Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra ... teoretické informatiky

Bakalářská práce

Detekce síťových tunelů v počítačových sítích

Zdeněk Rosa

Vedoucí práce: Ing. Tomáš Čejka

13. května 2014

Poděkování

Za odborné a metodické vedení, cenné rady a připomínky při tvorbě bakalářské práce upřímně děkuji Ing. Tomáši Čejkovi.

Dále bych chtěl poděkovat organizaci CESNET, z. s. p. o., za poskytnutí potřebných nástrojů a dat pro tvorbu této práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či spracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2014 .....

Odkaz na tuto práci

Rosa, Zdeněk. *Detekce síťových tunelů v počítačových sítích.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstract

This thesis is focused on a detection of communication tunnels over DNS and HTTP. This text describes the design and the implementation of a detection module to discover the tunnels over DNS. The module is based mainly on a combination of statistical analysis of several traffic and payload features. Because of huge volume of DNS data from large networks, the thesis analyzes appropriate data structures for efficient insert, search and delete of network addresses and domain names. The detection module was evaluated using a simulation environment and using a real network traffic from the perimeter of the CESNET2 infrastructure Czech National Research and Education Network (NREN). Results of the detection module that was loaded by the real network traffic are presented at the end of this work.

Keywords anomaly detection, tunneling protocols, DNS, HTTP, efficient data structures, IP addresses, domain names.

Abstrakt

Tato práce se zabývá průzkumem známých metod detekce tunelování přes DNS a HTTP, dále návrhem a implementací modulu pro detekci tunelů přes DNS. Detekční modul provádí analýzu síťových toků a analýzu obsahu DNS zpráv. V této práci byla provedena analýza vhodných datových struktur, které umožňují efektivní ukládání, vyhledávání a mazání síťových adres a doménových jmen. Detekční modul byl otestován v simulovaném prostředí virtuální sítě a na reálných datech ze sítě CESNET2. V závěru textu práce jsou uvedeny výsledky detekce z modulu nasazeného do reálného síťového provozu

Klíčová slova detekce anomálií, tunelování protokolů, DNS, HTTP, efektivní datové struktury, IP adresy, doménová jména.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Sítová bezpečnost je v dnešní době velice důležitým aspektem, na který bychom neměli zapomínat. Většina z nás bere internet jako nedílnou součást života, využíváme jej na mobilních telefonech, počítačích, televizích a mnoha dalších zařízeních. Při komunikaci na internetu bychom neměli opomíjet i jeho negativní stránku a měli bychom se snažit vyvarovat nástrahám, které nám hrozí.

Tunelování protokolů je jedna z možností napadení počítačové sítě. Pro tunelování je potřeba dvou stran a to klienta a serveru. Klient nacházející se obvykle v síti s omezeným přístupem k internetu, se přes nějaký povolený protokol (například DNS) připojí ke vzdálenému serveru, který tato omezení nemá. Klient kóduje a zapouzdřuje data do tunelovaného protokolu. Server přijímající taková data je nastavený, aby je dekódoval a poslal regulérně na internet. Odpovědi, které mu přijdou, zapouzdří jako odpověď daného protokolu a pošle je zpět klientovi. Účelem je, aby tato komunikace vypadala jako legitimní a nikdo jí na první pohled nerozpoznal. Tato komunikace však porušuje pravidla dané sítě, ale především ohrožuje síť. Útočník může síťový provoz zneužít k nekalým účelům, jako je například vynášení informací, stahování škodlivého softwaru, zpřístupnění lokální sítě z vnějšku a mnoho dalších.

Cíl bakalářské práce

Nástroje, jako je firewall nebo proxy server, se snaží omezit komunikaci tak, aby zamezily nevyžádanému přístupu a eliminovaly tak riziko porušení bezpečnosti komunikace po síti. Existují však techniky, které umožňují tyto nástroje obejít. Jedná se například o tunelování přes protokoly DNS nebo HTTP. Cílem práce je nalézt řešení pro detekci síťových tunelů vedených přes DNS nebo HTTP a implementovat toto řešení ve formě detekčního modulu. Na základě dohody s vedoucím bakalářské práce bude provedena analýza mechanismů a nástrojů pomocí obou protokolů a dále bude implementován modul detekující DNS tunely a jiné anomálie. Výsledek této práce (detekční modul) by měl

přispět ke zvýšení síťové bezpečnosti.

Ze zadání vyplývají následující úkoly:

- nastudovat problematiku tunelování pomocí protokolů DNS a HTTP, které nejsou primárně určeny pro vytváření komunikačních tunelů,
- nalézt či navrhnout metody pro detekci tunelů na základě naměřených dat obsahujících tunelované spojení,
- navrhnout strukturu detekčního modulu a diskutovat použití efektivních algoritmů a datových struktur,
- implementovat navržený detekční modul,
- otestovat funkčnost modulu v simulaci na experimentálním prostření,
- otestovat modul na provozu sítě CESNET2.

Modul je vytvářen ve spolupráci s organizací CESNET, z. s. p. o. (Cesnet) a je součástí systému Nemea, který slouží pro analýzu a detekci anomálií na síti CESNET2.

Má bakalářská práce je rozdělena do pěti částí:

V kapitole 1 se zabývám analýzou tunelů vytvořených pomocí DNS. Nejprve velice stručně popisuji protokol DNS a možnosti jeho tunelování. Poté představuji několik tunelovacích nástrojů a známé metody pro detekci tunelování přes DNS. Nakonec analyzuji síť obsahující tunelovanou komunikaci.

V kapitole 2 se zabývám analýzou dostupných tunelovacích nástrojů a mechanismů používajících pro přenos uživatelských dat HTTP.

V kapitole 3 se zabývám návrhem modulu pro detekci tunelování přes DNS. Nejprve je proveden výběr vhodných metod pro detekci, poté je popsáno jejich propojení do funkčního celku. Nakonec je na názorném diagramu zrekapitulována aktivita modulu.

V kapitole 4 se zabývám popisem konkrétní implementace modulu. Nejprve popisuji výběr nástrojů pro tvorbu modulu, poté předzpracování dat, které bude modul přijímat. Protože modul vyžaduje efektivní ukládání a vyhledávání určitých dat, věnuji se výběru vhodných datových struktur a jejich implementaci. Nakonec je popsána implementace samotného modulu.

V kapitole 5 popisuji výsledky testování modulu. Modul byl nejprve testován v simulovaném prostředí virtuální sítě a poté na anonymizovaných datech ze sítě CESNET2. Na konci této kapitoly se nachází zhodnocení funkčnosti modulu.

Kapitola 1

Analýza tunelování přes DNS

1.1 DNS

Domain Name System neboli DNS je hierarchický systém doménových jmen. Je tvořen DNS servery a DNS protokolem. Slouží především pro překlad doménových jmen na IP adresy a zpět. Protokol používá většinou port 53 a to jak TCP tak UDP, komunikuje na principu dotaz, odpověď.

DNS servery jsou organizovány hierarchicky, stejně jako názvy domén. Jedná se o strom s jedním kořenem nazývaným nazývaným doména nultého řádu. Na tento kořen se postupně větví potomci.

Celé doménové jméno se skládá z částí oddělených tečkami. Napravo se nachází nejobecnější doména (top level domain), která se směrem doleva konkretizuje. Vyhledávání probíhá od top level domain, až po konkrétní záznam. Nazpět je poté vrácena adresa tohoto konkrétního serveru. Bližší popis v [23].

Pro DNS existuje mnoho typů záznamů [19], v textu se budu zabývat těmito:

- A
  IP adresa verze 4, která je přiřazena určitému doménovému jménu.
- AAAA
  IP adresa verze 6.
- CNAME
  Jedná se kanonické jméno, které směřuje na stejnou adresu. Typicky se používá pro automatické doplnění *www*.
- MX
  Adresa poštovního serveru určité domény. Obsahuje prioritu serveru a doménové jméno serveru.
- NS
  Autoritativní server pro danou doménu.

1.2 Tunelování přes DNS

DNS tunelování se používá tam, kde máme určitým způsobem blokovaný přístup mimo lokální síť. Mohou být povoleny pouze určité protokoly, například pouze HTTP. Pro navázání síťové komunikace za použití doménových jmen je zapotřebí DNS překlad. Proto je na této síti povolen přístup k lokálnímu DNS serveru, který má privilegium přeposílat dotazy na autoritativní servery, které se nacházejí na internetu. Lokální server proto může být zneužíván pro přeposílání jinak blokované komunikace do internetu.

Při DNS tunelování se data pro odeslání zakódují jako hostname dané domény určené pro překlad. Existuje několik druhů kódování pro přenos dat. Kódovaná hostname je přidána k doménovému jménu vzdáleného serveru. Celý dotaz pro překlad je poté zaslán lokálnímu DNS serveru, který přeposílá požadavek dále autoritativnímu serveru. Přeposílání trvá tak dlouho, dokud se žádost o překlad nedostane až k požadovanému vzdálenému počítači. Ten místo překladu kódovanou hostname dekóduje. Poté zpracuje data, která byla v žádosti o překlad zaslána, a nakonec odpoví klientovi pomocí DNS response. Data pro přenos tentokrát uloží do některého z textových atributů v odpovědi. Tento mechanismus je znázorněn na obrázku 1.1.

