FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Systém pro evidenci podezřelých skupin síťových adres
Student:	Bc. Lenka Stejskalová
Vedoucí:	Ing. Tomáš Čejka
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Seznamte se s problematikou botnetů ve smyslu synchronizované koordinované množiny infikovaných strojů, které se zapojují do distribuovaných škodlivých aktivit.
Prozkoumejte systém pro modelování reputace podezřelých entit: Network Entity Reputation Database (NERD) [1], který obsahuje historii nahlášených bezpečnostních událostí.
Prozkoumejte hlášení o bezpečnostních událostech sdílených pomocí systému Warden [2] a jejich podmnožinu hlášenou open source systémem NEMEA [3,4].
Navrhněte systém/úložiště pro evidování skupin podezřelých entit, které jsou hlášeny společně v bezpečnostních hlášeních a nebo vykazují podobné chování.
Při návrhu uvažujte stárnutí dat a vývoj chování adres i skupin adres v čase.
Navržený systém implementujte.
Otestujte výkon vzniklého systému.

Seznam odborné literatury

[1] http://nerd.cesnet.cz
[2] https://warden.cesnet.cz
[3] https://github.com/CESNET/NEMEA
[4] T.Cejka, et al.: "NEMEA: A Framework for Network Traffic Analysis," in *12th International Conference on Network and Service Management (CNSM 2016)*, Montreal, Canada, 2016.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 30. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Systém pro evidenci podezřelých skupin síťových adres

Bc. Lenka Stejskalová

Katedra počítačových systémů
Vedoucí práce: Ing. Tomáš Čejka

6. května 2018

Poděkování

Ráda bych poděkovala svému vedoucímu diplomové práce Ing. Tomáši Čejkovi za odborné vedení, za pomoc a rady při zpracování této práce. Mé poděkování patří též celému týmu ve sdružení CESNET za spolupráci při vytváření systému pro evidování skupin podezřelých adres. Velké poděkování náleží mé rodině a přátelům za podporu, trpělivost a povzbuzování po dobu mého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 6. května 2018 .....

Odkaz na tuto práci

Stejskalová, Lenka. *Systém pro evidenci podezřelých skupin síťových adres.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá analýzou provozu jdoucího z podezřelých síťových adres a jejím rozdělováním těchto adres do skupin. Cílem práce je vytvořit systém pro evidování skupin podezřelých adres, které jsou hlášeny společně v bezpečnostních hlášeních nebo vykazují podobné chování. V práci je provedena analýza vstupních dat a analýza systémů NERD, NEMEA a Warden. Práce se zabývá definováním botnetu a rozdělováním útoků v síti. V práci byl navrhnut a implementován systém pro evidenci podezřelých skupin síťových adres. Tento systém byl otestován na zkušebních datech. Systém byl napsán v jazyce Python.

Klíčová slova botnet, distribuované útoky, NERD, NEMEA, Warden, IDEA, Python

Abstract

This thesis deals with analysis of traffic going from suspicious network addresses and distribution into groups of addresses. The goal of this thesis is to create a system for grouping suspicious network addresses, which are reported together in security reports or show similar behavior. In the thesis, analysis of input data is done along with analysis of systems NERD, NEMEA

and Warden. This thesis deals with defining a botnet and division of network attacks. System for grouping suspicious network addresses was designed and implemented in this thesis. This system was tested on test data. System was implemented in Python.

Keywords botnet, distributed attacks, NERD, NEMEA, Warden, IDEA, Python

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Botnet je skupina zařízení, která synchronizovaně provádí výpočetně náročné úkoly, nejčastěji se jedná o provádění distribuovaných útoků. Členové botnetu — boti — jsou často počítače nic netušících uživatelů. Botnety aktuálně tvoří velmi nebezpečnou potenciální hrozbu pro všechny systémy. Botnety mohou útočit velkou silou, obzvlášť jde-li o botnet o mnoha stovkách i tisících botů. Zároveň je tvůrce botnetu díky náhodnému rozmístění botů špatně identifikovatelný. Obrana před distribuovanými útoky hraje význačnou roli v obraně celého systému.

Součástí obrany je systém Intrusion Detection System (IDS). Tento systém pro odhalení průniku monitoruje síťový provoz a odhaluje podezřelé aktivity, které by mohly vést k narušení bezpečnosti systému. Systém IDS je zdrojem hlášených detekovaných bezpečnostních událostí, které se v rámci „Incident response“ řeší. Sdílení informací z těchto hlášení může pomoci získat globální pohled.

Motivací k vytvoření systému pro evidování skupin podezřelých síťových adres bylo vytvořit systém, který by ke stávajícímu řešení připojil informaci o tom, zda je podezřelá síťová adresa s nějakými jinými adresami ve skupině, tedy zda v nejhorším možném případě je adresa součástí nějakého botnetu. Systém by v ideálním případě odhaloval skupiny strojů zabývající se nekalou činností.

Výsledkem práce bude implementovaný systém v jazyce Python, který umožní evidovat síťové adresy podezřelé ze škodlivé činnosti ve skupinách. Toto rozdělování do skupin bude probíhat s ohledem na druh činnosti, dobu činnosti a další aspekty potřebné k vhodnějšímu určení skupiny. Tyto skupiny je dále nutné pravidelně aktualizovat. Systém bude umět vložit nové síťové adresy získané z různých zdrojů, případně je přiřadit k vhodné skupině a na žádost jiného systému přes REST API vrací seznam ostatních síťových adres skupiny, kde se daná adresa nachází. Příslušnost adresy ve škodlivé skupině totiž může implikovat i blokování dalších v minulosti méně škodlivých adres. Všechny tyto zprávy budou chodit ve formátu IDEA.

Práce byla vytvořena pod dohledem týmu ze sdružení CESNET. Sdružení CESNET se zabývá výzkumem a vývojem v oblasti informačních a komunikačních technologií, poskytuje počítačové sítě svým členům a řeší bezpečnostní incidenty v síti CESNET2. Práce byla vytvořena v síti CESNET2.

Souvisejícími systémy jsou systém NEMEA, systém NERD a systém Warden. Systém NEMEA je detekční systém pro analýzu síťového provozu. Skládá se z více modulů propojených vlastním rozhraním [1]. Systém NERD shromažďuje informace o všech škodlivých entitách v síti a spravuje nad nimi reputační databázi [2]. Systém Warden umožňuje sdílení a využití informací o detekovaných anomáliích pro více zapojených týmů současně [3].

Práce je rozdělena na dvě části, teoretickou a praktickou. Teoretická část se zabývá definováním botnetů, síťových útoků a popisem souvisejících systémů. V kapitole 3 je uvedení do problematiky útoků, bezpečnostních hrozeb a botnetů. V kapitole 4 jsou popsány související systémy. V praktické části je provedena analýza dat, návrh nového modulu, tento modul je implementován a otestován. V kapitole 5 jsou popsána vstupní data a důležitost atributů v záznamech. V kapitole 6 je systém navržen. Kapitola 7 popisuje, jak byl systém realizován. V kapitole 8 je provedeno testování systému.

Kapitola 1

Motivace

Motivací k návrhu systému pro evidenci skupin podezřelých síťových adres bylo rozšířit množinu nástrojů pro evidenci informací o podezřelých entitách. Tento systém má za úkol držet si informace o záznamech z bezpečnostních událostí a z těchto informací vytvářet skupiny síťových adres z událostí velmi podobné charakteristiky. Systém tak vytváří evidenci skupin podezřelých síťových adres.

Součástí práce je analýza dostupných informací o podezřelých IP adresách a hledání podobně se chovajících. Motivací je propojit informace z nezávislých zdrojů jako jsou různé detekční nástroje do souvislostí, zjistit, které události spolu mohou souviset a v čem se typicky liší od ostatních. Průběžná analýza může do budoucna odhalovat koordinované korelované chování různých zařízení, které by mohlo mít škodlivé následky.

Použití takového systému by hrálo roli při čištění síťového provozu. V případě, že podezřelá síťová adresa získala v systému NERD velmi špatné reputační skóre (případně provoz jdoucí z této adresy bude zakázán), lze do tohoto systému poslat požadavek na informace o skupině adres s událostmi podobné charakteristiky a zvážit snížení reputačního skóre i u těchto dalších adres.

Další motivací pro vytvoření takového systému je možnost propojit informace v systému NERD s informacemi z přidaných modulů či systémů, které nejsou součástí systému NERD. Systém NERD tak od nich nemá žádné informace. Vytvářený systém evidence podezřelých adres by tak mohl rozšířit databázi škodlivých entit v systému NERD.

Kapitola 2

Cíl práce

Hlavním cílem diplomové práce je vytvořit systém pro evidování skupin podezřelých síťových adres. Systém bude nazván Group of IPs (GRIP).

Součástí práce je návrh a realizace algoritmu pro vyhledávání podobných IP a jejich shlukování do skupin, které jsou ukládány a aktualizovány. Tento systém bude umět přidávat novou síťovou adresu dle záznamu o bezpečnostním incidentu a přidá ji k odpovídající skupině podezřelých síťových adres. Přidávání nové síťové adresy bude fungovat pomocí navrženého algoritmu, který z množiny bezpečnostních událostí a dalších zdrojů informací vybere skupinu síťových adres s prakticky totožnými záznamy bezpečnostních incidentů.

Důležitou součástí navrženého systému je programové rozhraní (API), které umožňuje pracovat s evidovanými informacemi v jiných systémech. API je popsáno v kapitole 6.3.1. Systém dokáže odpovědět na HTTP dotaz jiných systémů, zda je určitá síťová adresa podezřelá, zda je v systému v nějaké skupině a tuto skupinu pošle jako odpověď.

Vzhledem k rychlému stárnutí evidovaných informací je součástí návrhu algoritmus pravidelné aktualizace, která projde uložená data – skupiny adres a smaže neaktuální.

Kapitola 3

Úvod do problematiky

Systém GRIP zpracovává informace z bezpečnostních hlášení. V bezpečnostních hlášeních jsou často události typu rozesílání spamu, infikování malwarem, útoky DDoS a další.

3.1 Útoky a bezpečnostní hrozby

3.1.1 Spam

Spam je dle [4] definován jako posílání nevyžádaných zpráv elektronickým systémem za účelem reklamy. Odesílatel těchto zpráv posílá zprávy bez ohledu na to, zda je příjemce chce přijmout nebo ne. V roce 2013 byl přijat zákon, který stanovuje pravidla posílání reklamních emailů.

3.1.2 Malware

Malware je označení pro škodlivý software. Mezi jeho škodlivé činnnosti patří krádež soukromých dat nebo vytvoření tzv. backdoor pro přístup útočníka do systému oběti. Dle [5] lze označit jako malware jakýkoli software, který dělá něco, o čem uživateli neřekne.

Malware se do zařízení může dostat stažením škodlivé aplikace. Dalším způsobem infikování je, že útočník využije zranitelnosti v operačním systému.

3.1.3 Denial of service

DoS je zkratka anglických slov Denial of Service. Jde o útok, jehož účelem je vypnout stroj či síť oběti a znedostupnit ji tak pro legitimní uživatele. Tento typ útoků je veden tím způsobem, že je buď cíl zavalen velkým množstvím provozu, který nezvládne zpracovat, nebo je cíli poslána informace vyvolávající pád systému [6]. V obou případech se zdroje znepřístupní. Oběť DoS útoku obvykle nepřijde o žádná data.

Existují dva základní typy scénářů DoS útoků, dle [7] se liší ve způsobu provedení:

- útok zahlcením linek a
- útok spotřebováním zdrojů.

Útok zahlcením linek Tento typ DoS útoku spočívá v tom, že útočník pošle takový datový tok k cíli, že jej cílová linka nedokáže rychle zpracovat a ucpe se. Běžný provoz se pak na linku nedostane, ač linka stále běží. Příkladem může být útok zvaný „ICMP flood“, který zneužívá špatně nakonfigurovanou síť a posílá falešné pakety, které prověří aktivitu každého prvku v síti místo jen jednoho, a tím zatěžuje síť [6].

Útok spotřebováním zdrojů Tento typ útoku *„se nesnaží ucpat linku k serveru, ale zahltit server samotný“* [7]. Zdroje nejčastěji spotřebovávané při útoku jsou aplikační zdroje a zdroje síťového rozhraní. V případě aplikačních útočník analyzuje aplikaci na serveru, vyhledá v ní slabé místo náročné na zpracování a využije jeho zranitelnost. Může se tak například stát, že aplikace pošle do databáze velký počet dotazů, které databáze nedokáže zpracovat a spadne.

Pokud se útočník zaměřuje na síťové zdroje, dojde k tomu, že je server zahlcen *„už na jádře operačního systému, respektive jeho TCP stacku. Ten je zahlcen požadavky, které pak nestíhá vyřizovat. Přestože může být server ve finále zatížen jen velmi málo a linka je tak prakticky prázdná, odpovědi přestanou přicházet.“* [7]

Jeden z typických útoků tohoto druhu se nazývá „SYN flood attack“. Útočník posílá velké množství SYN paketů, které značí, že uživatel chce se serverem navázat spojení. Server na tento požadavek pošle odpověď - SYN-ACK paket a očekává, že přijde od uživatele potvrzující paket ACK. Ten ale při útoku nikdy nedorazí. Server drží velké množství napůl otevřených spojení, dokud nedojde k vyčerpání možností serveru, server pak přestane na tento typ požadavků reagovat, čímž se stane nepřístupný pro legitimní uživatele.

3.1.4 Distributed denial of service

Rozšířený DoS útok se nazývá Distributed Denial of Service. Jedná se o typ DoS útoku, kdy se na útoku na jeden cíl podílí více synchronizovaných systémů. Výsledek může být pro oběť kritický. Lokalizování zdroje útoku je velmi obtížné, neboť řada zdrojů jsou náhodně rozmístěné kompromitované systémy [6].

Často se k DDoS útoku zneužívají počítače nic netušících uživatelů tím způsobem, že se do počítače dostane škodlivý kód, počítač se pak částečně dostane pod kontrolu útočníků. *„Další ohroženou skupinou zařízení jsou různé*

WiFi routery a další podobná zařízení. Ty dnes často obsahují plnohodnotný operační systém, který je náchylný na podobné chyby jako osobní počítače.“ [7] Jedná se o zařízení, která jsou připojená k internetu a zároveň snadné pro převzetí kontroly, často se zastaralým firmwarem a továrními přihlašovacími údaji.

K DDoS útokům jsou také často využívány botnety.

3.2 Botnet

Botnet je síť propojených kompromitovaných spolupracujících strojů [8][9]. Myšlenkou za vznikem takových sítí je fakt, že nejúčinnější možností, jak dosáhnout výpočetně náročného cíle, je použití velkého množství málo využívaných endpointů.

Členy botnetu jsou často počítače či jiná zařízení *„nic netušících uživatelů, které jsou obvykle napadeny nějakým druhem malware“* [7]. Stroj může být malwarem napaden mnoha způsoby, nejčastěji se tak stane spuštěním zranitelné aplikace s bezpečnostními hrozbami, instalací malwaru či komunikací po síti. Bot poté informuje svého tvůrce, že je připraven k činnosti, dokáže komunikovat s jinými boty a přijímat příkazy od hlavního serveru — tvůrce botnetu.

Hlavní server se v případě botnetu nazývá C&C server — zkratka anglického „command and control“. C&C server dává botům příkazy, ke komunikaci s boty používá nejčastěji standardní internetové protokoly jako HTTP či IRC.

Účelem botnetu je provádět úkoly principem „rozděl a panuj“. Velké množství strojů se pustí do své části úkolu a tím se k výpočetně náročnému cíli dostanou snáze. Nejčastějšími aktivitami jsou dle [7][9][8]:

- spoluúčast na provádění distribuovaného DoS útoku,
- rozesílání spamu,
- sbírání citlivých dat,
- hádání hesel,
- skenování sítě,
- generování falešného provozu na webových stránkách kvůli finanční odměně,
- umísťování reklam apod.

