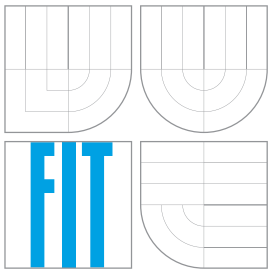
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AGREGACE HLÁŠENÍ O BEZPEČNOSTNÍCH UDÁLOSTECH

AGGREGATION OF SECURITY INCIDENT REPORTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DANIEL KAPIČÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÁCLAV BARTOŠ

BRNO 2016

Abstrakt

V tejto práci sa venujem analýze bezpečnostných hlásení zo systému Mentat vo formáte IDEA a návrhu a implementácii metód pre ich agregáciu a koreláciu. V analýze dát poukážem na rozmanitosť hlásení. Ďalej predstavím návrh jednoduchého frameworku a systému šablón, ktorý slúži ako základ pre tvorbu agregačných a korelačných metód. Tiež popíšem návrh a implementáciu jednotlivých metód. Na záver vyhodnotím niektoré navrhnuté metódy na dostupných vzorkách dát. Ukážem, že navrhnuté agregačné metódy v niektorých prípadoch dokážu znížiť množstvo hlásení až o 90% pri zachovaní všetkých podstatných informácií.

Abstract

In this thesis, I present analysis of security incident reports in IDEA format from Mentat and their aggregation and correlation methods design and implementation. In data analysis, I show huge security reports diversity. Next, I show design of simple framework and system of templates. This framework and system of templates simplify aggregation and correlation methods design and implementation. Finally, I evaluate designed methods using Mentat database dumps. The results showed that designed methods can reduce the number of security reports up to 90% without loss of any significant information.

Kľúčové slová

agregácia hlásení, agregácia hlásení o bezpečnostných udalostiach, korelácia hlásení, korelácia hlásení o bezpečnostnýchch udalostiach, mentat, warden, idea, sec

Keywords

aggregation of reports, aggregation of security incident reports, correlation of reports, correlation of security incident reports, mentat, warden, idea, sec

Citácia

KAPIČÁK, Daniel. *Agregace hlášení o bezpečnostních událostech*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bartoš Václav.

Agregace hlášení o bezpečnostních událostech

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Václava Bartoša. Ďalšie informácie mi poskytol pán Ing. Pavel Kácha. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Daniel Kapičák
23. mája 2016

Poďakovanie

Ďakujem Ing. Václavovi Bartošovi a Ing. Pavlovi Káchovi za odbornú pomoc a veľkú ochotu.

Obsah

1 Úvod **3**

2 Incidenty v počítačových sieťach **4**
2.1 Skenovanie portov . . . 4
2.1.1 Princíp . . . 5
2.2 Spam . . . 5
2.3 Viacnásobný pokus o prihlásenie . . . 5
2.4 Porušovanie autorských práv . . . 5
2.5 Vulnerable . . . 5
2.6 Denial of Service . . . 5
2.6.1 Reflektívny amplifikačný útok . . . 6
2.7 Malware . . . 6
2.7.1 Počítačový vírus . . . 7
2.7.2 Počítačový červ . . . 7
2.7.3 Trojský kôň . . . 7
2.7.4 Spyware . . . 7
2.7.5 Dialer . . . 7
2.7.6 Rootkit . . . 7
2.8 Botnet . . . 7

3 Detekcia nežiadúceho obsahu v počítačových sieťach **9**
3.1 NetFlow/IPFIX . . . 9
3.2 Honeypot . . . 9
3.2.1 Systémy honeypot s nízkou mierou interakcie . . . 10
3.2.2 Systémy honeypot s vysokou mierou interakcie . . . 10
3.3 Intrusion Detection System . . . 10
3.3.1 Host-based intrusion detection system . . . 10
3.3.2 Network intrusion detection system . . . 10
3.4 Security Information and Event Management . . . 10

4 Mentat **11**

5 Warden **13**
5.1 Architektúra . . . 13
5.1.1 Warden klienti . . . 13
5.1.2 Warden udalosť . . . 14

6 Intrusion Detection Extensible Alert **15**
6.1 Popis . . . 15
6.1.1 Základná štruktúra . . . 15
6.1.2 Definícia . . . 15
6.2 Klasifikácia . . . 17
6.3 Príklady . . . 18

7 Analýza dát **20**
7.1 Recon.Scanning . . . 23
7.2 Availability.DoS a Availability.DDoS . . . 23
7.3 Attempt.Login . . . 24
7.4 Attempt.Exploit . . . 24
7.5 Fraud.Copyright . . . 24
7.6 Vulnerable.Config . . . 25
7.7 Intrusion.Botnet . . . 25
7.8 Abusive.Spam . . . 25

8 Návrh a implementácia **26**
8.1 Vytváranie a pridávanie udalostí do kontextov . . . 27
8.1.1 Šablóna . . . 27
8.2 Agregácia hlásení Recon.Scanning . . . 29
8.3 Agregácia hlásení Availability.DoS a Availability.DDoS . . . 31
8.4 Agregácia hlásení Attempt.Login . . . 33
8.5 Agregácia hlásení Attempt.Exploit . . . 34
8.6 Agregácia hlásení Fraud.Copyright . . . 35
8.7 Agregácia hlásení Vulnerable.Config . . . 36
8.8 Agregácia hlásení Intrusion.Botnet . . . 37
8.9 Agregácia hlásení Abusive.Spam . . . 38
8.10 Korelácia hláseni Recon.Scanning a Attempt.Login . . . 39
8.11 Korelácia hláseni Recon.Scanning a Attempt.Exploit . . . 42

9 Vyhodnotenie **44**
9.1 Vplyv veľkosti agregačného okna na veľkosť databázy . . . 44
9.1.1 Recon.Scanning . . . 44
9.1.2 Fraud.Copyright . . . 46
9.1.3 Attempt.Login . . . 48
9.2 Spotreba pamäti . . . 50
9.2.1 Recon.Scanning . . . 51
9.2.2 Fraud.Copyright . . . 52
9.2.3 Attempt.Login . . . 52
9.3 Rýchlosť . . . 53

10 Záver **54**

Literatúra **55**

Prílohy **56**
Zoznam príloh . . . 57

A Obsah CD **58**

Kapitola 1

Úvod

Každým dňom sú celé organizácie a dokonca aj samotní ľudia čoraz viac závislí na spoľahlivosti a bezpečnosti internetového pripojenia. S narastajúcim dopytom po spoľahlivosti a bezpečnosti rastie potreba eliminácie nežiadúceho obsahu v počítačových sieťach. Ako nežiadúci obsah môžeme označiť akýkoľvek útok alebo neštandardné získavanie informácií o počítačovej sieti a systémoch v nej pripojených. Odpoveďou na tieto problémy sú systémy ako Intrusion Detection System (IDS), honeypot, rôzne firewally a pod. inštalované v počítačových sieťach, ktoré dokážu vyhodnocovať sieťovú prevádzku a v prípade detekcie podozrivej komunikácie vyvolať určitú akciu. Napríklad vygenerovať hlásenie o incidente. Postupom času nasadzovaním ďalších detekčných systémov do rôznych miest v počítačových sieťach alebo nasadzovaním rôznych typov systémov s odlišnou politikou vyhodnocovania sieťovej prevádzky vznikaju buď duplicitné hlásenia, alebo hlásenia, ktoré nesú odlišné informácie, no stále hlásenia o tom istom incidente. Pri dlhších alebo zložitejších útokoch skladajúcich sa z viacerých fáz vzniká hlásení niekoľko. Tu prichádza na rad systém označovaný ako Security Information and Event Management (SIEM). SIEM na základe hlásení získaných z viacerých IDS, smerovačov, systémov honeypot a pod. dokáže tieto hlásenia agregovať a v ideálnom prípade korelovať. To výrazným spôsobom redukuje množstvo hlásení, čo vedie jak k úspore miesta v databáze, tak k sprehladneniu množstva hlásení. Nevyhnutnosť korelácie je zrejmá. Pri veľmi veľkom objeme hlásení je dnes nemožné pre človeka hľadať v hláseniach všetky súvislosti a tým identifikovať napríklad všetky fázy zložitejších útokov. SIEM na základe tejto analýzy dokáže vyvolať rôzne akcie. Jeho výhodou je najmä pre človeka prehľadnejší výstup. Jedným z takýchto systémov je systém Mentat [4].

Kapitola 2

Incidenty v počítačových sieťach

Za incident v počítačových sieťach sa dá považovať akýkoľvek útok alebo iný nežiadúci obsah. Útok v počítačovej sieti môžeme klasifikovať ako akýkoľvek pokus odhaliť, zmeniť, obmedziť alebo získať neautorizovaný prístup k zdrojom v počítačovej sieti. Útoky sú vo väčšine prípadov vykonávané niekým so zlými úmyslami. Do tejto kategórie spadajú tzv. Black Hats [3]. Ďalšou kategóriou sú White Hats [3] a Gray hats [3]. Do týchto dvoch kategórii spadajú ľudia, ktorí útoky nevykonávajú so zlými úmyslami. Väčšinou sú vykonávané za účelom testovania, zvyšovania bezpečnosti alebo za inými obrannými účelmi. Do kategórie Gray Hats spadajú tí, ktorí nemajú na útok povolenie druhej strany. Útoky možno klasifikovať podľa ich pôvodu, t.j. útok z jedného alebo z viacerých zdrojov. V druhom prípade sa útok označuje ako distribuovaný. K realizácii distribuovaného útoku sa využíva botnet. Útoky taktiež možno rozdeliť podľa postupu pri útoku alebo podľa typu zneužitých slabých miest. Útoky môžu byť fyzické, t.j. poškodenie alebo odcuzdenie fyzických častí počítačovej siete a jej súčastí. Ostatné útoky sú útoky logické. Tieto útoky majú za cieľ zmeniť logiku počítačov alebo protokolov využívaných v počítačovej sieti. Takéto zmeny sú výhodné pre útočníka, no pre používateľov znamenajú nepredvídané správanie. Software určený na vykonávanie logických útokov sa označuje ako malware.

Ako nežiadúci obsah v počítačových sieťach tiež možno označiť akúkoľvek aktivitu tretej strany, ktorá vedie k získaniu informácií o stave počítačovej siete a o stave zariadení, ktoré sú k nej pripojené. Takéto aktivity sú typicky vykonávané útočníkmi ako prípravná fáza pred zahájením útoku. Počas prípravnéj fázy útočník zistuje čo najviac informácií o sieti a zariadeniach na nej, typicky topológiu siete, služby, ktoré sú spustené na zariadeniach v sieti a iné.

V nasledujúcich podkapitolách popíšem najčastejšie typy incidentov zachytených systémom Mentat.

2.1 Skenovanie portov

Skenovanie portov [1] je spôsob zisťovania otvorených sieťových portov na systémoch v počítačovej sieti. Vďaka informácii o otvorených portoch je tak možné nájsť služby, ktoré sú na jednotlivých systémoch spustené. Skenovanie portov je v počítačovej sieti považované za nežiadúci obsah, pretože informácie získané touto technikou sa často použijú na odhalenie zranitelných miest systémov v počítačovej sieti.

2.1.1 Princíp

Služby v počítačových sieťach fungujú typicky na princípe klient-server. Aby bolo možné pripojiť sa k serveru, server musí neustále počúvať na konkrétnom sieťovom porte a odpovedať na pokusy o komunikáciu od klientov. Komunikácia medzi klientom a serverom prebieha väčšinou pomocou protokolov TCP alebo UDP. V obidvoch prípadoch je možné vytvoriť fiktívneho klienta, ktorý je schopný predstierať, že má o službu záujem a tak si vyžiadať reakciu servera. Na základe reakcie od servera sa dá jednoducho rozhodnúť o prítomnosti danej služby, ktorej príslušný port skúmame. Z dôvodu komplexnosti rodiny protokolov TCP/IP nie je ich implementácia na všetkých systémoch úplne totožná. Vďaka týmto drobným odlišnostiam je možné od seba rozoznať dva rôzne operačné systémy a dokonca v niektorých prípadoch aj verzie týchto operačných systémov.

2.2 Spam

Spam je v terminológii počítačových sietí nevyžiadaná hromadne posielaná správa prakticky rovnakého obsahu. Je využívaný zväčša na šírenie reklamy. Správy sú šírené najmä prostredníctvom e-mailu, ale sú tiež šírené napríklad prostredníctvom instant messaging.

2.3 Viacnásobný pokus o prihlásenie

Viacnásobný pokus o prihlásenie naznačuje pokus o zisťovanie alebo hádanie hesiel. Pre uhádnutie hesla je často využívaný útok hrubou silou [5]. V kryptografii to je pokus o rozlúštenie šifry bez znalosti kľúča. V praxi ide o tzv. slovníkový útok, teda skúšanie určitej obmedzenej množiny kombinácií. Tomuto útoku niekedy predchádza skenovanie portov, pomocou ktorého útočník odhalí potencionálne zranitelné miesta týmto typom útoku.

2.4 Porušovanie autorských práv

Tento typ nežiadúceho obsahu vzniká šírením komerčného software alebo iných materiálov chránených autorským právom bez platnej licencie. Hlásenia tejto kategórie v systéme Mentat pochádzajú najmä zo sledovania siete bittorrent.

2.5 Vulnerable

Termín Vulnerable (zraniteľnosť) neoznačuje priamo bezpečnostný incident ako taký. Označuje systémy, ktoré sú napadnuteľné nejakou známou zraniteľnostou, prípadne sa dajú zneužiť k vykonaniu útoku treťou stranou.

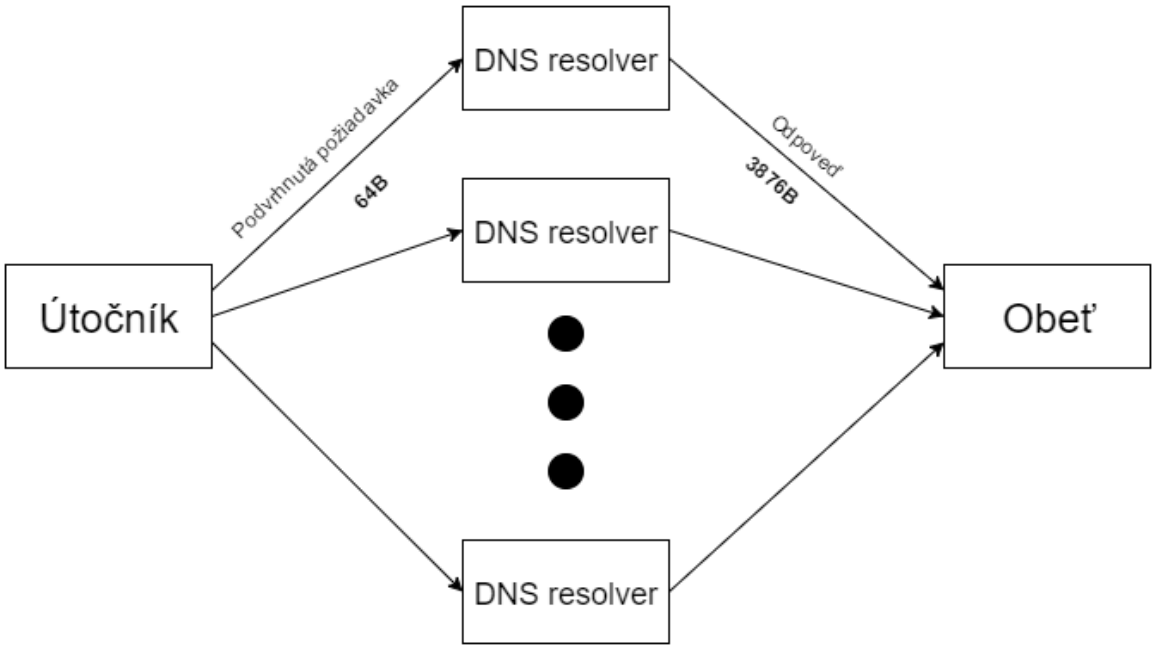
2.6 Denial of Service

Denial of Service (DoS) alebo Distributed Denial of Service (DDoS) je technika útoku na systém alebo systémy v počítačovej sieti, ktorá má za cieľ znížiť alebo úplne obmedziť dostupnosť. DDoS útoky sú vykonávané dvoma alebo viacerými osobami alebo robotmi. DoS útoky sú vykonávané jednou osobou alebo jedným systémom. Jeden z najčastejších typov útoku umožňuje zahltenie cieľového systému alebo systémov falošnými požiadavkami, a to

natoľko, že systém, prípadne systémy nie sú schopné odpovedať na legitímne požiadavky alebo reagujú tak pomaly, že sa javia ako nedostupné.

2.6.1 Reflektívny amplifikačný útok

Amplikfikačné útoky sú rozšírenou formou DDoS útoku. Útočník pomocou DNS resolverov alebo NTP serverov voľne dostupných na internete zosiluje prevádzku smerovanú k obeti a tak preťaží jeho kapacity, či už výpočtový výkon alebo dostupnú šírku pásma. Obeť nie je schopná odpovedať na reálne požiadavky od užívateľov čím sa stáva služba, ktorú poskytuje nedostupná. Dôvodom, prečo sa tento druh útoku nazýva amplifikačný je to, že útočník potrebuje disponovať relatívne malou rýchlosťou pripojenia do internetu a aj napriek tomu je schopný zahltiť obeť prevádzkou mnohonásobne prekračujúcou jeho možnosti.



Obr. 2.1: DNS amplifikačný útok.

Na obrázku 2.1 je znázornený princíp útoku DNS amplification. Podvrhnutím zdrojovej IP adresy sa docieli to, že odpoveď nebude smerovaná späť k útočníkovi ale k obeti, ktorej IP adresu útočník uviedol pri vytvorení požiadavky na DNS resolver ako svoju. Podvrhnutie zdrojovej IP adresy je pomerne jednoduché. DNS je postavené na UDP a v UDP neprebieha žiadna verifikácia toho, či zdrojová IP adresa uvedená v požiadavke je skutočná adresa odosielateľa. Použitím jednoduchých nástrojov je útočník schopný posielať tisícky požiadaviek s podvrhnutou zdrojovou IP adresou na DNS resolvery, ktoré odošlú na danú zdrojovú IP adresu omnoho väčšie odpovede než sú im príslušné požiadavky.

2.7 Malware

Malware [6] je všeobecné označenie pre škodlivý software. Je to akýkoľvek software, ktorý má za cieľ narušiť činnosť počítača, zbierať citlivé informácie alebo získať prístup do privátnych počítačových systémov. Patria sem napríklad vírusy, červy, trojské kone, spyware, dialers a rootkits.

2.7.1 Počítačový vírus

Počítačový vírus je program, ktorý sa dokáže replikovať pridaním vlastného kódu do iných programov alebo spustitelných súborov. Do systému sa môže dostať spustením infikovaného hostiteľského programu alebo spustitelného súboru. To znamená, že pre svoje šírenie potrebuje hostiteľa.

2.7.2 Počítačový červ

Počítačový červ je program, ktorý sa dokáže automaticky kopírovať na iné systémy v počítačovej sieti. Hneď ako infikuje systém, využije jeho sieťové prostriedky na ďalšie šírenie. Počítačový červ na rozdiel od počítačového vírusu nepotrebuje k svojmu šíreniu hostiteľský program alebo spustitelný súbor.

2.7.3 Trojský kôň

Trojský kôň je škodlivý kód alebo program, ktorý je zvyčajne pribalený k software z nedôveryhodných zdrojov. Trojský kôň úmyselne vykonáva nejakú skrytú činnosť ako napríklad krádež hesiel, mazanie súborov, vytváranie zadných vrátok, pripájanie k iným počítačom. Nevykonáva replikáciu samého seba.

