VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EFEKTIVNÍ DETEKCE SÍŤOVÝCH ANOMÁLIÍ S VYUŽITÍM DNS DAT

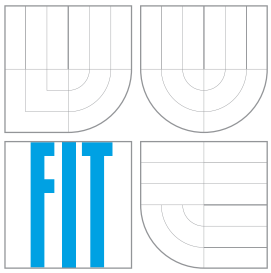
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ FOMICZEW

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EFEKTIVNÍ DETEKCE SÍŤOVÝCH ANOMÁLIÍ S VYUŽITÍM DNS DAT

EFFECTIVE NETWORK ANOMALY DETECTION USING DNS DATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ FOMICZEW

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL KOVÁČIK

BRNO 2015

Abstrakt

Tato práce se zabývá návrhem a implementací systému pro efektivní detekci síťových anomálií s využitím DNS dat. Zvýšení efektivity detekčního systému je dosaženo kombinací více spolupracujících detektorů a různých detekčních technik. Vstupní data detekčního systému představují data o síťových tocích a paketech ve formátech *NetFlow*, *IPFIX* a *pcap*. Hlavní důraz je kladen na detekci tunelování přes DNS. Práce také obsahuje popis Systému doménových jmen (DNS) a anomálií s ním spojených.

Abstract

This thesis describes the design and implementation of system for effective detection of network anomaly using DNS data. Effective detection is accomplished by combination and cooperation of detectors and detection techniques. Flow data in *NetFlow* and *IPFIX* formats are used as input for detection. Also packets in *pcap* format can be used. Main focus is put on detection of DNS tunneling. Thesis also describes Domain Name System (DNS) and anomalies associated with DNS.

Klíčová slova

DNS, anomálie, tunelování, DoS, detekce, NetFlow, IPFIX, pcap, IP

Keywords

DNS, anomalies, tunneling, DoS, detection, NetFlow, IPFIX, pcap, IP

Citace

Jiří Fomiczew: Efektivní detekce síťových anomálií s využitím DNS dat, diplomová práce, Brno, FIT VUT v Brně, 2015

Efektivní detekce síťových anomálií s využitím DNS dat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Kováčika. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bc. Jiří Fomiczew
24. května 2015

Poděkování

Chtěl bych poděkovat panu Ing. Michalu Kováčikovi za odborné vedení a vstřícnou pomoc při vypracování této práce. Dále bych chtěl poděkovat mé rodině, která mě podporovala po celou dobu studia.

Obsah

Kapitola 1

Úvod

Počítačové sítě se v poslední době stávají nepostradatelné a jejich správné fungování je základním předpokladem pro většinu odvětví současného světa. Bohužel také dramaticky přibývá počet útoků a nepřátelských aktivit, které se na síti dějí. Je tedy v našem zájmu pracovat na metodách detekce a prevence vůči stávajícím i novým typům síťových útoků.

Dle posledních zpráv společností zabývajících se bezpečností (např. [49, 21, 54]) se každým rokem zvyšuje počet útoků na Internetu. Poměrně velká část těchto útoků pak směřuje na službu DNS (dle [49] tvořily útoky na systém DNS či s jeho využitím 16 % z celkového množství útoků v roce 2014, v roce 2013 to bylo dokonce 21 %). Proto se v této práci soustředím na útoky a anomálie související se systémem DNS.

Systém DNS[1] je jednou ze základních služeb Internetu. Poskytuje překlad IP adres na doménová jména a naopak [42]. To je důležité nejen pro uživatele, kteří si nejsou schopni zapamatovat IP adresy, avšak doménová jména ano, ale i pro většinu existujících aplikací. Proto je velmi důležité zajistit nejen funkčnost systému DNS, ale i eliminovat možnosti jeho zneužití.

Útokům na DNS nahrává fakt, že je služba decentralizovaná a při jejím návrhu nebyl kladen zvláštní důraz na její bezpečnost. Útok tedy může probíhat také mimo cíl (například změnou dat na jiných serverech), což ztěžuje detekci takového útoku. Služba díky použití transportního protokolu UDP také umožňuje podvržení zpráv či jejich částí.

V souvislosti s DNS existuje několik druhů útoků a anomálií. Nejčastějším útokem využívajícím systém DNS je *reflexivní útok pro odepření služby*[2], který oplývá jednoduchostí a velkou účinností (v roce 2013 proběhl takový útok na web `www.spamhaus.org` a dle [21] byl za pomocí třiceti tisíc volně dostupných DNS serverů vygenerován datový tok 300 *Gbps* – největší v historii). Kategorii zneužití systému DNS představuje například *tunelování pomocí DNS*. To většinou slouží pro obcházení bezpečnostní politiky sítě a neoprávněný přístup na Internet. Často také slouží pro skrytí komunikace *malware*[3] s útočníkem [30]. Dalším nebezpečím je podvržení informací v systému DNS, takže útočník může přesměrovat uživatele na falešné servery za účelem *phishingu*[4] či instalace malware.

Díky charakteru služby DNS většinou není možné ji úplně zakázat a eliminovat tak s ní související hrozby. Proto je třeba vyvíjet nové a zdokonalovat stávající metody pro detekci a obranu proti anomáliím a útokům, které DNS zneužívají.

[1] *Domain Name System* – systém doménových jmen.
[2] Cílem útoku je zahlcení cílového serveru a tím způsobení jeho nedostupnosti.
[3] Škodlivý software.
[4] Snaha získat důvěrné informace uživatele vydáváním se za někoho jiného (např. imitací webové stránky).

Cílem této práce bylo vytvořit detekční systém, který bude efektivním způsobem detekovat vybrané typy útoků a anomálií na základě vstupních DNS dat. Zvýšení efektivity detekčního systému je dosaženo kombinací více spolupracujících detektorů a různých detekčních technik. Vstupní data detekčního systému pak představují data o síťových tocích ve formátech *NetFlow* a *IPFIX*. To vede k dalšímu zvýšení efektivity. Podporován je také formát *pcap* včetně zachytávání paketů přímo na síťovém rozhraní. Pro testování a možnost použití bez dalších závislostí byla vytvořena samostatná verze programu. Alternativou je druhá verze programu určená pro systém *Nemea*, který slouží k analýze a detekci síťových anomálií a je nasazen v reálném provozu. Výsledný program se soustředí na detekci tunelování přes DNS, ale po změně nastavení je možné jej využít i pro detekci jiných typů útoků.

Práce je členěna do několika kapitol. Ve druhé kapitole je uveden popis systému DNS a jeho součástí, způsobu komunikace a celkové architektury. Ve třetí kapitole jsou popsány útoky a anomálie spojené se systémem DNS, a také způsoby jejich detekce. Čtvrtá kapitola se věnuje zachytávání a získávání síťových dat včetně krátkého popisu formátů *pcap, NetFlow* a *IPFIX*. Pátá kapitola probírá návrh detekčního systému, jeho jednotlivé komponenty a jejich komunikaci. Šestá kapitola popisuje implementaci navrženého systému a řešení souvisejících problémů. V sedmé kapitole je pak provedeno ověření detekčních schopností a dalších aspektů implementovaného systému. V rámci *Semestrálního projektu* bylo zpracováno prvních pět kapitol této práce.

Kapitola 2

Systém doménových jmen

DNS (*Domain Name System* – systém doménových jmen) je jednou ze základních služeb Internetu. Primárním úkolem DNS je převod *doménových jmen* či *doménových adres* na *IP adresy*, nebo opačně. Díky DNS tak mohou uživatelé psát do svého prohlížeče např. www.seznam.cz (doménová adresa) místo 77.75.72.3 (IP adresa). Jak je patrné z tohoto příkladu, motivací pro vytvoření a zavedení systému DNS bylo zejména zjednodušení přístupu k jednotlivým serverům v Internetu (doménová jména jsou pro člověka snadněji zapamatovatelná než IP adresy) [41].

Překlad adres pomocí systému DNS využívá drtivá většina aplikačních protokolů jako HTTP či FTP, kdy po zadání doménového jména (např. serveru, na který se chceme pomocí FTP připojit) je pomocí DNS toto jméno přeloženo na IP adresu a až následně je zahájena další komunikace již s touto konkrétní IP adresou [41]. V současnosti je ale systém DNS využíván i ostatními internetovými službami jako elektronická pošta (např. protokol SMTP) či IP telefonie, které ho využívají i k jiným účelům než jenom k překladu adres (více v kapitole 2.1.5).

2.1 Architektura systému DNS

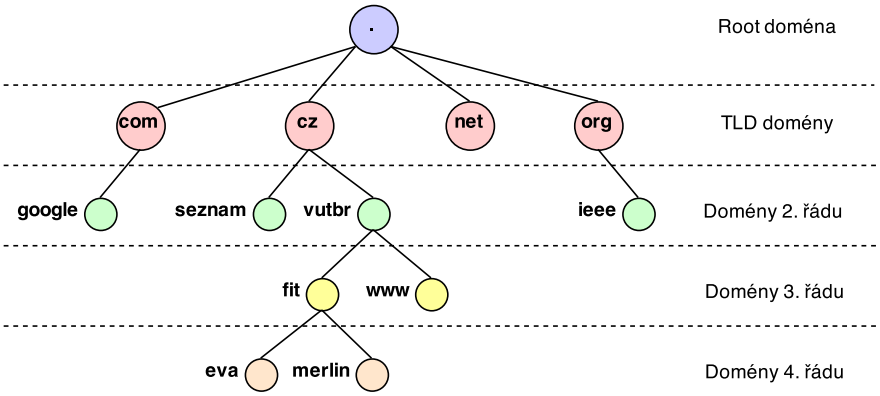
Teoreticky může služba DNS obsahovat mapování všech existujících IP adres na doménová jména a opačně. Prakticky není nutné znát toto mapování pro všechny IP adresy (není to nutné např. pro některé rezervované či privátní adresy, a ne všechny adresy musí mít přiřazeny doménové jméno), nicméně stále zůstává rozsáhlá databáze, ve které je nutno rychle vyhledávat požadovanou informaci.

Z tohoto důvodu je systém DNS tvořen celosvětovou distribuovanou databází, která je umístěna na DNS serverech, které mohou být také označovány jako *doménové servery* či *jmenné servery*. Ty pak obsahují jednotlivé části databáze, jejíž celkové uspořádání a struktura je definována *prostorem doménových jmen* [44].

2.1.1 Prostor doménových jmen

Databáze systému DNS je uspořádána do stromové struktury s jedním kořenem (obsahuje hrany a uzly – acyklický graf). *Doména* je pak podstrom ve stromu doménových jmen a *doménové jméno* je cesta mezi uzlem představujícím vrchol domény a kořenovým uzlem [44, 42].

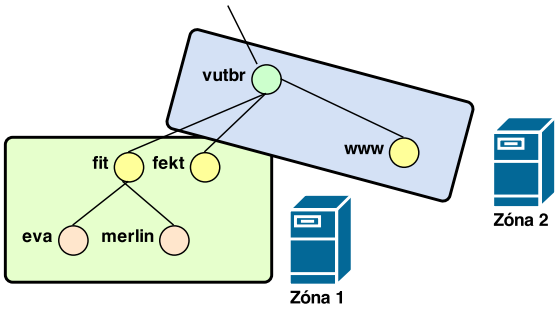
Jednotlivé uzly stromu jsou pojmenovány textovým řetězcem, který tvoří část výsledného doménového jména. Kořenový uzel celého stromu má prázdné jméno – řetězec délky



Obrázek 2.1: Příklad stromu doménových jmen.

nula („“). Domény, které se ve stromu nacházejí přímo pod kořenem (ve vzdálenosti 1 od kořene) jsou označovány jako *domény nejvyšší úrovně* (*TLD – Top Level Domain*). Ty jsou rozděleny buď tématicky (např. `com`, `net`) nebo dle státu (`cz`). Seznam TLD domén je dostupný v [5]. Domény (subdomény) druhé úrovně pak spadají pod TLD domény (např. `vutbr.cz.`), domény třetí úrovně jsou subdoménami domén druhé úrovně (např. `fit.vutbr.cz.`) atd. Jednotlivá koncová zařízení patřící do určité domény pak představují listové uzly stromu (např. server `www` v doméně `fit.vutbr.cz.`) [47]. Strom doménových jmen ilustruje obrázek 2.1.

Plné doménové jméno (*FQDN – Fully Qualified Domain Name*) je posloupnost jmen jednotlivých uzlů na cestě ke kořeni stromu, kdy jednotlivá jména uzlů jsou oddělena tečkou (např. `www.fit.vutbr.cz.`). Tečka na konci takového jména odděluje jméno kořenového uzlu nulové délky a v praxi ji ve většině případů nemusíme psát, protože ji za nás automaticky před překladem doplní aplikace [44, 47, 41].



Obrázek 2.2: Příklad rozdělení doménového stromu na zóny.

Abychom mohli celý prostor doménových jmen nějak rozdělit na jednotlivé fyzické servery, lze strom rozdělit do zón. *Zóna* je fyzická část prostoru doménových jmen pod jednotnou správou a nemusí být shodná s doménou. Více domén může být součástí jedné zóny,

ale naopak i jedna doména může zasahovat do více zón. Zóna je pak umístěna na jednom fyzickém serveru a poskytuje autoritativní informace o spravovaných doménách [44]. Příklad je uveden na obrázku 2.2. Zónu můžeme rozdělit na několik dalších částí (zón), jejichž správu můžeme delegovat na někoho jiného. Právě tímto je umožněna distribuovaná správa celého systému DNS, kdy jednotlivé organizace spravují své zóny [44, 55].

2.1.2 DNS servery

Zóny z prostoru doménových jmen jsou rozmístěny na jednotlivé DNS servery. Jejich úkolem je odpovídat na DNS dotazy. Pokud je dotaz směřován na data, za které je zodpovědný konkrétní server (např. určitá zóna či doména), pak tento server poskytuje *autoritativní odpovědi*. Naopak neautoritativní odpovědi mohou být neaktuální, nebo neúplné.

V systému DNS rozlišujeme následující typy serverů [41, 42]:

- **Primární server** (*master*) – poskytuje autoritativní odpovědi k dotazům na domény, které spravuje. V každé doméně existuje právě jeden primární server DNS.
- **Sekundární server** (*slave*) – je kopií primárního serveru. Data si průběžně aktualizuje z primárního serveru pomocí tzv. *přenosu zón*. Stejně jako primární server poskytuje i sekundární server autoritativní odpovědi.
- **Záložní server** (*caching-only*) – slouží pro snížení zátěže a zrychlení systému DNS. Funguje podobně jako *proxy servery* – ukládá si odpovědi na dotazy klientů, a ty pak po určitou dobu využívá. Díky tomu ale neposkytuje autoritativní odpovědi (nemusí mít za všech okolností nejaktuálnější data).
- **Kořenový server** (*root*) – je autoritativním serverem pro TLD domény a tedy říká, které servery jsou zodpovědné za konkrétní domény nejvyšší úrovně. Kořenové servery jsou naprosto nezbytnou součástí systému DNS, proto v současnosti existuje třináct těchto serverů (označeny písmeny A – M) rozmístěných po celém světě. Na kořenové servery jsou kladeny zvláštní nároky, které jsou popsány v RFC 2870 [17].

2.1.3 Rezoluce dotazu

Pro vyhledávání informací v systému DNS slouží program zvaný *resolver*. Ten zajišťuje komunikaci s DNS servery – posílá dotazy a interpretuje odpovědi na tyto dotazy [44]. Většinou je resolver implementován v rámci operačního systému (OS) a je možné jej následně využívat prostřednictvím specializovaných programů (např. `dig`, `nslookup`) nebo pomocí metod při programování vlastního programu. Resolver komunikuje s lokálním serverem DNS[1], který představuje nejbližší bod systému DNS.

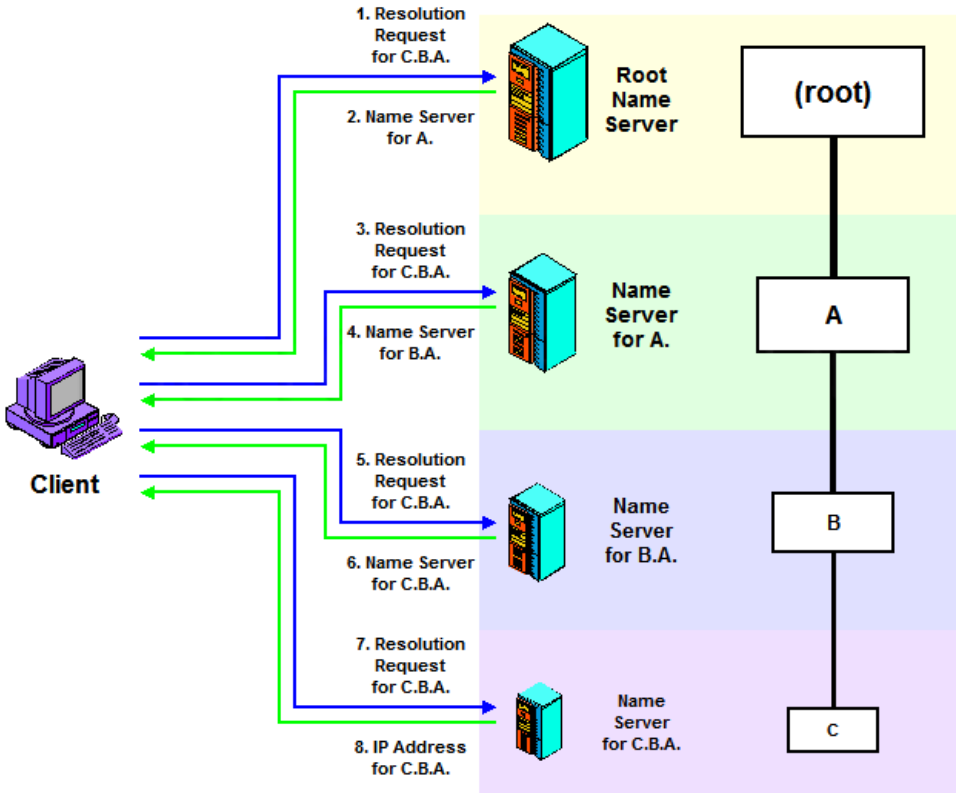
Samotná rezoluce dotazu obnáší průchod stromem doménových jmen, a to od kořene směrem k listům. Při překladu doménového jména tedy postupujeme v řetězci nesoucím dané jméno odzadu dopředu [42]. Například při překladu `www.fit.vutbr.cz` na IP adresu se dotážeme jednoho z kořenových serverů na adresu serveru, který spravuje TLD doménu `cz`. Tohoto serveru se poté dotážeme na adresu serveru spravujícího doménu `vutbr`. S jeho pomocí získáme adresu autoritativního serveru pro jeho subdoménu `fit`. Toho se poté již

[1] Adresa lokálního DNS serveru je nakonfigurována v síťovém subsystému operačního systému – manuálně, nebo pomocí protokolu DHCP.

můžeme zeptat na adresu koncového zařízení (serveru) se jménem `www`. Tento autoritativní server pro doménu `fit.vutbr.cz` by nám poté měl poslat odpověď s hledanou IP adresou. V systému DNS existují dva základní typy dotazů [42, 40]:

- **Rekurzivní dotaz** – server se postará o kompletní vyřízení dotazu. Pokud on sám nezná odpověď, postupně se dotazuje ostatních serverů, dokud odpověď nezíská (viz obrázek 2.4). Server tedy obstará celý průběh překladu, což zvyšuje jeho zátěž. Tento způsob se obvykle používá u lokálních DNS serverů.
- **Iterativní dotaz** – server vrátí nejlepší možnou odpověď v rámci svých znalostí. Pokud nezná přímo konkrétní odpověď, vrátí adresy serverů, které by mohly dané informace mít. Tazatel se poté musí dále ptát těchto serverů (viz obrázek 2.3). Tento způsob je kvůli snížené zátěži využíván zejména u kořenových serverů a obecně u serverů blízko kořenu stromu doménových jmen.

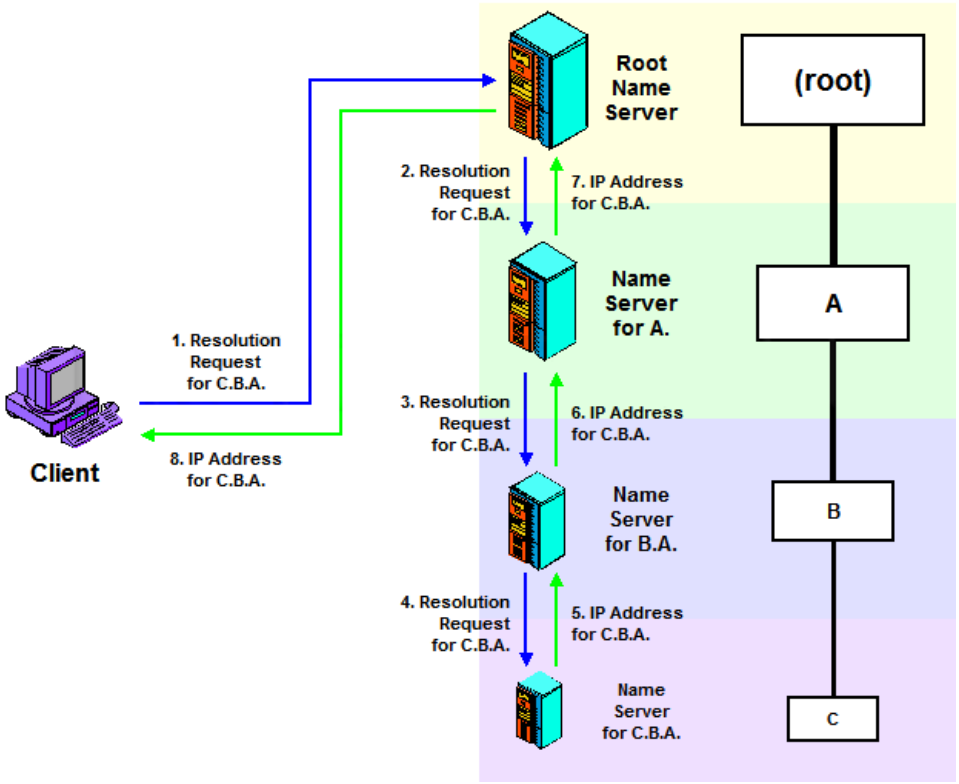
Který typ dotazů server podporuje je určeno nastavením daného serveru.



Obrázek 2.3: Ukázka iterativního zpracování dotazu. Obrázek převzat a upraven z [40].

V praxi je proces rezoluce dotazu optimalizován s ohledem na snížení zátěže na systém DNS, takže se využívá dočasné ukládání odpovědí na naše dotazy do *cache*[2], ze které pak tyto odpovědi mohou být po určitou dobu využity bez nutnosti opětovného překladu. Cache může být umístěna jak na lokálním serveru DNS, tak i v rámci resolveru v OS.

[2]Vyrovnávací paměť sloužící pro uložení dočasných záznamů.



Obrázek 2.4: Ukázka rekurzivního zpracování dotazu. Obrázek převzat a upraven z [40].

2.1.4 Reverzní mapování adres

V některých případech je nutné umožnit *zpětné (reverzní) mapování* jmenných adres na IP adresy, tedy pro danou IP adresu najít její doménové jméno. Problémem v tomto případě je uspořádání stromu doménových jmen, kdy vyhledávání probíhá od nejobecnějších informací (TLD domény), které jsou uvedeny na konci doménového jména, k těm více specifickým (konkrétní subdomény), které jsou uvedeny na začátku doménového jména. V případě IP adresy jsou ovšem nejobecnější informace uvedeny na začátku adresy (adresa sítě), zatímco specifičtější informace na konci. Tedy přesně opačně. Abychom při vyhledávání nemuseli prohledávat úplně celý strom doménových jmen (a porovnávat všechny záznamy s hledanou IP adresou), což by byla z hlediska náročnosti nerealizovatelná operace, má reverzní mapování adres zvláštní pravidla.

Při vyhledávání tedy nejprve musíme otočit hledanou IP adresu (z adresy `123.214.25.16` uděláme adresu `16.25.214.123`) a tu poté hledat ve speciální doméně `in-addr.arpa.`, ve které jsou adresy hierarchicky uloženy po bytech [44]. Budeme tedy hledat `16.25.214.123.in-addr.arpa.`, a to standardním způsobem. Důvodem k tomuto je, aby bylo stejně jako u domén možno delegovat správu jednotlivých sítí jejich jednotlivým správcům (například budeme spravovat síť `123.214.25.0/24`, a tedy i doménu pro reverzní překlad `25.214.123.in-addr.arpa.`, do které přidáme mapování jednotlivých koncových zařízení).

Reverzní mapování může být problematické z pohledu korektnosti získaných údajů, protože hledané adrese můžeme přiřadit libovolné doménové jméno. Z tohoto důvodu je vhodné ještě následně ověřit, že získané doménové jméno je skutečně na dané adrese (dotazem na

překlad získaného jména na IP adresu a porovnáním s původně hledanou adresou).

Reverzního mapování můžeme využít například při detekci neautorizovaných zařízení připojených do sítě, kdy můžeme ověřit, jestli má daná IP adresa v DNS reverzní záznam (pokud nemá, jedná se o neautorizované zařízení) [42].

2.1.5 DNS záznamy

Jak již bylo naznačeno, systém DNS může pro jednotlivé domény obsahovat různé druhy informací. Tyto informace jsou organizovány do tzv. *záznamů*. Každý záznam má svůj typ (čeho se záznam týká), jméno (jméno uzlu, ve kterém je záznam uložen) a data (jaká je jeho hodnota). Každý záznam dále obsahuje i jiné parametry, jako např. TTL (maximální doba platnosti záznamu – jestli smí být záznam uložen v cache, a případně jak dlouho), délku dat a třídu záznamu [43, 55, 42].
Mezi nejčastější typy záznamů dle [18, 42] patří:

A (*Address record*) obsahuje mapování doménového jména na IPv4 adresu.

AAAA (*IPv6 address record*) obsahuje mapování doménového jména na IPv6 adresu.

PTR (*Pointer record*) je určen pro reverzní překlad (převod IP adresy na doménové jméno). Obsahuje adresu v reverzní podobě, viz kapitola 2.1.4.

SOA (*Start Of Authority*) obsahuje jméno primárního DNS serveru pro danou doménu. Každá zóna má pouze jeden záznam SOA.

NS (*Name Server*) obsahuje jméno autoritativního serveru pro danou doménu. Pokud je v doméně více autoritativních serverů, můžeme určit primární server pomocí záznamu SOA.

MX (*Mail Exchanger*) obsahuje jméno poštovního serveru pro danou doménu a jeho prioritu (který server budeme preferovat v případě více poštovních serverů v doméně).

CNAME (*Canonical Name*) umožňuje na jeden počítač mapovat více jmen (aliasů) – např. serveru `serv.domena.cz` přiřadíme alias `www`, což způsobí, že při překladu se doménové jméno `www.domena.cz` bude mapovat na `serv.domena.cz`.

SRV (*Service record*) poskytuje dodatečné informace o službách. Používá se pro nalezení serveru, který poskytuje určitou službu (používá se např. v *IP telefonii* pro vyhledání *SIP serveru*). Také lze využít pro rozložení zátěže mezi více serverů či zajištění redundance (pomocí priorit a vah).

TXT (*Text record*) slouží pro uložení textových záznamů. Obvykle obsahuje bližší informace o serveru, případně organizaci, která jej spravuje.

2.1.6 Protokol DNS

Pro komunikaci resolveru s DNS serverem se používá protokol DNS (definován v [43]). Jedná se o jednoduchý textový protokol využívající systém dotaz – odpověď. Na každý položený dotaz by tedy měla existovat odpověď (této skutečnosti můžeme využít např. u *stavových firewallů*, kdy můžeme blokovat odpovědi na neexistující dotazy).

Komunikace standardně probíhá na serverovém portu 53 a pro zajištění doručení zpráv se využívá transportních protokolů TCP a UDP. Standardně se používá UDP, nicméně

pro odpovědi větší než 512 bytů se použije TCP [43]. Při použití UDP musíme řešit určité komplikace v podobě možné ztráty *paketu*[3] nebo příchodu paketů v nesprávném pořadí (využívá se identifikační číslo v hlavičce paketu).

2.2 Registrace domén

Zodpovědnost za přidělování a správu doménových jmen má organizace ICANN (*Internet Corporation for Assigned Names and Numbers*). Ta určuje jednotlivé správce, kteří mají na starost příslušné TLD domény (např. pro doménu `.cz` je to sdružení CZ.NIC [2]). Ti mohou následně provádět registraci nových domén nebo ji umožnit přes pověřené registrátory [42]. Seznam takových registrátorů pro Českou republiku najdete v [3].

Pro zjišťování informací o registrovaných doménách lze použít databázi *WHOIS*[4], kterou vedou jednotliví registrátoři. Pro přístup do databáze je možné použít nástroj `whois`[5] nebo webové rozhraní jednotlivých registrátorů – např. [4] nebo [6].

[3] Datová zpráva putující sítí.

[4] Pro přístup k databázi je využíván stejnojmenný protokol, který je definován v RFC 3912 (`http://tools.ietf.org/html/rfc3912`).

[5] Standardně dostupný v operačních systémech založených na systému Unix, ale jsou k dispozici i implementace pro systém Windows.

Kapitola 3

Anomálie a útoky na DNS

Se zvyšujícím se počtem útoků na Internetu v posledních letech můžeme pozorovat přibývající počet typů útoků, které se čím dál více soustřeďují i na základní služby Internetu. Protože je DNS jednou z klíčových internetových služeb (na její funkčnosti je závislá řada aplikací), nevyhýbají se nové typy útoků ani systému DNS. Naopak, v poslední době získávají útoky na DNS na popularitě [49].

Protože je komunikace s DNS stěžejní pro většinu aplikací, je pro útoky výhodné využívat právě službu DNS, neboť ji není možné jednoduše zakázat jako celek nebo pomocí jednoduchých filtrovacích pravidel filtrovat všechny hrozby. Stejně tak je důležité zajistit funkčnost a dostupnost DNS jako celku, protože výpadek může postihnout i většinu závislých aplikací.

Útokům na DNS nahrává fakt, že DNS je otevřený, decentralizovaný systém, což útočníkům umožňuje stát se jeho součástí (např. nasazením podvodných DNS serverů), nebo ovlivnit data v něm uložená. Svou roli hraje i absence jakéhokoliv šifrování či zajištění integrity dat[1] (odpovědí na dotazy), což útočníkovi umožňuje provést *man in the middle útok*, kdy útočník zachytí odpověď od serveru, pozmění ji a následně odešle původnímu příjemci.

Další možností podvržení odpovědi je odeslání falešné odpovědi přímo útočníkem, k čemuž útočník potřebuje odhadnout nebo zjistit identifikační číslo DNS dotazu (celkem 65 536 možností[2]), IP adresu a port odesílatele dotazu.

Jinou slabinou může být i *cachování* odpovědí, protože pokud změníme data v cache (*cache poisoning*, více v kapitole 3.1.2), server po určitou dobu považuje daná data za správná a bez ověření je posílá v odpovědích na dotazy. Řešením této slabiny je nasazení systému podepisování jednotlivých záznamů – DNSSEC[3].

Útoky můžeme dělit do dvou kategorií podle cíle útoku [50]:

- **Útoky cílící na službu DNS samotnou** – cílem útoku jsou samotné DNS servery nebo služba jako taková.
- **Útoky využívající DNS k útokům na jiné systémy** – pomocí DNS provádíme útok na jiný systém či využijeme informací z DNS pro provedení jiného útoku.

V následujícím textu si rozebereme příklady jednotlivých útoků s jejich popisem a analýzou jejich nebezpečnosti. U útoků a anomálií, kterými se budeme v práci dále zabývat, zmíníme i možnosti jejich detekce.

[1] Zajištění, že data nebudou během přenosu změněna.
[2] ID paketu je šestnáctibitové, tedy $2^{16} = 65\,536$ možností.
[3] Zabezpečení přenosu DNS dat pomocí asymetrické kryptografie. Více viz [42].

3.1 Útoky cílené na DNS

U této kategorie útoků je jejich cílem přímo služba DNS samotná, potažmo jednotlivé DNS servery. Mezi základní typy těchto útoků patří *útoky pro odepření služby* (DoS) nebo útoky zneužívající zranitelnosti v software DNS serveru, jako například útok typu *buffer overflow* (využívá implementačních chyb pro neoprávněný přístup do paměti DNS serveru), nebo *exploity* (postupy pro vykonání původně nezamýšlené činnosti softwaru) známé pro danou implementaci DNS serveru. Informace obsažené v této kapitole čerpají především z [50, 38] a [51].

3.1.1 DoS útoky

Cílem útoku DoS (*Denial of Service*) je odepření dostupnosti služby, v tomto případě DNS serveru. Útok DoS je realizován nadměrným počtem dotazů a spojení, které vyčerpají zdroje serveru a způsobí citelné zpomalení jeho činnosti, nebo v horším případě úplné zahlcení. Pro úspěšné provedení útoku je třeba poměrně velký počet dotazů nebo spojení, k čemuž se v praxi využívá *botnetů*[4] (v tomto případě mluvíme o distribuovaném útoku DoS – *DDoS*), nebo tzv. *amplification* útoku. Jeho podstatou je vygenerování velkého datového toku, který má za následek zahlcení cílového serveru. Více o tomto typu útoku naleznete v kapitole 3.2.1.

DoS útoky jsou jedním z nejběžnějších typů útoků[5] (i kvůli své jednoduchosti) a obvykle jsou součástí jiných, složitějších útoků (viz útok cache poisoning níže).

V souvislosti se systémem DNS rozlišujeme následující typy DoS útoků [50]:

- **DNS flood útok** – útočník odesílá velké množství dotazů na cílový DNS server za účelem vyčerpání jeho zdrojů.
- **Rekurzivní útok** – cílí na servery umožňující rekurzivní zpracování dotazů. Útočník posílá množství různých dotazů, jejichž odpovědi nemá cílový server uloženy v cache. Cílový server se na ně tedy rekurzivně dotazuje, což způsobí jeho zahlcení.
- **Reflexivní útok** – využívá DNS servery a jejich odpovědi pro útok na stanovený cíl. Více o tomto útoku se dozvíte v kapitole 3.2.1.
- **Garbage útok** – zahlcení DNS serveru dosahuje pomocí zahlcení DNS *parseru*[6] nadměrnými pakety nebo pakety s nesmyslným obsahem.

Možným řešením pro omezení DoS útoku je konfigurace DNS serveru pro omezení počtu dotazů z jedné IP adresy, nebo limit počtu spojení pro celý server. Omezení škod způsobených DoS útoky je pak možné pomocí vhodné redundance DNS serverů.

3.1.2 DNS *cache poisoning*

Účelem cache poisoning útoku („otrávení cache“) je podvržení dat v cache DNS serveru. Výsledkem může být podvržení IP adresy a následné přesměrování dotazů na server útočníka. Toho se využívá zejména při *phishingu* (technika pro získání citlivých údajů uživatele pomocí falešných webových stránek imitujících známé služby).

