegovaný.

Tentokrát využijeme príkaz s presnejšími parametrami, pričom sa zameriame len na SYN flag protokolu TCP, nakoľko on je ten, vďaka ktorému sa v útoku dosahuje polootvorených spojení.

5.3.1 Vykonanie útoku a analýza záznamov

Najprv budeme útočiť priamo na RPi:

$ hping3 --flood -S -d 100 192.168.88.254

Zvolili sme veľkosťou dát v pakete 100 bajtov a mód flood, vďaka ktorému budú pakety po sebe posielané čo najrýchlejšie.

V momente začal byť log striedavo zaplavovaný 2 typmi záznamov, pričom jeden z nich je možné vidieť na obrázku 5.7.

Druhý typ je prakticky rovnaký, až na parametre *signature_id* a *signature*:

”signature_id”: 2210045
”signature”: ”SURICATA STREAM Packet with invalid ack”

Rovnaká časová značka pri oboch záznamoch nám hovorí o to, že sa jedná o jeden paket, ktorý vyhovoval dvom pravidlám. Jedno pravidlo upozorní na neplatný ACK v TCP pakete, druhé zase na neplatný ACK v TCP pakete s RST flagom. Zachytený paket mal ACK=101, nakoľko sme zvolili veľkosť dát 100 bajtov. Ak však veľkosť dát zmeníme na 0 bajtov, Suricata prestane upozorňovať na neplatný ACK. S takýmito paketmi si síce vie dať rady oveľa ľahšie, čoho následkom nie je jej úplne odstavenie, ale len značné spomalenie. Pri vyššej frekvencii zasielania paketov (DDoS SYN flood) by však absencia dát nebola žiadny problém.

Každopádne, hlavný dôvod, prečo nemôžme detekciu postaviť na neplatných ACK v paketoch s RST flagom je, že paket s RST flagom nie je zasielaný vždy.

```
{
  "timestamp": "2019-12-14T17:58:04.792283+0100",
  "flow_id": 2079829668927179,
  "in_iface": "eth0",
  "event_type": "alert",
  "src_ip": "192.168.88.254",
  "src_port": 0,
  "dest_ip": "192.168.88.245",
  "dest_port": 49883,
  "proto": "TCP",
  "flow": {
    "pkts_toserver": 1,
    "pkts_toclient": 1,
    "bytes_toserver": 154,
    "bytes_toclient": 54,
    "start": "2019-12-14T17:58:04.792267+0100"
  },
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 2210046,
    "rev": 2,
    "signature": "SURICATA STREAM SHUTDOWN RST invalid ack",
    "category": "Generic Protocol Command Decode",
    "severity": 3
  }
}
```


Obr. 5.7: Jedna z opakujúcich sa výstrah.

Ako príklad si vezmime útoku na Windows klienta. Počas tohto útoku so stanovenou veľkosťou dát v paketoch 500 bajtov bolo rozhranie zariadenia zahltené natoľko, že s predvolenou veľkosťou pingu na router bol úspešný približne každý ôsmi. Pri 1000 bajtoch úspešnosť pingu klesla na 0%. V skutočnosti bolo však pri teste využité podstatne pomalšie bezdrôtové rozhranie, z dôvodu čoho došlo na Windows klientovi oveľa rýchlejšie ku kritickému využitiu siete. Čo sa však týka Suricaty, neupozornila nás na žiadnu podozrivú aktivitu, nakoľko Windows klient nezasielal žiadne pakety s RST flagom.

5.3.2 Tvorba vlastného pravidla

Z dôvodu nevyhovujúcej detekcie si potrebujeme na tento útok vytvoriť vlastné pravidlo. Spôsob zostavovania pravidiel je rovnaký ako u nástroja Snort, viď [27]. Najjednoduchší spôsob, ako tento útok zachytiť, bude sledovať počet paketov so SYN flagom a varovať, keď za určitý stanovený časový interval počet paketov so SYN flagom s rovnakou destináciou presiahne určitú hodnotu.

Dôležitou otázkou ale bude, aký časový interval a počet paketov treba zvoliť.

Tieto hodnoty by sa mali líšiť podľa normálneho prenosu v danom bode v sieti, kde je Suricata umiestnená. Interval si určíme tak, aby jednak nebol príliš krátky, nakoľko by sme v tom prípade museli zvoliť nižšiu aj druhú hodnotu (počet paketov). Nechcem totiž, aby bola príliš vysoká pravdepodobnosť, že bude označený nárazový legitímny prenos. Interval však nemôžme zvoliť až príliš dlhý, nakoľko by musel byť zvýšený aj povolený počet paketov a útok by bol z toho dôvodu označený neskoro. Naviac by útočník vedel nárazovo posielať v danom intervale menší počet paketov, ako by bola hodnota určovaná pravidlom, a bez detekcie zneužiť krátku nedostupnosť daného zariadenia.

Experimentálne zvolíme interval 3 sekundy. Počet paketov zvolíme v rozmedzí ohraničenom najvyšším legitímnym prenosom, ktorý môžme v danom bode očakávať, a najnižším počtom paketov, ktorý by vedel v danom časovom intervale najzraniteľnejšie zo zariadení v sieti patrne ovplyvniť v jeho činnosti. Týmto zariadením je v našej sieti samotné RPi, ktoré okrem nízkej hardvérovej výkonnosti musí pakety spracovať aj v rámci monitorovacej činnosti Suricaty.

Určíme si, že v našej sieti môžme napríklad očakávať maximálne 10 zariadení bežných užívateľov. Na Windows klientovi sme pomocou Wiresharku za minútu odchytili pri velice aktívnom prehliadaní webových stránok presne 320 paketov s flagom SYN. Na dvadsať zariadení to teda bude 3200 paketov. Z tohoto čísla si vypočítame, že za 3 sekundy to bude 160 paketov. Aby sme skutočne predišli falošne pozitívnym nálezom, toto číslo pre istotu zvýšime, nakoľko takýto nízky počet paketov ani za veľmi krátku dobu nijak RPi neovplyvní. Zvolíme si napr. hodnotu 500.

V málo frekventovaných sieťach ako je tá naša väčšinou nie je ťažké zvoliť si vhodné parametre pre jednoduché pravidlá. V iných sieťach by však správna voľba parametrov mohla vyžadovať omnoho viac času a úsilia.

Naše prvé pravidlo teda bude vyzerať nasledovne:

alert tcp any any -> any any (msg:"DOS ATTACK TCP_SYN_flood"; flags: S; threshold: type both, track by_dst, count 500, seconds 3; classtype: attempted-dos; sid: 1000001; rev: 1;)*

alert – počas útoku chceme byť varovaní

tcp – útok sa odohráva pomocou TCP paketov

any any -> any any – útoky chcem detegovať bez ohľadu na IP adresu a port zdroja či cieľa

msg:"DOS ATTACK TCP_SYN_flood" – vystihujúci názov signatúry

flags: S* – počítať sa budú iba TCP pakety, ktoré majú flag SYN (ak by nebol použitý znak „*", boli by počítané iba tie pakety, ktoré obsahujú iba flag SYN)

threshold:

type both – používame obidva typy thresholdu, jeden funguje pre nás ako počítadlo, vďaka ktorému nás pravidlo varuje po prekročení daného počtu vyhovujúcich paketov za určitý čas, a druhý zase v kombinácii s prvým limituje počet výstrah za stanovený čas na jednu.

track by_dst – počítať sa budú pakety pre každú cieľovú IP adresu zvlášť

count 500 – počet vyhovujúcich paketov, ktorý vyvolá akciu

seconds 3 – čas, za ktorý musí byť detegovaný daný počet vyhovujúcich paketov, aby bola vyvolaná akcia, zároveň stanovuje interval medzi jednotlivými výstrahami

classtype: attempted-dos – klasifikácia signatúry, priraďuje prioritu

sid: 1000001 – volíme z hodnôt 1000000 – 1999999, ktoré sú podľa Emerging Threats určené pre vlastné pravidlá

rev: 1 – udáva verziu pravidla s daným *sid*

5.3.3 Testovanie nového pravidla

Vytvoríme si v priečinku */var/lib/suricata/rules/* nový súbor, napr. s názvom *custom.rules*, a vložíme do neho naše nové pravidlo. Ďalej editujeme konfiguračný súbor */etc/suricata/suricata.yaml*, nájdeme si výraz „- suricata.rules" a rovnako do ďalšieho riadku dopíšeme „- custom.rules". Týmto sme pridali súbor s novým pravidlom do konfiguračného súbor a po najbližšom spustení Suricaty bude načítané. Naviac je vhodné zakomentovať predvolený súbor s pravidlami, aby nás pri teste nerušili žiadne iné výstrahy.

Nové pravidlo odskúšame s nasledovnými *hping* príkazmi:

- *hping3 -q -c 1200 -i u1 -S 192.168.88.242* (RPi)
- *hping3 -q -c 1200 -i u1 -S 192.168.88.1* (router)
- *hping3 -q -c 1200 -i u1 -S 192.168.88.243* (klient)
- *hping3 -q -c 1200 -i u1 -S -U -P 192.168.88.243* (flagy SYN, URG, PSH)
- *hping3 -q -c 500 -i u1 -S 192.168.88.243* (500 paketov)
- *hping3 -q -c 499 -i u1 -S 192.168.88.243* (499 paketov)
- *hping3 --flood -S -d 1460 192.168.88.243* (neobmedzený flood)

5.3.4 Výsledok testu

Po zadaní každého z príkazov okrem posledných dvoch nás Suricata jedenkrát upozornila na „DOS ATTACK TCP_SYN_flood", po predposlednom príkaze nás neupozornila vôbec a po poslednom nás upozorňovala každé tri sekundy. Ukážku záznamu obsahujúceho výstrahu vyvolanú novým pravidlom je možné vidieť na obr.