2.7.4 Spyware

Spyware je program, ktorý bez vedomia používateľov zistuje o nich a o ich počítači informácie a odosiela ich cudzej osobe. Zbierané informácie môžu byť rôzne. Napríklad e-mailové adresy, navštívené internetové stránky a iné. Najnebezpečnejším druhom spyware je tzv. keylogger. Keylogger zaznamenáva všetky stlačené klávesy, a teda tento druh spyware môže bez problémov získať prístupové mená a heslá, ktoré používateľ zadal na klávesnici.

2.7.5 Dialer

Dialer je program, ktorý využíva vytáčané spojenie cez telefónnu linku (Dial Up). Dialery presmerujú číslo, pomocou ktorého sa používateľ pripája na internet na audiotextové číslo. Niektoré dialery dokážu nastaviť spojenie tak aby zostalo aktívne aj po zatvorení prehliadača. Technológia vytáčaného spojenia cez telefónnu linku je dnes už raritou, a preto je už pravdepodobnosť vzniku takéhoto incidentu veľmi malá.

2.7.6 Rootkit

Rootkit je program alebo balík programov, ktorý umožňuje vytvoriť, spravovať a utajiť prostredie pre útočníka na kompromitovanom systéme. Rozlišujeme binary rootkits, ktoré modifikujú systémové súbory, ďalej kernel rootkits, ktoré modifikujú komponenty jadra a library rootkits. Tie prepisujú systémove knižnice.

2.8 Botnet

Termínom botnet sa označujú internetoví roboti, ktorí fungujú autonómne. V súčasnosti je termín botnet spájaný práve s malware, kedy botnet označuje skupinu počítačov na počítačovej sieti, ktoré sú infikované určitým druhom malware, ktorý je centrálne riadený.

Potom botnet môže vykonávať nežiadúcu činnosť ako sú DDoS útoky, rozposielanie spamu a podobne.

Kapitola 3

Detekcia nežiadúceho obsahu v počítačových sieťach

Pre detekciu nežiadúceho obsahu v počítačových sieťach existuje veľmi veľa metód a prístupov. Jedným z častých prístupov je monitorovanie sieťovej prevádzky pomocou NetFlow/IPFIX a vyhodnocovanie zozbieraných dát. Ďalším rozšíreným prístupom je systém Honeypot. Pre vyhodnocovanie sieťovej prevádzky sa tiež používajú systémy Intrusion Detection System (IDS). Následný zber informácii o incidentoch zachytených rôznymi detekčnými systémami vykonávajú systémy Security Information and Event Management (SIEM).

3.1 NetFlow/IPFIX

NetFlow je otvorený protokol vyvinutý spoločnosťou Cisco Systems určený pôvodne ako doplnok k ich smerovačom. NetFlow je v súčasnosti najrozšírenejší priemyselný štandard pre monitorovanie počítačových sietí na základe IP tokov. Tok je v NetFlow definovaný ako postupnosť paketov so zhodnou päticou cieľová/zdrojová IP adresa, cieľový/zdrojový port a číslo protokolu. Pre každý tok je zanamenávaný čas vzniku, dĺžka trvania, veľkosť prenesených dát a iné. NetFlow architektúra sa typicky skladá z niekoľkých exportérov a jedného kolektoru. Exportér je pripojený k monitorovanej časti siete a analyzuje sieťovú prevádzku. Na základe zachytených IP tokov exportér generuje NetFlow štatistiky a tie odosiela na kolektor. Kolektor je zariadenie s veľkou úložnou kapacitou. Kolektor teda získava štatistiky z viacerých exportérov. Nad týmito štatistikami zvyčajne bežia rôzne vizualizačné aplikácie a analyzátory.

Internet Protocol Flow Information Export (IPFIX) bol vytvorený pracovnou skupinou IETF na základe potreby univerzálneho štandardu pre export informácií o IP tokoch. Štandard definuje ako sú informácie o IP tokoch formátované a prenášané z exportéru na kolektor.

3.2 Honeypot

Honeypot je systém, ktorého cieľom je priťahovať potencionálnych útočníkov a zaznamenávať ich činnosť. Systémy honeypot detekujú činnosť neoprávnených zdrojov príchádzajúcich do systému. Sledovanie zdrojov je plne automatické. Jedným z hlavných využití systémov honeypot je detekcia malware s dostatočným predstihom a analýza jeho chovania. Získané informácie sa následne môžu využiť pre aktualizáciu vírusových databáz.

3.2.1 Systémy honeypot s nízkou mierou interakcie

Systémy honeypot s nízkou mierou interakcie simulujú len niekoľko funkcii transportnej vrstvy. Tento typ systému je výhodný vďaka rýchlej nasaditeľnosti a jednoduchému detekevaniu zmapovaných hrozieb. Avšak detekcia nových hrozieb je vo väčšine prípadov nemožná.

3.2.2 Systémy honeypot s vysokou mierou interakcie

Systémy honeypot s vysokou mierou interakcie simulujú celý reálny systém so všetkými funkciami a službami. Tento spôsob implementácie umožňuje napadnutie celého simulovaného systému vrátane honeypotu. Z tohoto dôvodu je údržba systému s vysokou mierou interakcie omnoho zložitejšia. Na druhej strane tento prístup umožňuje detekovať nové typy útokov.

3.3 Intrusion Detection System

Intrusion Detection System (IDS) je systém, ktorý monitoruje sieťovú prevádzku a snaží sa identifikovať nežiadúcu činnosť. Hlavnými činnosťami IDS sú detekcia podozrivých aktivít, ktoré by mohli viesť k vzniku bezpečnostného incidentu v operačnom systéme alebo v počítačovej sieti a prípadný aktívny zásah proti nim. Ako som už načrtol, IDS sa nezaoberá len samotnými pokusmi o narušenie bezpečonosti ale aj o detekciu správania, ktoré bezpečnostným incidentom prechádza. IDS by mal byť schopný rozlíšiť útoky z vnútra siete a externé útoky. Po detekcii udalosti IDS vygeneruje hlásenie (Alert), zapíše udalosť do logu a prípadne zastaví činnosť, ktorá spôsobila vyvolanie tejto udalosti (aktívny IDS). Aktívne IDS sú tiež označované ako Intrusion Prevention System (IPS).

3.3.1 Host-based intrusion detection system

Host-based intrusion detection system (HIDS) pozostáva z programového agenta, ktorý sa pokúša detekovať útoky pomocou analýzy systémových volaní, činnosti aplikácií, súborového systému a podobne.

3.3.2 Network intrusion detection system

Network intrusion detection system (NIDS) je nezávislá platforma, ktorá monitoruje sieťovú prevádzku od viacerých hostiteľov. Senzory NIDS bývajú umiestnené priamo na sieťových prvkoch.

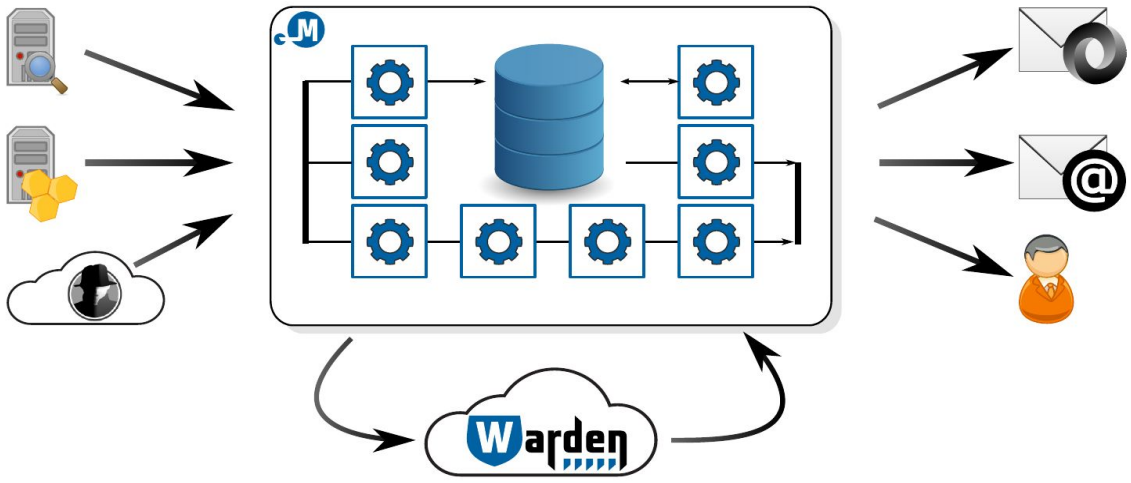
3.4 Security Information and Event Management

Security Information and Event Management (SIEM) je manažment bezpečnostných informácií a udalostí. Kombinuje Security Information Management (SIM), ktorý sa zaoberá dlhodobým ukladaním udalostí, ich analýzou a hlásením problémov a Security Event Management (SEM), ktorý sa zaoberá monitorovaním infraštruktúry, korelovaním udalostí a generovaním upozornení (Alerts) v reálnom čase. Hlavné ciele SIEM sú pružnejšia a rýchlejšia reakcia na útoky, úspešnejšia detekcia útokov, zefektívnenie správy infraštruktúry a získavanie automaticky vytvorených štatistík o infraštruktúre.

Kapitola 4

Mentat

Mentat [4] je distribuovaný modulárny SIEM navrhnutý pre monitoring sietí všetkých veľkostí. Jeho architektúra umožňuje príjem, ukladanie, analýzu, spracovanie a reakciu na veľké množstvo bezpečnostných udalostí pochádzajúcich z najrôznejších zdrojov, napríklad systémov honeypot, sieťových sond, analyzátorov logov, detekčných služieb tretích strán a podobne. Systém Mentat je vyvíjaný ako OpenSource projekt. Systém Mentat je teda platforma umožňujúca jednotný zber udalostí z najrôznejších heterogénnych zdrojov a ich následné jednotné vyhodnocovanie, spracovanie a prehľadávanie.



Obr. 4.1: Prehľad systému Mentat [4].

Systém Mentat je navrhnutý ako modulárny distribuovaný systém s dôrazom na bezpečnosť, rozšíritelnosť a škálovatelnosť. Jadro systému je implementované na podobnom

princípe ako MTA Postfix. Skladá sa z viacerých pomerne jednoduchých modulov. Tento prístup umožňuje jednoduchú paralelizovatelnosť a rozšíriteľnosť. Všetky moduly používajú jednotnú kostru a framework, čo znamená, že je veľmi jednoduché pridávať ďalšie a tak rozširovať funkčnosť. Systém používa dokumentovo orientovanú NoSQL databázu MongoDB pre perzistentné ukladanie dát. Ako dátový model je používaná IDEA [7], ktorá bola navrhnutá s ohľadom na popis širokého spektra bezpečnostných udalostí a s ohľadom na budúcu rozšíriteľnosť. IDEA je podrobne popísaná v kapitole 6.

Kapitola 5

Warden

Systém Warden [8] predstavuje spôsob efektívneho zdieľania informácií o detekovaných bezpečnostných incidentoch. V súčasnej dobe je vyvíjaný, testovaný a prevádzkovaný pre potreby siete národného výskumu a vzdelávania CESNET2, ktorú prevádzkuje združenie CESNET. V blízkej budúcnosti je však plánovaný rozvoj systému Warden ako OpenSource projektu.

Veľa CERT/CSIRT tímov sa v súčasnosti venujú vývoju a prevádzke systémov v oblasti monitorovania sieťovej prevádzky a odhaľovaniu anomálií. Tieto nástroje bývajú založené na báze NetFlow, IDS, systémov honeypot a iných. Ich výstupy využívajú CERT/CSIRT tímy pre svoju potrebu a zabezpečenie vlastnej siete. Tieto výstupy by ale mohli byť využité aj inými bezpečnostnými tímami pôsobiacimi v iných sieťach. Pre tento účel bol navrhnutý systém, prostredníctvom ktorého môžu CERT/CSIRT tímy jednoducho zdieľať informácie o detekovaných bezpečnostných incidentoch.

Warden umožňuje CERT/CSIRT tímom jednoducho a efektívne zdieľať a využívať informácie o detekovaných anomáliach. Správci sietí môžu do systému Warden na dobrovoľnej báze zasielať dáta, ktoré sa nemusia týkať ich siete a naopak si sťahujú dáta, ktoré sú užitočné pre bezpečnosť siete pod ich správou.

5.1 Architektúra

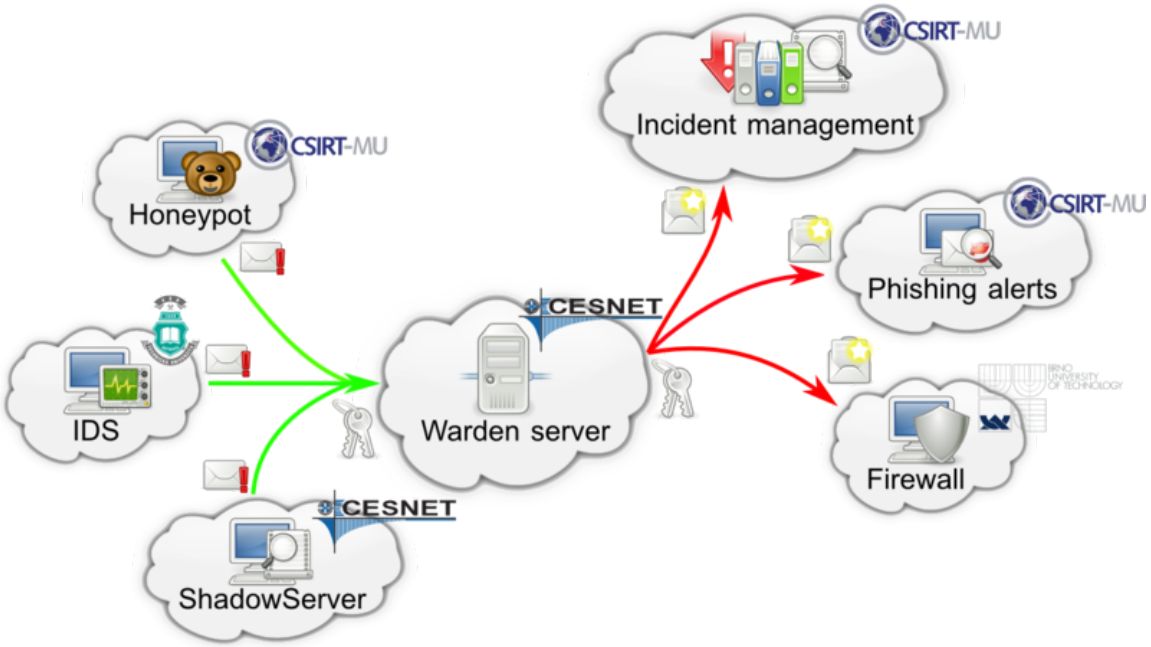
Systém sa skladá z hlavného Warden serveru obstarávajúceho všetku zásadnú činnosť a dvoch typov klientov. Prvým je odosielajúci klient, ktorý sa stará o dodávanie informácii poskytovaných zapojenou organizáciou na Warden server. Druhý typ predstavuje odoberajúci klient určený pre získavanie informácií zapojenou organizáciou. Klienta je možné nastaviť pre príjem požadovaného typu udalostí.

Serverová časť systému Warden zaistuje prijatie a ukladanie informácií zasielaných klientami. Poskytuje tiež rozhranie ku všetkým platným uloženým odalostiam. Odosielanie a prístup k udalostiam na serveri je chránený autentizáciou pomocou X.509 certifikátu.

5.1.1 Warden klienti

Odosielajúci klient poskytuje základnú funkcionalitu potrebnú pre odosielanie informácii o hrozbe zachytenou zapojenou organizáciou na server systému Warden. Organizácia tak môže jednoducho odosielať zo svojich detekčných nástrojov informácie do systému Warden.

Odoberajúci klient poskytuje organizácii jednoduchý nástroj umožňujúci stiahnúť z Warden servera požadované informácie a zobraziť ich, prípadne ich zapísať do súboru.



Obr. 5.1: Architektúra systému Warden [8].

5.1.2 Warden udalosť

Udalosť v systéme Warden predstavuje informácie o zdroji hrozby zaznamenané niektorým zo zapojených členov. Informácie sú získavané z rôznych zdrojov. Môžu to byť detekčné systémy prevádzkované v sieti spolupracujúcej organizácie, ako sú napríklad IDS, systémy honeypot a monitoring útokov na autentizáciu SSH, dáta tretích strán ako napríklad Shadowservers, Honeynet a korelované alebo agregované dáta. Reálny obsah jednotlivých udalostí sa líši podľa typu. Použitý je štruktúrovaný formát IDEA.

Kapitola 6

Intrusion Detection Extensible Alert

Intrusion Detection Extensible Alert (IDEA) je formát používaný v systémoch Mentat a Warden. Cieľom tohto projektu je navrhnúť metódy na agregáciu a koreláciu dát práve v tomto formáte. Preto sa v tejto kapitole budem podrobne venovať práve tomuto formátu.

6.1 Popis

Deskriptívne dátové formáty, špeciálne formáty pre bezpečnostné udalosti sú založené na modeli kľúč:hodnota, kde kľúč môže byť jednoduchý token alebo cesta v strome. Formát IDEA nie je výnimkou.

6.1.1 Základná štruktúra

Formát IDEA používa rovnaký dátový model ako JSON vďaka čomu je ľahko reprezentovateľný dátovými typmi najčastejšie používaných programovacích jazykov. Serializovaný môže byť však inak (XML, YAML či dokonca BSON) a to najmä v NoSQL, či dokumentovo orientovaných databázach.

Formát je definovaný ako najviac dvojúrovňový strom kľučov a hodnôt. To má za následok len jednu úroveň nejednoznačnosti v reprezentácii relačných modelov (okrem polí) a tiež predchádza nedostatku predikovateľnosti a vyhľadateľnosti pri mnohoúrovňových či rekurzívnych schémach.

6.1.2 Definícia

Formát IDEA definuje niekoľko najčastejšie používaných kľučov. Na druhej strane formát umožňuje rozšírenia pridaním vlastných kľúčov ak nekolidujú s už definovanými kľúčmi v špecifikácii. V prípade kolízie mien kľúčov je len na strane príjemcu, akým spôsobom kolidujúce kľúče spracuje. To všetko znamená, že pri validácii správ je kontrolovaná správnosť len známych kľúčov definovaných v špecifikácii, všetko ostatné zostane nedotknuté.

V nasledujúcej časti tejto sekcie popíšem známe kľúče definované v špecifikácii.

- **Format**: Identifikácia verzie IDEA.
- **ID**: Unikátny identifikátor správy.