Dle [7] jsou zde tři zásadní výhody: *„nízká pořizovací cena, velká kapacita a anonymita tvůrce. Při správně provedeném útoku nemáte šanci zjistit informace o útočníkovi. Znáte jen IP adresy jakýchsi „bílých koní“, kterými se stali nic netušící uživatelé svých počítačů.“*

3.2.1 Obrana proti botnetu

Podle zdroje z [9] si útočníci obecně vybírají snadné cíle. Nemá pro ně cenu složitě získávat heslo, když velká část zařízeních připojených k internetu stále využívá výchozí tovární heslo, někdy dokonce žádné zabezpečení. Existují veřejně dostupné seznamy dvojic přihlašovací jméno — heslo jako například ten, který použil botnet MIRAI [10]. Pro útočníky tak není nic jednoduššího, než vyzkoušet tyto dvojice k přihlášení na zařízení oběti.

Existuje pár základních pravidel, které by měl dodržovat každý uživatel jakéhokoli zařízení připojeného k internetu:

- použití antivirového řešení, které detekuje instalovaný malware a odstraní ten, který už v zařízení je,
- pravidelné aktualizace operačního systému,
- nestahovat neznámé soubory a neotevírat odkazy z neznámých emailových adres,
- pravidelné změny hesel,
- zákaz použití známých hesel či dokonce dvojic uživatelné jméno - heslo.

3.2.2 Historie

Vznik prvních botnetů se řadí k roku 2000. Jejich původní záměr byl neškodný. Boti z botnetů měli organizovaně pracovat na výpočetně náročných problémech a společně najít řešení.

Tradiční botnety jsou skupiny počítačů a serverů infikovaných malwarem. Dle [11] se postupně tradiční botnety vyvíjely v IoT botnety. Tento druh botnetu je skupina IoT zařízení jako jsou CCTV kamery, lednice či tzv. chůvičky, zařízení s připojením k internetu. Rozdíl oproti tradičnímu botnetu je v tom, že tato zařízení dokážou malware, kterým byla infikována, rozšířít na další zařízení a tím rozšířit botnet o další členy. Ve výsledku zatímco tradiční botnet se skládá z tisíců až desetitisíců zařízení, IoT botnet se bez problému a během relativně krátké doby rozšíří až na milion zařízení.

Oblíbenost IoT botnetů roste s počtem IoT zařízení. Dle [11] k roku 2016 existovalo přibližně 6,4 miliardy IoT zařízeních, přičemž dle předpokladů do roku 2020 bude připojeno 20 miliard takových zařízení.

Mezi nejznámější IoT botnety patří Linux.Aidra. Byl objeven roku 2012 kvůli nadměrnému počtu Telnet spojení na IoT zařízení.

Dalším známým IoT botnetem je Bashlite (též Gayfgt, Qbot, Lizkebab či Torlus), objevený v roce 2014. Byly publikovány jeho zdrojové kódy, je považován za předchůdce botnetu MIRAI.

Jeden z nejznámějších botnetů poslední doby je MIRAI. MIRAI se poprvé objevil 1. srpna 2016, kdy útočil na blog „Krebs on Security“ a hostingového poskytovatele OVH. Šířil se rychle, do konce listopadu 2016 si podrobil

přes 600 000 IoT zařízení, většinu z nich pouze tím, že zkoušel 64 nejznámějších výchozích přihlašovacích údajů.

Mirai funguje ve dvou módech:

- replikační mód,
- útočný mód.

V replikačním módu se rozšiřuje, skenuje Internet a útočí na jednotlivé cíle. Jakmile nalezne zranitelné zařízení, infikuje ho. V útočícím módu botnet útočí na cíle všemi zotročenými zařízeními synchronizovaně.

Co se týče síly útoků, MIRAI útočil na poskytovatele hostingu OVH silou až 1 TB/s ze 145 000 zařízení. Tyto útoky jsou brány jako dosud největší, měly by být upozorněním před zranitelností IoT zařízení.

Architektura MIRAI je tvořena tzv. „infrastructure clustering“, tedy sdružováním infrastruktury do clusterů — skupin spolupracujících strojů. Reverzním inženýrstvím byly zjištěny IP adresy a domény C&C strojů, identifikováno bylo 33 nezávislých C&C clusterů.

Jednotlivé clustery často byly aktivní ve stejnou dobu. Zdroj [12] píše, že MIRAI nebyl spravován jedním člověkem, ale skupinou útočníků, kteří pouštěli své vlastní varianty útoků z různých důvodů.

Dne 21. října 2016 proběhl útok na jednoho z nejznámějších DNS poskytovatelů DYN, to způsobilo znepřístupnění různých populárních webových stránek (Amazon, Paypal, Reddit, Twitter, Github a další). Další obětí se od 31. října téhož roku stal Lonestar Cell, jeden z největších liberijských telekomunikačních operátorů. Koncem listopadu 2016 došlo k výpadku německého internetového poskytovatele Deutsche Telekom poté, co 900 000 jeho směrovačů bylo kompromitováno. Tento útok byl překvapivě proveden chybnou verzí MIRAI, která místo infikováním stroj vypnula.

Zdroj [12] svůj článek uzavírá tím, že MIRAI útoky půjdou do historie jako bod zvratu, kdy se IoT zařízení staly novou normou v provádění DDoS útoků.

Z tohoto článku vyplývají minimální bezpečnostní opatření před infikování vlastního IoT zařízení:

- eliminovat výchozí přihlašovací údaje,
- povolit automatické vyhledávání a instalaci aktualizací IoT zařízení,
- nastavit limit pro počet pokusů k přihlášení.

Kapitola 4

Související systémy

Souvisejícími systémy jsou systémy NERD, NEMEA a Warden. Tyto systémy jsou vytvářeny týmem působícím v síti CESNET2 ve sdružení CESNET.

4.1 NERD

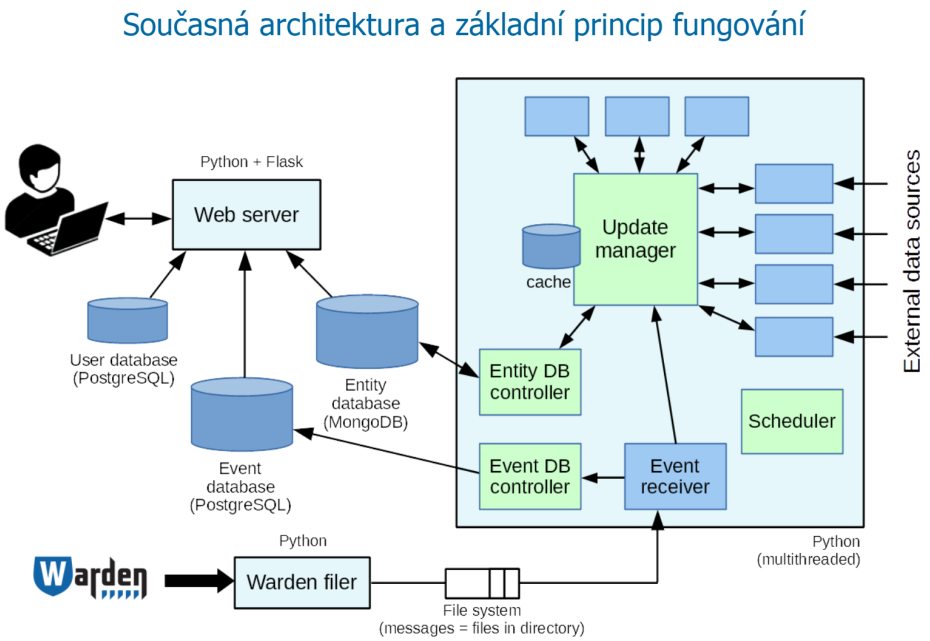
Network Entity Reputation Database[1] (NERD), je systém tvořící databázi síťových entit. V databázi drží IP adresy, sítě, domény, autonomní systémy a další entity. Dohromady vytváří *„seznam známých zdrojů škodlivých aktivit na internetu a všechno, co o nich víme“* [13]. Systém NERD ve výsledku může sloužit jako systém pro upozornění na potenciálně škodlivé IP adresy, pro vyšetřování bezpečnostních incidentů a pro generování blacklistů. K datům lze snadno přistupovat přes web prostřednictvím REST API.

Systém přijímá hlášení o bezpečnostních událostech z Wardenu (viz níže) a jemu podobných systémů pro detekci škodlivého provozu. Tato hlášení systém rozšíří o další informace z externích zdrojů. Hlášení jsou obohacena o *hostname*, geolokaci IP adres, případnou přítomnost v některém z blacklistů a další data. Nakonec se všechny informace shrnou do výsledného ohodnocení, tzv. reputačního skóre. To značí, jak velkou hrozbu daná entita představuje.

Databázi tvoří záznamy entit. Tyto entity se skládají z IP adresy, BGP prefixu, identifikátoru autonomního systému a jména organizace. Záznam navíc obsahuje metainformace o nahlášených událostech (počet událostí, jejich kategorie a detektor, který událost detekoval), hostname, geolokace, přítomnost na blacklistech a výsledné reputační skóre. Všechny tyto informace je důležité pravidelně aktualizovat, pokud se záznam za posledních 14 dní nezmění (nepřibyde žádná událost k dané adrese), je záznam smazán.

Celý systém je modulární — je tvořen moduly, které jsou navrženy tak, že každý má svou konkrétní funkci a nezasahuje do funkce jiného. Mezi existující moduly patří modul pro nalezení geolokace, modul pro dotazy do blacklistů,

[1] https://nerd.cesnet.cz/



Obrázek 4.1: Obrázek architektury systému NERD je převzat z [13].

EventReceiver zpracovávající události přicházející z Wardenu, Cleaner, který maže stará data, a další.

„Všechny změny v záznamu řídí UpdateManager“ [13], přijímá žádosti o nějakou změnu v záznamu, načte záznam dané entity, volá funkce jednotlivých modulů, je-li třeba, a změny poté zapisuje zpět do databáze.

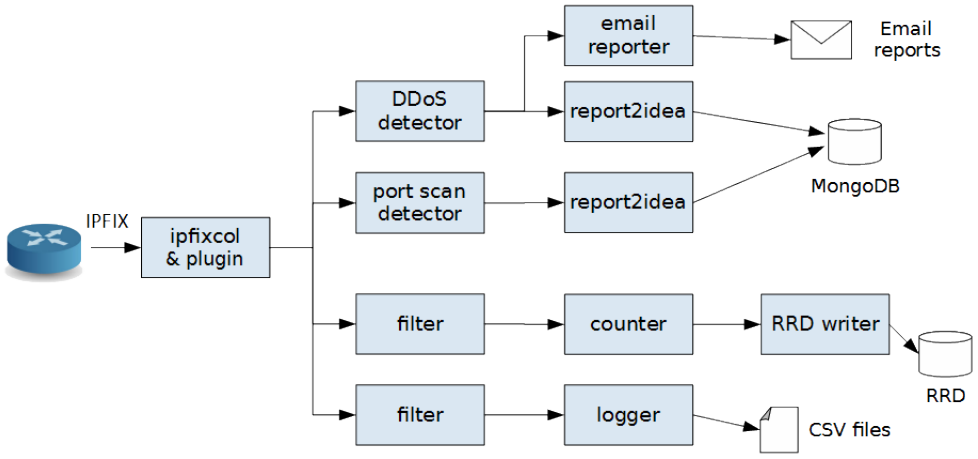
Schéma architektury je znázorněno na Obrázku 4.1.

4.2 NEMEA

Network Measurements Analysis[2] (NEMEA) je stream-wise systém pro analýzu síťového provozu a detekci anomálií. Systém je modulární a založený na síťových tocích. Moduly zpracovávají příchozí zprávy, které obsahují informace o síťových tocích reprezentující detekované bezpečnostní události [14][15].

Systém NEMEA se skládá z několika modulů s různou funkcionalitou, ale jednotně řízené. Jedná se o moduly pro detekci různých typů podezřelého síťového provozu, pro počítání statistik ohledně provozu, pro filtrování a agregaci zpráv a pro hlášení událostí, které byly detekovány [14]. Příklad konfigurace systému je naznačena na Obrázku 4.2.

[2]http://nemea.liberouter.org/



Obrázek 4.2: Obrázek příkladu konfigurace systému NEMEA je převzat z [14].

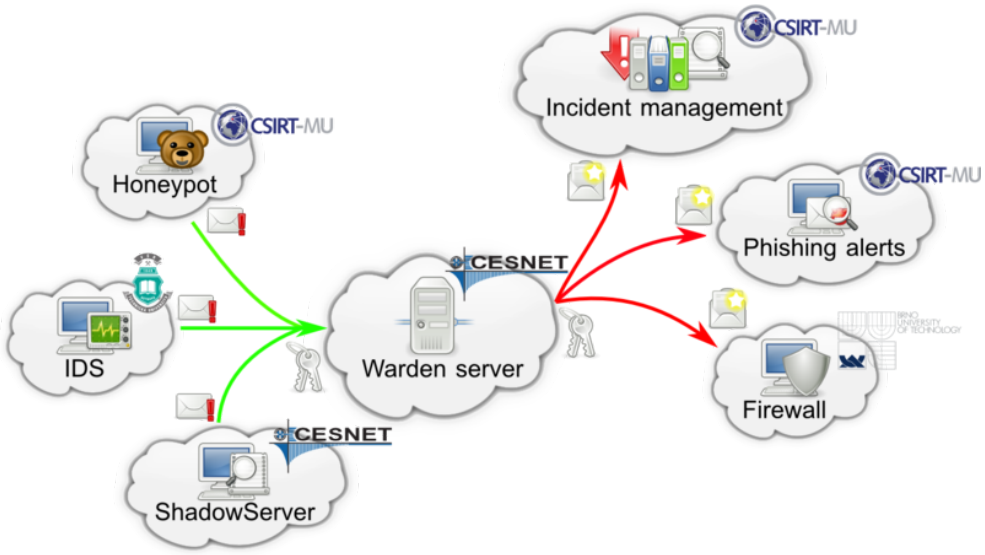
4.3 Warden

Warden[3] je *„systém pro sdílení informací o detekovaných bezpečnostních událostech“* [3]. Umožňuje jednoduše a efektivně sdílet informace o detekovaných anomáliích mezi zapojenými bezpečnostními týmy. Detektory zapojených týmů posílají bezpečnostní hlášení centrálnímu serveru a ten je rozesílá klientům. Data ze systému poskytují další užitečné informace k zajištění bezpečnosti a monitoringu sítě.

Kromě systému pro sdílení informací slouží Warden i pro zvýšení bezpečnosti sítě. Dle [16] systém pomáhá i bezpečnostním týmům při řešení událostí, je využit při aktivní obraně sítí a služeb a propojuje podobné systémy ve světě.

„Systém se skládá z hlavního Warden serveru, obstarávajícího veškerou zásadní činnosti, a dvou typů klientů.“ [17] Prvním typem klienta je odesílající klient, který dodává informace hlavnímu serveru - odesílá data z detekčních nástrojů. Druhým typem je odebírající klient. Ten získává požadované informace, lze ho nastavit pro příjem konkrétního typu událostí. Architektura je vidět na Obrázku 4.3.

[3] https://warden.cesnet.cz//cs/index



Obrázek 4.3: Obrázek architektury systému Warden je převzat z [17].

Kapitola 5

Analýza vstupních dat

Součástí práce je algoritmus, který rozdělí IP adresy bezpečnostních událostí do skupin. Aby bylo možné takový algoritmus navrhnout, je nutné provést analýzu dat, které je možné pozorovat v záznamech z bezpečnostních událostí.

5.1 Cíle analýzy dat

Hlavním cílem analýzy je určit, jaká data bude systém zpracovávat, jak tato data spolu mohou souviset, které části záznamů jsou důležité a které méně důležité. Analýza dat ukáže, s jakou pravděpodobností se které druhy záznamů budou zpracovávat, jaké atributy budou patrně obsahovat a co daný záznam popisuje za událost. Ukáže, jaké kombinace kategorií událostí se často vyskytují a které atributy u těchto kombinací považovat za důležité při určování souvislosti.

5.2 Předpoklady

Ještě před zahájením analýzy dat bylo definováno několik předpokladů pro tvoření skupin adres:

- Více zdrojů ve stejném záznamu spolu mohou souviset.
- Zařízení nakažená stejným malwarem se budou chovat stejně.
- Zařízení, která vykazují podobné chování, se dají vložit do skupiny, která spojuje tyto podobně se chovající.
- Zařízení chovající se podobně v rámci časového úseku se dají vložit do skupiny.
- Pokud najdeme nějaké evidentně nakažené zařízení, tak podobně se chovající zařízení by mohlo být rovněž nakažené.

