

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Vztahová analýza síťových toků

BAKALÁŘSKÁ PRÁCE

Aneta Jablunková

Brno, podzim 2021

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracovala samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používala nebo z nich čerpala, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Aneta Jablunková

Vedoucí práce: RNDr. Milan Čermák, Ph.D.

Poděkování

Ráda bych poděkovala panu RNDr. Milanu Čermákovi, Ph.D. za odborné vedení a cenné rady při zpracování bakalářské práce.

Shrnutí

Bakalářská práce se zabývá rozšířením nástroje GRANEF o zpracování a analýzu síťových toků. Sada nástrojů GRANEF je vyvíjena na Masarykově univerzitě a umožňuje analyzovat síťový provoz zachycený ve formě paketů a podporuje řešení bezpečnostních incidentů pomocí nového způsobu analýzy síťového provozu na základě vztahů. Bezpečnostní týmy běžně pracují se síťovými toky a nemusí mít přístup k paketovému zachytu. Nabízí se tedy rozšíření nástroje GRANEF o nahrání síťového provozu v podobě síťových toků do databáze Dgraph a o možnost vložená data analyzovat. V práci jsou shrnuty poznatky o monitorování síťového provozu, grafových databázích a databáze Dgraph. Dále je popsáno rozšíření databázového modelu a samotná implementace rozšíření ve všech relevantních modulech nástroje GRANEF. Výsledkem je rozšíření, které umožňuje zpracovávat a analyzovat síťové toky stejným způsobem jako v případě paketového zachytu.

Klíčová slova

analýza, síťový provoz, grafová databáze, síťové toky, dgraph, monitorování síťového provozu, GRANEF

Obsah

Úvod	1
1 Monitorování síťového provozu	2
1.1 Síťové toky	3
1.2 Architektura monitorování síťových toků	5
1.2.1 Pozorování paketů	5
1.2.2 Měření a export síťových toků	6
1.2.3 Sběr dat	7
1.2.4 Analýza dat	8
2 Analýza síťového provozu pomocí grafové databáze	10
2.1 Grafové databáze	10
2.2 Dgraph	11
2.3 Sada nástrojů GRANEF	14
3 Extrakce a transformace síťového provozu	18
3.1 Extrakce dat	18
3.2 Databázové schéma	19
3.3 Transformace dat	22
3.4 Parsery podle typu protokolu	24
4 Analytické dotazy na zvoleném datasetu	26
4.1 Dataset	26
4.2 Příklad možné analýzy na základě alertu	28
5 Evaluace implementace	39
Bibliografie	42

Úvod

Analýza síťových toků je součástí monitorování síťového provozu, který zajišťuje funkčnost a zabezpečení sítě. Skládá se ze čtyř fází: pozorování paketů, exportu dat, sběru dat a následnou analýzou [1]. Analýzu zachytu lze provádět různými způsoby, které se liší stylem dotazování a pohledem na analyzovaná data. Jedním z nových způsobů vyvíjených na CSIRT-MU je využití grafových databází, které díky podpoře škálovatelnosti mají úplně nový pohled na data a práci s nimi.

Cílem práce je rozšířit sadu nástrojů GRANEF (Graph-Based Network Forensics) o zpracování a analýzu síťových toků. Aktuální implementace nástroje GRANEF umožňuje bezpečnostním analytikům analyzovat zachycený síťový provoz ve formě paketů pomocí vztahů a řešit nalezené bezpečnostní incidenty. Pro znázornění výsledků analýz využívá nástroj GRANEF grafovou databázi Dgraph. Aby mohly být zachycené síťové toky nahrány do databáze je nutné je převést do RDF [2] trojic a databázové schéma vhodně upravit (aktuálně je navrženo pro analýzu dat z paketů).

Bakalářská práce je rozdělena na dvě teoreticky a tři prakticky zaměřené kapitoly. První kapitola se věnuje síťovým tokům, jejich monitorování a analýze. Druhá kapitola se zabývá vztahovou analýzou. Popisuje moderní grafové databáze, databázi Dgraph a sadu nástrojů GRANEF.

Třetí kapitola popisuje implementaci dvou modulů. Extrakční modul převádí zachycené síťové toky na obousměrné. Transformační modul konvertuje síťové toky do formátu vhodného pro grafovou databázi Dgraph. Čtvrtá kapitola je věnovaná popisu datasetu na kterém se testovaly nové analytické dotazy a obsahuje popis analytického postupu při nahlášení bezpečnostní hrozby. Poslední kapitola shrnuje poznatky z jednotlivých modulů, testuje jejich výkonnost a navrhuje vhodné optimalizace.

Výsledkem práce jsou implementované moduly, které rozšiřují nástroj GRANEF o zpracování a analýzu síťových toků. To umožní analytikům z CSIRT využít sadu nástrojů GRANEF nejen pro hloubkovou analýzu paketů, ale i analýzu síťových toků.

1 Monitorování síťového provozu

Monitorování síťového provozu je jedním z hlavních přístupů pro zajištění funkčnosti a zabezpečení sítě, jejích služeb a uživatelů. V rámci monitorování probíhá analýza síťového provozu, která umožňuje detekovat útoky a předcházet jim, sledovat vytíženost sítě a řadu dalších funkcí nezbytných pro správu sítě. Proces monitorování spočívá ve sběru a záznamu dat z důležitých síťových zařízení (obvykle se jedná o zařízení, které je umístěno tak, že nám dává co největší možný přehled o tom, co se v síti děje). Využívají se dva základní mechanismy: aktivní a pasivní [1].

Aktivní monitorování funguje na základě dotazování a komunikaci s monitorovanými síťovými prvky [1]. Dochází k vkládání testovacího provozu do sítě a na základě reakce se odvozuje stav sítě, nebo jednotlivých hostů. Poskytuje tím přehled o výkonu služeb v reálném čase. Výhodou jsou detailní informace o monitorovaných síťových prvcích. Nevýhodou tohoto přístupu je možné zahlcení a zpomalení sítě způsobené přidáním testovacím provozem. Aktivní monitorování se používá například pro zjišťování topologie sítě, kontrolu dostupnosti a měření kvality služeb. Je implementováno v nástrojích jako *Nmap* [3], *Ping*, *Traceroute*.

Pasivní přístupy sledují stávající síťový provoz, který zachytí při průchodu pozorovacím bodem v síti. Používají se například pro identifikaci síťových útoků, nebo hledání anomálií v síťovém provozu. Síťový provoz je možné sledovat jednou z následujících tří přístupových metod: *in-line*, *zrcadlení*, *build-in* [1]. Metoda *in-line* sleduje síťový provoz pomocí snímacího zařízení (např. pomocí TAP = test access point, test přístupového bodu) umístěného na monitorovanou linku mezi dva prvky sítě. Snímací zařízení je navrženo, aby poskytovalo trvalý přístupový port pro pasivní sledování a duplikuje síťový provoz [4]. *Zrcadlení* využívá funkci označovanou jako *zrcadlení portů* nebo *SPAN*, která je přítomná ve většině zařízení pro předávání paketů [5]. Dochází při ní k zrcadlení paketů z jednoho nebo více portů na jiný port. Na rozdíl od *in-line* metody zde dochází k úpravě monitorovaného provozu (například přidává artefakty týkající se rozdílu časování, a mění pořadí paketů). I přes její nedostatky je tato metoda často využívána kvůli své široké dostupnosti a nízké ceně. Poslední me-

toda *build-in* je přímo zabudovaná ve směrovacích zařízeních. Software pro monitorování sítě je obvykle pouze doplňkovou funkcí. V případě vysoké zátěže provozu zařízení přiděluje prostředky ke splnění své hlavní funkce a prostředky pro monitorování sítě jsou sníženy (což je nežádoucí například v případě DDos útoku). Tudíž tato metoda není vhodná pro vysokorychlostní sítě.

Při monitorování sítě existují různé úrovně viditelnosti: monitorování celých paketů nebo jen jejich částí. Záchytu a analýze celých paketů se věnuje hloubková inspekce paketů (DPI). Díky tomu má přístup k velkému objemu informací extrahovaných ze síťového provozu [6]. Taková vysoká viditelnost má náročné požadavky na výpočetní zdroje. Je vhodná spíše pro analýzu cíleného síťového provozu než pro monitorování celé sítě. Naopak monitorování síťových toků analyzuje pouze informace z hlaviček paketů, což umožňuje vysokou propustnost a relativně nízké požadavky na zdroje [1]. Vzhledem k efektivní agregaci dokáže zpracovat síťový provoz generovaný tisíci uživateli. Umožňuje například sledovat vytíženost sítě, monitorovat uživatele, provoz specifických nástrojů a online služeb, usnadňuje detekci anomálií a zvyšuje síťovou bezpečnost. Tato práce je zaměřena na analýzu síťového provozu rozsáhlejších sítí, kde je nejvhodnější využívat monitorování síťových toků.

1.1 Síťové toky

Síťový tok je definován jako sada IP paketů nebo rámců procházejících pozorovacím bodem v síti během určitého časového intervalu [7]. Všechny pakety patřící k určitému síťovému toku mají sadu společných vlastností. Každá vlastnost je definována jako výsledek funkce, která je aplikovaná na hodnoty:

- jednoho nebo více polí hlavičky paketu (např. cílová IP adresa), přenosu (např. číslo cílového portu), nebo aplikace (např. pole RTP (Real-time Transport Protocol) [RFC 3550]);
- jedna nebo více charakteristik samotného paketu (např. počet návěstí MPLS (Multiprotokolové přepojování podle návěstí) atd.);
- jedno nebo více polí odvozených ze zpracování paketů (např. IP adresa dalšího skoku, výstupní rozhraní atd.);

Paket náleží k síťovému toku, pokud zcela splňuje všechny definované vlastnosti síťového toku.

Mezi základní vlastnosti síťových toků patří například zdrojová a cílová IP adresa, zdrojový a cílový port, protokol a další klíčové vlastnosti paketů [8]. Vlastnosti, které jsou společné pro všechny agregované pakety jsou klíčem síťového toku. Pro specifické potřeby analýzy lze vlastnosti rozšířit o další informace (např. časová razítka, kdy komunikace probíhala). Jednoduchý příklad síťového toku, znázorňující komunikaci přes protokol HTTPS (Hypertext Transfer Protocol Secure), je na obrázku 1.1.

Flow start	Duration	Source IP Adr:Port	Destination IP Adr:Port	Protocol	Flags	Packets	Bytes
15:06:44.278	0.211	10.1.4.47:55878	40.112.75.175:443	TCP	AP.SF	20	11321
15:06:44.365	0.171	40.112.75.175:443	10.1.4.47:55878	TCP	AP.SF	19	12629

Obrázek 1.1: Záznamy jednosměrných síťových toků reprezentující HTTPS spojení

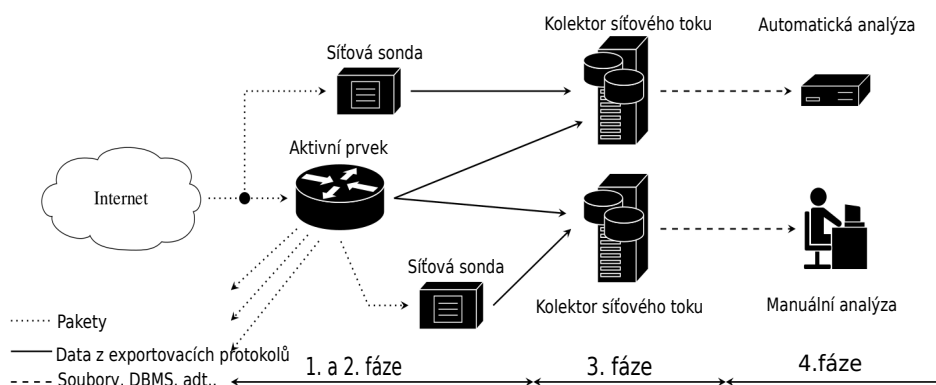
Existují aplikační síťové toky, které jsou rozšířené o informace extrahované z aplikační vrstvy paketu. Proces, kdy se extrahují aplikační data, může ovlivnit vytváření samotného síťového toku. Například síťový tok reprezentující plynulé HTTP spojení se může rozpadnout na několik síťových toků na základě jednotlivých HTTP požadavků [9].