- **AltNames**: Alternatívne identifikátory. Reťazce, ktoré pomáhajú správu priradiť k informáciám špecifickým pre systém, ktorý správu vygeneroval.
- **CorrelID**: Pole identifikátorov (ID) správ, ktoré sú zdrojom informácií pre vytvorenie tejto správy v prípade, že bola vytvorená na základe korelácie iných správ.
- **AggrID**: Pole identifikátorov (ID) správ, pre ktoré je táto správa výsledkom ich agregácie.
- **PredID**: Pole identifikátorov (ID) správ, ktoré sú neaktuálne a informácie v nich sú nahradené touto správou.
- **RelID**: Pole identifikátorov (ID) správ, ktoré nejakým spôsobom súvisia s touto správou.
- **CreateTime**: Čas vytvorenia IDEA správy.
- **DetectTime**: Čas detekcie udalosti, ktorú táto správa popisuje.
- **EventTime**: Predpokladaný začiatok útoku/udalosti.
- **CeaseTime**: Predpokladaný koniec útoku/udalosti.
- **WinStartTime**: Začiatok agregačného okna, v ktorom bola udalosť pozorovaná.
- **WinEndTime**: Koniec agregačného okna, v ktorom bola udalosť pozorovaná.
- **ConnCount**: Počet spojení, ktoré boli nadviazané.
- **FlowCount**: Počet simplexných tokov.
- **PacketCount**: Počet prenesených rámcov.
- **ByteCount**: Počet prenesených bajtov.
- **Category**: Pole kategórií udalosti (detailne popísane v kapitole 6.2).
- **Ref**: Pole URI, ktoré odkazujú na zdroje súvisiacie s udalosťou (URI alebo URN).
- **Confidence**: Číselná hodnota, ktorá pre detektor určuje dôveryhodnosť jeho vlastnej detekcie konkrétnej udalosti (0 - nedôveryhodná, 1- žiadne pochybnosti).
- **Description**: Krátky text zrozumiteľný pre človeka.
- **Note**: Doplňujúca poznámka zrozumiteľná pre človeka.
- **Source**: Pole objektov, ktoré nesú informácie o zdrojoch.
  - **Type**: Pole kategorií, ktoré bližšie špecifikujú tento zdroj.
  - **Hostname**: Pole s hostiteľskými menami tohto zdroja.
  - **IP4**: Pole IPv4 adries tohto zdroja.
  - **MAC**: Pole MAC adries tohto zdroja.
  - **IP6**: Pole IPv6 adries tohto zdroja.
  - **Port**: Pole portov, ktoré používal tento zdroj.

  - **Proto**: Pole protokolov, ktoré používal tento zdroj.
  - **URL**: Pole URL adries tohto zdroja.
  - **Email**: Pole e-mailových adries (napr. reply-to adresy v phishingových správach).
  - **AttachHand**: Identifikátory príloh súvisiacich s týmto zdrojom.
  - **Note**: Doplňujúca poznámka zrozumiteľná pre človeka.
  - **Spoofed**: Boolean hodnota, ktorá je nastavená na True v prípade, že tento zdroj je podvrhnutý.
  - **Imprecise**: Boolean hodnota, ktorá je nastavená na True v prípade, že informácie o tomto zdroji boli zadané nepresne.
  - **Anonymised**: Boolean hodnota, ktorá je nastavená na True v prípade, že informácie o tomto zdroji su anonzmizované alebo neúplne.
  - **ASN**: Pole ASN hodnôt tohto zdroja.
  - **Router**: Pole s informáciami o smerovačoch/rozhraniach na trase v sieti. Tieto údaje bývajú špecifické pre každú organizáciu a mimo danej organizácie nedávajú význam.
  - **Netname**: Identifikátor siete podľa databázy regionálneho registrátora.
  - **Ref**: Pole odkazov na zdroje súvisiace s touto udalosťou.
- **Target**: Pole objektov, ktoré nesú informácie o cieľoch. Štruktúra objektu pre cieľ je totožná so štruktúrou objektu pre Source.
- **Attach**: Pole objektov nesúcich ďalšie informácie a dáta súvisiace s touto udalosťou.
- **Node**: Pole objektov, ktoré nesú informácie o detektoroch a ďalších zariadeniach ako sú agregátory, korelátory a iné.
  - **Name**: Názov detektoru. Názov musí byť unikátny a zvyčajne reprezentuje celok v hierarchii v rámci organizácie, do ktorej detektor patrí a jeho meno.
  - **Type**: Pole reprezentujúce jednotlivé typy detektorov.
  - **SW**: Pole mien detekčných programov.
  - **AggrWin**: Veľkosť agregačného okna.
  - **Note**: Dopĺnujúca poznámka zrozumiteľná pre človeka.

Podrobnejší popis vrátane popisu dátových typov je uvedený v špecifikácii [9].

6.2 Klasifikácia

Bezpečnostné udalosti sa klasifikujú podľa kategórií popísaných v špecifikácii formátu IDEA. Všetky aplikovateľné kategórie musia byť použité. Ak si nie sme istý presným zaradením, môžeme bezpečnostnú udalosť klasifikovať len hlavnou kategóriou.

Pri vytváraní správy by mal autor čo najlepšie popísať bezpečnostnú udalosť existujúcimi menami kategórií. Nové kategórie/subkategórie môžu byť použité len ak sa žiadne existujúce nedajú aplikovať. To by sa malo stať iba v prípade, že vznikne nový typ bezpečnostnej udalosti. V prípade, že je udalosť generovaná strojom, môže nový typ bezpečnostného incidentu zaradiť do kategórie Other.

Pri pomenovaní nových kategórií je nevyhnutné dodržať camel case. Skratky môžu byť použité len ak sú všeobecne známe - napríklad DDoS alebo ak sú založené na celom mene - napríklad Recon. Čísla, podtržítko, a mínus znamienko sú rezervované na prípadné rozšírenia obsahujúce rôzne oddeľovače a podobne.

V tabuľke 6.1 popíšem tie kategórie bezpečnostných udalostí, ktorými sa budem zaoberať v tejto práci.

Kategória	**Subkategória**	**Popis**
Abusive	Spam	Nevyžiadané hromadné e-maily, to znamená, že príjemca neudelil overitelné oprávnenie na odoslanie správy a správa je odoslaná ako časť väčšej kolekcie správ, z ktorých všetky majú funkčne podobný obsah.
Malware	Virus	Program, ktorý je zámerne vložený do systému za zlým zámerom. Interakcia s používate?om je pre aktiváciu kódu nevyhnutná.
	Worm	
	Trojan	
	Spyware	
	Dialer	
	Rootkit	
Recon	Scanning	Posielanie požiadaviek na systém za účelom odhalenia slabých miest v systéme. To zahŕňa niektoré druhy testovacích procesov na zber informácii o hostiteteľoch, službách a účtoch. Prikladý: fingerd, DNS quering, ICMP, SMTP, skenovanie portov a host sweeping.
Atempt	Login	Viacnásobný pokus o prihlásenie sa (slovníkový útok).
Intrusion	AdminCompromise	Úspešná kompromitácia systému alebo aplikácie (služby). To môže byť vykonané vzdialene využitím známej alebo novej zraniteľnosti, ale tiež neautorzizovaným lokálnym prístupom. Zahŕňa tiež možnosť byť súčasťou botnetu.
	UserCompromise	
	AppCompromise	
	Botnet	
Availability	DoS	Systém je vystavený obrovskému množstvu požiadaviek, čo má za následok veľke oneskorenie alebo pád systému. Príklady DoS: ICMP a SYN floods, Teardrop útok a mail-bombing.
	DDoS	DDoS útoky sú zvyčajne založené na DoS útokoch smerovaných z botnetu, ale existujú aj iné druhy ako napríklad DNS Amplification útok.
Fraud	Copyright	Propagovanie kópii nelicencovaného komerčného software alebo iných materiálov podliehajúcich autorskému zákonu.
Vulnerable	Open	Zraniteľnosť, riešiteľná patchom, update a pod. Zraniteľnosť je odhalená prostredníctvom Nessus a iných nástrojov.
	Config	Zraniteľnosť, riešiteľná prenastavením.

Tabuľka 6.1: Klasifikácia niektorých bezpečnostných udalostí v IDEA.

6.3 Príklady

V tejto časti ukážem niekoľko príkladov správ uložených vo formáte IDEA. Z ukážok boli vynechané niektoré menej podstatné polia, ktoré nie sú dôležité pre názornosť.

```
{
  "Format": "IDEA0",
  "ID": "aecfe500−4ebf−4f59−941f−c7dadf5a798c",
  "DetectTime": "2015−12−30T10:16:58Z",
  "Category": ["Recon.Scanning"],
  "ConnCount": 1236,
  "Description": "Ping scan",
  "Source": [
    {
      "IP4": ["192.168.0.111"],
      "Proto": ["icmp"]
    }
  ],
  "Target": [
    {
      "Proto": ["icmp"],
      "IP4": ["192.168.200.0/24"],
      "Anonymised": true
    }
  ]
}
```


Ukážka 6.1: Skenovanie portov.


```
{
   "ID" : "957b3f64−3977−4286−bbd0−098ce22a82ae",
   "FlowCount" : 79563,
   "Format" : "IDEA0",
   "Node" : [
      {
         "Type" : ["Relay"],
         "SW" : [ "Mentat", "warden_filer−receiver"],
         "Name" : "cz.cesnet.mentat.warden_filer"
      },
      {
         "Type" : [ "Flow", "Statistical"],
         "Name" : "cz.cesnet.nemea.hoststats"
      }
   ],
   "Target" : [
      {
         "Proto" : [ "udp", "dns"],
         "IP4" : [ "192.168.100.1" ]
      }
   ],
   "Category" : ["Availability.DoS"]
}
```


Ukážka 6.2: DNS Amplification.

Kapitola 7

Analýza dát

Pre analýzu rôznych typov bezpečnostných hlásení a testovacie účely mám k dispozícii obrazy databázy systému Mentat z obdobia od 1.6.2015 do 1.7.2015 (A) a z obdobia od 1.11.2015 do 1.12.2015 (B). Obrazy A a B celkovo obsahujú okolo 63 miliónov hlásení rôznych kategórií.

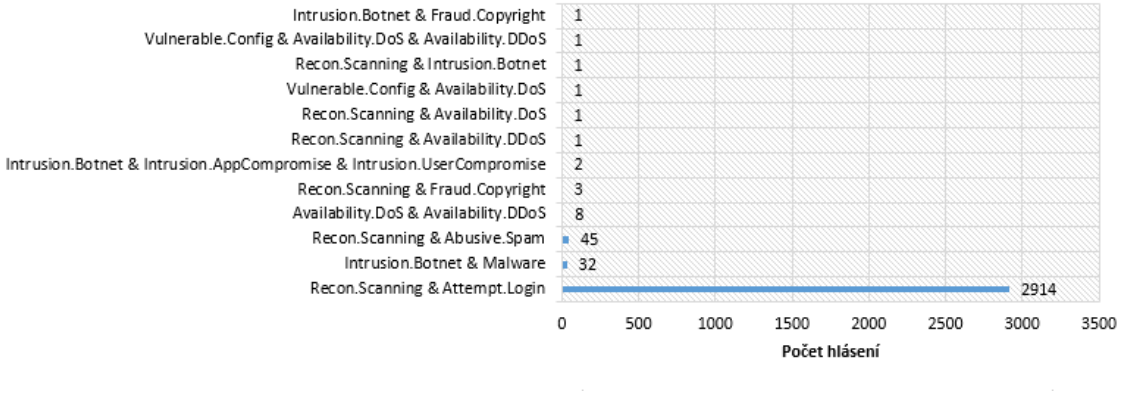
Kategória	**Počet**
Fraud.Copyright	176790
Vulnerable.Config	6616
Vulnerable	331
Intrusion.AppCompromise	36
Intrusion.UserCompromise	36
Recon.Scanning	27295094
Intrusion.Botnet	10145
Availability.(D)DoS	7100
Abusive.Spam	6943
Malware	9981
Attempt.Login	1718566

Tabuľka 7.1: Podiel typov bezpečnostných hlásení v obraze A.

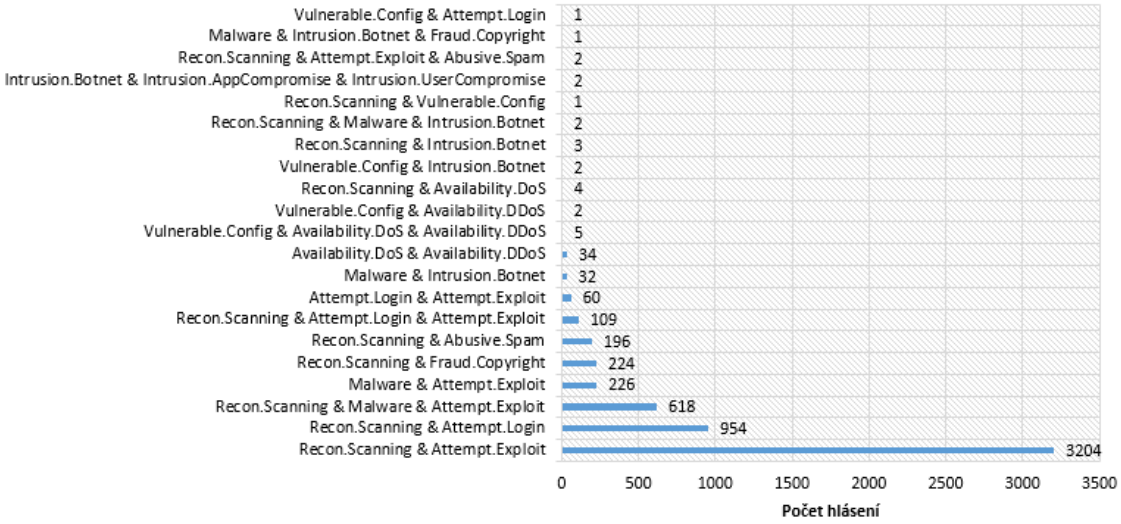
Kategória	Počet
Attempt.Exploit	20501
Availability.DDoS	28686
Recon.Scanning	32578157
Attempt.Login	961008
Vulnerable.Config	4027
Availability.DoS	46433
Abusive.Spam	11852
Intrusion.Botnet	17749
Malware	19779
Intrusion.AppCompromise	13
Intrusion.UserCompromise	13
Fraud.Copyright	4023

Tabuľka 7.2: Podiel typov bezpečnostných hlásení v obraze B.

V tabuľkách 7.1 a 7.2 je znázornený podiel všetkých typov hlásení v dostupných vzorkách dát. Už na prvý pohľad vidieť, že veľmi výrazný podiel hlásení tvoria hlásenia o skenovaní portov. Významnu časť z nich tvoria hlásenia, ktoré majú iba jeden cieľový uzol, no pre skenovanie portov je typické, že cieľov je niekoľko. V prípade horizontálneho skenovania portov je to viacero IP adries a v prípade vertikálneho skenovania portov je to viacero portov. To naznačuje, že o jednom skenovaní existuje hneď niekoľko hláseni, z ktorých každé nesie informáciu práve o jednom skenovanom cieli. V hláseniach sa tiež vyskytujú duplicity a niektoré dlhé skenovania sú nahlásené viackrát. Vzhľadom na to, že hlásenia o skenovaní portov tvoria jednoznačne najväčšiu zložku hlásení vo vzorke dát a vzhľadom na spomenuté skutočnosti sa otvára veľký priestor na agregáciu týchto hlásení.



Obr. 7.1: Histogram výskytov rôznych kombinácií kategórií k jednej IP adrese (A).



Obr. 7.2: Histogram výskytov rôznych kombinácií kategórií k jednej IP adrese (B).

Histogramy 7.1 a 7.2 ukazujú počet výskytov rôznych kombinácií kategórií k jednej IP adrese.

Z takto získaných dát sa dá vysledovať istá nadväznosť jednotlivých typov hlásení. Ak je napríklad z jednej konkrétnej IP adresy zaznamenaná udalosť spadajúca do kategórie Recon.Scanning a následne z tej istej adresy je zaznamenaná udalosť spadajúca do kategórie Attempt.Login, je pravdepodobné, že tieto dve aktivity súvisia. To znamená, že aj keď sa jedná o dve odlišné kategórie hlásení, môžeme na ne pozerať ako na jeden bezpečnostný incident zložený z dvoch, prípadne aj viacerých fáz. Na druhej strane, to že dva rôzne typy hlásení majú rovnakú IP adresu nemusí nutne znamenať, že tieto hlásenia spolu súvisia. Dôležité sú časové rozostupy medzi jednotlivými udalostami alebo ďalšie charakteristiky špecifické pre konkrétny typ udalosti.

V databáze sa nachádzajú hlásenia z roznych zdrojov [10], ako interných tak aj externých. Interné zdroje sú zdroje v sieti CESNET2 a externé zdroje sú zdroje iných organizácií. Každý zdroj hlásení môže mať svoj vlastný spôsob generovania hlásení o bezpečnostných incidentoch. To znamená, že nastávajú situácie kedy napríklad dva na sebe nezávislé detektory zaznamenajú ten istý incident a nimi vygenerované hlásenia nemusia byť totožné. Či už z dôvodu, že každý detektor má inak nastavenú politiku tvorby hlásení (jeden môže niektoré polia zanedbať a naopak) alebo jeden z detektorov môže zachytiť väčšiu časť bezpečnostného incidentu ako ten druhý.

Kedže IDEA obsahuje indikátory pre podvrhnuté, neúplne alebo anonymizované informácie či už o zdroji alebo o cieli, v návrhu musí byť zohladnený aj tento aspekt. Rovnako hoci sa v dátach nevyskytovali hlásenia o skenovaní portov, ktoré by obsahovali viacero zdrojov, formát IDEA to umožňuje. Preto s touto skutočnosťou musím v návrhu počítať.

Aspekty ako zložitejšie bezpečnostné incidenty skladajúce sa z niekoľkých fáz alebo odlišnosti v generovaní hlásení musím samozrejme brať do úvahy, z čoho vzniká problém korelácie bezpečnostných hlásení.

V analýze konkrétnych typov hlásení o bezpečnostných udalostiach som si ako prvé vybral hlásenia z kategóre Recon.Scanning. Je to z toho dôvodu, že týchto hlásení je jednoznačne najviac a relatívne jednoducho sa agregujú.

7.1 Recon.Scanning

Ako som už písal vyššie, hlásenia o skenovaní portov tvoria jednoznačne najväčšiu časť zo všetkých hlásení v danej vzorke dát. Pri tak veľkom množstve hlásení už človek stráca prehľad. Aj z tohto dôvodu a aj z dôvodu úspory miesta v databáze by bolo veľmi výhodné medzi týmito hláseniami nájsť súvislosti a redukovať ich počet. Prvým krokom je nájsť vhodnú mieru abstrakcie. Teda určiť, na základe čoho môžeme prehlásiť o ľubovoľných dvoch bezpečnostných hláseniach o skenovaní portov, že spadajú pod rovnakú bezpečnostnú udalosť.

Hlásenia o skenovaní portov pochádzajú zo systémov Hoststats a LaBrea. Každý z detekčných systémov, či už je to v tomto prípade Hoststats alebo LaBrea generuje hlásenia o skenovaní portov iným spôsobom. Okrem drobných odlišností medzi štruktúrou správ je najväčším rozdielom to, že Hoststats na rozdiel od LaBrea neuvádza žiadne informácie o cieľoch. Všeobecne by však nebolo vhodné keby navrhnuté agregačné metódy rozlišovali medzi jednotlivými typmi detekčných systémov. To by malo za následok nutnosť neustáleho pridávania podpory pre nové detekčné systémy. Aj z tohto dôvodu musí byť použitá vhodná miera abstrakcie pri práci s dátami.

Nikde však nie je presne definované kedy ešte môžeme o dvoch hláseniach prehlásiť, že popisujú tu istú udalosť. Záleží to viacmenej na tom, kedy zlúčenie dvoch hlásení má pre nás ešte zmysel. Informácie, ktoré považujem za dôležité pre rozhodovanie o tom či dve hlásenia o skenovaní portov popisujú v podstate jeden bezpečnostný incident sú informácie o zdroji skenovania a časové rozmedzie, v ktorom vznikla udalosť, na základe ktorej boli hlásenia vygenerované. V tomto prípade zdroj môžeme rozlíšiť len na základe jeho IP adresy. V prípade zhody IP adries ešte musíme porovnať či potencionálne zlúčiteľné hlásenia spadaju do vhodne zvoleného časového rozmedzia. Jeho veľkosť opäť záleží na potrebách administrátora alebo politikách organizácie. Ak teda ľubovoľné dve hlásenia o skenovaní portov majú rovnakú zdrojovú IP adresu a spadajú do rovnakého časového okna, budem ich považovať za zlúčiteľné.