Dalším přístupem k tunelování přes DNS je vložení binárních dat do paketu s hlavičkou DNS. Takovýto paket musí být přímo poslán na určitou IP adresu, jinak by data nebyla doručena na konkrétní server. DNS servery by si sice přeposílaly žádost, ale protože by v paketu nebyla reálná destinační doména, žádost by nemohla být úspěšně doručena.

1.2.1 Kódování dat

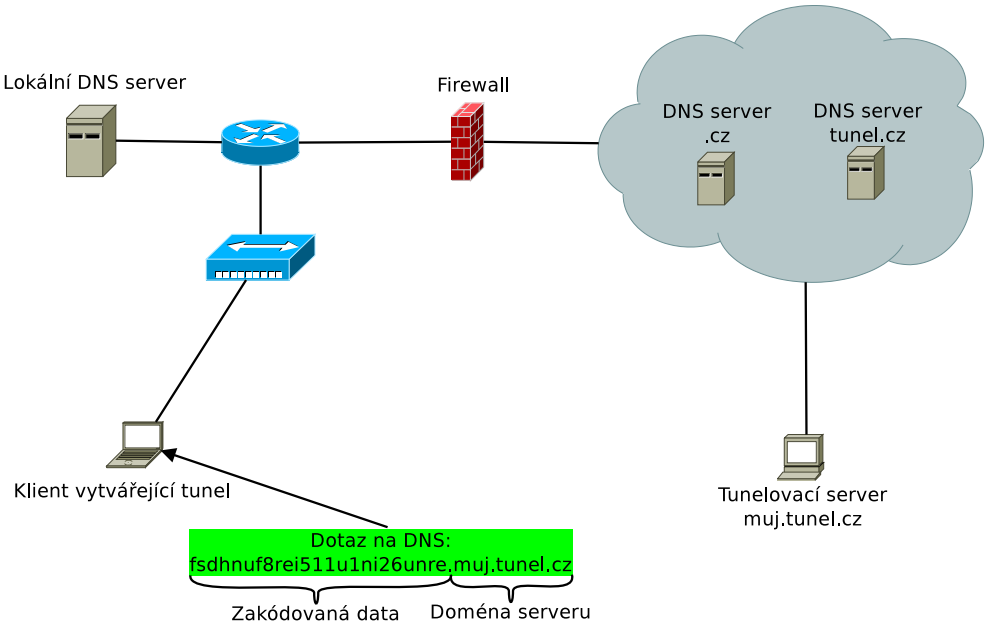
Klient pro odeslání dat potřebuje data nejprve zakódovat takovým způsobem, aby byla přenositelná jako textový řetězec (obsahovala pouze tisknutelné znaky). Tento text je poté zapouzdřen do domény. Existuje několik metod kódování, kterými se liší jednotlivé tunelovací nástroje.

Podle [17] mají domény následující omezení: maximální délka celé domény 255 znaků a délka každé ze poddomén 63 znaků.

Článek [17] zároveň uvádí možné typy kódování dat, které se liší především v počtu kódovaných bitů. Počet bitů ovlivňuje přenosovou rychlost komunikace tunelu.

Base32 kódování
: Base32 je pěti bitové kódování, kdy využíváme 32 hodnot. Toto kódování se používá pro DNS žádosti, ve kterých se mohou vyskytovat písmena A–Z (doména není case-sensitive), číslice 0–9 a znak „-“, celkem tedy 37 možných hodnot.

Base64 kódování
: Base64 je šesti bitové kódování, kdy využíváme 64 hodnot. Toto kódo-



Obrázek 1.1: Notebook má kvůli firewallu blokovaný přístup do sítě. Vytváří DNS tunel, data kóduje do DNS žádostí s příponou domény svého tunelovacího serveru *muj.tunel.cz*. Takto zakódované žádosti odesílá na lokální DNS server, který nezná odpověď, a proto žádost přeposílá na autoritativní server *.cz*. Ten ví kde se nachází DNS server *tunel.cz* a odesílá k němu data. Tento server přeposílá data serveru *muj.tunel.cz*, který se již zbytek domény nesnaží přeložit a rovnou dekóduje zakódovaná data ve zbytku domény. Zpět odešle odpověď zakódovanou jako DNS response.

vání se používá pro odpovědi zakódované v záznamu TXT, ve kterém se mohou vyskytovat písmena A–Z (case-sensitive), číslice 0–9 a znaky „-+“, celkem tedy 64 možných hodnot.

Binární (8 bitové) kódování

Jedná se o osmi bitové kódování. To znamená, že se data použijí tak jak jsou (nemusí se kódovat), pouze se přidají informace nutné pro komunikaci. Tato data se poté posílají v experimentálním NULL záznamu. Výhoda tohoto kódování je vyšší propustnost. Tento formát nepodporují všechny DNS servery. Může se tak stát, že paket bude zahozen.

NetBIOS kódování

NetBIOS je kódování, popsané například v [5], které se snaží zneviditelnit počtem použitých písmen. Každý bajt se rozloží na dvě části, kde každá má 4 b. Ke každé části se přičte hexadecimální číslo $0x41$ (znak 'A' z ASCII tabulky). Výsledkem je dvojice písmen, která může

být z rozsahu AA–PP. Toto kódování je velice neefektivní, kvůli velikosti kódované informace v datech, ale je mnohem lépe chráněné proti detekci.

Hex kódování
Každý bajt je kódovaný do dvou bytů a to pouze přepisem do hexadecimální podoby. Například $'A' = 0x41$, bude zakódováno jako „41“. Kódování je podobně ztrátové jako NetBIOS, ale na rozdíl od něj, zde se bude vyskytovat podezřele mnoho číslic. Bližší informace o kódování v článku [5].

1.3 Tunelovací nástroje

Existuje mnoho nástrojů na tunelování protokolů [22]. Tyto nástroje jsou volně dostupné na internetu. Některé společnosti [33] dokonce zprostředkovávají za poplatek připojení tunelu k jejich serverům. To znamená, že útočník si pouze stáhne program, zaplatí poplatek za připojení a vybere si, přes který protokol chce komunikovat. U verzí zdarma si musí uživatel nainstalovat klientskou aplikaci na počítač v síti, ze kterého chce tunelovat a serverovou aplikaci například na domácím počítači.

Článek [17] uvádí několik nejznámějších programů pro tunelování protokolu DNS:

Iodyne
Nástroj [8] napsaný v jazyce C, poprvé vydán v v roce 2006. Spustitelný pod: Windows, Linux, Mac OS X, Android.

Dns2tcp
Nástroj [3] napsaný v jazyce C, spustitelný pod Linuxem, klientská strana může být i na Windows. Odpovědi kóduje do TXT záznamu.

DNScapy
Nástroj [4] data šifruje pomocí SSH a poté posílá přes DNS. Odpovědi kóduje do CNAME nebo TXT záznamu, anebo obou zároveň.

DNScat
Vyšli dvě verze, první v roce 2004/2005, spustitelná na Unixových systémech. Odpovědi kóduje do CNAME a A záznamu.

Druhá verze [5] byla napsána v roce 2010. Je spustitelná pod systémy Linux, Mac OS X a Windows. Žádosti kóduje pomocí NetBIOS nebo Hex. Odpovědi kóduje do A, AAAA, CNAME, NS, TXT, a MX záznamu.

Ostatní
Existuje plno dalších nástrojů na tunelování. Popis mnoha z nich lze nalézt v [17].

1.4 Známé metody detekce

Detekci rozdělíme jako v článku [17] na payload a traffic analysis. Traffic analysis je detekce na základě statistických údajů, které zaznamenáváme z komunikace (velikost dat, frekvence komunikace, ...), tato analýza probíhá vždy po určitém časovém úseku nebo po naměření potřebných údajů. Naproti tomu payload analysis může probíhat okamžitě po přijetí paketu. Je to analýza zabývající se konkrétním obsahem paketu.

1.4.1 DNS

- Payload analysis
  - Velikost žádosti a odpovědi

    Útočník se snaží dosáhnout co nejvyšší propustnosti, proto do žádosti ukládá co nejvíce dat. Názvy hostname tak dosahují až 63 znaků a celková doména je dlouhá až 255 znaků [17]. Podle [18] bychom měli jako podezřelé brát všechny hostname delší jak 52 znaků. Neúměrně velké budou i odpovědi na dotazy přicházející ze serveru.
  - Entropie doménových jmen

    Podobně jako je tomu v přirozeném jazyce, který má díky pravděpodobnosti výskytu znaků určitou entropii, tak i doménová jména mají svou. Pokud má doménové jméno vysokou entropii, tedy pravděpodobnost výskytu znaků je rovnoměrně rozdělená (všechny znaky mají podobnou pravděpodobnost výskytu), můžeme tuto informaci považovat za indikátor nelegitimního provozu.
  - Statistická analýza

    Doménová jména většinou obsahují pouze málo číslic. Opačně je tomu u jmen tunelovaných domén, ve kterých se objevuje mnoho číslic. Dalším znakem je počet unikátních písmen v doméně. Kódování způsobuje rovnoměrné rozdělení jednotlivých písmen. Proto má tunelovaná doména více unikátních písmen než legitimní.
  - Specifické znaky

    Každý tunelovací nástroj přidává do dat určité informace, které dopomáhají ke zpětnému dekódování dat. Z pravidla jsou tyto informace posílány v každé žádosti/odpovědi. Tunelovacích nástrojů existuje spoustu a každý může mít specifické znaky odlišné. Proto je téměř nereálné implementovat modul, který by zvládl detekovat všechny nástroje.