Záznamy síťového toku mohou být jednosměrné, nebo obousměrné. Jednosměrný síťový tok obsahuje pro jedno připojení mezi hostiteli sítě dva vygenerované záznamy. Tento způsob usnadňuje práci exportérům, které nemusí kombinovat informace z obou směrů. Avšak pro některé analýzy jsou jednodušší na použití obousměrné síťové toky, protože obsahují oba směry spojení v jednom záznamu. Při vzniku obousměrných síťových toků se oba směry spojí do jednoho a hodnoty společných atributů se až na výjimky agregují (výjimkami například jsou IP adresy a porty). Obousměrné síťové toky snižují nároky na paměť, protože sdílené informace jsou uloženy pouze jednou.

Využívají se v mnoha odvětvích: pro uchovávání dat (podle české vyhlášky č. 357/2012 Sb. zákona o elektronických komunikacích), profilování, účtování, sledování vytíženosti sítě, z bezpečnostního hlediska k detekci útoků a analýzu incidentů [1].

1.2 Architektura monitorování síťových toků

Obecná architektura používaná k monitorování síťových toků se skládá ze čtyř fází vyobrazených na obrázku 1.2. V první fázi jsou pakety pozorovány a zachyceny v jednom, nebo více pozorovacích bodech [1]. Poté jsou v rámci druhé fáze zpracovány do síťových toků a exportovány kolektorům. Při třetí fázi dochází ke sběru síťových toků a jejich předzpracování na následující analýzu. Poslední fází je samotná analýza, která může být prováděna ručně, nebo automatizovaně pomocí analytického programu. V reálném použití nemusí být toto obecné schéma architektury dodrženo a první s druhou fází mohou být spojeny v jednom zařízení. Potom toto zařízení nazýváme exportér síťového toku. Pokud slouží pouze k exportování síťových toků a nemá žádnou další funkci, označujeme jej jako síťovou sondu.



Obrázek 1.2: Architektura monitorování síťových toků [1]

1.2.1 Pozorování paketů

Pozorování paketů začíná pozorovacím bodem. To je zařízení v síti, které má schopnost duplikovat provoz sledované sítě a odesílat pakety. Obvykle je umístěno na hranici lokální sítě (jedná se například o výstupní bránu), aby poskytovalo co největší možný přehled o síťovém provozu [1]. Zařízení je připojeno pomocí již zmíněných přístupových metod. Proces pozorování paketů se skládá ze dvou hlavních

částí – záchyt paketů a přidávání časových razítek. Tyto dva procesy se provádí pro každý pozorovaný paket.

Při záchytu paketů je nejprve paket zachycen z linky kartou síťového rozhraní (Network Interface Card) [1]. Provede se několik kontrol (například kontrola chyb, kontrolní součty) a paket se přesune na zásobník, odkud je poskytován do knihoven a aplikačních rozhraní určených pro záchyt síťového provozu. Pro Windows je dostupná *npcap* [10] a na Linux *libpcap* [11] nebo *libtrace* [12]. Tyto knihovny poskytují funkce pro záchyt i filtrování paketů a slouží pro proces měření a vytváření záznamů síťových toků.

Druhým krokem je přidání časových razítek. Během této fáze je každému paketu přiděleno přesné časové razítko pozorování, které je nezbytné pro další zpracování a analýzu. Může být prováděno za pomoci hardwaru i softwaru.

Volitelné procesy, které se používají ke snížení pracovní zátěže, jsou zkrácení paketů, vzorkování a filtrování. Dávají si za cíl zmenšit objem dat, se kterými se bude pracovat v dalších fázích. Během procesu krácení paketů jsou ponechány pouze bajty potřebné pro zpracování síťových toků, ostatní jsou zkráceny. Základní síťové toky se vyvážejí pouze z hlaviček paketů, tudíž se celá datová část paketu zahodí, což významně snižuje objem dat. Při vzorkování se získává reprezentativní podmnožina monitorovaného provozu, což může mít za následek ztrátu informací a je potřeba to v pozdější analýze zohlednit. Filtrování paketů deterministicky „odděluje všechny pakety, které mají určitou vlastnost, od těch, které ji nemají“ [13]. Při správně zvoleném filtru, se zahodí jen ta data, která nebudou během analýzy potřeba.

1.2.2 Měření a export síťových toků

Exportéry jsou klíčovou složkou v monitorování síťových toků. Jejich funkce přímo ovlivňuje, kolik síťového provozu lze zpracovat, jak se budou generovat síťové toky a jak se předají kolektorům. Základní součástí exportéru je proces měření, který zajišťuje analýzu příchozích paketů a jejich agregaci do síťových toků. V případě méně výkonných exportérů dochází k vzorkování paketů a filtrování síťových toků, které se rozlišuje na filtrování podle společných vlastností a filtrování založené na výstupu hashovací funkce [14]. Filtrování podle společných vlastností je vhodné, když jsou zajímavé pouze ty

záznamy toku, které se týkají konkrétního hosta, podsítě, nebo jen určitých portů apod. U filtrování založeném na hashovací funkci se vyberou pouze záznamy, kde hodnota klíče leží v předem definovaném rozsahu hodnot. Toto filtrování lze využít při výběru skupin záznamu toků z různých pozorovacích bodů.

Druhou částí exportéru je samotný proces exportu síťových toků do kolektoru. Při procesu exportu jsou odeslány záznamy o síťovém toku například pomocí exportovacího protokolu NetFlow v.5 [15], v.9 [16] nebo IPFIX [7]. Tyto protokoly rozdělí záznam síťového toku a jeho šablonu, aby se snížilo množství přenášených dat. Například IPFIX má záhlaví a obsah zprávy, kde záhlaví má pevnou strukturu a zahrnuje metainformace o zprávě – číslo verze protokolu, délka zprávy, čas exportu, pořadové číslo a ID pozorovací domény [7]. Obsah zprávy se skládá ze sady záznamů síťových toků, ve kterých jsou specifikovány informace o síťových tocích, šablony obsahující další informace pro kolektory.

Exportéry se dělí na hardwarové a softwarové. Hardwarové exportéry mají obvykle vyšší propustnost, zatímco softwarové jsou flexibilnější, pokud jde o funkčnost [1]. Softwarové exportéry jsou také méně nákladné. Dalším rozdělením může být na open-source a placené exportéry. Mezi open-source se řadí například *nProbe* [17], *fProbe* [18] nebo *YAF* [19]. Nejznámějšími placenými nástroji jsou *Cisco NGA* [20] a *Flowmon Probe* [21].

1.2.3 Sběr dat

Sběr dat je prováděn pomocí jednoho nebo více kolektorů [1]. Kolektory přijímají, ukládají a předzpracovávají záznamy síťových toků z jednoho nebo více exportérů. Kromě ukládání poskytují i další funkce pro zpracování a analýzu záznamů síťových toků. Klíčovou funkcí je dotazovací systém, který umožňuje uložená data vypisovat podle zvolených filtrů a parametrů. Měl by obsahovat i agregační a souhrnné funkce pro získání přehledu o monitorované síti s grafickým rozhraním. Další funkcí je možnost anonymizace přijatých síťových toků, která je nutná pro sdílení dat mimo organizaci a pro vědecké účely, aby se zachovalo soukromí uživatelů. Velmi užitečnou funkcí je i možnost transformace záznamu síťových toků do jiných souborových formátů, jakými jsou například JSON a CSV.

Záznamy síťových toků mohou být uloženy v různých formátech, které ovlivňují funkčnost a výkon kolektorů, protože definují jak a při jaké rychlosti lze data číst a zapisovat. Dělí se na dva základní typy: volatilní a nevolatilní. Volatilní uložení se kvůli své rychlosti využívá pro zpracování dat. Nevolatilní uložení je vhodnější pro ukládání dat po delší dobu. Kvůli rozdílu v rychlosti čtení a zápisu těchto dvou typů úložišť se data komprimují. Rozlišujeme několik různých typů nevolatilních úložišť: prostý soubor (například binární nebo textový), relační databáze (například MySQL) a sloupcově orientované databáze (například Fastbit) [1]. V poslední době se využívají i frameworky pro zpracování velkého množství dat (např. Hadoop).

1.2.4 Analýza dat

Při analýze dat se spojí výsledky všech předchozích fází. Podle provedení se dělí na manuální a automatickou. Manuální analýza se zaměřuje na dlouhodobé ukládání síťových toků, které jsou poté zpracovány analytikem pomocí specializovaných nástrojů, umožňujících získávat informace o událostech nebo různých chybách zachycených v síťovém provozu. Tyto nástroje obsahují dotazovací funkce a grafické rozhraní.

Automatická analýza, je doplňkem manuální, na základě průběžného zkoumání nově uložených záznamů síťových toků detekuje anomálie a narušení síťového provozu. Nad těmito záznamy se provádějí různé statistické analýzy, aby se zjistily odchylky od normálního chování, prohledaly se známé vzory síťových útoků nebo se použily pokročilejší analytické funkce, založené na strojovém učení. Výsledkem je upozornění popisující zjištěnou anomálii nebo útok. Upozornění se posílá analytikovi, který incident dále prověří. Při práci s rozsáhlejšími daty si automatická analýza klade vysoké nároky na čas, výpočetní zdroje a úložnou kapacitu.

Analýza síťového provozu se dále rozděluje na tři oblasti: analýzu síťových toků, detekci síťových útoků a monitorování výkonu. Analýza síťových toků je nejzákladnější a nejpoužívanější [1]. Vzhledem k tomu, že se zařízení pro export síťových toků umísťují na strategických místech v síti, lze pozorovat síťový provoz od velkého počtu hostitelů, který se analyzuje pomocí následujících funkcí:

- Procházení a filtrování síťových toků.
- Statistický přehled – nejčastěji se zaměřuje na hosty, autonomní systémy nebo služby, které generují nejvíce síťového provozu.

Detekci síťových útoků můžeme rozlišit na dva základní typy. Prvním typ analyzuje komunikaci mezi hostiteli. Podezřelé hostitele identifikuje pomocí blacklistů (černých listin), což jsou seznamy IP adres, které byly identifikovány jako zdroje škodlivých aktivit (například posílaly SPAM zprávy nebo byly součástí botnetu). Druhý typ umožňuje modelovat síťové útoky z hlediska chování v síti. Dokáže detekovat útoky, které ovlivňují metriky jako objem provozu v podobě paketů a bajtů, podezřelá čísla portů a další [22]. Jedná se o následující útoky a malwary: DDoS útoky, skenování sítě, botnetová komunikace a šíření červů.

Monitorování výkonu služeb je potřebné pro zajištění kvality služeb. Cílem je sledovat stav aplikací běžících v síti a vytíženost sítě. Ověřují dodržení dohody o úrovni kvality služeb a jaký má dopad na koncového uživatele.