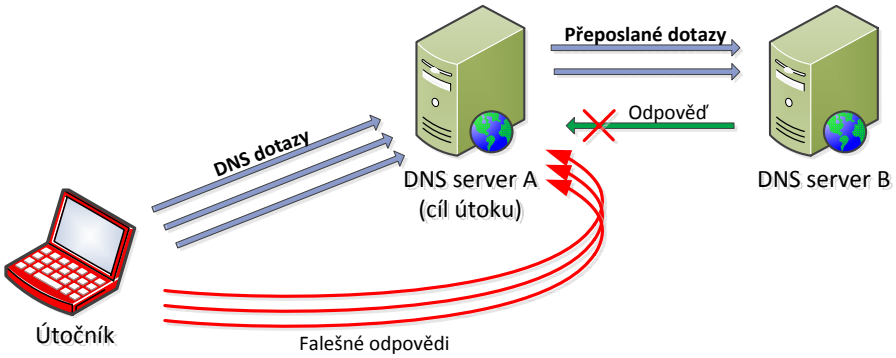
[4] Skupina počítačů ovládaná útočníkem bez vědomí jejich uživatelů (obvykle jsou jednotlivé počítače napadeny *malware*, který způsobí jejich začlenění do botnetu), která současně útočí na vybraný cíl.

[5] Dle bezpečnostní zprávy společnosti Radware [49] je DDoS dokonce nejčastější typ útoku.

[6] Program, který z jednotlivých zpráv protokolu DNS extrahuje data užitečná pro další zpracování.

Útok probíhá následovně: Pokud se při překladu resolver dotazuje autoritativního serveru nějaké domény, zareaguje útočník a odešle falešnou odpověď dříve než daný autoritativní server (útočník potřebuje znát adresu tohoto serveru, aby ji mohl podvrhnout ve falešné odpovědi). Následně bude daná falešná odpověď uložena v cache a tedy i používána v následujících odpovědích [36].

Prakticky útočník zašle dotaz na doménové jméno, které chce podvrhnout. Ihned poté začne danému serveru posílat falešné odpovědi obsahující adresu dosazenou útočníkem. Situaci ilustruje obrázek 3.1. Pro zajištění toho, že útočníkovy odpovědi budou přijaty dříve než pravé odpovědi od autoritativního serveru dané domény, je možné na tento autoritativní server současně provést DoS útok a tím zpomalit jeho činnost [50].



Obrázek 3.1: Schéma útoku DNS cache poisoning.

Pro úspěšné provedení útoku musí útočník ještě „uhádnout“ ID dotazu (QID – *Query ID*), pomocí kterého je prováděna jednoduchá autentizace DNS paketů. QID je možné uhádnout například útokem hrubou silou (zkoušením – celkem 65 536 možností).

Obranou proti cache poisoning útoku je podepisování a ověřování záznamů pomocí DNSSEC, což útočníkovi znemožní podvržení odpovědi. Detekce útoku pomocí flow dat je popsána v [36]. Je založena na detekci opakovaných odpovědí během určitého časového okna od dotazu.

3.1.3 *Fast flux* domény

Jako *fast flux* domény jsou označovány škodlivé domény, u nichž se rychle mění příslušející DNS záznamy. Obvykle se jedná o domény, které jsou využívány jako řídicí kanál pro komunikaci s malware nebo pro phishing. Aby nemohly být jednoduše zablokovány, mění se u těchto domén neustále IP adresa (je nutné zablokovat danou doménu, blokování IP adres nestačí). S tím souvisí i nízké *TTL*[7], které záznamy těchto domén většinou mají [59, 38].

Jednoduchá, ale nepřesná detekce těchto domén je možná na základě nízkého TTL. Jinou možností je zaznamenávání počtu různých IP adres pro dané FQDN a po překročení stanoveného prahu jeho blokace. Obecně ale detekci není možné provádět pouze s NetFlow daty (neobsahují položky jako TTL a informaci o překládané doméně) [59].

[7] *Time To Live* – doba platnosti daného záznamu v DNS.

3.2 Útoky necílené přímo na DNS

Do této kategorie patří útoky a anomálie, které využívají vlastností a možností služby DNS pro vykonání útoku na jiné služby, a nebo pro získání důvěrných informací uživatele. Další možností je využívání služby DNS v rozporu s jejím původním účelem (např. pro tunelování) [50]. Následující informace čerpají především z [38] a [37].

Příkladem anomálií zneužívajících DNS může být *cybersquatting* (případně označován jako *domain squatting*), což je úmyslná registrace, nebo využití doménových jmen za účelem vlastního obohacení či poškození druhé strany. Typickým příkladem je registrace domén s atraktivními nebo jinak strategickými názvy za účelem jejich následného prodeje za mnohem vyšší cenu. Útočníci zde využívají například exspirace domén, kdy je po určitém čase nutné domény znovu registrovat. Pokud tak včas neučiní původní majitel domény, mohou si ji útočníci snadno registrovat na sebe (tato technika se také označuje jako *domain sniping*).

Další možností je tzv. *typosquatting*, kdy si útočníci registrují domény, které vzniknou špatným zadáním (např. překlep nebo podobnost písmen) nějaké jiné známé domény (např. místo `www.google.com` zadáme `www.gogle.com`). Takové domény se obvykle pokouší uživatele zneužít například k phishingu či instalaci malware [59, 12]. Obranou proti těmto doménám je jejich blokování na základě tzv. *blacklistu*[8].

Jinou variantou zneužití DNS je *domain name hijacking*, kdy je celá doména ukradena původnímu majiteli (například krádeží identity a změnou údajů u registrátora domény). Ukradené domény poté bývají využívány k phishingu.

3.2.1 DNS reflexivní útok

Cílem reflexivního DNS útoku je provedení (D)DoS útoku za použití DNS serverů. V tomto případě nejsou DNS servery cílem útoku, ale naopak se podílejí na jeho realizaci.

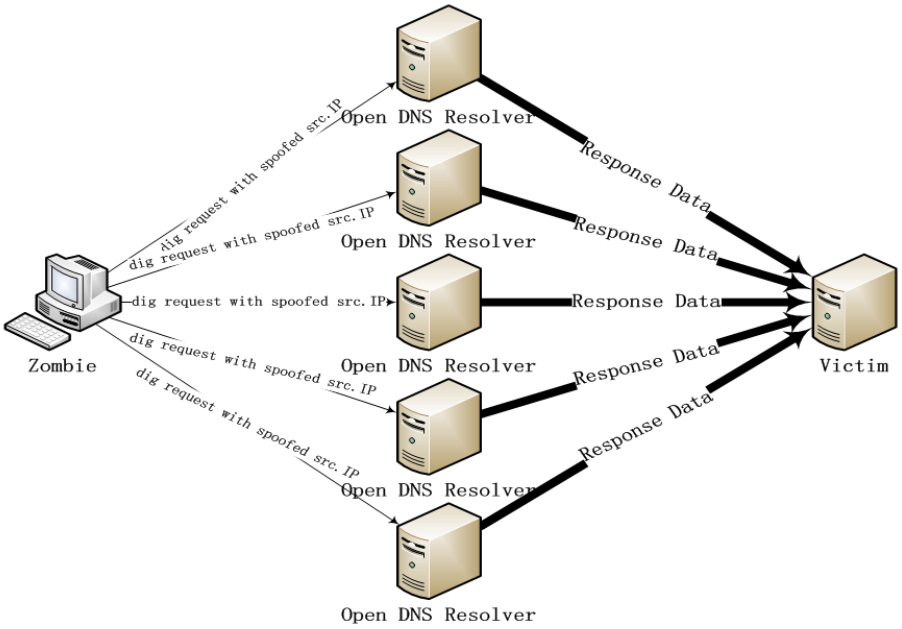
Principem útoku je zasílání odpovědí na DNS dotazy na určitou cílovou adresu, kterou nápor odpovědí zahltí. Cílová adresa útoku je podvržena jako zdrojová IP adresa v dotazech na DNS server (tzv. *spoofing*). Pokud je cílem útoku služba DNS na cílovém serveru, potom je třeba podvrhnout i zdrojový port. To je možné, protože v UDP paketech s dotazy není nijak zajištěna integrita dat. Útočník tedy vysílá množství takto podvržených dotazů na DNS servery, které své odpovědi směřují na stanovený cíl (proto „reflexivní útok“), u kterého způsobí zahlcení. To ilustruje obrázek 3.2. Popis a ukázky útoku lze nalézt v [57, 51].

Důvodem, proč se k reflexivnímu útoku využívá právě DNS, je schopnost tzv. *amplifikace*, tedy asymetrické velikosti dotazu a odpovědi, kdy odpověď může být několikanásobně větší než dotaz samotný. To je výhodné právě pro DoS útoky, protože takto můžeme vygenerovat mnohem větší datový tok než by bylo možné v jiných případech[9].

V případě normálních odpovědí se obvykle amplifikační efekt pohybuje okolo třínásobku velikosti dotazu. Pro útočníka je ale výhodné předem si zjistit, který typ dotazu vrátí největší možnou odpověď a následně posílat dotazy daného typu. Obvykle se jedná o dotazy typu `ANY` (vrátí všechny záznamy pro danou doménu bez ohledu na typ záznamu), `TXT` či `RRSIG` a `DNSKEY` (záznamy sloužící pro podepisování a ověřování záznamů pomocí DNSSEC). Ty mohou přinést amplifikační efekt v desetinásobcích původní velikosti dotazu. Vyššího amplifikačního efektu lze dosáhnout se speciálně vytvořenými záznamy, které mohou mít až maximální velikost odpovědi (4096 bytů) [51]. Ty ale vyžadují přístup útočníka na daný DNS server tak, aby je na něj mohl útočník umístit.

[8] Seznam zákeřných domén, které budou blokovány.

[9] Dalším příkladem protokolu, který se zneužívá pro amplifikační DoS útoky je protokol NTP.



Obrázek 3.2: Schéma reflexivního DNS útoku. Obrázek převzat z [45].

Pro znesnadnění detekce útoku na straně DNS serveru je vhodné kombinovat různé typy dotazů tak, aby nebylo možné jednoduše detekovat dlouhou posloupnost totožných dotazů.

Reflexivní útok je velmi populární, protože je velmi jednoduchý na provedení (pro realizaci lze například využít veřejné DNS servery umístěné na Internetu), a zároveň díky podvržení zdrojových IP adres v paketech dotazů není možné jednoduše vypátrat útočníka, protože cíl útoku vidí, že pakety přicházejí z DNS serverů. Sofistikovanější útoky využívají pro pokládání dotazů botnet, což dále znemožňuje odhalení identity útočníka a snadné zablokování útoku.

Obranou proti DNS reflexivnímu útoku může být *firewall*[10] (např. zakážeme `ANY` dotazy), který ale útok pouze zmírní. Jiným využitím firewallu může být kontrola zdrojových adres v paketech na výstupu z přidělené sítě. Tím můžeme zabránit odesílání podvržených dotazů z dané sítě.

Dalším způsobem obrany je *DNS dampering*, kdy DNS server přiděluje jednotlivým IP adresám, které mu zasílají požadavky, „trestné body“. Body jsou přidělovány na základě typu dotazu nebo velikosti odpovědi. Jakmile počet bodů překročí stanovenou mez, server na určitou dobu (body postupem času ubývají) přestane reagovat na dotazy od dané IP adresy a zahazuje je. Problémem tohoto způsoby ochrany je, že není možné zabránit *falešně pozitivním* chybám, kdy jsou zablokováni i legitimní uživatelé [51].

Jinou možností je metoda *RRL*[11], která limituje počet unikátních odpovědí DNS serveru pro určitou skupinu IP adres (určitou podsíť, např. /24). Po překročení stanoveného prahu server přestane odpovídat na dotazy této skupiny adres. Tato metoda může značně omezit rozsah útoku, nicméně útočník může stále rozložit útok mezi více serverů a použít různé typy dotazů, takže část útoku se stejně uskuteční [51].

[10] Provádí kontrolu obsahu paketů a na základě množiny pravidel je povolí ke zpracování nebo ne.
[11] *Response Rate Limiting*

Detekci reflexivního útoku je možno provádět několika způsoby. Základní metody detekce popisuje práce [50]. Detekovat lze nárůst počtu dotazů od konkrétní IP adresy či odpovědí na konkrétní adresu. Také odchylka od typického rozložení velikosti DNS paketů může signalizovat probíhající útok. K takové detekci stačí pouze obyčejná NetFlow data.

V článku [35] je navržen detekční mechanizmus mapující DNS dotazy a odpovědi pomocí databáze. Do databáze jsou ukládány počty dotazů a odpovědí pro jednotlivé IP adresy. Jakmile je překročen práh rozdílu počtu dotazů a odpovědí, je daná adresa označena jako podezřelá a následně případně zablokována úpravou pravidel firewallu. Problémem tohoto řešení je nízká rychlost a propustnost (kvůli práci s databází).

Detekcí přímo s využitím NetFlow dat se zabývá článek [33]. Navrhuje algoritmy pro detekci IP adres odesílajících a přijímajících podezřelé dotazy. Algoritmy jsou založeny na prahování dle počtu paketů v toku nebo průměrné velikosti paketů. Autor tvrdí, že algoritmus dosahuje úspěšnosti detekce až 93 %.

Další metodu detekce pak popisují články [26] a [52], kde autoři navrhují pro detekci využít *Bloomových filtrů*[12]. Filtry v tomto případě slouží pro uložení informace o příchozích dotazech a její následné ověření při odesílání odpovědí. Pro dosažení rozumných rychlostí je nicméně vhodné toto řešení implementovat v hardwaru.

3.2.2 Tunelování přes DNS

DNS tunneling je zástupcem anomálií, které využívají službu DNS původně nezamýšleným způsobem. Podstatou tunelování přes DNS je zapouzdření a přenos dat pomocí DNS (DNS zde hraje roli tunelovacího protokolu). Pomocí DNS dotazů a odpovědí jsou data přenášena mezi klientem a *tunelovacím serverem DNS*, který je umístěn mimo lokální síť klienta a umožňuje překlad mezi tunelovanými a normálními daty (princip podobný jako u VPN[13]). Tuto situaci ilustruje obrázek 3.3.

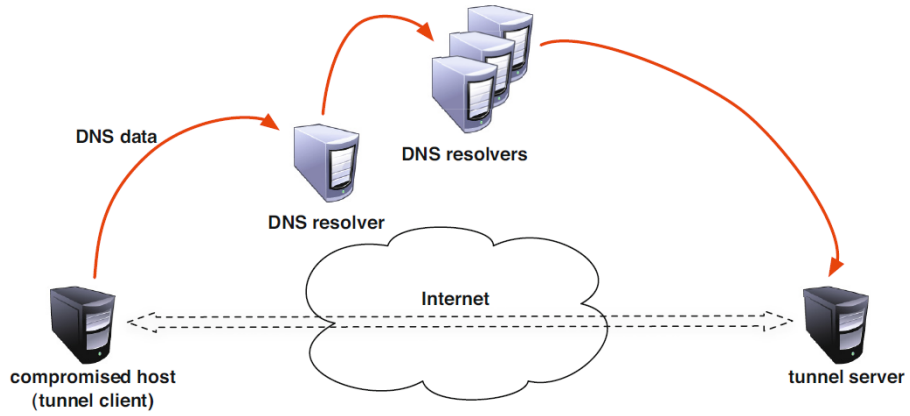
Prakticky tunelování probíhá tak, že klient zapouzdří data do dotazu směřujícím na doménu, pro kterou je tunelovací DNS server autoritativním serverem (například doména `xy.com`). Data jsou zakódována (například *Base32* či *Base64* kódováním) a zapouzdřena do dotazu (např. do dotazu typu A vložíme zakódovaná data jako subdoménu `ORUGS43JO-N2GK43U.xy.com`). Následně je proveden klasický překlad (přes normální DNS servery), při kterém se data dostanou až k autoritativnímu serveru `xy.com`. Autoritativní server (který je zároveň tunelovacím serverem) data dekóduje a interpretuje [50]. Poté může klientovi poslat jiná data například ve formě CNAME odpovědi s hodnotou `ONSXE5TFOJZGK43Q.xy.com`. Prakticky se pro tunelování využívají různé typy záznamů. Například záznamy TXT, v nichž mohou být data zakódována pomocí Base64 kódování [30].

Protože je pro přenos normálních dat využito pouze protokolu DNS, je většinou DNS tunelování zneužíváno pro neoprávněné připojení k Internetu v případech, kdy je zpoplatněno, nebo k obcházení bezpečnostních prvků a skryté připojení ze zabezpečené sítě. Jelikož ve většině případů musí být firewall nakonfigurován k propouštění DNS paketů, můžeme pomocí DNS tunelování uskutečnit komunikaci, kterou by jinak firewall zablokoval. Toho útočníci využívají například u sofistikovaného malware, který komunikuje se servery útočníka skrytým kanálem (*command and control*), právě pomocí DNS tunelování [29].

Nevýhodou DNS tunelování je menší přenosová rychlost a nutnost kódování dat alfanumericky, což zvyšuje objem přenášených dat.

V současnosti existuje několik volně dostupných nástrojů pro tunelování přes DNS. Liší se zejména použitými typy záznamů či podporovanou platformou. Následuje přehled [30]:

[12] Datová struktura, která umožňuje prostorově efektivní uložení množiny elementů za účelem dotazování



Obrázek 3.3: Obecné schéma tunelování přes DNS. Obrázek převzat z [29].

- `dns2tcp` – používá TXT a KEY záznamy. Platformy: Linux, Windows.
- `DNScat` – používá A, AAAA, CNAME, NS, TXT, MX záznamy. Platforma: Unix.
- `iodine` – využívá TUN/TAP rozhraní[14], široké spektrum podporovaných platforem.
- `OzymanDNS` – SSH tunel přes DNS, používá TXT záznamy.

Detekci DNS tunelování lze založit především na výskytu kódovaných dat v DNS paketech. Zakódovaná data se nepodobají „obyčejnému“ obsahu DNS paketů a lze je tedy detekovat. Usnadnění detekce napomáhá také to, že tunely většinou pracují s jednou doménou a mění pouze subdomény. Rozdělení dat podle domén tak může usnadnit detekci. Článek [14] popisuje detekční metodu založenou na frekvenční analýze znaků obsažených v DNS paketech. Metoda porovnává rozložení četnosti jednotlivých znaků v DNS paketu oproti rozložení četnosti znaků anglického jazyka.

Dalšími variantami detekce je detekce založená na vyhledávání bigramů [48] či n-gramů [15]. Principem těchto metod je vytvoření seznamu nejfrekventovanějších n-gramů pro normální data a následné porovnávání se seznamem četnosti n-gramů u dat, nad nimiž probíhá detekce. Tunelovaná data mají v porovnání s normálními daty menší výkyvy ve frekvencích jednotlivých n-gramů v seznamu.

Jiné možnosti detekce jsou popsány v [30]. Autor popisuje metody založené na velikosti dotazů a odpovědí (při tunelování se využívají co nejdelší názvy subdomén), dále detekci na základě entropie (tunelované domény mají vyšší entropii než normální domény), ale i detekci na základě vyhledávání známých vzorů v zachycených datech (specifické vzory pro jednotlivé tunelovací nástroje).

V článku [29] navrhli autoři několik detektorů, které pracují přímo s flow daty. Mezi detekční metody patří prahování na základě velikosti paketů (i časově závislé). V dalším článku [36] autoři využili pro detekci DNS tunelů pomocí flow dat metodu založenou na entropii.

---

na příslušnost v dané množině.

[13] *Virtual Private Network* – umožňuje připojení do zabezpečené privátní sítě přes veřejnou síť (např. Internet).

[14] Virtuální síťová rozhraní v operačním systému, která umožňují tunelování pro konkrétní aplikaci.

Kapitola 4

Formáty síťových dat

Abychom mohli provádět detekci síťových anomálií, musíme nějakým způsobem efektivně pracovat se síťovými daty. První možností přístupu k síťovým datům je přístup přímo na úrovni fyzických síťových zařízení, kdy bychom instalovali detekční mechanizmy přímo do jejich hardwaru nebo operačního systému. Protože tato varianta pro nás není reálně možná, obvykle musíme síťová data nějakým způsobem ukládat a následně provádět detekci nad těmito daty.

Možností pro ukládání síťových dat máme několik. Asi nejjednodušší možností je ukládat celé pakety tak, jak putují sítí. Toto řešení má výhodu v tom, že neztratíme nic z užitečných dat, takže můžeme provádět libovolně obsáhlou analýzu zachycených paketů. Velkou nevýhodou tohoto řešení jsou ale paměťové nároky. Pokud bychom chtěli tímto způsobem ukládat a analyzovat data z nějaké vytížené linky, museli bychom pracovat s desítkami až stovkami gigabytů dat, které by byly zachyceny během krátké doby. Zástupcem tohoto řešení je formát *pcap*.

Jinou, prakticky mnohem využívanější možností jsou data o síťových tocích. *Síťový tok*[1] je definován jako množina paketů procházejících bodem pozorování za určitou dobu a mající společné vlastnosti (existuje množina atributů, jejichž hodnoty mají všechny pakety patřící do jednoho toku stejné) [22]. Prakticky jsou atributy tvořeny položkami IP paketu (například jeden tok tvoří pakety mající stejnou zdrojovou adresu a port). Konec toku potom detekujeme například zachycením příznaků RST nebo FIN (u TCP), anebo vypršením časovače (doba od přijetí posledního paketu).

Jeden tok tedy může tvořit jeden, ale i několik paketů. Tím ušetříme značné množství paměti, protože data o více paketech jsou obsaženy v jednom záznamu. Dalším faktorem je to, že u síťových toků obvykle ukládáme pouze omezené množství informací z každého paketu, což na jednu stranu dále snižuje paměťové nároky, ale na druhou stranu přicházíme o podrobnější obsah paketů. Zástupci tohoto způsobu práce se síťovými daty jsou protokoly *NetFlow*, *IPFIX* a jejich další varianty.

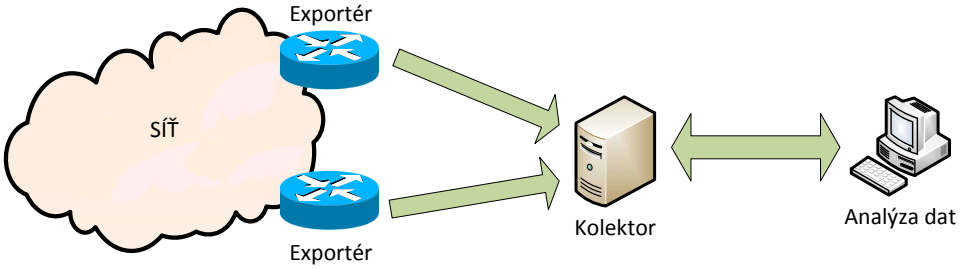
Z výše uvedeného vyplývá, že pro účely detekce síťových anomálií by bylo vhodnější použít první způsob (ukládání celých paketů), ale z praktického hlediska je nutné využít druhý způsob, tedy detekci provádět nad *flow* daty (zejména z důvodu paměťových nároků, ale i rychlosti zpracování). U některých typů anomálií jako (D)DoS útoky navíc většinou postačují obecné informace z hlavičky paketu, a bylo by tak zbytečné ukládat celý jejich obsah.

[1] Také označován jako *IP flow* či *flow*.

4.1 Zachytávání síťových toků

Pro zachycení dat o síťových tocích musíme do sítě umístit dvě zařízení – *exportér* a *kolektor*.

Exportér je prvek, který vytváří záznamy o tocích a následně je odesílá kolektoru. Exportér je obvykle umístěn přímo na určitém směrovači (aktivací příslušné volby operačního systému), ale může se jednat i o samostatný prvek, kterému data posílají jednotlivé *sondy* umístěné v síti.



Obrázek 4.1: Komponenty sběru flow dat.

Kolektor sbírá data od jednotlivých exportérů a ukládá je. Komunikace mezi exportéry, sondami a kolektory probíhá pomocí příslušného protokolu, např. NetFlow. Obvykle jsou data od exportéru ke kolektoru zasílány pravidelně po určitých blocích, ale existují i výjimky, kdy si kolektor o data zažádá přímo. Kolektor obvykle pro účely monitorování sítě umožňuje vytváření souhrnných statistik a přehledů o zachycených datech, nebo jejich agregaci [20].

4.2 pcap

Pcap[2] představuje formát pro ukládání celých síťových paketů. Jeho výhodou je uložení kompletního obsahu paketů, tedy neztratí se žádná podstatná informace. Nevýhodou je ovšem velká paměťová náročnost a také menší rychlost zpracování dat. Obvykle je využíván nejpopulárnějšími programy pro zachytávání paketů, jako `Wireshark` a `tcpdump`. Další popis formátu lze nalézt v [9].

Tento formát je vhodný zejména pro případy, kdy potřebujeme provádět inspekci samotných DNS dat (či jiných dat vyšších protokolových vrstev) v jednotlivých paketech, tedy spíše pro detekci složitějších anomálií. Naopak pro detekci založenou pouze na znalosti komunikujících IP adres či portů není příliš vhodný, zejména s ohledem na paměťovou náročnost. Pro tyto případy je vhodnější využít formáty pracující s daty o síťových tocích.

4.3 NetFlow

Jedním ze široce používaných protokolů pro zachytávání a práci se síťovými toky je *NetFlow*. NetFlow je proprietárním protokolem firmy *Cisco* a existuje v několika verzích. Nejrozšířenější jsou verze 5 a 9, přičemž až verze 9 byla veřejně popsána v RFC 3954 [22]. Tato verze byla také základem pro vývoj protokolu IPFIX.

NetFlow používá pro identifikaci toku pět až sedm položek: Zdrojovou a cílovou IP adresu, zdrojový a cílový port, typ protokolu (L3) a případně i typ služby (*ToS*) a rozhraní

[2] *Packet CAPture* – zachycení paketů.

daného zařízení. Záznamy o jednotlivých tocích obsahují ještě dodatečné informace v podobě *next hop adresy* či *TCP flags* bitů. Verze 9 dále přidává podporu pro zpracování IPv6, MPLS, VLAN a dalších informací [20].

Důležitou změnou ve verzi 9 je také přidání *šablon*, které umožňují přidání dalších monitorovaných položek do NetFlow paketu. Seznam možných položek naleznete v [22]. Struktura paketu je tedy variabilní (používají se pole typu *TLV*[3]), takže není nutné kvůli změně položek měnit software na exportérech nebo kolektoru [20]. Mezi oblíbené volně dostupné programy pro práci s NetFlow patří například `nfdump`, jehož součástí je i kolektor `nfcapd`.

NetFlow data jsou vhodná pro detekci anomálií, u kterých sledujeme hustotu a směr komunikace jednotlivých uzlů (IP adres), jako například u *DoS* či *cache-poisoning útoků*. Naopak neobsahují položky aplikačních protokolů, takže není možné provádět plnohodnotnou detekci anomálií, které se právě v aplikačních datech skrývají (například detekce *fast flux domén*).

4.4 IPFIX

IPFIX[4] je protokol pro práci se síťovými toky, definován jako standard organizace *IETF*[5] v RFC 5101 [23] (aktuální verze je k nalezení v RFC 7011 [24]). Standard vychází z NetFlow verze 9.

Podobně jako NetFlow, i IPFIX identifikuje tok podle položek z hlaviček paketů, ale u IPFIX lze navíc zvolit, které položky budou tok identifikovat (lze tedy použít klasické pětice jako u NetFlow, ale i vybrat další atributy). Pakety mají také variabilní strukturu, takže umožňují přidání dalších položek do exportovaných dat, ale navíc umožňují variabilně měnit velikost jednotlivých polí, což je vhodné například pro přenos doménových jmen.

IPFIX tak představuje variabilnější verzi NetFlow a navíc je na rozdíl od NetFlow otevřeným, nikoli proprietárním standardem. Díky možnosti exportu potřebných datových položek aplikačních protokolů (včetně položek DNS paketu jako typ dotazu, hledané doménové jméno, TTL a další) je navíc formát IPFIX vhodný i pro detekci složitějších anomálií (podobně jako pcap, ale s mnohem menšími paměťovými nároky). Z tohoto pohledu se tedy IPFIX jeví jako ideální volba pro navrhovaný detekční systém.

[3] *Type, Length, Value* – pole obsahují typ dat, jejich délku a nakonec i jejich hodnotu.
[4] *Internet Protocol Flow Information eXport*
[5] *Internet Engineering Task Force*, více informací na `http://www.ietf.org/`

Kapitola 5

Návrh systému pro efektivní detekci anomálií

V této kapitole se budeme věnovat shrnutí požadavků a návrhu systému, který bude schopen provádět efektivní detekci DNS anomálií. V našem případě se bude jednat o detekci tunelování přes DNS, ale struktura a vlastnosti detekčního systému budou využitelné i pro detekci jiných anomálií.

Cílem kapitoly je identifikace jednotlivých částí systému, jejich popis a návrh způsobu jejich komunikace. V závěru kapitoly je nastíněna možnost implementace jednotlivých částí systému pomocí *frameworku*[1] *Nemea*.

5.1 Charakteristiky systému

Navrhovaný systém bude provádět detekci vybraných DNS anomálií nad množinou vstupních dat. Výsledkem by měly být souhrnné informace o zachyceném útoku, pomocí kterých by byla možná následná implementace bezpečnostních opatření za účelem zabránění dalšího útoku.

Systém jako takový bude obsahovat několik vzájemně propojených komponent:

- **Vstupní rozhraní** – tímto rozhraním budou do systému proudit vstupní data, nad kterými bude následně probíhat detekce anomálií. Konkrétnější popis věnující se problematice vstupních dat naleznete v kapitole 5.1.1.
- **Detekční logika** – bude se skládat z detektorů, které budou sloužit k detekci jednotlivých anomálií. Jednotlivé detektory budou implementovat různě složité detekční metody a jejich vzájemným propojením či korelací budou moci zvýšit celkovou efektivitu detekce. Problematika detektorů je popsána v kapitole 5.1.2.
- **Výstupní rozhraní** – jeho úkolem bude zpracovat výstupy detektorů a zobrazit výsledky uživateli.

Detailní informace k propojení jednotlivých komponent a interní komunikaci mezi nimi naleznete v kapitole 5.2.

[1] Sada knihoven implementující určitou sadu služeb, která zjednodušuje implementaci výsledného systému.

5.1.1 Vstupní data

Vstupními daty pro detekci DNS anomálií jsou pakety protokolu DNS, které typicky poznáme podle portu 53 (ať už zdrojového či cílového). Protože pro účely detekce nepotřebujeme celý obsah paketů, využijeme pouze některé jejich části a snížíme tak objem dat, které musíme zpracovat. Právě pro tyto účely poslouží datové formáty NetFlow a IPFIX, jejichž popis naleznete v kapitole 4.

Protože plánujeme implementaci různých detekčních metod, které také používají různé formáty, bylo by vhodné mít oba tyto formáty jako vstupní, aby detektor mohl vybrat preferovaný formát. Tato možnost nebude problémem u analýzy statických dat (např. uložené bloky síťového provozu), kdy je možné data předem převést do požadovaného formátu.

U detekce nad daty v reálném čase (tedy například při umístění detekčního systému na sondě uvnitř sítě) je nejprve nutné extrahovat potřebná data (převodem do požadovaného formátu), a až následně je možné provést analýzu. Aktivní síťové sondy nicméně většinou stejně pracují s těmito formáty, takže by to ani v tomto případě nemělo působit komplikace.

5.1.2 Detektory anomálií

Pod *detektorem* si v našem systému můžeme představit komponentu, která bude implementovat jednotlivé algoritmy pro detekci DNS anomálií. Předpokládá se, že v systému bude umístěno více detektorů tak, aby mohly být vzájemně propojeny. Zároveň je jasné, že propojení detektorů bude mít vliv na celkovou výkonnost systému, ale i přesnost detekce.

Základní myšlenka pro zefektivnění současných způsobů detekce DNS anomálií je ta, že v detekčním systému sloužícím pro detekci určité anomálie budou dva vzájemně propojené detektory. První detektor bude pracovat nad daty NetFlow, které obecně ponesou méně hodnotných informací pro detekci DNS anomálií než IPFIX data. Tento detektor bude implementovat algoritmy, které provádějí detekci nad NetFlow daty a zároveň jsou rychlé. Účelem tohoto detektoru je, aby mohl s velkou propustností identifikovat („vytipovat“) data, která pravděpodobně představují nějakou anomálii. Důležitá je zde především nenáročnost detekčních postupů, aby bylo možné dosáhnout vysoké rychlosti a propustnosti při rozumné kvalitě detekce.

Druhý detektor pak bude komunikovat s prvním detektorem a nějakým způsobem přejímat jím vytipovaná data. Nad těmito daty, jejichž objem bude zákonitě menší než u prvního detektoru, bude druhý detektor provádět sofistikovanější, a tedy i časově a výpočetně náročnější analýzu. Ideálně by měl druhý detektor mít přístup k IPFIX datům, které odpovídají datům prvního detektoru, u nichž by měl pro detekci anomálií více informací (vybrané položky protokolu DNS). Druhý detektor tedy bude implementovat náročnější algoritmy detekce, které by zároveň měly poskytnout i větší přesnost detekce anomálií. Teprve druhý detektor pak rozhodne, jestli se jedná o anomálii či nikoliv.

Konkrétní možnosti, způsoby zapojení a komunikace mezi detektory jsou dále diskutovány v kapitole 5.2.

Idea propojení dvou detektorů je tedy taková, že nebudeme zbytečně provádět podrobnou analýzu veškerého síťového provozu, ale pouze jeho části. A to té části, u které je větší pravděpodobnost, že obsahuje data potřebná pro detekci anomálií. Tímto by měla klesnout celková náročnost detekce anomálií DNS a zároveň se díky kombinaci jednotlivých detekčních metod zvýšit celková přesnost detekce.

Detekční algoritmy budou implementovány v závislosti na jejich vhodnosti pro detekci konkrétní DNS anomálie a také dle jejich implementační a výpočetní náročností. Důležitá

bude také schopnost kombinace a korelace jejich výsledků. Výčet konkrétních algoritmů ale není předmětem tohoto návrhu.

5.2 Interní komunikace a propojení komponent

Jednotlivé detektory budou „připojeny“ ke vstupnímu rozhraní, které bude těmto detektorům zasílat potřebná data (v potřebném formátu). Zároveň budou muset detektory komunikovat mezi sebou, aby si mohly předávat informace o detekovaných anomáliích.

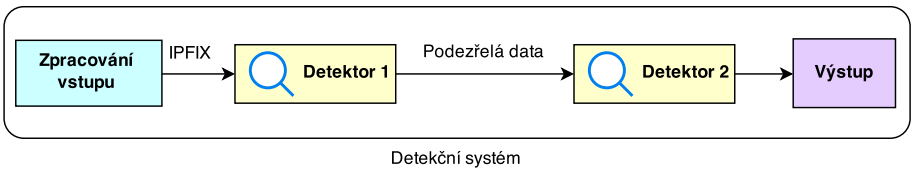
Každý detektor bude vždy provádět analýzu pouze určitého bloku dat, protože není možné detekci anomálií provádět nad nekonečným proudem dat, a to zejména s ohledem na případnou paměťovou náročnost takového řešení. Otázkou zůstává, jak určit délku (velikost) tohoto bloku. V zásadě se nabízejí dvě řešení, a to *časový interval* (např. všechny pakety přijaté během jedné minuty), nebo *limit počtu paketů* (např. jeden blok bude tvořit 500 DNS paketů). Volba tohoto parametru je důležitá zejména s ohledem na seskupení co možná největšího počtu paketů souvisejících s detekovanou anomálií. Stanovení vhodné veličiny a její hodnoty bude předmětem testování implementovaného systému s ohledem na co největší efektivitu pro daný typ anomálie.