7.2 Availability.DoS a Availability.DDoS

Hlásenia patriace do kategórii Availability.DoS a Availability.DDoS pochádzajú z troch rôznych zdrojov. V dvoch prípadoch ide o systémy Hoststats. Nejedná sa však úplne o totožné systémy. V jednom prípade je použitá novšia implementácia systému Hoststats. To znamená, že hoci ide o systémy s rovnakým názvom, nemusí to znamenať, že budú generovať hlásenia rovnakým spôsobom. Ďalším systémom, ktorý generuje hlásenia spadajúce do týchto kategórií je systém Amplificationdetector.

Ako už názvy kategórií napovedajú, jedná sa o hlásenia spadajúce pod DoS a DDoS útoky. V tomto prípade však systémy, z ktorých pochádzajú vyššie spomenuté hlásenia nevedia určiť, či sa jedná o distribuované DoS útoky alebo nie. Preto pre potreby tejto práce nebudem medzi kategóriami Availability.DoS a Availability.DDoS rozlišovať.

Pod (D)DoS útoky patrí široká škála rôznych druhov a spôsobov týchto útokov. Preto by nebolo vhodné deliť hlásenia len na základe ich kategórie. V dostupnej vzorke dát drvivá väčšina (D)DoS hlásení predstavuje hlásenia o DNS amplifikačných útokoch. Zvyšné hlásenia predstavujú hlásenia o SYN flood útokoch. Aj z tohto dôvodu sa ďalej zameriam na hlásenia o DNS amplifikačných útokoch. Hlásenia o DNS amplikifačných útokoch sa dajú identifikovať podľa použitého portu a protokolu. Názov útoku napovedá, že je využívaný DNS protokol. DNS protokol využíva UDP a port 53. V hláseniach sú uvedené protokoly

dva a to UDP a DNS. Pri analýze som zistil, že všetky hlásenia o DNS amplifikačných útokoch majú v poli Description reťazec "DNS amplification".

Či už Hoststats alebo Amplificationdetector generuje hlásenia iným spsôsobom. Pre potreby agregácie hlásení o DNS amplifikačných útokoch je najzásadnejším údajom cieľ v prípade, že chceme agregovať správy patriace k jednému konkrétnemu útoku. V opačnom prípade zmysel tiež dáva agregovať správy podľa zdroja útoku, za ktorý je väčšinou označený zneužitý DNS server. Tu však nastáva problém. V prípade Hoststats je vždy buď uvedený cieľ alebo zdroj. To znamená, že nikdy nevieme s určitostou povedať, podľa čoho hlásenia o DNS amplifikačných útokoch agregovať.

7.3 Attempt.Login

Hlásenia patriace do kategórie Attempt.Login pochádzajú z viacerých zdrojov. Sú to systém Hoststats, niekokľo systémov Kippo (SSH honeypot) a systém Cowrie. Systém Cowrie je odnož systému Kippo.

Medzi štruktúrou hlásení z Hoststats, Kippo a Cowrie je niekoľko odlišností. Najzásadnejší vplyv na návrh agregačnej metódy má absencia informácie o zdroji útoku vo väčšine hlásení z Hoststats. Z Hoststats taktiež pochádza veľká väčšina hlásení. Všetky dostupné hlásenia sú hlásenia o pokuse o prihlásenie na SSH hrubou silou.

7.4 Attempt.Exploit

Hlásenia patriace do kategórie Attempt.Exploit pochádzajú z troch rôznych zdrojov. Vo všetkých troch prípadoch sú to systémy Dionaea (Honeypot).

Ako som už spomenul, hlásenia pochádzajú z troch rôznych zdrojov no vo všetkých prípadoch ide o rovnaký systém. Z toho plynie predpoklad, že hlásenia budú mať rovnakú štruktúru. Analýzou jednotlivých hlásení som zistil, že štruktúra hlásení z jednotlivých zdrojov je naozaj takmer zhodná. V prípade jedného z nich sú však cieľové adresy anonymizované.

7.5 Fraud.Copyright

Hlásenia patriace do kategórie Fraud.Copyright pochádzajú iba z jedného zdroja pod označením bittorrent.

Kedže hlásenia pochádzajú iba z jedného zdroja, nie je tu tak veľký priestor na agregáciu ako pri hláseniach z viacerých zdrojov. V prípade jedného zdroja hlásení môžeme agregáciu využiť na zmenu stupňa abstrakcie, ktorú používa systém generujúci hlásenia. To znamená, že ak napríklad jeden utočník útočí na systém viackrat v priebehu určitého krátkeho časového intervalo a v každom opakovaní zmení parametre útoku, detekčný systém to môže vyhodnotiť ako samostatné útoky a vygenerovať príslušný počet hlásení. Na druhej strane pre správcu siete postačuje iba jedno hlásenie. Ďalej detekčné systémy môžu pracovať v intervaloch a ak niektoré dlhšie útoky trvajú viac intervalov, systém ich nahlási viackrát. Tým sa tiež otvára priestor na istú formu agregácie.

7.6 Vulnerable.Config

Hlásenia patriace do kategórie Vulnerable.Config pochádzajú z externých zdrojov. Hlásenia sú spracovávane na serveri kde je tiež prevádzkovaný systém LaBrea a v správach je ako zdroj uvedený práve tento systém.

Hlásenia kategórie Vulnerable.Config informujú o zraniteľnostiach spôsobených zlými nastaveniami nejakých služieb. Typov zraniteľností môže existovať nespočetne veľa a niekedy sa vyžaduje ich kategorizácia.

7.7 Intrusion.Botnet

Hlásenia patriace do kategórie Intrusion.Botnet tak isto ako v prípade hlásení kategórie Vulnerable.Config pochádzajú z externých zdrojov a sú spracovávané na serveri kde je tiež prevádzkovaný systém LaBrea. Rovnako je v hláseniach ako zdroj uvedený práve tento systém.

V databáze sa vyskytujú dve podoby hlásení tejto kategórie. V obidvoch prípadoch však hlásenia informujú o tom istom, a teda, že sa niekto pripojil na kontrolný server. V prvom prípade je adresa bota uložená v Source a položka Type v Source dodatočne informuje o tom, že sa jedná o adresu bota. Adresa kontrolného servera je uložená v položke Target. V druhom prípade je situácia trochu zložitejšia. Cieľová adresa je známa, no je uložená v Attach v poli Content. Pričom pole Target vyplnené nie je.

7.8 Abusive.Spam

Hlásenia patriace do kategórie Abusive.Spam pochádzajú z dvoch zdrojov Vinovago a LaBrea.

Hlásenia sú pomerne jednoduché. V oboch prípadoch pozostávajú iba z niekoĺkých časových údajov, položky Description, z IP adresy zdroja a prílohy. V prípade LaBrea je v zdroji uvedený ešte protokol. Hlásenia z LaBrea sú však iba dve.

Kapitola 8

Návrh a implementácia

Návrh riešenia je založený na myšlienke práce s kontextami podľa nástroja Simple Event Correlator (SEC) [2]. Konktext sa skladá zo skupiny hlásení, ktoré majú byť agregované. Každé nové hlásenie je buď priradené do existujúceho kontextu a podľa nakonfigurovaných pravidiel je zlúčené so staršími hláseniami v kontexte, alebo ak žiadny vhodný kontext neexistuje, je vytvorený kontext nový. O príslušnosti hlásenia do určitého kontextu rozhodujú hodnoty niektorých položiek v prichádzajúcom hlásení. Ak sa napríklad požadované hodnoty kľučov zhodujú, hlásenie môže byť pridané do kontextu a spojené so staršími hláseniami v kontexte. Kontext má tiež určenú dobu expirácie. Kontext expiruje ak po určenú dobu nie je do kontextu pridané žiadne hlásenie. Ak kontext expiruje, vygeneruje sa hlásenie vo formáte IDEA, ktoré je výsledkom agregácie hlásení v kontexte.

Rozmanitosť hlásení o bezpečnostných udalostiach a variabilita formátu IDEA ma prinútila zamyslieť sa nad univerzálnejším riešením problému. Táto myšlienka vyustila do návrhu jednoduchého frameworku, ktorý uľahčuje implementáciu nových metód pre koreláciu a agregáciu hlásení.

Celý navrh je postavený na triede ContextMgr, ktorá inicializuje zoznam kontextov a implementuje metódu ctx_process(). Metóda ctx_process() zastrešuje prácu s kontextami, teda vytváranie konktextov podľa potreby, pridávanie hlásení do kontextov a prípadné mazanie expirovaných kontextov. Ďalej trieda ContextMgr implemetuje niekoľko ďalších pomocných metód a tiež definuje niekoľko abstraktných metód.

```
def ctx_process(self, event):

        ctx = self.match(event)
        if ctx:
            if ctx in self.contexts:
                self.ctx_append(self.contexts[ctx], event)
            else:
                self.contexts[ctx] = self.ctx_create(event)

        aggr_events = []
        remove = []
        for ctx in self.contexts:
            if self.ctx_expire(self.contexts[ctx]):
                aggr_events.append(self.ctx_close(self.contexts[ctx]))
                remove.append(ctx)
```

```
for ctx in remove:
    del self.contexts[ctx]

return aggr_events
```

Ukážka 8.1: Implementácia metódy ctx_process() v jazyku Python.

Zásadná je metóda match(), ktorá v prípade, že hlásenie patrí do danej kategórie, vráti identifikátor špecifický pre kontext, do ktorého hlásenie patrí. To však nutne neznamená, že takýto kontext už existuje. Ak neexistuje, dôjde k vytvoreniu nového kontextu. Metódu match() je nutné pri vytváraní novej triedy pre príslušnú agregačnú alebo korelačnú metódu reimplementovať.

Ďalšími metódami, ktoré vyžadujú špecifickú funkcionalitu pre danú agregačnú alebo korelačnú metódu sú ctx_create() a ctx_append(), a to z toho dôvodu, že každá metóda pre agregáciu alebo koreláciu rôznych kategórií hlásení má svoje špecifické potreby pre vytvorenie a incializáciu kontextu (ctx_create()) a pridávanie hlásení do kontextu (ctx_append()). Nutnosť implementovať tieto funkcie by však vyžadovala značné úsilie pri vytváraní nových agregačných a korelačných metód a v konečnom dôsledku by navrhovaný jednoduchý framework strácal význam a skoro vôbec by neuľahčoval realizáciu nových riešení. Preto som zvolil kompromis, pri ktorom odpadá nutnosť reimplementácie metód ctx_create() a ctx_append() a na druhej strane zavádza koncept šablón pre prácu s hláseniami.

8.1 Vytváranie a pridávanie udalostí do kontextov

Ako som už naznačil, zavedením šablón pre prácu s hláseniami odpadá potreba reimplementácie metód ctx_create() a ctx_append(). Implementácia týchto a niektorých ďalších pomocných metód sa nachádza v triede SimpleAggregator, ktorá dedí triedu ContextMgr.

Konštruktor objektu triedy SimpleAggregator príjima ako jeden z parametrov šablónu pre prácu s hláseniami. Šablónu tvorí sada parametrov, ktorá určuje akým spôsobom sa má pridať hlásenie do kontextu a v prípade vytvárania nového kontextu, ako sa má nové hlásenie v kontexte vytvoriť.

8.1.1 Šablóna

Návrh šablóny je inšpirovany formátom IDEA. Jej štruktúra vrátane pomenovaní kľúčov je prakticky totožná so štruktúrou hlásenia vo formáte IDEA, ktoré má byť výsledkom agregácie hlásení z jedného kontextu. Hodnoty príslušné jednotlivým kľúčom nesú názvy operácií, ktoré majú byť vykonané pri spájaní hlásení. Jednotlivé operácie sú popísane v tabuľke 8.1 Oproti definícii IDEA, definícia šablóny definuje ďalší kľúč default, ktorý určuje aká operácia sa aplikuje na hodnotu kľúča, ktorý nie je v šablóne uvedený. Kľúč default môže byť nastavený pre každú úroveň v IDEA hlásení zvlášť.

Operácia	Popis
concat	Konkatenácia. Operácia concat spojí retazec s príslušným kľúčom v kontexte s reťazcom s tým istým kľúčom v práve spracovávanom hlásení.
omit	Kľúč a ani jeho hodnota sa do kontextu nepridajú.
maxtime	Porovná časový údaj s prísušným kľúčom v kontexte a práve spracovávanom hlásení a do kontextu vloží väčší z nich.
mintime	Porovná časový údaj s prísušným kľúčom v kontexte a práve spracovávanom hlásení a do kontextu vloží menší z nich.
merge	Konkatenuje pole s príslušným kľúčom v kontexte s poľom v práve spracovávanom hlásení. Pole môže obsahovať ľubovoľné hodnoty akceptovatelné formátom IDEA. V prípade, že polia obsahujú jednoduché objekty, výsledkom je pole bez duplicitných objektov. Ak sa polia skladajú zo štruktúrovaných objektov, vo výsledku sa můžu nachádzať duplicity.
sum	Sčítava pole s príslušným kľúčom v kontexte s poľom v práve spracovávanom hlásení. Pole môže obsahovať ľubovoľné číselné hodnoty akceptovatelné formátom IDEA.
same	Operácia porovná hodnoty príslušného kľúča v kontexte s hodnotou v hlásení. Táto operácia sa vykonáva v prvom priechode objektom, kedy sa vyhodnocujú všetky operácie same a same_if_ present. Na základe výsledkov týchto operácii sa rozhodne či je nejaký objekt v hlásení zlúčiteľný s tým v kontexte. Operácia same môže byť aplikovaná len na položky, ktoré sú súčasťou objektu umiestneného v nejakom poli v IDEA hlásení.
same_if_present	Pracuje rovnako ako operácia same s tým rozdielom, že ak sa daný kľúč v aktuálne spracovávanom hlásení nenachádza, vyhodnotí to ako zhodujúcu sa položku.
append_self_id	Pridá ID aktuálneho hlásenia do poľa s príslušným kľúčom.
lambda	Operácia lambda v tomto prípade predstavuje funkciu lambda v jazyku Python. To znamená, že na miesto operácie je možné priamo v šablóne definovať lambda funkciu, ktorá vždy príjma dva parametre A a B. Parameter A predstavuje hodnotu príslušného kľuča v kontexte. Parameter B predstavuje hodnotu príslušného kľúča v aktuálne spracovávanom hlásení.

Tabuľka 8.1: Popis operácii pre prácu s udalostami.

V tabuľke 8.1 sú popísané všetky operácie, ktoré podporuje systém šablón. Operácia merge nesie so sebou isté obmedzenia. Z popisu v tabuľke plynie, že v prípade konkatenácie polí obsahujúcich štruktúrované objekty operácia merge môže vo výsledku spôsobiť výskyt duplicít. Preto túto operáciu na polia so štruktúrovanými objektami aplikovať neodporúčam. Jedinečnosť štruktúrovaných objektov vo výslednom poli je možné dosiahnúť použitím štruktúrovaného kľuča. Štruktúrovaný kľúč je použitý v ukážke 8.2 pre položky Source a Target. Takto definované kľuče automaticky určujú predpis konkatenácie, podľa ktorého možno konkatenovať polia so štruktúrovanými objektami. Určujú teda pravidlo, na základe, ktorého možno rozhodnúť, ktoré objekty v konkatenovaných poliach sú zhodné a akým spôsobom ich treba zlúčiť (napr. sčítanie čítačov, konkatenácia reťazcov a pod.). Ak sa niektorý objekt nedá zlúčiť so žiadnym iným objektom v poli, objekt sa pripojí na koniec

poľa.

```
{
  "CreateTime": "maxtime",
  "DetectTime": "mintime",
  "Category": "lambda a,b: [\"Aggregation\"]",
  "Note": "concat",
  "@default": "omit",
  "Source": {
    "Type": "merge",
    "IP4": "same",
    "Port": "merge",
    "@default": "omit"
  },
  "Target": {
    "Type": "merge",
    "IP4": "merge",
    "Port": "merge",
    "@default": "omit"
  }
}
```

Ukážka 8.2: Príklad šablóny.

V ukážke 8.2 je príklad popisu šablóny pre udalosti spadajúce do kategórie Recon.Scanning. V príklade rozhodne nie je popísaná úplna šablóna. Príklad bol pre zachovanie prehľadnosti a názornosti značne zjednodušený. V ukážke je použitá ako hodnota aj lambda funkcia, ktorú formát šablóny umožňuje použiť ak žiadna z preddefinovaných možností nevyhovuje.

Šablóny búdu dané konfiguráciou v súbore, preto pri potrebných zmenách nebude nutný zásah do implementácie. Koncept šablón tak zjednoduší prácu v testovacej fáze tejto práce kedy mi umožní rýchlu zmenu spôsobu vytvárania a spájania správ bez nutnosti zásahu do implementácie.

8.2 Agregácia hlásení Recon.Scanning

Vďaka frameworku je návrh a implementácia agregačnej metódy pre kategóriu hlásení Recon.Scanning pomerne jednoduchá. S využitím triedy SimpleAggregator už stačí reimplementovať iba metódu match(), ktorá ako parameter prijíma hlásenie o bezpečnostnej udalosti vo formáte IDEA. Jej výsledkom je kľúč identifikujúci konktext, do ktorého dané hlásenie patrí.

```
class ReconScanning(Aggregator.SimpleAggregator):

    def __init__(self, _templatepath = '', _timeout = None, _count = None):
       Aggregator.SimpleAggregator.__init__(self, _templatepath, _timeout,
                                            _count)

    def match(self, event):
        if "Recon.Scanning" in event["Category"]:
            return tuple(event["Source"][0]["IP4"])
```

Ukážka 8.3: Implementácia triedy ReconScanning v jazyku Python.

Ukážka 8.3 zobrazuje možnú implementáciu triedy ReconScanning reprezentujúcu agregačnú metódu pre hlásenia kategórie Recon.Scanning. Trieda dedí od triedy SimpleAggregator. Trieda SimpleAggregator je súčasťou navrhnutého frameworku. V takto vytvorenej triede ReconScanning je už iba potrebné redefinovať metódu match(). Kľuč identifikujúci kontext je tvorený IP adresou zdroja. Analýza ukázala, že hlásenia patriace do kategórie Recon.Scanning nemajú uvedených viac ako jednu zdrojovú IP adresu. Preto v tomto prípade pri tvorbe kľúča uvažujem iba s jednou zdrojovou IP adresou. V prípade ak by existovali hlásenia patriace do kategórie Recon.Scanning s viac ako jednou zdrojovou IP adresou, je takéto hlásenia nutné buď predspracovať a rozdeliť na hlásenia s jednou zdrojovou IP adresou alebo ako kľúč uvažovať všetky zdrojové IP adresy. Ako som načrtol vyššie, spôsobov ako identifikovať hlásenie je viacero. Preto tvorba kľúča do istej miery záleží na potrebách správcu siete.


```
{
  "Category": "lambda a,b: [\"Recon.Scanning\"]",
  "AggrID": "append_self_id",
  "AltNames": "merge",
  "DetectTime": "mintime",
  "EventTime": "mintime",
  "CeaseTime": "maxtime",
  "Ref": "merge",
  "Description": "concat",
  "Note": "concat",
  "Source": {
    "Type": "same_if_present",
    "Hostname": "merge",
    "IP4": "same",
    "Port": "merge",
    "Proto": "same_if_present",
    "URL": "merge",
    "Email": "merge",
    "AttachHand": "merge",
    "Note": "concat",
    "ASN": "merge",
    "Router": "merge",
    "Netname": "merge",
    "Ref": "merge",
    "@default": "omit"
  },
  "Target": {
    "Type": "same_if_present",
    "Hostname": "merge",
    "IP4": "same_if_present",
    "Port": "merge",
    "Proto": "same_if_present",
    "URL": "merge",
    "Email": "merge",
    "AttachHand": "merge",
    "Note": "concat",
    "ASN": "merge",
    "Router": "merge",
    "Netname": "merge",
    "Ref": "merge",
```

```
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
  },
  "Attach":"merge",
  "@default": "omit"
}
```

Ukážka 8.4: Šablóna pre agregáciu hlásení spadajúcich do kategórie Recon.Scanning.