- Traffic analysis
  - Množství komunikace na IP adresu

    Před navázáním spojení se serverem, zasílá klient několik žádostí o překlad domény. Rozdílně je tomu při tunelování, kdy v řádech několika sekund jsou na daný server posílány stovky dotazů.
  - Množství komunikace na doménu

    Další možností je kontrola nadměrného počtu požadavků na určité doménové jméno n-tého řádu. Nevýhodou je, že na jeden server může odkazovat několik zcela rozdílných domén. To by znamenalo snížení úspěšnosti detekce.
  - Počet hostname na doménu

    Při tunelování má každý požadavek unikátní hostname, ale vyšší řád domény je stále stejný. Měřením počtu hostname na určitou doménu, můžeme zjistit počet potomků. Pokud je jich více než určitý počet, může se jednat o tunel.

1.5 Analýza provozu na síti

Pro analýzu jsem potřeboval získat data, ve kterých se nachází tunelovaná komunikace a data, která jsou legitimní. Pro zachytávání komunikace jsem použil nástroj Wireshark [15].

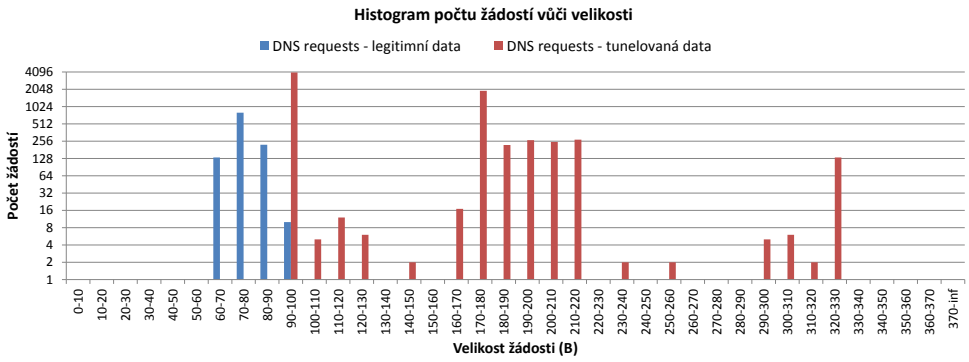
Pro vytvoření tunelu jsem použil nástroj Wi-Free [33]. Tento nástroj poskytuje připojení k DNS tunelovacímu serveru poskytovatele aplikace. Dotazy určené pro DNS tunelovací server museli projít přes několik jiných serverů, než se dostaly k cíli. Díky tomu byly rychlost, frekvence, ztrátovost paketů a další naměřené parametry stejné jako v reálném provozu. Jedinou nevýhodou bylo, že nástroj je komerční a jeho užití je zpoplatněné. Poskytovatel [33] však nabízí možnost nástroj vyzkoušet na komunikaci s daty poskytnutými přímo od něj.

Pro měření dat byl v simulované síti spuštěn tunelovací nástroj, který načítal stránky generované poskytovatelem aplikace, zároveň v síti byla přenášena legitimní DNS komunikace. Záznam trval 6 minut a celé měření bylo prováděno opakovaně.

1.5.1 DNS žádosti

Na obrázku 1.2 je histogram žádostí. Horizontální osa určuje velikost paketu v bajtech a vertikální osa určuje počet paketů této velikosti. Modře je znázorněna legitimní komunikace a červeně komunikace obsahující tunely. Je vidět, že tunelovaná komunikace má větší velikost než legitimní komunikace, která zde má od 70 do 90 bytů. Největší extrémy se nachází ve velikosti 90–100 B.

Ukázka doménovým jmen, která se v žádostech objevila:



Obrázek 1.2: Histogram počtu žádostí v závislosti na velikosti paketu. (Šesti minutový záznam komunikace)

```
Paaapiamci1gq.x2w.us
Paaapiamei1ia.x2w.us
Paaapiaici1lq.x2w.us
Paaapiaiei1mq.x2w.us
```

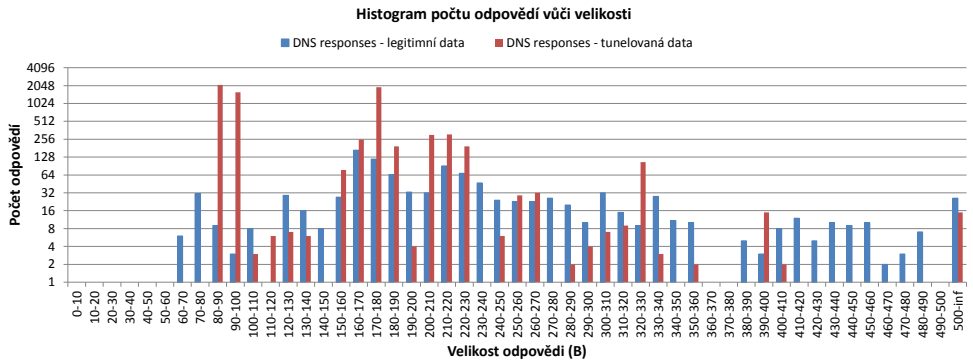
Společná část doménových jmen v ukázce je *x2w.us*, což pokládáme za určení tunelovacího serveru. Na tento server budou z ostatních DNS serverů přeposílány dotazy zakončené touto sekvencí. Nyní se dostáváme ke třetímu řádu domény. Jak je vidět, prefix této poddomény, *Paaapia* je stejný u všech paketů. Zřejmě se jedná o ID přidělené serverem poskytovatele nebo o specifický znak tunelovacího nástroje. Zbytek řetězce je u každé domény unikátní. Jedná se zřejmě o potvrzovací sekvence, které udržují navázané spojení (stream), nebo zprávy potvrzující přijatá data.

Další extrém se nachází ve velikosti 170–180 B. Pokud se podíváme na payload v této oblasti, zjistíme, že se zde nachází dotazy typu:

```
Daaapiamaij.-p++++++++9XPbqY9P88QeZ3AaixLb-TDyoqugQ6w7Sy8JcDJW+
    mEhvE5.OnM1y1r6vqWQEGEJAnZdAmE3o8+jM.x2w.us
Daaapiaibin.Z+++++++++8
    mrTIG60RbO5NIvKmMUO3f7m0Ilgo87oGbf2VH5nsMJiP0y.
    hpq89ss5zigfwuoEWJfkftTTKlQTM.x2w.us
```

Opět žádosti začneme rozebírat od konce domény. Jako v předchozím případě *x2w.us* je jednoznačné určení tunelovacího serveru. Poddoména třetího a čtvrtého řádu jsou již zakódovaná data. Jak můžeme vidět, jsou zde malá i velká písmena, číslice, občas se objeví i znak „-“ nebo „+“ z toho usuzuji, že se jedná o kódování Base64. Četnost jednotlivých znaků je zde až na začátek poddomény čtvrtého řádu rovnoměrná. Poddoména pátého řádu je zřejmě identifikátor komunikace.

Další část komunikace ve velikosti 180–220 B je již méně četnější. Žádosti vypadají takto:



Obrázek 1.3: Histogram počtu odpovědí v závislosti na velikosti paketu. (Šesti minutový záznam komunikace)

```
Daaapiaicij.XVB+++++++8
    tKJ9Yxdn9KGNEOObaL6hTIalBrYa3dsfMcrmKaf7ByzOtx.uA6Ns-
    nvsdwd3jscPHKwsyCbgLqFBPVefC+9p-j3ZVCOO-02rFaBluggP.
    DVXgXitEYO4St8.x2w.us
```

Jedná se o komunikaci stejného typu. Tento nástroj používá jako maximální délku poddomény 58 znaků, proto přibyl další řád poddomény, ve kterém jsou kódovány data.

Poslední a největší část komunikace ve velikosti 320–330 B, již není tak častá. Žádosti vypadají takto:

```
Daaapiaicab.FV++++++++9-J8C8FR3bL+P3L+ZLPb2XZCvg7LYNqwo-
    BvjMjODlt4U91.
    sv7KFx672PumRw8Zkz2gZWUaFhuNaKOfQ2IsVKRZMh5I3vp5U1aq05qQV.
    o8ht+jU2qSNm5rqNbdXdDPTnaf8a39lUYGOfFV2JE8lO6JaJOXDdDoSkg.DAC
    -GMaj7klra4TVy3+bnTO9jl4lhIk+AkavZiqgKy3fjakMjSzIDgKvg.x2w.us
```

Žádosti jsou velice podobné předchozím, jediný rozdíl je délka domény. Opět došlo k navýšení počtu poddomén oproti minulému případu, z důvodu přetečení maximální velikosti domény (58 znaků), kterou používá tento nástroj.