Systém tedy bude muset implementovat i určitý typ front či vyrovnávacích pamětí, do kterých si detektory budou ukládat relevantní data.

5.2.1 Možnosti propojení komponent

Způsob propojení jednotlivých komponent uvnitř detekčního systému může mít značný dopad na paměťovou náročnost výsledného systému, ale i na jeho výkonnost. V našem případě je třeba co nejvýhodněji propojit komponentu zpracovávající vstupní data s oběma detektory, případně ještě propojit oba detektory mezi sebou.

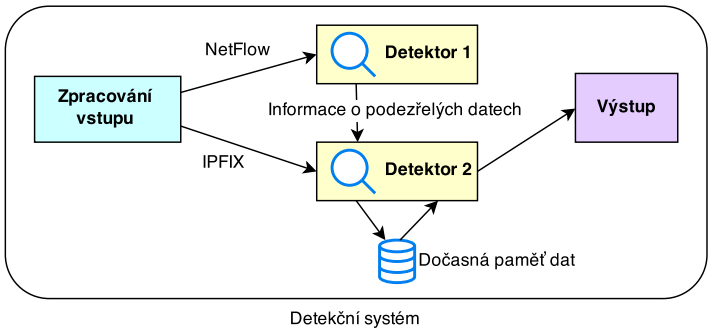
Problémem je, že první detektor se spokojí pouze s „obyčejnými“ NetFlow daty, zatímco druhý detektor bude vyžadovat rozšířená IPFIX data. Jednou možností tedy je, že i první detektor by musel pracovat s IPFIX daty, což by nutně znamenalo jeho zpomalení a tedy i snížení efektivity. Na druhou stranu to umožní spojit oba detektory za sebe, takže první detektor bude pouze přeposílat vybraná data druhému detektoru, který nad nimi následně provede analýzu. Propojení je znázorněno na obrázku 5.1.



Obrázek 5.1: První možnost propojení komponent detekčního systému.

Druhou možností je, že vstupní komponenta bude distribuovat jak NetFlow, tak IPFIX data jednotlivým detektorům. V tomto případě ovšem vyvstává problém, že druhý detektor může zahájit detekci nad konkrétním blokem dat až poté, co tento blok dat zpracuje první detektor. Druhý detektor by tedy musel implementovat vyrovnávací paměť, do které by musel ukládat bloky dat, které mohou být vhodné pro detekci anomálií. Zároveň by musela

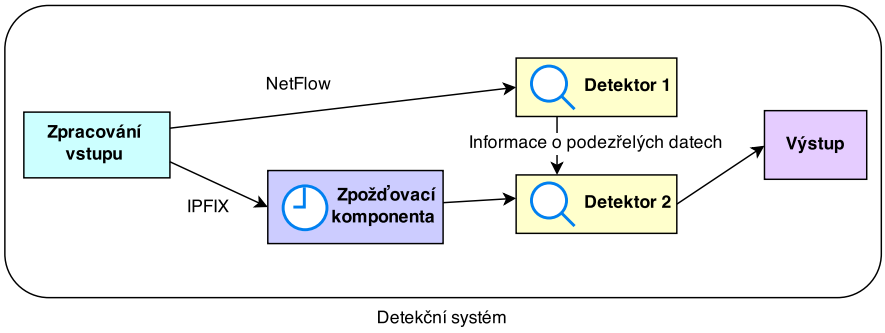
probíhat komunikace mezi oběma detektory, jejímž obsahem by byly informace identifikující podezřelé bloky dat nebo jednotlivé pakety (například podle IP adres a portů). Druhý detektor by poté musel vyhledat potřebná data ve své vyrovnávací paměti a následně nad nimi provést detekci. Tuto situaci znázorňuje obrázek 5.2.



Obrázek 5.2: Druhá možnost propojení komponent detekčního systému.

Třetí možností, která zjednodušuje druhou možnost zapojení, je vytvoření speciální komponenty, která by realizovala zpoždění vstupních dat. Tato komponenta by v zásadě implementovala vyrovnávací paměť, ve které by ukládala data po určitý časový interval. Tento interval by musel být určen s ohledem na dobu detekce prvního detektoru.

Zpožďovací komponenta by pak byla umístěna mezi vstup a druhý detektor tak, aby data přišla na vstup druhého detektoru zpožděna právě o čas doby detekce prvního detektoru. Propojení znázorňuje obrázek 5.3. První detektor by jako v předchozím případě odesílal druhému detektoru potřebná data (identifikující „vytipované“ pakety). Rozdílem ovšem je, že v tomto případě by druhý detektor mohl potřebná data filtrovat přímo ze zpožděného vstupu, zatímco v předchozím případě by musel data ještě dodatečně hledat v paměti.



Obrázek 5.3: Třetí možnost propojení komponent detekčního systému.

Které zapojení z výše uvedených možností je nejvýhodnější, nelze říci bez experimentálního ověření. Obecně se ale jeví nejvýhodněji první varianta (zapojení obou detektorů sériově), protože bude nutné vstupní data zpracovat pouze jednou a následně se budou pouze přeposílat mezi ostatními komponentami systému, a to okamžitě po jejich zpracování (nebude tak docházet ke zbytečnému čekání).

U druhé a třetí varianty zapojení by naopak bylo nutné zpracovávat vstupní data několikrát (zpracováním je myšlena extrakce užitečných dat z jednotlivých paketů či toků). U třetí varianty by navíc bylo nutné řešit synchronizaci, aby nedocházelo ke zbytečnému čekání.

5.3 Framework Nemea

Pro zjednodušení implementace výsledného systému se nabízí možnost založit jej na frameworku *Nemea*[2], který je vyvíjen na FIT VUT ve spolupráci se sdružením CESNET[3].

Nemea představuje systém pro automatizovanou analýzu síťových dat zachycených na síti v reálném čase. Systém je možné poskládat z jednotlivých stavebních bloků, nazvaných *moduly*, které jsou vzájemně propojeny pomocí *rozhraní*. Jednotlivé moduly zpracovávají vstupní data a výsledky zpracování odesílají pomocí výstupního rozhraní dalším modulům. Systém dále slibuje vysokou propustnost dat a možnost nasazení do reálného provozu [7].

Velkou výhodou systému Nemea je, že obsahuje spoustu již implementovaných modulů, například pro zpracování a práci se vstupními daty (podporuje i formáty NetFlow a IPFIX), ale i právě pro detekci síťových anomálií. Případná implementace detektorů DNS anomálií by tak doplnila již stávající kolekci.

Z pohledu předchozích informací by byl framework Nemea velmi vhodným kandidátem pro implementaci našeho detekčního systému. Díky existujícím modulům pro práci s formáty NetFlow a IPFIX by odpadla nutnost práce na modulech pro vstup a výstup, a pak by bylo možné se plně soustředit na implementaci detekčních algoritmů a jejich spolupráci za účelem zefektivnění detekce DNS anomálií.

[2] *NEtwork MEasurements Analysis* – analýza měření sítí.

[3] *Czech Education and Scientific NETwork* (síť sdružující české vysoké školy a vědecká pracoviště). Více informací naleznete na webu [1].

Kapitola 6

Implementace navrženého detekčního systému

V této kapitole jsou uvedeny detaily týkající se implementace navrženého detekčního systému. Popsána je jeho architektura a činnost jednotlivých programových celků. Následuje popis modulů pro zpracování vstupních dat. V podkapitole 6.4 pak je podrobně popsán detekční proces různých variant detektorů pro detekci tunelování přes DNS.

Systém pro detekci síťových anomálií je implementován v podobě programu, který analyzuje vstupní data (možnost různých vstupních formátů) a výsledky provedené analýzy následně uloží do souboru. Výstupní soubor obsahuje detaily o detekovaných hrozbách a také statistické informace ohledně detekčního cyklu. Uživatel může výsledky detekce ovlivnit díky konfiguračnímu souboru, ve kterém může specifikovat hodnoty jednotlivých parametrů ovlivňujících detekční proces (viz kapitola 6.4).

Samotný program je napsán v programovacím jazyce *C++*. Tento jazyk je jedním z nejrozšířenějších a nejoblíbenějších programovacích jazyků současnosti [11]. Mezi jeho výhody patří podpora pokročilých technik jako *objektově orientované programování*[1], podpora výjimek, šablon a další. Jeho další předností je také vysoká rychlost výsledného programu oproti jiným jazykům (jedná se o kompilovaný jazyk, ale velkou roli samozřejmě hraje způsob, jakým je program napsán). Jazyk C++ jsem pro implementaci vybral zejména díky mým zkušenostem s tímto jazykem, ale i díky jeho výše uvedeným přednostem.

Aby byly možnosti nasazení detekčního systému co nejširší, byly vytvořeny dvě verze detekčního systému – *samostatná verze*, která představuje kompletní detekční systém jako jeden spustitelný soubor a *Nemea verze*, kde je detekční systém implementován jako modul do *frameworku Nemea* (krátký popis lze nalézt v kapitole 5.3).

Samostatná verze programu vznikla jako první z důvodu testování a ladění programu a detekčních kritérií nad vlastními daty. Tato verze programu umožňuje statickou detekci (nad uloženými soubory) s různými datovými formáty (NetFlow, IPFIX, pcap), ale také podporuje detekci v reálném čase díky možnosti zachytávání paketů na zvoleném síťovém rozhraní.

Druhá verze programu v podobě modulu do systému Nemea byla implementována zejména díky možnosti reálného nasazení na servery, které již systém Nemea využívají. Oproti samostatné verzi programu tato verze podporuje pouze detekci v reálném čase. Samotný detekční algoritmus funguje totožně jako u samostatné verze programu.

[1] Také označováno zkratkou *OOP*.

6.1 Překlad a spuštění programu

Před prvním spuštěním detekčního systému je nejprve nutné ho přeložit ze zdrojových kódů. Před tímto krokem je třeba zkontrolovat, zdali Váš systém splňuje potřebné systémové požadavky uvedené v kapitole 6.1.1. Pokud ano, můžete program přeložit příkazem `make`.

Po úspěšném překladu je již možno program spustit (příkazem `./detector` *parametry*). Podporované parametry zjistíte příkazem `./detector -h`.

Výše uvedené kroky předpokládají systém *Unixového typu* (*Linux, Unix*). Na ostatních systémech se mohou vyskytnout problémy s kompatibilitou použitých knihoven, díky kterým by nebylo možné program přeložit.

6.1.1 Systémové požadavky

Aby bylo možné přeložit první variantu detekčního systému (samostatný program) ze zdrojových kódů a spustit na Vašem počítači, je třeba splnit několik následujících požadavků:

- **Překladač jazyka C++ s částečnou podporou standardu C++11** – protože program využívá některé nové prvky jazyka C++, které nejsou podporovány ve starších verzích jeho standardu (C++98), je nutné použít překladač s alespoň částečnou podporou nových standardů (referenční překladač `gcc` s parametrem `-std=c++0x` či `-std=c++11`).
- **Knihovna `libpcap`** – je knihovna sloužící pro čtení souborů ve formátu pcap, která také umožňuje zachytávání paketů v reálném čase. Knihovna není dodávána s programem a je třeba ji nainstalovat samostatně.
- **Knihovna `libnfdump`** – je knihovna sloužící pro čtení binárních souborů s toky NetFlow vytvořených programem `nfcapd`. Tato knihovna je již v základu přibalena s programem ve zkompilované podobě (pro systém Linux/x86) v souboru `libnfdump.a` a je dynamicky linkována při překladu. Pokud Vám tento způsob nevyhovuje, je třeba si knihovnu stáhnout a nainstalovat samostatně.

Pro instalaci druhé verze detekčního systému (modul do systému Nemea) musíte splnit tyto požadavky:

- **Systém Nemea** – je třeba mít nainstalován systém Nemea [7] včetně potřebných systémových knihoven. Systém Nemea není dodáván s programem a je třeba jej nainstalovat samostatně.
- **Překladač jazyka C++ s částečnou podporou standardu C++11** – viz samostatná verze programu.

Pokyny pro instalaci potřebných knihoven jsou uvedeny v souboru `Readme.txt`, který je přibalen s programem.

6.2 Struktura programu

Jak již bylo naznačeno v návrhu, celý detekční systém je rozdělen na několik samostatných celků, které spolu komunikují (v rámci komunikace spolu sdílejí data). Tyto celky můžeme nazvat jako *moduly*. Jednotlivé moduly jsou v programu reprezentovány *třídami*, přičemž

každý modul je reprezentován jednou hlavní třídou. Kromě hlavních tříd modulů se v programu nachází také několik obecných či pomocných tříd, které mohou využívat jednotlivé moduly ve své implementaci.

Mimo tříd a jejich třídních metod se v programu vyskytuje také několik obecných funkcí, které jsou samostatně k dispozici. Jedná se především o pomocné funkce, například funkce pro převod IPv4 adresy z její binární podoby (struktura `in_addr`) na textový řetězec (typ `string`).

Velmi důležitou součástí implementace detekčního systému jsou také datové struktury, které určují jednotný formát dat, se kterým jednotlivé moduly či součásti modulů (třídy či funkce) pracují.

Posledním důležitým prvkem programu je *hlavní funkce* `main`, která je zároveň první spuštěnou funkcí programu. Úkolem funkce `main` je kontrola konfiguračních parametrů programu (viz 6.2.4) a následné spuštění jednotlivých modulů (dle přijatých konfiguračních parametrů). Jakmile všechny spuštěné moduly dokončí svou práci, je posledním úkolem této funkce uložení souhrnných statistik o průběhu detekce do zvoleného výstupního souboru.

Zdrojové kódy deklarace tříd, obecných funkcí a datových struktur jsou uloženy v *hlavičkových souborech* (soubory s příponou `.h`). Definice jednotlivých třídních metod či obecných funkcí jsou pak uloženy ve *zdrojových souborech* (soubory s příponou `.cpp`), které mají stejné jméno jako hlavičkový soubor, ke kterému příslušejí.

6.2.1 Moduly

Moduly představují jednotlivé funkční celky detekčního systému, které mezi sebou komunikují a předávají si data. Každý modul tak provádí specifickou činnost a výsledky pak předává dalším modulům dle jejich zapojení.

Abychom dosáhli lepší výkonnosti výsledného detekčního systému, je každý modul spuštěn v samostatném *vláknu*[2]. Každý modul tedy pracuje samostatně, ale zároveň může pracovat více modulů současně (na systémech s *vícejádrovými* či *vícevláknovými* procesory, nebo na *víceprocesorových* systémech). Tato skutečnost tedy umožňuje, že zatímco vstupní modul zpracovává další data ze vstupu, první detektor může současně provádět detekci nad již zpracovanými vstupními daty. Stejně tak druhý detektor může současně provádět detekci nad jinými daty, které mu byly předány z prvního detektoru.

Důvodem použití vláken místo *procesů* bylo, že vlákna mají menší režii než procesy. Zároveň jednotlivá vytvořená vlákna a hlavní program sdílejí adresový prostor, takže si mohou jednoduše předávat data díky sdíleným proměnným [53]. U procesů by bylo sdílení dat mnohem složitější, stejně jako celá jejich režie. Drobnou nevýhodou paralelizace je nutnost synchronizace vláken při modifikaci sdílených dat, ale tomuto problému se nevyhneme ani při použití procesů. Jak je synchronizace řešena v implementovaném systému se dozvíte v kapitole 6.2.3.

Pro implementaci vláken v programu jsou využity standardní vlákna jazyka C++ (`std::thread`), které do jazyka přibyly až v nových verzích jazyka (C++11). Vlákna jednotlivých modulů jsou vytvářena a zase rušena (jakmile dokončí svou činnost) ve funkci `main`, tedy v hlavní funkci programu.

[2] Mechanizmus vláken umožňuje paralelní vykonání různých větví programu. Vlákna jsou odlehčenou variantou *procesů*.

Jaké konkrétní moduly tedy v programu nalezneme? Následuje přehled:

- **Modul pro zpracování vstupu** – tento modul zajišťuje zpracování vstupu (různé vstupní formáty). Jeho činnost zahrnuje čtení dat, extrakci užitečných dat do interních struktur a tvorbu tzv. *bloků dat* (více se dozvíte v kapitole 6.3). Výstupem modulu jsou pak bloky dat, které jsou následně podrobeny detekci.
- **Detektor 1** – provádí základní detekci nad celými bloky dat. Na základě výsledků detekce buď předá blok dat k dalšímu zpracování druhému detektoru, nebo ho zahodí (vymaže data). Další informace naleznete v kapitole 6.4.2.
- **Detektor 2** – vykonává pokročilou detekci nad obsahem jednotlivých datových bloků (zkoumá jednotlivé pakety či toky). Výsledky detekce ukládá do výstupního souboru. Více informací je uvedeno v kapitole 6.4.3.

Každý uvedený modul může mít několik různých implementací. Jednotlivé implementace se liší zejména formátem vstupních dat, který ovlivňuje vnitřní struktury jednotlivých modulů. V programu tak existují například tři různé implementace modulu pro zpracování dat, které pracují s jednotlivými vstupními formáty – IPFIX, NetFlow a pcap.

6.2.2 Propojení modulů

V návrhu byly představeny některé možnosti zapojení výše uvedených modulů. První variantou bylo zapojení všech modulů do série, další varianty potom počítaly s paralelním zapojením detektorů, přičemž se lišily ve způsobu synchronizace vstupu druhého detektoru s výstupem prvního detektoru (dočasná paměť nebo zpožďovací komponenta).

Ve výsledné implementaci je nakonec použita první varianta (tedy sériové zapojení modulů[3]). K výběru první varianty mě nakonec vedla následující úvaha o efektivitě jednotlivých variant propojení.

Sériová varianta představuje logickou posloupnost, kdy nejprve zpracujeme vstupní data a ihned po jejich zpracování je předáme prvnímu detektoru. Ten následně provede detekci a vybraná (detekovaná) data odešle druhému detektoru, který může pokračovat v detekci nad těmito konkrétními, obdrženými daty. Tato posloupnost v sobě zahrnuje implicitní synchronizaci, kdy jednotlivé detektory mohou zahájit detekční proces až po obdržení dat. Zároveň detektory pracují přímo s konkrétními daty určenými k detekci, což detekci urychluje oproti nutnosti nového zpracování dat či jejich filtrace (například vyhledáváním konkrétních dat v dočasné paměti). Určitým omezením tohoto přístupu je skutečnost, že první detektor má k dispozici i data, která nevyužívá k detekci (data aplikačního protokolu DNS). To ve výsledku vede ke zbytečnému zdržení v modulu zpracování vstupu, který detailně zpracovává i pakety či toky, které budou následně zahozeny hned u prvního detektoru. Nicméně se domnívám, že toto drobné zdržení není natolik limitujícím faktorem (vzhledem k potenciálním zdržením při synchronizaci dat u dalších variant zapojení). Samotný první detektor s těmito aplikačními daty nijak nepracuje, takže u něj se žádné další zdržení neprojeví.

Druhá varianta zapojení detektorů paralelně s načítáním dat z dočasné paměti přináší hned několik činností, které na první pohled prodlouží detekční proces a tedy i sníží efektivitu výsledného řešení. Nejprve je to nutnost zpracování dat ve dvou různých formátech najednou (což je nejen časově, ale i prostorově náročné). Komponenta pro zpracování

[3] Moduly ve skutečnosti pracují paralelně, ale data si předávají v následující sériové posloupnosti: *zpracování vstupu – detekce 1 – detekce 2*.

vstupu by musela zpracovávat data v obou formátech a následně je předávat detektorům. Nevyhnuli bychom se přitom stejnému problému jako u sériové varianty, tedy že by byly zbytečně zpracovány i pakety či toky, které by vyřadil hned první detektor. V tomto případě je problém ovšem ještě horší, neboť bychom tyto zbytečné toky či pakety ukládali v dočasné paměti u druhého detektoru, který by je z ní odstranil až po nějaké době. To je zároveň i dalším problémem tohoto řešení, protože by druhý detektor ještě musel požadovaná data hledat právě v dočasné paměti. Režie spojená s tímto vyhledáváním u sériové varianty odpadá.

Třetí varianta zapojení detektorů pak využívá zpožďovací komponentu. Zde také nastává problém s nutností zpracování dat v různých formátech, ale naopak odpadá problém s vyhledáváním dat ve sdílené paměti. Ten je nahrazen jiným problémem, a sice synchronizací zpožďovací komponenty s prvním detektorem tak, aby druhý detektor mohl právě včas odfiltrovat nepotřebná data a mohl přímo číst pouze data určená pro další detekci. Synchronizace by musela být precizní, protože jinak by druhý detektor buď zmeškal příjem užitečných dat, nebo by na ně naopak musel čekat déle než by bylo nutné v případě sériové varianty.

6.2.3 Sdílení dat mezi moduly

Mezi jednotlivými moduly detekčního systému je třeba sdílet dva druhy dat. Prvním jsou data, která musí být dostupná v celém systému, tedy všem modulům. Zde patří například konfigurační parametry celého systému, objekt pro uložení detekčních statistik od jednotlivých modulů (statistiky jsou uloženy na konec výstupního souboru) nebo příznak zastavení běhu programu, který je využit v případě předčasného ukončení běžícího programu (např. *signálem SIGINT*).

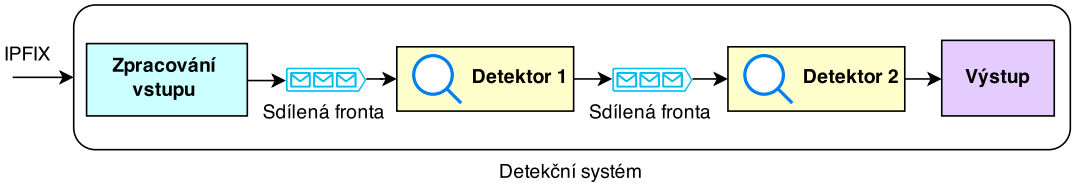
Tato data jsou mezi jednotlivými moduly sdílena ve formě globálních proměnných programu (využíváme toho, že jednotlivá vlákna mohou přistupovat ke globálním proměnným). Výhodou je, že u nich není nutné řešit synchronizaci jednotlivých vláken, protože jsou data buď nastavena ještě před vytvořením vláken a moduly je pouze čtou (parametry systému), nebo jednotlivá vlákna zapisují do různých proměnných a zároveň jsou zapsaná data čtena pouze vláknem hlavního programu – funkcí `main` (detekční statistiky, dočasné ukazatele).

Druhým typem sdílených dat jsou *bloky*[4] vstupních dat, se kterými pracují detektory. Zde je potřeba jiného modelu sdílení dat než v předchozím případě, protože některé moduly data vytvářejí, zatímco jiné s nimi pracují. Proto je zde využito modelu *producent – konzument*, kdy producent data vytváří v nepravidelných intervalech a konzument čeká až budou data dostupná a následně je zpracovává. Zároveň platí, že k těmto datům přistupují maximálně dvě různá vlákna.

Konkrétně jsou jednotlivé bloky dat předávány ve formě ukazatelů na instance třídy představující blok dat (třída `ip_storage`). Tyto ukazatele jsou pak umísťovány do sdílené fronty, kterou představuje třída `shared_queue_ip_storage`. Ta zajišťuje synchronizaci jednotlivých vláken, které k ní přistupují (přístup k frontě představuje *kritickou sekci*[5]. Fronta samotná je realizována standardní frontou jazyka C++ (`std::queue`). Synchronizaci pak zajišťují další standardní prostředky téhož jazyka (od standardu C++11) – *semafor* (`std::mutex`) a synchronizační primitivum (`std::condition_variable`), které umožňuje notifikaci čekajícího vlákna (vzbudí vlákno čekající na data od producenta, jakmile jsou data dodána).

[4] V anglických článcích jsou tyto bloky nazývány *bins*, tedy „koše“ dat.

[5] Úsek programu, jehož provádění jedním procesem (vláknem) vylučuje jeho současné provádění jiným procesem (vláknem) [53].



Obrázek 6.1: Umístění sdílených front v implementovaném systému.

V detekčním systému jsou celkem dvě tyto sdílené fronty. První je umístěna mezi modul pro zpracování vstupních dat (producent) a první detektor (konzument). Druhá fronta je umístěna mezi první detektor (producent) a druhý detektor (konzument). Propojení je znázorněno na obrázku 6.1. Samotné instance těchto sdílených front jsou pak vytvořeny jako globální proměnné, takže jsou přístupné všem vláknům programu.

Uvedené řešení sdílení dat umožňuje rychlou a pohodlnou výměnu dat mezi jednotlivými detekčními moduly. Jednoduchá synchronizace dvou vláken nad přístupem ke sdílené frontě pak minimalizuje dobu čekání na data, stejně jako synchronizační režii. Tyto skutečnosti nepochybně přispívají k větší efektivitě navrženého detekčního systému.

Výše uvedené struktury a třídy jsou k nalezení ve zdrojových souborech `shared.cpp` a `shared.h`.

6.2.4 Konfigurace detekčního systému

Aby mohl uživatel detekčního systému ovlivňovat jeho výsledky a přizpůsobit jeho činnost svým požadavkům, má k dispozici tři nástroje. První možností nastavení jsou *parametry programu*. Ty jsou uváděny při spuštění programu v podobě *přepínačů* (`./detector` *parametry*). Parametry programu slouží k nastavení činnosti programu jako takové (typ detekce, cesta ke vstupním souborům a podobně). Parametry mohou být uváděny v libovolném pořadí (pro zpracování parametrů programu je využita knihovna `Getopt`) a je kontrolována jejich korektnost (povolené kombinace parametrů, korektnost vstupních hodnot).

Druhou možností konfigurace detekčního systému je *konfigurační soubor*. To je soubor textového formátu, který obsahuje konfigurační parametry a jejich hodnoty. Zde uvedené konfigurační parametry se vztahují k samotnému detekčnímu procesu a většinou se jedná o nastavení *prahových hodnot* jednotlivých detekčních parametrů. Konkrétní detekční parametry jsou uváděny v kapitole 6.4.

Konfigurační soubor je využíván, pokud je programu předán programovým parametrem `-c`. Pokud není konfigurační soubor specifikován, je použito základní nastavení jednotlivých detekčních parametrů (to je možné nalézt v souboru `detectionsettings.h`). Pokud konfigurační soubor neexistuje, je možné ho vytvořit programovým parametrem `-m` (pro vytvoření se použije základní nastavení detekčních parametrů).

Pokud je konfigurační soubor specifikován při spuštění programu, provede se v rámci zpracování programových parametrů jeho *syntaktická analýza* a následná extrakce hodnot jednotlivých parametrů. Každý parametr je uveden na samostatném řádku a má následující formát:

```
<jméno parametru> = <hodnota>
```

Parametr	Popis
`-h`	Vytiskne nápovědu programu (včetně seznamu parametrů).
`-i <file>`	Specifikuje vstupní soubor nebo název vstupního síťového rozhraní.
`-r <folder>`	Specifikuje adresář se vstupními soubory.
`-o <file>`	Specifikuje výstupní soubor (výsledky detekce).
`-p`	Vstupní soubor je ve formátu pcap.
`-n`	Vstupní soubor je ve formátu NetFlow (soubor CSV).
`-d`	Vstupní soubor je ve formátu NetFlow (binární data).
`-x`	Vstupní soubor je ve formátu IPFIX (soubor CSV).
`-e`	Vstupem jsou pakety zachycené na síťovém rozhraní, jehož název je specifikován parametrem `-i`.
`-s`	Vytvoří soubor se statistickými daty dle vstupních souborů. Tento soubor se pak využívá k detekci (viz kapitola 6.4.1).
`-t <file>`	Specifikuje soubor se statistickými daty předem vytvořený použitím parametru `-s`.
`-m <file>`	Vytvoří nový konfigurační soubor se základními hodnotami parametrů.
`-c <file>`	Specifikuje konfigurační soubor s nastavením detekce (může být vytvořen použitím parametru `-m`).
`-l <file>`	Vytvoří nový seznam povolených domén (*whitelist*) se základním seznamem povolených domén.
`-w <file>`	Specifikuje seznam povolených domén (může být vytvořen použitím parametru `-l`).
`-D <level>`	*Ladící režim.* Dle nastavené úrovně vypisuje ladící data (podrobnosti o paketech či blocích dat).

Tabulka 6.1: Přehled podporovaných parametrů samostatné verze programu. Detailnější popis jednotlivých parametrů je uveden v nápovědě programu.

Hodnotou je buď celé číslo, nebo textový řetězec (dle parametru). U každého parametru je uveden komentář s jeho popisem a vzorovou hodnotou. Komentáře lze do konfiguračního souboru vkládat vložením znaku „`#`“ na začátek příslušného řádku. Bílé znaky a prázdné řádky jsou syntaktickým analyzátorem také ignorovány.

Třetím konfiguračním nástrojem je seznam povolených domén (často označován jako *whitelist*). Zde může uživatel specifikovat, které domény budou vyloučeny z detekce (provoz týkající se těchto domén nebude detekován). Jedná se textový soubor se stejnými pravidly jako u konfiguračního souboru. Jediným rozdílem je, že soubor neobsahuje parametry, ale povolené domény (na každém řádku je uvedena jedna povolená doména). Domény musí být uvedeny s maximálním počtem subdomén odpovídajícím počtu subdomén, které se ukládají do agregační mapy (viz kapitola 6.4.3).

6.3 Zpracování vstupu

Úkolem modulu pro zpracování vstupu je extrakce užitečných dat ze vstupu a následná tvorba *bloků* či *košů* dat. Tento problém ovšem lze dekomponovat na několik menších kroků.

Prvním krokem je čtení jednotlivých záznamů ze vstupních souborů (či síťového rozhraní). Čtení probíhá ve formě *záznamů o tocích* či *paketů*. Každý načtený záznam či paket je poté nutné zpracovat. Úkolem zpracování je extrahovat a uložit potřebné informace do

interní struktury, se kterou se dále pracuje v celém detekčním systému. Každý typ dat má definován svou interní strukturu. Jednotlivé struktury a jejich položky jsou popsány v kapitolách věnujících se jednotlivým formátům. V tomto kroku se zároveň odfiltrují veškeré záznamy, které se netýkají DNS provozu (poznáme je tak, že protokol v IP hlavičce není UDP či ani jeden z portů není port 53[6]). Podporované protokoly *síťové vrstvy*[7] jsou IPv4 i IPv6. Adresy jsou ukládány do *unie*, ve které může být v jednom okamžiku uložen pouze jeden typ adresy. To nám umožní snížení paměťové náročnosti.

Následně je potřeba jednotlivé záznamy (nyní již ve formě interních struktur) seskupit do jednotlivých datových bloků. Seskupování lze provádět dvěma různými způsoby. Prvním způsobem je seskupení určitého počtu záznamů, kdy každý blok obsahuje stejný počet záznamů (vyjma posledního bloku). Počet záznamů v bloku určuje parametr `bin_packets` v konfiguračním souboru. Tento způsob tvorby bloků dat je velmi jednoduchý, nicméně takový blok neříká nic o časové posloupnosti záznamů, které obsahuje (naplnit dva takové bloky může trvat různě dlouhou dobu).

Druhou možností je seskupit záznamy do bloků trvajících určitý čas. Jeden blok tedy může být dlouhý např. 20 s. Délku bloku lze definovat parametrem `bin_time` v konfiguračním souboru. Jednotlivé bloky tedy nemusí mít stejný počet záznamů, ale obsahují všechny záznamy v daném časovém rozmezí (začátek až konec bloku). To přináší nové možnosti práce s jednotlivými bloky dat – např. detekci založenou na časově závislé statistice o obsahu jednotlivých datových bloků (jak funguje se dozvíte v kapitole 6.4). Toto řešení je složitější než bloky pevné délky, protože musíme hlídat časy jednotlivých záznamů, abychom mohli včas začít tvořit nový blok. V některých případech je situace ještě složitější, protože se můžeme setkat s nesprávným pořadím jednotlivých záznamů (potom musíme ještě provádět řazení záznamů dle času – viz kapitola 6.3.1). U síťových toků uvažujeme čas záznamu jako čas ukončení toku, protože je pro řazení vhodnější (toky jsou exportovány po jejich ukončení, takže novější toky mají novější koncový čas, zatímco jejich počáteční čas může být různý díky různé délce jednotlivých toků). U paketů (pcap) je čas záznamu časem přijetí paketu.

Uživatel detekčního systému může zvolit, jaký způsob tvorby bloků použije, nicméně existují určitá omezení. U formátů IPFIX a NetFlow není možné využít časově závislou statistickou detekci spolu s bloky dat pevné délky, protože by nebylo možné porovnávat data jednotlivých bloků právě z důvodu jejich rozdílné doby trvání. Pokud tedy chceme spolu s těmito vstupními formáty využít bloky pevné délky, musíme použít alternativní *detekční schéma*. Formát pcap toto omezení nemá, protože detektor pracující s tímto formátem nevyužívá časově závislou statistickou detekci.

Při tvorbě jednotlivých bloků dat se zároveň počítají statistické informace o každém bloku, které jsou následně využívány prvním detektorem. To má tu výhodu, že první detektor nemusí vůbec procházet jednotlivé záznamy bloku a z nich následně počítat statistiky, ale může přistoupit rovnou k již spočítaným statistikám pro daný blok. Jedná se tedy o další zefektivnění detekčního procesu.

Poslední činnost, kterou modul pro zpracování vstupu může provádět, je filtrace běžných typů DNS dotazů. Ta umožňuje odfiltrovat dotazy typu A, AAAA a PTR, které dle [18] tvoří až 80 % veškerého DNS provozu. Nevýhodou v tomto případě je, že ztratíme možnost detekce anomálií, které tyto záznamy využívají. Velkou výhodou je ale rapidní pokles objemu dat, který musí detektory následně zpracovat při jejich inspekci. Filtrace běžných typů dotazů je specifikována parametrem `filter_common_qtypes` v konfiguračním souboru,

[6] Standardní port pro komunikaci s DNS serverem.
[7] Dle referenčního modelu ISO/OSI.

lze ji tedy aktivovat či deaktivovat dle požadavků uživatele.

Zajímavostí je, že při filtrování těchto záznamů jsou filtrované záznamy zaznamenávány do statistik u příslušných bloků dat (ale nejsou do těchto bloků ukládány). To je důležité z toho důvodu, aby mohly detektory pracující s těmito statistickými informacemi správně vyhodnotit příslušné bloky dat, přestože všechna data v blocích uložena nejsou (jinak bychom museli mít k dispozici zvláštní sadu statistických informací[8] pro filtrované a nefiltrované vstupy).

6.3.1 Řazení síťových toků

Oproti návrhu přineslo nasazení detekčního systému v reálných podmínkách rozsáhlé sítě nové problémy, které je třeba řešit. Původní návrh počítal s tím, že vstupní data o síťových tocích budou logicky řazena dle času jednotlivých toků (konkrétně dle času ukončení každého toku). Jak se ale ukázalo, ve skutečnosti přicházejí jednotlivé toky v různém pořadí a tyto výkyvy v časech jednotlivých toků mohou být i v řádu minut (viz tabulka 6.2). To je pravděpodobně způsobeno tím, že data o tocích přicházejí na kolektor (který toky ukládá) z více různých sond (exportérů) v rámci celé sítě. Přijetí dat z různých sond tedy trvá různou dobu a způsobuje rozdíly v časech u ukládaných dat.

```
2015-03-17T16:18:20.154
2015-03-17T16:17:07.262
2015-03-17T16:17:10.344
2015-03-17T16:16:49.378
2015-03-17T16:16:52.153
2015-03-17T16:16:46.523
2015-03-17T16:17:07.331
```

Tabulka 6.2: Příklad rozptylu časů u jednotlivých toků (v tomto pořadí jsou toky uloženy ve vstupním souboru). Převzato z reálných dat.