K těmto předpokladům je nutné při implementaci algoritmu, který bude vytvářet skupiny, přihlédnout.

5.3 Formát záznamů

Kazdý záznam je ve formátu IDEA. Formát je převzat z formátu JSON. Skládá se z dvojic klíčů a hodnot, kde hodnoty mohou nabývat datových typů integer, string, timestamp, pole hodnot či slovník dalších dvojic klíčů a hodnot. Je důležité zdůraznit, že klíče nejsou povinné, proto každý záznam v IDEA zprávě může obsahovat jiné klíče. Z tohoto důvodu jeden záznam není snadno porovnatelný s jinými záznamy. Tento formát je blíže specifikován ve zdroji [18]. Příklad takového záznamu je zobrazen níže.

```
{
"DetectTime": "2017-11-29T14:24:15Z",
"Category": ["Attempt.Login", "Test"],
"Target": [
          {"Port": [22],
           "Proto": ["tcp", "ssh"]}
          ],
"Format": "IDEA0",
"Node": [
        {"Type": ["Flow", "Statistical"],
         "SW": ["Nemea", "brute_force_detector"],
         "Name": "cz.cesnet.nemea.bruteforce"}
        ],
"Source": [
          {"IP4": ["195.113.44.19"],
           "Proto": ["tcp", "ssh"]}
          ],
"FlowCount": 16,
"ID": "099fc5ad-660b-4126-9dc1-b1c092658a1d",
"CreateTime": "2017-11-29T14:24:51Z",
"Description": "Multiple unsuccessful login attempts on SSH"
}
```

Nejdůležitějšími a nejzajímavějšími atributy jsou `Source`, `Target`, `Category`, `Description` a dvojice `EventTime` a `CeaseTime`. Jedná se postupně o pole zdrojů („source of trouble“), pole cílů, kategorii, která je záznamu přidělena, lidský popis události a dvojice časů — čas, kdy událost začala, a čas, kdy událost skončila.

5.4 Průběh analýzy

Základní analýza proběhla v podobě spuštěného skriptu nad vzorkem dat. Testovací vzorek dat se skládal z 1 000 000 reálných záznamů o bezpečnostních událostech. Systém Warden generuje až 12 záznamů za vteřinu, s průměrnými 10 záznamy za vteřinu tak 1 000 000 záznamu tvoří přibližně 27,778 hodin monitorování sítí. Sběr testovaných dat probíhal náhodně v intervalu 10 dnů.

První část analýzy zkoumá data z hlediska druhů kategorií a jejich podobnost v rámci kategorie. Skript zjišťuje, kolik záznamů dané kategorie obsahuje dané atributy, v případě, že jde o atribut s číselnou hodnotou, sleduje jeho průměrnou hodnotu a rozptyl. Tyto hodnoty potom znamenají, jaká je pravděpodobnost, že záznam dané kategorie bude obsahovat dané atributy. Pokud například jen 60 % záznamů určité kategorie obsahovalo nějaký atribut, nelze spoléhat, že v záznamu bude. Pak je důležité určit, zda tento atribut je pro určení skupiny podstatný nebo nepodstatný. Pokud by byl podstatný, pak by se záznam neobsahující tento atribut ignoroval. Pokud by byl nepodstatný, netvořil by rozhodující prvek při určení skupiny, případně by pouze zpřesňoval určování skupiny.

Druhá část analýzy pracuje se záznamy jako s posloupností v čase. Hledá záznamy, které spolu souvisí, hledá, jaké atributy mají společné. Analýza se zaměřuje zejména na atributy `Source` a `Target`, dále různé časy záznamů. Testuje, jak spolu souvisí různé kategorie záznamů a co znamená jejich součinnost. Tato část analýzy určila, kolik atributů záznamů je nutných pro určení skupiny. Též ladí podmínky přidělování skupin tak, aby se netvořilo příliš skupin. Zároveň je nutné se vyvarovat podmínkám, kterými by velká část záznamů neprošla a tím pádem by pak algoritmus vytvářel příliš málo skupin. Je třeba vytvořit rovnováhu mezi rychlostí zpracování, objemem dat grafové databáze na disku a odpovídajícím a zároveň co největším množstvím vytvořených skupin. Tato část analýzy byla připravena na základě výsledků z předchozí části.

5.5 První části analýzy

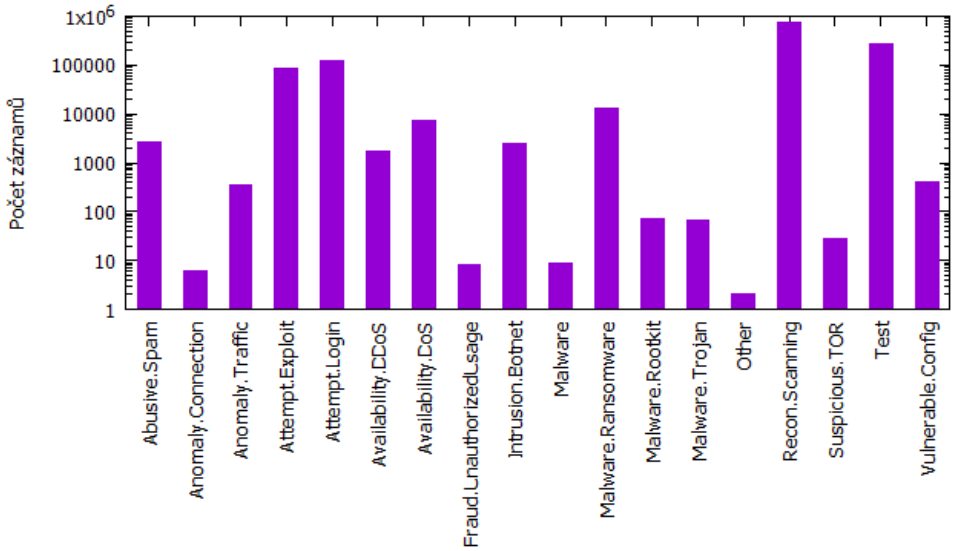
První část analýzy se zabývala praktickým definováním kategorie z hlediska jejich atributů. V datech o 1 000 000 záznamů bylo nalezeno 18 druhů kategorií událostí. Jak je vidět z Tabulky 5.1 a z grafu na Obrázku 5.1, většinu všech kategorií uvedených v datech tvoří kategorie *Recon.Scanning*. Tato kategorie znamená to, že útočník vyšle požadavky na systém kvůli hledání slabého místa v systému. Jedná se nejčastěji o skenování portů, dále posílání DNS dotazů, zjišťování informací o účtech a službách systému, posílání ICMP či SMTP paketů.

Další velkou skupinou je kategorie *Test*, která slouží pro testovací účely. V době implementace šlo o testování blacklistů.

Poslední výraznou skupinu tvoří kategorie *Attempt.Exploit* a kategorie *At-*

Tabulka 5.1: Tabulka počtu záznamů obsahující danou kategorii v testovaných datech.

Kategorie	Počet záznamů
Abusive.Spam	2738
Anomaly.Connection	6
Anomaly.Traffic	354
Attempt.Exploit	83991
Attempt.Login	125288
Availability.DDoS	1743
Availability.DoS	7107
Fraud.UnauthorizedUsage	8
Intrusion.Botnet	2555
Malware	9
Malware.Ransomware	13352
Malware.Rootkit	70
Malware.Trojan	67
Other	2
Recon.Scanning	762363
Suspicious.TOR	28
Test	270246
Vulnerable.Config	400



Obrázek 5.1: Graf podílu kategorií v záznamech v logaritmickém měřítku.

tempt.Login. V obou případech jde o pokus o kompromitování systému. V prvním případě útočník zneužil zranitelné místo se standardizovaným identifikátorem (například CVE). V druhém případě se útočník snažil kompromitovat systém vícenásobnými pokusy o přihlášení. Útočník zkouší vkládat známá hesla, pokouší se o prolomení hesla či hrubou silou zjistit heslo.

Kategorie *Abusive.Spam* upozorňuje na možné přijetí spamových zpráv. Kategorie *Anomaly* znamená, že se přihodilo něco nečekaného, co by mohlo tvořit bezpečnostní problém. Kategorie *Availability.DoS* a *Availability.DDoS* upozorňuje na události, kdy byl systém pod náporem mnoha požadavků, což způsobilo zpoždění operací či pád aplikací. U záznamu s kategorií označenou *Fraud.UnauthorizedUsage* útočník použil zdroje k nepovolenému použití, například emaily s obsahem, který se podílí na nelegálním profitování. Kategorie *Intrusion.Botnet* upozorňuje, že systém či aplikace byly napadeny a jsou součástí nějakého botnetu. Všechny varianty kategorie *Malware* znamenají, že součástí systému je škodlivý software. To, o jaký typ softwaru se jedná, je označeno v druhé části názvu kategorie. Kategorií *Other* jsou označeny ty události, které nebylo možné více identifikovat. Kategorie *Suspicious.TOR* je hlášení, které říká, že hlášená adresa je exit node. Poslední kategorie *Vulnerable.Config* upozorňuje, že v systému existuje zranitelné místo, které by se mohlo stát místem vniku útočníka, nejčastěji otevřené resolvery a neaktualizovaná databáze nalezitelných virů. Přesný popis kategorií je uveden na [18].

V dalším kroku byla vytvořena tabulka znázorňující, zda pro dané kategorie je vysoká pravděpodobnost, že atributy budou obsaženy v záznamu. Z tabulky bylo vidět, že mezi kategoriemi jsou rozdíly, lze tak přesněji vystihnout danou kategorii a tím lépe zhodnotit, zda dva záznamy spolu souvisí či nikoli. Následující seznam atributů jsou atributy, jejichž vysoká pravděpodobnost výskytu v záznamu je rozdílná mezi kategoriemi a lze je použít pro lepší definování skupiny:

- Description
- _CESNET
- Note
- Target
- EventTime
- Source
- CreaseTime
- ConnCount
- CreateTime

- ByteCount
- PacketCount
- Ref
- Attach
- Confidence
- AltNames
- FlowDataSource

5.6 Minimalizace

Jedním z požadavků na vytvořený algoritmus je rychlost zpracovávání dat a zároveň co nejpřesnější určení skupiny souvisejících shodných či podobných záznamů. To jsou protichůdné požadavky. Aby se dosáhlo obou požadavků, bylo zvoleno jako řešení promazání dat, která mohou tvořit nepřesnosti ve výsledku. Algoritmus bude ignorovat záznamy, které neobsahují atributy nutné pro dané kategorie, protože nejsou přesně určené. Povinné atributy budou ty, u kterých se v první části analýzy ukázala vysoká pravděpodobnost výskytu pro dané kategorie. K těmto atributům budou přidány další, které v analýze měly vysoké procento pravděpodobnosti, že budou obsaženy v záznamu. To zaručí, že záznamů, které se budou zpracovávat, bude méně, ale budou přesněji určené.

Dále budou ignorovány záznamy, které neobsahují atribut `Source`.

Aby se proces ještě více zefektivnil, lze ignorovat některé kategorie, které nemá smysl umisťovat do skupin. Jedná se o kategorie:

- *Other*
- *Suspicious.TOR*
- *Test*

5.7 Druhá část analýzy

Druhá část analýzy je tvořena skriptem, který prochází data v čase a vyhledává jejich podobnosti. Skript nejprve prochází jednotlivé skupiny a sleduje jejich časy. Tím pozoruje, jak jsou události dané kategorie rozdělené v čase. Lze tím i zjistit, jakou dobu zpětně spolu mohou události souviset. V ostatních případech jde o časový údaj znázorňující maximální dobu, ve které hledat v historii související záznamy. Časy byly ručně zpracovány. Výsledné časy jsou vidět v následující Tabulce 5.2. Většina těchto časů se rovná jedné hodině,

Tabulka 5.2: Tabulka časů znázorňující dobu, jak hluboko v historii budou záznamy prohledávány.

Kategorie	Počet záznamů	Odhadovaná doba [hodiny]
Abusive.Spam	2738	1
Anomaly.Connection	6	0.25
Anomaly.Traffic	354	1
Attempt.Exploit	83991	1
Attempt.Login	125288	1
Availability.DDoS	1743	1
Availability.DoS	7107	1
Fraud.UnauthorizedUsage	8	0.5
Intrusion.Botnet	2555	1
Malware	9	0.25
Malware.Ransomware	13352	1
Malware.Rootkit	70	1
Malware.Trojan	67	1
Recon.Scanning	762363	1

zejména kvůli velkému množství záznamů dané kategorie. Je proto možné, že menší časy byly způsobeny nízkým počtem záznamů.

Dále skript sleduje, jaké jsou společné hodnoty atributů pro danou kategorii v záznamech sebraných za dobu zvolenou v předchozí části běhu skriptu.

Skript nejprve sledoval počet IP adres v atributech `Source` a `Target`. Skript pracoval pouze se záznamy, které obsahovaly tyto dva atributy. Protože atribut `Target` nebyl uveden ve 100 % zaznamů, ukázal skript výsledky pouze pro kategorie, které vždy obsahovaly tento atribut, výsledky se tedy projevily dle výskytu tohoto atributu. Tím se oddělily kategorie, které lze sledovat pro podobnost v atributech `Source` a `Target` v čase. Atribut `DetectTime`, který je dle předchozí části analýzy obsažen ve všech záznamech, byl použit pro sledování času. Skript pozoroval počet všech IP adres v atributech `Source` a `Target` a vedle toho pozoroval počet různých IP adres v těchto atributech. Tyto IP adresy sbíral za poslední hodinu od kontrolovaného záznamu. V Tabulce 5.3 je zobrazen výsledný maximální počet IP adres pro dané kategorie (`Max`) a maximální počet unikátních IP adres pro dané kategorie (`Uniq`).

Kategorie *Malware* nikdy v hodinovém časovém rozsahu nenalezla záznam stejné kategorie, lze tedy z důvodu efektivity tuto kategorii vypustit ze sledování pro podobnost v atributech `Source` a `Target` v čase. Kategorie *Recon.Scanning* sice neobsahuje atribut `Target` vždy, má ale tolik záznamů, že vždy za poslední hodinu nějaké záznamy nalezne.

Pro zbývající kategorie je třeba vytvořit část algoritmu určenou pouze pro danou kategorii.

Poslední část analýzy sledovala existenci atributů `IP4`, `IP6` a `Email` v atri-

Tabulka 5.3: Tabulka maximálních počtů IP adres (sloupce Max) a maximálních počtů unikátních IP adres (sloupce Uniq) v atributech Source a Target v záznamech za poslední hodinu od sledovaného záznamu.

Kategorie	Source		Target	
	Max	Uniq	Max	Uniq
Anomaly.Traffic	64	9	247	246
Attempt.Exploit	22	16	22	2
Attempt.Login	171	46	171	19
Availability.DDoS	6	2	6	2
Malware	1	1	1	1
Recon.Scanning	27019	16716	197465	12360

butech `Source` a `Target`. V Tabulkách 5.4 a 5.5 jsou uvedeny výsledky, Tabulka ukazuje odděleně výsledky jednotlivých atributů — počet záznamů, které obsahovaly atribut, a průměrný počet hodnot v atributu záznamu.

Z výsledků vyplývá, že ani jeden záznam neobsahoval atribut `Email` ani u `Source`, ani u `Target`. Počet IP adres uvedený v atributech atributu `Source` je vždy roven jedné IP adrese. Oproti tomu počty IP adres uvedených v atributech atributu `Target` se velmi liší. Zároveň pokud bude porovnán počet IP adres verze 4 a verze 6, lze dle Tabulky 5.6 konstatovat, že IP adres verze 6 je menšina. Síťové adresy verze 6 proto nebudou v algoritmu brány v úvahu, stejně jako atribut `Email`.

5.8 Návrh algoritmu

Samotný algoritmus se dělí na dvě části. První část tvoří Python funkce, která prohledává záznamy za poslední hodinu a hledá podobnosti, díky kterým by dva záznamy mohly patřit společně do skupiny. Na konci funkce vrátí seznam adres z atributu `Source`, které patří do společné skupiny, a nebo seznam identifikátorů záznamů, které ukázaly podobnost.