1.5.2 DNS odpovědi

Na grafu 1.3 je vidět histogram komunikace. Stejně jako u žádostí, horizontální osa znázorňuje velikost paketů v bajtech a vertikální osa znázorňuje počet paketů. Červenou barvou je označena tunelovaná komunikace a modrou legitimní komunikace.

Rozdíl legitimních odpovědí oproti legitimním žádostem je především ve velikosti paketů. Zde nastává hlavní špička ve velikosti 160–240 B. V odpovědích se většinou nachází několik A záznamů, AAAA záznamů, ojediněle CNAME, TXT, NS záznamy.

Největší podíl na komunikaci má tunel. Hlavní špička se nachází ve velikosti 80–100 B. Jsou to odpovědi na dotazy, které zajišťují udržení komunikace (stream). Posílá se jen kopie dotazu se záznamem NULL, který je prázdný.

Další významný podíl na komunikaci se nachází ve velikosti 150–230 B a 320–330 B. Zde se posílají data zakódované v NULL záznamu. Nástroj používá 8 bitové kódování, které má větší propustnost.

Tunely v odpovědích se vyhledávají mnohem hůře než v žádostech, protože jak je z grafu vidět, i legitimní komunikace může dosahovat různé velikosti.

Kapitola 2

Analýza tunelování přes HTTP

2.1 HTTP

Hypertext Transfer Protocol neboli HTTP se používá od roku 1990, detailně je popsaný v [31]. Dnes se používá pro přenášení jakýchkoliv dat a díky tomu se stal jedním z nejvíce používaných protokolů na internetu. Funguje na principu dotazu a odpovědi, kdy klient odesílá dotaz/žádost a od serveru mu přichází odpověď. Tento protokol většinou komunikuje na portu 80.

Existuje několik metod pro dotazování [31], nejznámějšími jsou:

- GET
  Požadavek o určitý obsah, ve kterém se můžou posílat i data potřebná pro identifikaci konkrétního obsahu. Data se posílají zakódované v URL.

- HEAD
  Stejný jako GET, ale místo celých dat odesílá jen hlavičku s informacemi o objektu (velikost objektu, poslední změna, ...).

- POST
  Odesílá uživatelská data na server. Používá se pro odesílání většího množství dat nebo pokud není vhodné data kódovat do URL.

- PUT, DELETE
  Používá ve webových službách REST pro manipulaci s daty.

- OPTIONS
  Dotaz na server, jaké podporuje metody.

- CONNECT
  Používá se pro trvalejší spojení se serverem skrze proxy.

2.2 Tunelování přes HTTP

HTTP tunelování se používá tam, kde nejsou pro komunikaci s internetem povolené určité protokoly, které chceme využívat nebo jsou blokované určité webové stránky, na které chceme přistupovat. To znamená, že se nacházíme v sítí s omezeným přístupem pomocí HTTP proxy serveru nebo firewallu. Abychom mohli využít tunel pro průchod veškeré komunikace skrz tyto zábrany, musí být na rozhraní sítě povolena HTTP komunikace.

Tunel se skládá ze dvou bodů a to serveru a klienta. Server poskytuje připojení přes HTTP, které je dostupné z internetu. Klient se připojuje k serveru ze sítě s omezeným přístupem. Uzly mezi sebou vytváří spojení, do kterého kódují veškerou potřebnou komunikaci. Takovéto spojení nazýváme HTTP tunelem (viz obrázek 2.1).

Existují tři varianty tunelování přes HTTP. První, nejjednodušší je přenášení binárních dat na portu 80. Data se posílají ve své binární podobě uložené v paketu se zdrojovým či cílovým portem 80. Tento způsob můžeme použít, pokud nám v cestě do internetu stojí pouze jednoduchý firewall blokující komunikaci na ostatních portech. Jedná se o stejně výkonný tunelovací nástroj, jako poskytují jiné legální tunelovací protokoly, ale zároveň o nejméně použitelnou variantu pokud je na síti proxy server, přes který tento typ komunikace neprojde.

Druhou variantou je použití metody CONNECT. Klient nejprve pošle žádost proxy serveru, zda může vytvořit HTTP CONNECT spojení. Jakmile proxy potvrdí toto spojení, klient a vzdálený server mezi sebou vytvoří TCP stream. Tento způsob se používá například pro přístup k webovým stránkám používajícím SSL (princip popsán v [21]). Můžeme tak velice jednoduše vytvořit tunel, ať jsme za proxy serverem či firewallem. Jedinou podmínkou je povolení metody CONNECT na HTTP proxy serveru.

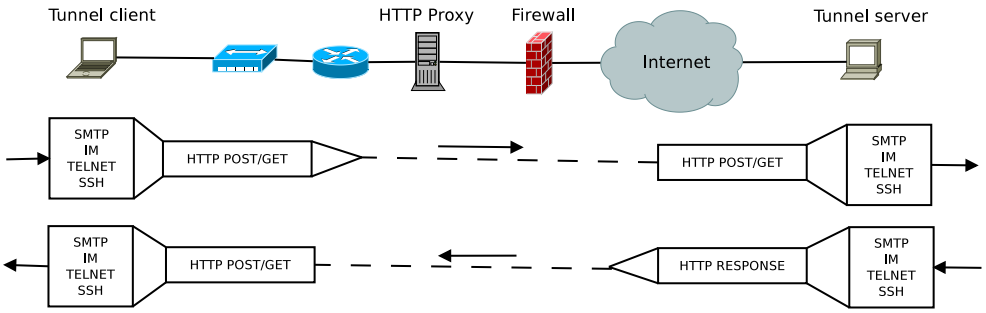
Třetí variantou je využití klasických HTTP metod. Tato varianta je nejpomalejší, ale zato jí nedetekuje proxy server ani firewall. Žádosti se posílají v HTTP GET, POST, PUT nebo DELETE metodě. Aplikace může náhodně střídat jednotlivé metody, tak aby se co nejlépe skryla. Nazpět přicházejí data zakódované jako HTTP odpovědi.

2.3 Tunelovací nástroje

Stejně jako pro DNS v sekci 1.3, i pro HTTP existuje mnoho tunelovacích nástrojů. Příklady několika programů:

- Universal HTTP Tunnel client-server

  Nástroj [13] používá pro odesílání dat metodu GET. Je spustitelný pouze na počítači s Windows server 2003.



Obrázek 2.1: Notebook má kvůli firewallu a HTTP proxy blokovaný přístup do internetu. Povolený je pouze protokol HTTP. Klient vytváří HTTP tunel, data kóduje do HTTP žádostí a odesílá je na vzdálený tunelovací server. Tento server dekóduje přijatá data a odešle odpověď zakódovanou jako HTTP response.

- Super Network Tunnel

  Nástroj [10], který umožňuje tunelovat v několika módech. Nejvýkonnějším je posílání dat přes port 80 (bez HTTP hlavičky), dalším je pomocí CONNECT metody a poté přes HTTP POST dotazy. Tento nástroj je určen pro operační systém Windows, je placený, ale umožňuje patnáctidenní vyzkoušení zdarma.

- Httptunnel

  Nástroj [6] posílající data přes GET nebo POST metodu. Jedná se o nástroj pod licencí GNU, spustitelný na Linuxu.

2.4 Známé metody detekce

Tunely přes protokol HTTP je velice těžké detekovat. Hlavním problémem detekce je četnost využití HTTP a možnost přes něj posílat data téměř libovolného typu. Dat je mnoho a proto je výpočetně náročné, detekovat tunely v reálném čase.

Detekci rozdělíme na payload analysis a traffic analysis.

- Payload analysis

  - Specifické znaky

    [16] Pokud přenášená data v HTTP nejsou šifrovaná, můžeme se pokoušet vyhledávat znaky jiných protokolů, jako například SMTP hlavičky. Pokud jsou data šifrovaná, detekce tímto způsobem je buď velice komplikovaná nebo nelze provést.

- Traffic analysis
  - Statistické metody

    Informacemi jako jsou frekvence komunikace, velikost paketů, rozestupy packetů a dalších, lze zkoumat chování sítě. V případě, že tyto hodnoty vymezují ze stanovených intervalů, může komunikace obsahovat tunel. Podrobněji je tato metoda popsána v [16] nebo v [24].

2.5 Analýza provozu na síti

Pro tunelování jsem použil nástroj Super Network Tunnel [10], který umožňuje použít metodu POST nebo CONNECT volitelně. Na jednom počítači jsem spustil serverovou část a na druhém klientskou. Oba počítače byly ve stejné lokální síti, což mohlo ovlivnit rychlost komunikace.