Abychom mohli správně použít tvorbu bloků dat na základě jejich délky (doby), musíme tedy nějakým způsobem seřadit vstupní data dle jejich času. To platí pouze pro síťové toky (tedy formáty IPFIX a NetFlow), u pcap souborů jsou pakety již seřazeny dle času jejich příchodu. Řazení je možné jednoduše provést nad statickými vstupními daty ještě před spuštěním detekčního systému, nicméně tato varianta není použitelná u živých dat (modul pro systém Nemea). Další nevýhodou je velká časová náročnost takové metody (zvláště pokud vezmeme v úvahu, že vstupem mohou být soubory o velikostech v řádu *gigabytů* obsahující miliony řazených záznamů).

Proto bylo implementováno řazení síťových toků za běhu programu a je součástí zpracování vstupních dat (jedná se o poslední krok při tvorbě bloků dat). Řazení bylo navrženo a implementováno s ohledem na jeho paměťovou náročnost, ale i rychlost.

Princip řadící metody by se dal přirovnat k mechanizmu *posuvného okna*[9], který se využívá u síťových protokolů zajišťujících spolehlivé doručení dat (např. TCP). Položkami tohoto okna jsou jednotlivé bloky dat, pro které určíme čas jejich začátku a konce (začátek a konec prvního bloku odvodíme z času prvního zpracovávaného záznamu, ostatní bloky pak

[8] Souhrnné statistické informace pro danou denní dobu, na jejichž základě probíhá detekce. Viz kapitola 6.4.1.

[9] V anglické literatuře lze nalézt pod pojmem *sliding window*.

navazují na časy předchozího bloku). Maximální počet takových bloků v okně je stanoven parametrem `num_of_sorting_bins` v konfiguračním souboru.

Řazení každého záznamu potom probíhá tak, že v okně hledáme příslušný blok, do kterého řazený záznam dle jeho času patří. Záznam poté umístíme do nalezeného bloku. Vyhledávání uvnitř okna využívá zvýšené pravděpodobnosti toho, že budeme řadit záznam do stejného bloku jako záznam předchozí. Tato situace nastane v případě, že budou vstupní záznamy částečně seřazeny (budou obsahovat sekvence záznamů s po sobě jdoucími časy), což je velmi časté. Při hledání správného bloku uvnitř okna tedy nejprve zkoušíme blok, do kterého jsme řadili poslední záznam. Pokud tento blok časově nevyhovuje, hledáme v následujících či předchozích blocích (dle hledaného času). Protože je okno implementováno jako *vektor* bloků, je jeho procházení na jednu či druhou stranu od výchozího bloku jednoduché.

Pokud je čas řazeného záznamu menší než čas prvního bloku v okně (nejstarší blok), potom jsme již zmeškali možnost záznam zařadit korektně (nemůžeme se vracet do minulosti), a tak zařadíme záznam do tohoto prvního (nejstaršího) bloku. Záznam tedy nemusí být zařazen vždy přesně do bloku, který mu dle časů přísluší. Druhým řešením by bylo daný záznam ignorovat (dále ho nezpracovat), což by ovšem v důsledku mohlo znamenat, že nedetekujeme potenciální hrozbu. Naopak, pokud záznam patří do bloku, který v okně ještě není (budoucnost), posouváme celým oknem tak dlouho, dokud se nepřesuneme nad hledaný blok. Každý jednotlivý posuv okna má za následek, že nejstarší blok je z okna vyřazen a umístěn do sdílené fronty pro zpracování prvním detektorem (obsah tohoto bloku se od této chvíle nemění) a do okna je přidán nový blok s časovým rozmezím následujícím za původně posledním blokem okna. Pokud tedy provádíme hodně posuvů najednou, může se stát, že skončíme s oknem plným prázdných (nových) bloků.

Jak plyne z předchozího popisu, pro celkovou rychlost a paměťovou náročnost řazení je nejdůležitějším parametrem počet bloků v okně. Pokud je moc velký, bude vyhledávání pomalejší a zároveň budeme uchovávat v paměti velké množství nedokončených bloků. Pokud je moc malý, pak může často docházet k nepřesnému řazení do prvního bloku (starší záznamy) či zkreslení statistických dat jednotlivých bloků, a tedy i nefunkční detekci. Proto je nutné volit velikost okna s ohledem na rozptyl časů v po sobě následujících vstupních záznamech.

Řazení je implementováno třídou `bin_sorter`, která je definována a deklarována v souborech `binsorter.cpp` a `binsorter.h`.

6.3.2 NetFlow

Vstupní data ve formátu NetFlow je v samostatné verzi detekčního systému možné použít ve dvou různých typech vstupních souborů. Oba typy vycházejí z balíku nástrojů pro práci s NetFlow daty – `nfdump` [32]. Ten patří k nejrozšířenějším nástrojům pro práci s NetFlow daty a neměl by tak být problém data v požadovaném formátu získat.

Prvním podporovaným typem vstupních souborů jsou binární soubory vytvořené programem `nfcapd`, což je vlastně NetFlow kolektor, který je součástí výše zmíněného balíku. Takto vytvořené soubory mají unikátní binární strukturu, se kterou umí pracovat ostatní nástroje balíku `nfdump`. Naštěstí existuje knihovna `libnfdump` [13], která je odvozena ze zdrojových kódů balíku `nfdump` a umožňuje čtení právě těchto binárních souborů. Tuto knihovnu jsem použil pro čtení uvedených binárních souborů.

Druhým podporovaným typem vstupních souborů jsou textové CSV[10] soubory. Tyto soubory obsahují záznamy na jednotlivých řádcích, přičemž jsou jednotlivé položky kaž-

[10] *Comma-separated values*, tedy hodnoty oddělené čárkami.

dého záznamu odděleny čárkou. Po načtení hlavičky můžeme určit pořadí jednotlivých hodnot (sloupců) a následně z jednotlivých řádků přečíst konkrétní hodnoty. CSV soubory s NetFlow daty podporovaného formátu lze vytvořit z výše uvedených binárních dat díky nástroji `nfdump`, který podporuje export právě do CSV souboru. Lze tak učinit následujícím příkazem:

```
nfdump -r (vstupní soubor) -o csv | cut -d, -f1-15 > (výstupní soubor CSV)
```

Ten zároveň zajistí uložení pouze nezbytných dat (programem `cut` jsou odstraněny nepotřebné položky, které neobsahují pro nás důležité informace).

Jednotlivé NetFlow záznamy jsou v detekčním systému reprezentovány strukturou `flow`, která obsahuje položky uvedené v tabulce 6.3. Je vidět, že záznam neobsahuje žádná specifická DNS data, takže není možné provádět inspekci těchto dat ani žádné detekční metody na ní založené.

Datový typ	Název	Popis
`unsigned long`	flowNum	Číslo záznamu.
`flow_time`	flowTime	Čas konce toku a jeho délka (doba trvání).
`bool`	ipv6	Příznak použití IPv6 adresy.
`ip46_union`	srcIP	Zdrojová IP adresa (unie pro oba typy adres).
`ip46_union`	dstIP	Cílová IP adresa (unie pro oba typy adres).
`u_int8_t`	ipProtocol	Protokol transportní vrstvy.
`u_int16_t`	srcPort	Zdrojový port.
`u_int16_t`	dstPort	Cílový port.
`u_int16_t`	packets	Počet paketů v toku.
`u_int32_t`	bytes	Počet bytů v toku.

Tabulka 6.3: Položky struktury `flow`.

Struktura `flow` je definována v souboru `common.h`. Načítání NetFlow dat a následnou tvorbu bloků dat implementují třídy `netflow_parser` a `netflow_bin_maker` umístěné v souborech `netflowparser.cpp` a `netflowparser.h`.

6.3.3 IPFIX

Vstupní data ve formátu IPFIX je možné v samostatné verzi detekčního systému dodat ve formě CSV souborů. Ty je možné vytvořit nástrojem `Flowmon Exporter` od firmy *INVEA-TECH*, konkrétně použitím modulu `DNS plugin` [39]. Ten je nasazen na serverech, ze kterých byla získána testovací data.

Protože se jedná o obyčejné textové soubory, není k jejich zpracování potřeba žádné externí knihovny. Každý tok je umístěn na jednom řádku, který obsahuje jeho jednotlivé položky. Zpracování tedy spočívá v extrakci a uložení těchto jednotlivých položek.

Datový typ	Název	Popis
`unsigned long`	flowNum	Číslo záznamu.
`flow_time`	flowTime	Čas konce toku.
`bool`	ipv6	Příznak použití IPv6 adresy.
`ip46_union`	srcIP	Zdrojová IP adresa (unie pro oba typy adres).
`ip46_union`	dstIP	Cílová IP adresa (unie pro oba typy adres).

u_int16_t	srcPort	Zdrojový port.
u_int16_t	dstPort	Cílový port.
u_int16_t	packets	Počet paketů v toku.
u_int32_t	bytes	Počet bytů v toku.
bool	msgType	Typ zprávy (dotaz/odpověď).
u_int16_t	rCount	Počet odpovědí na dotaz.
u_int16_t	qType	Typ dotazu (RR Type).
u_int32_t	ttl	TTL odpovědi.
u_int16_t	rdLen	Délka dat odpovědi.
string	qName	Hledané doménové jméno.
string	rData	Řetězec dat odpovědi.
u_int8_t	rCode	Kód odpovědi (RCODE).

Tabulka 6.4: Položky struktury `ipfix_flow`.

Jednotlivé IPFIX záznamy jsou v detekčním systému reprezentovány strukturou `ipfix_flow`, která obsahuje položky uvedené v tabulce 6.4. Záznam v tomto případě obsahuje i DNS data, která je možné využít k sofistikovanější detekci DNS anomálií.

Struktura `ipfix_flow` je definována v souboru `common.h`. Načítání IPFIX dat a následnou tvorbu bloků dat implementují třídy `ipfix_parser` a `ipfix_bin_maker` umístěné v souborech `ipfixparser.cpp` a `ipfixparser.h`.

6.3.4 pcap

Vstupní data ve formátu pcap je v samostatné verzi detekčního systému možné použít díky dvěma odlišným možnostem vstupu. Obě možnosti vycházejí z knihovny pro práci s pcap soubory a zachytávání paketů – `libpcap` [10]. Na ní mimo jiné staví související analyzátor síťového provozu `tcpdump`.

První možností vstupu je přímé použití pcap souborů, tedy binárních souborů s uloženým síťovým provozem na úrovni jednotlivých paketů. Ty je možné číst právě díky knihovně `libpcap`, která umožňuje čtení tohoto typu souborů. Knihovna podporuje čtení přímo binárního obsahu paketů, ze kterého je následně nutné extrahovat konkrétní datové položky s využitím znalosti struktury hlaviček jednotlivých protokolů (v našem případě jsou to následující hlavičky: Ethernet, IP, UDP, DNS). Extrakce užitečných informací je tedy v tomto případě o něco složitější než např. u CSV souborů. Binární pcap soubory lze získat např. využitím oblíbeného síťového analyzátoru *Wireshark* či už zmiňovaného analyzátoru `tcpdump`.

Druhou možností vstupu je *živé odchytávání paketů* přímo ze síťového rozhraní. V tomto případě program analyzuje probíhající síťový provoz na specifikovaném rozhraní. Tato možnost je tedy vhodná pro detekci anomálií v reálném čase. Živé odchytávání provozu je také umožněno využitím knihovny `libpcap` a oproti načítání dat z pcap souboru se liší pouze minimálně (práce se zdrojem dat je lehce odlišná, ale také je nutné extrahovat potřebné datové položky z hlaviček jako v prvním případě). Nejvýraznější změnou pro uživatele oproti načítání dat ze souborů je nutnost spuštění programu s právy *superuživatele*, bez kterých operační systém programu nepovolí přímý přístup k síťovému rozhraní.

Jednotlivé pcap záznamy jsou v detekčním systému reprezentovány strukturou `pkt`, která obsahuje položky uvedené v tabulce 6.5. Záznam stejně jako u IPFIX obsahuje i DNS data, která je možné využít k sofistikovanější detekci DNS anomálií.

Struktura `pkt` je definována v souboru `common.h`. Načítání pcap dat a následnou tvorbu

Datový typ	Název	Popis
`unsigned long`	pktNum	Číslo paketu.
`flow_time`	pktTime	Čas přijetí paketu.
`bool`	ipv6	Příznak použití IPv6 adresy.
`ip46_union`	srcIP	Zdrojová IP adresa (unie pro oba typy adres).
`ip46_union`	dstIP	Cílová IP adresa (unie pro oba typy adres).
`u_int16_t`	srcPort	Zdrojový port.
`u_int16_t`	dstPort	Cílový port.
`u_int16_t`	ipLen	Velikost paketu dle IP hlavičky.
`u_int8_t`	ipProtocol	Protokol transportní vrstvy.
`bool`	msgType	Typ zprávy (dotaz/odpověď).
`u_int16_t`	qCount	Počet dotazů.
`u_int16_t`	rCount	Počet odpovědí na dotaz.
`u_int16_t`	qType	Typ dotazu (RR Type).
`u_int32_t`	ttl	TTL odpovědi.
`u_int16_t`	rdLen	Délka dat odpovědi.
`string`	qName	Hledané doménové jméno.
`string`	rData	Řetězec dat odpovědi.
`u_int8_t`	rCode	Kód odpovědi (RCODE).

Tabulka 6.5: Položky struktury `pkt`.

bloků dat implementuje třída `pcap_parser` umístěná v souborech `pcapparser.cpp` a `pcapparser.h`.

6.3.5 Nemea UniRec

V systému Nemea jsou vstupní data do jednotlivých modulů distribuována díky platformě *TRAP*[11], která zajišťuje přenos dat mezi jednotlivými moduly celého systému (umožňuje například i komunikaci modulů spuštěných na různých počítačích). K tomu využívá různé způsoby komunikace: sdílenou paměť, *Unix sockety* a *TCP sockety*.

Samotná data jsou předávána ve formě *UniRec*[12] záznamů. Každý záznam se skládá z několika položek. Množina položek, obsažených v záznamu, je definována v tzv. *vzoru* (*template*), který definuje každý modul (modul generující data musí mít stejný výstupní vzor jako modul data přijímající) [7]. Každá položka vzoru má svůj typ a název. Typy jsou definovány centrálně a odpovídají polím z hlaviček jednotlivých síťových protokolů (např. vstupní IP adresa nebo port). Položky mohou být i dynamické, tedy mít různou velikost (např. doménové jméno extrahované z DNS paketu).

Jelikož je náš detekční systém implementován jako modul systému Nemea, jeho modul pro zpracování vstupních dat využívá výše zmíněných principů. Ze vstupního rozhraní čte UniRec záznamy se vzorem „`<COLLECTOR_FLOW>,<DNS>`“, takže každý záznam obsahuje položky o síťových tocích odpovídajících IPFIX tokům, které jsou použity u samostatné verze programu (záznam obsahuje základní data o toku i DNS data důležitá pro sofistikovanější detekční metody).

Z každého záznamu jsou pak extrahovány potřebné položky a uloženy do struktur,

[11] *TRaffic Analysis Platform* – platforma pro analýzu síťového provozu.
[12] *Unified Record* – záznam sjednoceného formátu.

se kterými dále pracují další části detekčního systému. Dle nastavených parametrů programu můžeme extrahovat pouze data toku NetFlow (potom se extrahují data odpovídající struktuře `flow` – viz tabulka 6.3, nebo rozšířená data IPFIX (extrahovaná data odpovídají struktuře `ipfix_flow` – viz tabulka 6.4).

Z jednotlivých záznamů se poté stejně jako u samostatné verze tvoří datové bloky, přičemž je i zde nutné záznamy řadit do správných bloků dle jejich koncového času.

Načítání dat a tvorbu bloků implementují třídy `netflow_bin_maker` a `ipfix_bin_maker` obsažené ve zdrojových souborech `netflowparser.cpp`, `ipfixparser.cpp` a odpovídajících hlavičkových souborech.

6.4 Detekční proces

Po zpracování vstupních dat a vytvoření odpovídajících datových bloků můžeme přistoupit k detekci nad těmito daty. Jak již bylo naznačeno v návrhu, detekční proces byl rozdělen na dva stupně (detektory), kdy první stupeň má za úkol odfiltrovat data, která s největší pravděpodobností neobsahují hledané anomálie. První detektor je tedy implementován s ohledem na rychlost tohoto filtrování, takže neprovádí žádnou hloubkovou inspekci dat. Detekci provádí nad celými bloky dat. Vybrané (podezřelé) bloky jsou zařazeny do sdílené fronty s druhým detektorem, kde čekají na své zpracování.

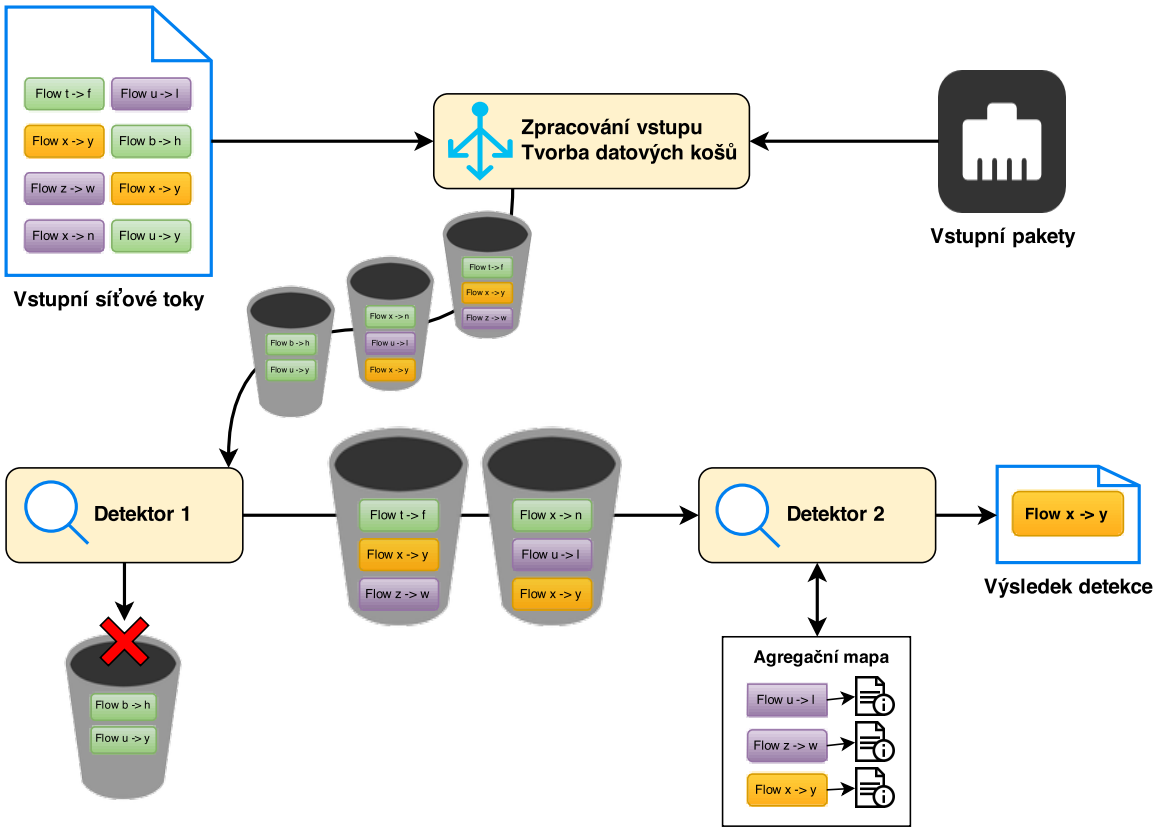
Druhý detektor následně provádí hloubkovou inspekci vybraných bloků dat. Hloubková inspekce v tomto případě znamená, že je zpracován a kontrolován obsah jednotlivých záznamů uložených v daném bloku. Konkrétní způsob zpracování a kontroly záznamů je samozřejmě vhodné přizpůsobit na míru typu anomálie, který chceme detekovat. Vhodná je např. agregace podezřelých dat a jejich následná inspekce.

V každém případě jsou ale detekční metody užité druhým detektorem mnohem náročnější na dobu zpracování než v případě prvního detektoru. Je tedy nutné s tím počítat a přizpůsobit tomu činnost prvního detektoru (aby se ve frontě před druhým detektorem nehromadilo příliš velké množství bloků čekajících na zpracování).

Výstupem druhého detektoru jsou záznamy o detekovaných anomáliích. Ty jsou ukládány do specifikovaného souboru a mají textový charakter. Každý záznam o detekci obsahuje identifikaci komunikujících stran (IPv4 nebo IPv6 adresy a porty), časové informace o útoku (zachycený počáteční a koncový čas) a seznam detekčních kritérií (z jakého důvodu byla komunikace vyhodnocena jako anomálie). Na konec výstupního souboru jsou přidány souhrnné statistiky shromážděné ze všech modulů programu (počty zpracovaných záznamů, statistika využití jednotlivých prahů apod.). Celý detekční proces ilustruje obrázek 6.2.

Pro ověření funkčnosti a efektivity navrženého systému bylo nutné vybrat vhodné anomálie a implementovat jejich detektory. Protože cílem této práce není implementace detektorů velkého množství různých anomálií, nýbrž zlepšení efektivity celého detekčního procesu, byla vybrána anomálie, k jejíž detekci lze přistupovat různými způsoby a následně porovnat jejich efektivitu. Touto anomálií je tunelování přes DNS.

K detekci tunelování přes DNS můžeme použít různé způsoby. Můžeme se pokoušet o detekci pouze s využitím NetFlow dat a sledovat pravidelnost komunikace či množství přenesených dat. Nebo můžeme detekci založit na inspekci samotných DNS dat a hledat v nich prvky typické právě pro tunelování přes DNS (délka přenášených doménových jmen, využití specifických typů dotazů apod.). Díky podpoře různých vstupních formátů dat můžeme ověřit oba způsoby detekce.



Obrázek 6.2: Ilustrace detekčního procesu.

V následujících kapitolách jsou popsány konkrétní detekční metody využité v jednotlivých detektorech, zaměřené na detekci tunelování přes DNS. Porovnání uvedených detekčních metod se věnuje kapitola 7.

6.4.1 Tvorba statistik určených pro detekci

Aby mohl být první detektor dostatečně rychlý a neprováděl inspekci jednotlivých záznamů v kontrolovaných datových blocích, je třeba hledat vyhovující detekční metody. Jednou z takových metod je detekce založená na časově závislých statistických informacích. Metoda byla popsána v článku [29], kde byla použita právě pro detekci tunelování přes DNS. Autoři článku naměřili vysokou účinnost této metody (pouze 1 % falešně pozitivních detekcí). Zároveň je její implementace relativně jednoduchá a slibuje tak dostatečnou rychlost pro použití v prvním detektoru.

Metoda funguje na principu prahování nad statistickými informacemi shromážděnými v každém datovém bloku. Práh je zároveň časově závislý (prahová hodnota se mění v závislosti na denní době), což umožňuje snížit počet falešně pozitivních detekcí a zpřesnit detekci. Shromažďovány jsou obecné informace o síťových tocích a metoda tak funguje s datovými formáty NetFlow a IPFIX. U dat ve formátu pcap není tato detekční metoda využívána.

Základem metody je seskupení síťových toků do bloků konstantní časové délky (v našem případě realizováno v modulu pro zpracování vstupních dat). Jakmile je každý blok zkompletován, jsou vypočítány souhrnné statistiky tohoto bloku (ukládání statistických dat ve

skutečnosti probíhá už při vkládání jednotlivých záznamů do bloku). Konkrétně se jedná o tyto statistické informace:

- `BPB` (*Bytes Per Bin*), tedy počet bytů v bloku (součet velikostí všech záznamů v bloku).
- `FPB` (*Flows Per Bin*), neboli počet záznamů v bloku (počet síťových toků v časovém rozmezí odpovídajícímu danému bloku).
- `BPPPB` (*Bytes Per Packet Per Bin*), tedy průměrný počet bytů v jednom paketu v rámci daného bloku (průměrná velikost jednoho paketu v rámci všech síťových toků obsažených v daném bloku).

Při detekci jsou využívány právě tyto statistiky, kdy se hodnoty daného datového bloku porovnávají s průměrnými hodnotami příslušné časové *periody*. Výpočet průměrných hodnot pro danou periodu probíhá tak, že jsou agregovány statistické informace z jednotlivých bloků, které svým časem zasahují do dané periody (např. všechny bloky dat, které nesou záznamy o síťových tocích v době od 15:00 do 15:30 daného dne). Doba trvání periody je stanovena parametrem `statistics_period` v konfiguračním souboru. Délka periody musí být stanovena tak, aby bylo možné pokrýt N periodami celý den (24 hodin). Vhodnou volbou jsou tedy hodinové či půlhodinové periody.

Výpočet průměrných hodnot pro jednotlivé periody dne musí být proveden předem nad vstupními daty, které zachycují typický provoz daného dne. V ideálním případě by tak měly být vytvořeny souhrnné statistiky pro každý den v týdnu. To je zároveň nevýhodou této metody, protože nemůžeme spustit detekci, aniž bychom měli k dispozici předem vytvořené souhrnné statistiky (detekci spustit můžeme, ale bude velmi nepřesná).

Souhrnné statistiky pro jednotlivé periody obsahují následující položky:

- `AVG BPB` (*Average Bytes Per Bin*), průměrný počet bytů v bloku vypočítaný ze všech bloků v dané periodě.
- `AVG FPB` (*Average Flows Per Bin*), průměrný počet záznamů v bloku během dané periody.
- `AVG BPPPB` (*Average Bytes Per Packet Per Bin*), průměrný počet bytů v jednom paketu v rámci všech bloků dané periody.
- `MAX BPB` Maximální hodnota `BPB` během dané periody (velikost největšího bloku dané periody).
- `MAX FPB` Maximální hodnota `FPB` během dané periody (největší počet záznamů v jednom bloku během dané periody).

Tyto souhrnné statistiky jsou poté uloženy do CSV souboru, který je následně využit při detekci (soubor obsahuje vypočítané statistiky pro jednotlivé denní periody). K výpočtu souhrnných statistik jednotlivých period je možné použít parametr programu `-s`.

```
start,end,avg bpb,avg fpb,avg bpppb,max bpb,max fpb
16:00:00,16:30:00,8729972.5000,54436.8333,125.1028,22592415,131469
16:30:00,17:00:00,7613411.0000,49312.3142,120.3719,21647361,126581
```

Tabulka 6.6: Příklad vytvořeného CSV souboru se souhrnnými statistikami jednotlivých časových period.

Tvorbu souhrnných statistik implementuje třída `netflow_statistics_maker`, jejíž kód je umístěn v souborech `netflowstatisticsmaker.cpp` a `netflowstatisticsmaker.h`.

6.4.2 Detektor 1 pro detekci DNS tunelování

Úkolem prvního detektoru je vyfiltrovat bloky dat, ve kterých se pravděpodobně nachází síťový provoz, odpovídající tunelování přes DNS. Aby mohl tuto činnost vykonávat rychle a bez nutnosti inspekce obsahu jednotlivých datových bloků, využívá k detekci statistických informací. Pro tuto činnost se spokojí pouze s obecnými daty o síťových tocích a je tak možné využít stejnou implementaci prvního detektoru jak pro NetFlow, tak pro IPFIX data.

Detekce probíhá tak, že detektor ve smyčce načítá bloky dat ze sdílené fronty, do které je přidává modul pro zpracování vstupních dat. Ten zároveň během tvorby jednotlivých bloků přímo počítá i statistiky, které ukládá ke každému bloku. To má tu výhodu, že detektor nemusí znovu procházet obsah jednotlivých bloků, aby tyto statistiky vypočítal. Ušetříme tak výpočetní čas a detekční proces zefektivníme. Na základě těchto statistik a zvoleného *detekčního schématu* je potom prováděna detekce. Detekované bloky jsou umístěny do sdílené fronty s druhým detektorem, který je následně zpracovává, zatímco bloky, které nebyly detekovány (a pravděpodobně tedy neobsahují data hledaného útoku) jsou smazány a nejsou dále nijak zpracovávány.

První detektor implementuje celkem dvě detekční schémata. Které detekční schéma bude použito, je definováno v konfiguračním souboru parametrem `d1_detection_schema`.

První detekční schéma využívá předem vytvořená, časově závislá statistická data, která byla vytvořena dle popisu v kapitole 6.4.1. V tomto případě musí být parametrem programu `-t` stanoven soubor obsahující souhrnná statistická data jednotlivých denních period. Ten musí být načten ještě před začátkem samotné detekce. Zpracování statistického souboru přitom má některá specifika.

Statistický soubor nemusí obsahovat všechny denní periody (pokud jsme vytvořili souhrnné statistiky pouze nad určitým vzorkem denních dat). Protože by chybějící statistická data pro požadovaný čas způsobovala problémy (detekce by vůbec nemohla proběhnout), jsou chybějící časové periody doplněny. Statistická data existujících period jsou zprůměrována a následně doplněna do chybějících period. Je jasné, že průměr všech existujících period nebude pro detekci moc přesný, nicméně alespoň bude možné detekci spustit bez problémů. Pro přesnou detekci je ale samozřejmě nejvhodnější mít k dispozici kompletní denní data či data intervalu, ve kterém provádíme detekci.

Během detekčního procesu pak musíme pro každý datový blok najít příslušná souhrnná statistická data. Musíme tedy dle koncového času bloku najít data odpovídající periody. Je důležité, aby byla statistická data vytvořena se stejným nastavením délky trvání datového bloku jako u probíhající detekce, jinak by vypočítané statistiky neodpovídaly zpracovávaným blokům a detekce by byla nepřesná.

Protože víme, že po sobě jdoucí bloky jsou seřazeny dle jejich koncového času a zároveň máme k dispozici seznam statistických dat pro všechny denní periody, můžeme optimalizovat jejich vyhledávání. Nejprve tedy zkoušíme periodu, do které spadal poslední zpracovaný blok (dle koncového času bloku a koncového času periody). Protože bloky po sobě následují, je velmi pravděpodobné, že nově zpracovávaný blok bude patřit do stejné periody. Pokud ne, jednoduše hledáme v následující periodě. Jakmile se přepneme do další periody, uložíme její index (data period jsou uložena ve vektoru) a ten použijeme při dalším vyhledávání.

Nemusíme tak pro každý blok procházet celý seznam existujících period. Vyhledání příslušné periody proběhne maximálně ve dvou krocích.

Jakmile máme průměrné statistiky pro odpovídající časovou periodu, můžeme přistoupit k detekci. Detekce spočívá v porovnání vypočítaných statistik zpracovávaného bloku s průměrnými statistikami pro danou denní periodu. Protože nemůžeme porovnávat tyto dvě hodnoty přímo, musí uživatel zvolit prahovou hodnotu, o kterou musí být konkrétní statistika zpracovávaného bloku překročena oproti souhrnné statistice. Aby tedy byl blok detekován, musí být splněna následující podmínka:

```
statistika zpracovávaného bloku >=
(průměrná statistika pro danou periodu + zvolený práh)
```

Konkrétně se využívají statistiky bloku `BPB`, `FPB` a `BPPPB`, viz 6.4.1. Práh pro každou z těchto položek lze definovat v konfiguračním souboru. Aby byl blok detekován, stačí aby byla překročena pouze jedna z uvedených statistik. Zároveň se zaznamenává, kolikrát byl který práh splněn. Tyto údaje jsou poté uvedeny na konci výstupního souboru.

Druhé detekční schéma pokrývá situaci, kdy nemáme nebo nechceme využít časově závislé souhrnné statistiky. Tato situace nám ale bohužel neposkytuje příliš možností, pokud chceme detekci provádět bez hlubší inspekce datových bloků. I v tomto případě tedy musíme využít statistik, které byly vypočítány při tvorbě bloků.

Kvůli jejich charakteristice nemůžeme využít vypočítané `BPB` a `FPB`, protože nemáme žádnou referenční hodnotu, se kterou bychom je mohli porovnávat (tyto parametry závisí na hustotě typického síťového provozu, který prochází místem detekce), a pro stanovení vhodné hraniční hodnoty bychom museli stejně provést sběr průměrných statistik. Zbývá nám jediná možnost, a to `BPPPB`, tedy průměrná velikost paketů v daném bloku. Tuto statistiku můžeme použít i samostatně, protože můžeme zhruba stanovit hraniční hodnotu i bez souhrnného měření (např. na základě měření provedených v příloze E).

Při stanovení prahu ale musíme i zde myslet na množství síťového provozu, které projde místem detekce za čas odpovídající jednomu bloku. S velkým množstvím síťového provozu se totiž nárůst průměrné velikosti paketů způsobený výskytem tunelování přes DNS či jiné anomálie rovnoměrně rozprostře do normálního provozu a průměrný nárůst velikostí v rámci celého bloku tak může být menší, než bychom očekávali. Tomu ale můžeme do jisté míry čelit například zmenšením bloků, takže podezřelý síťový provoz více vynikne.

První detektor je implementován třídou `netflow_detector1`, která je definována a deklarována v souborech `netflowdetector1.cpp` a `netflowdetector1.h`. Pro načítání a zpracování statistických souborů slouží třída `netflow_statistics`, která se nachází ve zdrojových souborech `netflowstatistics.cpp` a `netflowstatistics.h`.

6.4.3 Detektor 2 pro detekci DNS tunelování

Zatímco první detektor se snaží vyhnout zpracování jednotlivých záznamů v datových blocích, druhý detektor tuto činnost provádí. První detektor vybere bloky, ve kterých by se hledaná anomálie mohla vyskytovat. Úkolem druhého detektoru je prověřit obsah těchto bloků a hledanou anomálii najít (potřebujeme identifikovat konkrétní komunikaci dvojice IP adres). Zároveň druhý detektor zpracovává méně dat, takže si může dovolit hloubkovou inspekci dat i přes nárůst doby zpracování.

Výjimkou je v tomto ohledu pcap detektor, který nemá první detektor, nýbrž pouze druhý detektor. K tomu bylo přistoupeno z důvodu rozdílné filozofie oproti ostatním formátům (pakety u pcap versus síťové toky u NetFlow a IPFIX). Zároveň se nepředpokládá, že by probíhala kontrola velkého množství pcap dat, protože na to jsou vhodné právě zbylé dva formáty. Pcap detektor tedy implementuje rovnou hloubkovou inspekci všech vstupních dat. Tato skutečnost nám ale může zároveň poskytnout srovnání rychlostí dvou přístupů, tedy určení efektivnějšího způsobu detekce (dva versus jeden detektor). Teoreticky ale nic nebrání vytvoření prvního detektoru i pro formát pcap (musel by využívat statistiky `BPB` a `BPPPB`).

Protože je každý formát dat specifický, existují celkem tři implementace druhého detektoru (pro IPFIX, NetFlow a pcap data). V principu jsou ale všechny verze založeny na stejné myšlence. Tou je agregace vybraných podezřelých záznamů z každého bloku (nebo i více bloků) a následná inspekce těchto agregovaných dat. K realizaci tohoto schématu jsou zapotřebí dvě sady prahů.