2 Analýza síťového provozu pomocí grafové databáze

Analýza síťového provozu je založena na dvou přístupech: interaktivní analýze surových dat a statistická analýze [23]. Interaktivní analýza využívá filtrování, agregace a extrahování smysluplných informací ze surových dat. Nástrojem, který tuto analýzu podporuje je např. Wireshark [24]. Nevýhodou tohoto přístupu jsou vysoké nároky na výpočetní prostředky, omezená vizualizace a automatizace analytických dotazů. Statistický přístup vizualizuje významné paketové prvky ve formě různých tabulek a přehledových vizualizací. Výhodou je zpracování velkého množství dat a poskytování přehledů pomocí interaktivních vizualizací. Při zpracování dat dochází k agregaci a analytik tak ztrácí přístup k surovým datům.

Výzkum vedený na CSIRT-MU si klade za cíl spojit výhody obou přístupů a umožnit analytikovi zkoumat zachycená data pomocí interaktivní vizualizace. Sada nástrojů (dále toolkit) GRANEF slouží k dosažení tohoto cíle a je zaměřena na analýzu síťového provozu na základě asociací. Toolkit GRANEF popisuje článek [23], a v této kapitole z něj budu vycházet. Analýze síťového provozu na základě asociací se věnují grafové databáze, které budou popsány v následující podsekci.

2.1 Grafové databáze

Grafové databáze jsou typem NoSQL (jiné než SQL, jsou založené na jiném schématu než, je relační) databází, jejichž struktura ukládání a metody dotazování jsou založeny na teorii grafů [25]. Využívají tři základní prvky: uzly, vztahy a vlastnosti. Uzly reprezentují entity, nebo objekty a vzájemně jsou propojeny vztahy. Uzly i vztahy mohou mít jednu, nebo více vlastností.

Znázornění závislostí pomocí grafů je univerzální a umožňuje modelovat všechny druhy scénářů, od konstrukce vesmírné rakety po komunikaci uživatelů na sociálních sítích. Grafický výpočetní modul umožňuje spouštět globální výpočetní algoritmy proti velkému množství dat. Je přizpůsoben k hledání a identifikaci shluků v datech, nebo

aby dokázal odpovídat na dotazy typu: „Kolik vztahů má v průměru každý na sociální síti?“

Populárními grafovými databázemi jsou Neo4j, ArangoDB, Dgraph a OrientDB [26]. První databází, do které se ukládali data při analýze síťového provozu na základě asociací, byla Neo4j [27].

2.2 Dgraph

Dgraph je horizontálně škálovatelná a distribuovaná GraphQL (Graph Query Language – dotazovací jazyk nad grafy) databáze s grafickým rozhraním [28]. Jako trvalé uložení využívá *Badger* [29], což je integrovaná a rychlá key-value databáze. Je nativní databází od základu navrženou pro GraphQL.

Cluster Dgraphu se skládá z různých uzlů (Zero, Alpha a Ratel) a každý uzel slouží jinému účelu [28]. Dgraph Zero přiřazuje servery do skupin a vyvažuje data mezi jednotlivými skupinami serverů. Dgraph Alpha uchovává predikáty a indexy. Predikáty jsou vlastnosti přidružené k uzlu, nebo vztah mezi dvěma uzly. Indexy jsou připojené k predikátům, aby bylo možné filtrovat pomocí příslušných funkcí. Pro lepší škálování databáze je možné mít spuštěných více uzlů Alpha. V tomto případě jsou predikáty zreplikované napříč celým clusterem a dotazy lze efektivně spouštět z libovolného uzlu. Ratel slouží jako UI ke spouštění dotazů, mutací a pozměňování schémat. Aby mohl být Dgraph spuštěn je nutné mít alespoň jeden Dgraph Zero a jeden Dgraph Alpha.

Pro plně funkční GraphQL API je nutné mít vytvořené schéma. Schéma reprezentuje databázi a je z něj vidět jaké operace může server provádět. Jsou psané v SDL (Schema Definition Language) [30] a definují typy objektů, pole představující data a hrany mezi jednotlivými objekty. Například schéma znázorňující databázi knih může vypadat takto:

```
genre: [uid] @reverse .
published_date: dateTime @index(year) .
rating: int .
name: string @index(exact, term) @lang .
author: [uid] .
type: [uid] .
```

```
# Types definition
type Author {
    name
    author.book
}

type Book {
    name
    published_date
    genre
    rating
}

type Genre {
    name
}
```

Dotazovacím jazykem je Dgraph Query Language (DQL) postavený na GraphQL. GraphQL je dotazovací jazyk nad API [31]. Obvykle představuje nějaký server, na který se posílají dotazy. Není spojený s konkrétní databází, nebo programovacím jazykem. Je silně typovaný, tzn. musí mít nadefinovaný typ pro všechna data, na která se dotazuje. Uživateli se tedy nemůže stát, že by dostal data, o která si neřekl. Všechny příchozí dotazy, nebo mutace napsané v GraphQL jsou za běhu přepisovány do DQL. DQL rozšiřuje jazyk GraphQL, umožňuje interagovat přímo s databází, podporuje hluboké procházení, složité dotazy a proměnné. Při vyhodnocení dotazu se najdou všechny uzly, které splňují kritéria, porovnají se s grafem a ty shodné se vykreslí.

Každý dotaz má v hlavičce název, který se pak zpropaguje do výsledku dotazu. Pokud bychom chtěli získat informace z uzlů, které jsou spolu propojeny hranou, použijeme složené závorky. Například dotaz na získání autora knihy Harry Potter a kámen mudrců bude vypadat takto:

```
{
  getAuthor(func: eq(name@en, "Harry Potter and
    the Philosopher's Stone")) {
    name@en
    published_date
    rating
    gendre {
      name
    }
  }
}
```

Základními kameny analýz síťového provozu jsou agregace a filtrování dat. Při použití filtrování si kořen dotazu najde počáteční sadu uzlů a vrátí hodnoty, potom pokračuje po hranách k dalším uzlům [28]. Uzly lze filtrovat buď u kořene uzlu, nebo na libovolné hraně.

```
{
  rowling(func: eq(name@en, "Joanne Rowling")) {
    name@en
    published_date
    author.book @filter(le(published_date,
      "2000")) {
      name@en
      published_date
    }
  }
}
```

Agregace využívají čtyři základní funkce: *min/max* pro nalezení minima/maxima, *sum* sečte všechny namapované hodnoty a *avg* vypočítá průměr z nalezených hodnot. Funkce *min* a *max* mohou být aplikované na datové typy: *int*, *float*, *string*, *dateTime*, *default*. *Sum* a *avg* pouze na *int*, *float*. Následující příklad vrátí datum poslední vydané knihy od autorky Joanne Rowling.

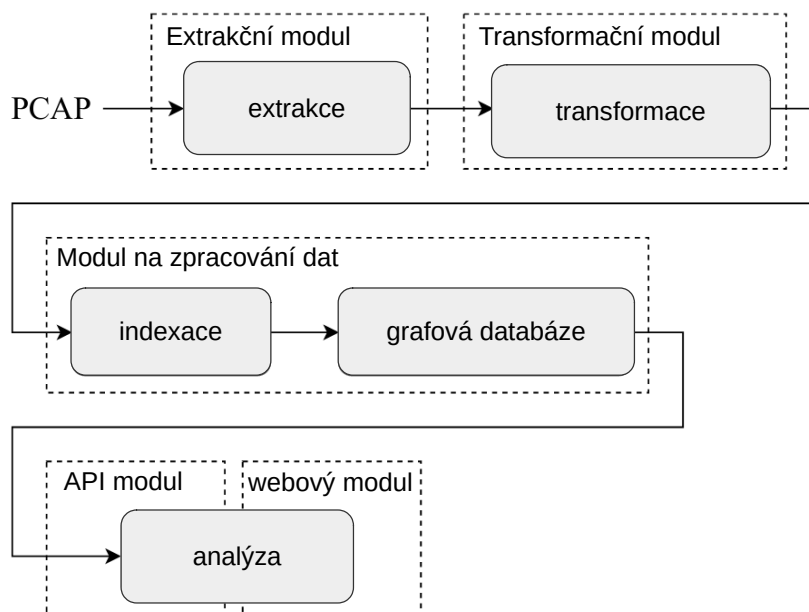
```
{
  author(func: allofterms(name@en, "Joanne
    Rowling")) {
    author.book {
      name@en
      x as published_date
    }
    max(val(x))
  }
}
```

2.3 Sada nástrojů GRANEF

Toolkit GRANEF využívá pro extrakci dat ze síťového provozu platformu Zeek [32]. Získané logy převede do formátu podporovaného grafovou databází Dgraph a transformovaná data do ní uloží. Uložená data jsou analytikovi prezentována přes API, nebo prostřednictvím webového rozhraní, které poskytuje vrstvu abstrakce nad databázovým dotazovacím jazykem. Umožňuje uživateli efektivně vyhledávat data, vizualizovat výsledky ve formě diagramu a provádět další analýzy. Primárním využitím toolkitu je hloubková inspekce paketů.

Toolkit je založen na grafové databázi Dgraph, která ukládá a analyzuje data poskytovaná nástrojem pro monitorování sítě Zeek. Umožňuje procházet, filtrovat a agregovat zachycená data a výsledky vizualizovat v grafu vztahů, který poskytuje širší kontext o analyzovaných datech. Dále obsahuje nástroje pro předběžné zpracování a analýzu dat. Tyto nástroje jsou implementovány jako samostatné moduly a každý je izolován v samostatném Dockerovém kontejneru [33]. Schéma jednotlivých modulů je vidět na obrázku 2.1. Rozdělení na samostatné moduly umožňuje snadné přidávání nových modulů (při zachování kompatibility se sousedními moduly), snadnou úpravu stávajícího modulu a ukládání průběžných mezivýsledků, které je pak možné vložit do jiných analytických nástrojů.

Modul odpovídající za extrakci dat obsahuje platformu Zeek, která zachycuje síťový provoz a extrahuje informace z hlaviček paketů a aplikačních vrstev. Výhodou nástroje Zeek je velké množství nastavení zpracování dat a rozšíření o nové metody extrakce. To umožňuje rozší-

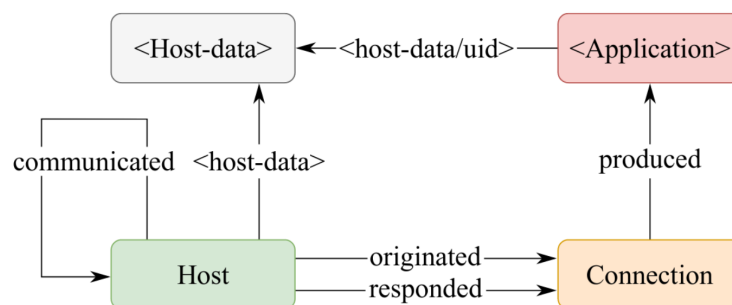


Obrázek 2.1: Schéma GRANEF modulů [23]

řit transformační skripty o podrobnější analýzu a extrahovat více informací o komunikaci, které nejsou k dispozici ve standardní konfiguraci. Modularita nástrojů GRANEF dovoluje připravit vícero kontejnerů s různými konfiguracemi a tím celkově rozšířit rozsah extrahovaných dat, které mohou být dále zpracovány transformačním modulem.

Transformační modul převádí vstupní soubory, obsahující logy získané z extrakčního modulu, do formátu RDF [2], který je akceptován databází Dgraph. Tuto konverzi dat provádí vlastní skript, který zpracovává vybrané protokoly záznam po záznamu. Každý protokol má předem známou sadu atributů, které bude obsahovat. Tato informace nám dává možnost si zpracování jednotlivých protokolů upravit podle potřeb analýzy, pro kterou se data připravují.