2.5.1 Metoda CONNECT

Nejprve jsem zachytával komunikaci tunelu vytvořenou metodou HTTP CONNECT. Plno webových aplikací komunikuje přímo přes port 80, aniž by se jednalo o HTTP. Komunikace je tak více a detekovat tunel je o něco obtížnější.

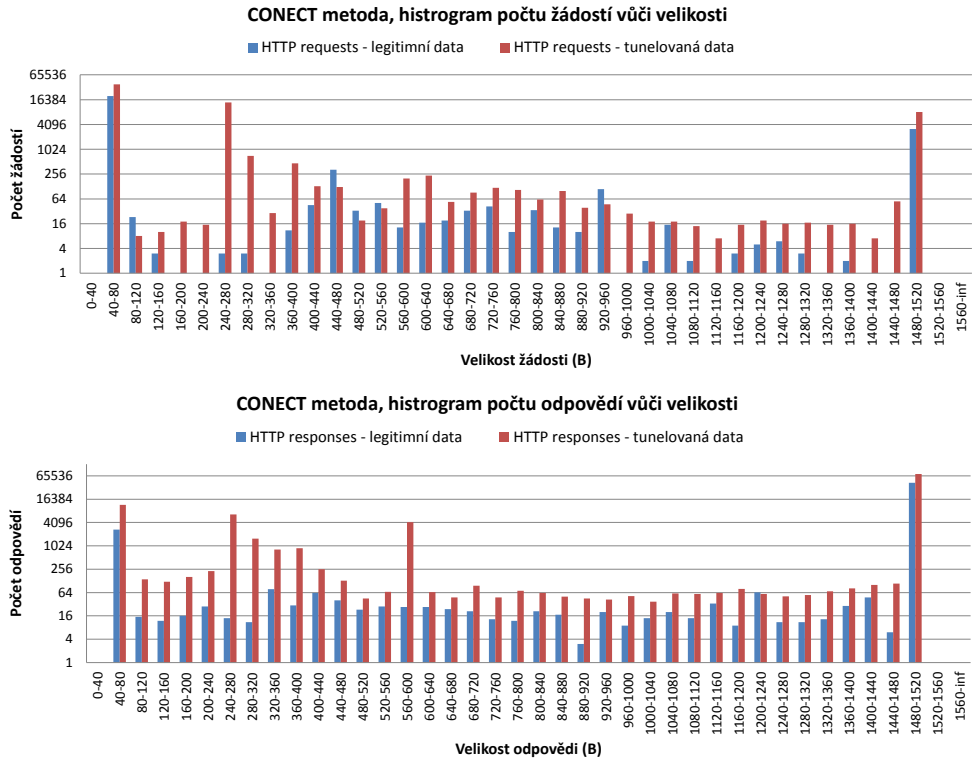
Na histogramech 2.2 je vidět počet žádostí a odpovědí v závislosti na velikosti paketu. Červeně je znázorněna tunelovaná komunikace a modře legitimní komunikace.

V grafu žádostí je vidět, že tunelované komunikace je mnohem více než legitimní. To je způsobené především tím, že komunikace všech aplikací byla vynucena přes port 80. Legitimní komunikace využívá především velikosti 40–80 B (dotazy a potvrzení komunikace) a 1480–1520 B (data). Tunelovaná komunikace navíc využívá i jiné velikosti, zde především 240–320 B.

V grafu odpovědí je opět vidět, že tunelované komunikace je mnohem více než legitimní. Legitimní majoritně využívá maximální velikosti paketu (MTU), ve které odesílá webové stránky ale i data, videa, obrázky a další. Oproti tomu tunelovaná komunikace navíc využívá i nižší velikosti.

Musíme brát v potaz, že objem legitimní komunikace je rozprostřen od klienta k mnoha serverům, naproti tomu tunelovaná komunikace má hlavní podíl objemu přenesených dat s jedním serverem. Jak je vidět na histogramu 2.3, objem přenesených dat jedné komunikace (unikátní klient, unikátní server) je znatelně vyšší u tunelované komunikace. Záleží samozřejmě na výběru dané komunikace, každá může být zcela odlišná. Zde byla za legitimní vybrána komunikace odesílající data na server, což můžeme poznat i z histogramu. Odesílají se „velké“ pakety a zpět přicházejí „malé“ potvrzení o přijetí.

Na histogramu 2.4 je vidět počet paketů v závislosti na rozestupu mezi pakety u několika adres legitimní a tunelované komunikace. Rozestupy závisí na



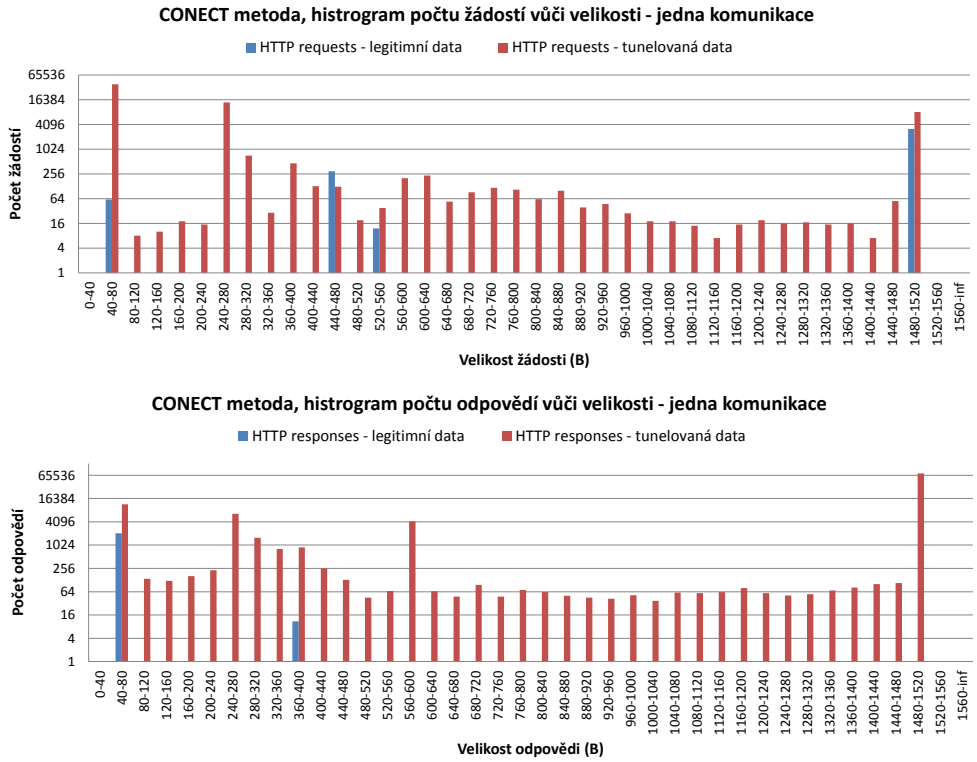
Obrázek 2.2: Histogramy počtu žádostí a odpovědí v závislosti na velikosti paketu pro veškerou komunikaci na portu 80, metoda CONNECT.

konkrétních aplikací. Pokud se jedná o aplikaci udržující dlouhodobé spojení, bude paketů v určitém intervalu více než v ostatních intervalech. Takovýto typ komunikace může být tunel, IM využívající HTTP a další. V našem případě jsou rozdělení, jak pro tunelovanou tak pro legitimní komunikaci, velice podobné. Jedinou použitelnou hodnotou jsou rozestupy delší než $1s$, u kterých můžeme předpokládat, že se nejedná o tunel.

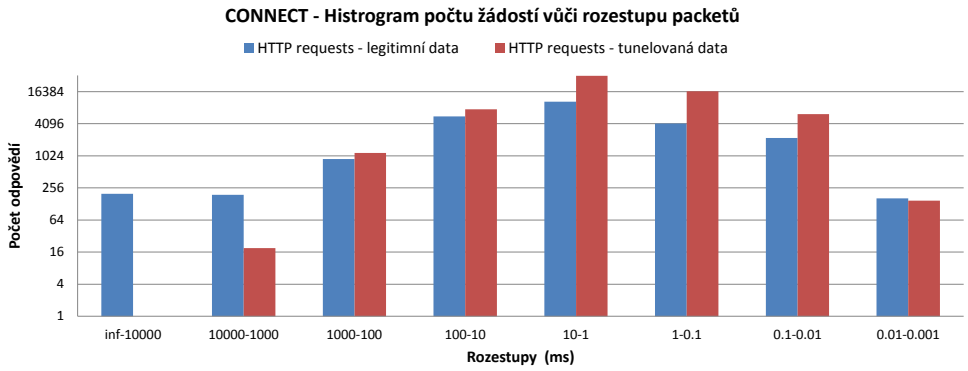
2.5.2 Metoda PUT

Pokud proxy server zakazuje použití metody CONNECT, můžeme použít metodu PUT, kterou proxy server povoluje téměř vždy. Následkem je částečné snížení propustnosti, ale tunelovací nástroj tak dokáže překonat i dobře zabezpečené proxy servery. V tomto případě můžeme filtrovat pouze pakety obsahující HTTP hlavičku, kterých je méně než paketů s portem 80.