První sada slouží k výběru konkrétních paketů nebo toků z každého bloku (zde se uplatní techniky pro detekci jednoho konkrétního záznamu, jako např. příliš dlouhé doménové jméno apod.). Na základě této první detekce jsou agregována specifická data daného záznamu.

Druhá sada prahů se vztahuje na agregované záznamy a teprve v této druhé fázi detekce je komunikace mezi dvěma stranami označena jako anomálie, nebo není. Zde použité prahy se týkají např. počtu paketů mezi komunikujícími stranami či velikosti přenesených dat.

Agregace podezřelých záznamů

Pro agregaci vybraných záznamů ve druhých detektorech je použito *asociativní pole*[13], které je implementováno ve formě *hašovací tabulky* dostupné v novém standardu C++ jako kontejner `std::unordered_map`. Ta mapuje záznamy se stejným klíčem na hodnotu příslušející danému klíči. Proto ji budeme dále označovat jako „mapu“.

Co je tedy v našem případě klíčem? Je to dvojice `(src IP, dst IP)`, tedy zdrojová a cílová IP adresa každého záznamu. Zde by jistě čtenáře napadlo, že bychom měli přidat i zdrojový a cílový port, abychom mohli spolehlivě odlišit jednotlivé toky. Porty ale schválně nejsou součástí klíče. Je to z toho důvodu, že tunelování může probíhat přes *lokální resolver*, a nebo přes *ne-lokální resolver*. Lokální resolver je v našem případě DNS server v lokální síti, který přeposílá naše tunelovací dotazy dále na tunelovací server. Používáme tedy v každém dotazu jiné zdrojové porty, takže by zachycená komunikace byla roztroušena po agregační mapě do mnoha různých záznamů. U ne-lokálního resolveru komunikujeme s tunelovacím serverem přímo a používáme tak stále stejné porty. Celá tunelovaná komunikace je v tomto případě uložena do jednoho dlouhého síťového toku a v agregační mapě by byla uložena v jednom záznamu. Porty tedy nejsou součástí klíče, nicméně jsou součástí agregovaných dat (z nich pak můžeme určit jednotlivé strany komunikace – klienta a server).

Aby mohl být klíč univerzální pro použití IPv4 i IPv6 adres, je implementován jako struktura `flow_key`, do které je možné vložit oba typy adres. Bylo nutné přetížit její operátor porovnání, abychom mohli porovnávat různé klíče (tedy dvojice IP adres). Pro správnou funkci hašovací mapy s tímto klíčem bylo také nutné implementovat kombinaci obou položek klíče do jedné hodnoty (k tomu byla využita C++ implementace hašovací funkce `std::hash` a také funkce `hash_combine`, která byla převzata ze sady rozšiřujících knihoven pro C++ *Boost* [34]).

[13] Pole záznamů ve tvaru `<klíč, hodnota>`, ve kterém můžeme vyhledávat podle klíče.

Hodnotou jsou pak agregovaná data ukládaná v instanci třídy `storage_data`. Data jsou agregována do následujících položek:

`occurences`	Celkový počet výskytů.
`packets`	Celkový počet paketů.
`bytes`	Celkový počet bytů.
`queries`	Celkový počet DNS dotazů (odpovědi nejsou počítány).
`rdlen_total`	Celková velikost přenesených dat v sekci RDATA DNS paketu.
`namelen_total`	Celková délka použitých doménových jmen (pokud ji podělíme počtem paketů, dostaneme průměrnou délku přenášeného doménového jména).
`bytes_only_queries`	Velikost přenesených dat pouze v DNS dotazech.
`srcports_map`	Mapa zdrojových portů spolu s počty jejich výskytů.
`dstports_map`	Mapa cílových portů spolu s počty jejich výskytů.
`qtypes_map`	Mapa použitých typů DNS dotazů s počty jejich výskytů.
`domains_map`	Mapa použitých doménových jmen s počty jejich výskytů. Do této mapy nejsou ukládána celá doménová jména, ale pouze doménová jména zkrácená na určitý počet subdomén (počet subdomén lze změnit v konfiguračním souboru). Zkrácená doménová jména jsou kontrolována vůči seznamu povolených domén.

Z uvedených agregovaných informací lze ještě dodatečně vypočítat nejrůznější statistiky, které také využíváme k detekci.

Samotná agregační mapa je umístěna ve třídě `ip_storage`, která vlastně představuje univerzální datový blok (může nést seznam záznamů daného bloku, nebo agregovat jednotlivé záznamy do mapy). Definována a deklarována je v souborech `ip_storage.cpp` a `ip_storage.h`

Agregace do mapy má výhodu v jednoduchosti a dobré funkčnosti. Nevýhodou ale může být menší rychlost agregace v případě agregace velkého množství položek (velká mapa). Výkonnost také klesá při použití IPv6 adres, protože porovnání dvojic IPv6 adres je náročnější než u IPv4 adres. Pro další zefektivnění detekce je tak vhodné hledat lepší a rychlejší způsoby agregace potřebných dat.

Detekční pravidla IPFIX

Detektor používající IPFIX data má na výběr celkem ze čtyř detekčních schémat. Volba detekčního schématu probíhá na základě parametru `d2_detection_schema` uvedeného v konfiguračním souboru. Jednotlivá detekční schémata se liší v tom, jaké záznamy jsou agregovány do agregační mapy. Před agregací jsou nejprve všechny agregované položky ověřeny vůči seznamu povolených domén (pokud zkrácené doménové jméno vyhovuje nějaké doméně ze seznamu povolených domén, není daný záznam agregován a tedy ani detekován).

První detekční schéma agreguje všechny záznamy bloku bez výjimky. Nejsou tak vůbec podrobně zkoumány jednotlivé záznamy, ale detekce probíhá pouze na základě prahů, vztahujících se na agregované záznamy. Toto schéma má tu nevýhodu, že agregujeme i položky, které bychom mohli s velkou pravděpodobností vyřadit, kdybychom použili prahy pro jednotlivé záznamy. Zároveň toto schéma není vhodné pro velké množství záznamů v bloku, protože agregace je potom pomalá.

Druhé detekční schéma agreguje pouze záznamy, které jsou označeny jako podezřelé první sadou prahů. Každý záznam tedy kontrolujeme a vybíráme, které záznamy budeme agregovat. Toto schéma sníží počet agregovaných položek, čímž agregaci urychlíme, nicméně musíme provádět inspekci jednotlivých záznamů, takže urychlení nemusí být tak velké (zejména pro malý počet položek v bloku).

Třetí detekční schéma agreguje pouze specifické typy DNS dotazů (a odpovědí), které jsou nejčastěji využívány pro tunelování přes DNS. Jedná se o záznamy typu MX, NS, TXT, NULL a CNAME. Nevýhodou je, že přicházíme o možnost detekce tunelování, které využívá ostatní typy záznamů (např. A, AAAA). Velkou výhodou je ale minimalizace počtu agregovaných položek, protože tyto typy záznamů tvoří malou část běžného DNS provozu. Typy záznamů byly zvoleny s ohledem na analýzu nástrojů pro tunelování přes DNS, která je uvedena v příloze E.

Čtvrté detekční schéma kombinuje předchozí dva přístupy. Agregovány jsou specifické typy dotazů (stejně jako ve třetím schématu), které jsou zároveň označeny první sadou prahů (stejně jako u druhého schématu). Výsledkem je ještě menší počet záznamů, které agregujeme, ale zvyšuje se náročnost zpracování každého záznamu.

Pokud využíváme druhé nebo čtvrté detekční schéma, jsou u každého záznamu kontrolovány některé charakteristiky. Pro porovnání slouží prahy, jejichž hodnoty jsou definovány uživatelem v konfiguračním souboru. Aby byl záznam agregován, stačí splnit libovolné následující kritérium:

1. Počet bytů v jednom síťovém toku.
2. Počet paketů v jednom síťovém toku.
3. Velikost samotného (jediného) paketu.
4. Nízká hodnota TTL DNS odpovědi (pouze u paketů s odpovědí).
5. Velikost DNS RDATA (pouze u paketů s odpovědí).
6. Velikost paketu s DNS dotazem (pouze u paketů s dotazy).
7. Příliš dlouhé doménové jméno (dotazy i odpovědi).
8. Výskyt subdomén stejné délky (jsou kontrolovány délky jednotlivých po sobě jdoucích subdomén – pokud po sobě následuje několik stejně dlouhých subdomén, je možné, že se jedná o tunelovaný provoz, viz analýza nástrojů `DNScat` či `Dns2tcp` v příloze E). Je stanovena minimální délka subdomén tak, abychom nevybírali krátké subdomény, kde je větší pravděpodobnost jejich stejné délky (např. `www.att.com`).
9. Výskyt *ne-ASCII*[14] či netisknutelných znaků v doménovém jméně či sekci RDATA. Je vypočítán poměr mezi normálními znaky a speciálními znaky (netisknutelné znaky

[14] ASCII (*American Standard Code for Information Interchange*) tabulka definuje kódování znaků anglické abecedy a dalších speciálních znaků.

a netypické znaky[15]). Po překročení stanoveného poměru (více jak 30 % speciálních znaků) je kritérium splněno. Speciální znaky v doménových jménech se vyskytovaly např. při použití nástroje `iodine` se záznamy typu NULL.

Do tohoto seznamu by zajisté bylo možné doplnit další detekční kritéria, nicméně je třeba myslet na jejich efektivitu (doba výpočtu, paměťové nároky). Proto byly vyloučeny některé popsané způsoby detekce DNS tunelování jako frekvenční analýza doménových jmen ([14]), vyhledávání n-gramů ([15]) nebo počet unikátních subdomén (paměťová náročnost a doba porovnání při ukládání kompletních subdomén).

Jakmile jsou záznamy agregovány do agregační mapy, můžeme přistoupit k závěrečné fázi detekce. Nejprve je ale nutné si ujasnit, kdy nastane ten moment, ve kterém budeme analyzovat agregovaná data. Analýza agregovaných dat musí probíhat v určitých intervalech, protože jinak by velmi rostly paměťové nároky programu (uložení velkého množství dat v paměti) a zároveň by narůstala doba agregace, která by zpomalovala celý detekční proces. Jednoduchou možností by tedy byla periodická analýza po uplynutí určitého času (nebo v našem případě počtu bloků).

My ale můžeme využít toho, že první detektor označuje podezřelé bloky, které zpracováváme ve druhém detektoru. Pokud totiž první detektor správně označí všechny bloky, během nichž probíhala komunikace, kterou chceme detekovat, tvoří tyto bloky navazující posloupnost. Dle počátečních a koncových časů jednotlivých bloků tedy můžeme určit, zda bloky následují hned po sobě či nikoliv (je mezi nimi časová mezera). Proto inspekci agregovaných dat provádíme po každém přerušení posloupnosti po sobě jdoucích bloků (inspekce tedy může proběhnout po jednom, ale i více blocích). Tímto způsobem máme jistotu, že v agregační mapě bude obsažena celá komunikace, kterou se snažíme odhalit. Zároveň je implementována i periodická kontrola po agregaci maximálního počtu bloků. Ta zabrání případům, kdy první detektor bude detekovat velmi dlouhou posloupnost bloků, což by mohlo značně zpomalit agregaci, nebo dokonce úplně vyčerpat dostupnou operační paměť. Maximální počet agregovaných bloků lze nastavit v konfiguračním souboru.

Inspekce agregační mapy probíhá průchodem celé mapy a aplikací poslední sady detekčních prahů na jednotlivé záznamy. Konkrétní hodnoty jednotlivých prahů lze nastavit v konfiguračním souboru. Kontrolují se následující kritéria:

1. Průměrná velikost paketu.
2. Celková velikost přenesených dat.
3. Počet výskytů v agregační mapě (kolikrát jsme přidali data k danému klíči – dvojici komunikujících IP adres).
4. Celkový počet DNS dotazů.
5. Velikost přenesených dat pouze v rámci DNS dotazů (odpovědi nejsou započítány).
6. Průměrná délka doménového jména.
7. Nejvyšší počet přístupů na jednu doménu (doménou se myslí původní FQDN zkrácené na definovaný počet subdomén tak, jak jsou ukládány do agregační mapy).

[15] Znaky, které nejsou typickým obsahem doménových jmen. Patří sem tyto znaky: `[]{}()<>$!':#*,%/&‘´˜§¨-?^+=|@;°˜`.

8. Nejvyšší počet použití jednoho typu DNS dotazů (vhodné zejména u detekčního schématu, kdy agregujeme pouze specifické typy dotazů).

Aby byl záznam označen jako detekovaný, musí být splněno minimální stanovené množství uvedených kritérií (lze nastavit v konfiguračním souboru). Podrobnosti o detekovaných záznamech jsou pak ukládány do výstupního souboru v textové podobě. Ukládá se identifikace obou komunikujících stran – IP adresy a porty (pokud je komunikace vedena na více různých portech, je místo konkrétního portu uveden řetězec „XXXXX“), seznam využitých domén spolu s počtem jejich využití, seznam použitých typů DNS dotazů s počtem jejich využití a seznam detekčních kritérií, která byla splněna. Příklad detekovaného záznamu je ukázán v tabulce 6.7.

Po provedení prvních měření nad reálnými daty byla implementována ještě speciální možnost výpisu detekovaných záznamů pouze pokud se jedná o komunikaci z klienta na DNS server (tedy zdrojový port není portem 53). Tato možnost je velmi efektivní pro detekci právě DNS tunelování, protože u něj typicky probíhá velké množství komunikace z klienta na server, která zároveň splní detekční kritéria. Můžeme tak efektivně odfiltrovat ty případy, kdy DNS server v rámci normální komunikace odpovídá velkým množstvím paketů či velkými pakety (které jsou následně detekovány) a snížit tak množství falešně pozitivních detekcí. Aktivaci této možnosti lze provést v konfiguračním souboru.

```
2015-05-06 10:19:15 - 2015-05-06 10:22:37
172.16.1.1:54732 -> 172.16.1.5:53
Used domains:
   tunnel.tunneltest.cz   1156x
Used query types:
   TXT   1156x
Detection criteria (matched 6/8):
   avg packet size = 212.673 B (detection threshold = 170), packets: 1156
   occurences in map = 1156 (detection threshold = 110)
   total number of queries = 1156 (detection threshold = 200)
   avg DNS query len = 166.673 (detection threshold = 50), packets: 1156
   most used domain count = 1156 (detection threshold = 110)
   most used qType count = 1156 (detection threshold = 110)
```

Tabulka 6.7: Příklad záznamu o detekci.

Jakmile je provedena inspekce agregovaných záznamů, je celá agregační mapa vymazána a můžeme pokračovat ve zpracování další sekvence datových bloků.

Všechny uvedené detekční metody implementuje třída `ipfix_detector2`, která je definována a deklarována v souborech `ipfixdetector.cpp` a `ipfixdetector.h`.

Detekční pravidla pcap

Pcap detektor využívá stejná detekční schémata, jako IPFIX detektor. Protože se ale jedná o zpracování paketů místo síťových toků a navíc mu nepředchází první detektor, je seznam kontrolovaných charakteristik jednotlivých paketů trochu kratší:

1. Velikost samotného (jediného) paketu.
2. Nízká hodnota TTL DNS odpovědi (pouze u paketů s odpovědí).
3. Velikost DNS RDATA (pouze u paketů s odpovědí).
4. Velikost paketu s DNS dotazem (pouze u paketů s dotazy).
5. Příliš dlouhé doménové jméno (dotazy i odpovědi).
6. Výskyt subdomén stejné délky.
7. Výskyt *ne-ASCII* znaků v doménovém jméně či sekci RDATA.

Jelikož chybí první detektor, nemůžeme provádět inspekci agregační mapy po přerušení sekvence bloků. Inspekci tedy provádíme pravidelně po N agregovaných blocích (hodnotu N lze nastavit v konfiguračním souboru a jedná se o maximální počet po sobě agregovaných bloků, který je využit také u NetFlow a IPFIX detektorů).

Při inspekci agregační mapy se kontroluje stejná sada kritérií jako u IPFIX detektoru (agregujeme stejná data jako u IPFIX detektoru).

Všechny uvedené detekční metody implementuje třída `pcap_detector1`, která je definována a deklarována v souborech `pcapdetector1.cpp` a `pcapdetector1.h`.

Detekční pravidla NetFlow

Protože u NetFlow dat nemáme k dispozici DNS data jednotlivých paketů, můžeme oproti jiným formátům využít pouze první dvě detekční schémata (agregace všech záznamů nebo agregace vybraných záznamů na základě první sady prahů).

Co se detekčních prahů týče, má NetFlow také svá specifika. Jelikož jsou u NetFlow dat toky vytvářeny pouze dle portů a IP adres, může být průběh celého síťového útoku zachycen v jednom či několika málo tocích. U těch se tak oproti IPFIX tokům (kdy jeden tok v zásadě odpovídá jednomu DNS paketu) projeví další vlastnost, a sice délka toku (doba trvání). U normální DNS komunikace je typické, že délka takového toku bude maximálně v řádu sekund, zatímco u tunelování se může délka toku pohybovat až v řádu minut[16]. Proto můžeme v konfiguračním souboru nastavit další detekční práh, a to délku jednoho toku.

Seznam kontrolovaných charakteristik jednotlivých záznamů je díky absenci DNS dat i tak kratší, než v předchozím případě:

1. Počet bytů v jednom síťovém toku.
2. Počet paketů v jednom síťovém toku.
3. Velikost samotného (jediného) paketu.
4. Doba trvání toku.

U inspekce agregační mapy je situace díky absenci DNS dat také mnohem jednodušší. Stejně jako v předchozím případě musíme zahrnout i kontrolu na délku jediného toku, protože i jediný tok může obsahovat záznam celého útoku. Pro tuto kontrolu se použije totožná prahová hodnota jako u kontroly jednotlivých záznamů. Kontrolují se následující kritéria:

[16] Dle experimentálně naměřených výsledků.

1. Průměrná velikost paketu.
2. Celková velikost přenesených dat.
3. Počet výskytů v agregační mapě (kolikrát jsme přidali data k danému klíči – dvojici komunikujících IP adres).
4. Doba trvání jednoho toku.

Díky absenci DNS dat je detekce v rámci druhého detektoru NetFlow poměrně rychlá, nicméně její přesnost oproti předchozím detektorům bude rozhodně horší.

Uvedené detekční metody implementuje třída `netflow_detector2`, která je definována a deklarována v souborech `netflowdetector2.cpp` a `netflowdetector2.h`.

6.5 Možnosti rozšíření programu

Detekční systém v současné podobě umožňuje přidávání dalších funkcí a nových možností. Protože je aplikace sestavena z jednotlivých modulů, nabízí se možnost jejich doplnění. Je možné implementovat nové moduly pro zpracování dalších vstupních formátů (formáty pro práci se síťovými toky jako např. *sFlow*), stejně jako nové detekční moduly.

Nové detekční moduly, umožňující detekci dalších anomálií v DNS datech, lze snadno vytvořit odvozením od již existujících modulů. Výhodou je, že můžeme beze změny využít existující moduly pro zpracování vstupních dat a soustředit se tedy pouze na detekční část implementace. Nové moduly v tomto případě pouze musí využít existující rozhraní v podobě sdílených front s datovými bloky. Pokud by vyžadovaly jiný postup zpracování, musely by být implementovány i nové moduly pro zpracování vstupu (nebo upraveny ty stávající). V závislosti na detekční metodě je možné využít existující funkčnosti, jako např. implementace agregační mapy, a tu pak použít v nových modulech. Další informace k postupu implementace nových modulů je možno nalézt v příloze C.

Co se úpravy stávajících modulů týče, přichází v úvahu doplnění dalších detekčních technik do stávajících detektorů tunelování přes DNS. Vhodné by například bylo přidat alternativní jednoduchou detekční metodu do prvního detektoru NetFlow, která by využívala jiného přístupu než statistické detekce. Také u druhých detektorů je prostor k přidání nových detekčních technik pro detekci jednotlivých paketů či toků, nebo vytvoření jiného způsobu agregace jednotlivých záznamů.

Pro reálné nasazení na vysokokapacitních linkách či uzlech s velkým objemem procházejících dat by pak bylo vhodné provést další revizi kódu s cílem urychlení výpočtů a operací. Asi by také bylo nutné implementovat jiný způsob agregace záznamů u druhého detektoru (hašovací mapa pravděpodobně není nejrychlejší způsob agregace), nebo urychlit způsob řazení toků dle času.

Vhodnou nástavbou celého detekčního systému by pak byl nástroj pro agregaci a zobrazení výsledků detekce (např. ve formě programu s grafickým uživatelským rozhraním). Tento nástroj by zpracoval výstupní soubor a přehledně zobrazil jeho obsah (mohl by například umožňovat pokročilé vyhledávání). Další možností by bylo napojení výstupu na firewall a automatická tvorba filtrovacích pravidel na základě výsledků detekce.

Kapitola 7

Testování

V této kapitole je popsán proces testování celého detekčního systému. Testování bylo zaměřeno na funkčnost (jestli systém správně detekuje hrozby) a na vyhodnocení efektivity jednotlivých detekčních technik (porovnání jejich využití při detekci, rozdíly v rychlosti a podobně).

Testování probíhalo ve virtualizovaném prostředí systému *Ubuntu* na počítači s procesorem *Intel Core 2 Quad Q9000* (čtyři jádra, 2 GHz) a 4 GB operační paměti. Nejedná se tak o nejmodernější technické vybavení a při interpretaci výsledků některých měření (zejména měření doby výpočtu) je třeba na tuto skutečnost brát ohled.

7.1 Zdroje vstupních dat

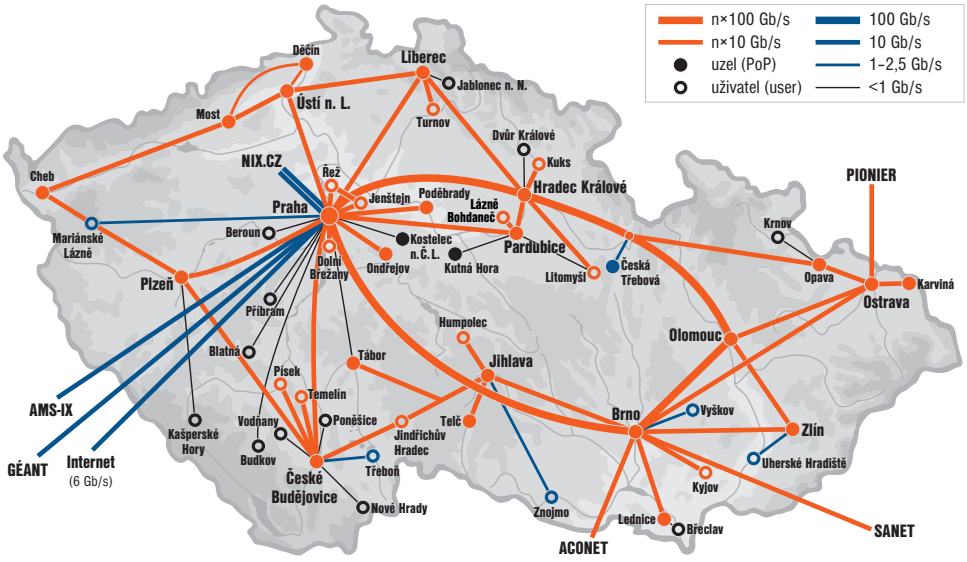
Před začátkem samotného testování bylo nejprve nutné získat vzorky vstupních dat. Bylo třeba získat jak vzorek běžného DNS provozu z reálné sítě, tak vzorky útoků (v našem případě tunelování přes DNS). Následně bylo nutné normální data i data útoku zkombinovat tak, jakoby útok probíhal během běžného síťového provozu.

7.1.1 Běžný DNS provoz

Aby bylo testování smysluplné, bylo třeba získat reálná data ze sítě, kde je velké množství DNS provozu. Pro tyto účely mi byly vedoucím práce poskytnuty ukázkové soubory anonymizovaných DNS dat získané v síti *CESNET2* (více informací o této síti naleznete v [19], topologie sítě je znázorněna na obrázku 7.1).

Konkrétně se jednalo o data zachycená na přístupových bodech mezi sítí CESNET2 a okolními, do ní připojenými sítěmi (AMS-IX, GEANT, SANET, ACONET a další). Data byla získána v rámci projektu *Liberouter*, který je se společností CESNET úzce provázán. Také mi byl umožněn přístup na server `benefizio.liberouter.org`, na kterém je nasazen a spuštěn modul pro export IPFIX DNS dat [39]. Bylo tak možné vytvořit nejen testovací CSV soubory, ale i otestovat v reálném prostředí verzi detekčního systému, určenou pro projekt Nemea.

Pro účely testování lokální verze programu byly vytvořeny soubory s NetFlow daty, zachycující síťové toky DNS v průběhu jednoho celého dne a také IPFIX CSV soubor, obsahující DNS data zachycená během zhruba 15 minut (viz tabulka 7.1). U IPFIX CSV souboru je poměrně krátká časová délka opodstatněna velikostí souboru, kdy soubor s toky zachycenými během jedné hodiny měl velikost zhruba 5,5 GB. Protože by pro testování bylo zbytečně náročné pracovat s tak velkým souborem, byl vytvořen uvedený 15minutový soubor.



Obrázek 7.1: Topologie sítě CESNET2. Obrázek převzat z [19].

Soubor	Typ	Velikost	Počet toků	Délka
nfcapd_1_dns	`nfcapd` binární soubor	663,5 MB	11 380 395	cca 12 hodin
nfcapd_2_dns	`nfcapd` binární soubor	401,6 MB	6 933 186	cca 12 hodin
ipfix_dns	IPFIX CSV soubor	1 364,2 MB	10 140 951	cca 15 minut

Tabulka 7.1: Výchozí soubory s normálním DNS provozem.

7.1.2 Útoky

Vzhledem k tomu, že se nelze spoléhat na výskyt útoku v běžné komunikaci, musely být vytvořeny testovací data obsahující útok. Protože jsem potřeboval mít kontrolu nad obsahem a typem útoku, vytvořil jsem virtuální testovací prostředí pro simulaci tunelování přes DNS. V něm byl nasazen DNS server a další dva počítače, které sloužily k simulaci tunelovacího klienta a serveru. Více o testovacím prostředí se můžete dočíst v kapitole E.1.

Tunelování probíhalo s využitím různých volně dostupných nástrojů pro tyto účely. Konkrétně se jednalo o nástroje `tcp-over-dns`, `dnscat`, `DNScat`, `Dns2tcp` a `iodine`. Charakteristiky všech nástrojů a další informace naleznete v příloze E.

Průběh tunelování byl zachycen síťovým analyzátorem *Wireshark* a následně uložen do souborů ve formátu pcap. Jsou zachyceny typické činnosti, ke kterým se tunelování přes DNS používá, tedy SSH komunikace (kterou může útočník využít v rámci řídicího kanálu pro malware), nebo prohlížení webu (používá se, pokud chceme obejít placený přístup k Internetu). Přehled zachycených souborů je uveden v tabulce 7.2.

Protože pcap soubory jsou pouze jedním z možných vstupních formátů, bylo nutné jejich obsah převést. Převod do NetFlow proběhl s využitím nástroje *FlowTraq Flow Exporter* [31], který podporuje export NetFlow dat z pcap souboru. NetFlow data byla následně zachycena a uložena kolektorem `nfcapd` do binárních souborů, které již můžeme využít na vstupu (nebo je převést do CSV).

Převod do formátu IPFIX CSV provádí program `pcaptoipfixcsv`, který využívá sou-

Název souboru (.pcap)	Paketů	Velikost	Délka	Popis
dns2tcp_ssh_TXT	4 618	843 kB	2 min 35 s	SSH komunikace, typ dotazů: TXT.
dns2tcp_web_TXT	51 312	14 432 kB	3 min 37 s	Prohlížení webu, typ dotazů: TXT.
dnscat_bash_CNAME	498	101 kB	4 min 13 s	SSH komunikace, typ dotazů: CNAME.
dnscat_bash_NS	262	60 kB	2 min 5 s	SSH komunikace, typ dotazů: NS.
dnscat_file_MX	260	69 kB	2 min 14 s	Přenos souborů, typ dotazů: MX.
DNScatB_ssh_CNAME	1 726	437 kB	3 min 16 s	SSH komunikace, typ dotazů: CNAME.
DNScatB_web_CNAME	1 956	280 kB	12 min 55 s	SSH komunikace a prohlížení webu, typ dotazů: CNAME.
tcp-over-dns_ssh_TXT	1 148	199 kB	2 min 30 s	SSH komunikace, typ dotazů: TXT.
tcp-over-dns_web_TXT	16 856	3 631 kB	13 min 7 s	Prohlížení webu, typ dotazů: TXT.
iodine_ssh_RAW	227	30 kB	2 min 57 s	SSH komunikace, typ dotazů: NULL.
iodine_ssh2_NULL	1 126	303 kB	1 min 18 s	SSH komunikace, typ dotazů: NULL.
iodine_web_RAW	9 353	6 323 kB	8 min 37 s	Prohlížení webu, typ dotazů: NULL. Použity *raw* pakety.
iodine_web2_RAW	5 622	3 932 kB	2 min 52 s	Prohlížení webu, typ dotazů: NULL. Použity *raw* pakety.
iodine_web3_NULL	9 122	4 992 kB	3 min 15 s	Prohlížení webu, typ dotazů: NULL.

Tabulka 7.2: Testovací soubory se zachyceným tunelováním přes DNS.

části implementovaného detekčního systému a jeho kód můžete nalézt v souboru `pcapto-ipfixcsv.cpp`. Jeho překlad je možné spustit příkazem `make convertor` (v základu není automaticky přeložen). Program očekává dva parametry (`-i` *vstupní soubor* a `-o` *výstupní soubor*). Výstupem je CSV soubor odpovídající očekávanému formátu IPFIX (exportuje se každý paket jako samostatný tok).

7.1.3 Sloučení útoku a normálního provozu

Poslední fází přípravy testovacích souborů bylo sloučení normálních dat a dat útoku. Nejjednodušší variantou bylo pracovat s textovými CSV formáty (u NetFlow i IPFIX dat). Pro tyto účely jsem tedy vytvořil *skript* `merger.py`. Ten je napsán v jazyce *Python*, který je pro úlohy podobného typu velmi vhodný (umožňuje jednoduchou manipulaci s řetězci a obsahuje knihovny pro práci přímo s CSV soubory).

Skript provádí sloučení dvou CSV souborů do jednoho výstupního podle specifikovaného času. Uživatel jako parametr skriptu specifikuje čas začátku útoku. U dat útoku jsou pak nahrazeny časové údaje jednotlivých toků datem odpovídajícím normálnímu provozu (datum je přečteno z prvního záznamu normálního síťového provozu). Čas je nahrazen kombinací uživatelem specifikovaného počátečního času a relativního času konkrétního toku od začátku souboru (je uložen čas prvního záznamu útoku a od něj se poté vypočítá rozdíl, který se přidá k počátečnímu času). Tímto způsobem jsou dodrženy relativní časové odstupy jednotlivých toků (paketů) útoku.

Jednotlivé záznamy se poté kombinují dle jejich času (čtou se postupně oba soubory a setřiďují se dle času). Výsledkem je CSV soubor s normálním síťovým provozem, do kterého je od specifikovaného času vložen útok při zachování jeho časových vlastností. Tento soubor již lze následně použít k funkčnímu testování.

7.2 Funkční testování

Cílem funkčního testování bylo ověření, že detekční systém správně rozpozná útok v rámci analýzy vstupního souboru. Pro testování byly použity vstupní soubory, vytvořené metodou popsanou v předchozí kapitole (sloučení dat útoku a normálních dat), ale i soubory obsahující pouze data útoku. Dalším cílem funkčního testování bylo ladění detekčních parametrů pro snížení počtu falešně pozitivních detekcí. Testování tak probíhalo průběžně po celou dobu implementace.

7.2.1 Pcap detektor

Testování pcap detektoru probíhalo na souborech uvedených v tabulce 7.2. Do těch nebyla přidána žádná další data, protože cílem tohoto testování bylo ověřit celý princip výběru a agregace podezřelých záznamů dle první sady detekčních prahů a následnou detekci za použití druhé sady prahů vztahující se na celou agregační mapu. Testování nad pcap soubory tak probíhalo úplně první a vedlo k upřesnění implementace použité u ostatních detektorů.

První sada prahů byla nastavena na základě manuální inspekce zachyceného provozu z jednotlivých tunelovacích nástrojů. Dle zjištěných poznatků také probíhalo přidávání a implementace jednotlivých detekčních parametrů jako subdomény stejné délky nebo příliš velký počet speciálních znaků.

Jak potvrdily výsledky, navržený postup byl funkční a bylo správně detekováno využití tunelování přes DNS ve všech testovacích souborech. Dle naměřených výsledků poté byly upraveny hodnoty jednotlivých prahů. Zejména bylo třeba určit orientační práh průměrné velikosti tunelovacích paketů a délky doménového jména v rámci agregace několika datových bloků najednou.

Protože pcap detektor a IPFIX detektor mají k dispozici v zásadě stejné detekční nástroje, jsou konkrétní naměřené výsledky uvedeny v následující kapitole, věnující se testování IPFIX detektoru.

7.2.2 IPFIX detektor

Pro testování IPFIX detektoru byly vytvořeny testovací soubory obsahující jak normální data, tak data útoku. Základem testovacích souborů byla zhruba šestiminutová sekvence normálního provozu, která obsahovala 4 999 999 IPFIX toků. Do ní byla zakomponována

data jednotlivých útoků (každý soubor z tabulky 7.2 zvlášť). Výsledné CSV soubory měly velikosti okolo 677 MB.

Z původního vzorku normálních dat byly také vytvořeny statistické soubory pro první detektor. Zde se ovšem vyskytl menší problém v tom, že při použití tak malého vzorku vstupních dat (ve smyslu doby trvání) mohou statistiky značně narušit poslední datové bloky, které obsahují výrazně méně toků než bloky předchozí. To je způsobeno tím, že je vstupní posloupnost toků neseřazená a po seřazení se v posledních blocích nachází pouze několik málo toků, které byly zaznamenány dříve, než ostatní toky z odpovídajícího časového období. V reakci na tuto skutečnost jsem tedy upravil implementaci tvorby statistik tak, aby se poslední blok do statistik nezapočítával.

Následně už mohlo být provedeno testování. Výsledky testování shrnuje následující tabulka. Ve sloupci „směr“ je naznačen směr detekované komunikace (K je tunelovací klient, S je tunelovací server). Tato informace je důležitá z pohledu možnosti detekce pouze komunikace od klienta na server.