Ukážka 8.4 zobrazuje jednu z možných verzii šablóny pre agregáciu hlásení kategórie Recon.Scanning. Táto šablóna bola navrhnutá mnou a navrhol som ju podľa vlastných potrieb. Potreby správcov najrôznejších sietí sa však môžu líšiť. V niektorých prípadoch sa môže vyžadovať aby bolo výsledné hlásenie menej podrobné alebo v niektorých prípadoh zase naopak. Ďalej môže existovať potreba minimalizovať veľkosť spravy kvôli úspore úložného priestoru a podobne. Od týchto všetkých aspektov sa odvíja výsledná podoba šablóny.

8.3 Agregácia hlásení Availability.DoS a Availability.DDoS

Ako som už naznačil v analýze. Hlásenia o DNS amplikifačnom útoku tvoria väčšinu hlásení kategórie Availability.DoS a Availability.DDoS a preto sa pri analýze zameriam práve na ne. A taktiež, že kategórie Availability.DoS a Availability.DDoS nebudem od seba rozlišovať.

Rovnako ako v prípade Recon.Scanning a aj ďalších prípadoch, ktoré popíšem neskôr, pri implementácii agregačne metódy pre hlásenia o DNS amplifikačných útokoch použijem navrhnutý framework.

Metódu pre agregáciu hlásení o DNS amplifikačných útokoch reprezentuje trieda DNSAmplification. Trieda DNSAmplification klasicky využíva triedu SimpleAggregator dostupnú v navrhnutom frameworku a redifunuje metódu match(), ktorá je v tomto prípade o niečo zložitejšia.

Pôvodne som navrhol riešenie, ktoré by agregovalo výlučne podľa cieľovej IP adresy aby som mohol agregovať hlásenia patriace k jednému útoku. Analýza však ukázala, že niektoré hlásenia o DNS amplifikačných útokoch obsahujú iba zdroj útoku a niektoré zase naopak iba cieľ útoku. Preto som prispôsobil implementáciu metódy match() tak, že ak sa v hlásení vyskytuje cieľ, kľúč sa zostaví na základe cieľovej IP adresy. Ak sa v hlásení cieľ nenachádza, metóda sa pokúsi zostaviť kľúč podľa zdrojovej IP adresy. Týmto spôsobom sa dosiahne vyššia miera agregácie.

Spôsobov ako agregovať hlásenia o DNS amplifikačných útokoch je niekoľko. Či už podľa cieľa, zdroja, alebo kombináciou obidvoch prístupov. Opäť to záleží na preferenciách a aktuálnych potrebách správcu siete.

```
{
  "Category": "lambda a,b: [\"Availability.DDoS\"]",
  "AggrID": "append_self_id",
  "AltNames": "merge",
  "DetectTime": "mintime",
  "EventTime": "mintime",
  "CeaseTime": "maxtime",
  "Ref": "merge",
```

```
  "Description": "concat",
  "Note": "concat",
  "Source": {
    "OutFlowCount": "sum",
    "InFlowCount": "sum",
    "OutPacketCount": "sum",
    "InPacketCount": "sum",
    "OutByteCount": "sum",
    "InByteCount": "sum",
    "Type": "same_if_present",
    "Hostname": "merge",
    "IP4": "same",
    "Port": "merge",
    "Proto": "same_if_present",
    "AttachHand": "merge",
    "Note": "concat",
    "Ref": "merge",
    "@default": "omit"
  },
  "Target": {
    "OutFlowCount": "sum",
    "InFlowCount": "sum",
    "OutPacketCount": "sum",
    "InPacketCount": "sum",
    "OutByteCount": "sum",
    "InByteCount": "sum",
    "Type": "same_if_present",
    "Hostname": "merge",
    "IP4": "same_if_present",
    "Port": "merge",
    "Proto": "same_if_present",
    "AttachHand": "merge",
    "Note": "concat",
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
  },
  "Attach":"merge",
  "@default": "omit"
}
```

Ukážka 8.5: Šablóna pre agregáciu hlásení o DNS amplifikačných útokoch.

Ukážka 8.5 zobrazuje šablónu, ktorú som navrhol vzhľadom k mnou zvolenému spôsobu agregácie. Šablóny pre agregáciu rôznych kategórii hláseni sa veľmi nelíšia. Všetky vychádzajú z rovnakého princípu spájania zdrojov a cieľov. Líšia sa len v niektorých detailoch, ktoré odzrkadľujú špecifické požiadavky správcov sietí. Šablóna pre agregáciu hlásení o DNS amplifikačných útokoch je špecifická pridaním pravidiel pre spájanie rôznych čítačov. Konkrétne OutFlowCount, InFlowCount, OutPacketCount, InPacketCount, OutByteCount a InByteCount. Tieto čítače nie sú súčasťou špecifikácie IDEA a vo výsledku budú tvoriť

sumu hodnôt týchto čítačov zo všetkých hlásení patriacich do daného kontextu.

8.4 Agregácia hlásení Attempt.Login

Z analýzy vyplýva, že hlásení kategórie Attempt.Login pochádzajucich zo systému Hoststats je jednoznačne najviac. Tvoria viac ako 90% z celkového množstva hlásení tejto kategórie. Aj napriek tomu, že prevažujú hlásenia, ktorým chýba údaj o zdrojovej IP adrese, som sa rozhodol pre agregáciu na základe zdrojovej IP adresy. Taktiež je možné agregovať hlásenia nielen podľa IP adresy, ale tiež podľa dvojice IP adresa a číslo portu. Ale to len v prípade ak by sme chceli hlásenia kategórie Attempt.Login ďalej deliť.

Metódu pre agregáciu hlásení kategórie Attempt.Login reprezentuje trieda AttemptLogin. Trieda AttemptLogin je odvodená od triedy SimpleAggregator dostupnej z navrhnutého frameworku a reimplementuje metódu match().

```
{
  "Category": "lambda a,b: [\"Attempt.Login\"]",
  "AggrID": "append_self_id",
  "AltNames": "merge",
  "DetectTime": "mintime",
  "EventTime": "mintime",
  "CeaseTime": "maxtime",
  "Ref": "merge",
  "Description": "concat",
  "Note": "concat",
  "Source": {
    "IP4": "same",
    "Port": "merge",
    "Proto": "merge",
    "Type": "same_if_present",
    "Hostname": "merge"
  },
  "Target": {
    "Type": "same_if_present",
    "Hostname": "merge",
    "IP4": "same_if_present",
    "Port": "merge",
    "Proto": "merge",
    "URL": "merge",
    "Note": "concat",
    "Netname": "merge",
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
  },
  "Attach":"merge",
  "@default": "omit"
}
```

Ukážka 8.6: Šablóna pre agregáciu hlásení kategórie Attempt.Login.

Ukážka 8.6 zobrazuje jednu z možných šablón pre agregáciu hláseni kategórie Attempt.Login. Ako pri všetkých šablónach, jej podoba sa z veľkej časti odvíja od potrieb správcu siete.

```
def match(self, event):
    if "Attempt.Login" in event["Category"]:
        if "Source" in event.keys():
            return tuple(event["Source"][0]["IP4"])
```

Ukážka 8.7: Implementácia metódy match() v jazyku Python (agregácia podľa zdrojovej IP adresy).

Ukážka 8.7 zobrazuje implementáciu metódy match() vo variante kde sa agreguje podľa cieľovej IP adresy. Kedže vo veľa prípadoch v hláseniach chýba údaj o zdroji, je potrebné pred vytvorením kľúča overiť prítomnosť položky Source v hlásení.

8.5 Agregácia hlásení Attempt.Exploit

Analýza ukázala, že všetký hlásenia aj ked z troch zdrojov sú takmer totožne až na anonymizáciu cieľov u dvoch zdrojov. Vďaka skoro až zanedbatelným odlišnostiam v hláseniach bude agregácia týchto hlásení pomerne jednoduchá.

Jednou z možných agregácii je agregácia podľa zdrojovej IP adresy. To znamená, že kľúčom rozlišujúcim jednotlivé kontexty bude zdrojová IP adresa. Týmto spôsobom sa dosiahne zlúčenie hlásení, ktoré informujú o útokoch z rovnakého zdroja. Exploit však môže byť orientovaný na slabiny v rôznych službách. Ak by správca siete mal záujem agregovať hlásenia nielen podľa zdrojovej adresy ale aj na základe toho pre akú službu je exploit určený, musel by kľúč kontextu zostaviť nielen zo zdrojovej adresy, ale aj z cieľového čísla portu, prípadne z názvu protokolu.

Metódu pre agregáciu hlásení kategórie Attempt.Exploit reprezentuje trieda AttemptExploit. Trieda AttemptExploit je odvodená od triedy SimpleAggregator dostupnej z navrhnutého frameworku a reimplementuje metódu match().

```
{
  "Category": "lambda a,b: [\"Attempt.Exploit\"]",
  "AggrID": "append_self_id",
  "DetectTime": "mintime",
  "EventTime": "mintime",
  "CeaseTime": "maxtime",
  "Description": "concat",
  "Note": "concat",
  "Source": {
    "IP4": "same",
    "Port": "merge",
    "@default": "omit"
  },
  "Target": {
    "IP4": "same_if_present",
    "Port": "merge",
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
```

```
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
  },
  "Attach":"merge",
  "@default": "omit"
}
```

Ukážka 8.8: Šablóna pre agregáciu hlásení kategórie Attempt.Exploit.

Ukážka 8.8 zobrazuje mnou navrhnutú šablónu pre agregáciu hlásení kategórie Attempt.Exploit. Kedže hlásenia zo všetkých zdrojov majú takmer rovnakú štruktúru a relatívne málo položiek, volil som jednoduchú šablónu s pravidlami len pre tie položky, ktoré hlásenia obsahujú. V prípade potreby samozrejme správca siete môže použiť rozsiaheljejšiu šablónu tak ako som to naznačil v návrhu šablón pre iné kategórie. Rozsiahlejšia šablóna môže obsahovať pravidlá aj pre položky, ktoré by sa mohli vyskytúť v potencionálnych nových hláseniach.

8.6 Agregácia hlásení Fraud.Copyright

Analýza ukázala, že hlásenia kategórie Fraud.Copyright pochádzajú iba z jedného zdroja. No ako som načrtol, aj napriek tomu tu existuje potenciál na agregáciu týchto hlásení.

Štruktúra hlásení je pomerne jednoduchá. Skladá sa iba z časových údajov, textovej poznámky, identifikácie zdroja (IP adresa a port) prílohy (informácia o nelegálnom obsahu) a identifikácie systému, ktorý toto hlásenie vytvoril. Pri tak jednoduchej štruktúre hlásení sa ponúka iba jedna možnosť ako agregovať tieto hlásenia a to podľa zdrojovej adresy.

Metódu pre agregáciu hlásení kategórie FraudCopyright reprezentuje trieda FraudCopyright. Trieda FraudCopyright je odvodená od triedy SimpleAggregator dostupnej z navrhnutého frameworku a reimplementuje metódu match().

```
{
  "Category": "lambda a,b: [\"Fraud.Copyright\"]",
  "AggrID": "append_self_id",
  "DetectTime": "mintime",
  "Note": "concat",
  "Source": {
    "IP4": "same",
    "Port": "merge",
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
  },
  "Attach":"merge",
  "@default": "omit"
}
```

Ukážka 8.9: Šablóna pre agregáciu hlásení kategórie Fraud.Copyright.

Ukážka 8.9 zobrazuje jednu z možných šablón pre agregáciu hlásení kategórie Fraud.Copyright. Vďaka jednoduchej štruktúre hlásení tejto kategórie som zvolil pomerne

jednoduchú a dá sa povedať aj základnú šablónu, ktorá zahŕňa pravidlá len pre tie položky, ktoré sa v dostupných hláseniach vyskytujú. Opäť o rozsiahlosti šablóny platí to isté, čo som naznačil pri popise šablóny pre agregáciu hlásení kategórie Attempt.Exploit.

8.7 Agregácia hlásení Vulnerable.Config

V analýze som v skratke popísal povahu hlásení kategórie Vulnerable.Config a tiež som načrtol problém rozmanitosti jednotlivých hlásení.

V prípade tejto kategórie hlásení je možností agregácie hneď niekoľko. Prvou najzákladnejšou možnostou je agregácia iba podľa zdrojovej adresy. Ďalšími možnosťami je agregácia podľa nie len zdrojovej adresy ale aj kombinácie zdrojovej adresy s rôznymi parametrami, ktoré identifikujú konkrétny druh zraniteľnosti. To však už závisí na konkrétnej situácii a na potrebách správcu siete. Preto popíšem len spomínaný základný spôsob agregácie.

Metódu pre agregáciu hlásení kategórie Vulnerable.Config reprezentuje trieda VulnerableConfig. Trieda VulnerableConfig je odvodená od triedy SimpleAggregator dostupnej z navrhnutého frameworku a reimplementuje metódu match().

```
{
  "Category": "lambda a,b: [\"Vulnerable.Config\"]",
  "AggrID": "append_self_id",
  "DetectTime": "mintime",
  "EventTime": "mintime",
  "CeaseTime": "maxtime",
  "Description": "concat",
  "Note": "concat",
  "Source": {
    "IP4": "same",
    "Port": "merge",
    "Proto": "same_if_present",
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
  },
  "Attach":"merge",
  "@default": "omit"
}
```

Ukážka 8.10: Šablóna pre agregáciu hlásení kategórie Vulnerable.Config.

Ukážka 8.10 zobrazuje mnou navrhnutú šablónu pre najzákladnejšiu formu agregácie hlásení kategórie Vulnerable.Config. Je orientovaná iba na aktuálnu podobu hlásení, nie je pripravená na potencionálne nové hlásenia s odlišnou štruktúrou. Príklad univerzálnejšej šablóny pokrývajúcu pravidlá takmer pre všetky položky z dostupnej špecifikácie formátu IDEA som ukázal v návrhu agregačnej metódy pre hlásenia kategórie Recon.Scanning.

8.8 Agregácia hlásení Intrusion.Botnet

V analýze hlásení spadajúcich do kategórie Intrusion.Botnet som popísal odlišnosť medzi jednotlivými hláseniami. Tiež som popísal problém so zlým umiestnením adresy kontrolného servera a dva typy hlásení spadajúce do kategórie Intrusion.Botnet.

Vzhľadom na to, že spomenuté dva typy hlásení si vyžadujú rozdielne kľúče pre identifikáciu kontextov, ponúka sa možnosť vytvoriť dve samostatné agregačné metódy. Pri tvorbe šablóny som však zistil, že šablóny pre tieto dva typy agregácii sú zhodné. Preto som sa rozhodol tieto typy zastrešiť jednou agregačnou metódou. Metódu pre agregáciu hlásení kategórie Intrusion.Botnet reprezentuje trieda IntrusionBotnet. Trieda IntrusionBotnet je odvodená od triedy SimpleAggregator dostupnej z navrhnutého frameworku a reimplementuje metódu match().

```
def match(self, event):
    if "Intrusion.Botnet" in event["Category"] and
       "Malware" in event["Category"]:
        tuple(event["Source"][0][IP4] + [event["Source"]["Note"]])
    elif "Intrusion.Botnet" in event["Category"]:
        tuple(event["Source"][0][IP4] + event["Source"][0][Type] +
              event["Target"][0][IP4] + event["Target"][0][Type])
```

Ukážka 8.11: Implementácia metódy match() triedy IntrusionBotnet v jazyku Python.

Ukážka 8.11 zobrazuje implementáciu metódy match() triedy IntrusionBotnet. Implementácia je realizovaná spôsobom, ktorý som popísal vyššie. Teda, že hoci ide o dva typy hlásení, tak obdiva zastrešuje jedna agregačná metóda. V tomto prípade metóda match() identifikuje jeden z dvoch možných typov hlásení a na základe toho vygeneruje príslušný kľúč. V prvom príklade kľúč pozostáva zo zdrojovej adresy a položky Note v Source, ktorá popisuje typ Malware. To znamená, že agregácia týchto hlásení prebieha na základe zdrojovej IP adresy a názvu malware. V druhom prípade kľúč pozostáva z kombinácie zdrojovej a cieľovej IP adresy a k nim príslušných položiek Type. To znamená, že sa agregujú len hlásenia, ktoré informujú o pripojení tej istej adresy k rovnakému kontrolnému serveru.

```
{
  "Category": "lambda a,b: [\"Intrusion.Botnet\"]",
  "AggrID": "append_self_id",
  "DetectTime": "mintime",
  "EventTime": "mintime",
  "CeaseTime": "maxtime",
  "Description": "concat",
  "Note": "concat",
  "Source": {
    "Type": "same_if_present",
    "IP4": "same",
    "Port": "merge",
    "Note": "concat",
    "@default": "omit"
  },
  "Target": {
    "Type": "same_if_present",
    "IP4": "same",
    "Port": "merge",
    "Note": "concat",
```

```
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
  },
  "Attach":"merge",
  "@default": "omit"
}
```

Ukážka 8.12: Šablóna pre agregáciu hlásení kategórie Intrusion.Botnet.

Ukážka 8.12 zobrazuje šablónu pre agregáciu hláseni kategórie Intrusion.Botnet.

8.9 Agregácia hlásení Abusive.Spam

Pri tak jednoduchej štruktúre akú majú hlásenia kategórie Abusive.Spam nie je príliš veľa spôsobov agregácie týchto hlásení. Sú dve možnosti. Prvou je agregácia na základe zdrojovej IP adresy. Druhou možnosťou je agregácia na základe zdrojovej IP adresy a bližšej špecifikácie hlásenia v poli Description.

Metódu pre agregáciu hlásení kategórie Abusive.Spam reprezentuje trieda AbusiveSpam. Trieda AbusiveSpam je odvodená od triedy SimpleAggregator dostupnej z navrhnutého frameworku a reimplementuje metódu match().

```
{
  "Category": "lambda a,b: [\"Abusive.Spam\"]",
  "AggrID": "append_self_id",
  "DetectTime": "mintime",
  "EventTime": "mintime",
  "CeaseTime": "maxtime",
  "Description": "concat",
  "Source": {
    "IP4": "same",
    "Proto": "same_if_present",
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
  },
  "Attach":"merge",
  "@default": "omit"
}
```

Ukážka 8.13: Šablóna pre agregáciu hlásení kategórie Abusive.Spam.

Ukážka 8.13 zobrazuje šablónu pre agregáciu hlásení kategórie Abusive.Spam.