Databázové schéma je navrženo, tak aby odráželo lidské vnímání počítačové sítě a tím zjednodušovalo analýzu. Každý uzel má přiřazený typ a podle schématu jsou uzly mezi sebou propojeny hranami. Na obrázku 2.2 můžeme vidět čtyři uzly, které jsou mezi sebou propojeny vztahy. Každý uzel obsahuje atributy, které jej specifikují. Napří-



Obrázek 2.2: Zjednodušené schéma znázorňující uzly a jejich hrany [23]

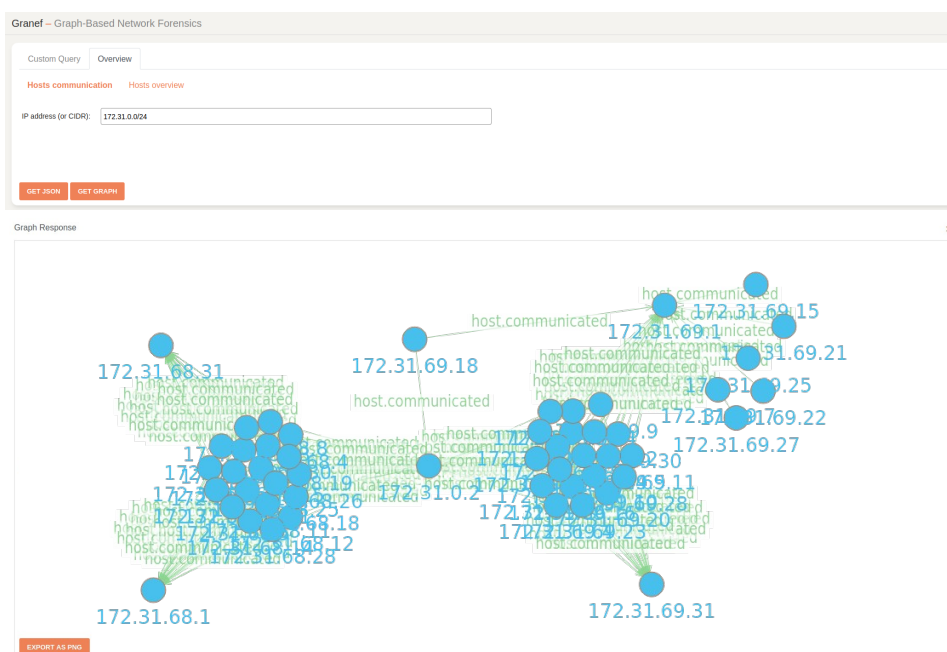
klad uzel hosta obsahuje IP adresu, uzel spojení informace o síťovém připojení, jeho délce, množství přenesených paketů a další. Všechny hrany jsou směrové, ale umožňují i zpětné dotazování. Hrany jdoucí z hosta ukazují, že host může komunikovat s jiným hostem, vytvářet a odpovídat na spojení a ví jaká data přes spojení poslal. Univerzálnost definice schématu dovoluje transformovat i jiné datové typy související s analýzou než ty podporované nástrojem Zeek. Zajímavé atributy mohou být označeny externím identifikátorem již ve fázi jejich definice, to později usnadní filtrování a agregace nad těmito daty.

Jádrem modulu, který se stará o zpracování dat, je cluster Dgraphu skládající se ze dvou výpočetních uzlů. Dgraph Zero slouží jako hlavní komponenta zodpovědná za databázi a analýzu. Zpracování dat provádějí uzly Dgraph Alpha obsahující indexovaná data. Ke zpracování dat je nutné mít spuštěný alespoň jeden Dgraph Zero uzel a jeden Dgraph Alpha uzel. Modul se skládá z indexování a databáze grafů, které komunikují přímo s Dgraphem. Během indexování se RDF trojice indexují a jsou ukládány do interní databázové struktury. K načítání dat do nového clusteru se využívá Dgraph Bulk Loader, který pracuje na konceptu MapReduce. Ten vhodně využívá dostupné prostředky a lze pomocí něj určit kolik uzlů Alpha bude potřebných. Díky těmto vlastnostem mohou být nad velkými objemy dat, prováděny rychlé analýzy.

Analytický toolkit GRANEF se skládá ze dvou modulů: aplikačního rozhraní (API modul) a webového rozhraní (webový modul).

2. ANALÝZA SÍŤOVÉHO PROVOZU POMOCÍ GRAFOVÉ DATABÁZE

Toto rozdělení podporuje větší univerzálnost celého řešení, protože je možné k API připojit další systémy bez nutnosti používat webové rozhraní. V rozhraní API jsou implementovány sady analytických dotazů, které jsou založeny na zkušenostech analytiků z týmu CSIRT-MU, kteří nástroj vyvíjí. Webové uživatelské rozhraní je uživatelsky přívětivější. Využívá API a v něm implementované sady dotazů. Výsledky dotazu zobrazuje v grafové vizualizaci, která umožňuje zobrazovat i velké grafy vztahů jednoduše a přehledně. Vizualizace jsou interaktivní a dovolují analytikovi zobrazit atributy zvolených uzlů, nad kterými může provádět další analytické dotazy. Použití již vytvořeného dotazu a jeho grafový výstup můžeme vidět na obrázku 2.3.



Obrázek 2.3: Webové uživatelské rozhraní

3 Extrakce a transformace síťového provozu

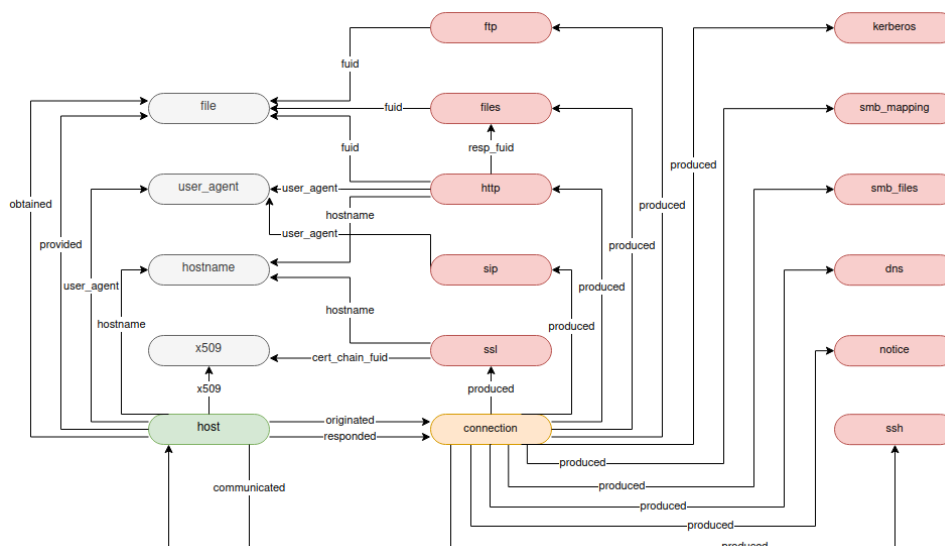
Toolkit GRANEF poskytuje plnou podporu pro hloubkovou analýzu paketů, aby ho bylo možné využít i pro analýzu síťových toků, je nutné ho rozšířit o extrakci a transformaci síťových toků, upravit databázové schéma a zvětšit sadu analytických dotazů. V této kapitole budou představeny dva nové moduly: extrakční modul a transformační modul. Oba tyto moduly už byly představeny v předchozí kapitole, ale ty se týkaly zpracování paketů. Nové moduly se zaměřují na zpracování síťových toků.

Proces zpracování síťových toků bude následující. Nejprve se síťový provoz zachycený kolektorem převede pomocí nástroje `nfdump` na oboustranné síťové toky. Následuje transformace získaných dat na soubory RDF a schema. V posledním kroku se tyto soubory nahrají do Dgraphu.

3.1 Extrakce dat

Za účelem přidání možnosti využít `nfdump` pro extrakci dat, byl do toolkitu GRANEF přidán nový extrakční modul (`Extraction Nfdump`). Jako vstup bere `nfcapd` soubory, které obsahují zachycený síťový provoz ve formě síťových toků. Ty jsou následně pomocí nástroje `nfdump` převedeny na obousměrné síťové toky a výsledný soubor je ve formátu `csv`. Vše se provede pomocí příkazu `nfdump -R <input_file/dir> -b -o csv`.

Extrakční modul dostane jako vstup cestu k adresáři, nebo souboru, který obsahuje zachycený síťový provoz. Aby bylo možné načíst všechny `nfcapd` soubory ze vstupního adresáře je nutné použít přepínač `-R`. Pokud dostane cestu k `nfcapd` souboru, tak načte daný soubor a nic dalšího rekurzivně nenačítá. Protože stávající databázové schéma toolkitu GRANEF není navrženo pro jednosměrné síťové toky, je nutný převod na obousměrné, čehož lze dosáhnout pomocí přepínače `-b`. Takový převod není úplně spolehlivý, jelikož k němu dochází až po sběru dat. V obousměrných síťových tocích je celá komunikace mezi dvěma hosty v jednom toku, což značně zjednoduší schéma, protože není nutné sledovat oba směry komunikace zvlášť. Další výhodou je zmenšení celkového objemu dat. Posledním použitým přepínačem



Obrázek 3.1: Databázové schéma pro analýzu paketů [34]

je `-o`, který definuje typ výstupního souboru. Jako formát výstupního souboru byl zvolený csv. Oproti formátu JSON zabírá na disku méně místa a zpracování dat je rychlejší, protože data lze rozdělit přes odělovací znak a není nutné mít speciální interpret pro syntaxi jako u formátu JSON.

3.2 Databázové schéma

Databázové schéma pro analýzu síťových toků vychází z již používaného schématu pro hloubkovou analýzu paketů (viz obrázek 3.1), kterou toolkit GRANEF používá. Podstatou je mít uzel Host propojený s uzlem Connection skrz hrany `originated` a `responded`. Uzel Connection je spojen s typem použitého protokolu, pomocí hrany `produced`. Aplikační uzly mohou a nemusí mít aplikační data. Zjednodušené schéma je vidět na obrázku 2.2.

Transformační modul pro převod síťových toků do RDF trojic, používá zjednodušené schéma, protože síťové toky neobsahují tolik aplikačních dat jako pakety. I obsah jednotlivých uzlů se liší, nejen že různé implementace kolektorů obsahují jiné zkratky v hlavičkách, ale

můžou se lišit i v rozsahu zachycených dat. Proto se obsah jednotlivých uzlů automaticky generuje ze vstupních dat za běhu programu. Celý proces vytváření schématu je implementovaný v rámci transformačního skriptu, nelze ho tedy spustit zvlášť.