Název útoku	**Směr**	**SK**	**Detekční kritéria**					
			PV	**VP [B]**	**DD**	**CV [B]**	**DV [B]**	**D**
dns2tcp_web	K → S	6	1 156	212,6	166,6	–	–	A
dns2tcp_web	S → K	5	25 636	433,2	–	11 105 467	–	–
dns2tcp_ssh	K → S	5	637	–	98,4	–	–	A
dns2tcp_ssh	S → K	4	2 292	219,3	–	–	–	–
dnscat_bash_NS	K → S	4	122	–	55,9	–	–	–
dnscat_bash_NS	S → K	5	122	317,2	55,9	–	–	–
dnscat_bash_CNM	K → S	5	214	–	55,4	–	–	A
dnscat_bash_CNM	S → K	5	214	265,2	55,4	–	–	–
dnscat_file_MX	K → S	4	119	–	54,0	–	–	–
dnscat_file_MX	S → K	5	119	403,9	54,0	–	–	–
DNScatB_ssh	K → S	6	412	203,0	157,0	–	–	A
DNScatB_ssh	S → K	5	731	362,3	101,2	–	–	–
DNScatB_web	K → S	5	121	178,9	132,9	–	–	–
DNScatB_web	S → K	5	171	296,4	102,6	–	–	–
iodine_ssh	K → S	0	–	–	–	–	–	–
iodine_ssh	S → K	5	60	210,7	70,8	–	–	–
iodine_ssh2	K → S	6	370	184,6	127,7	–	–	A
iodine_ssh2	S → K	5	543	350,5	92,6	–	–	–
iodine_web	K → S	6	3 286	–	107,0	–	388 306	A
iodine_web	S → K	8	5 884	984,0	107,0	5 790 049	5 790 049	A
iodine_web2	K → S	6	1 997	–	107,0	–	262 153	A
iodine_web2	S → K	8	3 573	1005,1	107,0	3 591 296	3 591 296	A
iodine_web3	K → S	7	3 192	198,0	141,0	–	632 033	A
iodine_web3	S → K	6	4 531	901,9	107,5	4 086 753	–	–
tcp-over-dns_web	K → S	5	148	197,9	149,8	–	–	–
tcp-over-dns_web	S → K	5	2 450	314,9	56,0	–	–	–
tcp-over-dns_ssh	K → S	5	211	–	112,8	–	–	A
tcp-over-dns_ssh	S → K	5	283	223,8	96,8	–	–	–

Tabulka 7.3: Výsledky testování IPFIX detektoru. Vysvětlivky použitých zkratek naleznete níže.

V tabulce 7.3 byly použity následující zkratky:

SK Počet splněných detekčních kritérií (z 8 možných).

PV Počet výskytů v agregační mapě (použit práh 50 paketů).

VP Průměrná velikost paketu [B] (použit práh 170 B).

DD Průměrná délka doménového jména (použit práh 50 znaků).

CV Celková velikost přenesených dat [B] (použit práh 1 MiB neboli 1 048 576 B).

DV Celková velikost přenesených dat pouze v dotazech [B] (použit práh 256 KiB neboli 262 144 B).

D Celkový počet dotazů (znak „A“ znamená překročení stanoveného prahu 200 paketů).

Tabulka 7.3 ukazuje výsledky měření. Všechny výsledky odpovídají testovací doméně `tunnel.tunneltest.cz`, která byla použita pro tunelování. Uvedená kritéria odpovídají těm, která jsou použita při kontrole agregační mapy (viz sekce 6.4.3). V seznamu jsou vynechána kritéria pro *nejvyšší počet přístupů na jednu doménu* a *nejvyšší počet použití jednoho typu DNS dotazů*. Tato kritéria byla splněna u všech uvedených útoků (s výjimkou *iodine_ssh*) a jejich hodnoty odpovídají sloupci **PV**.

Z naměřených výsledků je vidět, že naprostá většina útoků byla správně detekována. Jedinou výjimkou byl útok *iodine_ssh*. Ten obsahoval nejmenší počet paketů ze všech testovaných útoků a zároveň přenášel i nejméně dat. Nebyl detekován při použití detekce pouze komunikace od klienta na server, protože v tomto směru obsahoval pouze cirka 30 paketů, které prošly první sadou prahů. Limit pro minimální počet agregovaných paketů byl při testování nastaven na 50. Při snížení tohoto limitu by tedy byla detekována i tato komunikace, nicméně zvýšil by se celkový počet falešně pozitivních detekcí. Proto je tento útok brán jako extrémní případ a v konečném nastavení prahů není uvažován.

Mírně zkreslené počty detekovaných paketů lze pozorovat u útoků *tcp-over-dns_web* a *DNScatB_web*, což je způsobeno jejich délkou. Protože byla vstupní sekvence normálních dat zhruba poloviční délky, je zbytek paketů těchto útoků umístěn do datových bloků, které nesplňují statistické prahy a jsou vyřazeny již prvním detektorem. Při použití delší sekvence normálních dat by tak počty detekovaných paketů těchto útoků byly větší.

Ve sloupci **VP** je patrný trend, kdy není splněn stanovený limit 170 B v případech, kdy je klient ve větší míře pouze příjemcem dat ze serveru a on sám odesílá pouze malé množství dat (např. při spuštění SSH relace na vzdáleném serveru klient data spíše přijímá, nežli odesílá).

Sloupec **DD** zase ukazuje vlastnost nástroje `dns2tcp`, který používá velké množství malých kontrolních paketů s krátkým doménových jménem, takže průměrná délka použitého doménového jména není tak velká, jako u ostatních nástrojů. Chybějící kontrolní pakety nejsou vůbec agregovány, což je vidět na rozdílu počtu agregovaných paketů a celkovém počtu paketů útoku (viz tabulka 7.2).

Sloupce **CV** a **DV** potvrzují, že limity pro celkovou velikost přenesených dat jsou splněny spíše u prohlížení webu, nežli u SSH komunikace.

Konečně sloupec **D** ukazuje, že překročení počtu dotazů lze očekávat ze strany klienta. Výjimkou jsou případy použití nástroje `iodine` s aktivovanou možností použití *raw* paketů (viz popis v kapitole E.2.5). Při převodu *raw* paketů na IPFIX toky jsou totiž nastaveny hodnoty tak, aby je bylo možné snadno detekovat (mimo jiné jsou všechny pakety označeny

jako dotazy, což zajistí překročení tohoto prahu i ze strany serveru). Protože v tomto případě není možné extrahovat ani reálné doménové jméno, jsou tyto pakety detekovány pod doménou `possible.iodineRawUdp.tunnel`, která je přidána při převodu na IPFIX toky.

Snížení počtu falešně pozitivních detekcí

Při výše uvedeném nastavení detekčních prahů bylo kromě útoků falešně detekováno také množství normální komunikace, a to zejména v režimu komunikace ze serveru na klienta. V opačném režimu byla situace mnohem lepší a falešná detekce byla jenom jedna. Po analýze falešně pozitivních dat tak byly zavedeny další implementační prvky s cílem snížení jejich množství.

Prvním krokem bylo zavedení prahu pro *minimální množství přístupů na nejpoužívanější doménu* v rámci množiny zkrácených domén agregovaných k jednomu klíči (dvojici komunikujících IP adres). Prahová hodnota byla nastavena na 20 přístupů. Díky tomuto kroku byly odfiltrovány komunikace, ve kterých se klient během krátké doby dotazuje serveru na velké množství různých domén a server mu odpovídá (u vzorových dat se jednalo o poměrně častý jev).

Druhým krokem bylo zavedení *whitelistu*, tedy seznamu povolených domén. Ten byl zaveden jako reakce na časté využívání databází pro kontrolu URL či emailového serveru. Kontrola u těchto databází probíhá mimo jiné ve formě DNS dotazů a odpovědí typu TXT, které měly charakteristiky podobné právě tunelování přes DNS (přenášely se dlouhé URL adresy jako RDATA, velké pakety a často se využívala jedna kontrolní doména). Příkladem takové domény je `url.zvelo.com`. Do seznamu povolených domén byly přidány některé domény vyskytující se v testovacích datech, ale i další domény dle informací na Internetu. Seznam si může uživatel samozřejmě upravit, nebo ho vůbec nepoužít.

Posledním krokem bylo nové nastavení detekčních prahů. Konkrétně se změnil práh pro *minimální počet agregovaných záznamů* na hodnotu 110 a *minimální počet splněných detekčních kritérií* na 4. Tyto hodnoty byly určeny na základě výsledků testování a vyhovují jim všechny útoky, kromě již míněného *iodine_ssh*.

S tímto nastavením tedy nemůžeme detekovat útoky s velmi malým objemem přenášených dat, ale naprosto minimalizujeme počet falešně pozitivních detekcí. V našem případě klesl počet falešných detekcí na nulu (i v případě komunikace serveru s klientem).

7.2.3 NetFlow detektor

Pro testování NetFlow detektoru byly vytvořeny testovací soubory obsahující jak normální data, tak data útoku. Základem testovacích souborů byla zhruba dvacetiminutová sekvence normálního provozu, která obsahovala 499 999 NetFlow toků. Do ní byla zakomponována data jednotlivých útoků (každý soubor z tabulky 7.2 zvlášť). Oproti IPFIX už se zde nevyskytoval problém s malou délkou vstupních dat, takže útoky mohly být odhaleny v celém jejich rozsahu. Výsledné CSV soubory měly velikosti okolo 55 MB. Z původního vzorku normálních dat byly také vytvořeny statistické soubory pro první detektor.

Výsledky testování shrnuje tabulka 7.4. Význam sloupců **PV**, **VP** a **CV** je stejný jako u tabulky 7.3, včetně použitých prahů. Sloupec **SK** (počet splněných detekčních kritérií) měl v tomto případě horní hranici pouze 4 kritéria. V tabulce se vyskytuje jeden nový sloupec:

DT Maximální doba trvání jednoho toku (nastaven práh 20 s).

Název útoku	Směr	SK	Detekční kritéria			
			PV	VP [B]	CV [B]	DT [s]
dns2tcp_web	K → S	2	–	–	2 131 345	30,4
dns2tcp_web	S → K	3	–	433,2	11 105 467	30,4
dns2tcp_ssh	K → S	1	–	–	–	30,2
dns2tcp_ssh	S → K	2	–	219,3	–	30,2
dnscat_bash_NS	K → S	0	–	–	–	–
dnscat_bash_NS	S → K	2	59	408,0	–	–
dnscat_bash_CNM	K → S	0	–	–	–	–
dnscat_bash_CNM	S → K	1	–	401,0	–	–
dnscat_file_MX	K → S	0	–	–	–	–
dnscat_file_MX	S → K	2	116	407,0	–	–
DNScatB_ssh	K → S	1	–	–	–	83,9
DNScatB_ssh	S → K	3	632	384,1	–	83,9
DNScatB_web	K → S	1	–	–	–	100,5
DNScatB_web	S → K	3	101	343,1	–	100,5
iodine_ssh	K → S	1	–	–	–	33,7
iodine_ssh	S → K	1	–	–	–	33,7
iodine_ssh2	K → S	1	–	–	–	32,0
iodine_ssh2	S → K	2	–	350,5	–	32,0
iodine_web	K → S	1	–	–	–	46,0
iodine_web	S → K	3	–	977,9	5 794 304	46,0
iodine_web2	K → S	1	–	–	–	35,1
iodine_web2	S → K	3	–	1005,1	3 591 296	35,1
iodine_web3	K → S	1	–	–	–	33,3
iodine_web3	S → K	3	–	902,7	4 086 397	33,3
tcp-over-dns_web	K → S	2	195	–	–	376,7
tcp-over-dns_web	S → K	4	4 143	324,9	1 352 435	376,7
tcp-over-dns_ssh	K → S	1	–	–	–	79,7
tcp-over-dns_ssh	S → K	3	61	345,4	–	79,7

Tabulka 7.4: Výsledky testování NetFlow detektoru. Použité zkratky jsou vysvětleny u tabulky 7.3 a výše.

Výsledky odpovídají detekci komunikace mezi IP adresami ze sítě `172.16.1.0/24` (u NetFlow dat nemůžeme detekovat dle DNS dat). Protože se jedná o privátní síť, je vyloučena detekce normálního provozu, spadajícího do této sítě (jedná se o privátní adresový prostor, který není směrován v rámci Internetu). Tabulka obsahuje všechna detekční kritéria.

Můžeme vidět, že správně detekována byla opět většina testovaných útoků. Výjimkou jsou útoky s využitím nástroje `dnscat`, které nebyly detekovány pouze v případě komunikace od klienta na server. To je způsobeno tím, že v tomto směru komunikace obsahovala toky tvořené vždy pouze jedním paketem krátké délky, takže tyto toky vůbec neprošly první sadou detekčních prahů a nebyly tedy vůbec agregovány. U ostatních nástrojů detekci komunikace od klienta dovoloval výskyt toků, které překročily práh pro celkovou délku toku (sloupec **DT**).

Komunikace od serveru ke klientovi byla detekována ve všech případech jak zásluhou dlouhých toků, tak průměrnou velikostí paketu (sloupec **VP**). V případě prohlížení webu

pak byl zpravidla překročen ještě limit celkového množství přenesených dat (sloupec **CV**).

Naopak u sloupce **PV**, tedy překročení počtu výskytů v mapě, byl stanovený práh 50 záznamů překročen pouze u malého počtu případů. Pro snížení množství falešně pozitivních detekcí je tedy vhodné posunout tento práh na hodnotu 100.

Snížení počtu falešně pozitivních detekcí

Oproti IPFIX je u NetFlow velký problém s množstvím falešně pozitivních detekcí. Pokud detekujeme komunikaci i ve směru od serveru ke klientovi, detekujeme kromě útoku řádově desítky jiných komunikací. Většina z nich navíc splňuje tři detekční kritéria, a sice *průměrnou velikost paketů* (hodnoty přes 1 000 B), *celkový počet výskytů v mapě* (řádově stovky) a *množství přenesených dat* (obvykle okolo 2,5 MB). V těchto případech se pravděpodobně jedná o odpovědi na dotazy typu ANY, nebo přenos velkého množství AAAA záznamů. Také se může jednat o kontroly URL adres pomocí DNS, jak bylo popsáno u testování IPFIX detektoru. Na rozdíl od IPFIX dat ale zde nemáme možnost filtrace (např. záznamů A, AAAA), takže těmto detekcím nemůžeme snadno zabránit.

Jediné spolehlivé kritérium pro detekci DNS tunelování nad NetFlow daty tak představuje *maximální délka zachyceného toku*. Toto kritérium bylo uplatněno u všech útoků vyjma už zmíněných (`dnscat`). Při kombinaci s detekcí pouze komunikace od klienta na server klesl počet falešně pozitivních detekcí na nulu.

Z výsledků měření ale plyne, že při vhodném nastavení konfiguračních parametrů (hlavně *velikost přenesených dat* a *počet výskytů v agregační mapě*) je možné NetFlow detektor použít i k detekci jiných anomálií. Takto lze detekovat například *DoS útok* nebo *reflexivní DNS útok*.

7.3 Porovnání efektivity různých detekčních schémat

Cílem této kapitoly je zhodnocení jednotlivých detekčních možností z pohledu jejich efektivity. Ta zahrnuje nejen počet správných nebo falešných detekcí, ale i rychlost, jakou byla detekce provedena.

7.3.1 Přínos prvního detektoru

Původní myšlenkou bylo, že první detektor poměrně výrazným způsobem sníží počet datových bloků, které bude muset zpracovávat druhý detektor. K jejímu splnění ale bohužel došlo pouze v případě NetFlow dat. U IPFIX dat nebyl efekt statisticky založeného filtrování bloků takový, neboť data útoku příliš nezměnila celkové statistiky v rámci statistické periody. To bylo způsobeno velkým počtem toků v jednotlivých blocích, kdy každý tok zhruba odpovídal jednomu paketu. Problematiku ilustrují následující tabulky 7.5, 7.6 a 7.7:

	AVG BPB [B]	**AVG FPB**	**AVG BPPPB [B]**
Normální data	102 947 270, 9	356 725, 1	297, 9
Normální data a útok	103 892 813, 3	360 388, 2	297, 8
Změna v %	0, 92	1, 01	−0, 03

Tabulka 7.5: Vypočítané statistiky IPFIX toků pro bloky délky 20 s.

Tabulky ukazují vypočítané statistiky pro bloky dat různé délky (použité zkratky jsou vysvětleny v kapitole 6.4.1). Použitým útokem byl *dns2tcp_web*, který obsahuje největší

	AVG BPB [B]	AVG FPB	AVG BPPPB [B]
Normální data	49 718 501, 7	172 368, 6	285, 1
Normální data a útok	50 174 970, 5	174 137, 0	284, 9
Změna v %	0, 92	1, 03	−0, 05

Tabulka 7.6: Vypočítané statistiky IPFIX toků pro bloky délky 10 s.

	AVG BPB [B]	AVG FPB	AVG BPPPB [B]
Normální data	24 440 207, 2	84 743, 5	257, 2
Normální data a útok	24 664 573, 2	85 612, 7	257, 0
Změna v %	0, 92	1, 03	−0, 06

Tabulka 7.7: Vypočítané statistiky IPFIX toků pro bloky délky 5 s.

počet paketů a zároveň má největší velikost přenášených dat. Výsledky ukazují, že přidání útoku do normálních dat změní statistiky pouze minimálně. Zároveň je vidět, že data jsou v jednotlivých blocích rozložena rovnoměrně, protože u bloků poloviční délky jsou hodnoty statistik zhruba poloviční. Změna délky bloku tedy v tomto případě nic neřeší.

Výsledkem je, že kvůli rovnoměrnému rozložení dat v jednotlivých blocích není jednoduché rozumně nastavit prahy pro statistickou detekci. Pravidelnost rozložení dat může ovlivnit mimo jiné i denní doba a po opadnutí denní špičky mohou být výsledky o něco lepší (testovaný vzorek dat byl zachycen v době okolo 10:00 ve všední den). Naprostá většina testovaných bloků ale splňuje alespoň jedno detekční kritérium (když má blok malý počet toků, má naopak vyšší průměrnou velikost paketu apod.). Do druhého detektoru je tedy posílána naprostá většina bloků a první detektor mu moc práce neušetří (první detektor v tomto případě odfiltroval pouze zhruba 8 % zpracovávaných bloků). Tuto situaci ilustruje následující tabulka 7.8, která zobrazuje souhrnná data po dokončení detekce. Poslední tři sloupce uvádějí počet uplatnění jednotlivých prahů.

Přijato bloků	Detekováno bloků	AVG BPB	AVG FPB	AVG BPPPB
15	13	7	4	2

Tabulka 7.8: Příklad detekce prvním detektorem u IPFIX dat.

U NetFlow je situace mnohem lepší. Testovaný vzorek dat byl ale na rozdíl od IPFIX dat zachycen v sobotu v době okolo 12:00, což nepochybně hraje svou roli. Objem provozu byl znatelně menší, především z pohledu průměrného počtu bytů v bloku (sloupec *AVG BPB*). Následují obdobné tabulky jako u IPFIX dat (7.9, 7.10 a 7.11).

	AVG BPB [B]	AVG FPB	AVG BPPPB [B]
Normální data	2 747 922, 1	7 094, 4	332, 7
Normální data a útok	2 939 771, 3	7 094, 7	327, 9
Změna v %	6, 98	0, 00	−1, 41

Tabulka 7.9: Vypočítané statistiky NetFlow toků pro bloky délky 20 s.

Stejně jako u IPFIX se jedná o výsledky při použití útoku *dns2tcp_web*. Jeho NetFlow reprezentace je specifická tím, že obsahuje poměrně málo toků, které jsou ale pro změnu dlouhé a obsahují hodně paketů. Oproti IPFIX lze pozorovat znatelné zvýšení průměrného počtu bytů v bloku, zatímco počet toků se takřka nezměnil. Také je zde vidět, že data jsou

	AVG BPB [B]	AVG FPB	AVG BPPPB [B]
Normální data	1 372 187, 7	3 543, 6	336, 9
Normální data a útok	1 467 422, 2	3 543, 8	334, 5
Změna v %	6, 94	0, 00	−0, 71

Tabulka 7.10: Vypočítané statistiky NetFlow toků pro bloky délky 10 s.

	AVG BPB [B]	AVG FPB	AVG BPPPB [B]
Normální data	685 034, 7	1 776, 3	355, 5
Normální data a útok	732 481, 3	1 776, 4	354, 2
Změna v %	6, 93	0, 00	−0, 36

Tabulka 7.11: Vypočítané statistiky NetFlow toků pro bloky délky 5 s.

v jednotlivých blocích rozložena rovnoměrně (hodnoty se zhruba půlí při zkrácení bloku na polovinu). Pro práci se statistickým detektorem tedy není délka bloku rozhodujícím parametrem.

Zajímavostí je, že sloučení útoku s normálními daty sníží průměrnou velikost paketů v bloku (potvrzeno více různými měřeními). To je zapříčiněno velkým množstvím malých kontrolních (*keepalive*) paketů, které útoky obsahují.

Oproti IPFIX datům první detektor u NetFlow dat detekuje (odesílá druhému detektoru) méně bloků, a jeho přínos je tak větší. Tuto situaci ilustruje následující tabulka, která zobrazuje souhrnná data po dokončení detekce. Poslední tři sloupce uvádějí počet uplatnění jednotlivých prahů.

Přijato bloků	Detekováno bloků	AVG BPB	AVG FPB	AVG BPPPB
71	45	17	1	27

Tabulka 7.12: Příklad detekce prvním detektorem u NetFlow dat.

Můžeme vidět, že v tomto případě první detektor odfiltruje zhruba 37 % všech bloků, což není špatný výsledek (na schopnostech detekce tunelování se nic nemění). Minimální využití prahu pro *průměrný počet toků v bloku* souvisí s charakteristikou útoku.

7.3.2 Využití detekčních prahů

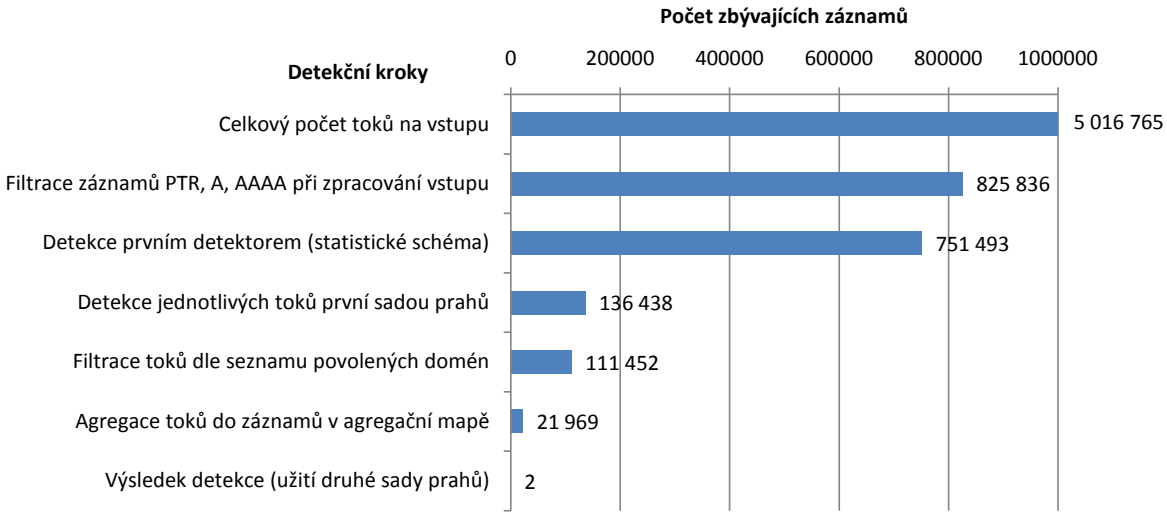
Na základě statistických informací posbíraných v průběhu detekce můžeme vyhodnotit efektivitu jednotlivých detekčních a filtračních kroků, které přispívají k jejímu urychlení.

IPFIX detekce

Pro IPFIX detekci je možné využít filtraci na několika úrovních. Jednotlivé filtrační kroky jsou popsány v grafu 7.2. Ten ukazuje posloupnost detekčních a filtračních kroků (v grafu vertikálně směrem dolů) spolu s jejich vlivem na počet záznamů, který je dále zpracován. Graf ukazuje detekci útoku *tcp-over-dns_web*, který byl sloučen s normálními daty. Výsledkem jsou dva korektně detekované záznamy o tunelované komunikaci (oba směry komunikace).

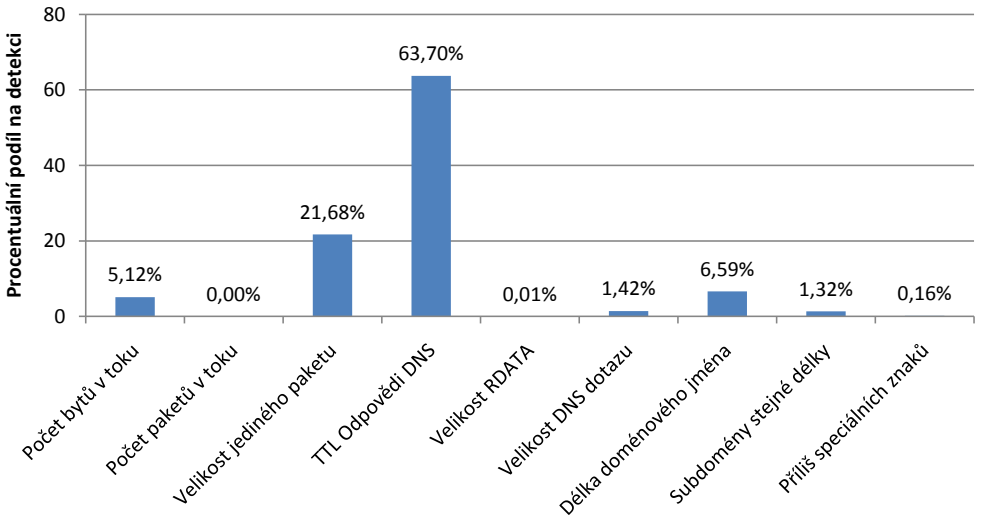
Jak je patrné z uvedených hodnot, největší snížení počtu zbývajících záznamů provede filtrace běžných typů záznamů již při zpracování vstupu. Má tak největší podíl na urychlení detekce a snížení paměťových nároků (ne však na přesnost detekce). Také se ukazuje nízká

efektivita prvního detektoru, který odfiltruje pouze malou část příchozích dat. Velký skok nastává při detekci jednotlivých paketů první sadou detekčních prahů ve druhém detektoru. Ten vybere pouze malou část záznamů, které jsou následně agregovány. Z agregovaných záznamů jsou pak druhou sadou prahů detekovány skutečné hrozby. Pro přesnost detekce je tedy druhá sada prahů rozhodující.



Obrázek 7.2: Průběh filtrace záznamů během detekčního procesu IPFIX.

V dalším grafu (7.3) si přiblížíme využití jednotlivých prahů první sady, která slouží jako výběr záznamů pro agregaci. Připomeňme, že k vložení záznamu do agregační mapy stačí, aby byl splněn libovolný z uvedených prahů (jinak se postupně kontrolují všechny prahy této sady). Graf zobrazuje procentuální využití jednotlivých prahů v průběhu stejného útoku jako v předchozím případě (*tcp-over-dns_web*).

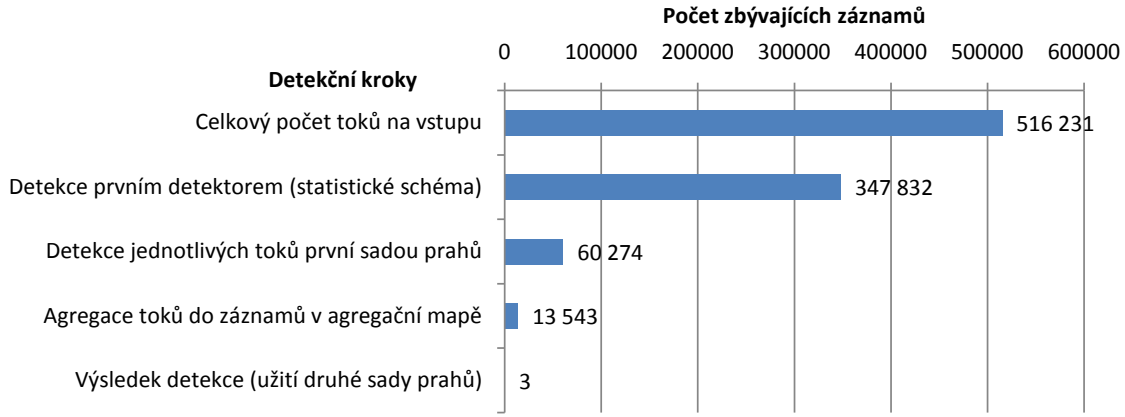


Obrázek 7.3: Využití jednotlivých prahů první sady během detekčního procesu IPFIX.

Poměrně překvapivě je nejčastěji využíván práh pro velikost TTL, který byl při testování nastaven na hodnotu 5 (prahu vyhovují pakety s menším TTL). Druhý nejčastěji využívaný práh detekuje jednotlivé pakety (toky obsahující jeden paket) s velikostí větší než 300 B. Poté stojí za zmínku ještě práh pro délku doménového jména (nastaven na hodnotu 50 znaků), nebo celkovou velikost všech paketů toku (více než 800 B). Ostatní prahy jsou využívány pouze výjimečně.

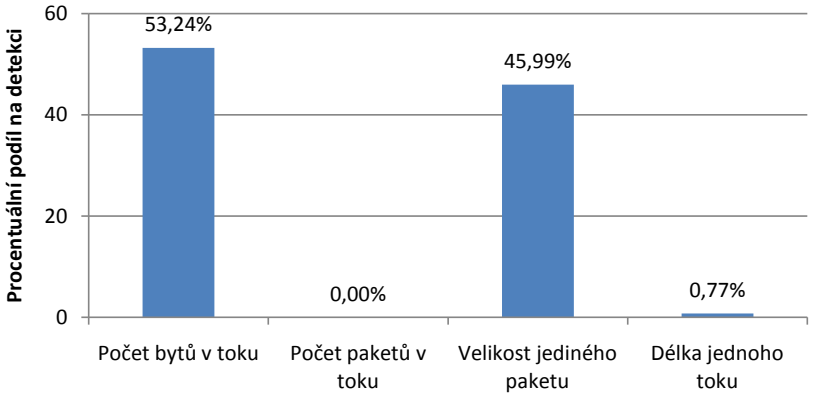
7.3.3 NetFlow detekce

Pro porovnání si ukážeme totožné grafy pro stejný útok při využití NetFlow dat.



Obrázek 7.4: Průběh filtrace záznamů během detekčního procesu NetFlow.

Protože u NetFlow nemůžeme odfiltrovat při zpracování vstupu nepotřebné záznamy, musíme se spolehnout na správnou činnost ostatních filtračních mechanizmů. Určitou práci odvede již první detektor, ale většina záznamů je vyřazena podobně jako u IPFIX až první sadou prahů (obrázek 7.4). O přesnosti detekce rozhodují detekční kritéria při kontrole agregační mapy. Vzhledem k omezeným možnostem u NetFlow dat pracují spolehlivě (výsledné tři detekce jsou všechny správně, protože během detekce byla prováděna inspekce agregační mapy několikrát).



Obrázek 7.5: Využití jednotlivých prahů první sady během detekčního procesu NetFlow.

Graf procentuálního využití první sady prahů (7.5) ukazuje, že důvodem k agregaci je u většiny toků jejich velikost. Limit pro počet paketů v toku byl nastaven na 20, což u tohoto typu útoku nebylo nikdy překročeno. U jiných útoků ale tento limit může být překročen. Práh pro délku toku s hodnotou 20 s byl v tomto případě překročen pouze výjimečně (když se jednalo o data útoku, což ho dělá nejefektivnějším). Ostatní prahy měly stejné hodnoty jako u IPFIX detekce.

7.3.4 Rychlost detekce

Cílem této kapitoly je porovnání jednotlivých detekčních metod z pohledu jejich rychlosti (doby běhu programu). Jako vstupní data sloužily soubory obsahující 1 000 000 toků normálního provozu (bez útoku). U IPFIX se jednalo pouze o jeden CSV soubor o velikosti 134 MB. U NetFlow to byly soubory dva. První byl CSV soubor o velikosti 109 MB, druhý byl binární soubor vytvořený kolektorem `nfcapd`, který měl velikost 60 MB (oba soubory byly z hlediska obsahu totožné).

Testování probíhalo na sestavě popsané v kapitole 7. Jako pevný disk byl použit klasický magnetický disk s 5 400 otáčkami (verze pro notebooky). Ten se také později ukázal jako největší brzda výkonu.

Měření doby zpracování probíhalo s různým detekčním nastavením. Každé měření probíhalo třikrát a výsledné hodnoty byly zprůměrovány. K měření času byl použit nástroj `time`, který je součástí systémů unixového typu. Měřeny jsou dva různé časy. *Reálný čas* představuje dobu od spuštění po konec běhu programu. Tato doba zahrnuje i provádění ostatních běžících procesů. *CPU čas* reprezentuje čas, po který byl program prováděn v CPU. Tento údaj nezahrnuje provádění ostatních běžících procesů.

Dalším prvkem měření bylo použití automatické optimalizace při překladu programu. Ta dokáže bez nutnosti změny kódu program urychlit. Pro tuto možnost byl použit překladač `gcc` a jeho optimalizační parametr `-O3` (standardně byl program překládán bez něj, pro přeložení programu s optimalizací je nutno překládat příkazem `make optim`).

Detekční schéma	**O**	**RČ** [s]	**PČ** [s]	**Z** [s]	**Z** [%]
Bez filtrace, agregace všech záznamů	–	66, 22	52, 84	–	–
Bez filtrace, agregace všech záznamů	-O3	58, 30	44, 75	7, 92	11, 96
Bez filtrace, agregace vybraných záznamů	–	58, 91	46, 65	7, 31	11, 03
Bez filtrace, agregace vybraných záznamů	-O3	52, 58	39, 95	13, 64	20, 59
S filtrací, agregace všech záznamů	–	53, 05	42, 64	13, 17	19, 89
S filtrací, agregace všech záznamů	-O3	47, 42	36, 85	18, 80	28, 39
S filtrací, agregace vybraných záznamů	–	52, 21	41, 89	14, 01	21, 16
S filtrací, agregace vybraných záznamů	-O3	46, 93	36, 37	19, 29	29, 13

Tabulka 7.13: Porovnání doby detekce IPFIX detektoru.

Prvním testem bylo porovnání doby zpracování různých detekčních schémat u IPFIX detektoru. Tabulka 7.13 ukazuje dosažené časy pro jednotlivá schémata (reálný čas je značen jako **RČ**, CPU čas je značen **PČ**). Sloupec **Z** udává urychlení reálného času oproti nejpomalejší variantě, která je uvedena v prvním řádku. Sloupec **O** pak indikuje použití optimalizace při překladu. Detekční schémata tvoří celkem čtyři varianty. Použití/nepoužití filtrace obecných typů DNS záznamů při zpracování vstupu a agregace všech záznamů, nebo agregace pouze vybraných záznamů podezřelých typů (dle první sady detekčních prahů v druhém detektoru).

Výsledky ukazují, že jenom filtrací obecných typů záznamů můžeme běh detekčního procesu zrychlit zhruba o 20 %. Dalšího urychlení o 10 % můžeme dosáhnout zmenšením počtu agregovaných záznamů. Tento krok má vliv nejen na urychlení programu, ale i na počet falešně pozitivních detekcí, které se objevily po agregaci všech záznamů bez výjimky. Filtrace záznamů při zpracování vstupu na počet falešných detekcí nemá vliv. Rozhodující je tedy použití schématu s agregací pouze podezřelých záznamů.

Použití optimalizace při překladu program ve všech případech urychluje zhruba o 10 %, což není špatný výsledek vzhledem k tomu, že tento krok vůbec nevyžaduje změnu v kódu programu.