Druhá část algoritmu je ve funkci kódu, který je součástí řešení Celery, v souboru `tasks.py`. Funkce v Celery dostane seznam IP adres a ty musí zpracovat tak, aby byly ve společné skupině, správně provázané, i za předpokladu, že některé adresy již v databázi jsou.

Algoritmus byl inspirován řešením projektu SABU [19].

Tabulka 5.4: Tabulka počtu (sloupce Počet) výskytů atributu IP4, IP6 a Email v atributu Source záznamů jednotlivých kategorií včetně průměrného počtu (sloupce Průměr) hodnot v daném atributu.

Kategorie	IP4		IP6		Email	
	Počet	Průměr	Počet	Průměr	Počet	Průměr
Abusive.Spam	0	1	0	0	0	0
Anomaly.Conn.	0	1	0	0	0	0
Anomaly.Traffic	348	1	6	1	0	0
Attempt.Exploit	83985	1	6	1	0	0
Attempt.Login	92027	1	4	1	0	0
Availability.DDoS	1743	1	0	0	0	0
Availability.DoS	23	1	0	0	0	0
Fraud.Unauth.	2	1	6	1	0	0
Intrusion.Botnet	0	1	0	0	0	0
Malware	7	1	2	1	0	0
Malware.Rans.	0	1	0	0	0	0
Malware.Rootkit	0	1	0	0	0	0
Malware.Trojan	67	1	0	0	0	0
Other	2	1	0	0	0	0
Recon.Scan.	603247	1	0	0	0	0
Suspicious.T.	0	1	0	0	0	0
Test	179503	1	18	1	0	0
Vulnerable.Conf.	0	1	0	0	0	0

Tabulka 5.5: Tabulka počtu (sloupce Počet) výskytů atributu IP4, IP6 a Email v atributu Source záznamů jednotlivých kategorií včetně průměrného počtu (sloupce Průměr) hodnot v daném atributu.

Kategorie	IP4		IP6		Email	
	Počet	Průměr	Počet	Průměr	Počet	Průměr
Abusive.Spam	0	0	0	0	0	0
Anomaly.Conn.	0	0	0	0	0	0
Anomaly.Traffic	348	3,25862069	6	1	0	0
Attempt.Exploit	83985	1	6	1	0	0
Attempt.Login	3382	1,127143702	0	0	0	0
Availability.DDoS	1743	1	0	0	0	0
Availability.DoS	434	1	0	0	0	0
Fraud.Unauth.	2	10	6	1	0	0
Intrusion.Botnet	0	0	0	0	0	0
Malware	7	1	2	1	0	0
Malware.Rans.	0	0	0	0	0	0
Malware.Rootkit	0	0	0	0	0	0
Malware.Trojan	67	1	0	0	0	0
Other	2	1	0	0	0	0
Recon.Scan.	799292	21,13230334	0	0	0	0
Suspicious.T.	0	0	0	0	0	0
Test	90869	23,23314882	14	1	0	0
Vulnerable.Conf.	0	0	0	0	0	0

Tabulka 5.6: Tabulka počtu IP adres verze 4 a verze 6 v atributech Source a Target.

Atribut	IP4	IP6
Source	960954	42
Target	19093258	34

Kapitola 6

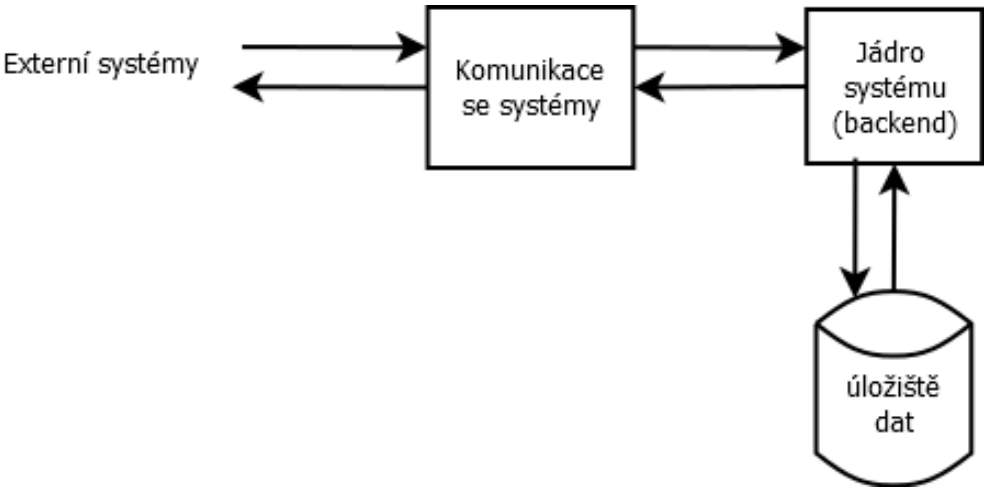
Návrh řešení a architektura

Systém GRIP byl navržen jako centrální modul s interním úložištěm a prostředkem pro komunikaci s ostatními systémy. Návrh je zobrazen na Obrázku 6.1.

6.1 Požadavky

Před návrhem systému bylo definováno několik požadavků, které jsou kladeny na vytvářený systém:

- umožnění dlouhého zpracování požadavků,
- mnoho příchozích požadavků za krátký časový interval,



Obrázek 6.1: Návrh systému GRIP složený z jádra systému, vstupy a výstupy systému a úložiště.

- zpracování libovolného textu ve formátu IDEA (viz Sekce 5.3),
- rychlý přístup a vyhledávání v interním úložišti,
- nízké požadavky na zdroje,
- použití programovacích jazyků Python nebo C,
- komunikace se systémy známým protokolem.

Systém byl navržen tak, aby v budoucnu bylo možné připojit k jeho zpracovávání více zdrojů, aby bylo možné se na výsledky dotazovat z více systémů ve stejnou chvíli a aby zpracovávání bylo tak rychlé, že zvládne nápor běžného provozu.

Po zvážení budoucího praktického využívání bylo třeba dodat systému frontu a vícevláknové zpracování, systém by tak nespoléhal na okamžité provedení požadavku a zároveň byl připraven na více požadavků s dlouhou dobou zpracování najednou. Jednotlivá vlákna by si z fronty vyzvedla požadavek a ten zpracovala.

6.1.1 Vícevláknové zpracování

Vícevláknové zpracování lze implementovat více způsoby — buď s pomocí Python modulu threading, nebo se službou zabývající se vyzvedáním zpráv z fronty nebo spustit samotný skript ve více vláknech. Řešení pomocí Python modulu threading vyžaduje spravování vláken. Využít službu, která sama vytvoří vlákna a ta si sama vyzvednou úkol z fronty, se zdá velmi efektivní. Není třeba řešit správu vláken. Skript spuštěný v několika vláknech je dalším možným řešení, ve kterém bychom se vyvarovali použití vláken a zacházení s nimi.

6.1.2 Fronta

Fronta v systému ulehčuje přijímání požadavků z více serverů.

Jednou z možností bylo vytvořit frontu pomocí Python modulu Queue a spravovat ji ve vlastním skriptu. Výhodou takto implementované fronty je její nenáročnost na systémové prostředky.

Druhou možností bylo využít opensource *message broker*. Toto řešení je náročnější na výkon, ovšem disponuje možností spravovat více front najednou, vytvářet uživatele a spravovat jejich přístupy k frontám. Nabízí se tak možnost využít již běžící *message broker* pro více aplikací. V případě oddělení systémů na různé servery je posílání požadavků do *message broker* instance jistě snazší.

S ohledem na budoucí rozšířené využití byl pro systém vybrán jako fronta *message broker*.

6.2 Jádro

Jádro systému tvoří služba Celery. Celery je *task manager*, z přijatých zpráv vytváří úlohy, které následně zpracovává. Výhodou Celery je to, že jej lze napojit na *message broker*. Celery je psaný v jazyce Python, lze jej jednoduše použít v Python skriptech. Celery napojený na *message broker* přijme z fronty požadavek, tento požadavek zpracuje a výsledek vloží do úložiště. Celery též běží ve více procesech, čímž se splní požadavek na vícevláknové zpracování, v tomto případě víceprocesové zpracování.

Celery je služba běžící jako aplikace, lze ji spouštět pomocí služby Supervisor. Tím je zaručeno, že Celery bude i s procesy spuštěn i po restartu serveru.

V architektuře je jádro označeno jako `backend`.

6.3 Vstupy a výstupy

Vstupy do systému GRIP jsou dva:

- přidání nového záznamu ze systémů,
- požadavek na promazání starých záznamů.

Požadavek na přidání záznamu může být odeslán pouze z interního systému a to tím způsobem, že je vložen do fronty požadavků. Požadavek obsahuje část `body`, jejíž obsah je ve formátu IDEA, jedná se o záznam z bezpečnostní události ze systému Warden či zpráva generovaná z jiných zdrojů (např. z nově vyvíjených modulů PassiveDNS, SpamDetector, IPBlacklist).

Požadavek na promazání starých záznamů je posílán do systému v pravidelných intervalech jednou za 14 dní, provede výmaz údajů, ke kterým nebyla za posleních 14 dní přidána žádná nová událost, z databáze.

Jako fronta byl vybrán *message broker* RabbitMQ. Tento message broker je podporován službou Celery. Pro RabbitMQ existuje Python modul `pika`, který umí spravovat fronty z Python skriptu. RabbitMQ může být následně použit i pro jiné účely souvisejících systémů.

6.3.1 Interakce se systémem

Interakce se systémem probíhá prostřednictví REST API. Pomocí HTTP požadavků lze poslat žádost o nalezení skupiny nebo síťové adresy a seznam všech skupin.

Žádost o nalezení skupin dle zadáné síťové adresy znamená, že některý ze systémů chce získat informace o dané síťové adrese — chce zjistit, zda-li adresa není v nějaké skupině s jinými podezřelými adresami. Žádost o nalezení síťových adres dle zadané skupiny znamená, že některý ze systémů chce získat informace o dané skupině — chce zjistit, zda-li skupina neobsahuje více

potenciálně škodlivých adres. Tyto požadavky jsou ve formátu síťové adresy, resp. identifikátoru skupiny.

Na požadavek o nalezení skupin dle zadáné síťové adresy vrací seznam skupin podezřelých adres dle zadané síťové adresy, která je součástí všech vyjmenovaných skupin. Pokud zadaná adresa není v žádné skupině nebo dokonce nemá žádný záznam v databázi, pak systém vrací prázdnou hodnotu. Na požadavek o nalezení síťových adres dle zadáné skupiny vrací seznam síťových adres dle zadané skupiny, které jsou součástí zadané skupiny. Pokud zadaná skupina neexistuje, pak systém vrací prázdnou hodnotu. Výsledek je odeslán ve formátu HTTP odpovědi.

Další možností je požádat o kompletní seznam skupin včetně zdůvodnění, proč byly členské adresy sloučeny do skupiny.

Tyto tři požadavky mají každý svůj API endpoint:

1. `/grip/groups/`
2. `/grip/groups/[IP_ADRESA]`
3. `/grip/group/[ID_SKUPINY]`

První endpoint vytvoří seznam všech skupin, které se v databázi nachází. Druhý endpoint vyhledá všechny skupiny, do kterých patří zadaná `IP_ADRESA`. Poslední endpoint vrátí seznam všech síťových adres, které jsou uvnitř skupiny s názvem `ID_SKUPINY`.

6.4 Úložiště

Při navrhování úložiště se zohledňovala velikost úložiště, rychlost práce a složitost vkládání a upravování. Dalším požadavkem je zpracování hlášení ve formátu IDEA, ve kterém každá zpráva může obsahovat jiné atributy.

Pokud by byl brán v úvahu raw file, dosáhlo by se minimální velikosti úložiště, ovšem velmi složité manipulace s daty.

MongoDB, tedy dokumentově orientovaná databáze, oproti tomu disponuje snadnou údržbou dat. Nevýhodou je, že MongoDB nedokáže ukládat zprávy s různými klíči. Jednotlivé záznamy je též nutné spojovat do skupin. Takové spojování, např. podle ID záznamu v databázi, je v MongoDB příliš složité a nepřehledné.

Oproti tomu grafové databáze se primárně zaměřují na spojování prvků v rámci celé databáze. Zprávy v IDEA formátu je třeba mírně upravit do formátu čitelného pro danou databázi, jedná se ovšem o minimální změny ve formátování vstupu. Další výhodou je, že jeden záznam v grafové databázi pojme libovolné množství atributů i různé názvy bez ohledu na ostatní záznamy a jejich atributy. Ve výsledku byla použita grafová databáze.

6.4.1 Grafová databáze

Grafové databáze využívají zcela odlišný jazyk než relační. Jazyk je uzpůsobený tomu, že jednotlivé záznamy v databázi jsou navzájem propojeny.

Protože všechny záznamy bezpečnostních událostí jsou ukládány do databáze systému NERD, není nutné vytvářet novou databázi stejných záznamů. Grafová databáze proto bude ukládat pouze hlavní informace včetně ID událostí, které jsou následně dohledatelné v databázi NERDu. Jejím hlavním účelem bude spojovat entity.

Uzly vkládané do systému se dělí na:

- Group,
- Ip,
- Note.

Uzel Group představuje skupinu, její název, časovou značku poslední modifikace a důvod jejího vytvoření. K uzlům typu Group se připojují uzly typu Ip. Tyto uzly představují jednotlivé IP adresy z atributů Source přijímaných hlášení. Uzly obsahují pouze název, kterým je vždy síťová adresa. Poslední typ uzlu je Note — indikátor události. Uzly Note se spojují s těmi uzly Ip, které jsou v atributu události. Uzel Note obsahuje ID, které je shodné s ID události. Lze pak dohledávat události v databázi systému NERD. Uzel dále obsahuje časovou značku, kdy byla událost detekována. Uzel Ip může být spojen s žádným nebo s více než jedním uzlem Group, stejně tak uzel Note může být spojen s více uzly Ip, avšak vždy je spojen alespoň s jedním. Uzel Group je vždy spojen s alespoň dvěma uzly Ip.

Bylo vybráno Neo4j jako grafová databáze. Neo4j je opensource řešení grafové databáze s Python modulem, který umožňuje pracovat s daty v databázi ze skriptů. Používá vlastní dotazovací jazyk Cypher inspirovaný SQL jazykem.

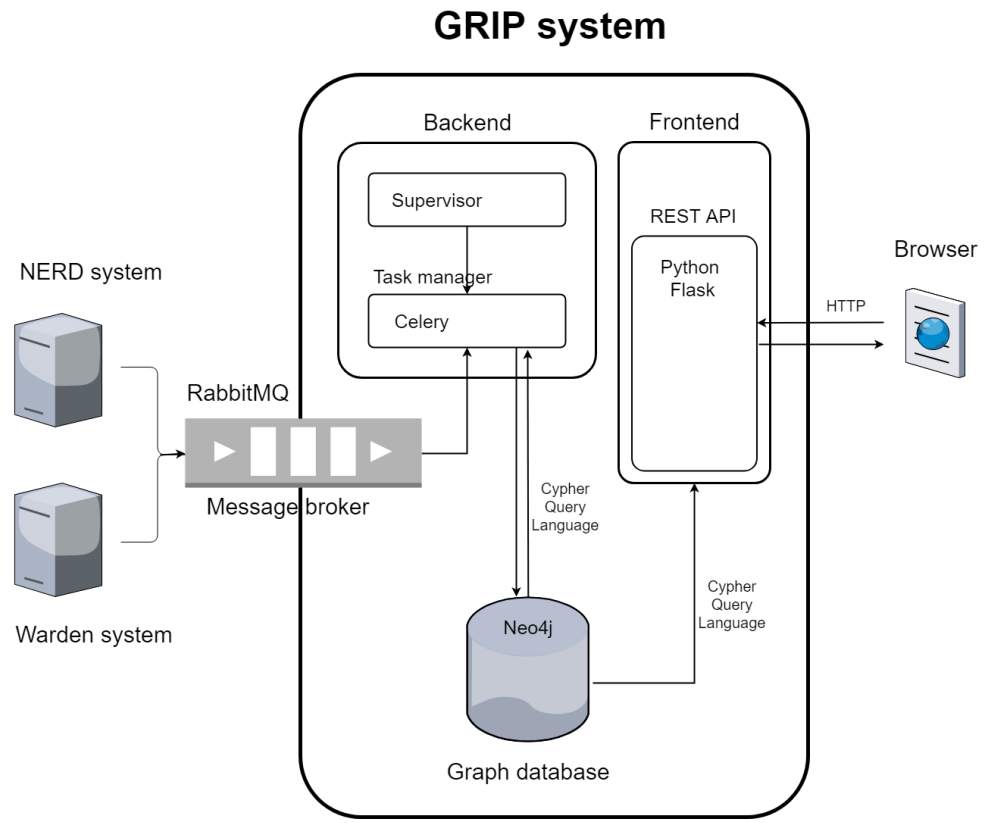
6.5 Architektura systému

Systém GRIP zobrazený na Obrázku 6.2 se skládá ze 3 částí:

- získání a uložení dat ze zdrojů včetně promazání starých dat (backend),
- zpracování požadavků o nalezení skupiny nebo síťové adresy (frontend),
- databáze entit.