5.8. Pravidlo teda robí to, čo má, a testom úspešne prešlo.


```
{
  "timestamp": "2019-12-17T21:59:23.942646+0100",
  "flow_id": 664209204142646,
  "in_iface": "eth0",
  "event_type": "alert",
  "src_ip": "192.168.88.239",
  "src_port": 1985,
  "dest_ip": "192.168.88.1",
  "dest_port": 0,
  "proto": "TCP",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 1000001,
    "rev": 1,
    "signature": "DOS ATTACK TCP_SYN_flood",
    "category": "Attempted Denial of Service",
    "severity": 2
  },
  "flow": {
    "pkts_toserver": 1,
    "pkts_toclient": 0,
    "bytes_toserver": 60,
    "bytes_toclient": 0,
    "start": "2019-12-17T21:59:23.942646+0100"
  }
}
```


Obr. 5.8: Ukážka výstrahy nového pravidla.

5.4 Detekcia útoku ARP spoofing

V tomto teste sa zameriame na protokol tretej vrstvy modelu OSI, avšak útoky na tento protokol sa priamo vzťahujú na vrstvu druhú. Týmto protokolom je ARP. Konkrétne sa budeme zaoberať útokom ARP spoofing a tým, ako si s ním Suricata poradí.

Samotný útok pre ňu môže byť problémom, nakoľko podľa dokumentácie nepracuje na linkovej vrstve a nemá prostriedky na detekciu útokov mierených na ARP protokol, ako aj MAC, CAM či VLAN útokov.

5.4.1 Vykonanie útoku a overenie jeho úspešnosti

Vykonanie útoku bude pomerne jednoduché. V Kali Linuxe budeme využívať nástroj príkazového riadku *arpspoof*, ktorý na sieti s prepínačom dokáže presmerovať pakety určené obeti tým, že neustále zasiela ARP odpovede, ktoré informujú switch o tom,

že IP adresa obete patrí k MAC adrese útočníka. Rovnakým spôsobom s ním dokážeme presmerovať prenos určený predvolenej bráne. Náš router našťastie switchom disponuje, tým pádom nepotrebujeme žiadne ďalšie zariadenie.

Na routeri zvolíme možnosť switchu *Switch All Ports*. Ešte pred tým, ako začneme zasielať ARP odpovede, musíme zaistiť to, že prenos bude môcť byť preposielaný obeti. Na to musíme mať povolený IP forwarding na Kali Linuxe, čoho dosiahneme príkazom:

$ *echo 1 > /proc/sys/net/ipv4/ip_forward*

Toto nastavenie sa aplikuje okamžite a po reštarte zariadenia už nebude platné.

Následne môžeme začať so samotným útokom. Ten bude spočívať z dvoch častí. V prvej sa pokúsime o narušenie prenosu v smere obeť -> router, súčasťou ktorej je zasielanie ARP odpovedí s informáciou, že IP adresa predvolenej brány je na MAC adrese útočníka. V druhej časti zase narušíme prenos v smere router -> obeť a jej súčasťou bude zasielanie ARP odpovedí s IP adresou obete a opäť MAC adresou útočníka (192.168.88.1 – router, 192.168.88.252 – Windows klient):

1. *arpspoof -i eth0 -t 192.168.88.252 192.168.88.1*
2. *arpspoof -i eth0 -t 192.168.88.1 192.168.88.252*

V okne terminálu s prvým príkazom môžeme vidieť opakované oznámenie:

```
8:[redacted]:a7 74:[redacted]:d0 0806 42: arp reply 192.168.88.1 is-at 8:[redacted]:a7
```

V terminály s druhým príkazom je to zase:

```
8:[redacted]:a7 74:[redacted]:98 0806 42: arp reply 192.168.88.252 is-at 8:[redacted]:a7
```

Séria znakov *8:xx:xx:xx:xx:a7* predstavuje MAC adresu útočníka.

Po spustení príkazov nás Suricata očakávane neupozornila na žiadnu hrozbu. Skúsme si teda overiť, či bol útok úspešný. Po otvorení ARP tabuľky na routeri si môžeme všimnúť, že k dvom IP adresám, z čoho jedna je adresa obete, je priradená rovnaká MAC adresa – MAC adresa útočníka (viď obr. 5.9).

-	DC	192.168.88.250	08:[redacted]:A7
-	DC	192.168.88.252	08:[redacted]:A7

Obr. 5.9: Ukážka z ARP tabuľky.

K IP adrese brány ostáva naďalej pridelená správna MAC adresa. To však ešte neznamená, že nebol útokom ovplyvnený samotný switch.

V Kali Linuxe si spustíme Wireshark a začneme monitorovať prenos na rozhraní pripojenom do siete (v našom prípade *eth0*). Ďalej aplikujeme filter *ip.addr==192.168.88.252 and http*. Po zadaní adresy *http://google.com* v prehliadači obete môžeme vo Wiresharku vidieť záznamy prezentované na obr. 5.10.

Time	Source	Destination	Protocol
25.512195390	192.168.88.252	172.217.23.238	HTTP
25.577352471	172.217.23.238	192.168.88.252	HTTP
25.689270531	192.168.88.252	216.58.201.100	HTTP
25.764338674	216.58.201.100	192.168.88.252	HTTP

Obr. 5.10: Záznam z Wiresharku.

V záznamoch si môžeme všimnúť IP adresu obete ako na strane zdroja, tak na strane cieľa. To znamená, že sa nám darí odchytávať prenos smerom od obete ale aj prenos smerom k obeti. Náš útok bol teda úspešný.

5.4.2 Analýza záznamov Suricaty

Suricata nás zatiaľ 4-krát upozornila na pakety s neplatnou časovou značkou (viď obr. 5.11). Všetky 4 výstrahy sú skoro rovnaké. Nahliadnime teda medzi riadky pravidiel Suricaty aby sme zistili, čo výstrahy vyvolalo:

alert tcp any any -> any any (msg:"SURICATA STREAM Packet with invalid timestamp"; stream-event:pkt_invalid_timestamp; classtype:protocol-command-decode; sid:2210044; rev:2;)

Podľa tohto pravidla nás Suricata varuje vždy, keď v TCP pakete zaznamená neplatnú časovú značku (invalid timestamp). Bohužiaľ, nijak priamo neukazuje na existenciu konkrétnej hrozby.

5.4.3 Výsledok testu

Z testu môžeme vyvodiť, že Suricata nedokáže rozpoznať a detegovať priebeh ARP spoofingu. Vytvorením pravidla na báze detekcie anomálií neplatných časových značiek by bolo teoreticky možné detegovať následky tohto útoku, avšak efektívnosť riešenia vzhľadom k falošne pozitívnym nálezom by mohla byť príliš nízka. Týmto môžme neschopnosť Suricaty monitorovať prenos protokolu ARP označiť za značný nedostatok, ktorý by bolo vhodné kompenzovať.

Na RPi sa teda pokúsime zabezpečiť nástroj schopný detegovať útoky mierené na tento protokol. Potenciálnym kandidátom je nenáročný nástroj Arpwatch generujúci

```
{
  "timestamp": "[redacted]T15:45:33.682306+0100",
  "flow_id": 2028912503463892,
  "in_iface": "eth0",
  "event_type": "alert",
  "src_ip": "192.168.88.252",
  "src_port": 49988,
  "dest_ip": "216.58.201.68",
  "dest_port": 443,
  "proto": "TCP",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 2210044,
    "rev": 2,
    "signature": "SURICATA STREAM Packet with invalid timestamp",
    "category": "Generic Protocol Command Decode",
    "severity": 3
  },
  "tls": {
    "sni": "www.google.com",
    "version": "TLS 1.3",
    "ja3": {
      "hash": "b20b44b18b853ef29ab773e921b03422",
      "string": "771,4865-4867-4866-49195-49199-52393-52392-49196-
1-49172-51-57-47-53-10,0-23-65281-10-11-35-16-5-51-43-13-45-28-21,
    },
    "ja3s": {
      "hash": "eb1d94daa7e0344597e756a1fb6e7054",
      "string": "771,4865,51-43"
    }
  },
  "app_proto": "tls",
  "flow": {
    "pkts_toserver": 89,
    "pkts_toclient": 157,
    "bytes_toserver": 10414,
    "bytes_toclient": 191762,
    "start": "[redacted]T15:45:32.565204+0100"
  }
}
```


Obr. 5.11: Ukážka výstrahy po zadaní adresy http://google.com.

záznamy odpozorovaných párov IP a MAC adries a schopný sledovania a upozorňovania na zmeny v párovaní či na nové páry. Tento nástroj bude potrebné otestovať a zhodnotiť jeho vhodnosť ako doplnok k Suricate. Inštaláciou a testovaním ARP-watchu sa zaoberá nasledujúca kapitola.

6 ARPwatch

Nástroj ARPwatch sleduje párovanie ethernet a ip adries, aktivitu zasiela do syslogu a určité zmeny oznamuje prostredníctvom mailu. ARPwatch používa pcap (3) knižnicu na počúvanie ARP paketov na lokálnom ethernetovom rozhraní.

6.1 Inštalácia nástroja ARPwatch

Nástroj nainštalujeme jednoducho príkazom:

sudo apt-get install arpwatch

Pre správne fungovanie potrebujeme vytvoriť súbor *arp.dat*, do ktorého sa budú ukladať adresné páry a povoliť ARPwatch na rozhraní, ktoré budeme monitorovať:

$ sudo touch /var/lib/arpwatch/arp.dat
$ sudo systemctl enable arpwatch@eth0

Na fungovanie mailu potrebujeme ešte nainštalovať ďalšie dva balíčky.

$ sudo apt-get install mailutils
$ sudo apt-get install sendmail

ARPwatch nepodporuje konfiguračný súbor, nastavuje sa pri spustení pomocou parametrov, napr.:

$ sudo arpwatch -i eth0 -m test@mail.com

Ďalšie možnosti spustenia je možné nájsť na stránke ARPwatch manuálu (viď [28]).