Vytváření schématu předchází skript, který z manuálové dokumentace nástroje nfdump vezme aplikaci, nebo protokol a jeho atributy ve formě zkratk a jejich definic. Vše uloží do csv souboru ve formátu *zkratka aplikace/protokol definice_zkratky*. Protože se některé aplikace/protokoly a definice zkratk skládají z více slov, jsou všechny převedeny do snake_case formátu. Tento skript dokáže vytáhnout všechny základní zkratky, spojit je s jejich definicí a zařadit pod příslušnou aplikaci. Pokud není definovaná aplikace, zařadí se zkratky pod aplikaci connection, protože obsahují základní informace o komunikaci mezi dvěma hosty. Co se myslí pod pojmem základních zkratk lze vidět na následující ukázce.

```
HTTP formats
%hhost      HTTP Hostname
%hhost1     HTTP Hostname (1st level domain)
%hhost2     HTTP Hostname (1st & 2nd level domain)
%hhost3     HTTP Hostname (1st, 2nd & 3rd level
            domain)
%hmethod    HTTP Method Type
%hrcode     HTTP Response Code
%hurl       HTTP URL
%hurl1      HTTP URL (first directory)
%hos        HTTP OS
%hosmaj     HTTP OS Major Version
%hosmin     HTTP OS Minor Version
%hosbld     HTTP OS Build Version
%happ       HTTP Application
%happmaj    HTTP Application Major Version
%happmin    HTTP Application Minor Version
%happbld    HTTP Application Build Version
```

Při takto definovaných zkratkách bude výstupní csv soubor vypadat následovně.

```
hhost HTTP_formats HTTP_Hostname
hhost1 HTTP_formats
      HTTP_Hostname_1st_level_domain
hhost2 HTTP_formats
      HTTP_Hostname_1st_AND_2nd_level_domain
hhost3 HTTP_formats
      HTTP_Hostname_1st_2nd_AND_3rd_level_domain
hmethod HTTP_formats HTTP_Method_Type
hrcode HTTP_formats HTTP_Response_Code
hurl HTTP_formats HTTP_URL
hurl1 HTTP_formats HTTP_URL_first_directory
hos HTTP_formats HTTP_OS
hosmaj HTTP_formats HTTP_OS_Major_Version
hosmin HTTP_formats HTTP_OS_Minor_Version
hosbld HTTP_formats HTTP_OS_Build_Version
happ HTTP_formats HTTP_Application
happmaj HTTP_formats
      HTTP_Application_Major_Version
happmin HTTP_formats
      HTTP_Application_Minor_Version
happbld HTTP_formats
      HTTP_Application_Build_Version
```

Při vytváření schématu nejprve dojde k namapování zkratk z hlavičky vstupního souboru obsahujícího síťový provoz, na jejich definice, které jsou obsažené ve výstupním csv souboru skriptu popsaného výše. Díky tomu budou mít atributy srozumitelnější názvy. V případě, že se některá zkratka v csv souboru nevyskytuje, je uživatel na tuto skutečnost upozorněn pomocí logování a daná zkratka zůstane jako název atributu. Tento problém nastává v případě specifitějšího zachytu síťového provozu, kde se objevují zkratky obsažené v rámci textu, nebo nejsou v manuálové dokumentaci obsaženy vůbec. Například při použití jiného nástroje pro zachyt síťového provozu. Následně se při zpracování jednotlivých řádků dat kontroluje typ, ten se ukládá do pomocného slovníku, ze kterého se na konci běhu programu vygenerují typy pro Draph schema. Potom co jsou všechny řádky úspěšně převe-

deny do RDF trojic, se spustí generování schématu. Všechny sloupce z hlavičky, které mají přiřazený datový typ (tzn. byly ve vstupních datech zastoupeny alespoň jednou) se zapíší do schématu a zařadí se pod příslušný typ protokolu. Ukládání typů se provádí pro kontrolu dat, aby se ve schématu nevyskytovaly protokoly, které ani nebyly zachyceny, ale pouze jsou zmíněny v hlavičce vstupního csv souboru. Na následujícím příkladu 3.2 můžeme vidět vytvořené schéma z atributů zmíněných v přechozí ukázce.

Uzel Host má pouze jeden atribut a tím je *host.ip* obsahující jeho IP adresu. Obsahuje pouze jeden atribut, protože jeho primárním cílem je znázornit komunikaci mezi hosty pomocí hran. Vzniká z hodnot z atributů aplikace *connection*. Uzel hosta, který inicioval spojení, získá IP adresu z atributu *Source_Address* (zdrojová adresa). Koncovému hostovi bude přidělena IP adresa z *Destination_Address* (cílová adresa). Tyto dva uzly mezi sebou budou propojeny hranou *host.communicated*, která nám říká, že spolu komunikovali. Hrana vede od uzlu, který inicializoval spojení ke koncovému uzlu. Oba uzly budou spojeny i s uzlem Connection z kterého mají IP adresy. Zdrojový host bude spojen hranou *host.Originated* a koncový host hranou *host.responded*.

3.3 Transformace dat

Pro transformaci síťových toků byl přidán do toolkitu GRANEF nový modul. Stejně jako stávající transformační modul obsahuje transformační skript, jehož výstupem jsou RDF a schema soubory. Oba transformační skripty jsou napsané v jazyce Python a obsahují parsery pro jednotlivé síťové protokoly. V aktuální verzi transformační skript pro síťové toky bere vstupní soubory ve formátu csv. Časem jde přidat i podpora pro soubory ve formátu JSON. Před spuštěním transformace je nutné mít vygenerovaný csv soubor skriptem, který je popsán v předchozí sekci a jeho úkolem je přiřadit k zkratkám jejich definice a aplikaci.

Jako povinný argument má transformační skript cestu ke vstupnímu souboru a mezi volitelné patří cesty k výstupním souborům, oddělovací znak pro csv soubory a úroveň logování. Pokud není vloženo jméno výstupního souboru je jako defaultní pojmenování použito

```
1 HTTP_formats.HTTP_Hostname: string @index(trigram, exact) .
2 HTTP_formats.HTTP_Hostname_1st_level_domain: string .
3 HTTP_formats.HTTP_Hostname_1st_AND_2nd_level_domain: string .
4 HTTP_formats.HTTP_Hostname_1st_2nd_AND_3rd_level_domain: string .
5 HTTP_formats.HTTP_URL: string .
6 HTTP_formats.HTTP_URL_first_directory: string .
7 HTTP_formats.HTTP_OS: string .
8 HTTP_formats.HTTP_OS_Major_Version: int .
9 HTTP_formats.HTTP_OS_Minor_Version: int .
10 HTTP_formats.HTTP_OS_Build_Version: int .
11 HTTP_formats.HTTP_Application: string .
12 HTTP_formats.HTTP_Application_Major_Version: int .
13 HTTP_formats.HTTP_Application_Minor_Version: int .
14 HTTP_formats.HTTP_Application_Build_Version: int .
15 HTTP_formats.HTTP_Method_Type: string .
16 HTTP_formats.HTTP_Response_Code: int .
17
18 type HTTP_formats {
19   HTTP_formats.HTTP_Hostname
20   HTTP_formats.HTTP_Hostname_1st_level_domain
21   HTTP_formats.HTTP_Hostname_1st_AND_2nd_level_domain
22   HTTP_formats.HTTP_Hostname_1st_2nd_AND_3rd_level_domain
23   HTTP_formats.HTTP_URL
24   HTTP_formats.HTTP_URL_first_directory
25   HTTP_formats.HTTP_OS
26   HTTP_formats.HTTP_OS_Major_Version
27   HTTP_formats.HTTP_OS_Minor_Version
28   HTTP_formats.HTTP_OS_Build_Version
29   HTTP_formats.HTTP_Application
30   HTTP_formats.HTTP_Application_Major_Version
31   HTTP_formats.HTTP_Application_Minor_Version
32   HTTP_formats.HTTP_Application_Build_Version
33   HTTP_formats.HTTP_Method_Type
34   HTTP_formats.HTTP_Response_Code
35 <~connection.produced>
36 }
```

Obrázek 3.2: Vytvořené schéma znázorňující atributy z protokolu HTTP

nfdump-dgraph, výchozím oddělovacím znakem pro formát csv je čárka a logování je nastavené na úroveň INFO.

Celý proces transformace lze rozdělit na: přejmenování atributů jednotlivých protokolů (toto převedení zkratk na jejich popisná jména bylo popsáno v sekci 3.2), převod dat ze vstupních síťových toků do RDF trojic a vytvoření schématu. Protože z některých síťových protokolů, chceme získat specifické informace, jsou v rámci transformace dat pro tyto protokoly implementovány speciální parsery.

Zachycený síťový provoz může nabývat velkých rozměrů, proto se u převodu informací ze síťových toků do RDF trojic, využívají vlákna. Na začátku si transformační skript vytvoří pro každé jádro procesoru jedno vlákno. Z výkonnostního hlediska musí být vláken stejně, nebo méně, než je jader v procesoru. Kdyby jich bylo více, tak by se musely vlákna přepínat mezi jádry, a to by zpomalovalo celou transformaci. V dalším kroku si rozdělí vstupní soubor na tolik částí, kolik má definovaných vláken. Každé vlákno poté dostane počáteční index a koncový index, aby vědělo odkud má začít soubor číst a kde skončit. Následně si otevře vstupní soubor, kde každý řádek reprezentuje jeden síťový tok. Řádky načítá postupně a spouští nad nimi všechny parsery, které nenulová data převedou do RDF trojic.

Každý zpracovaný síťový tok musí mít své unikátní id. Kdyby se v implementaci nepoužívaly vlákna, stačilo by vzít `_:` a za to přidat číslo řádku. Aby se u vláken zachovala unikátnost id, dostane vlákno písmeno od a-z, postupně podle pořadí, jak se vytváří. Id řádku pak bude vypadat následovně: `_[a-z]číslo řádku`.

3.4 Parsery podle typu protokolu

Vstupní csv soubor má na každém řádku jeden síťový tok, v němž jsou uloženy informace z různých síťových protokolů. Obsahuje například základní informace o spojení mezi dvěma hosty a v případě, že spolu komunikovali přes DNS zaznamenává i jejich odpovědi. V důsledku toho, že některé síťové protokoly chceme zpracovávat specificky jsou použity speciální parsery. U DNS, TLS a HTTP je žádoucí z vstupních dat vytáhnout hostname. U DNS se provádí kontrola odpovědi serveru. Pro všechny ostatní síťové protokoly se využívá generický parser, který jen vezme všechna data a převede je do RDF formátu.

Může se stát, že vstupní soubor bude v některých záznamech obsahovat speciální znaky. Je tedy nutné je ošetřit. Například před každé zpětné lomítko bylo nutné přidat ještě jedno a některé znaky se musely úplně smazat (např. `\n`, nebo `\t`), aby mohla proběhnout následná indexace dat a nahrání do databáze Dgraph.

Jeden csv řádek obsahuje všechna data o použitých protokolech podle hlavičky, to zapříčiní velké množství prázdných dat. Například pokud v rámci jednoho síťového toku nedošlo k HTTP spojení, obsahují HTTP sloupce prázdná data jako jsou nuly, nebo "N/A". Proto se na začátku každého parseru definují tato nulová data, aby se v průběhu vytváření RDF trojic ignorovala a nezabírala místo v databázi. Může se i stát, že v hlavičce jsou definované sloupce pro velmi specifické odpovědi, ke kterým nedochází vždy. V tomto případě se nullova data ze síťového toku převedou do RDF trojic a nulové hodnoty se nezpracují.

Podle toho, jaký nástroj se pro záchyt síťových toků využije, je daná časová přesnost časové známky (timestamp). Například některé nástroje zaznamenávají časové složky s přesností na sekundy, jiné dokonce až na mikrosekundy. S časovým formátem s přesností na mikrosekundy je náročné pracovat. Po konzultaci s analytiky z týmu CSIRT-MU se došlo k závěru, že taková přesnost není nutná. Mikrosekundy se tedy zahazují a ponechává se přesnost na milisekundy.

4 Analytické dotazy na zvoleném datasetu

Přidání nových analytických dotazů do API modulu toolkitu GRANEF bylo posledním krokem při přidávání podpory zpracování síťových toků. Dotazy jsou formulovány obecně, aby je bylo možné použít v různých scénářích a zaměřují se na detekci nejčastějších síťových útoků.

Předlohou nových dotazů byly již stávající, které aplikační rozhraní toolkitu GRANEF obsahuje. Ty se upravily pro potřeby analýzy síťových toků, protože na rozdíl od analýzy paketů, síťové toky neobsahují tolik detailních informací o spojení. Na druhou stranu poskytují lepší náhled, co se v síti děje.