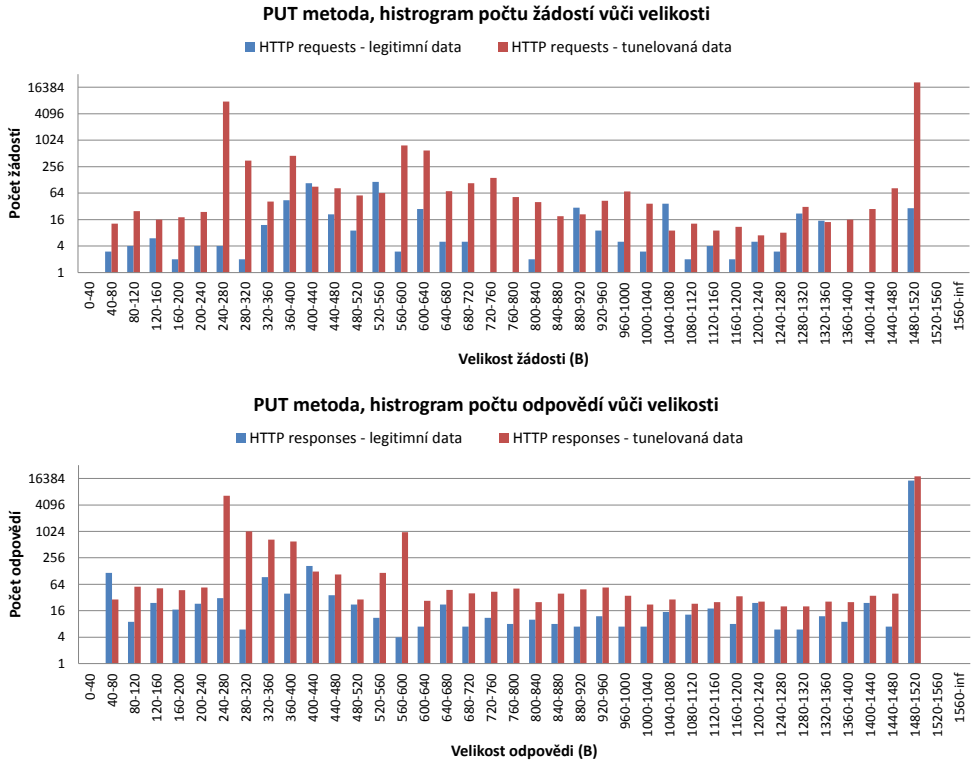
Na obrázku 2.5 je histogram počtu žádostí a odpovědí v závislosti na velikosti paketu. Žádosti obsahují více paketů tunelované než legitimní komunikace, především při větší velikosti paketu. Odpovědi jsou již o něco vyrovna-



Obrázek 2.3: Histogramy počtu žádostí a odpovědí v závislosti na velikosti paketu na jednu IP adresu, metoda CONNECT.



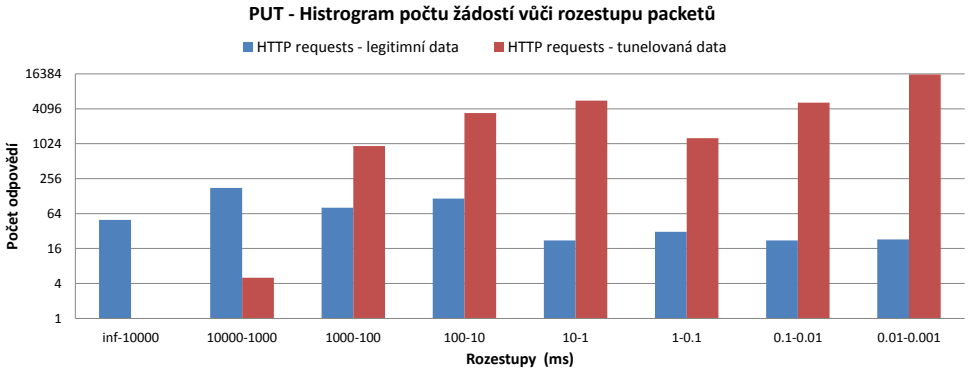
Obrázek 2.4: Histogramy počtu žádostí v závislosti na rozestupu mezi pakety, metoda CONNECT.



Obrázek 2.5: Histogramy počtu žádostí a odpovědí v závislosti na velikosti paketu, metoda PUT.

nější, u tunelované i legitimní komunikace se využívá maxima velikosti paketu. Tunelovaná komunikace je navíc značně využívána i v ostatních velikostech. V porovnání žádostí a odpovědí tunelované komunikace si lze všimnout, že v obou je stejně velký počet paketů ve velikosti 240–280 B.

Na histogramu 2.6 je vidět počet žádostí v závislosti na rozestupu mezi pakety. Je vidět významný rozdíl mezi tunelovanou a legitimní komunikací. Legitimní komunikace využívá především delší rozestupy, naopak tunelovaná využívá kratší rozestupy. Tento atribut lze použít pro účely detekce.



Obrázek 2.6: Histogramy počtu žádostí v závislosti na rozestupu mezi pakety, metoda PUT.

Kapitola 3

Návrh modulu pro detekci tunelování přes DNS

V této kapitole jsou popsány vhodné metody pro detekci anomálií (především tunelů) protokolu DNS na páteřní síti a jejich spojení do funkčního celku. Musel jsem brát v potaz, že modul bude dostávat data od sondy, která se nachází na páteřní síti. Z toho plyne potřeba v reálném čase zpracovávat obrovské množství dat, proto by měl být modul navržen tak, aby dokázal analyzovat data v nejkratším možném čase.

Analýza komunikace na páteřní síti má oproti LAN síti mnohé rozdíly. Budeme zaznamenávat mnohem více dat. Nemůžeme se spoléhat, že pokud zachytíme žádost o překlad, zachytíme i odpověď, která může jít jinou cestou než přišla žádost. Může se stát, že přijmeme pouze jednu podezřelou žádost a jiná komunikace k nám nedorazí. Tyto rozdíly jsem musel brát v potaz a navrhnout detekční modul tak, aby byl schopen fungovat na páteřní síti.

Pro potřeby detekce nestačí pouze nahlásit, že se v komunikaci se nachází tunel. Je potřeba nahlásit i z jaké IP adresy pochází. Proto všechny metody detekce budou prováděny vždy pro určitou IP adresu. Pokud půjde o dotaz, bude to IP adresa zdroje a pokud půjde o odpověď, bude to IP adresa cíle. IP adresa se nahlašuje z důvodu identifikace původu anomálie, ale přesto se nemusí jednat o tvůrce anomálie, protože IP adresa může být podvržena (IP spoofing [26]).

3.1 Traffic analysis

3.1.1 Střední hodnota a rozptyl

Již z analýzy sítě v sekci 1.5 je vidět, že vhodným parametrem pro měření bude střední hodnota a rozptyl velikosti paketů. Výběrovou střední hodnotu velikosti paketů (průměr) spočítáme podle vzorce 3.1, kde x_i je velikost paketu i a n je celkový počet paketů. Výběrový rozptyl spočítáme podle vzorce 3.2.

$$\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.1}$$

$$s_n{}^2 = \frac{1}{(n-1)}\left(\sum_{i=1}^{n}(x_i^2) - \overline{X}_n{}^2 * n\right) \tag{3.2}$$

Abychom znali střední hodnotu a rozptyl legitimní komunikace, spočítáme tyto hodnoty pro analyzovaná legitimní data ze sekce 1.5. Pro žádosti po zaokrouhlením vyjdou hodnoty $\overline{X}_n = 76$ a $s_n{}^2 = 29$. Pro odpovědi $\overline{X}_n = 229$ a $s_n{}^2 = 9179$.

Nyní známe průměrné hodnoty legitimní komunikace. Stanovíme si tedy meze (minima a maxima střední hodnoty a rozptylu), které budou značit podezřelost komunikace. Pokud komunikace na určité IP adrese nebude v daném intervalu, budeme jí označovat jako podezřelou. Vypočítané hodnoty jsou závislé na konkrétní síti, proto jsou meze v modulu nastavitelné. Pokud interval zvětšíme, sníží se procento podezřelých IP adres. Naopak pokud interval zmenšíme, procento podezřelých IP adres se zvýší.

Za tunel budeme považovat komunikaci, jejíž střední hodnota nebo rozptyl přesáhne legitimní maximum. Nahlašovat budeme i pokud střední hodnota nebo rozptyl přesáhne hranici minima. Zřejmě v tomto případě nepůjde o tunel, ale může se zde jednat o jinou anomálii.

3.2 Payload analysis

3.2.1 Počet přístupů k doméně

Měříme jak často se přistupuje k jednotlivým doménám. Pro tunel je typické, že každá doména se překládá pouze jednou. Naopak legitimní domény jsou překládány vícekrát. Naměříme-li tedy velký počet domén s unikátními přístupy budeme IP adresu s těmito žádostmi považovat za podezřelou.