6.2 Syslog správy

ARPwatch zvláštne chovanie či nové adresné páry sysloguje pomocou nasledovných správ. Prvé 4 z týchto správ sú taktiež oznamované mailom.

new activity – pár adries bol použitý prvýkrát po šiestich alebo viac mesiacoch
new station – ethernetová adresa bola zaznamenaná prvýkrát
flip flop – ethernetová adresa sa zmenila z naposledy videnej adresy na druhú naposledy videnú adresu
changed ethernet address – stanica zmenila ethernetovú adresu
ethernet broadcast – MAC ethernetová adresa stanice je broadcastovou adresou

ip broadcast – ip adresa stanice je broadcastovou adresou
bogon – zdrojová ip adresa nie je lokálnou v lokálnej podsieti
ethernet broadcast – MAC alebo ARP ethernetového adresa zdroja obsahovala samé nuly alebo samé jednotky
ethernet mismatch – MAC ethernetová adresa zdroja sa nezhodovala s adresou v ARP pakete
reused old ethernet address – ethernetová adresa sa zmenila z naposledy videnej adresy na tretiu (alebo vyššiu) naposledy videnú adresu (podobné flip flop)

Viac o správach je možné nájsť na stránke ARPwatch manuálu (viď [20]).

6.3 Detekcia útoku ARP spoofing

ARPwatch vďaka nastavenému port mirroringu na routery bude môcť odchytávať ARP pakety z ethernetového rozhrania. Nástroj spustíme príkazom:

sudo arpwatch -i eth0

Ako prvé budeme pravdaže testovať schopnosť zachytiť útok ARP spoofing, teda jeho príznaky. Príznakom by mala byť minimálne zmena ethernetovej adresy pre určitú IP, očakávame teda správu „changed ethernet addres".
Adresné páry pred útokom:

router	74:xx:xx:xx:xx:98	192.168.88.1
obeť	c8:xx:xx:xx:xx:d1	192.168.88.253
útočník	08:xx:xx:xx:xx:a7	192.168.88.254

Útok spustíme v Kali Linuxe príkazmi:

1. *arpspoof -i eth0 -t 192.168.88.253 192.168.88.1*
2. *arpspoof -i eth0 -t 192.168.88.1 192.168.88.253*

Po chvíli útok zastavíme. V syslogu sa nám objavilo niekoľko záznamov z ARPwatchu (viď obr. 6.1).

Ako prvú vidíme očakávanú správu vypovedajúcu o zmene ethernetovej adresy. Ide o IP adresu obete, pri ktorej sa eth. adresa obete (v zátvorkách) zmenila na eth. adresu útočníka. Následne boli zaznamenané 4 správy „flip flop", ktoré hovoria o striedaní eth. adries útočníka a obete. Ďalších 5 správ bolo zaznamenaných počas ukončovania útoku, kedy útočník zasielal odpovede zo správnym párom adries obete, avšak zo svojej eth. adresy. Eth. adresa paketu sa teda nezhodovala s adresou zdroja

```
arpwatch: changed ethernet address 192.168.88.253 08:[redacted]:a7 (c8:[redacted]:d1) eth0
arpwatch: flip flop 192.168.88.253 c8:[redacted]:d1 (08:[redacted]:a7) eth0
arpwatch: flip flop 192.168.88.253 08:[redacted]:a7 (c8:[redacted]:d1) eth0
arpwatch: flip flop 192.168.88.253 c8:[redacted]:d1 (08:[redacted]:a7) eth0
arpwatch: flip flop 192.168.88.253 08:[redacted]:a7 (c8:[redacted]:d1) eth0
arpwatch: ethernet mismatch 192.168.88.253 08:[redacted]:a7 (c8:[redacted]:d1) eth0
arpwatch: ethernet mismatch 192.168.88.253 08:[redacted]:a7 (c8:[redacted]:d1) eth0
arpwatch: ethernet mismatch 192.168.88.253 08:[redacted]:a7 (c8:[redacted]:d1) eth0
arpwatch: ethernet mismatch 192.168.88.253 08:[redacted]:a7 (c8:[redacted]:d1) eth0
arpwatch: ethernet mismatch 192.168.88.253 08:[redacted]:a7 (c8:[redacted]:d1) eth0
arpwatch: flip flop 192.168.88.253 c8:[redacted]:d1 (08:[redacted]:a7) eth0
```

Obr. 6.1: Syslog záznamy ARPwatchu.

tohoto paketu. Po týchto správach bola zaznamenaná už len jedna „flip flop” správa hovoriaca o zmene na eth. adresu obete (dôsledok ukončenia útoku).

ARPwatch je schopný detegovať príznaky ARP spoofingu a tým pádom aj významnú časť MITM útokov využívajúcich túto techniku. Aj vďaka jeho ostatným schopnostiam odhaliť zvláštne chovanie (už zo svojej podstaty nebezpečného) ARP protokolu ho teda môžme považovať za vhodné rozšírenie detekcie.

7 Návrh systému detekcie útokov v lokálnej sieti využívajúceho open-source nástroje

Táto kapitola sa venuje návrhu systému, ktorý bude pracovať na jednodoskovom počítači Raspberry Pi 4. Jeho úlohou bude (in-line) detekcia útokov v lokálnej sieti zaistená nástrojom Suricata IDS, ktorý bude doplňovať ďalší nástroj ARPwatch schopný detegovať nezvyčajné správanie a útoky prirodzene nebezpečného protokolu ARP často využívaného v rámci útokov na lokálnej sieti.

7.1 Operačný systém a nastavenia routeru

Na RPi 4 bude bežať operačný systém **Raspbian GNU/Linux 10 (buster)**, ktorý bol priamo vyvinutý pre toto zariadenie.

Na switchy vstavanom v MikroTik routery bude nastavený port mirroring (aspoň z jedného portu), aby bolo možné monitorovať všetky dáta pretekajúce cez daný port/porty a odchytávať prípadné útoky. Toto nastavenie je potrebné pre nástroj Suricata, ARPwatch by sa obišiel aj bez neho.

7.2 Nastavenie Suricaty

Suricata, ktorá bude v tomto systéme hrať najvýznamnejšiu rolu, bude (momentálne) najnovšej verzie, t.j. verzia 5.0.3. Túto verziu si buď postavíme zo zdrojového kódu (rovnako ako v podkapitole 5.1.3), alebo ju pre Raspbian (založený na Debiane) nainštalujeme z *Debian Backports*.

Ďalej nakonfigurujeme všetko čo bude treba, aby Suricata bežala bez chýb, syslogovala všetky alerty a vytvárala všetky potrebné logy. Alerty bude *rsyslog* zapisovať do samostatného logu, odkiaľ ich budeme čerpať na prezentovanie vo vytvorenej aplikácii.

Taktiež podľa odporúčaní zakážeme určité pravidlá vyvolávajúce falošne pozitívne výstrahy a na rozšírenie detekcie pridáme ďalšie zdroje pravidiel, z ktorých bude *suricata-update* dané pravidlá nahrávať. Zo zdrojov, ktoré ma nástroj na správu pravidiel v zozname, povolíme *et/open* (základné pravidlá), *ptresearch/attackdetection* a *tgreen/hunting*.

7.2.1 Popis pravidiel vybratých na detekciu útokov

AttackDetection pravidlá sú vyvinuté pre detekciu útokov na L3 vrstve tímom Attack Detection Team, ktorý sa zaoberá hľadaním nových zraniteľností a zero days

rovnako ako aj TTP (taktikami, technikami a procedúrami) hackerov a malwaru.

Hunting pravidlá sa okrem iného zaoberajú anomálnou detekciou škály techník útočníkov ako MS Office makrá, využitie zraniteľných funkcií či škodlivých skriptov v rôznych jazykoch, vyvolávanie PowerShellu (v payloade) apod.

Do zoznamu zdrojov by ďalej bolo vhodné pridať ešte jeden „open" zdroj, viď [29]. Jedná sa o kolekciu pravidiel detegujúcich rôzne škodlivé chovanie (zneužívanie shellov, crypto mining, využitie CobaltStriku, backdoor, atď).

7.3 Nastavenie ARPwatchu

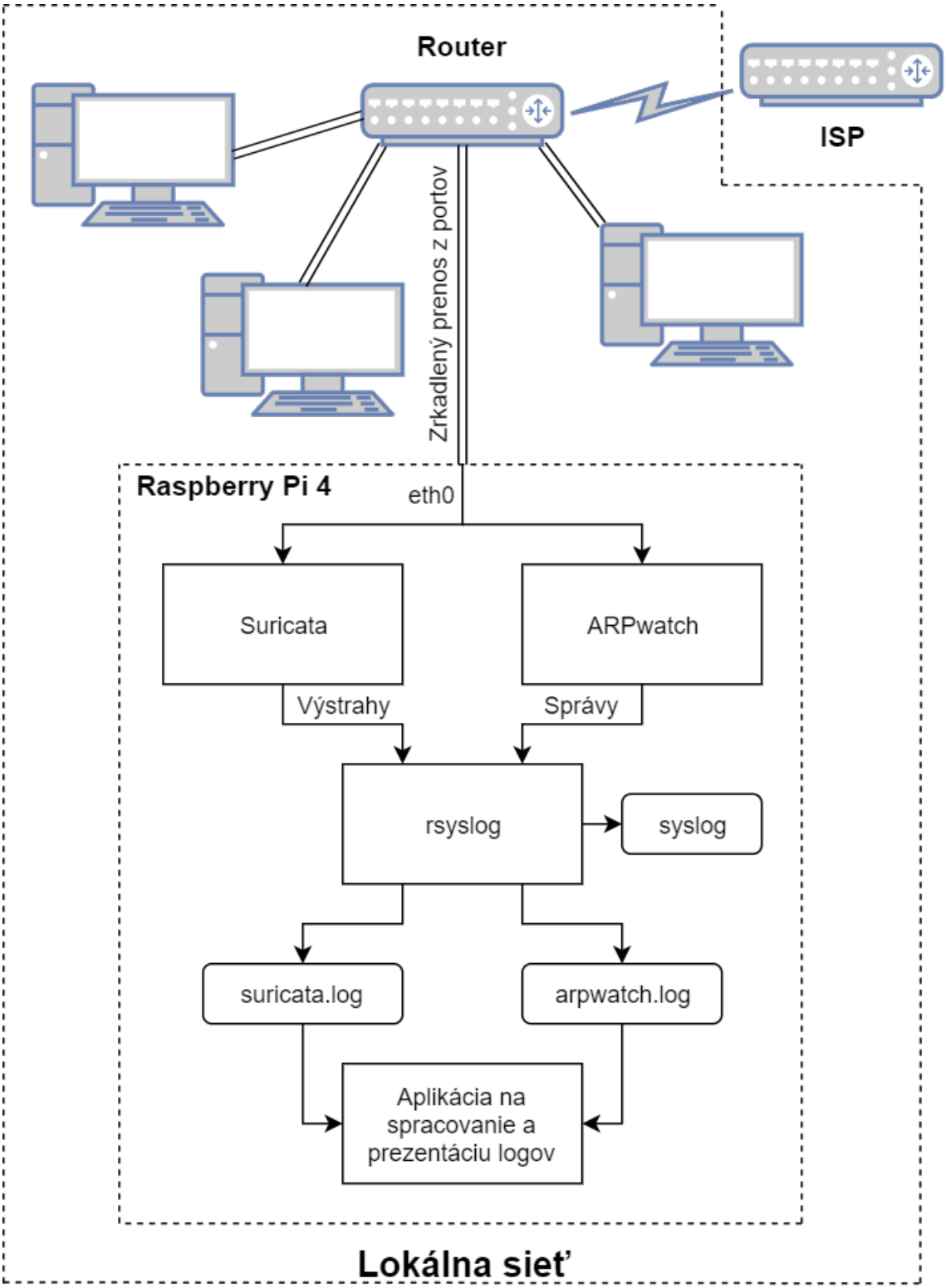
Čo sa týka nástroja ARPwatch, ten do syslogu zaznamenáva už predvolene a nie je potrebné ho nijak konfigurovať. Jediné, čo bude treba vykonať, je rovnako ako u Suricaty nakonfigurovať *rsyslog* aby zapisoval všetky záznamy do samostatného logu na ďalšie spracovanie.