8.10 Korelácia hláseni Recon.Scanning a Attempt.Login

Agregáciu hlásení kategórie Recon.Scanning a taktiež agregáciu hlásení kategórie Attempt.Login som popísal v kapitolách 8.2 a 8.4. Histogramy 7.1 a 7.2 ukazujú, že hlásenia týchto dvoch kategórii majú v niektorých prípadoch rovnakú zdrojovú adresu. To môže znamenať, že tieto hlásenia na seba nejakým spôsobom nadväzujú. V prípade pokusov o prihlásenie je možné, že útočník pred útokom vykoná prípravnú fázu kedy sa pokúsi skenovaním portov zistiť, ktoré porty resp. služby sú otvorené. Ak sa teda vyskytnú hlásenia týchto dvoch kategórii s rovnakou zdrojovou IP adresou v dostatočne malom časovom rozostupe, je veľmi pravdepodobné, že spolu súvisia. Preto môže byť výhodné o tejto skutočnosti informovať formou hlásenia. V prípade dostupných vzoriek analýza ukázala, že hlásenia kategórie Attempt.Login obsahujú zdrojovú IP adresu len v zhruba 1/4 prípadov. To pravdepodobne výrazne redukuje počet nájdených korelácii. Aj napriek tejto skutočnosti však predstavím metódu, ktorá je založená na zdrojovej IP adrese a to z toho dôvodu, že iný spôsob, ktorý by dokázal medzi týmito kategóriami hlásení nájsť koreláciu neexistuje.

Na rozdiel od agregácie hláseni jednej kategórie, pri vytváraní hlásenia, ktoré poskytuje novú informáciu o nájdení korelácie medzi rôznymi kategóriami hlásení neexistuje spôsob akým by sa dali spojiť hlásenia rôznych kategórii tak aby boli zachované všetky dôležité informácie. Preto pri nájdení korelácie medzi správami korelačná metóda vygeneruje hlásenie, ktoré iba informuje o korelácii s odkazom na príslušné hlásenia.

Korelačná metóda však nemôže byť vystavaná nad triedou SimpleAggregator. V prípade, že pri uzatváraní kontextu kontext neobsahuje všetky kategórie hlásení, medzi ktorými sa hľadá korelácia, znamená to, že sa žiadna korelácia nenašla a z uzatváraného kontextu nemôže byť vygenerované hlásenie. Vzhľadom na tom, že trieda SimpleAggregator pri uzatváraní kontextu automaticky generuje nové hlásenie, musíme túto triedu mierne upraviť.

Pre potreby korelácie som navrhol triedu SimpleCorrelator, ktorá dedí triedu SimpleAggregator a reimplementuje metódu ctx_close(), ktorá uzatvára kontext a jej výsledkom je nové hlásenie.

```
class SimpleCorrelator(Aggregator.SimpleAggregator):

    def __init__(self, _templatepath = '', _timeout = None,
                 _count = None, _cats = []):
        Aggregator.SimpleAggregator.__init__(_templatepath,
                                             _timeout, _count)
        self.cats = _cats

    def ctx_close(self, ctx):

        for cat in self.cats:
            if cat not in ctx["cats"]:
                return None

        return ctx['merged_event']
```

Ukážka 8.14: Implementácia triedy SimpleCorrelator v jazyku Python.

Ukážka 8.14 zobrazuje implementáciu triedy SimpleCorrelator. Z ukážky som pre prehľadnosť odstránil niektoré implementačné detialy. Jediným rozdielom oproti pôvodnej metóde ctx_close() je to, že táto metóda pri uzatváraní kontextu kontroluje či sú v kontexte

obsiahnuté všetky kategórie hlásení potrebné na vytvorenie korelácie. Framework v kontexte udržuje zoznam prijatých kategórii a na záver sa porovnáva so zoznamom kategórii určenom parametrom konštruktora _cats.

Metódu pre koreláciu hlásení kategórie Recon.Scanning a Attempt.Login reprezentuje trieda ReconScanningAttemptLogin. Trieda ReconScanningAttemptLogin je odvodená od triedy SimpleCorrelator dostupnej z navrhnutého frameworku a reimplementuje metódu match().

```
import Correlator

class ReconScanning(Correlator.SimpleCorrelator):

    def __init__(self, _templatepath = '', _timeout = None,
                 _count = None):
      Correlator.SimpleCorrelator.__init__(self, _templatepath,
                                   _timeout, _count,
                                   ["Recon.Scanning","Attempt.Login"])

    def match(self, event):
        if "Recon.Scanning" in event["Category"] or
           "Attempt.Login" in event["Category"]:
            return tuple(event["Source"][0]["IP4"])
```

Ukážka 8.15: Implementácia triedy ReconScanningAttemptLogin v jazyku Python.

Ukážka 8.15 zobrazuje implementáciu triedy ReconScanningAttemptLogin. V ukážke sú vynechané niektoré implementačné detaily. Princíp vytvárania nových tried pre korelačné metódy je veľmi podobný tomu pre vytváranie tried pre agregačné metódy. Pri vytváraní triedy pre korelačnú metódu je nutné definovať zoznam kategórii, ktoré tvoria koreláciu. Metóda match() je stále založená na rovnakom princípe. V prípade korelácie je však typické, že pracuje viac ako s jednou kategóriou hlásení. V tejto ukážke je zobrazený prípad, kedy metóda vyhľadáva koreláciu medzi hláseniami kategórii Recon.Scanning a Attempt.Login len na základe zdrojovej IP adresy a samozrejme v určitom časovom rozmedzí.

```
{
  "Category": "lambda a,b: [\"Recon.Scanning\", \"Attempt.Login\"]",
  "CorrelID": "append_self_id",
  "DetectTime": "mintime",
  "EventTime": "mintime",
  "CeaseTime": "maxtime",
  "Description": "lambda a,b: \"Correlation:\" +
     str(Conversions.ip_bin2ipaddress(CURRENTEVENT[\"Source\"][0][\"IP4\"])) +
     \"performed port scans followed by login attempts.\"",
  "Note": "See original alerts referenced by CorrelID for more information",
  "Source": {
    "IP4": "same",
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
```

```
  },
  "@default": "omit"
}
```

Ukážka 8.16: Šablóna pre koreláciu hlásení kategórii Recon.Scanning a Attempt.Login.

Ukážka 8.16 zobrazuje mnou navrhnutú šablóne pre koreláciu hlásení kategórii Recon.Scanning a Attempt.Login. Hlásenia vytvorené pomocou tejto šablóny majú iba informatívny charakter. Pri tejto šablóne je nutné popísať niekoľko skutočností. Niektoré položky, ktoré sú uvedené v šablóne sa vôbec nevyskytujú v prichádzajúcich hláseniach. No aj napriek tomu je potrebné takéto položky do výsledného hlásenia pridať a prípadne ich pri každom novom hlásení v danom kontexte aktualizovať pomocou príslušnej operácie. Z tohto dôvodu som do navrhnutého frameworku zaviedol logiku vyhodnocovania hlásení tak, že na začiatku pri vytváraní kontextu sú do kontextu pridané všetky položky podľa šablóny. Ak sa niektoré položky zo šablóny v hlásení, ktoré vyvolalo vytvorenie kontextu nenachádzajú, ich hodnota je nastavená na None. Následne sa každá položka v takto vytvorenom kontexte vyhodnotí pomocou príslušnej operácie definovanej v šablóne. Na začiatku pri vytváraní kontextu však operácie, ktoré vyžadujú dva vstupné parametre samozrejme nemajú k dispozícii druhý parameter. Tieto operácie a vôbec všetky operácie sú na tento prípad úspôsobené. Ak je aspoň jeden z parametrov None, operácia sa nevykoná. Vyhodnocovanie hodnôt jednotlivých položiek však musí byť vykonané už pri výskyte prvého hlásenia hoci v tom čase ešte neexistuje žiadne ďalšie hlásenie, s ktorým by mohlo byť spojené a aplikácia operácii, ktoré sú určené na spájanie hlásení nie je uplatnená. A to z toho dôvodu, že operácie, ktoré pracujú iba s jedným parametrom, alebo lambda funkcie, ktoré nemusia pracovať so žiadnym parametrom musia byť aplikované už pri prvom hlásení.

Pri pridávaní hlásení do kontextov sa aktualizujú všetky položky, ktoré obsahuje práve spracovávané hlásenie a ktoré sú zároveň popísané šablónou. Položky, ktoré sa nevyskytujú v práve spracovávanom hlásení sú následne aktualizované tiež. V prípade, že ich aktualizácia vyžaduje aplikáciu operácie, ktorá vyžaduje dva parametre, druhým parametrom je hodnota None. Týmto docielime to, že sa pri každom novom hlásení aktualizujú aj položky, ktorých aktualizácia nevyžaduje novú hodnotu z príchodzieho hlásenia ale je nutné aby táto položka bola aktualizovaná pri každom novom hlásení. Takouto položkou môže byť napríklad čítať aktualizovaný pomocou lambda funkcie. Tiež sa aktualizujú položky, ktorých aktualizácia vyžaduje hodnotu nejakej inej položky z nového hlásenia. Napríklad operácia append_self_id musí byť aplikovaná aj v prípade, že položka, do ktorej táto operácia pridáva nové údaje sa nevyskytuje v prichádzajúcom hlásení. Niktorým položkám, ktoré sú pridané do kontextu už pri prijatí úvodného hlásenia nemusí byť nikdy pridelená žiadna hodnota. Takéto položky je zbytočné uvádzať vo výslednom hlásení. Preto sa pri uzatváraní kontextu z výsledného hlásenia odstránia.

Zvláštnu pozornosť je tiež potrebné venovať aktualizácii položiek, ktoré predstavujú zoznam. Takýmito položkami sú napríklad položky Source, Target a iné zo špecifikácie formátu IDEA. Položky predstavujúce zoznam sa aktualizujú tak, že pri pridávaní nového objektu do zoznamu sa aktualizujú položky iba v danom objekte. V prípade, že sa v zozname nájde objekt, s ktorým je nový objekt zlúčitelný, zlučí sa a následne sa v zlúčenom objekte aktualizuju ostatné položky, ktoré neboli predmetom zlučovania ale sú obsiahnuté v šablóne. Položky v ostatných objektov sa neaktualizujú. Týmto sa dosiahne stav kedy môžeme do každého objektu zaviesť pomocou lambda funkcie napríklad čítače, ktoré budú zachytávať počet objektov, ktoré boli do tohto jedného objektu zlúčené.

V šablóne na ukážke 8.16 je v prípade položky Description definovaná lambda funkcia,

ktorá využíva niekoľko prostriedkov navrhnutého frameworku. Prvým je premenná CURRENTEVENT, ktorá obsahuje aktuálne hlásenie v podobe dátového typu slovník jazyka Python. Druhým je balík Conversions, ktorý som napísal pre potreby implementácie frameworku a ktorý je dostupný priamo z neho. Balík Conversions obsahuje funkcie pre konverziu časových údajov a IP adries do a z rôznych formátov.

V tejto kapitole som popísal návrh triedy SimpleCorrelator pre koreláciu hlásení o bezpečnostných udalostiach. Rovnako som popísal aj návrh konkrétnej korelačnej metody a šablóny pre koreláciu hlásení kategórii Recon.Scanning a Attempt.Login. Tiež som popísal vlastnosti navrhnutého frameworku, ktoré využíva trieda reprezentujúca metódu pre koreláciu hlásení spomínaných dvoch kategórii hlásení.

8.11 Korelácia hláseni Recon.Scanning a Attempt.Exploit

Agregáciu hlásení kategórie Recon.Scanning a taktiež agregáciu hlásení kategórie Attempt.Exploit som popísal v kapitolách 8.2 a 8.5. Histogramy 7.1 a 7.2 ukazujú, že hlásenia týchto dvoch kategórii majú v niektorých prípadoch rovnakú zdrojovú IP adresu. To môže znamenať, že tieto hlásenia na seba nejakým spôsobom nadväzujú. V prípade vykonávania exploitov môže byť v prípade, že útočník nemá vedomosť o všetkých aktívnych službách na zariadení výhodné vykonať prípravnú fazu. Prípravna fáza spočíva v skenovaní portov na adrese daného zariadenia. Tým útočník môže odhaliť, ktoré všetky služby sú na zariadení spustené a odhaliť tak ďalšie príležitosti na vykonanie exploitov. Ak sa teda vyskytnú hlásenia týchto dvoch kategórii s rovnakou zdrojovou IP adresou v dostatočne malom časovom rozostupe, je veľmi pravdepodobné, že spolu súvisia.

Metódu pre koreláciu hlásení kategórie Recon.Scanning a Attempt.Exploit reprezentuje trieda ReconScanningAttemptExploit. Trieda ReconScanningAttemptExploit je odvodená od triedy SimpleCorrelator dostupnej z navrhnutého frameworku a reimplementuje metódu match(). Metóda je prakticky totožná s metódou pre koreláciu hlásení kategórie Recon.Scanning a kategórie Attempt.Login.

```
{
  "Category": "lambda a,b: [\"Recon.Scanning\", \"Attempt.Exploit\"]",
  "CorrelID": "append_self_id",
  "DetectTime": "mintime",
  "EventTime": "mintime",
  "CeaseTime": "maxtime",
  "Description": "lambda a,b: \"Correlation:\" +
     str(Conversions.ip_bin2ipaddress(CURRENTEVENT[\"Source\"][0][\"IP4\"])) +
     \"performed port scans followed by exploit.\"",
  "Note": "See original alerts referenced by CorrelID for more information",
  "Source": {
    "IP4": "same",
    "@default": "omit"
  },
  "Node": {
    "Type": "merge",
    "SW": "merge",
    "Name": "same",
    "@default": "omit"
  },
  "@default": "omit"
```

}

Ukážka 8.17: Šablóna pre koreláciu hlásení kategórii Recon.Scanning a Attempt.Exploit

Ukážka 8.17 zobrazuje mnou navrhnutú šablónu pre koreláciu hlásení kategórii Recon.Scanning a Attempt.Exploit. Šablóna sa principiálne v ničom nelíši od šablóny pre koreláciu hlásení Recon.Scanning a Attempt.Login.

Kapitola 9

Vyhodnotenie

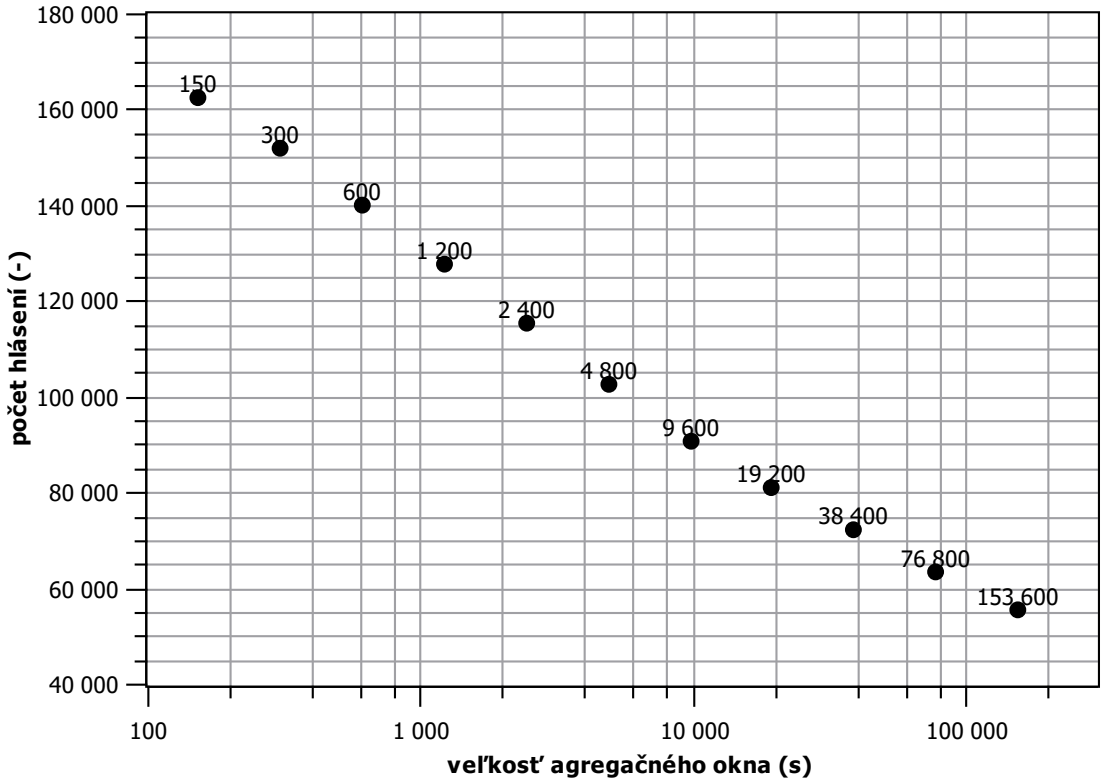
V tejto kapitole ukážem rozdiely medzi veľkosťou databázy pred aplikovaním agregačných a korelačných metód a po ich aplikácii. Ďalej vyhodnotím výkonnostné požiadavky frameworku. Konkrétne spotrebovanú pamäť a rýchlosť.

9.1 Vplyv veľkosti agregačného okna na veľkosť databázy

Aby som čo najlepšie ukázal zmenu veľkosti databázy, použijem hlásenia tých kategórii, ktoré majú v dostupných vzorkách dát najväčšie zastúpenie. Zmenu veľkosti databázy vyhodnotím v závislosti od veľkosti agregačného okna.

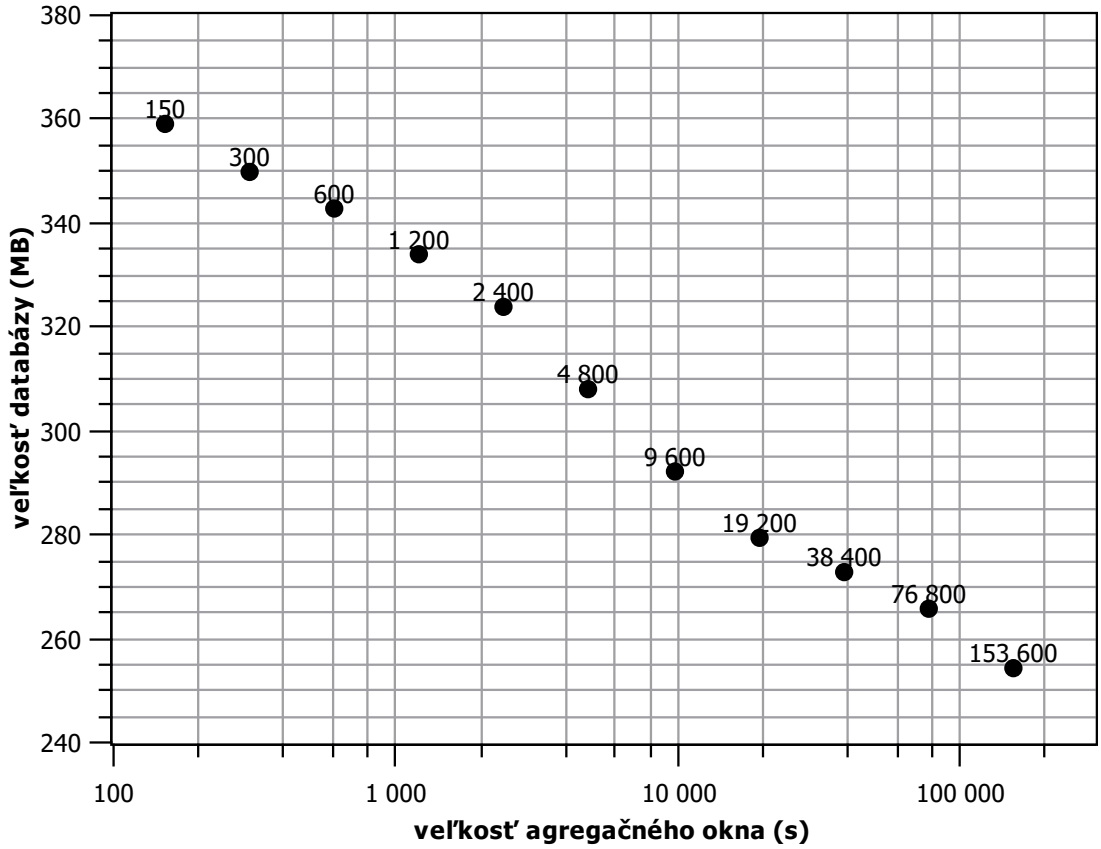
9.1.1 Recon.Scanning

Hlásení kategórie Recon.Scanning je v dostupných vzorkách dát jednoznačne najviac. Hlásenia tejto kategórie tvoria viac ako 90% všetkých hlásení. To predstavuje v prípade vzorky A približne 27 miliónov hláseni a v prípade vzorky B približne 32 miliónov hlásení. Vzhľadom na to, že pri vyhodnotení je potrebné spracovať tieto hlásenia niekoľkokrát, čas potrebný pre vyhodnotenie by bol neúnosný. Z toho dôvodu som sa rozhodol z hlásení kategórie Recon.Scanning vytvoriť vhodnú testovaciu vzorku, ktorá by skrátila čas vyhodnotenia na prijateľnú dobu. Testovacia vzorka obsahuje približne 2 milióny hlásení, ktoré sú vyfiltrované zo vzorky A na základe zdrojovej IP adresy. Do testovacej vzorky sú zaradené všetky hlásenia, ktoré majú v zdrojovej IP adrese 10,11,12,13,14 a 15 bit nastavený na jednotku. To predstavuje zhruba 1/64 z celkového počtu hlásení kategórie Recon.Scanning. Ak by som hlásenia vybral náhodne, ovplyvnilo by to agregovatelnosť hlásenia, pretože napríklad z dvoch agregovatelných hláseni by jedno mohlo byť pridané do vzorky a druhé nie. To by znamenalo, že by nedošlo k agregácii. Tým, že je kľučom k agregácii zdrojová IP adresa, rozhodol som sa vybrať podmnožinu IP adries a zahrnúť všetky správy z daných adries.