První prvek systému — získání a uložení dat ze zdrojů včetně promazání starých dat — je vyřešeno frontou požadavků a zpracovávacím skriptem. Řešení má nainstalovanou službu RabbitMQ[4], která zajišťuje správu front. Vedle fronty běží služba Celery[5], která se stará o transport zprávy z *message*

[4] https://www.rabbitmq.com/
[5] http://www.celeryproject.org/



Obrázek 6.2: Architektura systému GRIP složená z úložiště, backendu a frontendu.

broker instance (v našem případě RabbitMQ) do skriptu v jazyce Python. Služba Celery se spouští jako jednotlivé *worker* procesy, které budou zpracovávat požadavky z fronty v RabbitMQ. Skript, který se stará o zpracování zpráv, při svém spuštění zavolá aplikaci Celery. Tento skript má připravené funkce pro zpracování různých druhů požadavků.

Zpracování požadavků o nalezení skupin, resp. síťových adres dle zadáné síťové adresy, resp. skupiny je řešeno pomocí REST API (více v kapitole 6.3.1). Byla vybrána aplikace Python Flask[6]. Důvodem k výběru bylo její snadné použití a zároveň použití v systému NERD. Systém odešle HTTP požadavek do Flask aplikace a ta vrátí odpověď.

Posledním prvkem je databáze entit. Je použita grafová databáze, ta propojuje záznamy mezi sebou, ukládání a vyhledávání je efektivnější než u relační databáze (zdůvodnění viz kapitola 6.4). Grafová databáze Neo4j[7] nabízí jak propojování záznamů mezi sebou bez ohledu na počet spojení (sousedů),

[6] http://flask.pocoo.org/
[7] https://neo4j.com/

tak vkládání zpráv o různých atributech. Neo4j je navíc opensource řešení a podporuje jazyk Python.

Kapitola 7

Implementace

Implementace řešení se skládá ze dvou částí:

- nastavení řešení a aplikování REST API,
- aplikování navrženého algoritmu na vytváření skupin.

Řešení zobrazené na Obrázku 7.1 se skládá z *message broker* instance RabbitMQ, grafové databáze Neo4j, služby Celery a služby Supervisor. Pro REST API řešení byl zvolen Python web framework Flask. Vývoj proběhl na serveru se systémem CentOS.

7.1 Supervisor

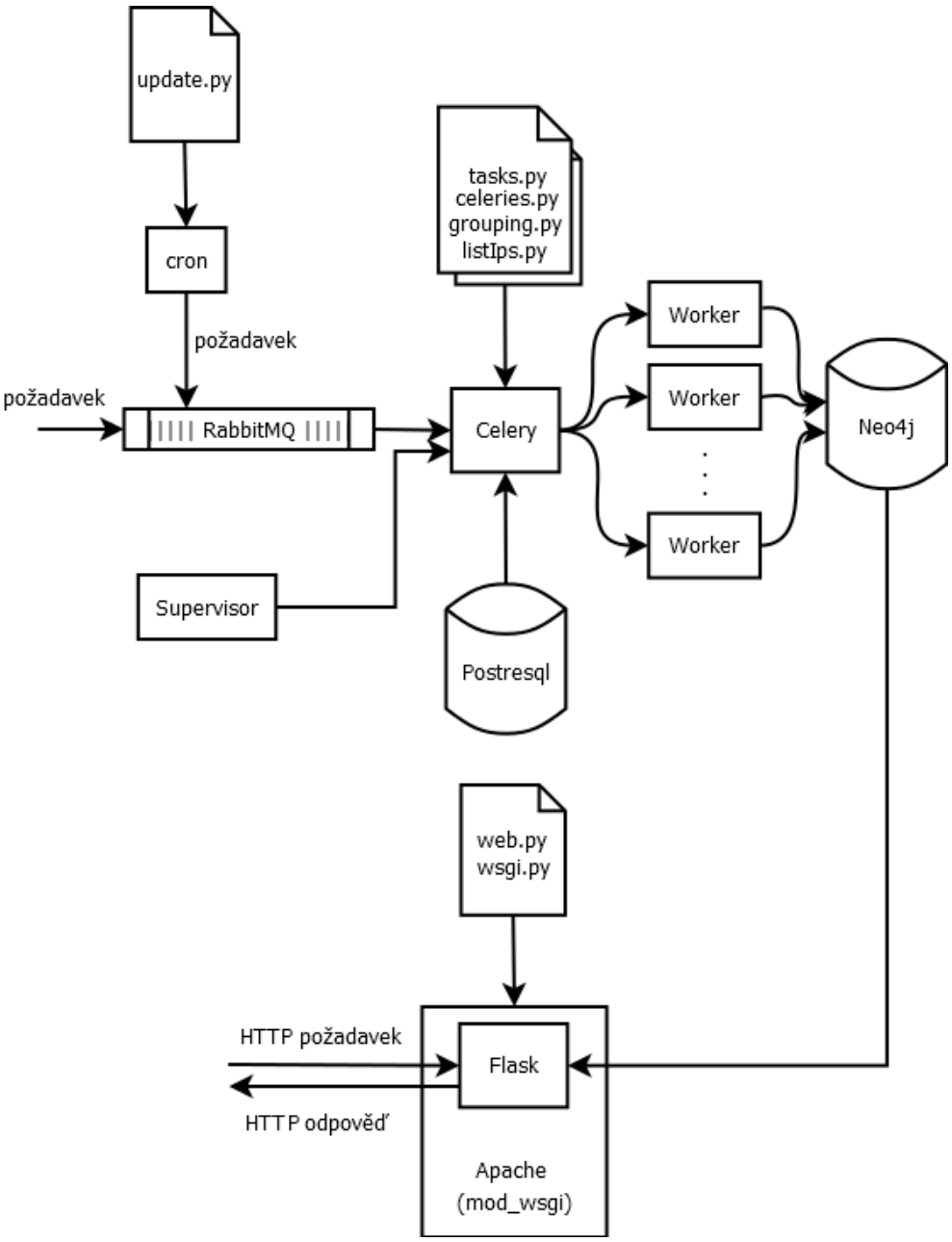
Supervisor[8] je UNIX klient/server software, který umožňuje uživateli kontrolovat procesy, které spustil. Tato služba zajišťuje opětovné spuštění procesů, které jsou v Supervisor nastavené, pokud by došlo k výpadku serveru. Není třeba vytvářet daemona (`init` skript), který by procesy hlídal.

Služba Supervisor byla nainstalována na server pomocí `easy_install`, což je součást `setuptools`. Po instalaci nebylo třeba Supervisor nijak nastavovat, pouze jej spustit.

Konfigurační soubory jsou ukládány do `/etc/supervisor.d/` jako soubory s koncovkou *.ini. Každý konfigurační soubor obsahuje:

- název, pomocí kterého lze později proces kontrolovat,
- příkaz, který se bude spouštět,
- adresář, který je procesem považován za domovský,
- atributy nastavující, zda bude proces spuštěn po startu serveru a kolik pokusů má na spuštění procesu,

[8] http://supervisord.org/



Obrázek 7.1: Schéma řešení systému GRIP.

- uživatel, pod kterým bude proces spuštěn,
- soubory pro logování.

Příklad takového souboru vypadá následovně:

```
[program:supervisor_grip_celery]
command=celery -A tasks.app -Q celery worker --loglevel=INFO
directory=/home/grip
autostart=true
autorestart=true
startretries=3
user=root
stderr_logfile=/var/log/grip/celery.err.log
stdout_logfile=/var/log/grip/celery.out.log
```

V systému GRIP služba Supervisor spravuje 2 procesy, spouští Celery a skript určený pro testování.

7.2 RabbitMQ

RabbitMQ[9] je *message broker*, jeho úlohou v systému GRIP je držet frontu požadavků. Byl vybrán díky tomu, že bude později využíván pro více účelů a v jazyce Python existuje dobře implementovaný modul `pika` pro manipulaci s RabbitMQ.

RabbitMQ byl nainstalován dle návodu vývojářů, byl použit návod pro systém CentOS. Během instalace byl vytvořen uživatel `rabbitmq`, který má vlastnická práva pouze k potřebným souborům, po spuštění služby běží RabbitMQ pod tímto uživatelem, což zaručuje bezpečnost.

Byly vytvořeny *exchange* `celery` a fronta `celery`. Do této fronty budou posílány požadavky s tělem zprávy speciálně definovaným tak, aby šlo později rozeznat, o jaký druh požadavku se jedná. Fronta je tak připravena do budoucna na více typů požadavků. Každá zpráva se tak povinně skládá z:

- označení druhu zprávy,
- identifikátoru zprávy ve formátu UUID,
- obsahu zprávy,
- volitelných argumentů pro zpracování.

Požadavek je do fronty poslán následujícím kusem kódu:

[9] https://www.rabbitmq.com/

```
credentials = pika.PlainCredentials('guest', 'guest')
parameters = pika.ConnectionParameters('localhost',
                                       5672,
                                       '/',
                                       credentials)
connection = pika.BlockingConnection(parameters)
channel = connection.channel()
channel.basic_publish(exchange='celery',
                      routing_key='celery',
                      body='{"task": "add",
                             "id": "' + str(uuid.uuid4()) + '",
                             "args": [' + json.dumps(idea) + '],
                             "kwargs": {}}',
                      properties=pika.BasicProperties
                          (content_type='application/json'))

connection.close()
```

Každá zpráva je označena jako `content_type='application/json'`, aby byla čitelná jako formát JSON.

7.3 Celery

Celery[10] je distribuovaná fronta úkolů založená na zpracování zpráv v reálném čase psaná v Pythonu. Úkoly jsou zpracovávány jedním nebo více procesy s použitím víceprocesového zpracování synchronně či asynchronně.

Celery v řešení hraje roli zpracovatele:

- požadavků na vkládání záznamů do databáze,
- požadavků na nalezení entit v databázi,
- požadavků na vymazání starých dat.

Celery bylo nainstalováno jako modul Pythonu, pomocí `pip`.

Procesy Celery byly spuštěny pomocí Supervisor, jako příkaz v konfiguračním souboru Supervisoru je

```
celery worker -A tasks --loglevel=INFO
```

který způsobí, že bude spuštěn základní proces Celery a dalších 8 *worker* procesů. Celery *worker* proces zpracovává jemu přidělené úlohy podle pravidel sepsaných v souboru `tasks.py`. Aby jednotlivé *worker* procesy nezpracovávaly

[10] http://www.celeryproject.org/

stejné úlohy, používá Celery víceprocesové zpracování, který předá jednu úlohu bezpečně pouze jednomu procesu.

Celery vyžaduje řešení, které mu bude předávat úlohy a které pak zpětně přijme výsledky zpracování úloh. Takovým řešením je obecně *message broker*. Celery podporuje příjem zpráv z více typů *message brokers*, byl vybrán RabbitMQ.

V souboru `tasks.py` jsou definovány funkce, které budou zpracovávat příchozí zprávy. Tyto funkce jsou označeny `@app.task`, to znamená, že jsou součástí Celery řešení. Celé označení vypadá následovně:

```
@app.task(serializer='json', name='add')
```

Funkce čtou zprávy ve formátu JSON. Každá funkce dostane jen zprávu s označením dané funkce, tím je zajištěno, že funkce bude umět zpracovat zprávu. Tento soubor importuje soubor `celeries.py`, který obsahuje definování aplikace Celery, tedy název (pouze označení) a spojení s *message broker* instancí včetně spojení s *message broker* instancí pro vracení výsledků.

Celery aplikace byla spuštěna následujícím příkazem:

```
app = Celery('celery',
             backend='amqp://localhost:5672',
             broker='amqp://localhost:5672')
app.start()
```

7.4 Neo4j

Neo4j[11] je jedna z nejrozšířenějších grafových databází. Její úlohou v řešení je držet informace o událostech, IP adresách, které se událostí týkají, a skupinách IP adres a tyto entity propojovat. Neo4j lze kontrolovat v jazyce Python.

Neo4j používá vlastní dotazovací jazyk Cypher inspirovaný SQL jazykem a se syntaxí vizuálně podobné „ASCII art“. Příklad dotazu, který vytvoří uzel Ip se jménem `10.0.1.123` a uzel Note s atributem `id` s hodnotou `abcd` a tyto dva uzly spojí vztahem `DESCRIBES`, vrací vložený uzel Ip:

```
CREATE (ip:Ip {name: 10.0.1.123})
CREATE (note:Note {id: abcd})
CREATE (note)-[:DESCRIBES]->(ip)
RETURN ip
```

Následující ukázka znázorňuje dotaz, který hledá všechny uzly Group, které jsou ve vztahu `PARTS_OF` s nějakým uzlem, který je ve vztahu `DESCRIBES` s uzlem Note s id `efgh`, vrací jméno a atribut `reason` nalezeného uzlu Group:

[11] https://neo4j.com/

```
MATCH (note:Note {id: efgh})
MATCH (note)-[:DESCRIBES]->(ip)
MATCH (ip)-[:PARTS_OF]->(group)
RETURN group.name, group.reason ORDER BY group.name,
    group.reason
```

Neo4j byl nainstalován jako server pro systém CentOS dle návodu vývojáře. Aby bylo možné databázi spravovat ze skriptů, byl do Pythonu doinstalován modul `neo4j-driver`. Databázi nebylo třeba dále nastavovat, pouze bylo nutno vytvořit nového uživatele. To je opatření přímo od Neo4j, zamezuje tím použití výchozího účtu.

7.5 Python Flask

Flask[12] je web framework Pythonu. Vytváří REST API, pomocí kterého je možné vyhledávat entity z grafové databáze. Lze poslat tři druhy požadavků:

- požadavek na seznam skupin,
- požadavek na seznam skupin, ve kterých se vyskytuje daná IP adresa,
- požadavek na seznam IP adres ve skupině daného názvu.

Flask byl nainstalován jako modul Pythonu. API je napsáno v souboru `web.py`, kde jsou definovány všechny end pointy a samotný start aplikace. Aplikace startuje s atributem `threaded=True`, což znamená, že je povoleno vícevláknové zpracování. Jsou definovány 3 end pointy pro každý druh požadavku:

- `/grip/groups`
- `/grip/groups/[IP_ADRESA]`
- `/grip/group/[ID_SKUPINY]`

Definice end pointu se skládá z definování cesty end pointu, metody HTTP a navrácení výsledků v podobě HTTP odpovědi. Všechny end pointy jsou definovány jako GET metody. Výsledek je vracen funkcí `jsonify` aplikace Flask. Tato funkce způsobí, že je výsledný seznam IP adres nebo skupin poslán jako `Flask.Response()` objekt, který nutně obsahuje `content_type` hlavičku `'application/json'`. Pokud by byl výsledek vrácen jako `json.dumps()`, byl by vracen jako kódovaný string a chyběla by mu MIME type hlavička.