7.4 Priebeh detekcie útokov

Ako bolo spomenuté, prenos na routery bude zrkadlený smerom k RPi. V RPi bude všetok prenos L3 a vyšších vrstiev analyzovaný Suricatou a ARP pakety L3 vrrstvy zase ARPwatchom. Suricata bude jednak vytvárať vlastné logy, ktoré budú ďalej spracovávané vo vytvorenej aplikácii, a taktiež bude syslogovať čisto výstrahy. ARPwatch bude syslogovať všetko, nakoľko každú novú alebo zvláštnu aktivitu na ARP protokole považujeme za potenciálnu hrozbu, avšak ARPwatch žiadnu hrozbu nemôže určiť jednoznačne.

Rsyslog bude triediť všetky syslogované záznamy do samostatných logov, osobitne pre Suricatu a ARPwatch. Tieto logy budú využité ako jeden zo vstupov vo vytvorenej aplikácii a budú priamo prezentované ako záznamy najvyššej priority.

Blokovú schému detekcie je možné vidieť na obr. 7.1.



Obr. 7.1: Bloková schéma systému detekcie útokov.

8 Praktické overenie návrhu systému

Pri zostavovaní systému začneme inštaláciou operačného systému na SD kartu s kapacitou 32 GB. Postup je rovnaký ako v podkapitole 5.1.2. Systém aktualizujeme, reštartujeme a prejdeme k inštalácii nástrojov. Na prácu s RPi môžme buď využiť samostatné vstupné a výstupné zariadenia, alebo použijeme bezplatnú aplikáciu TeamViewer (ktorá je v tomto prípade využitá).

8.1 Príprava Suricaty

Po inštalácii TeamVieweru a úspešným prepojením s Windows klientom nainštalujeme nástroj Suricata z Debian backports:

*$ echo "deb http://http.debian.net/debian buster-backports main" > *
/etc/apt/sources.list.d/backports.list
$ sudo apt-get update
$ sudo apt-get install suricata -t buster-backports

V našom prípade bolo potrebné získať a exportovať vyžadované kľúče pomocou *gpg*, aby bolo možné daný zdroj pridať. Kľúče boli potom pridané pomocou *apt-key* príkazu. Naviac bolo treba ešte pridať zdroj *stretch-backports* kvôli inštalácii závislostí potrebných na fungovanie Suricaty.

8.1.1 Správa pravidiel

Nainštalujeme *suricata-update*, aktualizujeme si zoznam zdrojov pravidiel a zobrazíme ich zoznam:

$ sudo apt-get install suricata-update
$ sudo suricata-update update-sources
$ sudo suricata-update list-sources

V zozname zdrojov sú uvedené voľne využiteľné ale aj komerčné zdroje. Nás budú pravdaže zaujímať bezplatné „open" zdroje. Jednotlivé zdroje boli spomenuté v predchádzajúce kapitole, ostatné voľné zdroje nebudú nijak priamo nápomocné k detekcii útokov. Povolíme ich nasledovne:

$ sudo suricata-update enable-source et/open
$ sudo suricata-update enable-source ptresearch/attackdetection
$ sudo suricata-update enable-source tgreen/hunting

Nový zdroj pridáme pomocou príkazu *add-source*, tímto bude aj automaticky povolený:

*$ sudo suricata-update add-source suricata-ids/rules *
https://raw.githubusercontent.com/suricata-rules
/suricata-rules/master/suricata-ids.rules

Ešte pred načítaním pravidiel zo zdrojov chceme zakázať pravidlá, ktoré sú známe kvôli falošne pozitívnym nálezom. Z rovnakého zdroja ako bol novo pridaný zdroj pravidiel (viď [29]) si stiahneme zoznam pravidiel vo forme textového súboru s názvom *disable.conf*, v ktorom sú uvedené konkrétne nevyhovujúce pravidlá, ale aj triedy výstrah, ktoré spravidla nemajú dostatočnú výpovednú hodnotu aby dokázali vierohodne určiť, že ide o útok alebo nejaké škodlivé správanie. Tento súbor vložíme do priečinku */etc/suricata*.

Po spustení *suricata-update* a načítaní pravidiel budú konkrétne pravidlá v súbore s pravidlami (*/var/lib/suricata/rules/suricata.rules*) zakomentované (zakázané). Súbor *disable.conf* je vhodné počas dlhodobej činnosti Suricaty aktualizovať o pravidlá, ktoré pravidelne vyvolávajú falošné výstrahy.

Poslednú vec, ktorú ohľadne pravidiel vykonáme, je úprava konfiguračného súboru */etc/suricata/suricata.yaml*. Táto úprava súvisí s pravidlami zo zdroja *ptresearch/attackdetection*, nakoľko niektoré z pravidiel sú mierené na detekciu v komunikácii šifrovanej protokolom TLS. Na ich aktiváciu bude treba odkomentovať a nastaviť *app-layer: => protocols: => tls: => encryption-handling:* **full**. Tým samozrejme trochu znížime výkon, v našom prípade by ale nemalo ísť o žiadnu zásadnú zmenu.

8.1.2 Logovanie

Ako prvé nakonfigurujeme Suricatu pomocou súboru */etc/suricata/suricata.yaml* tak, aby syslogovala všetky výstrahy. To dosiahneme zmenením hodnoty *outputs: => - syslog: => enabled:* **no** na **yes**.

Ďalej nakonfigurujeme *rsyslog* aby syslogované výstrahy zo Suricaty zapisoval do samostatného logu. V adresári /etc/rsyslog.d teda vytvoríme súbor, napr. *surwatch.conf* a zapíšeme do neho nasledujúci riadok:

local5. /var/log/suricata.log*

Suricata sysloguje výstrahy pod „facility" *local5*, druhá časť zapísaného riadku obsahuje súbor (log), do ktorého budú záznamy zapisované.

8.2 Príprava ARPwatchu

ARPwatch nainštalujeme rovnako ako v podkapitole 6.1 vrátane všetkých dodatočných krokov, aby bol schopný bezproblémového chodu.

Rovnako ako pri Suricate chceme triediť správy z ARPwatchu do samostatného logu. Preto zapíšeme opäť do súboru */etc/rsyslog.d/surwatch.conf* nasledujúci riadok:

if $programname == 'arpwatch' then /var/log/arpwatch.log

V tomto prípade záznamy vyberáme podľa názvu programu, ktorý záznam vyvolal. Žiadnu ďalšiu konfiguráciu ARPwatch nevyžaduje.

8.3 Vlastná rsyslog šablóna

Po odskúšaní logovania si môžme všimnúť, že rsyslog záznamy zapisuje týmto štýlom:

Jan 01 23:45:01 suricata[1000]: správa ...

Tieto záznamy však budeme ďalej spracovávať, preto nám po prvé nevyhovuje zápis dátumu, nakoľko chcem pracovať s číslami, ktoré sa dajú omnoho ľahšie zoraďovať, a po druhé nepotrebujeme vedieť ID procesu, ktorý záznam vyvolal. Tým, že zapisujeme do samostatných logov prakticky nepotrebujeme ani samotný *„prorgamname"*, čiže názov programu, ale je vždy dobré vedieť, keď sa na hocijaký záznam niekto pozrie pred spracovaním, čo daný záznam vyvolalo. Náš template (šablóna) teda bude vyzerať nasledovne:

$template SurwatchTemplate,"%$year%-%$month%-%$day%
%timegenerated:12:19:date-rfc3339% %programname%%msg%\n"

Túto šablónu zapíšeme na začiatok súboru */etc/rsyslog.d/surwatch.conf* a na záznamy z oboch nástrojov ju použijeme nasledujúcim spôsobom:

if $programname == 'arpwatch' then /var/log/arpwatch.log;SurwatchTemplate
local5. /var/log/suricata.log;SurwatchTemplate*