Obr. 9.1: Počet hlásení po aplikácii agregačnej metódy pre kategóriu hlásení Recon.Scanning v závislosti na veľkosti agregačného okna.

V grafe na obrázku 9.1 je zobrazený výsledný počet agregovaných hlásení v závislosti na veľkosti agregačného okna. Pri každej hodnote vykreslenej v grafe je uvedená príslušná veľkosť agregačného okna. Pôvodná vzorka hlásení pred aplikáciou agregačnej metódy obsahovala 2 064 315 hlásení. Už pri agregačnom okne nastavenom len na 150 sekúnd sa podarilo pôvodných 2 064 315 hlásení zredukovať len na 162 990 hlásení, teda len na 7,9%. Z grafu vyplýva, že čím väčšie je agregačné okno, tým je miera agregácie vyššia, ale tiež rastú nároky na zdroje (viď. ďalej). S exponenciálnym predlžovaním agregačného okna sa počet agregovaných hlásení znižuje približne lineárne. Konkrétna vhodná veľkosť agregačného okna sa v tomto prípade nedá veľmi dobre určiť.

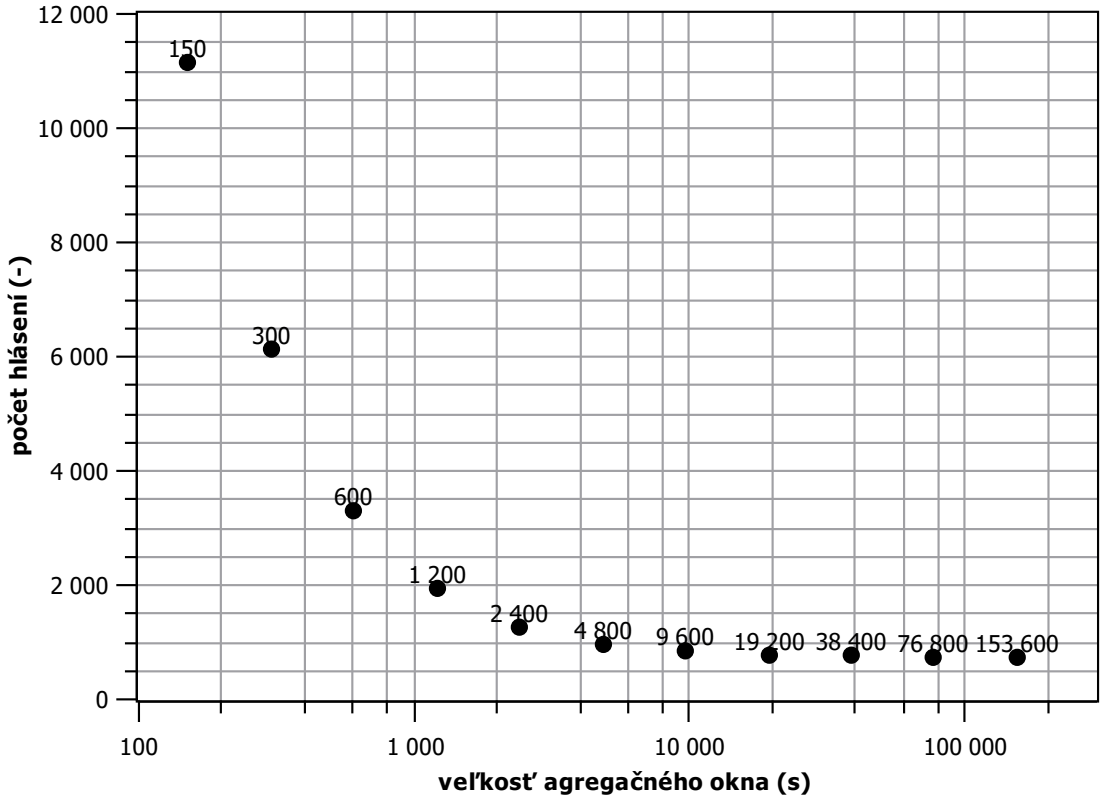


Obr. 9.2: Veľkosť databázy po aplikácii agregačnej metódy pre kategóriu hlásení Recon.Scanning v závislosti na veľkosti agregačného okna.

V grafe na obrázku 9.2 je zobrazená veľkosť výslednej databázy, ktorá vznikla aplikáciou agregačnej metódy na pôvodnú vzorku hlásení. Veľkosť databázy je zobrazená v závislosti na veľkosti agregačného okna. Veľkosť databázy sa tým zmenšila z pôvodných 4 000,8 MB na 359,2 MB. Ďalšie zväčšovanie agregačného okna dokáže veľkosť ďalej znížiť, aj keď nie už tak výrazne. Percentuálne zníženie veľkosti databázy nezodpovedá zníženiu počtu záznamov v databáze, pretože agregované hlásenia sú zvyčajne oveľa väčšie než hlásenia pôvodné.

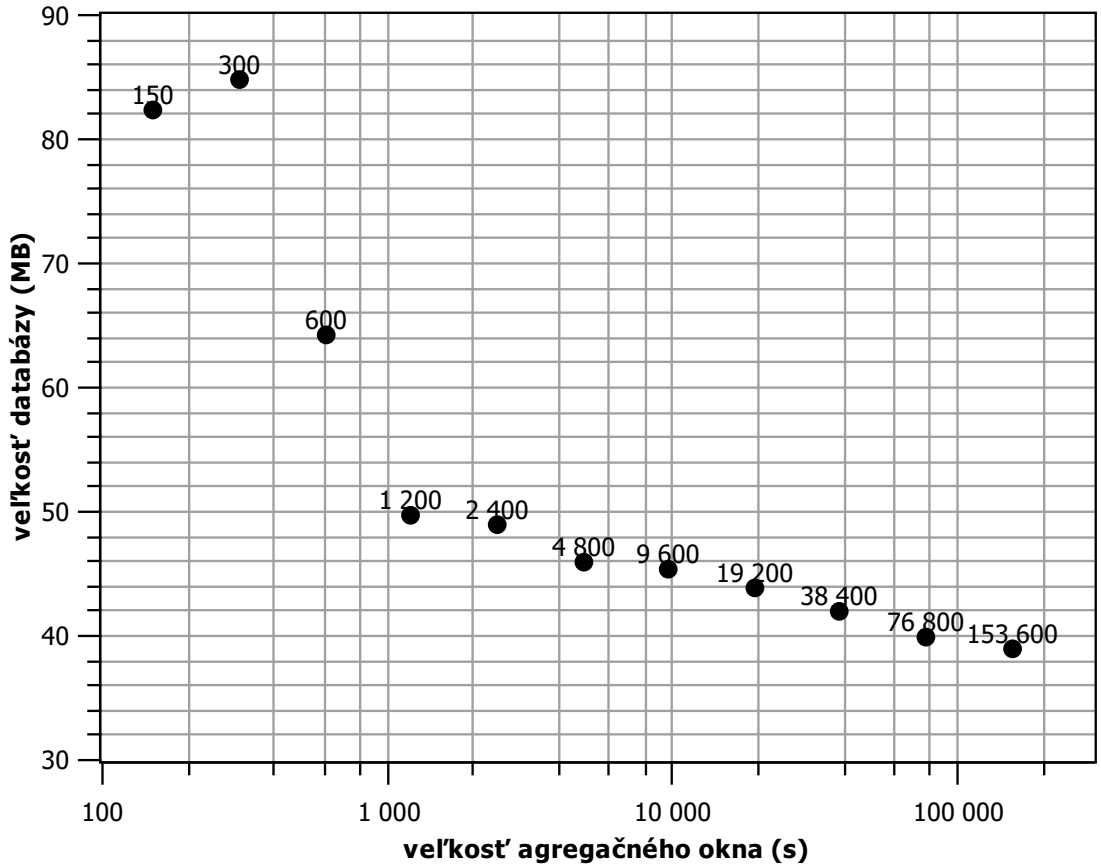
9.1.2 Fraud.Copyright

Hlásenia kategórie Fraud.Copyright majú v dostupných vzorkách hlásení jedno z najväčších zastúpení. Pre vyhodnotenie som použil hlásenia zo vzorky A, kde je ich výskyt početnejší. Vo vzorke A je 176 790 hlásení kategórie Fraud.Copyright.



Obr. 9.3: Počet hlásení po aplikácii agregačnej metódy pre kategóriu hlásení Fraud.Copyright v závislosti na veľkosti agregačného okna.

V grafe na obrázku 9.3 je zobrazený výsledný počet agregovaných hlásení v závislosti na veľkosti agregačného okna. Pôvodná vzorka hlásení pred aplikáciou agregačnej metódy obsahovala 176 790 hlásení. Rovnako ako v prípade agregácie hlásení kategórie Recon.Scanning, výraznú agregáciu hlásení som zaznamenal už pri agregačnom okne nastavenom len na 150 sekúnd. V prípade hlásení kategórie Fraud.Copyright možno v grafe na obrázku vidieť, že od bodu kedy je agregačné okno nastavené na 4 800 sekúnd (80 minút) a viac, sa zvyšovanie miery agregácie ustáli a počet výsledných hlásení sa už takmer nezmení.

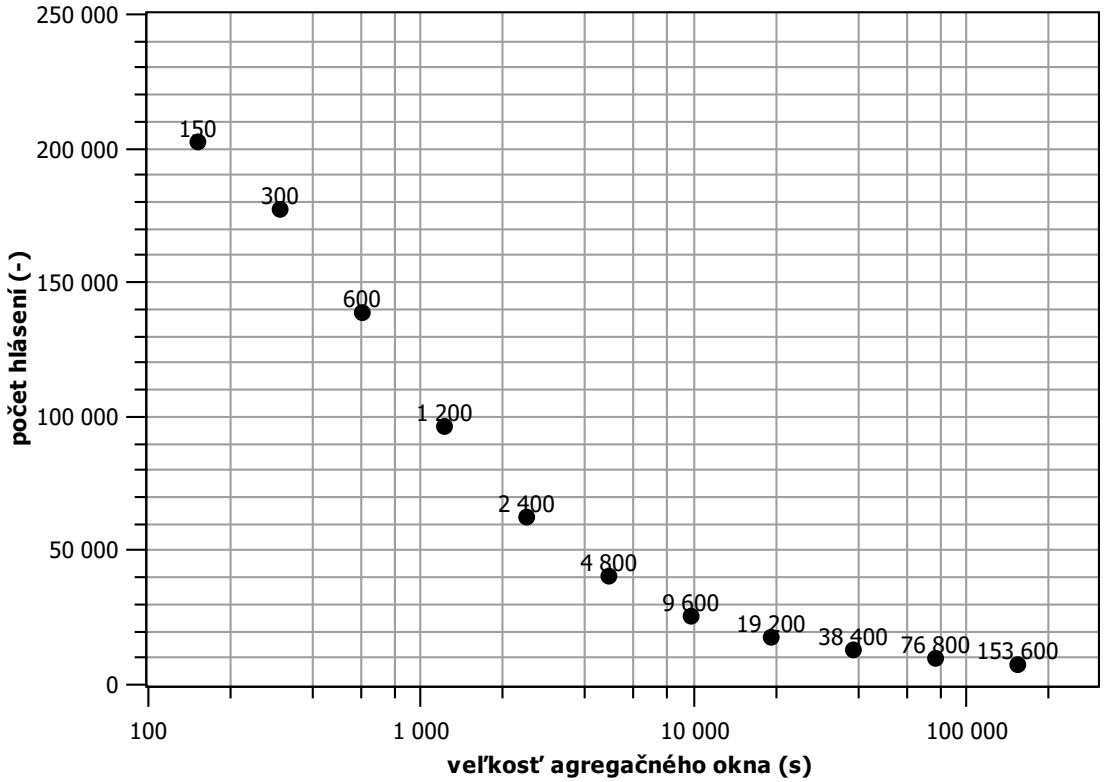


Obr. 9.4: Veľkosť databázy po aplikácii agregačnej metódy pre kategóriu hlásení Fraud.Copyright v závislosti na veľkosti agregačného okna.

V grafe na obrázku 9.4 je zobrazená veľkosť výslednej databázy, ktorá vznikla aplikáciou agregačnej metódy na pôvodnú vzorku hlásení. Veľkosť databázy je zobrazená v závislosti na veľkosti agregačného okna. Pôvodná vzorka hlásení má veľkosť 342,6 MB. Vzhľadom na to, že sa od určitej hranice už počet výsledných agregovaných hlásení už nemenil, rovnako sa ustálila aj výsledná veľkosť databázy. Z grafu možno pozorovať, že pri agregačnom okne nastavenom na 300 sekúnd je veľkosť databázy väčšia než v prípade agregačného okna nastaveného na 150 sekúnd. Kedže sa jedná o rozdiel iba niekoľko MB, bude to pravdepodobne zapríčinené spôsobom ukladania záznamov v Mongo databáze (zarovnanie, fragmentácia a pod.).

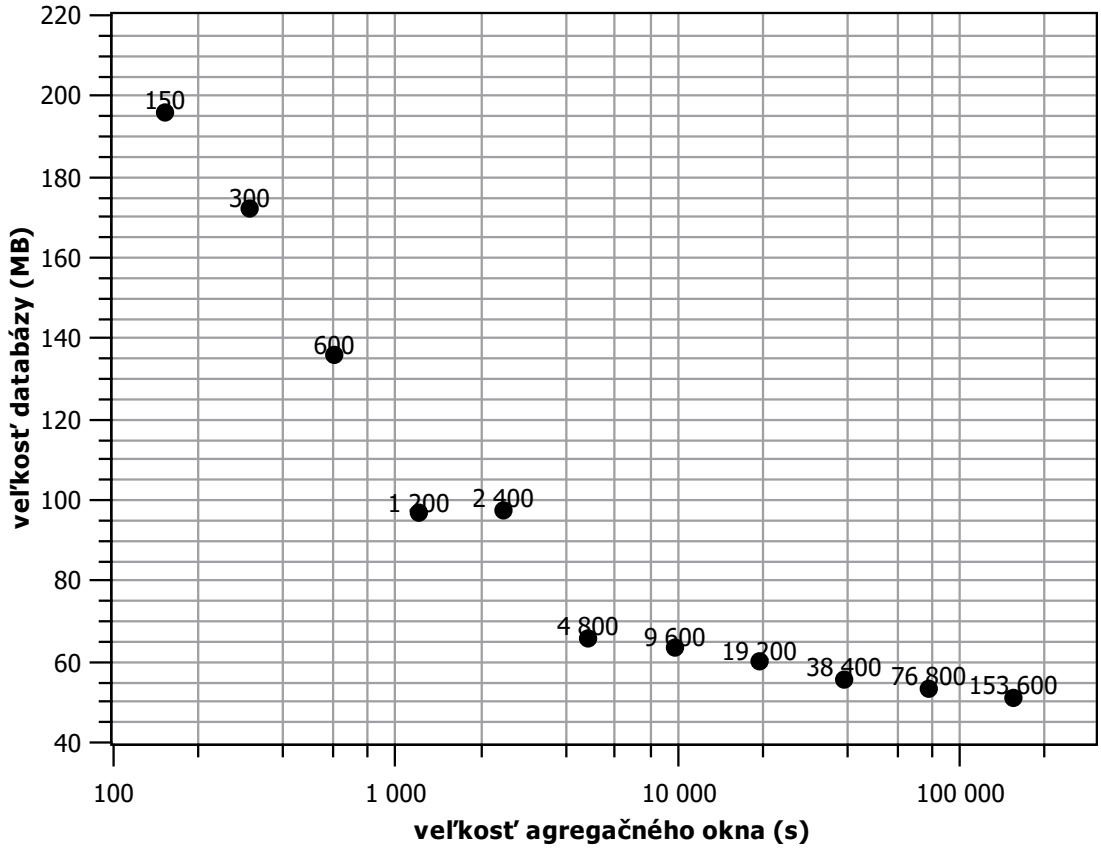
9.1.3 Attempt.Login

Pre vyhodnotenie agregačnej metódy pre hlásenia kategórie Attempt.Login som použil hlásenia zo vzorky B. Vo vzorke B sú k dispozícii hlásenia z viacerých zdrojov v celkovom počte 1 660 060. Z toho 438 847 obsahuje informáciu o zdroji útoku. Vzhľadom na to, že agregačná metóda agreguje hlásenia len na základe zdrojovej IP adresy, ako vzorku pre vyhodnotenie použijem len tie hlásenia, ktoré obsahujú informáciu o zdroji.



Obr. 9.5: Počet hlásení po aplikácii agregačnej metódy pre kategóriu hlásení Attempt.Login v závislosti na veľkosti agregačného okna.

V grafe na obrázku 9.5 je zobrazený výsledný počet agregovaných hlásení v závislosti na veľkosti agregačného okna. Pôvodná vzorka hlásení pred aplikáciou agregačnej metódy obsahovala 438 847 hlásení. Po agregácii s nastaveným agregačným oknom len na 150 sekúnd sa počet hlásení znížil až na polovicu, presne na 202 890 (46,2%) hlásení. Ďalšie zväčšovanie agregačného okna viedlo k omnoho väčšej agregácii hlásení.



Obr. 9.6: Veľkosť databázy po aplikácii agregačnej metódy pre kategóriu hlásení Attempt.Login v závislosti na veľkosti agregačného okna.