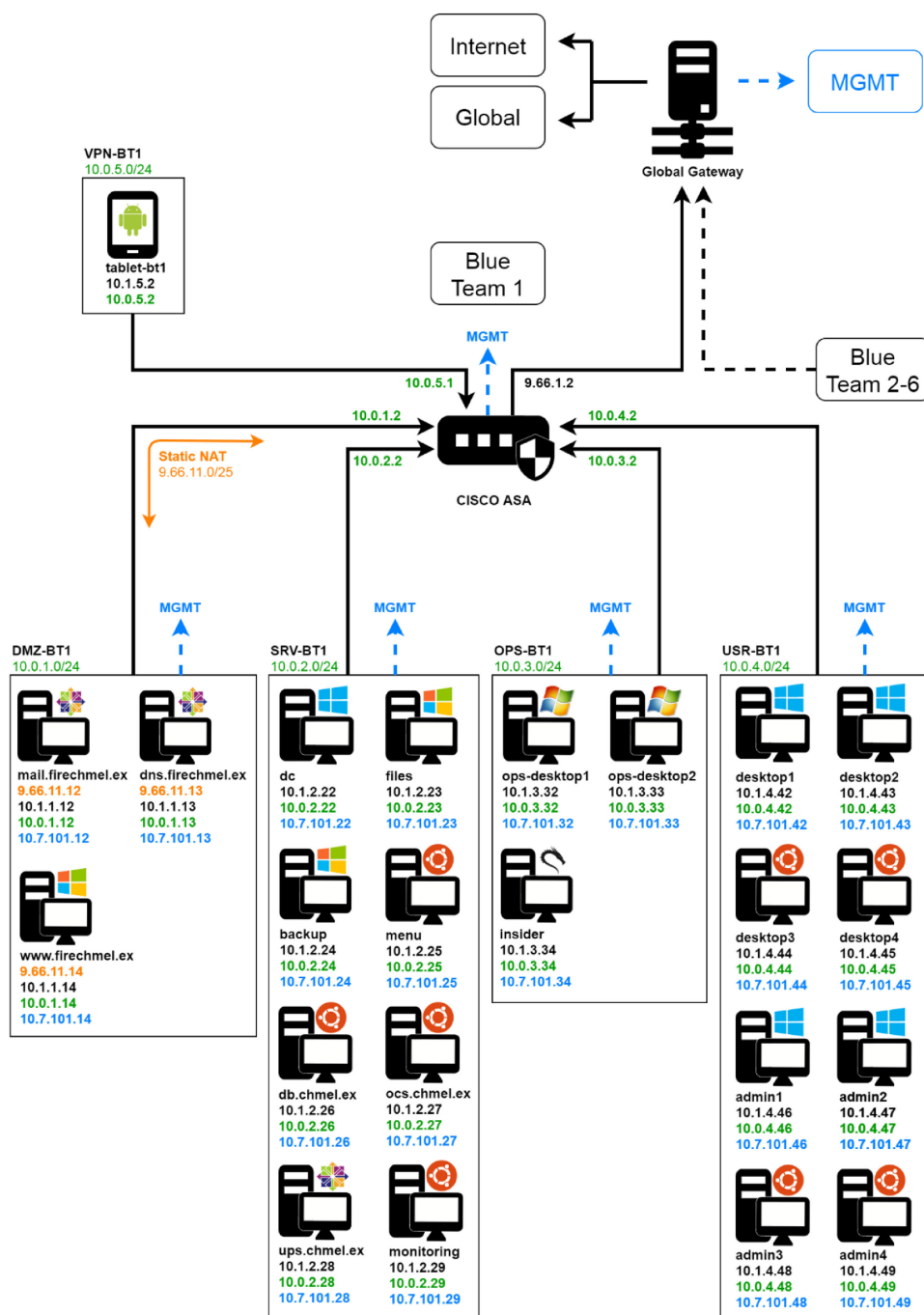
4.1 Dataset

Datasetem využitým pro testování analytických dotazů je záchyt síťového provozu z dvoudenního bezpečnostního cvičení Cyber Czech z roku 2019, který popisuje článek [35] a v následujících odstavcích z něj budu vycházet. Dataset obsahuje záchyt síťového provozu ve formě síťových toků ze strojů s operačními systémy Linux a Windows. Topologie cvičné sítě byla navržena stejně jako jsou navrženy sítě v moderních organizacích. Zahrnuje síťová zařízení s běžnými operačními systémy a aplikacemi. Výsledkem jsou síťová data, která jsou stejná jako data viděná v reálném světě, bez nutnosti anonymizace, připravená ke zpracování a analýze. Schéma zjednodušené topologie je vidět na obrázku 4.1.

Topologie sítě se skládá ze dvou samostatných sítí: cvičné sítě a kontrolní sítě. Cvičná síť je na obrázku znázorněna plnými čarami, zatímco kontrolní síť je znázorněna modrými přerušovanými čarami. Podpůrná kontrolní síť má hvězdicovou strukturu a je navržena tak, aby umožňovala přístup do všech zařízení během celého cvičení.

Cvičení se účastnilo několik týmů rozdělených do dvou skupin. Modrá skupina měla na starost zabezpečení sítě a řešení hrozeb vyvolaných červenou skupinou. Červená skupina měla za úkol útočit na síť útoky podle rozpisu a snažit se získat co nejvíc informací o fiktivní společnosti. Útoky se nemusely stát přesně v čas, kdy byly stanoveny

4. ANALYTICKÉ DOTAZY NA ZVOLENÉM DATASETU



Obrázek 4.1: Zjednodušená topologie [35]

v rozpisu, ale během cvičení měly proběhnout. Část rozpisu vypadala takto:

čas	útok	závažnost útoku
2:00 - 2:40	webový útok	vysoká
2:30 - 3:00	Denial of service	střední
3:00 - 4:00	Únik dat	vysoká
3:30 - 4:00	Sabotáž kritické aplikace	velmi vysoká

Síťový provoz generovaný během cvičení byl zachycen na výstupní bráně (Global Gateway), protože je to jediný bod v síti kudy prochází komunikace mezi červenou a modrou skupinou. Zařízení jsou v datasetu identifikována svými globálními adresami, protože výstupní brána patří do globální sítě. Ve schématu 4.1 jsou znázorněny černou barvou. Dataset neobsahuje žádné informace, které se nedostaly až k výstupní bráně. Konkrétně se jedná o interakci modrého týmu s interními webovými a mailovými servery a interakce červeného týmu s vnějšími internetovými zdroji.

4.2 Příklad možné analýzy na základě alertu

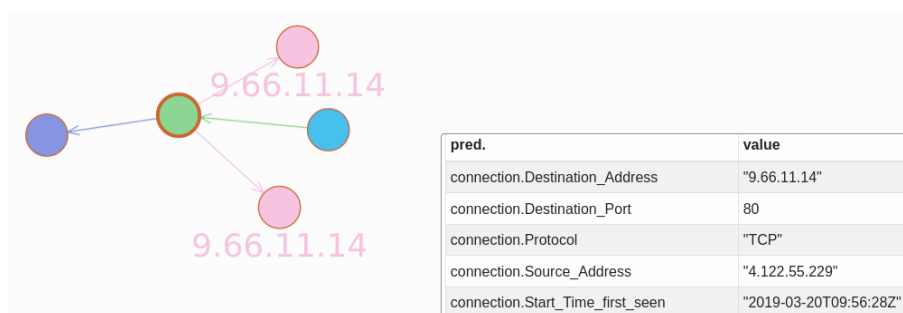
Jako ukázkou simulace běžné analýzy jsme společně s analytiky z týmu CSIRT-MU vybrali následující událost zachycenou v daném datasetu. Pro přiblížení postupu analytika při řešení bezpečnostního incidentu bylo stanoveno pět otázek, které budou v průběhu analýzy zodpovězeny.

Víme, že se malware dostal do počítače pomocí požadavku POST z protokolu HTTP na IP adresu 9.66.11.14:80/back.php z 4.122.55.229 pomocí curl/7.61.0 a nahrál se spustitelný soubor s názvem „file.exe“. Odpovídající TCP spojení začíná v 09:56:02 20. 3. 2019 a trvá 0,045245 sekund.

S kým komunikoval host 4.122.55.229 můžeme vidět na grafu 4.2 získaného dotazem *connection_info*. Ten najde uzel hosta s IP adresou 4.122.55.229. Vezme hranu *host.originated* jdoucí z nalezeného uzlu reprezentující spojení, které vytvořil. Nad touto hranou se provedou dva filtry, které zajistí, aby se vypsalý jen spojení probíhající v době útoku. U nalezeného spojení se vypíší atributy reprezentující čas, kdy

komunikace začala, IP adresy komunikujících hostů, použitý protokol a port. Pro ověření se vypíše host, který na komunikaci odpověděl pomocí zpětné hrany *host.responded*. Abychom prověřili, že se skutečně jednalo o HTTP požadavek, se vypíše uzel znázorňující HTTP spojení, ke kterému se dostane přes hranu *connection.produced*. Dotaz *connection_info* vypadá následovně:

```
{
  connection_info(func: eq(host.ip, cidr,
    "4.122.55.229")) {
    label : host.ip
    host.ip
    host.Originated
    @filter(gt(
      connection.Start_Time_first_seen,
      "2019-03-20T09:56:00")
    AND
    lt(connection.Start_Time_first_seen,
      "2019-03-20T10:56:10"))
    {
      connection.Start_Time_first_seen
      connection.Source_Address
      connection.Destination_Address
      connection.Protocol
      connection.Destination_Port
      ~host.responded {
        label : host.ip
      }
      connection.produced {
        expand(HTTP_formats)
      }
    }
  }
}
```



Obrázek 4.2: Komunikace s hostem 4.122.55.229

Hledal útočník nějaké zranitelnosti a zneužil některou z nich?

Nejprve musíme zjistit, kdo inicializoval spojení se serverem předtím, než se incident udál, abychom mohli prověřit, zda nedošlo ještě k nějakým útokům před nahráním souboru na server. Dotaz *responded_hosts* nalezne uzel Hosta s IP adresou 9.66.11.14, poté na hranu *host.responded* použije filtr, který vezme jen ty Connection uzly, kde je atribut *connection.Start_Time_first_seen* menší než časová známka, kdy proběhl útok. Na získané uzly se použije agregace podle atributu *connection.Source_Address* a vrátí se počet získaných uzlů se stejnou zdrojovou IP adresou. Implementace dotazu *responded_hosts* je následující:

```
{
  responded_hosts(func: allof(host.ip, cidr,
    "9.66.11.14")) {
    label : host.ip
    host.responded
    @filter(lt(
      connection.Start_Time_first_seen,
      "2019-03-20T09:56:02"))
    @groupby(connection.Source_Address)
    {
      count(uid)
    }
  }
}
```

Z výsledného JSON výstupu zjistíme následující spojení a jejich četnost.