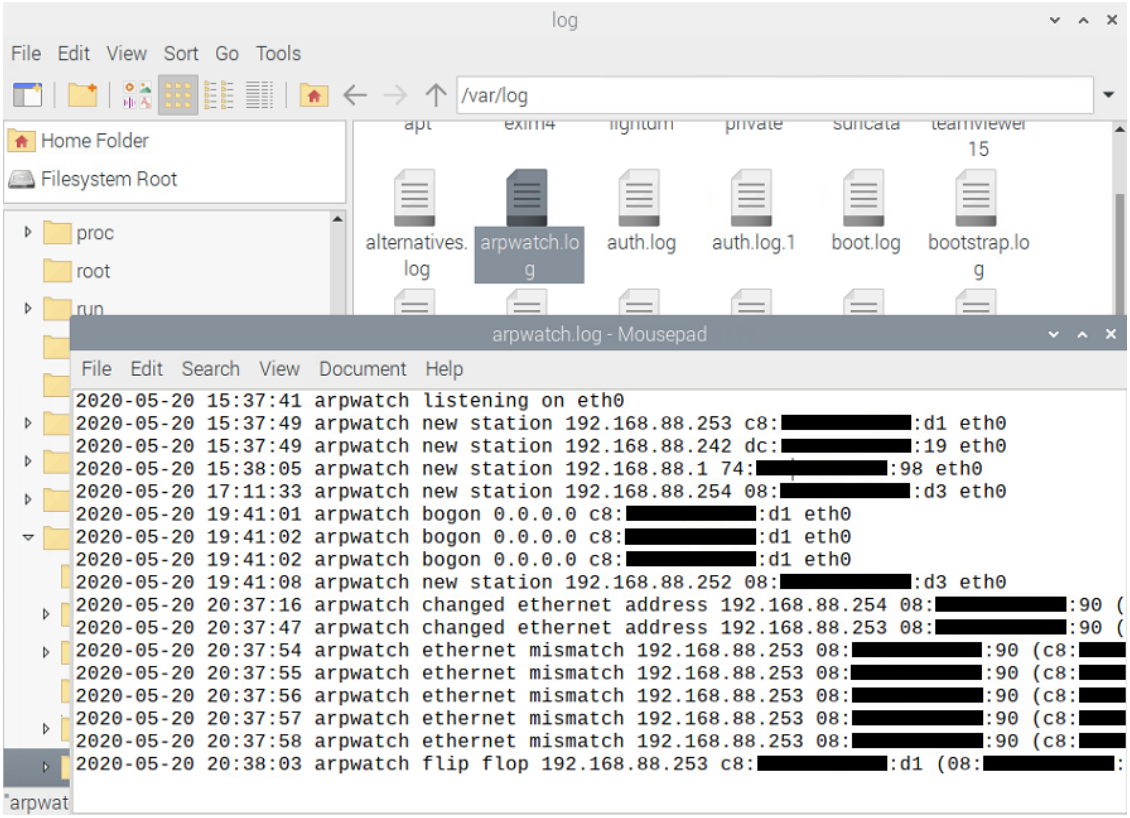
V tomto momente máme všetko prichystané, môžme teda začať s testovaním.

8.4 Testovanie systému

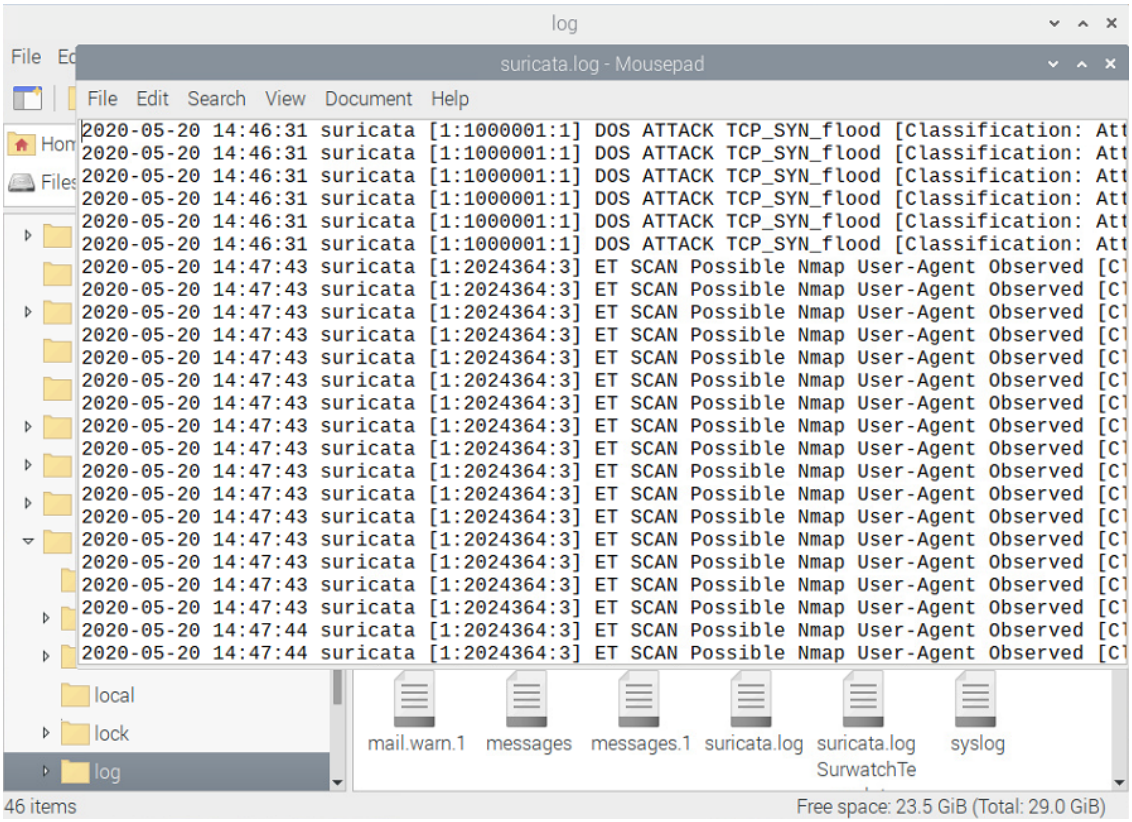
Testovanie bude prebiehať tak, že Suricatu aj ARPwatch najprv spustíme a potom budeme pomocou Kali Linuxu útočiť na klienta s IP adresou *192.168.88.253* a na router.

Útoky budú predstavovať SYN flood z podkapitoly 5.3 (do Suricaty bolo vložené aj vlastné pravidlo na detekciu tohto útoku), ARP spoofing z podkapitoly 5.4 a nakoniec využijeme nástroj Armitage (zo staršieho vydania Kali Linux 2019.1) a jeho funkciu „Hail Mary", ktorá postupne vyskúša všetky dostupné exploity pre daný systém routeru v staršej databáze Metasploit Framework. Armitage má s novými exploitmi často problémy, preto už ani nie je v nových vydaniach Kali Linuxu predvolene zahrnutý).

Po skončení útokov sa v logoch objavili záznamy o rôznych útokoch, vrátane TCP Syn flood, Nmap scanu, záznamy z anomálnej detekcie o snahe využiť známe zraniteľnosti, získať administrátorské práva, o nájdení spustiteľného kódu, atď. Ukážky záznamov v textovom editore je možné vidieť na obr. 8.1 a 8.2.



Obr. 8.1: Ukážka z arpwatch.log.



```
2020-05-20 14:46:31 suricata [1:1000001:1] DOS ATTACK TCP_SYN_flood [Classification: Att
2020-05-20 14:46:31 suricata [1:1000001:1] DOS ATTACK TCP_SYN_flood [Classification: Att
2020-05-20 14:46:31 suricata [1:1000001:1] DOS ATTACK TCP_SYN_flood [Classification: Att
2020-05-20 14:46:31 suricata [1:1000001:1] DOS ATTACK TCP_SYN_flood [Classification: Att
2020-05-20 14:46:31 suricata [1:1000001:1] DOS ATTACK TCP_SYN_flood [Classification: Att
2020-05-20 14:46:31 suricata [1:1000001:1] DOS ATTACK TCP_SYN_flood [Classification: Att
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:43 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:44 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
2020-05-20 14:47:44 suricata [1:2024364:3] ET SCAN Possible Nmap User-Agent Observed [Cl
```

Obr. 8.2: Ukážka zo suricata.log.

9 Návrh a realizácia aplikácie na spracovanie a prezentáciu logov

Aplikácia, ktorá má okrem iného disponovať grafickým užívateľským rozhraním, bude písaná v jazyku Python, nakoľko má bežať na Linuxovom operačnom systéme Raspbian. Túto bezkonkurenčnú voľbu na vývoj jednoduchej GUI aplikácie bude doplňovať balíček PyQt5 (založený na cross-platform toolkite Qt), ktorý umožní efektívne zostaviť užívateľské rozhranie.

9.1 Návrh aplikácie

Táto aplikácia, ako je už z názvu kapitoly zrejmé, má vhodne prezentovať logy. Tieto logy by mala vedieť prezentovať rovnako ako v neupravenej (raw) forme, tak aj vo forme, v ktorej si bude vedieť užívateľ zoradiť záznamy podľa jednotlivých údajov, zoskupiť záznamy, ak sú údaje viacero po sebe idúcich záznamov rovnaké, či mať možnosť ináč tieto záznamy sprehľadniť.

Ďalej by bolo vhodné mať v rámci aplikácie možnosť aspoň minoritne ovládať nástroje, poprípade v rámci nej vytvoriť možnosť vykonávania úkonov, ktorých nutnosť pravidelného opakovania sa dá očakávať. Medzi takéto úkony by mal patriť napríklad reset logov či vzhľadom k rôznorodým pravidlám Suricaty (viď 8.1.1) možnosť ich zakázania pomocou jednoduchého dialógu.

Čo sa týka Suricaty, bola by škoda nevyužiť jej údajmi nabitý *eve.json* log. Preto by bolo vhodné implementovať metódu na vyhľadávanie záznamov v rámci tohto logu (napr. pomocou kľúčového slova), pravdaže s ohľadom na výkon, nakoľko tento log môže obsahovať až gigabajty dát.

9.2 Stavba GUI

GUI je vytvorené pomocou aplikácie Qt Creator, ktorej výstupom je C++ kód zapísaný v *.ui* súbore predstavujúcom užívateľské rozhranie. Na prepísanie kódu do jazyka Python podľa štandardov balíčka PyQt5 bol využitý nástroj *pyuic5*, výstupom ktorého je súbor formátu *.py*. Tento súbor nie je vhodné nijak vlastnoručne opravovať, preto boli všetky potrebné zmeny, ktorých Qt Creator nie je schopný, vykonané priamo v kóde vlastnej aplikácie. Trieda predstavujúca užívateľské rozhranie je z tohto súboru na využitie v aplikácii importovaná.