V grafe na obrázku 9.6 je zobrazená veľkosť výslednej databázy, ktorá vznikla aplikáciou agregačnej metódy na pôvodnú vzorku hlásení. Veľkosť databázy je zobrazená v závislosti na veľkosti agregačného okna. Veľkosť databázy sa z pôvodných 668 MB zmenšila na 196,2 MB už pri agregačnom okne nastavenom na 150 sekúnd. Od momentu, kedy bolo agregačné okno nastavené na viac ako 4 800 sekúnd sa už veľkosť databázy výrazne nemenila. Z grafu tiež možno vypozorovať, že v prípade agregačného okna nastaveného na 4 800 sekúnd je výsledná veľkosť databázy o niekoľko MB väčšia ako v prípade agregačného okna nastaveného len na 2 400 sekúnd. Túto anomáliu pripisujem rovnakým príčinám ako v prípade agregaácie hlásení kategórie Fraud.Copyright.

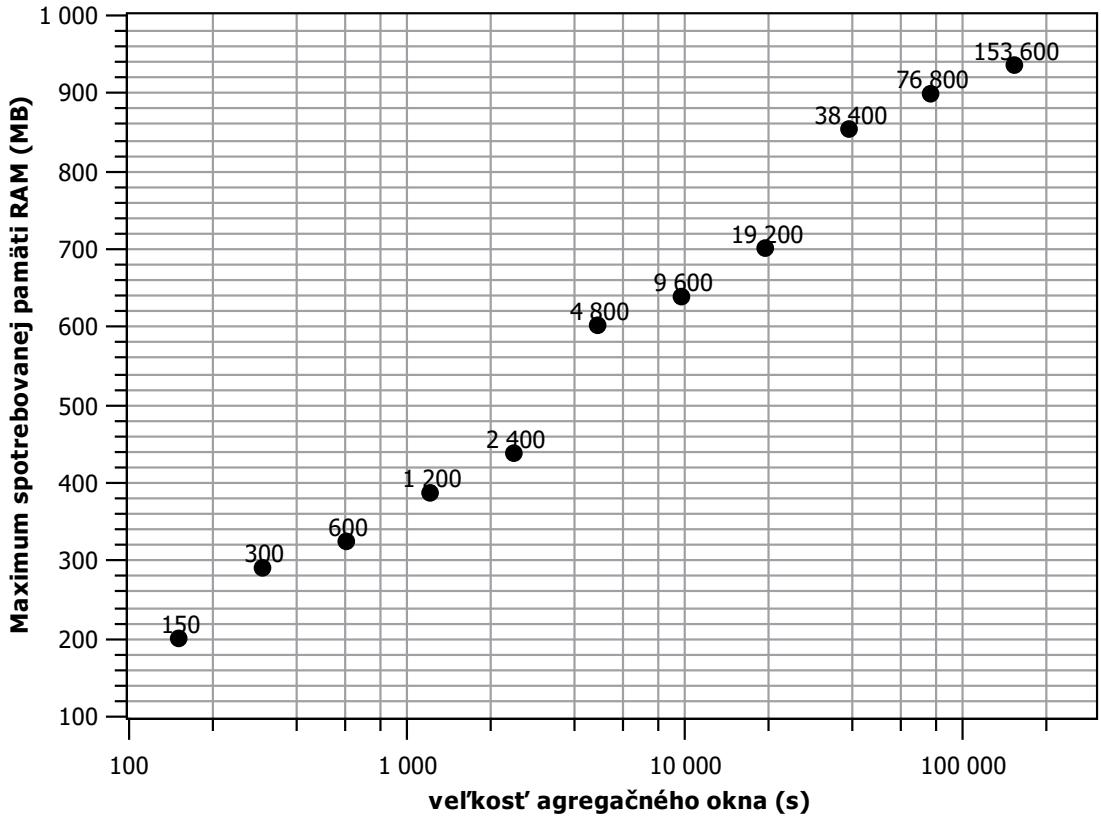
9.2 Spotreba pamäti

V tejto sekcii vyhodnotím pamäťovú náročnosť implementácie vybraných agregačných metód. V grafoch znázorním maximálnu spotrebovanú pamäť RAM procesom vykonávajúcim agregáciu v závislosti od veľkosti agregačného okna.

9.2.1 Recon.Scanning

V grafe na obrázku 9.7 je zobrazené maximum spotrebovanej operačnej pamäti v závislosti na veľkosti agregačného okna. Využitie pamäti je z časti závislé na povahe vstupných dát. Množstvo spotrebovanej operačnej pamäti je samozrejme závislé na množstve hlásení, ktoré je nutné spracovať za sekundu ale aj tiež od počtu vytvorených kontextov, ktoré je nutné vytvoriť na určitý počet hlásení. Graf na obrázku tiež ukazuje, že čím je väčšie agregačné okno, tým je množstvo spotrebovanej pamäti väčšie. To je spôsobené tým, že agregátor musí v jednom okamžiku uchovávať v pamäti kontexty z väčšieho časového úseku. Nárast veľkosti agregačného okna však nie je úmerný spotrebovanej operačnej pamäti. Je to spôsobené tým, že s veľkosťou agregačného okna úmerne nenarastá počet kontextov uchovávaných v operačnej pamäti. Agregátor je hlásenia čoraz viac schopný zaradiť do už existujúcich kontexotov.

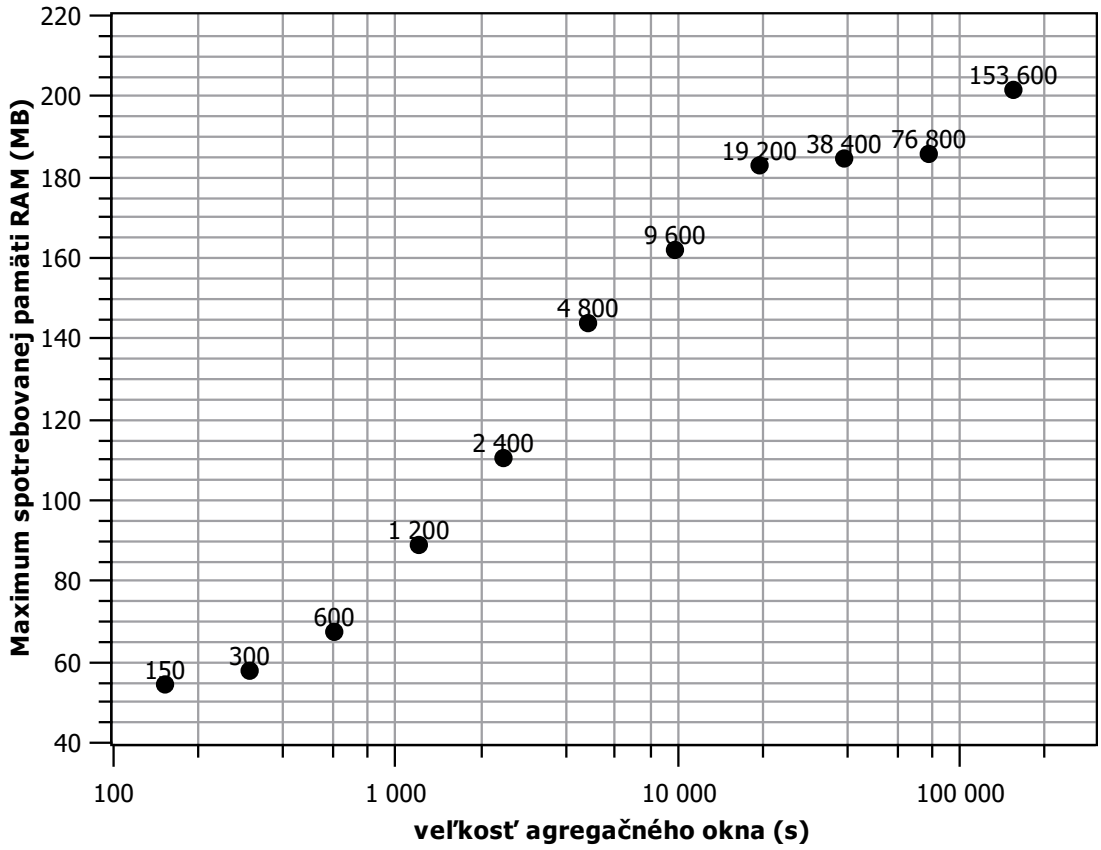
Pri krátkych agregačných oknách sa množstvo spotrebovanej pamäti RAM pohybuje v rádoch stoviek MB a pri najväčšom testovanom agregačnom okne je menšie než 1 GB, čo je vzhľadom k možnostiam dnešných výpočtových prostriedkov úplne prijateľné. Musím však poznamenať, že uvedená maximálna spotreba operačnej pamäti je len orientačná. Uskutočnené meranie je totiž závislé na tom, akým spôsobom garbage collector jazyka Python uvoľňuje pamäť. Uvoľňovanie pamäti nie je úplne deterministické a opakované testy ukazujú výsledky, ktoré sa navzájom líšia až o 30%.



Obr. 9.7: Maximálna veľkosť využitej pamäti RAM v závislosti na veľkosti agregačného okna (Recon.Scanning).

9.2.2 Fraud.Copyright

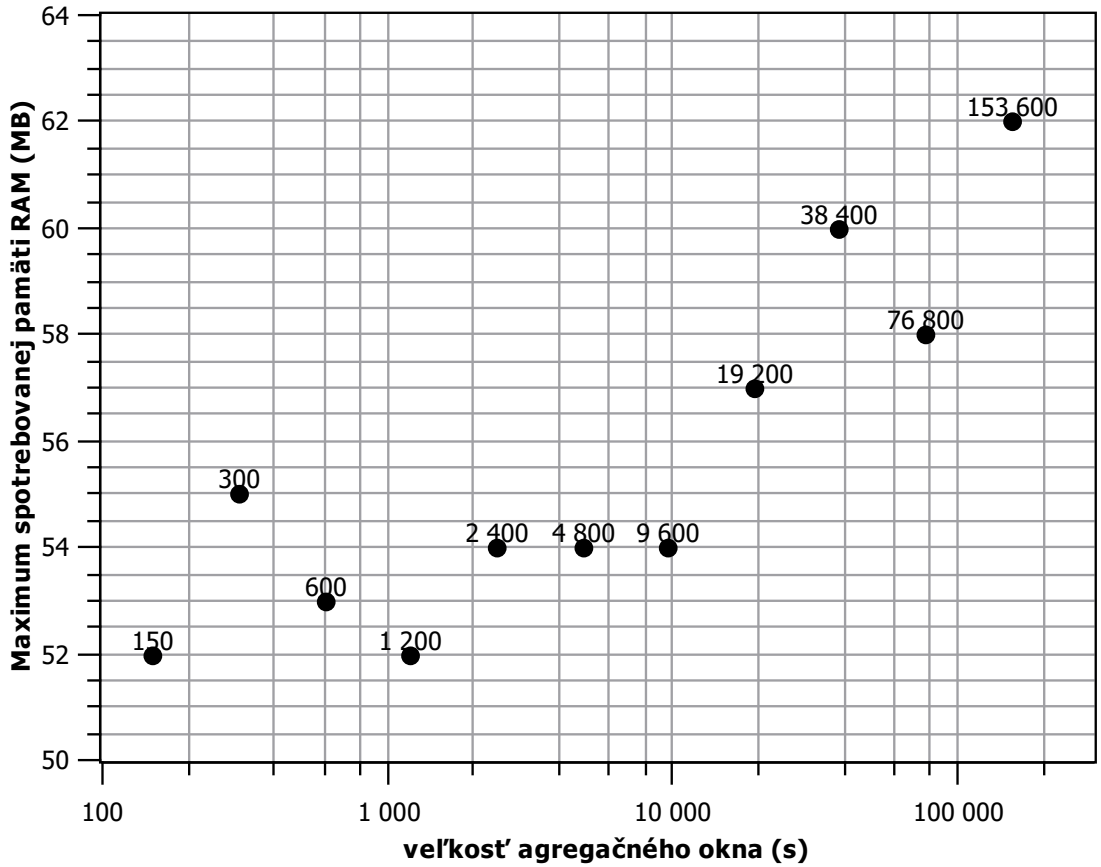
V grafe na obrázku 9.8 je zobrazené maximum spotrebovanej operačnej pamäti v závislosti na veľkosti agregačného okna. V tomto prípade je však veľkosť využitej operačnej pamäti podstane menšia ako v prípade agregácie hlásení kategórie Recon.Scanning. Je to spôsobené tým, že hlásenia kategórie Fraud.Copyright majú v určitom časovom intervale podstatne menší výskyt ako hlásenia kategórie Recon.Scanning. Tak isto ako v prípade agregácie hlásení kategórie Recon.Scanning nie je nárast spotreby operačnej pamäti úmerný zväčšovaniu agregačného okna.



Obr. 9.8: Maximálna veľkosť využitej pamäti RAM v závislosti na veľkosti agregačného okna. (Fraud.Copyright).

9.2.3 Attempt.Login

V grafe na obrázku 9.9 je zobrazené maximum spotrebovanej operačnej pamäti v závislosti na veľkosti agregačného okna. V tomto prípade sa pri použitý rôzne veľkých agregačných okien spotreba operačnej pamäti veľmi nelíšila.



Obr. 9.9: Maximálna veľkosť využitej pamäti RAM v závislosti na veľkosti agregačného okna (Attempt.Login).

9.3 Rýchlosť

Jedným z dôležitých kritérii pre použiteľnosť navrhnutých agregačných a korelačných metód a vôbec celého frameworku je ich rýchlosť. Framework musí byť dostatočne rýchly na to, aby dokázal spracovávať všetky hlásenia, ktoré obdrží systém Mentat. Zo štatistík vyplýva, že denne systém Mentat obdrží približne 1,2 milióna hlásení. Ztoho plynie požiadavka na spracovanie minimálne 14 hlásení za sekundu. Vyhodnotenie rýchlosti spracovania hlásení som vykonal na serveri benefizio na adrese benefizio.liberouter.org. Pre vyhodnotenie som použil vzorku 10 000 hlásení, ktorých spracovanie trvalo 9,930 sekundy. To predstavuje spracovanie približne 1 007 hlásení za sekundu. Pri vyhodnotení som použil vzorku hlásení uloženú na serveri. Diskové operácie spojené s čítaním jednotlivých hlásení však majú výrazný vplyv na rýchlosť spracovania hlásení. Pri získavaní hlásení priamo z počítačovej siete tento vplyv odpadá. Aj napriek pomerne pomalým diskovým operáciám však spracovanie 1 007 hlásení za sekundu poskytuje viac než dostatočnú rezervu.

Kapitola 10

Záver

Cieľom tejto práce bola analýza bezpečnostných hlásení zo systému Mentat vo formáte IDEA a návrh a implementácia metód pre agregáciu a koreláciu týchto hlásení. Myšlienku návrhu som prevzal z konceptu kontextov pre jednoduchú koreláciu udalostí nástrojom Simple Event Correlator (SEC). Základ aplikácie tvorí jednoduchý framework, ktorý poskytuje základnú funkcionalitu pre prácu s hláseniami a kontextami. Pre tento krok som sa rozhodol kvôli rozmanitosti bezpečnostných hlásení a veľkej variabilite formátu IDEA. Framework implementuje metódy pre vytváranie kontextov a pre pridávanie nových hlásení do kontextov. Pre odlišnosti medzi jednotlivými agregačnými a korelačnými metódami vo vytvárani kontextov a pridávaní IDEA hlásení do kontextov som navrhol systém šablón, ktorý umožňuje špecifikovať spôsob vytvárania kontextov a pridávania hlásení do kontextov. Vďaka systému šablón je teda navrhnutý framework schopný prispôsobiť činnosť funkcií pre vytváranie kontextov a pridávanie IDEA hlásení do kontextov. Pri návrhu a implementácii konkretných agregačných a korelačných metód postavených na tomto jednoduchom frameworku odpadá nutnosť reimplementácie spomínaných funkcií. Jednoducho stačí dodať frameworku sadu parametrov v podobe šablóny. Vo výsledku pri implementácii konkrétnej agregačnej alebo korelačnej metódy stačí doimplementovať spôsob vyhľadávania kontextov pre nové udalosti čím sa zjednoduší a zrýchli celý proces vývoja nových špecifických metód.

Vo vyhodnotení som ukázal význam agregácie hlásení o bezpečnostných incidentoch. V prípade hlásení kategórie Recon.Scanning už pri agregačnom okne nastavenom len na 150 sekúnd sa podarilo pôvodných 2 064 315 hlásení zredukovať len na 162 990 hlásení, teda len na 7,9%. To predstavuje redukciu veľkosti databázy z pôvodných 4 000,8 MB na 359,2 MB. Ďalej som tiež ukázal maximálnu spotrebu operačnej pamäti. Spotreba operačnej pamäti aj pri najväčšom testovanom agregačnom okne v prípade agregácie hlásení kategórie Recon.Scanning nepresiahla 1GB. Kedže hlásenia kategórie Recon.Scanning tvoria viac ako 90% zo všetkých hlásení, spotreba operačnej pamäti by aj pri agregácii všetkých kategórií hlásení súčasne nepresiahla 2GB, čo je na pomery dnešných výpočtových prostriedkov úplne prijateľné. Na záver som overil rýchlostné požiadavky. Zo štatistík vyplýva, že denne je potrebné spracovať približne 1,2 milióna hlásení. To predstavuje potrebu spracovať približne 14 hlásení za sekundu. Meraním som dosiahol rýchlosť spracovania 1 007 hlásení za sekundu.

Testy dokázali funkčnosť a užitočnosť navrhnutého frameworku v praxi. Framework bude integrovaný do systému Mentat prevádzkovaného združením CESNET, kde pomôže znížiť množstvo spracovávaných a ukladaných hlásení.

Literatúra

[1] McNab, C.: *Network Security Assessment*. O'Reilly, 2004, iSBN 0-596-00611-X.

[2] Rouillard, J. P.: Real-time log file analysis using the Simple Event Correlator (SEC). In *USENIX LISA*, USENIX, 2004.

[3] Sterling, B.: *The Hacker Crackdown: Law and Disorder on the Electronic Frontier*. Bantam Books, 1993, iSBN 0-553-56370-X.

[4] WWW stránky: Mentat. `https://mentat.cesnet.cz`, [cit. 2015-12-19].

[5] WWW stránky: Útok hrubou silou. `https://en.wikipedia.org/wiki/Brute-force_attack`, [cit. 2015-12-19].

[6] WWW stránky: Malware. `https://en.wikipedia.org/wiki/Malware`, [cit. 2015-12-20].

[7] WWW stránky: IDEA. `https://idea.cesnet.cz`, [cit. 2015-12-24].

[8] WWW stránky: Warden. `https://warden.cesnet.cz`, [cit. 2015-12-24].

[9] WWW stránky: IDEA specification. `https://idea.cesnet.cz/en/definition`, [cit. 2015-12-29].

[10] WWW stránky: Mentat sources. `https://csirt.cesnet.cz/cs/services/mentat`, [cit. 2015-12-31].

Prílohy

Zoznam príloh

A Obsah CD **58**

Príloha A

Obsah CD

Obsahom CD je priečinok src, v ktorom sa nachádzajú zdrojové texty frameworku. V priečinku src sa tiež nachádza priečinok paramfiles, v ktorom sú uložené šablóny pre agregáciu a koreláciu hlásení. Ďalej sa na CD nachádza priečinok doc, v ktorom sú uložené všetky zdrojové texty a prílohy potrebné pre preklad písomnej správy. Na CD je taktiež priložený súbor pisomnasprava.pdf, ktorého obsahom je písomná správa.