Communicated Host IP	count
4.122.55.221	2540
27.3.0.196	304
4.122.55.5	16
172.16.1.1	10

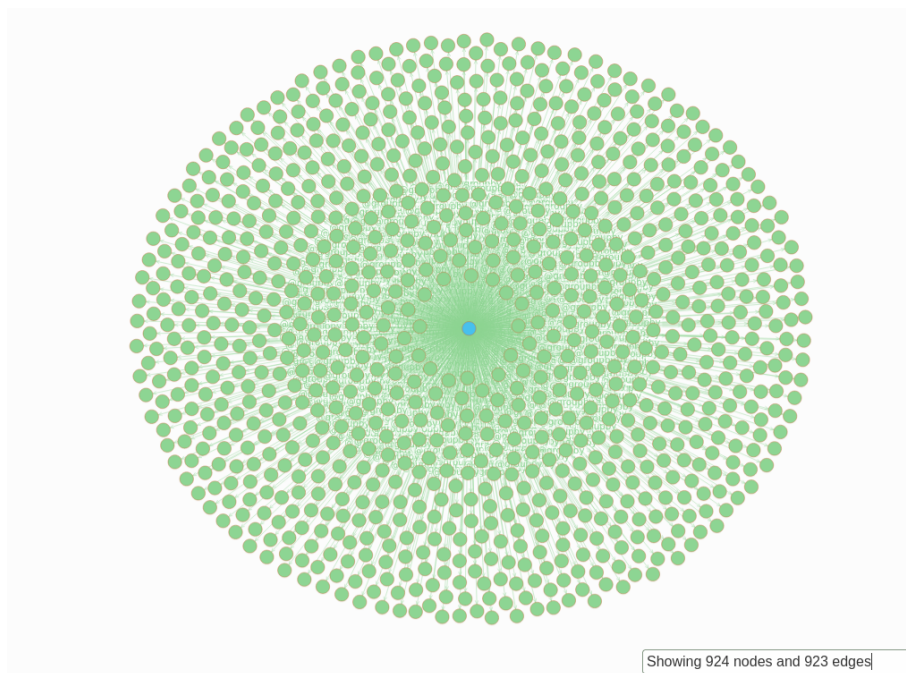
Postupně prověříme všechny hosty a začneme s 4.122.55.221. Protože je vidět velké množství spojení, zkusíme prověřit přístup k portům, pomocí příkazu *resp_port_usage_between_two_hosts*. Ten nejprve vytvoří proměnnou *connectionsWithWantedHost* s uloženou komunikací mezi hosty s IP adresami 4.122.55.221 a 9.66.11.14. Potom pomocí funkce *getRespPortUsage* vrátí kolikrát byl koncový port při komunikaci použit. Následující ukázka znázorňuje implementaci dotazu *resp_port_usage_between_two_hosts*:

```
query resp_port_usage_between_two_hosts
  ($host_ip1: string, $host_ip2: string)
{
  var(func: eq(host.ip, $host_ip1)) {
    connectionsWithWantedHost as host.Originated
    @cascade {
      ~host.responded @filter(eq(host.ip,
        $host_ip2)) {}
    }
  }
  getRespPortUsage
    (func: uid(connectionsWithWantedHost))
    @groupby(connection.Destination_Port) {
      responded_count : count(uid)
    }
}
```

Ve výsledném grafu 4.3 si můžeme všimnout velkého počtu použitých portů, je jich skoro 1000. Po nahlédnutí do výpisu JSON, zjistíme, že kromě standartního portu 80, který využívá HTTP. Jsou porty použity jednou až šestkrát. Což vypadá na použití nějakého nástroje

4. ANALYTICKÉ DOTAZY NA ZVOLENÉM DATASETU

pro skenování portů (například nástroje *nmap*, ten obsahuje dokonce nastavení, kde skenuje 1000 nejpoužívanějších portů).



Obrázek 4.3: Využití portů v komunikaci mezi hosty 4.122.55.221 a 9.66.11.14

Menší ukázka výsledného JSON výpisu vypadá následovně:

```
...
{
  "connection.Destination_Port": 2003,
  "responded_count": 1
},
{
  "connection.Destination_Port": 3328,
  "responded_count": 1
},
{
  "connection.Destination_Port": 5987,
  "responded_count": 1
}
```

```
},  
{  
  "connection.Destination_Port": 13782,  
  "responded_count": 1  
},  
{  
  "connection.Destination_Port": 19842,  
  "responded_count": 1  
},  
{  
  "connection.Destination_Port": 3,  
  "responded_count": 2  
},  
{  
  "connection.Destination_Port": 9,  
  "responded_count": 2  
},  
{  
  "connection.Destination_Port": 13,  
  "responded_count": 2  
},  
...  

```

Pokud stejný postup zopakujeme pro IP adresu 27.3.0.196. Zjistíme, že na necelých 200 portů se pokusil host 9.66.11.14 připojit jednou až dvakrát. Opět to vypadá na skenování dostupných portů. U poslední IP adresy 4.122.55.5 se využívaly pouze HTTP porty 80 a 443 a není zde vidět žádná podezřelá aktivita.

Komunikoval stroj s nějakou další nebezpečnou IP adresou?

Už jsme zjistili, co se dělo před nahráním souboru, teď je potřeba zjistit s kým host 9.66.11.14 komunikoval po nahrání. Dotaz *originated_hosts* nejprve nalezne uzel hosta s IP adresou 9.66.11.14 a projde jeho hrany *host.originated*. Nad nimi spustí filtrování, aby atribut *connection.Start_Time_first_seen* byl větší, než je čas, kdy došlo k útoku. Výsledné hrany agreguje podle cílové IP adresy a vypíše jejich četnost během komunikací. Dotaz *originated_hosts* vypadá následovně:

```
{
    originated_hosts(func: allop(host.ip, cidr,
        "9.66.11.14")) {
        label : host.ip
        host.originated
        @filter(gt(
            connection.Start_Time_first_seen,
            "2019-03-20T9:56:02"))
        @groupby(connection.Destination_Address)
        {
            count(uid)
        }
    }
}
```

Z odpovědi zjistíme IP adresy a jejich četnost, které jsou znázorněny v následující tabulce. Po prozkoumání komunikace mezi nalezenými hosty, co se serverem 9.66.11.14 komunikovali nejvíce, zjistíme, že host 216.58.201.67 komunikoval pomocí protokolu HTTP a není vidět žádná podezřelá aktivita. Komunikace s hostem 4.122.55.111 proběhla pomocí HTTPS a data jsou zašifrovaná, tudíž nejdou vidět žádné detailní informace. Host 104.103.90.39 používal HTTP i HTTPS a opět není žádná podezřelá komunikace vidět.

Response Host IP	count
216.58.201.67	14
4.122.55.111	13
104.103.90.39	8
4.122.55.5	2
52.178.147.240	1
216.58.201.106	1
204.79.197.200	1
152.199.19.161	1

Odeslal stroj velké množství dat mimo síť?

Statistiku poslaných bajtů zjistíme pomocí příkazu *sum_input_bytes*, který najde hosta s IP adresou 9.66.11.14 a vezme všechny spojení na hraně *host.originated*. Nalezená spojení vyfiltruje, aby zůstali jen ty, co proběhli až po útoku a agreguje je podle koncové IP adresy. Výsledkem agregace je součet všech vstupních bajtů. Dotaz *sum_input_bytes* je definován následovně:

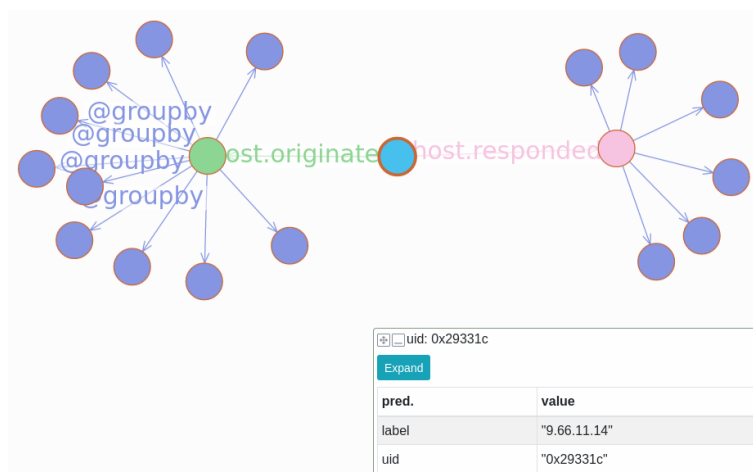
```
{
  sum_input_bytes(func: allof(host.ip, cidr,
    "9.66.11.14")) {
    label : host.ip
    host.originated
    @filter(gt(
      connection.Start_Time_first_seen,
      "2019-03-20T9:56:02"))
    @groupby(connection.Destination_Address)
    {
      sum(connection.Input_Bytes)
    }
  }
}
```

Zjistíme, že objem dat je relativně malý (482 kB) a tudíž se nemusíme obávat úniku dat. V opačném případě bychom prověřili ty IP adresy, na které bylo posláno velké množství dat.

Host IP	sum of bytes
4.122.55.111	4945762
52.178.147.240	36812
104.103.90.39	29730
216.58.201.67	12238
152.199.19.161	11366
204.79.197.200	9269
4.122.55.5	6682
216.58.201.106	184

Komunikoval server se zařízeními v lokální síti a pokud ano, jak?

Nejprve prověříme, s kým komunikoval server pomocí své veřejné IP adresy 9.66.11.14. Tuto IP adresu vložíme jako argument do příkazu `count_communication_of_host`, který vrátí seznam zařízení, s kterými zadaný host komunikoval a kolikrát ke komunikaci došlo. Výsledný graf je vidět na obrázku 4.4. Tento dotaz byl zvolen proto, abychom měli lepší přehled o tom, s kým server komunikoval.



Obrázek 4.4: Zařízení s kterými server 9.66.11.14 komunikoval

```
query count_communication_of_host($host_ip:
  string)
{
  count_communication_of_host(func: allof(host.
    ip, cidr, $host_ip)) {
    label : host.ip
    host.Originated
      @filter(gt(
        connection.Start_Time_first_seen,
        "2019-03-20T9:56:02"))
    @groupby(connection.Destination_Address)
    {
      count(uid)
    }
    host.responded
  }
}
```

4. ANALYTICKÉ DOTAZY NA ZVOLENÉM DATASETU

```
@filter(gt(
    connection.Start_Time_first_seen,
    "2019-03-20T9:56:02"))
@groupby(connection.Source_Address)
{
    count(uid)
}
}
```

Z grafu vyčteme následující hosty, s nimiž server komunikoval.

Responded Host IP	count
4.122.55.111	14
216.58.201.67	7
172.16.1.1	4
104.103.90.39	3
65.55.252.93	2
4.122.55.5	1
216.58.201.106	1
204.79.197.200	1
152.199.16.161	1
129.90.41.7	1

Další tabulka popisuje hosty, kteří navázali spojení se serverem.

Originated Host IP	count
4.122.55.5	51
172.16.1.1	20
77.51.161.30	16
4.122.55.221	8
4.122.55.229	2
4.122.55.1	1

4. ANALYTICKÉ DOTAZY NA ZVOLENÉM DATASETU

Abychom prověřili i privátní IP adresu serveru, spustíme dotaz *count_communication_of_host* a jako vstupní argument vložíme IP adresu 10.1.1.14. Zjistíme, že tentokrát komunikoval pouze server s následujícími hosty.

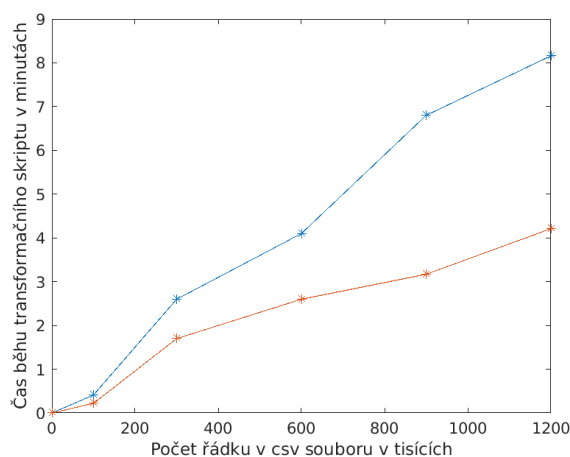
Responded Host IP	count
4.122.55.111	45
216.58.201.67	9
216.58.201.106	3
52.138.216.83	2
204.79.197.200	2
104.103.90.39	2
65.55.252.93	1
40.77.226.249	1
40.70.184.83	1
152.199.19.161	1
13.107.4.50	1

Všechny dotazy použité při ukázce analýzy je možné zkopírovat a vložit jako *custom_query* do webového rozhraní toolkitu GRANEF. V případě, že pro analýzu používáme aplikační modul je cesta k dotazu následující */flow/<název dotazu>*.