[12] http://flask.pocoo.org/

7.5.1 Požadavek na seznam všech skupin

Definice end pointu `/grip/groups/` vypadá následovně:

```
@app.route('/groups/', methods=['GET'])
def find_list():
    ...
    return jsonify(list)
```

Nástrojem, který vrátí výsledek HTTP dotazu, je `curl`. `curl` je konzolová aplikace podporující HTTPS. Jeho použití na daném end pointu je naznačeno na následující ukázce:

```
curl https://localhost/grip/groups/
```

Odpověď API vypadá následovně:

```
[
  [
    "7EXH1K8L5304RXRHO82AFN43ZU4PYOBO",
    "Ideal match"
  ],
  [
    "CTO8FJVHHII1MA7OB5O82T2D2TW18GXA",
    "Ideal match"
  ],
  [
    "VJCKVRQWH9EK9KTB9V7XKAAQ0N3PQ5FP",
    "Individual match on category"
  ],
...
  [
    "XL6125WHQBXNZLBRNA7R3CK1S6HMRFFM",
    "Ideal match"
  ]
]
```

7.5.2 Požadavek na seznam skupin dle dané IP adresy

Definice end pointu `/grip/groups/[IP_ADRESA]` vypadá následovně:

```
@app.route('/groups/<int:ip1>.<int:ip2>.<int:ip3>.<int:ip4>',
    methods=['GET'])
def find_ip(ip1, ip2, ip3, ip4):
    ...
    return jsonify(list)
```

HTTP dotaz na daný end point je naznačen na následující ukázce:

```
curl https://localhost/grip/groups/10.1.11.12
```

Odpověď API vypadá následovně:

```
[
  [
    "7EXH1K8L5304RXRHO82AFN43ZU4PYOBO",
    "Ideal match"
  ],
  [
    "CTO8FJVHHII1MA70B5O82T2D2TW18GXA",
    "Ideal match"
  ]
]
```

7.5.3 Požadavek na seznam IP adres dle dané skupiny

Definice end pointu `/grip/group/[ID_SKUPINY]` vypadá následovně:

```
@app.route('/group/<string:botnet>', methods=['GET'])
def find_groups(botnet):
    ...
    return jsonify(list)
```

HTTP dotaz na daný end point je naznačen na následující ukázce:

```
curl https://localhost/grip/group/CTO8FJVHHII1MA70B5O82T2D2TW18GXA
```

Odpověď API vypadá následovně:

```
[
  "150.129.133.100",
  "150.129.133.112",
  "150.129.133.13",
  "150.129.133.136",
  "150.129.133.140",
  "150.129.133.153",
  "150.129.133.176",
  "150.129.133.178",
  "150.129.133.192",
  "150.129.133.196",
  "150.129.133.208",
  "150.129.133.21",
```

```
  "150.129.133.221",
  "150.129.133.23",
  "150.129.133.37",
  "150.129.133.4",
  "150.129.133.46",
  "150.129.133.61",
  "150.129.133.74",
  "150.129.133.98"
]
```

7.5.4 Nasazení aplikace Flask

Flask je nasazen na webovém serveru Apache pomocí modulu `mod_wsgi`. WSGI definuje rozhraní mezi webovým serverem a aplikací, která na něm má být spuštěna. Toto řešení bylo zvoleno proto, aby bylo API dostupné na standardním portu 443, přestože na serveru Apache již je spuštěn web.

Kromě definic end pointů byl vytvořen konfigurační soubor `grip.conf` pro Apache, který nastavuje přístup IP adresám nebo doménovým jménům, definuje parametry spuštění aplikace a nastavuje adresu WSGI souboru. Dále byl vytvořen soubor `wsgi.py`, který definuje, která aplikace ve kterém souboru má být spuštěna. API je přístupné pouze z localhostu.

7.6 Interakce jednotlivých částí

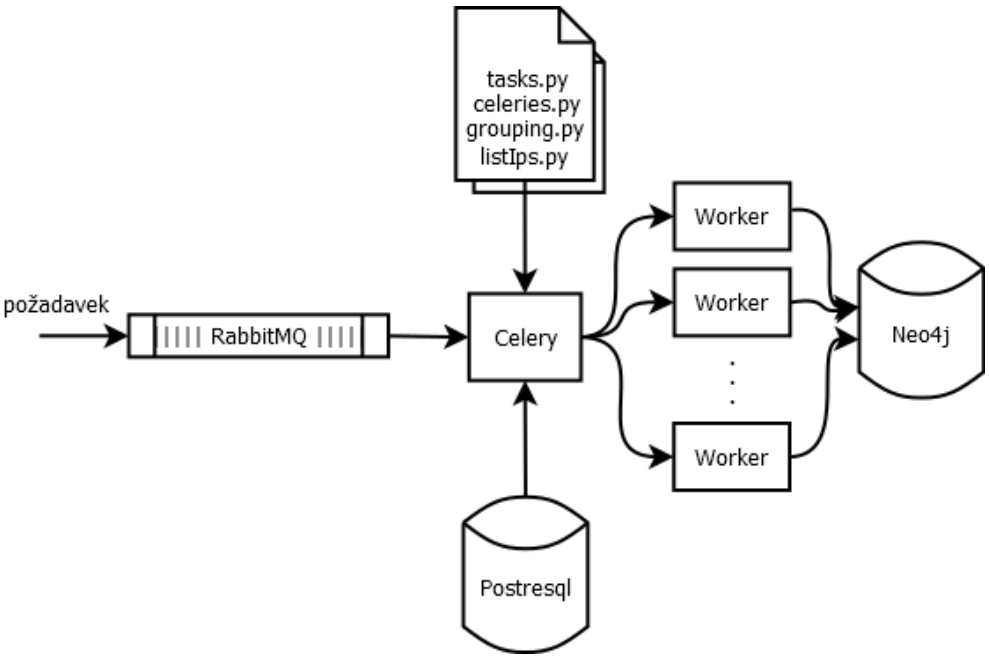
Všechny tyto součásti spolupracují a vytváří řešení pro kontrolu, zpracování a monitorování požadavků.

7.6.1 Přidání záznamu

Proces přidání záznamu vyžaduje frontu, zpracovatele a databázi, kam uložit záznam. Na Obrázku 7.2 je zobrazeno schéma přidávání záznamu.

V prvním kroku přijde požadavek na přidání záznamu o bezpečnostní události. Tento požadavek přijde ze systému Warden nebo systému NERD. Požadavek je vložen do služby RabbitMQ do *exchange* `celery` s *routing key* `celery`. Je označen identifikátorem UUID, typem požadavku `add` a samotná zpráva je čitelná ve formátu JSON. Zprávou je záznam o události ve formátu IDEA, zpráva tedy musela být mírně upravena, aby byla později čitelná pro grafovou databázi Neo4j. Funkce `createNode`, která zprávy transformuje, je v souboru `tasks.py`. Grafová databáze neumí vkládat datový typ slovník jako hodnotu klíče uzlu, proto jsou ve zprávě všechny hodnoty obsahující list slovníků přepsány. Příklad transformace atributů je uveden na následující ukázce s atributem `Source`.

Atribut `Source`:



Obrázek 7.2: Proces přidání záznamu z fronty RabbitMQ do grafové databáze Neo4j.

```
"Source": [
      {
         "Type": ["Phishing"],
         "IP4": ["192.168.0.2-192.168.0.5", "192.168.0.10/25"]
      }
   ]
```

byl přepsán na dva atributy:

```
"Source_Type": ["Phishing"],
"Source_IP4": ["192.168.0.2-192.168.0.5", "192.168.0.10/25"]
```

RabbitMQ pošle požadavek díky *routing key* do spravné fronty `celery`. Zde požadavek čeká, dokud si ho někdo nevyzvedne.

Služba Celery je připojená k frontě `celery`, pokud se ve frontě objeví nějaká zpráva, převezme ji a předá některému z volných *worker* procesů.

Worker proces pracuje se soubory `tasks.py`, `celeries.py`, `grouping.py` a `listIps.py`. V souboru `tasks.py` vybere funkci označenou jako funkce pro zpracování zpráv Celery, vybere takovou, jejíž název je shodný s označením zprávy v RabbitMQ — `add`. Provede funkci `add`, která zkontroluje, zda zpráva obsahuje všechny povinné atributy a do grafové databáze Neo4j předá list IP adres, které jsou v atributu `Source`. Tím se vytvoří skupina IP adres,

které jsou v jedné události a logicky proto spolu souvisí. Dále se spustí funkce, která hledá podobnosti v záznamech a IP adresách v čase (více o algoritmu v sekci 7.7). Výsledek znovu odešle databázi Neo4j.

Posledním krokem ve schématu je práce grafové databáze Neo4j. Funkce, které přidávají nebo modifikují data v databázi, jsou v souboru `tasks.py`. Algoritmus prohledává databázi na již existující IP adresy, které jsou do funkce předány. Pokud by IP adresa v databázi existovala, přidá k ní další událost a případně spojí s dalšími IP adresami. Pokud IP adresa v databázi neexistuje, přidá nový uzel Ip, přidá událost a spojí ji do případných skupin. Algoritmus přidávání do databáze je zobrazen na Obrázcích 7.3 a 7.4.

Tím je proces přidávání záznamu ukončen. Není nutné odpovědět systému, který odeslal požadavek. Pokud nebyl úspěšně dokončen, systém GRIP mezitím mohl přijmout další požadavky, které již byly obslouženy a mohly se na neúspěšně zpracovaný požadavek odkazovat, čekáním na úspěšné dokončení požadavku by se celý proces zpomalil.

7.6.2 Nalezení IP adresy nebo skupiny

Procesy nalezení IP adresy nebo skupiny jsou prováděny stejným principem přes REST API. Schéma toho řešení je na obrázku 7.5. Výhodou tohoto řešení je jednoduchost provedení požadavku.

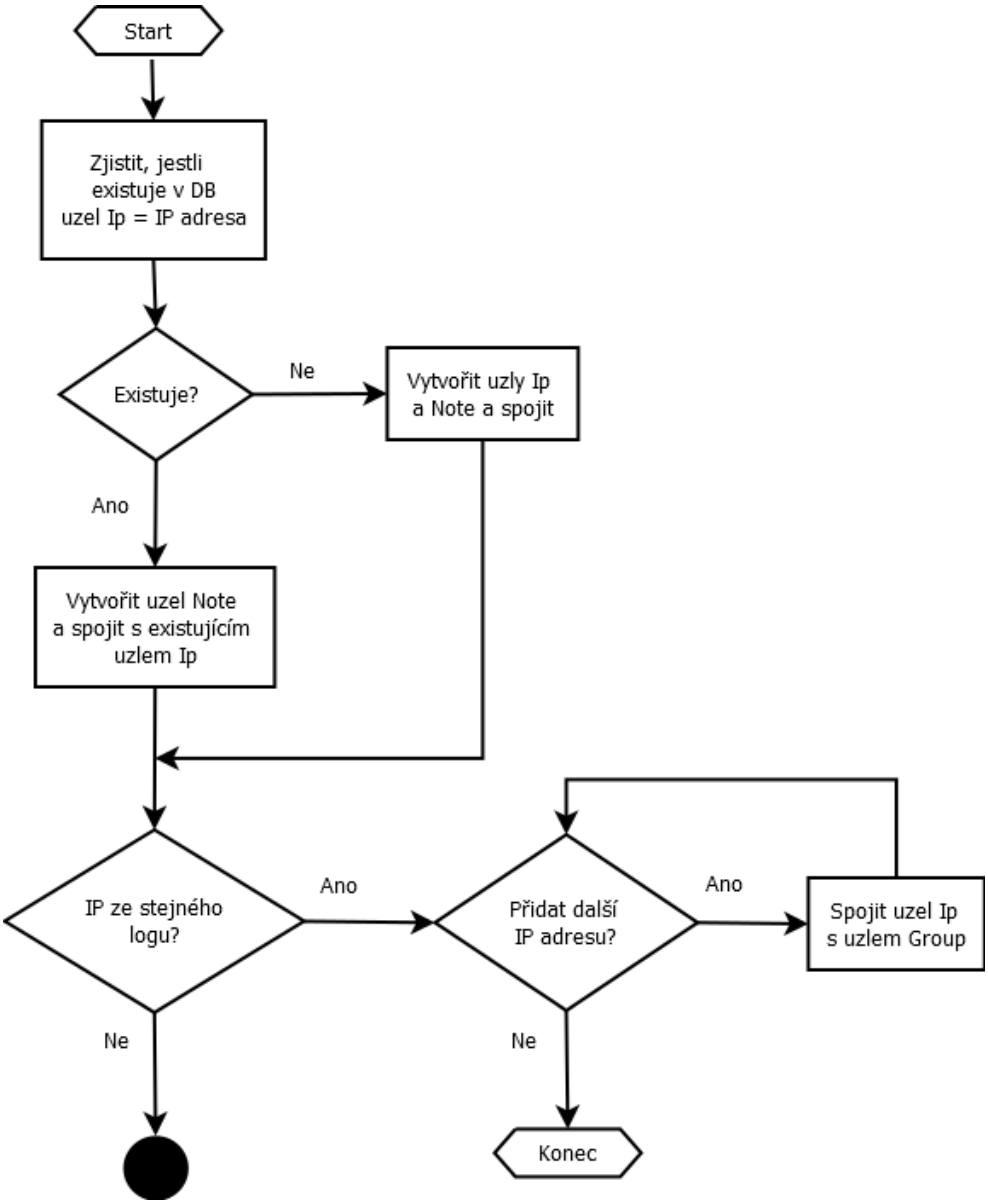
Požadavek v tomto případě přijde z localhostu — ze systému NERD.

Webový server Apache zjistí, že request je posílán na end point systému GRIP. Dle souboru `wsgi.py` je přesměrován k souboru `web.py`, kde najde definice end pointů. Spustí funkci určenou pro daný end point.

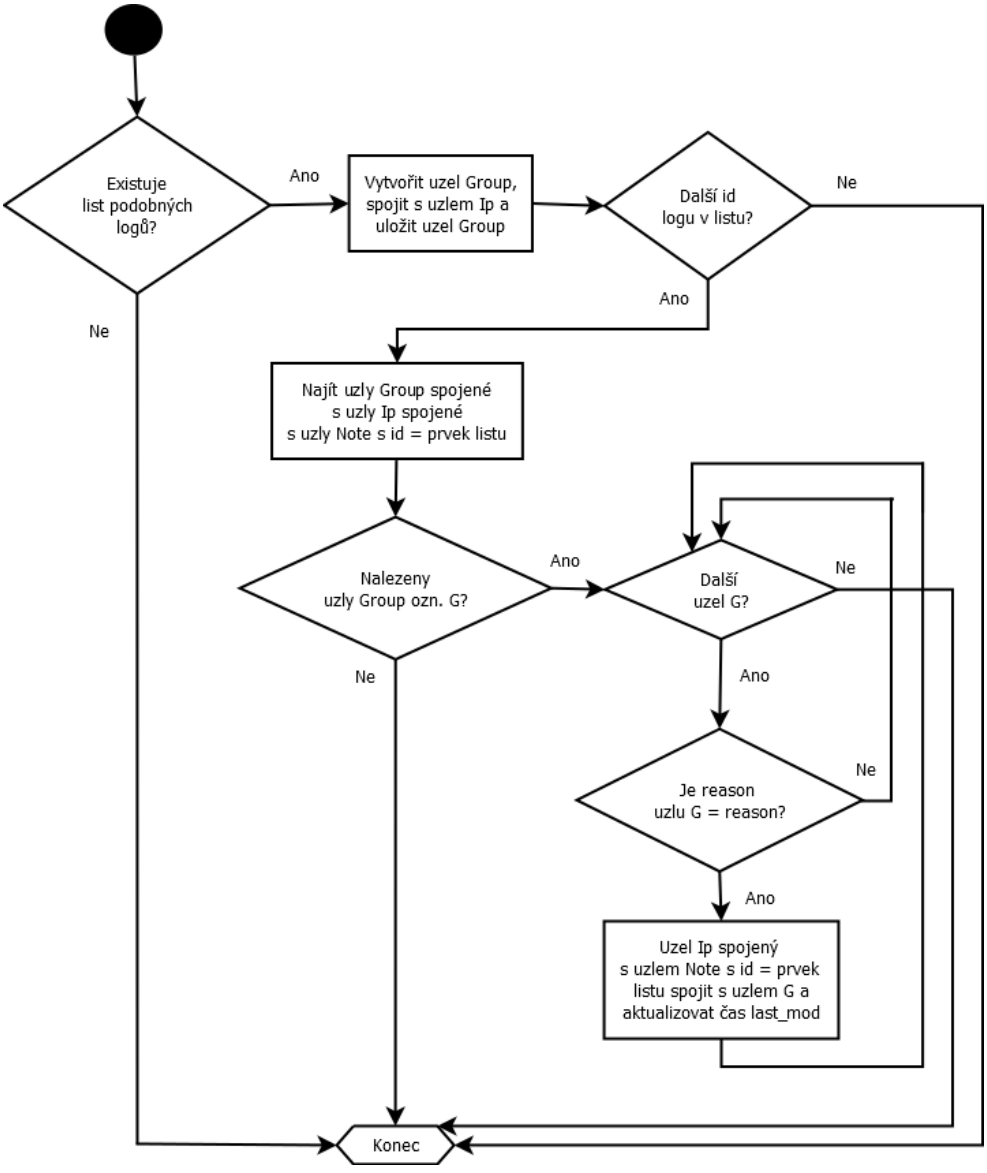
Funkce zavolá Neo4j, aby našel hledané entity. Databáze vrátí výsledek ve formátu Python slovníku. Výsledek JSON formátu je vrácen jako *Response* objekt s hlavičkou `content_type = 'application/json'` — odpověď přímo čitelná jako formát JSON.