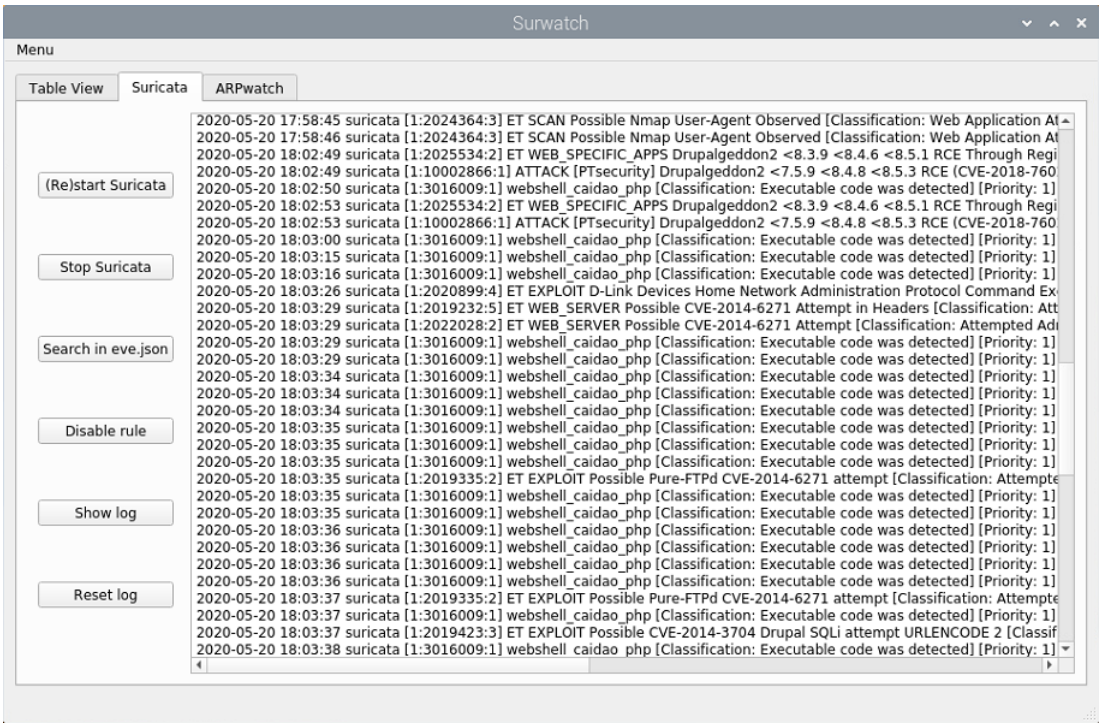
GUI jednak obsahuje menu s možnosťami a hlavnú časť pozostávajúcu z viacerých kariet, z ktorých každá zobrazuje rôznym spôsobom rôzne logy. Hlavná časť taktiež obsahuje určité ovládacie prvky daného nástroja. Poslednou časťou okna je stavový

riadok (status bar), pomocou ktorého je užívateľovi oznamovaný prípadný úspech alebo podrobnosti o neúspechu určitej operácie.

9.3 Popis možností aplikácie a jej metód

9.3.1 Karta „Suricata”

Hlavnú časť grafického rozhrania aplikácie tvorí sada troch kariet. Jedna z nich je karta s názvom „Suricata” (viď obr. 9.1), ktorá obsahuje sadu tlačidiel a pole na prezentáciu riadkov textu. Do tohto poľa je po spustení aplikácie vypísaný obsah logu *suricata.log*.



Obr. 9.1: Suricata karta aplikácie Surwatch.

Táto karta obsahuje nasledujúce tlačidlá:

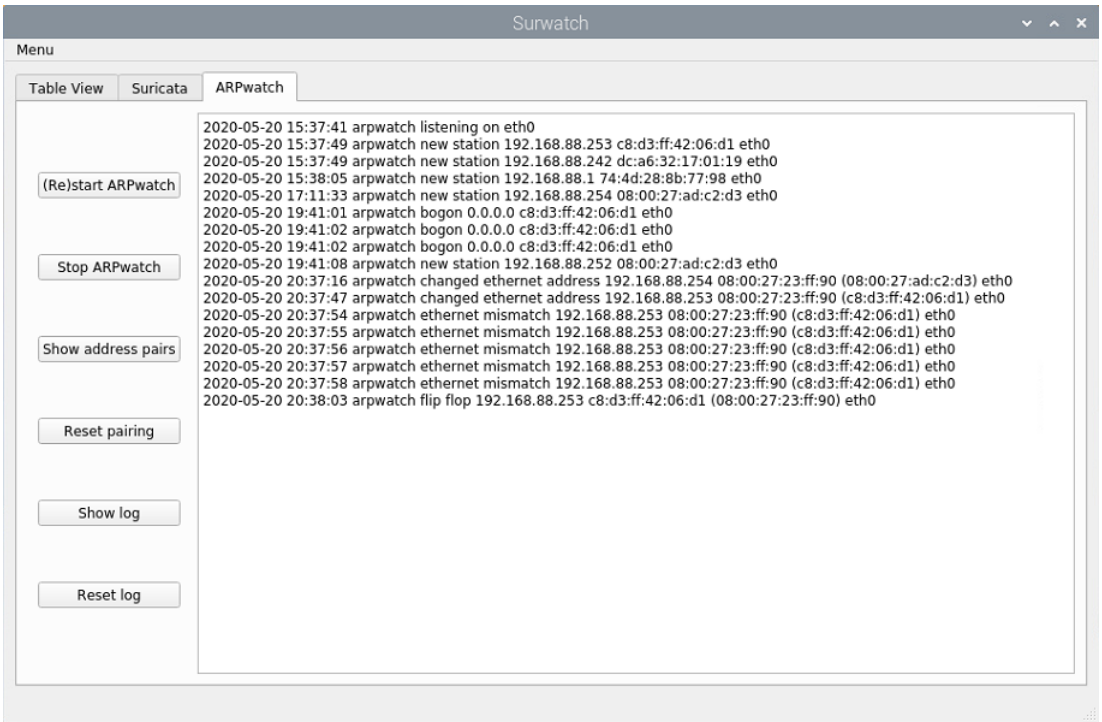
1. „(Re)start Suricata” tlačidlo – slúži na spustenie Suricaty alebo, v prípade, že už je spustená, na jej reštart.
2. „Stop Suricata” tlačidlo – zastaví Suricatu.
3. „Search in eve.json” tlačidlo – otvorí dialógové okno na zadanie hľadaného výrazu v logu *eve.json*. Po potvrdení a vyhľadaní výsledkov ich vypíše do textového poľa. V prípade, že ich je viac ako 100, zapíše prvých 100 výsledkov do poľa a všetky výsledky zapíše do súboru *search_eve.json*, na čo je užívateľ

upozornený pomocou stavového riadku. Časť kódu metódy zodpovednej za vyhľadávanie (aj s popisom) je možné vidieť v prílohe na obr. A.1.

4. „Disable rule” tlačidlo – otvorí dialógové okno na zadanie ID pravidla, ktoré má byť pridané na koniec súboru *disable.conf* a následným spustením aktualizácie pravidiel zakázané. Užívateľ môže aktualizáciu odmietnuť a manuálne pravidlá aktualizovať neskôr.
5. „Show log” tlačidlo – načíta aktuálne záznamy zo *suricata.log* a zobrazí ich v textovom poli.
6. „Reset log” tlačidlo – pomocou dialógového okna overí záujem užívateľa o reset logu a po potvrdení vymaže a znovu vytvorí *suricata.log* a reštartuje *rsyslog*.

9.3.2 Karta „ARPwatch”

Ďalšia karta nesie názov „ARPwatch” (viď obr. 9.2) a taktiež obsahuje sadu tlačidiel a pole na prezentáciu riadkov textu. Do tohto poľa je po spustení aplikácie vypísaný obsah logu *arpwatch.log*.



Obr. 9.2: ARPwatch karta aplikácie Surwatch.

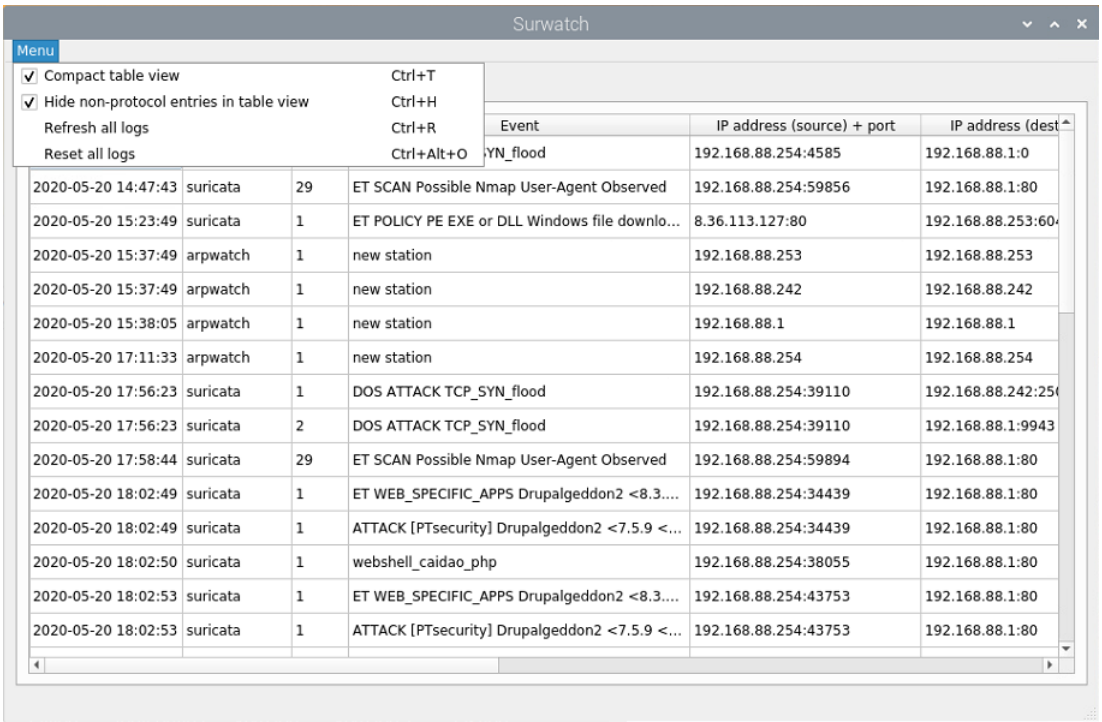
Táto karta obsahuje nasledujúce tlačidlá:

1. „(Re)start ARPwatch” tlačidlo – slúži na spustenie ARPwatchu alebo, v prípade, že už je spustený, na jeho reštart
2. „Stop ARPwatch” tlačidlo – zastaví ARPwatch

3. „Show address pairs” tlačidlo – do textového poľa zobrazí zaznamenané IP/ethernet adresné páry vrátane hlavičky s popisom údajov.
4. „Reset pairing” tlačidlo – pomocou dialógového okna overí záujem užívateľa o reset adresného párovania a po potvrdení zastaví ARPwatch (ak je spustený), vymaže a znovu vytvorí *arpdat.log* a prípadne ARPwatch znova spustí.
5. „Show log” tlačidlo – načíta aktuálne záznamy z *arpwatch.log* a zobrazí ich v textovom poli.
6. „Reset log” tlačidlo – po potvrdení v rámci dialógového okna vymaže a znovu vytvorí *arpwatch.log* a reštartuje *rsyslog*.