U NetFlow dat bylo méně možností k otestování různých detekčních schémat (také zde máme na výběr z agregace záznamů všech či pouze vybraných, ale nemůžeme provádět filtrování dat při zpracování vstupu). Důležitým srovnáním ale bylo zpracování stejných dat v různých vstupních formátech (textový CSV soubor a binárně uložená data).

Detekční schéma	Vstup	O	RČ [s]	PČ [s]	Z [s]	Z [%]
Agregace všech záznamů	CSV	–	47, 78	40, 40	–	–
Agregace všech záznamů	CSV	-O3	42, 21	33, 83	5, 09	11, 65
Agregace vybraných záznamů	CSV	–	45, 69	36, 92	2, 10	4, 39
Agregace vybraných záznamů	CSV	-O3	40, 25	31, 48	7, 54	15, 25
Agregace všech záznamů	binární	–	14, 84	11, 73	32, 94	68, 94
Agregace všech záznamů	binární	-O3	12, 88	8, 65	34, 90	73, 36
Agregace vybraných záznamů	binární	–	13, 06	8, 44	34, 72	72, 66
Agregace vybraných záznamů	binární	-O3	12, 10	6, 54	35, 69	74, 68

Tabulka 7.14: Porovnání doby detekce NetFlow detektoru.

Naměřené hodnoty (tabulka 7.14) ukazují překvapivě velký rozdíl ve zpracování jednotlivých vstupních formátů. Práce s binárními daty je o celých 70 % rychlejší než s daty textovými. To je způsobeno tím, že zatímco u binárních dat můžeme při extrakci jednotlivé hodnoty pouze překopírovat, u textových dat je nejprve musíme převést do číselného tvaru, a až následně s nimi můžeme pracovat. Formát CSV je tak vhodný pouze pro testování, ale pro reálné nasazení detekčního systému je třeba využít binárních dat.

Dále se ukazuje, že automatická optimalizace i v tomto případě urychlí program o zhruba 10 % (v případě binárních dat to je o něco méně). Rozdíl v počtu agregovaných záznamů není tak výrazný, jako u IPFIX detekce (při agregaci nemusíme kontrolovat zkrácená doménová jména).

Profilování

Profilování je metoda dynamické analýzy programu za jeho běhu. Využívá se pro identifikaci úseků kódu, ve kterých program při zpracování tráví nejvíce času. Tyto úseky poté mohou být podrobeny zvláštní optimalizaci. V našem případě by optimalizace těchto úseků byla námětem k dalšímu vylepšení detekčního systému.

Pro profilování byl použit *Valgrind* [8], konkrétně jeho nástroj *Callgrind*. Ten vytváří graf volání jednotlivých funkcí za běhu programu a výsledky ukládá do zvláštního souboru. Pro zobrazení výsledků profilování jsem použil *KCachegrind* [58], který umí tento graf zobrazit (včetně procentuálního vyjádření, kolik času program tráví v každé funkci) nebo vyhledávat v grafu jednotlivé funkce programu.

Všechny výsledné grafy a výstupní soubory jsou přiloženy na CD.

Při inspekci výsledků profilování bylo zjištěno, že u zpracování CSV zabere zhruba 90 % času zpracování vstupu, zatímco detekce pouze okolo 9 % (zbytek je režie vláken apod.). U binárních dat byl tento poměr jiný, zpracování vstupu zabralo zhruba 75 % a detekce 24 % celkového času (dle počtu agregovaných položek).

U IPFIX detekce byl čas detekce druhým detektorem rozdělen mezi aplikaci první sady prahů, vkládání dat do agregační mapy, inspekce agregační mapy a mazání nepotřebných bloků dat. Inspekce agregační mapy zabrala pouze asi desetinu času. Polovina času byla rovnoměrně rozdělena mezi aplikaci první sady prahů a vkládání dat do agregační mapy. Zbývající čas spotřebovalo mazání nepotřebných bloků dat (bloky obsahují vektory se stovkami tisíc položek). Použitá agregace hašovací tabulkou tedy detekci nezpomalila tolik, jak by se mohlo zdát.

U NetFlow detekce zabralo většinu času druhého detektoru vkládání dat do agregační mapy (zhruba polovinu doby). Detekce první sadou prahů nebyla náročná a trvala méně než desetinu času. U NetFlow je tedy agregační mapa více limitujícím faktorem než u IPFIX detekce. Zbývající čas byl rovnoměrně rozdělen mezi inspekci agregační mapy a mazání nepotřebných bloků.

Nejpodstatnější ale byla analýza modulu pro zpracování vstupu. U CSV souborů byla doba zpracování vstupu rozdělena mezi dvě funkce. První funkce se starala o načtení řádku souboru a jeho rozdělení na jednotlivé sloupce (ty ukládá do vektoru). Funkce tedy neposkytovala příliš možností pro optimalizaci. Její podíl na celkové době zpracování byl takřka 30 %. Druhá funkce prováděla konverzi jednotlivých sloupců z textových dat do binární podoby a hodnoty ukládala do struktury představující jeden záznam. Tato funkce zabrala zhruba 60 % celkového času.

Rozhodující bylo, že v rámci této druhé funkce bylo celkem 80 % jejího trvání (tedy zhruba 55 % celkové doby běhu programu) způsobeno obecnou funkcí `get_int_from_str`, která provádí převod řetězce na číslo. Jednalo se o pouhý převod řetězce na číslo s následnou kontrolou chyb, celkem asi sedm řádků kódu. K převodu byla použita C++ třída `stringstream`. Zde spočívala podstata slabé výkonnosti. Dle porovnání rychlosti různých způsobů převodu textu na číslo v C++ [27] je totiž `stringstream` velmi pomalou metodou. Na základě tohoto zjištění jsem tedy změnil implementaci převodu na funkci `strtol`, která je dle uvedeného srovnání až 20krát rychlejší. Pro porovnání jsem tedy provedl několik nových měření.

Detekční schéma	**Detektor**	**PRČ** [s]	**NRČ** [s]	**Z** [%]
Bez filtrace, agregace všech záznamů	IPFIX	66, 22	39, 48	40, 38
Bez filtrace, agregace vybraných záznamů	IPFIX	58, 91	32, 52	44, 79
S filtrací, agregace vybraných záznamů	IPFIX	52, 21	26, 31	49, 61
Agregace všech záznamů	NetFlow	47, 78	25, 63	46, 35
Agregace vybraných záznamů	NetFlow	45, 69	24, 99	45, 31

Tabulka 7.15: Změna rychlosti programu po optimalizaci načítání CSV souborů.

Tabulka 7.15 ukazuje výsledky provedených měření. Sloupec **PRČ** ukazuje *původní reálný čas*, sloupec **NRČ** *nový reálný čas*. Sloupec **Z** nese hodnotu procentuálního zrychlení. Měření potvrdila významné urychlení běhu programu. Pouze drobnou úpravou jedné funkce tak bylo dosaženo až 50 % zrychlení programu.

Analýza zpracování binárních vstupů probíhala nad NetFlow daty. Doba zpracování vstupu (která tvořila zhruba 80 % celkové doby běhu programu) byla rozložena mezi dvě funkce. Čtvrtinu celkového času tvořila funkce pro třídění toků dle jejich času (vyhledání správného bloku). Zbylou polovinu času zabrala funkce pro zpracování vstupních binárních dat. I v ní se totiž prováděl neoptimální převod, tentokrát čísla na řetězec (s využitím C++ třídy `ostringstream`). Po předchozí zkušenosti jsem tedy i tento převod nahradil rychlejší variantou (`snprintf`) a provedl nová měření.

Detekční schéma	Detektor	PRČ [s]	NRČ [s]	Z [%]
Agregace všech záznamů	NetFlow	14, 84	11, 07	25, 40
Agregace vybraných záznamů	NetFlow	13, 06	9, 98	23, 58

Tabulka 7.16: Změna rychlosti programu po optimalizaci načítání binárních souborů.

V tomto případě již nebylo urychlení tak výrazné jako u CSV souborů, průměrně se jednalo o urychlení zhruba o čtvrtinu. Ale i to není špatný výsledek.

Díky profilování se tedy program podařilo značně urychlit. Ukázky procentuálního využití modulu pro zpracování vstupu před a po urychlení naleznete v příloze D.

7.4 Testování v reálném provozu

Posledním krokem testování bylo potvrzení funkčnosti systému v reálném nasazení. To probíhalo na serveru `benefizio.liberouter.org` s využitím verze programu určené pro systém Nemea. Detekce běžela nepřetržitě osm hodin, během kterých bylo prověřeno 300 milionů IPFIX toků.

Testování pomohlo upřesnit seznam povolených domén. Do seznamu přibyla například doména `j.e5.sk`, kterou využívá ke kontrole doménových jmen antivirová aplikace od společnosti *Eset*. Komunikace s touto doménou byla velmi podobná tunelování přes DNS, protože využívala TXT záznamy, ve kterých byla zakódována data v několika subdoménách o délce 63 znaků.

Bohužel se opět ukázala problematická efektivita prvního detektoru, pro který je velmi důležitá vhodná volba statistických prahů a existence denních statistik. Ta byla ovšem částečně způsobena nedostatkem statistických dat. V našem případě detekce probíhala na základě jediné statistické periody (půl hodiny), která byla použita po celou dobu detekce. Výsledkem bylo, že z celkem 1 394 vytvořených bloků dat (každý o délce 20 s) bylo detekováno 1 257 bloků. První detektor tedy v tomto případě odfiltroval pouze 10 % dat.

V kontrastu s nízkou efektivitou prvního detektoru se ukázala poměrně vysoká přesnost druhého detektoru, který za dobu detekce označil celkem 18 podezřelých komunikací. 12 z nich by bylo odfiltrováno po provedené aktualizaci seznamu povolených domén. Zbylých 6 záznamů mělo podobnou charakteristiku jako ostatní domény používané pro bezpečnostní kontroly, ale na rozdíl od nich nebyla využita jedna centrální doména. Pro eliminaci těchto falešně pozitivních detekcí bychom museli zvýšit hodnoty detekčních prahů, či minimální počet výskytů v agregační mapě.

Do budoucna by bylo vhodné zachytit a analyzovat větší počet denních statistik, které jsou využity v prvním detektoru. Na základě lepší znalosti průměrných hodnot či výkyvů v různou denní dobu bychom mohli lépe nastavit detekční prahy a optimalizovat tak činnost prvního detektoru.

7.5 Shrnutí

Implementovaný detekční systém splňuje požadavky, které na něj byly kladeny. Při doporučeném nastavení dokáže správně detekovat naprostou většinu útoků typu tunelování přes DNS s nulovým, nebo minimálním počtem falešně pozitivních detekcí. Při nevhodném nastavení se může počet falešných detekcí zvýšit. Volba nastavení tak dává uživateli možnost vyladit detekci dle jeho požadavků, co se rychlosti a přesnosti detekce týče. Zároveň je takto možné detekovat i jiné typy útoků (zejména při využití NetFlow detektoru).

Zvolené propojení jednotlivých modulů detekčního systému sériově pomocí sdílených front se ukázalo jako výhodné, neboť doba detekce prvním detektorem je velmi malá a data se tak bez zbytečného čekání dostanou ke druhému detektoru. Bohužel efektivita zvoleného algoritmu pro statistickou detekci nebyla příliš vysoká, zejména co se IPFIX dat týče.

Z pohledu detekce je tak nejdůležitější správná volba jednotlivých prahů u druhého detektoru. Zejména je nutné využít první sady prahů, protože jinak se agreguje dostatek dat na to, aby vyvolaly falešně pozitivní detekci.

Pro celkovou rychlost detekce je důležitá volba vstupního formátu (binární či textová data), ale i překlad programu s automatickou optimalizací kódu. U IPFIX dat je z hlediska zvýšení rychlosti důležitá volba filtrace obecných typů záznamů už při zpracování vstupu.

Kapitola 8

Závěr

Tato práce popisuje návrh systému pro efektivní detekci síťových anomálií s využitím DNS dat. Nejprve tak bylo nutné nastudovat principy fungování systému DNS, které shrnuje druhá kapitola.

Dalším krokem bylo studium útoků a anomálií, které se vyskytují v souvislosti se systémem DNS. Třetí kapitola práce tedy popisuje principy nejvýznamnějších hrozeb a útoků. U významných útoků jsou zmíněny existující přístupy využívané k jejich detekci. Všechny materiály, ze kterých jsem čerpal, jsou zmíněny v seznamu literatury.

Čtvrtá kapitola diskutuje možnosti a způsoby získávání síťových dat. Jsou popsány hlavní formáty pro ukládání síťových dat – *pcap, NetFlow* a *IPFIX*. Pro snížení náročnosti a zvýšení efektivity detekce jsou pro implementaci výsledného systému vybrány formáty pracující se síťovými toky – *NetFlow* a *IPFIX*. Pro účely testování je nicméně podporován i formát *pcap*.

V páté kapitole následuje návrh detekčního systému. Systém je navržen pro detekci nad záznamy síťových toků či paketů uložených ve výše zmíněných formátech. Pro zvýšení efektivity detekce síťových anomálií je zvoleno zapojení dvou detektorů, přičemž první detektor implementuje rychlé detekční metody nad omezeným spektrem datových atributů. Druhý detektor naopak implementuje rozšířené (složitější) detekční metody, které potřebují větší počet datových atributů a jsou i výpočetně náročnější. První detektor tedy určí, která data budou podrobena důkladnější analýze.

Šestá kapitola obsahuje podrobnosti o implementaci navrženého systému. Implementace je zaměřena na detekci *tunelování přes DNS*, ale po změně nastavení lze detekovat i jiné anomálie. Program je vyhotoven ve dvou verzích. První, samostatná verze je určena pro testování či práci nad staticky uloženými daty, zatímco druhá verze je implementována jako modul pro systém *Nemea* a může tak být využita v reálném provozu.

V sedmé kapitole je popsán proces testování. To ukázalo, že implementovaný detekční systém je při odpovídajícím nastavení schopen přesně detekovat případy tunelování přes DNS. Detekce byla testována nad reálnými daty útoků sloučenými s normálním síťovým provozem, ale i pouze nad normálním provozem. Pro její přesnost je důležité vhodně zvolit prahy jednotlivých detekčních parametrů, jinak může dojít k *falešně pozitivním detekcím.*

Na závěr je diskutováno porovnání efektivity jednotlivých detekčních technik a jejich podíl na celkovém výsledku. V rámci měření rychlosti detekce pak byla provedena ještě dodatečná optimalizace programu s cílem jeho urychlení.

Přínosem této práce je potvrzení funkčnosti navrženého systému, tedy myšlenky spolupráce různých detekčních modulů s cílem dosažení vyšší efektivity. Aplikaci je možné dále rozšiřovat a doplnit tak možnost detekce jiných síťových anomálií, anebo přidat podporu nových formátů vstupních dat.

Literatura

[1] *CESNET* [online]. 2014 [cit. 2014-12-27]. Dostupné z: <http://www.cesnet.cz/>.

[2] *CZ.NIC: O sdružení* [online]. 2014 [cit. 2014-12-27]. Dostupné z: <http://www.nic.cz/page/351/>.

[3] *CZ.NIC: Registrátoři poskytující také služby koncovým držitelům* [online]. 2014 [cit. 2014-12-27]. Dostupné z: <http://www.nic.cz/whois/registrars/list/1/>.

[4] *CZ.NIC: Vyhledávání v registru (Whois)* [online]. 2014 [cit. 2014-12-27]. Dostupné z: <http://www.nic.cz/whois/>.

[5] *IANA: Root Zone Database* [online]. 2014 [cit. 2014-12-27]. Dostupné z: <http://www.iana.org/domains/root/db>.

[6] *IANA: WHOIS Service* [online]. 2014 [cit. 2014-12-27]. Dostupné z: <http://www.iana.org/whois>.

[7] *Liberouter: Nemea* [online]. 2014, [Aktualizováno 2014-09-12] [cit. 2014-12-16]. Dostupné z: <https://www.liberouter.org/technologies/nemea/>.

[8] *Valgrind* [online]. 2014 [cit. 2015-05-18]. Dostupné z: <http://valgrind.org/>.

[9] *The Wireshark Wiki: Libpcap File Format* [online]. 2014, [Aktualizováno 2013-07-29] [cit. 2014-12-28]. Dostupné z: <http://wiki.wireshark.org/Development/LibpcapFileFormat>.

[10] *Tcpdump & libpcap* [online]. 2015, [Aktualizováno 2015-03-11] [cit. 2015-04-18]. Dostupné z: <http://www.tcpdump.org/>.

[11] *TIOBE Software: TIOBE Index for March 2015* [online]. 2015, [Aktualizováno v březnu 2015] [cit. 2015-04-03]. Dostupné z: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>.

[12] Banerjee, A., Rahman, M. S. a Faloutsos, M. SUT: Quantifying and Mitigating URL Typosquatting. *Computer Networks*. Září 2011, roč. 55, č. 13. S. 3001 – 3014. ISSN 1389-1286.

[13] Bartoš, V. *Libnfdump: Library for reading nfdump files* [online]. 2014, [Aktualizováno 2014-08-13] [cit. 2015-04-18]. Dostupné z: <http://sourceforge.net/projects/libnfdump/>.

[14] Born, K. a Gustafson, D. Detecting DNS Tunnels Using Character Frequency Analysis. In *Proceedings of the 9th Annual Security Conference*. Las Vegas, Nevada: [b.n.], duben 2010.

[15] Born, K. a Gustafson, D. NgViz: Detecting DNS Tunnels Through N-gram Visualization and Quantitative Analysis. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*. New York, NY, USA: ACM, 2010. S. 47:1 – 47:4. CSIIRW '10. ISBN 978-1-4503-0017-9.

[16] Bowes, R. *Tcp-over-dns* [online]. 2010, [Aktualizováno 2010-07-07] [cit. 2015-05-18]. Dostupné z: <https://wiki.skullsecurity.org/Dnscat>.

[17] Bush, R., Karrenberg, D., Kosters, M. et al. *Root Name Server Operational Requirements* [RFC 2870 (Best Current Practice)]. červen 2000. Dostupné z: <http://www.ietf.org/rfc/rfc2870.txt>.

[18] Castro, S., Zhang, M., John, W. et al. Understanding and Preparing for DNS Evolution. In *Traffic Monitoring and Analysis*. Berlin, Heidelberg: Springer, duben 2010. Lecture Notes in Computer Science, sv. 6003. ISBN 978-3-642-12364-1.

[19] CESNET. *Topologie sítě CESNET2* [online]. 2014, [Aktualizováno 2014-12-02] [cit. 2015-05-17]. Dostupné z: <http://www.cesnet.cz/sluzby/pripojeni/topologie/>.

[20] Cisco. *Introduction to Cisco IOS NetFlow: A Technical Overview* [online]. květen 2012 [cit. 2014-12-27]. Dostupné z: <http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html>.

[21] Cisco. *Cisco 2014 Annual Security Report* [online]. 2014 [cit. 2014-12-28]. Dostupné z: <http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf>.

[22] Claise, B. *Cisco Systems NetFlow Services Export Version 9* [RFC 3954 (Informational)]. říjen 2004. Dostupné z: <http://www.ietf.org/rfc/rfc3954.txt>.

[23] Claise, B. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information* [RFC 5101 (Proposed Standard)]. leden 2008. Dostupné z: <http://www.ietf.org/rfc/rfc5101.txt>.

[24] Claise, B., Trammell, B. a Aitken, P. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* [RFC 7011 (INTERNET STANDARD)]. září 2013. Dostupné z: <http://www.ietf.org/rfc/rfc7011.txt>.

[25] Dembour, O. a Collignon, N. *HSC: Dns2tcp* [online]. 2010, [Aktualizováno 2012-05-02] [cit. 2015-05-18]. Dostupné z: <http://www.hsc.fr/ressources/outils/dns2tcp/>.

[26] Di Paola, S. a Lombardo, D. Protecting Against DNS Reflection Attacks with Bloom Filters. In *Proceedings of the 8th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Berlin, Heidelberg: Springer-Verlag, 2011. S. 1 – 16. DIMVA'11. ISBN 978-3-642-22423-2.

[27] Didriksen, T. *C++ Convert String to Int Speed* [online]. 2010, [Aktualizováno 2010-04-15] [cit. 2015-05-20]. Dostupné z: <http://tinodidriksen.com/2010/02/16/cpp-convert-string-to-int-speed/>.

[28] Ekman, E., Andersson, B. a Bezemer, A. *Kryo.se: iodine* [online]. 2014, [Aktualizováno 2014-06-16] [cit. 2015-05-18]. Dostupné z: <http://code.kryo.se/iodine/>.

[29] Ellens, W., Żuraniewski, P., Sperotto, A. et al. Flow-Based Detection of DNS Tunnels. In *Emerging Management Mechanisms for the Future Internet*. Berlin, Heidelberg: Springer, 2013. S. 124 – 135. Lecture Notes in Computer Science, sv. 7943. ISBN 978-3-642-38997-9.

[30] Farnham, G. *Detecting DNS Tunneling*. Swansea: SANS Institute, únor 2013. 32 s. Dostupné z: <http://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152>.

[31] FlowTraq. *Flow Exporter* [online]. 2015 [cit. 2015-04-18]. Dostupné z: <http://www.flowtraq.com/corporate/product/flow-exporter/>.

[32] Haag, P. *NFDUMP: Netflow collecting and processing tools* [online]. 2014, [Aktualizováno 2014-12-01] [cit. 2015-04-18]. Dostupné z: <http://sourceforge.net/projects/nfdump/>.

[33] Huistra, D. Detecting Reflection Attacks in DNS Flows. In *19th Twente Student Conference on IT*. Enschede: University of Twente, 2013. Dostupné z: <http://referaat.cs.utwente.nl/conference/19/paper>.

[34] James, D. *Boost C++ Libraries: Combining hash values* [online]. 2008 [cit. 2015-04-27]. Dostupné z: <http://www.boost.org/doc/libs/1_58_0/doc/html/hash/combine.html>.

[35] Kambourakis, G., Moschos, T., Geneiatakis, D. et al. Detecting DNS Amplification Attacks. In *Proceedings of the Second International Conference on Critical Information Infrastructures Security*. Berlin, Heidelberg: Springer-Verlag, 2008. S. 185 – 196. CRITIS'07. ISBN 3-540-89095-5, 978-3-540-89095-9.

[36] Karasaridis, A., Meier Hellstern, K. a Hoeflin, D. Detection of DNS Anomalies using Flow Data Analysis. In *Global Telecommunications Conference, GLOBECOM '06.* New York: IEEE, listopad 2006. ISSN 1930-529X.

[37] Kováčik, M. Detekcia sieťových anomálií s využitím DNS dát. In *PAD 2013: Počítačové architektury a diagnostika*. Plzeň: Západočeská univerzita v Plzni, 2013. S. 33 – 38. ISBN 978-80-261-0270-0.

[38] Kováčik, M. *Detekce síťových anomálií a bezpečnostních incidentů s využitím DNS dat*. FIT VUT v Brně, 2014. 34 s. Pojednání k tématu dizertační práce.

[39] Kováčik, M. *Liberouter: DNS plugin* [online]. 2014, [Aktualizováno 2014-04-23] [cit. 2015-04-18]. Dostupné z: <https://www.liberouter.org/technologies/dns-plugin/>.

[40] Kozierok, C. M. *DNS Basic Name Resolution Techniques: Iterative and Recursive Resolution* [online]. 2005, [Aktualizováno 2005-09-20] [cit. 2014-12-29]. The TCP/IP Guide. Dostupné z: <http://www.tcpipguide.com/free/t_DNSBasicNameResolutionTechniquesIterativeandRecurs.htm>.

[41] Kurose, J. F. a Ross, K. W. *Computer Networking: A Top-Down Approach*. 6. vyd. New Jersey: Pearson, 2012. ISBN 0132856204, 9780132856201.

[42] Matoušek, P. *Systém DNS* [Studijní opora pro předmět ISA – Síťové aplikace a správa sítí]. 2011. FIT VUT v Brně.

[43] Mockapetris, P. *Domain names: implementation and specification* [RFC 1035 (INTERNET STANDARD)]. listopad 1987. Dostupné z: <http://www.ietf.org/rfc/rfc1035.txt>.

[44] Mockapetris, P. *Domain names - concepts and facilities* [RFC 1034 (INTERNET STANDARD)]. listopad 1987. Dostupné z: <http://www.ietf.org/rfc/rfc1034.txt>.

[45] NSFOCUS. *Bandwidth Consumption DDoS Attacks and Mitigation Methods – Part 2: DNS Amplification Attack* [online]. 2013, [Aktualizováno 2013-11-18] [cit. 2014-12-29]. Dostupné z: <http://nsfocusblog.com/2013/11/18/bandwidth-consumption-ddos-attacks-and-mitigation-methods-part-2/>.

[46] Pietraszek, T. *DNScat* [online]. 2005, [Aktualizováno 2005-09-16] [cit. 2015-05-18]. Dostupné z: <http://tadek.pietraszek.org/projects/DNScat/>.

[47] Postel, J. *Domain Name System Structure and Delegation* [RFC 1591 (Informational)]. březen 1994. Dostupné z: <http://www.ietf.org/rfc/rfc1591.txt>.

[48] Qi, C., Chen, X., Xu, C. et al. A Bigram based Real Time DNS Tunnel Detection Approach. *Procedia Computer Science*. 2013, roč. 17, č. 0. S. 852 – 860. First International Conference on Information Technology and Quantitative Management. ISSN 1877-0509.

[49] Radware. *Global Application & Network Security Report 2014 – 2015* [online]. 2014 [cit. 2014-12-28]. Dostupné z: <http://www.radware.com/ert-report-2014/>.

[50] Roolvink, S. *Detecting attacks involving DNS servers: A Netflow data based approach*. prosinec 2008. Enschede: University of Twente. Dostupné z: <http://essay.utwente.nl/58497/>.

[51] Rozekrans, T., Mekking, M. a Koning, J. de. *Defending against DNS reflection amplification attacks*. Amsterdam: University of Amsterdam, únor 2013. 42 s. Dostupné z: <http://www.nlnetlabs.nl/downloads/publications/report-rrl-dekoning-rozekrans.pdf>.

[52] Sattar, U., Naqash, T., Zafar, M. et al. Secure DNS from amplification attack by using modified bloom filters. In *Digital Information Management (ICDIM), 2013 Eighth International Conference on*. New York: IEEE, 2013. S. 20 – 23. ISBN 9781479906147.

[53] Silberschatz, A., Galvin, P. B. a Gagne, G. *Operating System Concepts*. 7. vyd. New Jersey: John Wiley & Sons, 2005. 886 s. ISBN 0-471-69466-5.

[54] Symantec. *2014 Internet Security Threat Report* [online]. 2014 [cit. 2014-12-28]. Dostupné z: <http://www.symantec.com/security_response/publications/threatreport.jsp>.

[55] Tanenbaum, A. S. *Computer networks*. 4. vyd. New Jersey: Prentice-Hall, 2003. 384 s. ISBN 0-13-066102-3.

[56] Valenzuela, T. *Tcp-over-dns* [online]. 2008 [cit. 2015-05-18]. Dostupné z: <http://analogbit.com/software/tcp-over-dns/>.

[57] Vaughn, R. a Evron, G. *DNS Amplification Attacks: Preliminary release* [online]. 2006 [cit. 2014-12-27]. Dostupné z: <http://crt.io/DNS-Amplification-Attacks.pdf>.

[58] Weidendorfer, J. *KCachegrind* [online]. 2013, [Aktualizováno 2013-04-05] [cit. 2015-05-18]. Dostupné z: <http://kcachegrind.sourceforge.net/>.

[59] Zdrnja, B., Brownlee, N. a Wessels, D. Passive Monitoring of DNS Anomalies. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA) 2007*. Berlin, Heidelberg: Springer, červenec 2007. S. 129 – 139. ISBN 978-3-540-73614-1.

Příloha A

Obsah CD

Přiložené CD obsahuje následující adresáře:

- /
  - Program
    - Config files ..... Vzorové konfigurační soubory.
    - Nemea version ..... Zdrojové kódy modulu pro systém Nemea.
    - Standalone version ..... Zdrojové kódy samostatné verze programu.
  - Testing
    - Merged traffic examples .... Ukázky kombinace útoku a normálního provozu.
    - Normal traffic examples ..... Ukázky normálního provozu.
    - Profiling ..... Výsledky profilování.
      - Callgrind files
      - Graphs
    - Tunneling captured files ..... Ukázky zachycených útoků.
      - IPFIX CSV
      - NetFlow CSV
      - NetFlow nfcapd
      - pcap
  - Text ..... Text práce a zdrojové soubory.
    - Source (LaTeX)

Příloha B

Metriky kódu

Samostatná verze programu

Počet zdrojových souborů:	30
Velikost zdrojových souborů:	254 kB
Počet řádků kódu (pouze vlastní kód):	8 481
Velikost spustitelného souboru:	2 579 kB (platforma Linux/x86)

Verze programu pro systém Nemea

Počet zdrojových souborů:	24
Velikost zdrojových souborů:	178 kB
Počet řádků kódu (pouze vlastní kód):	6 051
Velikost spustitelného souboru:	1 834 kB (platforma Linux/x86)

Příloha C

Manuál pro přidání dalších detektorů

Při přidávání nových detektorů je nutné, aby detektory využívaly vzájemné propojení sdílenou frontou. Detektor může být realizován jedinou funkcí, ve které ovšem musí periodicky zpracovávat datové bloky ze sdílené fronty. Sdílené fronty jsou deklarovány jako globální proměnné, takže přístup k nim je bezproblémový. Pokud implementujeme první detektor, musíme bloky číst z fronty `shared_bin_pointers` a detekované bloky ukládat do fronty `shared_bin_pointers_filtered_by_d1`. S tou pak musí pracovat druhý detektor. Inspiraci lze hledat ve funkci `process_queue` v souborech `netflowdetector1.cpp` a `netflowdetector2.cpp`.

Bloky jsou předávány pouze ve formě ukazatelů (jsou alokovány na hromadě). Pokud tedy s blokem dále nechceme pracovat, musíme uvolnit alokovanou paměť. Příklady průchodu blokem a kontroly jednotlivých záznamů první sadou prahů naleznete v souboru `netflowdetector2.cpp` ve funkcích `detect_in_bin` a `detect`. Definice struktur, představující jednotlivé pakety či toky, naleznete v souboru `common.h`.

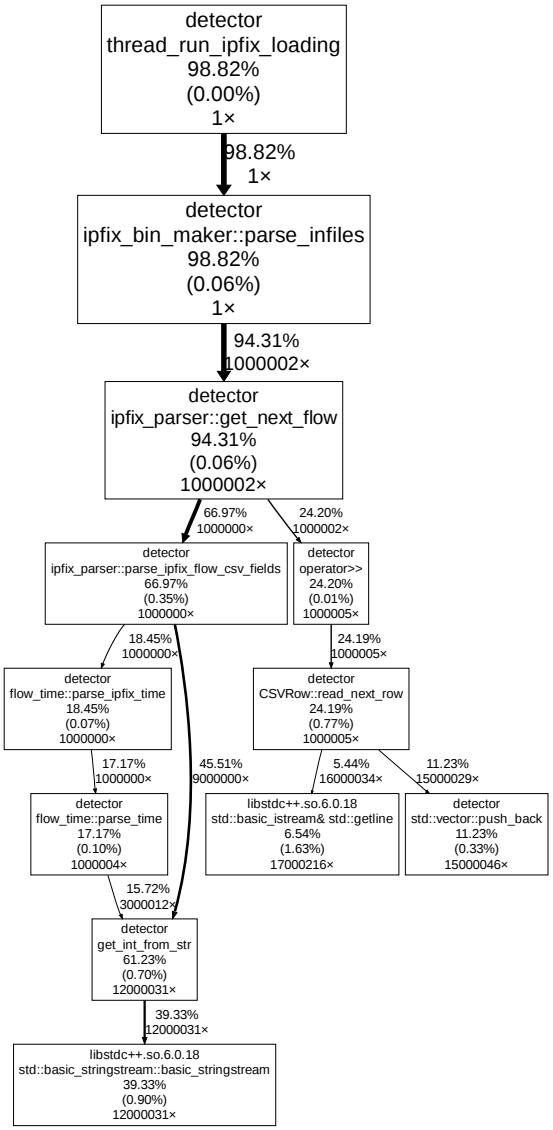
Jako agregační mapu lze použít univerzální třídu `ip_storage`. Ukázku jejího použití představuje funkce `put_ip_data_in_map` v souboru `netflowdetector2.cpp`. Průchod agregační mapou, aplikaci druhé sady prahů a uložení výsledku detekce ilustruje funkce `inspect_aggregated_bins` ve stejném souboru.

Pokud se spokojíte se současnými možnostmi zpracování vstupu, není třeba v implementaci těchto modulů nic měnit. Pokud ovšem chcete přidat nový vstupní formát, je třeba provést několik nezbytných kroků. Během zpracování vstupu je třeba zařídit extrakci položek jednotlivých vstupních záznamů a jejich uložení do příslušné interní struktury. Tyto struktury musí být ukládány do bloků, u kterých musíme hlídat jejich délku. Pro řazení záznamů do bloků dle jejich času lze využít třídu `bin_sorter` a její metodu `sort_into_bin`. Třída automaticky provádí ukládání jednotlivých bloků do fronty sdílené s prvním detektorem. Příklad jejího použití naleznete v metodě `parse_infiles` v souboru `ipfixparser.cpp`.