Pokud se během testování ukáže, že vícevláknové zpracování pro obsluhu požadavků systému NERD nestačí, lze do definice aplikace Flask přidat atribut `processes=n`, kde `n` je číslo větší než 1. To způsobí, že bude spuštěno více procesů aplikace.

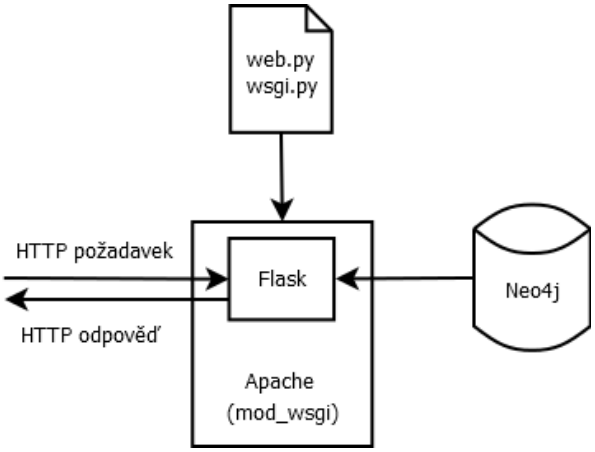
Pokud by ani tak aplikace nezvládla zátěž požadavků, je systém GRIP připraven na použití řešení pomocí fronty RabbitMQ. Rozdíl je v tom, že požadavek je ve frontě označen jako `find_ip` či `find_group` podle toho, zda hledá seznam skupin dle zadané IP adresy nebo seznam IP adres dle zadané skupiny. Databáze vyhledá entity a vrátí výsledek Celery workeru, který jej vloží do zpětné fronty nastavené v Celery. Pro každý vrácený výsledek je vždy vytvořena nová fronta s názvem stejným jako bylo ID požadavku, tím odesílatel požadavku najde vždy správnou odpověď ke svému požadavku. Fronta se po určitém čase sama smaže. Toto řešení má výhodu v tom, že požadavky mohou při zátěži serveru čekat ve frontě.



Obrázek 7.3: Algoritmus přidání záznamu do grafové databáze Neo4j, algoritmus řeší samotné vkládání dat do databáze, 1. část.



Obrázek 7.4: Algoritmus přidání záznamu do grafové databáze Neo4j, algoritmus řeší samotné vkládání dat do databáze, 2. část.



Obrázek 7.5: Nalezení IP adresy nebo skupiny v grafové databázi Neo4j dle HTTP požadavku.

7.6.3 Promazání záznamů

K promazání záznamů dochází jednou za 14 dní. Proto dává smysl nechat spouštění určit plánovačem `cron`.

V `cron` tabulce je definován záznam, který určí spuštění skriptu každý sudý týden. Skript odešle požadavek do fronty stejně jako v případě posílání požadavku o přidání záznamu. Požadavek bude tentokrát označen jako `update`.

Požadavek se přes Celery *worker* proces dostane k databázi Neo4j, kde provede promazání dat od starých údajů. Schéma algoritmu promazání dat je na schématu 7.6. Databáze nevrací žádný výsledek.

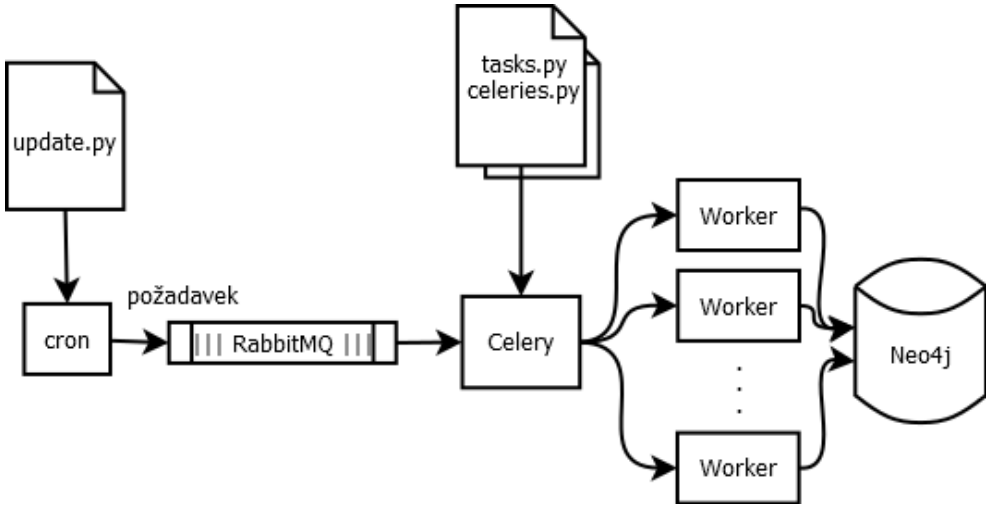
7.7 Algoritmus vytváření skupin

Na začátku algoritmus zjišťuje, zda záznam neobsahuje jednu z ignorovaných kategorií. V kapitole 5 bylo rozhodnuto, že budou ignorovány následující kategorie:

- *Other*
- *Suspicious.TOR*
- *Test*

Zároveň algoritmus zjišťuje, zda záznam obsahuje minimální počet atributů. Za minimální počet atributů jsou považovány následující atributy:

- Source
- Target



Obrázek 7.6: Promazání starých záznamů v grafové databázi Neo4j volané dle požadavku z plánovače cron.

- Category
- Description
- EventTime
- CeaseTime
- DetectTime

Pokud záznam tyto atributy neobsahuje, je ignorován, protože neobsahuje dostatek informací potřebných k základnímu rozlišení záznamů.

Pokud záznam obsahuje všechny potřebné atributy a vhodné kategorie, dostane se do další fáze algoritmu. Algoritmus se připojí k databázi systému NERD, kde jsou ukládány všechny záznamy událostí. Do této databáze se dotazuje na všechny záznamy staré 1 hodinu od času uvedeného v atributu `DetectTime`. Do databáze posílá dotaz:

```
SELECT idea
 FROM events
WHERE  detecttime > (now() at time zone 'utc')
   - interval '1 hour'
ORDER BY detecttime DESC;
```

Algoritmus získá zprávy v JSON formátu, které uloží v datovém typu slovník do pole. Toto pole poté prohledává.

Algoritmus se následně dostane do části, kde prohledává záznamy vybraných kategorií a zjišťuje, zda se nějaký jiný záznam neshoduje s právě kontrolovaným záznamem — hledá ideální shodu. Podmínky shody jsou nastaveny tak, že alespoň polovina IP adres atributů `Source` a `Target` kontrolovaného záznamu se musí shodovat s IP adresami ve stejných atributech v jiném záznamu. Tato část algoritmu se provádí jen na záznamech s kategoriemi, které byly výsledkem analýzy. Pouze u těchto kategorií je vyšší pravděpodobnost, že tímto algoritmem naleznou podobnost s jiným záznamem. Jedná se o kategorie:

- *Anomaly.Traffic*
- *Attempt.Exploit*
- *Attempt.Login*
- *Availability.DDoS*
- *Recon.Scanning*

Diagram této části algoritmu je zobrazen na Obrázku 7.7.

Další část algoritmu je rozčleněna na menší případy. Tyto případy jsou implementované pro různé kategorie. Záznamy různých kategorií jsou tak zpracovávány jiným způsobem, což zajistí přesnější výsledky. Schéma celého algoritmu je znázorněno na Obrázku 7.8.

Algoritmus umožňuje prohlédávat podobnost záznamů i pro různé kategorie záznamů. Jedná se o dvojice kategorií, u kterých dává smysl hledat jejich souvislost. Typickým příkladem je kombinace *Recon.Scanning* a *Attemp.Login*, kdy útočník nejprve skenuje síť, hledá zranitelnosti. Poté se pokouší o přihlášení do systému použitím nalezené zranitelnosti.

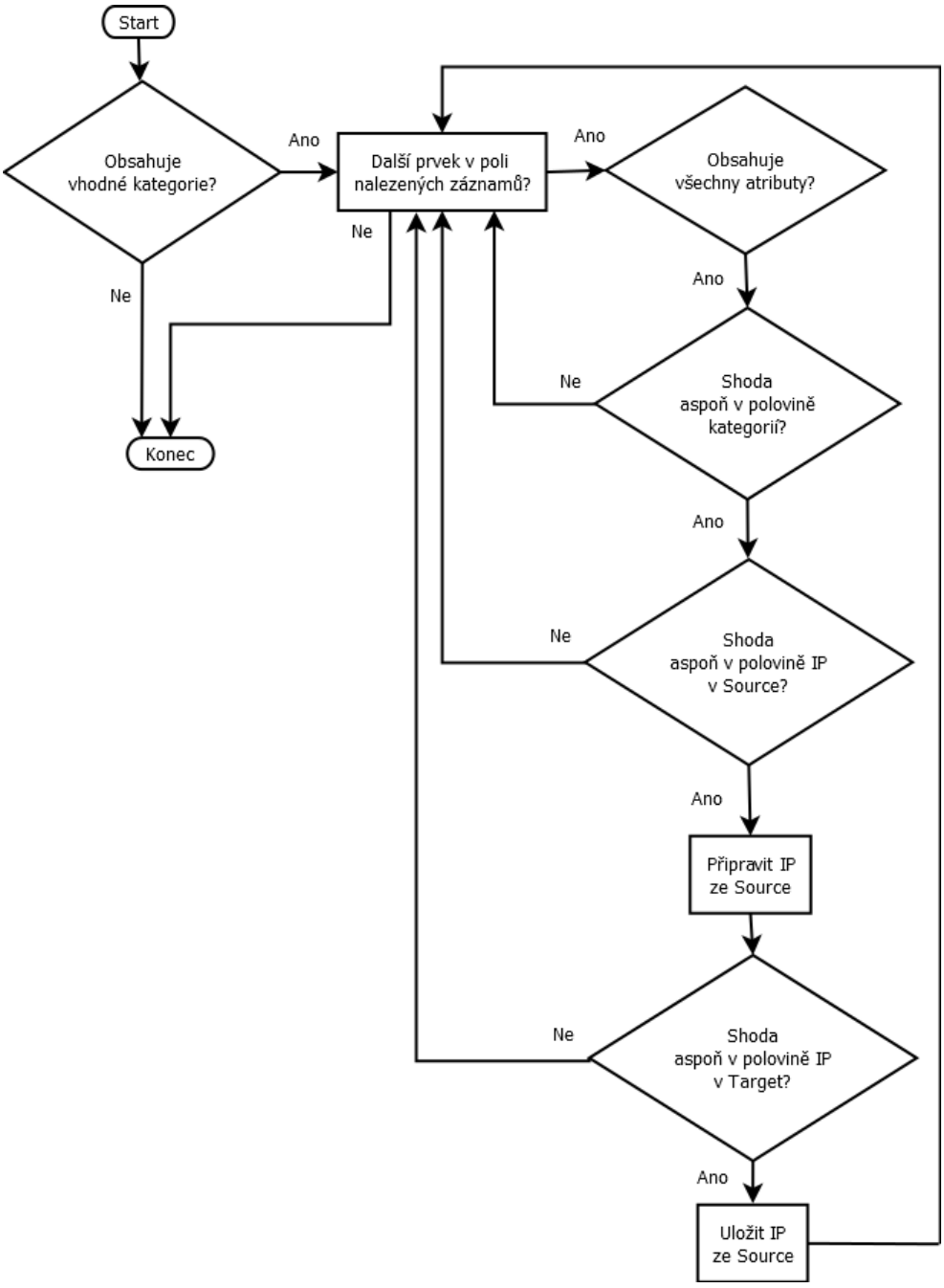
Podobnou kombinaci kategorií, které spolu mohou souviset, tvoří dvojice *Recon.Scanning* a *Attempt.Exploit*, kdy útočník skenuje síť a hledá zranitelnosti, které se poté pokusí využít pro škodlivou činnost. *Attempt.Exploit* často značí pokus o ovládnutí systému či instalaci nežádoucího softwaru.

Další dvojicí jsou kategorie *Recon.Scanning* a *Intrusion.Botnet*. Tato posloupnost událostí značí, že útočník nejprve skenoval síť a hledal v ní nějaké zranitelnosti, zranitelnost našel, využil jí a úspěšně kompromitoval systém, v tomto případě se systém stal součástí botnetu.

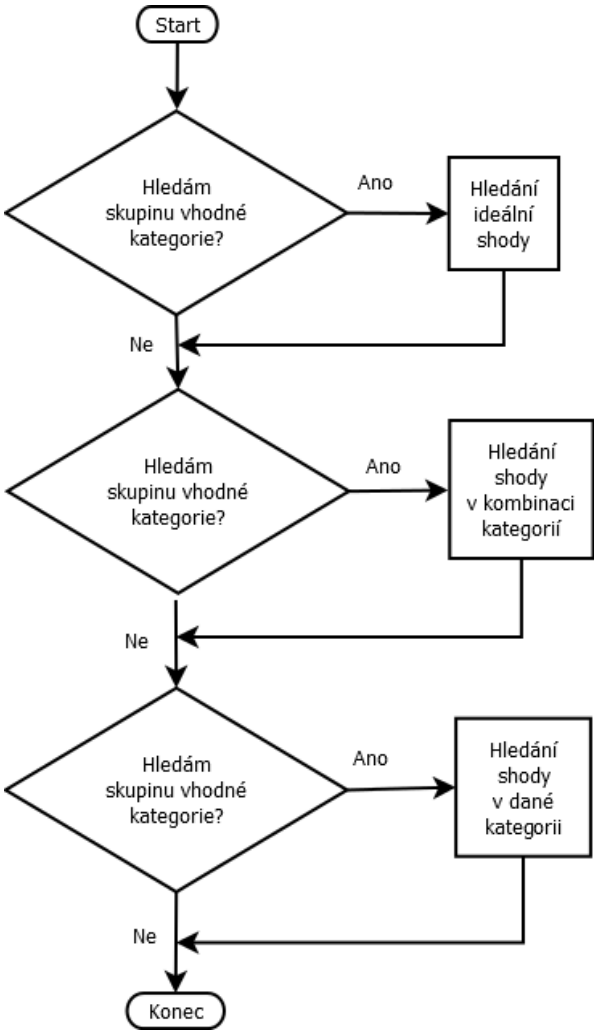
Tyto dvojice jsou kontrolovány na existenci alespoň jedné stejné IP adresy v atributech `Source` i `Target`. Dále je kontrolováno jméno atributu `Node`, které značí jméno detektoru, který událost detekoval. Algoritmus tak požaduje, aby oba záznamy z dvojice byly detekovány stejným detektorem.

Algoritmus následně obsahuje další části kódu, které prohledávají záznamy pouze určených kategorií a vyhledávají podobnost typickou pro tyto kategorie.

Rozdělováním algoritmu na menší části se pro některé kategorie záznamů může stát, že ve výsledku budou v několika skupinách z rozdílných důvodů.



Obrázek 7.7: Diagram části algoritmu hledající ideální shodu.



Obrázek 7.8: Diagram algoritmu vytváření skupin — vizualizace pořadí jednotlivých částí.