5 Evaluace implementace

Sada nástrojů GRANEF má funkci pro spuštění jednotlivých modulů popořadě, jak jsou definovány ve schématu 2.1. Pokud máme nějaký záchyt síťového provozu ve formátu PCAP stačí spustit příkaz `./granef.py -a -d <DATA_DIRECTORY> -i <TRACE_FILE>.pcap` a moduly se postupně spustí v následujícím pořadí: extrakční modul, transformační modul, indexace, nahrání do databáze, analýza. I nově vytvořené moduly pro zpracování síťových toků lze spustit tímto způsobem, je ale potřeba přidat přepínač `-c granef-flow.yml`. Tento přepínač zajistí, aby se při spuštění použil konfigurační soubor, ve kterém jsou definovány docker image obsahující moduly pro síťové toky. Formát vstupních souborů do extrakčního modulu je nfcapd.

Transformační modul využívá paralelního zpracování, aby byl schopný v co nejkratším čase zpracovat velké vstupní soubory. Původně byl implementován sekvenčně, ale toto řešení se ukázalo jako nevhodné při zpracování souborů o velikosti několika gigabajtů. Menší ukázka mezi jednotlivými implementacemi je vidět na obrázku 5.1. K měření času, jak dlouho skript běžel jsem využila linuxový příkaz `time` a skript běžel na stroji s čtyřjádrovým procesorem. Tudíž červená barva značí skript, který k zpracování využil čtyři vlákna. Modrá barva znázorňuje jednovláknový běh.



Obrázek 5.1: Měření rychlosti transformačního skriptu

Závěr

Cílem mé bakalářské práce bylo rozšířit sadu nástrojů GRANEF o zpracování a analýzu síťových toků. Dříve než bylo možné implementovat nové moduly, bylo nutné se seznámit s problematikou zpracování síťových toků a problematikou analýzy síťového provozu na základě vztahů za pomoci nástroje GRANEF.

Na základě získaných poznatků jsem navrhla a implementovala moduly pro extrakci a transformaci síťových toků do nástroje GRANEF. Extrakční modul převádí síťové toky ze souborů formátu *nfcapd* na obousměrné síťové toky. Transformační modul je převede do formátu RDF a vytvoří databázové schéma. Výsledné soubory jsou podporované databází Dgraph, do které se nahrají. Následně jsem rozšířila aplikační rozhraní nástroje GRANEF o analytické dotazy zaměřené na analýzu síťového provozu ve formě síťových toků. Analytické dotazy byly otestované nad datasetem získaným ze cvičení Cyber Czech z roku 2019.

Výhodou přidání podpory síťových toků do nástroje GRANEF je jeho širší využití při analýze síťového provozu. Teď podporuje nejen analýzu paketů, ale i síťových toků. Analýza síťových toků je méně náročná na výpočetní zdroje. To umožňuje analytikovi nahrát velké množství síťových toků do databáze, které poté může analyzovat.

Na tuto práci by se dalo navázat rozšířením optimálnějšího načítání dat do transformačního skriptu. Analytik by měl možnost si označit soubory, které se mají prioritně zpracovat a nahrát do databáze. Zbývající soubory by se zpracovávali postupně na pozadí. Dalším rozšířením by mohl být nový analytický modul, který by monitoroval síťový provoz a snažil se detekovat podezřelé aktivity.

Obsah přílohy

- readme – shrnuje obsah přílohy
- /extrakční_modul/ – implementace extrakčního modulu
- /transformační_modul/ – implementace transformačního modulu
- /API_modul/ – implementovala jsem pouze analytické dotazy týkající se síťových toků (jedná se o soubory ve složce routers začínající slovem flow)
- /API_modul/handler/ – tento modul jsem neimplementovala, je zde přiložen pouze pro účely testování API modulu
- /ukázka_transformace/ – ukázka výstupních souborů z transformačního modulu

Bibliografie

1. HOFSTEDE, R.; ČELEDA, P.; TRAMMELL, B.; DRAGO, I.; SADRE, R.; SPEROTTO, A.; PRAS, A. Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS*. 2014, roč. 16. ISSN 1553-877X. Dostupné také z: <https://ieeexplore.ieee.org/document/6814316>.
2. RDF WORKING GROUP. *RDF* [online]. 2014. Ver. 1.1 [cit. 2021-10-31]. Dostupné z: <https://www.w3.org/RDF/>.
3. GORDON, L. *nmap*. 2021. Ver. 7.90. Dostupné také z: <https://nmap.org>.
4. WEBER, J. *The Fundamentals of Passive Monitoring Access*. 2006. Dostupné také z: <https://archive.nanog.org/meetings/nanog37/presentations/joy-weber.pdf>.
5. ZHANG, J.; MOORE, A. Traffic Trace Artifacts due to Monitoring Via Port Mirroring. In: 2007, s. 1–8. ISBN 1-4244-1289-7. Dostupné z doi: 10.1109/E2EMON.2007.375317.
6. AWATI, R.; SCARPATI, J. *deep packet inspection (DPI)*. SearchNetworking, 2017. Dostupné také z: <https://searchnetworking.techtarget.com/definition/deep-packet-inspection-DPI>.
7. CLAISE, B.; TRAMMELL, B.; AITKEN, P. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. 2013-09. RFC, 7011. IETF. Dostupné také z: <http://tools.ietf.org/rfc/rfc7011.txt>.
8. LUCAS, Michael W. *Network flow analysis*. 1st ed. San Francisco: No Starch Press, 2010. ISBN 978-159327-203-6.
9. VELAN, P. *Application-Aware Flow Monitoring* [online]. 2018 [cit. 2021-12-07]. Dostupné také z: <https://is.muni.cz/th/a2fxd/>. Disertační práce. Masarykova univerzita, Fakulta informatiky, Brno. Vedoucí práce Pavel ČELEDA.
10. LYON, G. *npcap*. 2021. Ver. 1.60. Dostupné také z: <https://nmap.org/npcap/>.

11. THE TCPCDUMP GROUP. *libpcap*. 2021. Ver. 1.10.0. Dostupné také z: <https://www.tcpdump.org/>.
12. WAND RESEARCH GROUP. *libtrace*. 2021. Ver. 4.0.16. Dostupné také z: <https://research.wand.net.nz/software/libtrace.php>.
13. ZSEBY, T.; MOLINA, M.; DUFFIELD, N.; NICCOLINI, S.; RASPALL, F. *Sampling and Filtering Techniques for IP Packet Selection*. 2009-03. RFC, 5475. IETF. Dostupné také z: <http://tools.ietf.org/rfc/rfc5475.txt>.
14. D'ANTONIO, S.; ZSEBY, T.; HENKE, C.; PELUSO, L. *Flow Selection Techniques*. 2013-09. RFC, 7014. IETF. Dostupné také z: <http://tools.ietf.org/rfc/rfc7014.txt>.
15. CISCO SYSTEMS, INC., SAN JOSE, CA AND USA. *Flexible NetFlow NetFlow V5 Export Protocol* [online]. 2006 [cit. 2021-05]. Dostupné z: <https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/fnetflow/configuration/xe-16-11/fnf-xe-16-11-book/fnf-v5-export.pdf>.
16. CLAISE, B. *Cisco Systems NetFlow Services Export Version 9*. 2004-10. RFC, 3954. IETF. Dostupné také z: <http://tools.ietf.org/rfc/rfc3954.txt>.
17. NTOP. *nProbe* [online]. 2021. Ver. 9.4 [cit. 2021-05-06]. Dostupné z: <https://www.ntop.org/products/netflow/nprobe>.
18. SURDU, B. *fProbe* [online]. 2003. Ver. 1.8 [cit. 2021-05-06]. Dostupné z: <http://manpages.ubuntu.com/manpages/bionic/man8/fprobe.8.html>.
19. CARNEGIE MELLON UNIVERSITY. *YAF* [online]. 2020. Ver. 2.0 [cit. 2021-05-06]. Dostupné z: <https://tools.netsa.cert.org/yaf/docs.html>.
20. CISCO SYSTEMS, INC. *Cisco NGA*. 2012. Dostupné také z: https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_generation/1-0/overview/NGAoverview.html.
21. FLOWMON NETWORKS A.S. *Flowmon Probe*. 2021. Dostupné také z: <https://support.kemptechnologies.com/hc/en-us/articles/4404510533901-Flowmon-11-1-Documentation>.

22. SPEROTTO, A.; SCHAFFRATH, G.; SADRE, R.; MORARIU, C.; PRAS, A.; STILLER, B. An Overview of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys and Tutorials*. 2010, roč. 12, s. 343–356. Dostupné z doi: 10.1109/SURV.2010.032210.00054.
23. ČERMÁK, M.; ŠRÁMKOVÁ, D. GRANEF: Utilization of a Graph Database for Network Forensics. In: SABRINA DE CAPITANI DI VIMERCATI, Pierangela Samarati (ed.). *Proceedings of the 18th International Conference on Security and Cryptography*. Portugal: SCITEPRESS, 2021, s. 785–790. ISBN 978-989-758-524-1. Dostupné z doi: <http://dx.doi.org/10.5220/0010581807850790>.
24. THE WIRESHARK TEAM. *Wireshark* [online]. 2021. Ver. 3.4.9 [cit. 2021-10-20]. Dostupné z: <https://www.wireshark.org/docs/>.
25. TAO, X.; LIU, Y.; ZHAO, F.; YANG, Ch.; WANG, Y. Graph database-based network security situation awareness data storage method. *J Wireless Com Network* 2018. 2018, roč. 294. Dostupné z doi: 10.1186/s13638-018-1309-9.
26. G2.COM, INC. *Best Graph Databases*. 2021. Dostupné také z: <https://www.g2.com/categories/graph-databases>. [online], [cit. 2021-8].
27. NEISE, P. Intrusion Detection Through Relationship Analysis. *SANS Institute*. 2016. Dostupné také z: <https://www.sans.org/white-papers/37352/>.
28. DGRAPH LABS, INC. *Dgraph Database Overview*. 2021. Dostupné také z: <https://dgraph.io/docs/dgraph-overview/>. [online], [cit. 2021-8].
29. DGRAPH LABS, INC. *Badger* [online]. 2021. Ver. 2.2007.4 [cit. 2021-05-06]. Dostupné z: <https://github.com/dgraph-io/badger>.
30. BURK, N. *Schema Definition Language* [online]. 2017 [cit. 2021-10-20]. Dostupné z: <https://www.prisma.io/blog/graphql-sdl-schema-definition-language-6755bcb9ce51>.
31. THE GRAPHQL FOUNDATION. *GraphQL*. 2021. Dostupné také z: <https://graphql.org/>. [online], [cit. 2021-8].

-
32. THE ZEEK PROJECT. *Zeek* [online]. 2021. Ver. 4.0.4 [cit. 2021-10-25]. Dostupné z: <https://zeek.org/>.
 33. THE DOCKER TEAM. *Docker* [online]. 2021. Ver. 20.10.9 [cit. 2021-10-25]. Dostupné z: <https://www.docker.com/>.
 34. ČERMÁK, M. *The 16th International Conference on Availability, Reliability and Security*. Graph-based Network Traffic Analysis for Incident Investigation. 2021. Dostupné také z: <https://is.muni.cz/auth/publication/1786778/cs>.
 35. TOVARŇÁK, D.; ŠPAČEK, S.; VYKOPAL, J. Traffic and log data captured during a cyber defense exercise. *Data in Brief*. 2020, roč. 31, s. 105784. issn 2352-3409. Dostupné z doi: <https://doi.org/10.1016/j.dib.2020.105784>.