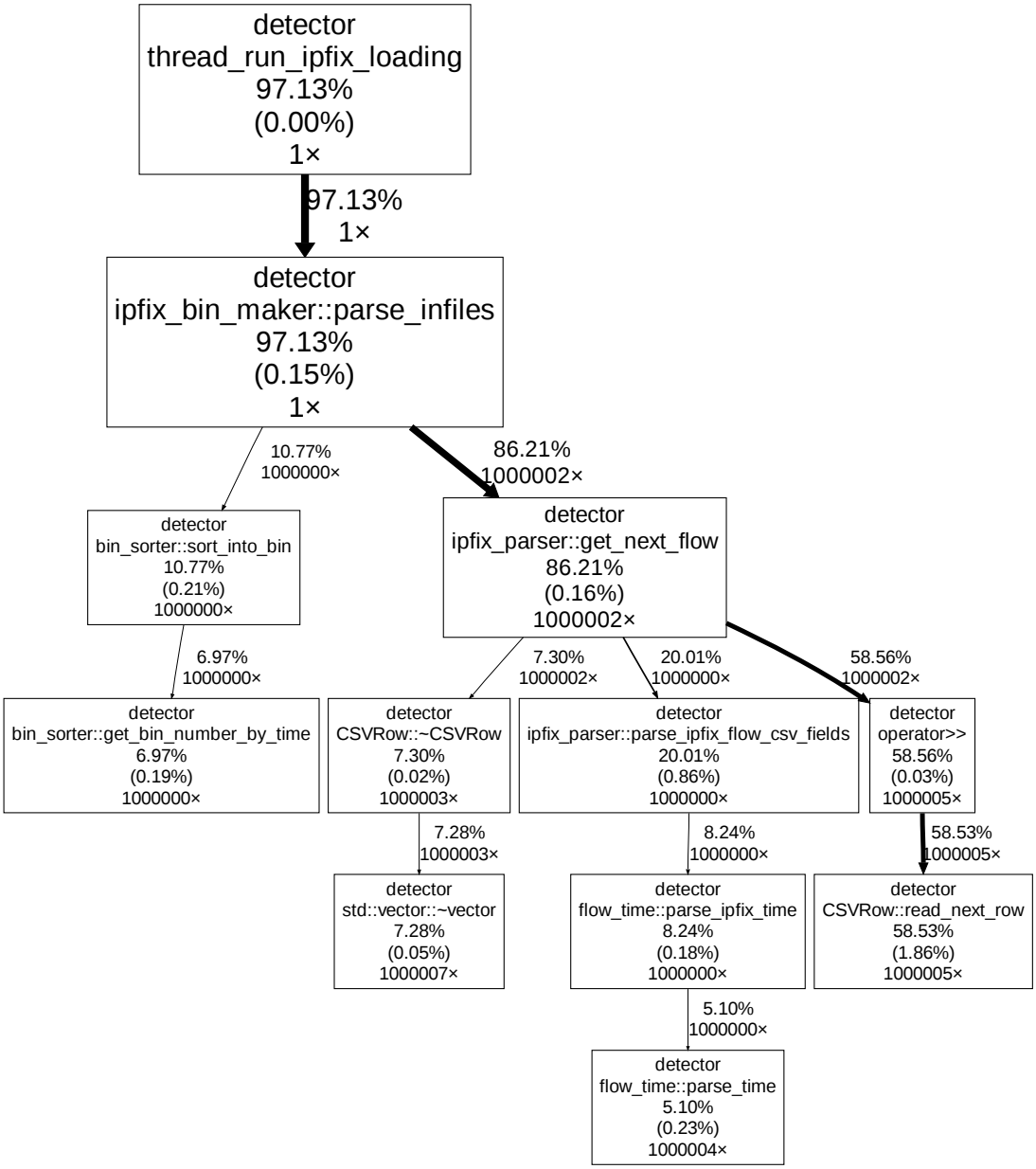
Nakonec je nutné přidat programové parametry pro spuštění nových detektorů a implementovat jejich spuštění. To je třeba udělat v souboru `main.cpp`. Nové parametry programu a jejich zpracování lze přidat ve funkci `parse_params`. Spuštění nových detektorů musí probíhat ve funkci `main`, a to vytvořením vláken pro zpracování vstupu a jednotlivé detektory. Těm musí být přiřazena výchozí funkce, kterou budou vykonávat. Vlákna mohou být následně spuštěna.

Příloha D

Ukázka výsledků profilování



Obrázek D.1: Zpracování IPFIX CSV vstupu před optimalizací.



Obrázek D.2: Zpracování IPFIX CSV vstupu po optimalizaci. Procenta vyjadřují, kolik času z celkové doby běhu programu bylo stráveno v dané funkci. Všimněte si, že se v grafu vůbec nevyskytuje funkce `get_int_from_str`, která před optimalizací zabírala 60 % doby běhu.

Příloha E

Analýza charakteristik vybraných nástrojů pro tunelování přes DNS

Tato kapitola se zabývá analýzou chování některých vybraných nástrojů pro tunelování přes DNS, které jsou volně dostupné.

Informace obsažené v této kapitole byly mimo jiné vypozorovány ze zachycené komunikace a nemusí tedy být platné obecně, v libovolném okamžiku (zejména velikosti paketů závisí na délce použité tunelovací domény). Některé informace byly ovšem převzaty z manuálů či zdrojových kódů samotných zkoumaných nástrojů.

Jednotlivé nástroje se zpravidla skládají ze dvou programů – *klienta* a *serveru*. Server obvykle poslouchá na portu 53 a přijaté dotazy pro nastavenou doménu zpracovává. Klient se pak připojuje a komunikuje se serverem s využitím různých typů záznamů DNS a v nich obsažených datech. Pro udržení spojení jsou většinou využity tzv. *keepalive* zprávy, které se posílají periodicky a slouží pro kontrolu spojení. Některé nástroje pak v pravidelných intervalech přenáší všechny zprávy, zatímco jiné přenáší data shlukově v případě potřeby.

Popisované nástroje se liší především podporovanými typy záznamů, ale mohou se odlišovat i zvláštními technikami pro zvýšení rychlosti či architektonickými prvky.

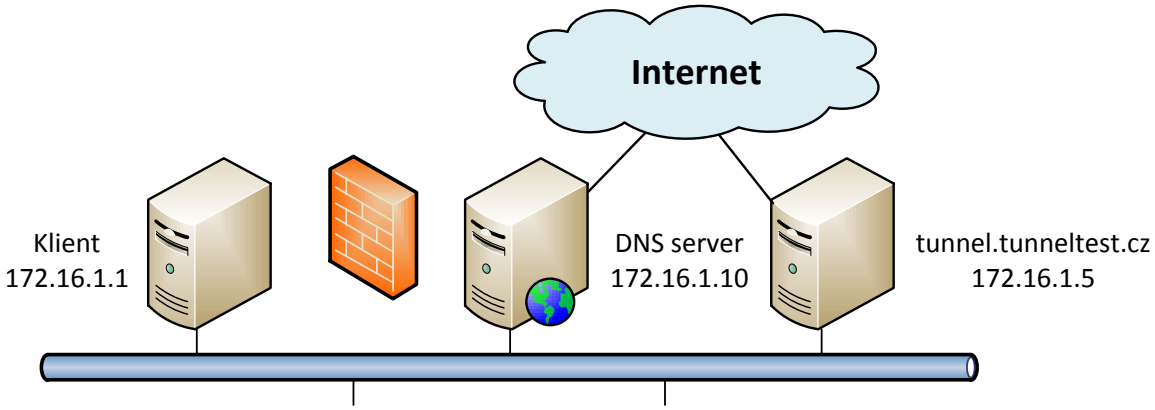
E.1 Testovací prostředí

Pro analýzu chování jednotlivých nástrojů bylo vytvořeno virtuální testovací prostředí. Díky němu pak byla zachytávána komunikace klienta se serverem a následně analyzována.

Prostředí se skládá ze tří virtuálních počítačů (byl využit virtualizační software `VirtualBox`), mezi kterými byla vytvořena privátní síť s adresami ze sítě `172.16.1.0/24`. Všechny počítače běží na OS `Ubuntu 12.04`. První počítač představuje DNS server s DNS software `bind`, na kterém byla vytvořena doména sloužící pro otestování tunelování – `tunneltest.cz` (byla vybrána neexistující doména). Ten je připojen jak do vnitřní sítě, tak do Internetu (kvůli překladu existujících adres).

Druhý počítač představuje tunelovací server `tunnel.tunneltest.cz`, na němž běží serverová část tunelovacích nástrojů a jehož IP adresa je uvedena v DNS serveru (jako autoritativní server pro doménu `tunnel.tunneltest.cz`). Hlavní DNS server tedy přeposílá dotazy na domény `*.tunnel.tunneltest.cz` právě na tento server. I druhý počítač je připojen do Internetu. Zde také probíhalo zachytávání komunikace klienta se serverem.

Třetí počítač představuje klientskou stranu tunelovacích nástrojů a není tedy připojen do Internetu, ale pouze do privátní sítě. Pro překlad jmen využívá DNS server na prvním počítači.



Obrázek E.1: Schéma testovacího prostředí.

E.2 Tunelovací nástroje

Tato kapitola popisuje jednotlivé tunelovací nástroje a také naznačuje spektrum velikostí paketů, které tyto nástroje používají. Grafy zobrazují informace o množině paketů specifické pro daný tunelovací nástroj. Jedná se pouze o DNS pakety zachycené v průběhu tunelování.

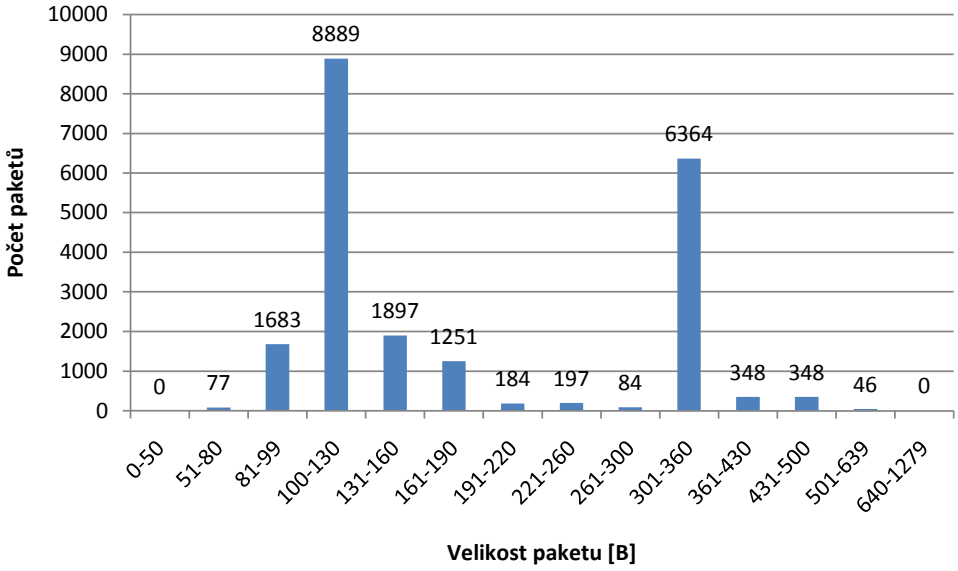
E.2.1 `tcp-over-dns`

Je nástroj pro tunelování napsaný v jazyce *Java*, dostupný z [56].

Zkoumaná data zahrnují SSH komunikaci s tunelovacím serverem a přístup na web pomocí *SSH SOCKS proxy*. Celkem se jedná o 21 368 paketů. Testování probíhalo se základním nastavením.

Typ záznamů	TXT, CNAME
Kódování dat	base63, base16, hexhack37. Komprese dat LZMA.
Časová charakteristika	Rychlost lze nastavit u klientského software (pps[1]). Pro základní nastavení je četnost paketů pro každou stranu komunikace cca 10 pps. U keepalive paketů může být i méně.
Velikosti paketů	Záleží na délce tunelovací domény a použitém typu záznamů. Keepalive pakety u výchozí domény a záznamů TXT: *query* = 110 B/121 B, *response* = 138 B.
TTL odpovědí	30 s
Průměrná délka domény	56, 7 znaků
Ostatní	Pro spuštění vyžaduje interpret jazyka *Java*.

[1] pps=*packets per second*, tedy počet paketů za vteřinu.



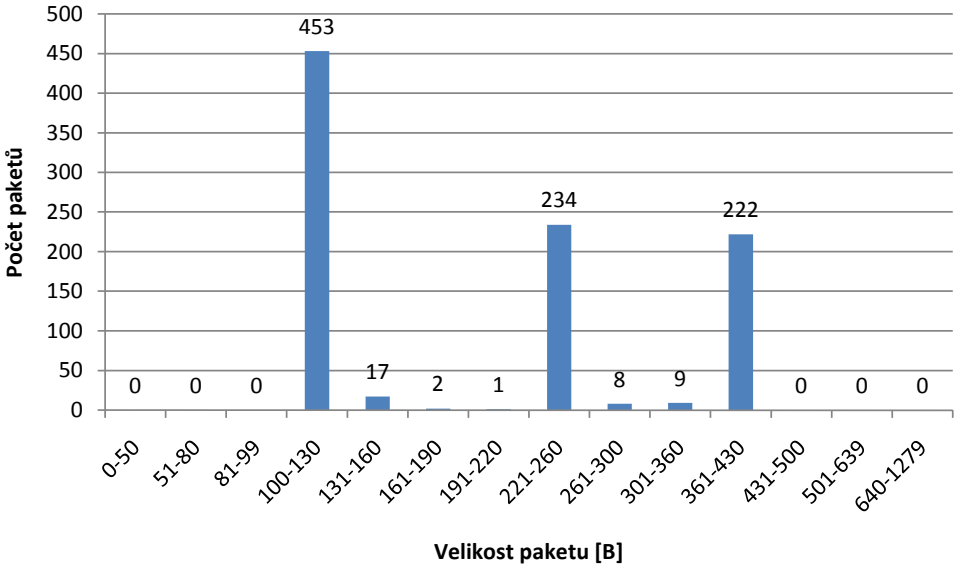
Obrázek E.2: Histogram velikosti DNS paketů nad zkušebními daty pro nástroj `tcp-over-dns`.

E.2.2 `dnscat`

Je jednoduchý nástroj pro tunelování napsaný v jazyce *C*, dostupný z [16].

Zkoumaná data zahrnují komunikaci s tunelovacím serverem spuštěním *bash* na serveru (nešifrováno) a přenos souborů mezi klientem a serverem (také nešifrováno). Celkem se jedná o 946 paketů.

Typ záznamů	NS, CNAME, MX, TEXT, A, AAAA
Kódování dat	NetBIOS, Hex
Časová charakteristika	Lze nastavit u klientského software (frekvence dotazů v *ms*). Pro základní nastavení je četnost paketů cca 1 pps (pro každou stranu komunikace – tedy jeden paket dotazu a jeden paket odpovědi).
Velikosti paketů	Záleží na délce tunelovací domény a použitém typu záznamů. Keepalive pakety u výchozí domény a záznamů CNAME: *query* = 125 B, *response* = 236 B. Keepalive pakety u výchozí domény a záznamů NS: *query* = 126 B, *response* = 239 B.
TTL odpovědí	1 s
Průměrná délka domény	56, 2 znaků
Ostatní	Umožňuje vzdálené spouštění programů. U základního nastavení je na začátku každé použité domény uvedena subdoména `dnscat` (tedy např. `dnscat.21.oyrbqolr.137b3.0.tunnel.tunneltest.cz`).



Obrázek E.3: Histogram velikosti DNS paketů nad zkušebními daty pro nástroj `dnscat`.

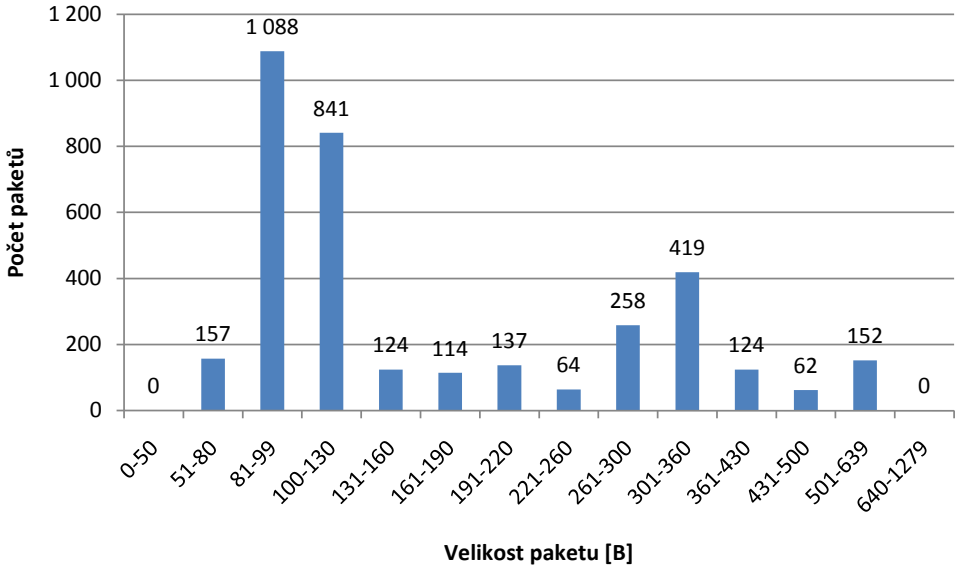
E.2.3 `DNScat`

Je nástroj pro tunelování napsaný v jazyce *Java*, dostupný z [46]. Pro tunelování IP protokolu využívá PPP[2] démona. Nezaměňovat s nástrojem `dnscat` uvedeným výše.

Zkoumaná data zahrnují SSH komunikaci s tunelovacím serverem včetně přenosu souborů. Celkem se jedná o 3 540 paketů.

Typ záznamů	CNAME, A
Kódování dat	Proprietární.
Časová charakteristika	Lze nastavit u klientského software (frekvence dotazů v *ms*). Pro základní nastavení je četnost paketů cca 1 pps (pro každou stranu komunikace – tedy jeden paket dotazu a jeden paket odpovědi).
Velikosti paketů	Záleží na délce tunelovací domény a použitém typu záznamů. Keepalive pakety u výchozí domény a záznamů CNAME: *query* = 89 B, *response* = 128 B.
TTL odpovědí	5 s
Průměrná délka domény	71, 2 znaků
Ostatní	Pro spuštění vyžaduje interpret jazyka *Java*. U základního nastavení jsou subdomény oddělovány tečkou vždy po 30 znacích (pro změnu je nutná změna kódu a nová kompilace programu).

[2] *Point-to-Point Protocol.*



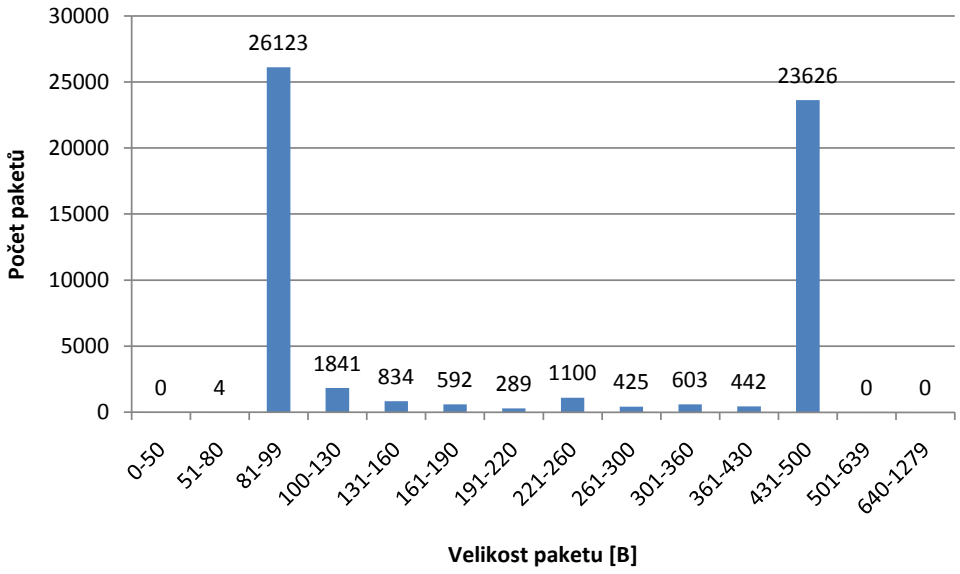
Obrázek E.4: Histogram velikosti DNS paketů nad zkušebními daty pro nástroj `DNScat`.

E.2.4 `Dns2tcp`

Je nástroj pro tunelování napsaný v jazyce *C*, dostupný z [25].

Zkoumaná data zahrnují SSH komunikaci s tunelovacím serverem a přístup na web pomocí *SSH SOCKS proxy*. Celkem se jedná o 55 879 paketů.

Typ záznamů	TXT, KEY
Kódování dat	Base64, možnost komprese.
Časová charakteristika	Rychlost nelze nastavit (automatická). U keepalive zpráv je četnost paketů cca 2 pps (pro každou stranu komunikace – tedy dva pakety dotazu a dva pakety odpovědi). U kontinuálního přenosu dat je četnost i několik desítek pps.
Velikosti paketů	Záleží na délce tunelovací domény a použitém typu záznamů. Keepalive pakety u výchozí domény a záznamů TXT: *query* = 91 B, *response* = 116 B.
TTL odpovědí	3 s
Průměrná délka domény	33, 1 znaků
Ostatní	U základního nastavení jsou subdomény oddělovány tečkou vždy po 63 znacích.



Obrázek E.5: Histogram velikosti DNS paketů nad zkušebními daty pro nástroj `Dns2tcp`.

E.2.5 `iodine`

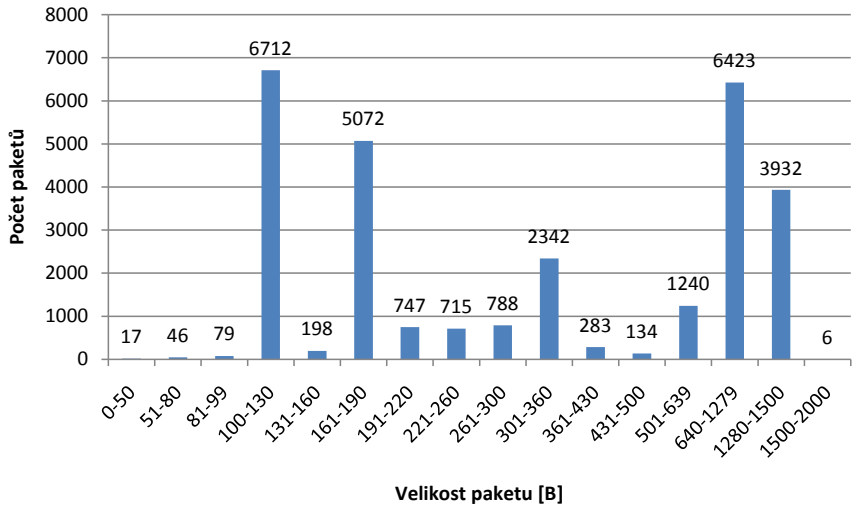
Je nástroj pro tunelování napsaný v jazyce *C*, dostupný z [28]. Pro přenos dat využívá TUN/TAP rozhraní. Oproti ostatním nástrojům používá netradiční typy záznamů, což mu umožňuje zvýšení rychlosti.

Zkoumaná data zahrnují SSH komunikaci s tunelovacím serverem a přístup na web pomocí *SSH SOCKS proxy*. Zkoumány byly jak normální, tak *raw* pakety (viz dále). Celkem se jedná o 28 734 paketů.

Typ záznamů	NULL, PRIVATE, TXT, SRV, MX, CNAME, A. Automatická detekce vybere nejvhodnější typ (pro nejlepší propustnost dat).
Kódování dat	Base32, Base64, Base128, RAW (bez kódování).
Časová charakteristika	Četnost keepalive zpráv lze nastavit, základní nastavení je $\frac{1}{4}$ pps. U kontinuálního přenosu dat je četnost i několik desítek pps.
Velikosti paketů	Záleží na délce tunelovací domény a použitém typu záznamů. Podporuje i velmi velké pakety u typu NULL (velikosti i přes 1300 B). Keepalive pakety u výchozí domény a záznamů NULL: *query* = 100 B, *response* = 103 B.
TTL odpovědí	0 s
Průměrná délka domény	212, 7 znaků

Ostatní

Klient se snaží odesílat tzv. *raw* UDP pakety, kdy celý obsah DNS paketu nahradí UDP paketem. Takové pakety se pak při inspekci jeví jako poškozené DNS pakety[3]. Toto chování lze vypnout.



Obrázek E.6: Histogram velikosti DNS paketů nad zkušebními daty pro nástroj `iodine`.

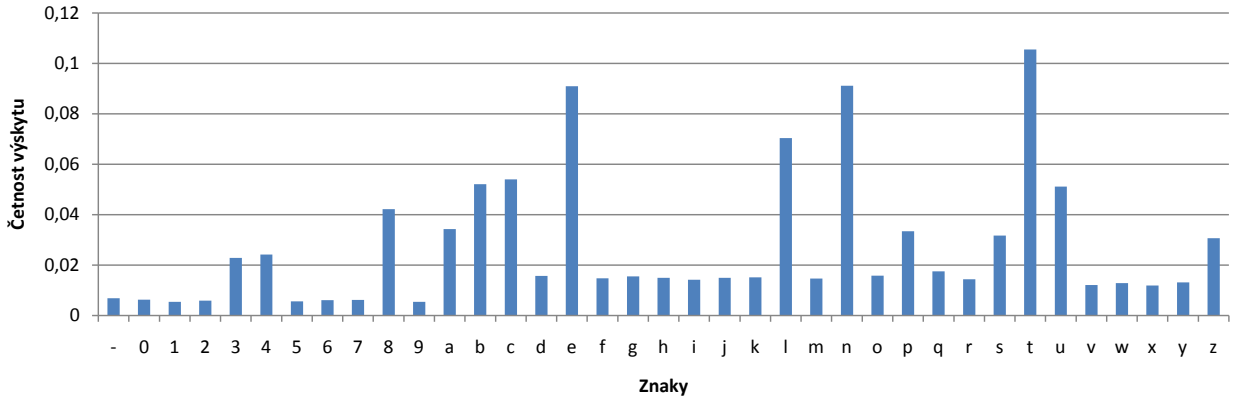
E.3 Frekvenční analýza doménových jmen využívaných tunelovacími nástroji

Tato kapitola popisuje pomocí grafů spektrum a četnost znaků využívaných v názvech domén, které jednotlivé nástroje používají k tunelování. Tato data mohou sloužit k porovnání s četností znaků normálních doménových jmen (standardního DNS provozu), a tedy možnosti detekce tunelování přes DNS na základě rozdílů v těchto datech.

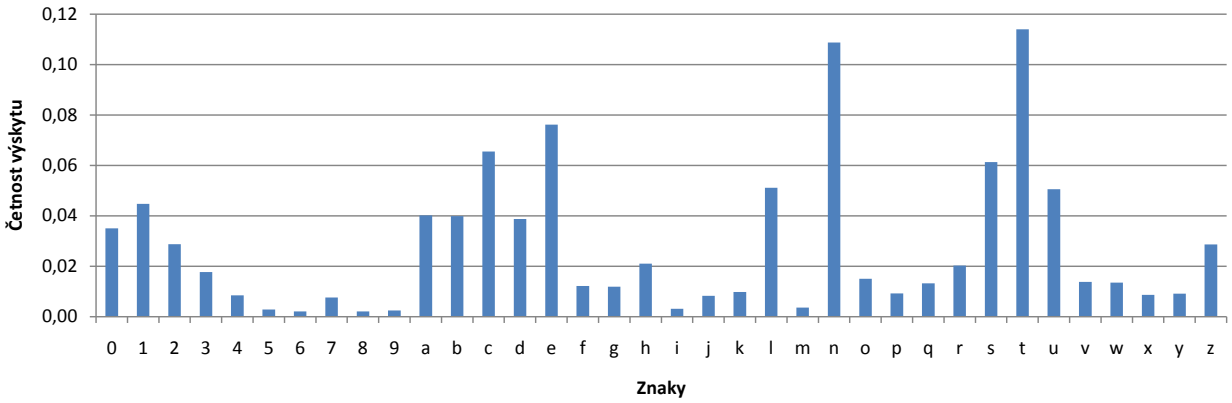
Pro analýzu byly využity názvy domén pouze z tunelovacích paketů (pakety obsahující doménu `tunnel.tunneltest.cz`). Množina paketů je pro každý nástroj specifická, nejedná se tedy o stejná data ani množství paketů. Doménová jména byla použita včetně tunelovací domény `tunnel.tunneltest.cz`, což je třeba brát v úvahu zejména u zvýšeného výskytu písmen *t,e,n*. Uvedené statistiky tak může ovlivnit i poměr *keepalive* paketů ku ostatním datovým paketům (časté keepalive pakety se projeví právě v nárůstu frekvencí výskytů písmen, které obsahuje tunelovací doména).

V posledním grafu (E.12) je pro porovnání ukázka četnosti znaků v doménových jménech u reálného provozu. Data byla vytvořena analýzou 1 500 000 doménových jmen (IPFIX vstupní soubor).

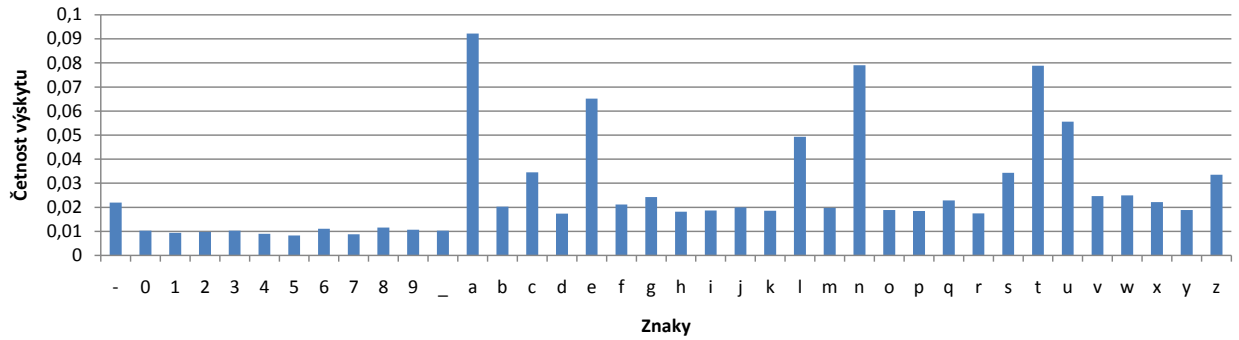
[3]Tyto pakety nelze zpracovat stejným způsobem jako u normálních DNS paketů, protože při jejich zpracování může dojít k neoprávněnému přístupu do paměti (to je způsobeno binárními daty libovolných hodnot na místech původně zpracovávaných hlaviček či při extrakci doménového jména). Po inspekci zdrojových kódů a protokolu nástroje `iodine` byla nalezena konstanta, díky které lze tyto pakety detekovat a vyhnout se tak jejich zpracování. Touto konstantou je hodnota `0x10D19E`, která je umístěna v hlavičce DNS paketu v sekci ID a horním bytu příznaků (*flags*). Při zpracování takového paketu jsou dosazeny hodnoty velikosti a domény tak, aby došlo k jeho detekci (nicméně detekce bude obsahovat nepravdivou tunelovací doménu).



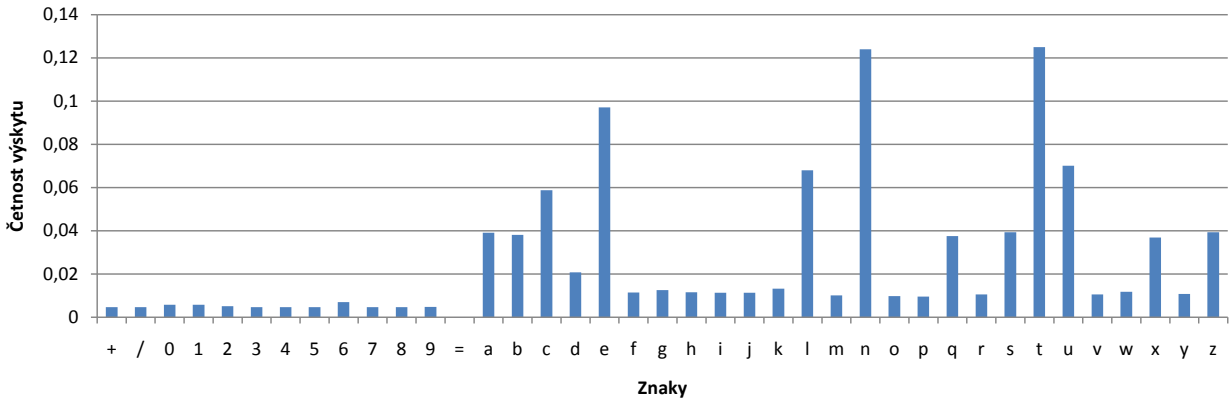
Obrázek E.7: Četnosti výskytů znaků v doménových jménech pro nástroj `tcp-over-dns`.



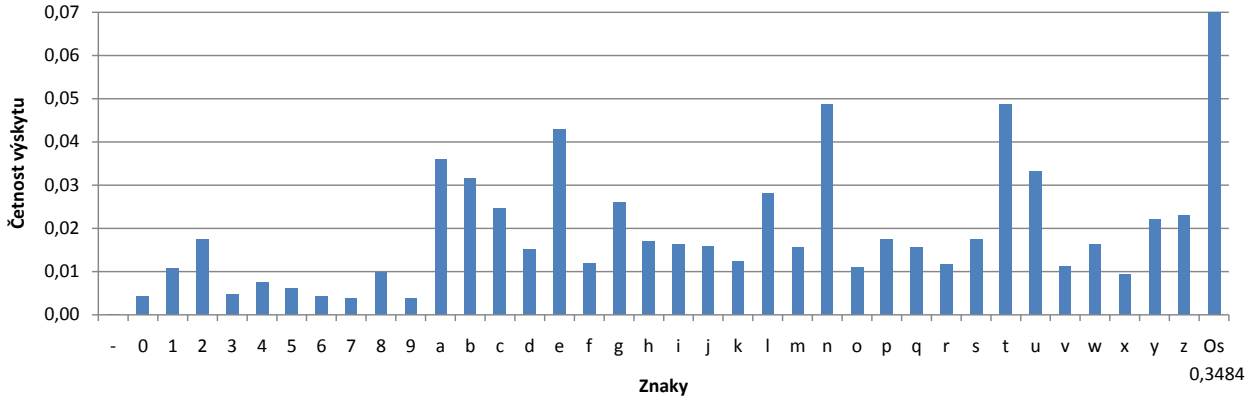
Obrázek E.8: Četnosti výskytů znaků v doménových jménech pro nástroj `dnscat`.



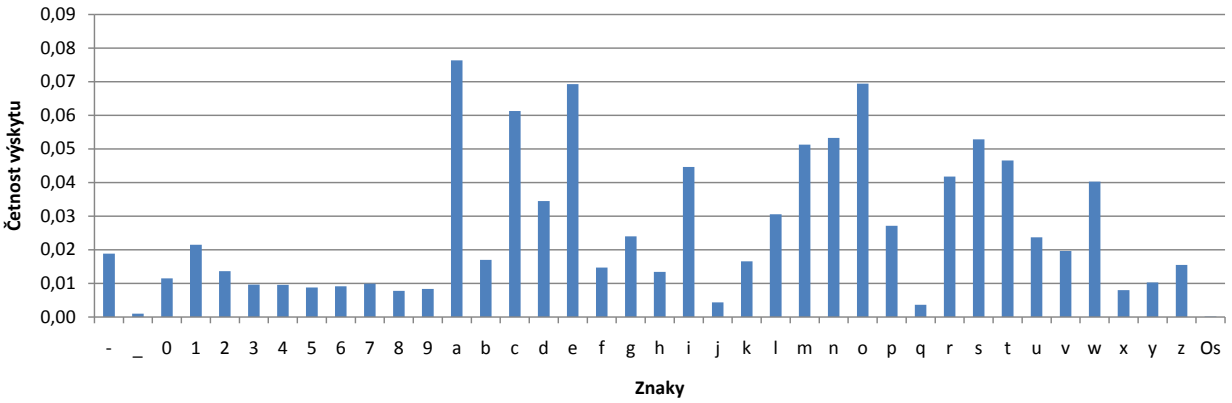
Obrázek E.9: Četnosti výskytů znaků v doménových jménech pro nástroj `DNScat`.



Obrázek E.10: Četnosti výskytů znaků v doménových jménech pro nástroj `Dns2tcp`.



Obrázek E.11: Četnosti výskytů znaků v doménových jménech pro nástroj `iodine`. Sloupec „Os“ byl zmenšen a jeho skuteční hodnota frekvence výskytu je uvedena pod ním. Tento sloupec představuje ostatní speciální znaky, které nejsou součástí anglické abecedy. Ty může nástroj využít díky použití záznamů typu NULL.



Obrázek E.12: Četnosti výskytů znaků v doménových jménech u normálního síťového provozu.