Algoritmus vrací všechny skupiny. Algoritmus na konci vrací pole slovníků, kde každý slovník představuje jednu skupinu. Každá skupina obsahuje pole IP adres a důvod, proč došlo ke spojení těchto IP adres. Výsledné pole je vstupem do funkce přidávající záznam do databáze Neo4j.

Kapitola 8

Testování

Testování systému GRIP bylo rozděleno na dvě části:

- testování funkčnosti systému,
- testování algoritmu vkládání.

8.1 Testování funkčnosti systému GRIP

Nejprve bylo provedeno testování funkčnosti samotného řešení systému GRIP. Je třeba zjistit, zda RabbitMQ fronta přijímá požadavky, zda je lze vybrat a zda je lze správně zpracovávat. K otestování byly vytvořeny skripty v jazyce Python, které posílaly fiktivní požadavky do fronty RabbitMQ. Bylo ověřeno, že spuštěný *message broker* RabbitMQ, služba Celery i grafová databáze Neo4j okamžitě reagují na přijatý požadavek a zpracují jej. Byly otestovány dva typy požadavku:

- `add`,
- `update`.

Výsledek zpracování požadavku typu `add` byl zkontrolován dotazem do grafové databáze Neo4j. Do databáze se lze připojit pomocí konzolové aplikace `cypher-shell`, která umožňuje provádět dotazy nad daty. Bylo ověřeno, že záznam v databázi má správný formát a hodnoty. Požadavek typu `update` byl testován při spouštění jednou za hodinu tak, že smazal všechny záznamy v databázi starší 15 minut. Byla ověřena korektní manipulace s časem záznamu v databázi. Při testování odesílání, zpracování a přijímání obou typů požadavků se nevyskytl žádný problém.

8.2 Testování algoritmu vytváření skupin

8.2.1 Statické testování

Algoritmus vkládání IP adres z událostí do skupin byl nejprve otestován manuálně. Nejprve byl otestován s použitím jedné zprávy ve formátu IDEA s kategorií *Recon.Scanning*. Tato kategorie byla vybrána z toho důvodu, že záznamů této kategorie je většina. Je tedy jistota, že algoritmus nějaké záznamy stejné kategorie najde, a zároveň je těchto kategorií tolik, že mohou už od začátku poskytovat náhled na dobu nejdelšího běhu algoritmu.

Testování skupiny *Recon.Scanning* na počet IP adres v atributech `Source` a `Target` ukázalo, že většina záznamů obsahuje pouze jednu IP adresu v obou atributech. Záznamů s dvouciferným počtem záznamů bylo méně než $\frac{1}{30}$. Záznamů se 100 a více IP adresami v atributu bylo velmi málo, méně než 1 ‰ případů.

Při testování se ukázalo, že během jedné hodiny je v databázi událostí uloženo mezi 50 000 a 70 000 záznamy. Dotaz do databáze, který vrátí tyto záznamy za poslední hodinu, trvá průměrně 5 vteřin. Dle počtu záznamů z dotazu do databáze to znamená, že do systému NERD přichází průměrně 18 záznamů za vtěřinu. Doba celého zpracování požadavku systémem GRIP trvá mezi 5–8 vtěřinami dle obsahu zprávy.

Služba Celery běží víceprocesově, fronta požadavků na přidání záznamů tak je odebírána více procesy. Při počátcích testování se ukázalo, že více než polovina záznamů neobsahuje dostatek informací pro určení skupiny. Další záznamy jsou ignorovány kvůli nevhodné kategorii, např. kategorii *Test*. Tím pádem je více než polovina záznamů zahozena už na začátku zpracovávání.

8.2.2 Spuštění do reálného provozu

Při testování algoritmu v reálném provozu byly pozorovány zejména následující faktory:

- zatížení serveru hledáním skupin,
- zatížení databáze systému NERD,
- požadavky grafové databáze Neo4j na zdroje.

Testování probíhalo tím způsobem, že záznamy, které byly posílány a ukládány do databáze PostgreSQL systému NERD, byly zároveň posílány do fronty `idea-nerd2grip` služby RabbitMQ. Fronta byla nastavena s časovým limitem 60 vteřin, to znamená, že po 60 vteřinách, kdy zpráva nebude vybrána z fronty, bude odstraněna. Testovat systém GRIP bylo umožněno kdykoli spuštěním skriptu vybírajícího zprávy z fronty `idea-nerd2grip` posílající je do fronty `celery`. Zprávy se tak nehromadily ve frontě. Na frontu `celery` byly připojeny čekající procesy služby Celery. Jakmile se ve frontě objevila zpráva,

procesy Celery ihned začaly zprávu zpracovávat. Během testování v reálném provozu nebyla zaznamenána žádná chyba související s během systému GRIP.

Ukázalo se, že po zpracování 6 000 záznamů algoritmus vytvoří přibližně 400 různých skupin síťových adres. Většinu skupin tvoří skupiny vytvořené částí algoritmu hledající ideální shodu. Většina záznamů, jejichž IP adresy jsou v nějaké skupině, jsou záznamy s kategorií *Recon.Scanning*. To je způsobeno velkým množstvím záznamů s touto kategorií.

Závěr

V rámci této diplomové práce byl vytvořen systém pro evidenci skupin podezřelých síťových adres (GRIP). Hlavní motivací k této práci byla snaha o vytvoření chybějícího nástroje pro analýzu informací o bezpečnostních událostech, který by se při vyhodnocování zaměřoval na podobnosti různých IP adres (entit). Shlukování entit do skupin vychází z předpokladu, že podezřelé entity nakažené stejným nebo podobným typem malwaru se budou chovat podobně a pravděpodobně budou vykazovat podezřelou aktivitu v podobném čase.

Důležitou součástí práce je vytvořený algoritmus, který dle nahlášených bezpečnostvních událostí určuje skupiny IP adres napříč časem. Tento algoritmus byl navržen s ohledem na potřebu zpracování dat v reálném čase a průběžné promazávání rychle stárnoucích dat. Algoritmus byl navržen, implementován a otestován, čímž vznikl funkční prototyp systému GRIP.

Navržený algoritmus vychází z důkladné analýzy dostupných testovacích dat, která obsahovala vzorky nahlášených bezpečnostních událostí z národní akademické sítě CESNET2 a spolupracujících organizací. Dle výsledků analýzy bylo rozhodnuto, na jaké atributy bude brát algoritmus ohled, které atributy budou povinné a jaké kategorie záznamů budou ignorovány. Analýza vstupních dat probíhala na vzorku dat o velikosti 1 000 000 záznamů o bezpečnostních událostech. Bylo zjištěno, že více než 75 % záznamů obsahuje kategorii *Recon.Scanning* (skenování sítě). Dále 25 % záznamů obsahuje kategorii *Test*, která slouží pro testovací účely, proto je ignorována. Analýza ukázala, že jednotlivé kategorie jsou, co se týče atributů uvedených v záznamu, vzájemně rozdílné. Bylo tak možné lépe vystihnout danou kategorii při hledání společných znaků v algoritmu pro určení skupin IP adres. Algoritmus byl inspirován řešením projektu SABU [19].

Systém GRIP je implementován v jazyce Python a je postaven nad existujícími open source technologiemi. Systém je tvořen grafovou databází Neo4j, která spojuje všechny uložené entity, dále je součástí systému *message broker* RabbitMQ, který spouští frontu požadavků na přidání záznamu do databáze.

Dalším prvkem je služba Celery, která slouží jako reprezentace úkolů, zpracovává požadavky a posílá výsledky do grafové databáze. Systém GRIP přijímá požadavky typu přidání záznamu do databáze a promazání záznamů v databázi prostřednictvím fronty RabbitMQ.

Systém GRIP umožňuje získávat informace o uložených entitách a jejich vazbách tvořících skupiny pomocí REST API. Toto rozhraní bylo vytvořeno pomocí standardního webového serveru doplněného o aplikaci Python Flask. Požadavky na nalezení skupin nebo IP adres jsou posílány přes protokol HTTPS. Celé toto řešení bylo otestováno.

Dalším krokem, který je plánovaný po odevzdání této diplomové práce, je integrace systému GRIP s ostatními systémy, které se ve sdružení CESNET používají pro automatickou analýzu bezpečnostních událostí, především se systémem NERD. Informace z GRIP je plánované využít například pro obranu infrastruktury. Další vývoj by měl také směřovat ke zkvalitnění zdrojů dat, které jsou podstatné pro vytváření skupin. Do budoucna se také nabízí vylepšit samotný heuristický algoritmus seskupování entit pomocí moderních metod jako jsou metody strojového učení.

Literatura

[1] CESNET: CESNET/Nemea. https://github.com/CESNET/Nemea, cit. 2018-01-18.

[2] CESNET: Network Entity Reputation Database (NERD). https://nerd.cesnet.cz/, cit. 2018-01-18.

[3] CESNET: Warden. https://warden.cesnet.cz//cs/index, 2017-07-11, cit. 2018-01-18.

[4] Dreamer, F.: Ham v Spam: what's the difference? https://blog.barracuda.com/2013/10/03/ham-v-spam-whats-the-difference/, 2013-10-03, cit. 2018-03-22.

[5] Fisher, T.: What Is Malware? https://www.lifewire.com/what-is-malware-2625933, 2018-01-16, cit. 2018-03-22.

[6] PaloAlto: WHAT IS A DENIAL OF SERVICE ATTACK (DoS)? https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos, 2018, cit. 2018-03-20.

[7] Krčmář, P.: Pronajmu botnet, ceník přiložen... aneb jak se dělá DDoS. https://www.root.cz/clanky/pronajmu-botnet-cenik-prilozen-aneb-jak-se-dela-ddos/, 2013-03-08, cit. 2018-03-18.

[8] Grossn, G.: Botnet Detection and Removal: Methods & Best Practices. https://www.alienvault.com/blogs/security-essentials/botnet-detection-and-removal-methods-best-practices, 2015-11-03, cit. 2018-02-15.

[9] Symantec: What is a Botnet? https://us.norton.com/internetsecurity-malware-what-is-a-botnet.htm, 2018, cit. 2018-03-18.

[10] Ragan, S.: Here are the 61 passwords that powered the Mirai IoT botnet. https://www.csoonline.com/article/3126924/security/here-are-the-61-passwords-that-powered-the-mirai-iot-botnet.html, 2016-10-03, cit. 2018-03-20.

[11] Radware: A Quick History of IoT Botnets. https://blog.radware.com/uncategorized/2018/03/history-of-iot-botnets/, 2018-01-08, cit. 2018-05-01.

[12] Bursztein, E.: Inside Mirai the infamous IoT Botnet: A Retrospective Analysis. https://www.elie.net/blog/security/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis, 2017-12-01, cit. 2018-02-12.

[13] Bartoš, V.: NERD: Úvod pro vývojáře. https://github.com/CESNET/NERD/raw/master/doc/NERD_intro_for_developers_(CZ).pdf, 2018-03-14, cit. 2018-03-25.

[14] Čejka, T.: NEMEA. http://nemea.liberouter.org/, 2018, cit. 2018-03-25.

[15] Cejka, T.; Bartos, V.; Svepes, M.; aj.: NEMEA: A Framework for Network Traffic Analysis. In *12th International Conference on Network and Service Management (CNSM 2016)*, 2016, doi:10.1109/CNSM.2016.7818417. Dostupné z: http://dx.doi.org/10.1109/CNSM.2016.7818417

[16] CESNET: Warden: O projektu. https://warden.cesnet.cz/cs/about_project, 2016-04-07, cit. 2018-03-25.

[17] CESNET: Warden: Architektura. https://warden.cesnet.cz/cs/architecture, 2015-12-23, cit. 2018-03-25.

[18] CESNET: Intrusion Detection Extensible Alert. https://idea.cesnet.cz/en/index, 2017-12-06, cit. 2018-03-22.

[19] CESNET: SABU. https://sabu.cesnet.cz/cs/start, 2018-03-01, cit. 2018-03-23.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

DDoS Distributed Denial of Service

DoS Denial of Service

GRIP Group of IPs

HTTP Hypertext Transfer Protocol

IDEA Intrusion Detection Extensible Alert

IDS Intrusion Detection System

IP Internet Protocol

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

IRC Internet Relay Chat

NEMEA Network Measurements Analysis

NERD Network Entity Reputation Database

REST Representational State Transfer

SMTP Simple Mail Transfer Protocol

WSGI Web Server Gateway Interface

Příloha B

Manuál pro konfiguraci systému GRIP

Manuál pro konfiguraci systému GRIP popisuje krok za krokem, jak nakonfigurovat řešení systému. Systém byl konfigurován na serveru s operačním systémem CentOS s webovým serverem Apache.

B.1 Supervisor

Nainstalovat python-setuptools:

```
pip install setuptools
```

Nainstalovat supervisor:

```
easy_install supervisor
```

Spustit supervisor:

```
/bin/supervisord
```

Do adresáře s konfiguračními soubory `/etc/supervisord.d/` vložit soubor `supervisor_grip_celery.ini`:

```
cp supervisor_grip_celery.ini /etc/supervisord.d/
```

B.2 RabbitMQ

Nainstalovat Erlang:

```
yum install erlang
```

Nainstalovat RabbitMQ:

```
yum install rabbitmq-server
```

Spustit RabbitMQ:

```
/sbin/service rabbitmq-server start
```

Povolit plugin `rabbitmq_management`:

```
rabbitmq-plugins enable rabbitmq_management
```

Stáhnout RabbitMQ Admin Management Command Line Tool:

```
wget http://{hostname}:15672/cli/rabbitmqadmin
```

RabbitMQ Admin je poté spouštěn jako Python script: `python rabbitmqadmin`. Nainstalovat Python modul `pika`:

```
pip install pika
```

B.3 Celery

Nainstalovat Celery:

```
pip install celery
```

Zkopírovat soubory `celeries.py`, `tasks.py`, `grouping.py` a `listIps.py` do libovolného adresáře. Cestu k adresáři přepsat v souboru `/etc/supervisord.d/supervisor_grip_celery.ini`.

B.4 Neo4j

Nainstalovat Neo4j:

```
wget http://debian.neo4j.org/neotechnology.gpg.key
rpm --import neotechnology.gpg.key
cat <<EOF>  /etc/yum.repos.d/neo4j.repo
    [neo4j]
    name=Neo4j Yum Repo
    baseurl=http://yum.neo4j.org/stable
    enabled=1
    gpgcheck=1
EOF
yum install neo4j
```

Spustit Neo4j:

```
systemctl enable neo4j
```

Nainstalovat Python modul `neo4j-driver`:

```
pip install neo4j-driver
```

S databází lze v příkazové řádce pracovat pomocí `cypher-shell` (výchozí přihlašovací údaje jsou neo4j/neo4j):

```
cypher-shell
```

Vytvořit nového uživatele Neo4j po přihlášení do `cypher-shell`:

```
CALL dbms.security.createUser('name', 'password', false);
```

B.5 Interakce jednotlivých částí

Vytvořit v RabbitMQ exchange `celery` typu `direct` a frontu `celery`:

```
python rabbitmqadmin declare exchange name=celery type=direct
python rabbitmqadmin declare queue name=celery durable=true
```

Pro spuštění skriptu v Supervisoru znovu načíst všechny konfigurační soubory a aktualizovat je:

```
supervisorctl reread
supervisorctl update
```

B.6 REST API

Server musí mít nainstalovaný web server a povolené WSGI. Manuál dále popisuje konfiguraci při instalovaném Apache serveru s modulem `mod_wsgi` Do adresáře `/etc/httpd/conf.d/` vložit soubor `grip.conf`:

```
cp grip.conf /etc/httpd/conf.d/
```

Zkopírovat soubory `web.py`, `wsgi.py` a složky `templates` a `static` do libovolného adresáře. Cestu k adresáři přepsat v souboru `/etc/httpd/conf.d/grip.conf`. Restartovat konfiguraci Apache:

```
/sbin/service httpd reload
```

Příloha C

Obsah přiloženého CD

```
readme.txt ..... stručný popis obsahu CD
src
    impl ..... zdrojové kódy implementace
    thesis ..... zdrojové soubory práce ve formátu LaTeX
        img ..... adresář obrázků v práci
text ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
```