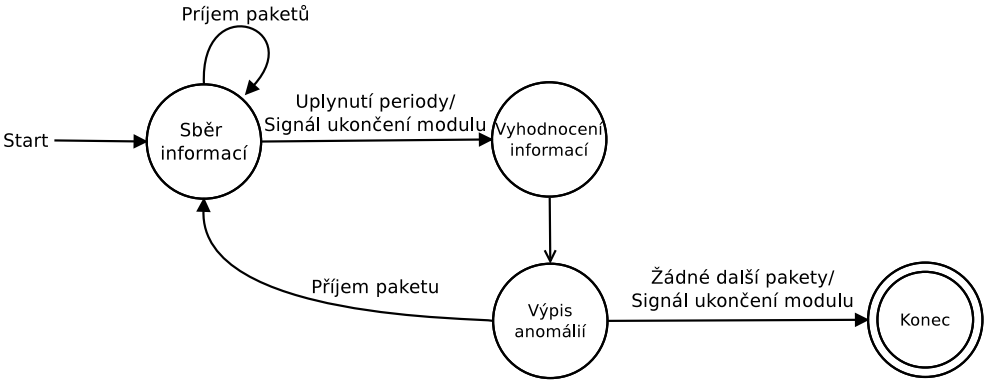
Protože v modulu počítáme informace o počtu přístupů k určité doméně, budeme nahlašovat i IP adresy, které požadují překlad určitého doménového jména podezřele mnohokrát. Mohlo by se jednat například o DOS útok.

3.2.2 Počet různých poddomén n-tého řádu

V tunelované komunikaci každý dotaz obsahuje doménu tunelovacího serveru, ta je druhého nebo vyššího řádu. Pokud počet potomků této domény překročí určitou mez, budeme považovat tuto komunikaci za podezřelou.

3.2.3 Počet unikátních znaků v řetězcích

Tam kde se vyskytuje textový řetězec, můžeme kontrolovat tento atribut. Jde především o text žádosti, TXT, CNAME, MX a NS záznam. Podle [17] bychom



Obrázek 3.1: Automat pro hledání anomálií v paketech.

měli nahlašovat každou doménu, která obsahuje více než 27 unikátních znaků. Záleží však především na typu kódování. Tato metoda nerozpozná tunely používající například kódování Hex nebo NetBIOS.

3.2.4 Podíl číslic v řetězcích

Stanovíme maximální podíl číslic v řetězcích. Musíme brát v potaz, že pokud by byla doména příliš krátká i jedna číslice by znamenala porušení této vlastnosti. Budeme tedy vybírat všechny domény, které obsahují více než stanovenou hranici číslic a zároveň procentuálně (podíl číslic vůči délce domény) přesahují určitou mez.

3.3 Propojení do celku

Popsané metody jsem spojil do celku a vytvořil tak nástroj pro detekci anomálií. Nástroj pracuje jako automat, který má čtyři stavy. Přechody automatu jsou znázorněny na diagramu 3.1.

Stavy automatu:

1. Sběr informací o IP adresách
   V tomto stavu dochází k přijímání paketů, aktualizování a zaznamenávání důležitých informací (více v sekci 3.3.1). Další funkcí tohoto stavu je základní payload analýza DNS žádostí a odpovědí (více v sekci 3.3.2). V tomto stavu se automat nachází pouze předem určený časový interval nebo dokud nepřijme signál o ukončení, poté přechází do stavu 2.

2. Vyhodnocení
   V tomto stavu se procházejí jednotlivé IP adresy, které byly zaznamenány v 1. stavu. Na každou z nich se spouštějí testy anomálií (více v sekci 3.3.3).

3. Výpis anomálií
   V tomto stavu se vypíší nalezené anomálie. Pokud automat přijal signál o ukončení nebo nejsou-li na vstupu další pakety, automat přechází do stavu 4, jinak přechází do stavu 1.

4. Konec
   Ukončení modulu.

3.3.1 Ukládané informace o IP adresách

Každá IP adresa je zkoumána zvlášť. Jakmile přijde paket, vyhledají se informace o příslušné IP adrese a aktualizují se. Pokud pro IP adresu neexistuje žádný záznam, vytvoří se nový.

Každá adresa uchovává tyto základní informace:

- počet přijatých žádostí,
- počet prázdných přijatých žádostí (žádosti bez řetězce),
- počet přijatých odpovědí,
- Součet velikosti žádostí (pro výpočet střední hodnoty velikosti žádostí),
- Součet velikosti odpovědí (pro výpočet střední hodnoty velikosti odpovědí),
- Součet druhých mocnin velikosti žádostí (pro výpočet rozptylu velikosti žádostí),
- Součet druhých mocnin velikosti odpovědí (pro výpočet rozptylu velikosti odpovědí),
- data histogramu počtu přijatých žádostí v závislosti na velikosti paketu,
- data histogramu počtu přijatých odpovědí v závislosti na velikosti paketu,
- data histogramu počtu unikátních písmen v žádosti v závislosti na velikosti paketu.

Každá IP adresa používá čtyři typy detekce anomálií (dále jen typy detekce). Typy detekce jsou rozdělené podle směru komunikace a druhu anomálie. Každý typ detekce uchovává rozšiřující informace o komunikaci, které dopomáhají přesnější detekci. Tyto informace jsou z důvodu operační a paměťové náročnosti uchovávány až ve chvíli podezření na danou anomálii. V následujícím seznamu jsou vypsány typy detekce anomálií zkoumané u IP adres.

1. Tunnel request - Detekce tunelů v žádostech.

   Uchovávané informace pro pokročilou detekci:

- podezřelé domény žádostí z tunelu,
- počet vyhledávání unikátních domén (domény vyhledávané pouze jednou),
- domény seřazené podle počtu vyhledávání vzestupně,
- množství potomků jednotlivých domén.

2. Tunnel response - Detekce tunelů v odpovědích.

   Uchovávané informace pro pokročilou detekci:

   - podezřelé CNAME, TXT, MX, NS záznamy,
   - počty stejných CNAME, TXT, NS, MX záznamů,
   - nejméně často opakované CNAME, TXT, NS, MX záznamy.

3. Other anomaly request - Detekce ostatních anomálií v žádostech.

   Uchovávané informace pro pokročilou detekci:

   - podezřelé domény žádostí z anomálie,
   - počet vyhledávání stejných domén,
   - domény seřazené podle počtu vyhledávání sestupně.

4. Other anomaly response - Detekce ostatních anomálií v odpovědích.

   Uchovávané informace pro pokročilou detekci:

   - podezřelé domény odpovědí z anomálie,
   - počet odpovědí bez domény,
   - domény seřazené podle počtu vyhledávání sestupně.

Každý typ detekce se může dostat do čtyř stavů, které jsou vypsané v následujícím seznamu. Stavový diagram 3.2 znázorňuje možné přechody mezi jednotlivými stavy.

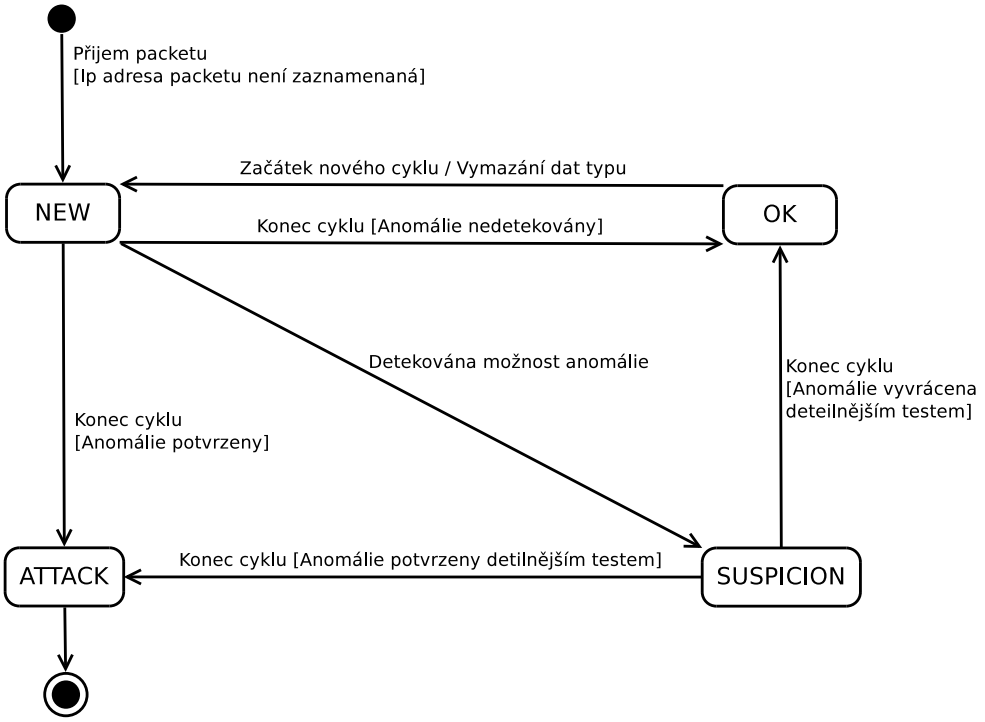
- NEW

  Počáteční stav, který je nastaven před příjímáním paketů.