9.3.3 Karta „Table View” a menu

Najpodstatnejšou kartou aplikácie je karta s názvom „Table View” (viď obr. 9.3), ktorý jej bol udelený z dôvodu, že zobrazuje záznamy od Suricaty aj ARPwatchu po riadkoch v jednej tabuľke, pričom každý údaj zo záznamu má pridelený vlastný stĺpec. Časť kódu metódy spracúvajúcej záznamy zo *suricata.log* je možné vidieť v prílohe na obr. A.2.



Obr. 9.3: Table View karta a menu aplikácie Surwatch.

V rámci tabuľky je možné záznamy podľa jednotlivých údajov zoradiť a stĺpce podľa vlastnej vôle preusporiadať. Taktiež je možné v menu zaškrtnúť možnosť „Compact table view”, čím sa z každej skupiny po sebe chronologicky idúcich riadkov

obsahujúcich rovnaké údaje (okrem času) zapíše len prvý (s najstarším záznamom). Naviac je pridaný nový stĺpec s názvom „Count”, ktorý obsahuje počet zlúčených riadkov v každej skupine. Časť kódu z metódy obstarávajúcej túto možnosť je možné vidieť v prílohe na obr. A.3.

Ďalšou možnosťou, ktorú je možné aktivovať zaškrtnutím „Hide non-protocol entries in table view”, je skrytie všetkých riadkov neobsahujúcich relevantné údaje pre detekciu útokov, čo je v rámci tabuľky indikované prázdnym políčkom so skratkou protokolu. Jedná sa najmä o správy z ARPwatchu o jeho spustení, zastavení či o chybové hlásenia.

Menu okrem toho obsahuje ďalšie dve možnosti. „Refresh all logs” vykoná obnovenie všetkých troch kariet, pričom v kartách „Suricata” a „ARPwatch” sa v textovom poli zobrazia aktuálne záznamy s príslušných logov a tabuľka sa taktiež aktualizuje. „Reset all logs” resetuje logy rovnakým spôsobom ako pri tlačidlách v kartách oboch nástrojov. Užívateľ je ako v prípade tejto možnosti, tak aj v prípade oboch tlačidiel na reset logov, upozornený, že logy budú po obnovení natrvalo stratené (pred obnovením si ich vie stále pozrieť či skopírovať z textového poľa).

Záver

Táto práca sa teoreticky aj prakticky zaoberal detekciou kybernetických útokov v lokálnych sieťach. Hlavnou náplňou teoretickej časti bol popis a rozdelenie systémov na detekciu prienikov. Ďalej boli spomenuté open-source nástroje, ktoré každý svojim spôsobom slúžia na detekciu a prípadnú prevenciu pred útokmi a škodlivým prenosom v lokálnej sieti. Značná časť sa zameriava na kategorizáciu metód detekcie útokov. Bola venovaná jednak všeobecne najrozšírenejšej metóde založenej na detekcii signatúr a jej nedostatkom, ale aj metódami detekcie anomálií. Samostatne bola taktiež spomenutá metóda stavovej analýzy protokolov, ktorá by z pohľadu využívaných techník mohla byť (a často aj je) radená pod metódy detekcie anomálií.

Praktická časť bola zo začiatku venovaná nástroju Suricata IDS a jeho testovaniu na jednodoskovom počítači Raspberry Pi. Suricata sa javila ako skutočne výkonný a pomerne užívateľsky prívetivý nástroj. Nejde však o nástroj, ktorý by bol z pohľadu detekčných pravidiel pravidelne spravovaný aktualizáciami a teda ktorý by bol schopný bez väčšieho zásahu autonómne pracovať. Tento nástroj skôr patrí do rúk odborníkov, ktorý jednak vedia na čo a kde ho chcú využiť a podľa toho si ho sami nakonfigurujú a priebežne spravujú.

Nakoľko Suricata nedisponuje možnosťami na detekciu útokov protokolu ARP, bola jej schopnosť detekcie rozšírená o nástroj ARPwatch. ARPwatch sleduje zo svojej podstaty nebezpečný ARP protokol často využívaný pri útokoch v lokálnych sieťach. Z tohto dôvodu je ARPwatch, ktorý je schopný zaznamenať zmeny v adresných párach či akékoľvek zvláštne správanie tohto protokolu, veľkým prínosom do systému detekcie, ktorého návrhom a praktickým overením sa zaoberali dve kapitoly tejto práce.

Tento systém nám umožnil aj vďaka prepracovaným sadám pravidiel Suricaty detegovať množstvo rôznych útokov a zabezpečiť logy o škodlivej či potenciálne škodlivej aktivite. Pravdaže ani zďaleka nedokáže odhaliť každú hrozbu číhajúcu na sieti, avšak potenciál, aby sa k tomu aspoň priblížil, jednoznačne má. Jediným problémom je veľký nedostatok potrebných pravidiel a následne aj výpočetnej kapacity.

Nakoniec bola vytvorená GUI aplikácia, ktorá spomenuté logy spracúva a prezentuje užívateľovi v prehľadnej tabuľkovej forme. Aplikácia okrem toho disponuje možnosťami zredukovať počet záznamov podľa ich opakujúceho sa či nepodstatného obsahu, vyhľadávaním v objemnom hlavnom logu Suricaty, ale aj rôznymi ovládacími prvkami pre oba využité nástroje.

Literatúra

[1] COMMITTEE ON NATIONAL SECURITY SYSTEMS: *CNSS Instruction No. 4009: National Information Assurance (IA) Glossary [online]. 2010. [cit. 15. 12. 2019].* Dostupné z: <https://www.hsdl.org/?abstract&did=7447/>

[2] ZUZČÁK, Matej: *Sieťové riziká a útoky na lokálnej sieti. SecIT.sk [online]. 2010. ISSN 2464-5354. [cit. 15. 12. 2019].* Dostupné z: <https://secit.sk/sk/content/sietove-rizika-utoky-na-lokalnej-sieti>

[3] PFLEEGER, Charles P. a Shari Lawrence PFLEEGER: *Analyzing computer security: a threat/vulnerability/countermeasure approach. Second printing. Upper Saddle River, NJ: Prentice Hall, c2012. ISBN 9780132789462.*

[4] SCARFONE, Karen A. a Peter M. MELL: *Guide to Intrusion Detection and Prevention Systems (IDPS) [online]. 2007. Special Publication (NIST SP) – 800-94. [cit. 15. 12. 2019].* Dostupné z: <https://www.nist.gov/publications/guide-intrusion-detection-and-prevention-systems-idps>

[5] EINWECHTER, Nathan: *An Introduction To Distributed Intrusion Detection Systems. Symantec Connect [online]. 07 Jan 2002. [cit. 15. 12. 2019].* Dostupné z: <https://www.symantec.com/connect/articles/introduction-distributed-intrusion-detection-systems>

[6] *Ossec for IPFire - forum.ipfire.org [online]. [cit. 25. 5. 2020].* Dostupné z: <https://forum.ipfire.org/viewtopic.php?t=15597>

[7] *IDS / IPS - Configuring the Snort Package | pfSense Documentation [online]. [cit. 25. 5. 2020].* Dostupné z: <https://docs.netgate.com/pfsense/en/latest/ids-ips/setup-snort-package.html>

[8] *robcowart/synesis_lite_suricata: Suricata IDS/IPS log analytics using the Elastic Stack [online]. [cit. 25. 5. 2020].* Dostupné z: <https://github.com/robcowart/synesis_lite_suricata>

[9] *Corelight - About Zeek - How Zeek / Bro Works [online]. [cit. 25. 5. 2020].* Dostupné z: <https://corelight.com/about-zeek/>

[10] *[SAMHAIN v3.0.11 & BELTANE v2.4.6] Host-based intrusion detection system (HIDS) [online]. [cit. 25. 5. 2020].* Dostupné z: <https://www.kitploit.com/2013/04/samhain-v3011-beltane-v246-host-based.html>

[11] *Fail2ban - Wikipedia [online]. [cit. 25. 5. 2020].* Dostupné z: <https://en.wikipedia.org/wiki/Fail2ban>

[12] *Sagan v1.2: analyze logs (syslog/event log/snmptrap/netflow/etc) [online]. [cit. 25. 5. 2020].* Dostupné z: <https://securityonline.info/sagan-analyze-logs/>

[13] BROX, Arnt: *Signature-Based or Anomaly-Based Intrusion Detection: The Practice and Pitfalls. SC Media [online]. May 1, 2002. [cit. 15. 12. 2019].* Dostupné z: <https://www.scmagazine.com/home/security-news/features/signature-based-or-anomaly-based-intrusion-detection-the-practice-and-pitfalls/>

[14] SADEK, Rowayda A., M. Sami SOLIMAN a Hagar S. ELSAYED: *Effective Anomaly Intrusion Detection System based on Neural Network with Indicator Variable and Rough set Reduction. IJCSI International Journal of Computer Science Issues [online]. 2013, 10(6, No 2). ISSN 1694-0784. [cit. 15. 12. 2019].* Dostupné z: <https://www.ijcsi.org/papers/IJCSI-10-6-2-227-233.pdf>

[15] BEETLE & SASHA *A strict anomaly detection model for IDS. Phrack Magazine [online]. 05.01.2000, 0xa Issue 0x38(0x0b[0x10]) [cit. 15. 12. 2019].* Dostupné z: <http://phrack.org/issues/56/11.html>

[16] GARCÍA-TEODORO, P., J. DÍAZ-VERDEJO, G. MACIÁ-FERNÁNDEZ a E. VÁZQUEZ: *Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security [online]. 2009, 28. DOI: 10.1016/j.cose.2008.08.003. ISSN 01674048. [cit. 15. 12. 2019].* Dostupné z: <https://linkinghub.elsevier.com/retrieve/pii/S0167404808000692>

[17] WHITMAN, Michael E. a Herbert J. MATTORD: *Principles of Information Security. 3rd ed. Boston: Thomson Course Technology, [2008]. ISBN 1-4239-0177-0.*

[18] SHENFIELD, Alex, David DAY a Aladdin AYESH: *Intelligent intrusion detection systems using artificial neural networks. ICT Express. 2018, 4(2), 95-99.*

DOI: 10.1016/j.icte.2018.04.003. ISSN 24059595. [cit. 25. 5. 2020]. Dostupné z:
<https://linkinghub.elsevier.com/retrieve/pii/S2405959518300493>

[19] TADDEO, Mariarosaria, Tom MCCUTCHEON a Luciano FLORIDI: *Trusting artificial intelligence in cybersecurity is a double-edged sword. Nature Machine Intelligence. 2019, 1(12), 557-560. DOI: 10.1038/s42256-019-0109-1. ISSN 2522-5839.. [cit. 25. 5. 2020].* Dostupné z:
<https://www.nature.com/articles/s42256-019-0109-1>

[20] LARUE-LANGLOIS, Renaud *Top 10 Intrusion Detection Tools: Your Best Free Options for 2019. AddictiveTips [online]. Aug 7, 2018. [cit. 15. 12. 2019].* Dostupné z:
<https://www.addictivetips.com/net-admin/intrusion-detection-tools/>

[21] *Suricata – All features. Suricata – Open Source IDS / IPS / NSM engine [online]. [cit. 15. 12. 2019].* Dostupné z:
<https://suricata-ids.org/features/all-features/>

[22] *Suricata – Docs. Suricata User Guide – Suricata unknown documentation [online]. [cit. 15. 12. 2019].* Dostupné z:
<https://suricata.readthedocs.io/en/suricata-5.0.0/>

[23] *MikroTik Routers and Wireless – Products: hAP ac². MikroTik Routers and Wireless [online]. [cit. 15. 12. 2019].* Dostupné z:
<https://mikrotik.com/product/hap_ac2>

[24] *Raspberry Pi 2 Model B. Teach, Learn, and Make with Raspberry Pi – Raspberry Pi [online]. [cit. 15. 12. 2019].* Dostupné z:
<https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>

[25] *Raspberry Pi 4 Tech Specs. Teach, Learn, and Make with Raspberry Pi – Raspberry Pi [online]. [cit. 15. 12. 2019].* Dostupné z:
<https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>

[26] *Debian Installation – Suricata – Open Information Security Foundation. Open Information Security Foundation [online]. [cit. 15. 12. 2019].* Dostupné z:
<https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Debian_Installation>

[27] *Non-Payload Detection Rule Options [online]. [cit. 15. 12. 2019].* Dostupné z: <http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node33.html>

[28] *Ubuntu Manpage: arpwatch [online]. [cit. 25. 5. 2020].* Dostupné z: <http://manpages.ubuntu.com/manpages/focal/en/man8/arpwatch.8.html>

[29] *GitHub – suricata-rules/suricata-rules: Suricata IDS rules [online]. [cit. 25. 5. 2020].* Dostupné z: <https://github.com/suricata-rules/suricata-rules/>

Zoznam symbolov, veličín a skratiek

AI Artificial Intelligence

ANN Artificial Neuron Network

ARP Address Resolution Protocol

CAM Content Addressable Memory

CNSS Committee on National Security Systems

CNSSI Committee on National Security Systems Instruction

CPU Central Processing Unit

DDoS Distributed Denial of Service

DHCP Dynamic Host Configuration Protocol

DIDS Distributed Intrusion Detection System

DLL Dynamic-Link Library

DNS Domain Name System

DoS Denial of Service

FTP File Transfer Protocol

GUI Graphical User Interface

HIDS Host-based Intrusion Detection System

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IDS Intrusion Detection System

IP Internet Protocol

IPS Intrusion Prevention System

JSON JavaScript Object Notation

MAC Media Access Control

MTU Maximum Transmission Unit

MITM	Man In The Middle
NAC	Network Access Control
NIDS	Network-based Intrusion Detection System
NSM	Network Security Monitoring
OS	Operating System
OSI	Open Systems Interconnection
RPi	Raspberry Pi
SIEM	Security Information and Event Management
SSH	Secure Shell
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UI	User Interface
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
YAML	YAML Ain't Markup Language

A Ukážky kódu aplikácie s komentármi

```
text, ok = QInputDialog.getText(self.ui, "Search in eve.json", "Search for:",
                                QLineEdit.Normal, "e.g. 2020-05-20T18:04:14")
try:
    if ok and text:
        self.ui.statusLabel.setText("Searching...")
        self.ui.repaint()
        counter = 0
        too_many = False
        with open(".temp_search", "w") as temp_search_file:
            with open("/var/log/suricata/eve.json") as eve:
                for line in eve:
                    if text in line:
                        counter += 1
                        if counter == 101: # Zapíš max. 100 výsledkov
                            vyhľadávania do textBrowser, ak ich je viac, zapíš
                            všetky do súboru search_eve.json.
                            too_many = True
                            temp_search_file.close()
                            temp_search_file = open("search_eve.json", "w")
                                # Odteraz zapisuj všetky riadky do súboru
                                search_eve.json.
                            with open(".temp_search", "r") as temp:
                                # Tiež tam zapíš doterajšie výsledky.
                                for l in temp:
                                    temp_search_file.write(l)
                        temp_search_file.write(line)
        cat = subprocess.Popen(["cat", ".temp_search"], stdout=subprocess.PIPE)
            # Prevedie do pipy všetok text zo súboru s výsledkami vyhľadávania.
        jq_output = subprocess.check_output(["jq", "."], stdin=cat.stdout)
            # Nechá jq transformovať text z pipy do prezentovacej formy.
```

Obr. A.1: Časť metódy na vyhľadávanie v eve.json.

```
with open("/var/log/suricata.log") as f:
    surLogLines = [x.strip() for x in f.readlines()]
for line in surLogLines:
    columns = ["" for x in range(10)]
    columns[0] = line[:19] # Dátum a čas
    columns[1] = "suricata" # Názov nástroja
    splitted = line[29:].split()
    if splitted[0].count(":") == 2:
        columns[7] = splitted.pop(0).split(":")[1] # SID
    else:
        # Ak záznam neobsahuje ID príznaku, potom zapíš všetko do premennej
        Popis udalosti a pokračuj v ďalšej iterácii.
        columns[2] = line[29:]
        self.table.append(columns)
        columns.clear()
        splitted.clear()
        continue
    event = ""
    while splitted[0] != "[Classification:":
        event += splitted.pop(0) + " "
    columns[2] = event.strip() # Popis udalosti
    splitted.pop(0)
    classification = ""
    while splitted[0][-1] != "]":
        classification += splitted.pop(0) + " "
    else:
        classification += splitted.pop(0)[:-1]
    columns[9] = classification # Klasifikácia
    splitted.pop(0)
    columns[8] = splitted.pop(0)[:-1] # Priorita
    columns[6] = splitted.pop(0)[1:-1] # Protokol
    columns[3] = splitted.pop(0) # IP adresa (zdroj) + port
    splitted.pop(0)
    columns[4] = splitted.pop(0) # IP adresa (cieľ) + port
    self.table.append(columns.copy()) # self.table je globálna premenná
        uchovávajúca celú tabuľku, z nej sa údaje načítavajú do tabuľky v GUI
    columns.clear()
    splitted.clear()
```

Obr. A.2: Časť metódy spracúvajúcej záznamy z logu Suricaty.

```
counter = []
for column in range(10):
    self.ui.mainTableWidget.setItem(0, column, QTableWidgetItem(self
        .table[0][column])) # Zapíš prvý záznam.
counter.append(1)
lastWrittenRow = 0
match = True
for row in range(1, len(self.table)):
    for column in range(1, 10): # Cyklus na porovnávanie všetkých údajov
        záznamu s posledným zapísaným záznamom (okrem času).
        if self.table[row][column] != self.table[lastWrittenRow][column]:
            match = False
            if (column == 3 or column == 4):
                if self.table[row][column].split(":")[0] == self
                    .table[lastWrittenRow][column].split(":")[0]:
                    # Ignoruej port v stĺpcoch s IP adresou.
                    match = True
            if match == False:
                break
    if match: # Inkrementuj posledné číslo v counter ak sú záznamy rovnaké.
        counter[self.ui.mainTableWidget.rowCount()-1] +=1
    else: # Ináč pridaj číslo (1) do counter a zapíš záznam do nového riadku.
        counter.append(1)
        self.ui.mainTableWidget.insertRow(self.ui.mainTableWidget.rowCount());
        for column in range(10):
            self.ui.mainTableWidget.setItem(self.ui.mainTableWidget.rowCount
                ()-1,
                column, QTableWidgetItem(self.table[row][column]))
            lastWrittenRow = row
        match = True # Obnov match späť na True pre ďalšiu iteráciu.
```

Obr. A.3: Časť metódy obstarávajúcej možnosť Compact table view.