- SUSPICION

  Stav podezření typu detekce na anomálii stejného druhu. Jakmile se typ detekce dostane do tohoto stavu, začne během sběru informací ukládat potřebná data určená k detailnější analýze.

  Pokud typ detekce skončí ve stavu *SUSPICION*, ukládání informací pokračuje i v dalším cyklu přijímání paketů. V tomto stavu může typ detekce zůstat maximálně n-krát, poté jsou rozšiřující informace vymazány a typ detekce nastaven na *NEW*.



Obrázek 3.2: Stavový diagram popisující přechody mezi stavy typů detekce

- ATTACK

  Stav, který znamená, že se v komunikaci IP adresy projevila anomálie daného typu. Tento stav adrese a typu detekce zůstává až do ukončení programu. Zároveň se z důvodu úspory paměti přestanou ukládat všechny další informace pro tento typ detekce na dané IP adrese.

- OK

  Do tohoto stavu se dostává typ detekce, u kterého testy neprokázaly žádnou anomálii. Data typu detekce s tímto stavem jsou na konci vyhodnocení smazány a typ detekce je změněn na stav NEW.

3.3.2 Payload analýza během přijímání paketů

Během přijímání paketů probíhá základní payload analýza. V té se vyskytují testy *počet unikátních písmen v řetězci* 3.2.3 a *počet číslic v řetězci* 3.2.4. Tyto testy se u žádostí spouští nad doménovým jménem a u odpovědí nad doménovým jménem, TXT, CNAME, NS a MX záznamem. Pokud některý z testů nalezne anomálii, daný typ detekce se dostane do stavu *SUSPICION*, ve kterém začne ukládat doplňující informace z podezřelých paketů.

3.3.3 Testy anomálií

Testy probíhají ve fázi vyhodnocení pro každou IP adresu a typ detekce zvlášť.

Prvním testovacím typem detekce je *Tunnel request*. Pokud se typ detekce nachází ve stavu *NEW* probíhá test střední hodnoty a rozptylu 3.1.1. Pokud je střední hodnota větší než stanovené maximum a rozptyl je mimo stanovený rozsah, poté typ detekce přechází do stavu *SUSPICION*, jinak do stavu *OK*. Dalšími testy jsou počet přístupů k doméně 3.2.1 a počet různých poddomén n-tého řádu 3.2.2. Tyto testy jsou prováděny pouze pokud se typ detekce nachází ve stavu *SUSPICION*. Typ detekce se dostává do stavu *ATTACK*, pokud je většina doménových jmen vyhledávána pouze jednou a pokud má většina dotazů stejnou doménu n-tého řádu (postfix zůstává stejný, mění se pouze prefix řetězce). V ostatních případech typ detekce zůstává ve stejném stavu.

Druhým testovacím typem detekce je *Other anomaly request*. Pokud se typ detekce nachází ve stavu *NEW* probíhá test střední hodnoty a rozptylu 3.1.1. Pokud je střední hodnota menší než stanovené minimum nebo rozptyl je mimo povolený rozsah, poté typ detekce přechází do stavu *SUSPICION*, jinak do stavu *OK*. Dalším testem je počet přístupů k doméně, který se provádí pouze pokud je typ detekce ve stavu *SUSPICION*. Typ detekce se dostává do stavu *ATTACK*, pokud většina žádostí obsahuje stejné doménové jméno, jinak zůstává ve stejném stavu.

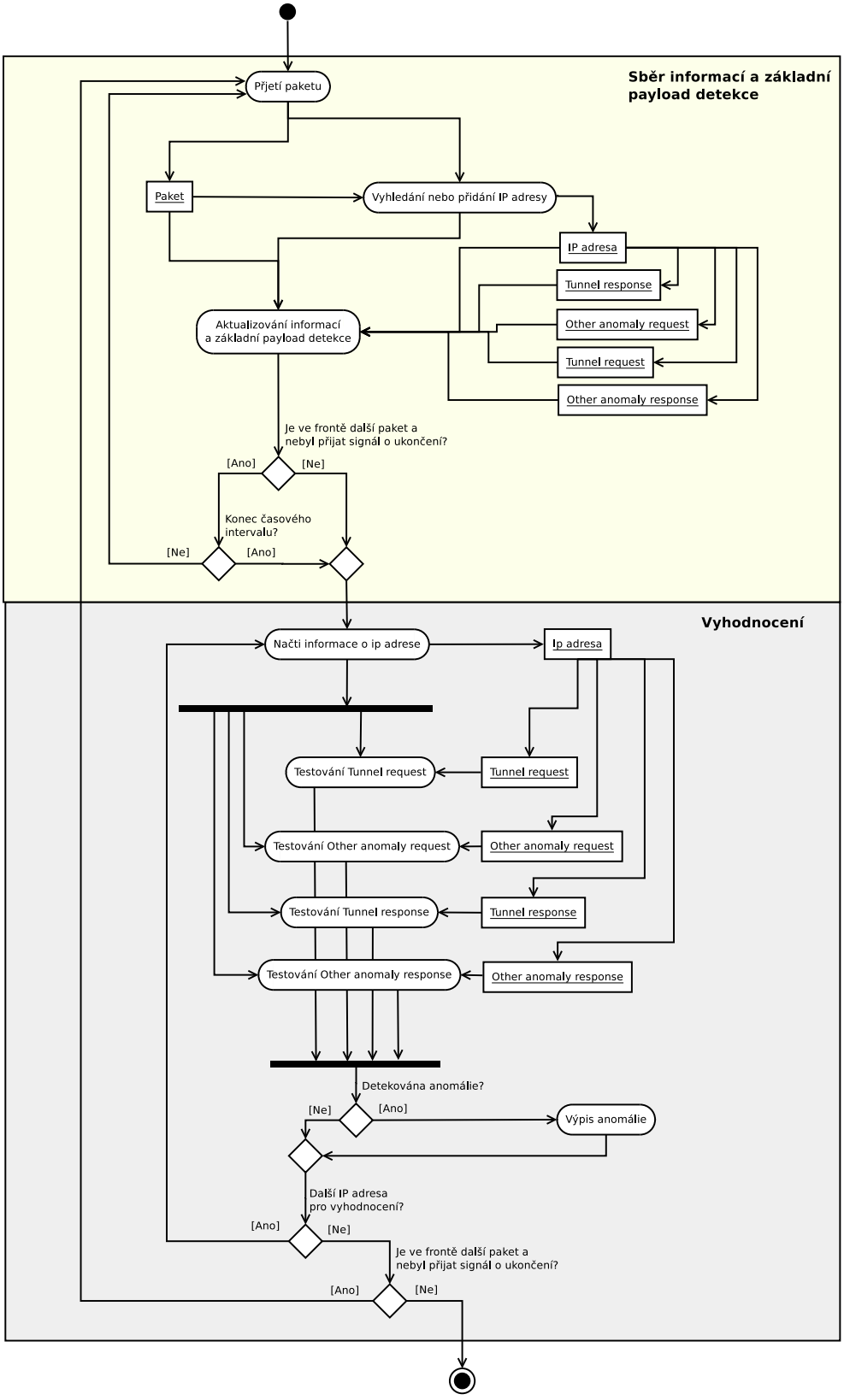
Třetím typem detekce je *Tunnel response*. Pokud se typ detekce nachází ve stavu *NEW* probíhá test střední hodnoty a rozptylu 3.1.1. Pokud je střední hodnota větší než stanovené maximum nebo rozptyl je mimo povolený rozsah, poté typ detekce přechází do stavu *SUSPICION*, jinak do stavu *OK*. Test počtu přístupů k doméně se provádí pouze pokud je typ detekce ve stavu *SUSPICION*. V tomto případě testujeme doménové jméno v odpovědi a záznamy TXT, CNAME, MX a NS. Pokud většina komunikace obsahuje některý z těchto záznamů a záznamy se neopakují, typ detekce přechází do stavu *ATTACK*, jinak zůstává ve stejném stavu.

Čtvrtým a zároveň posledním typem detekce je *Other anomaly response*. Pokud se typ detekce nachází ve stavu *NEW* probíhá test střední hodnoty a rozptylu 3.1.1. Pokud je střední hodnota nebo rozptyl mimo povolený rozsah, poté typ detekce přechází do stavu *SUSPICION*, jinak do stavu *OK*. Dalším testem je počet přístupů k doméně, který se provádí pouze pokud je typ detekce ve stavu *SUSPICION*. Typ detekce se dostává do stavu *ATTACK*, pokud většina odpovědí obsahuje stejné doménové jméno nebo neobsahuje žádné doménové jméno, jinak zůstává ve stejném stavu.

3.4 Shrnutí

Modul je automat pracující v cyklu a pro každou IP adresu rozlišuje čtyři typy detekce, které mohou nabývat čtyř stavů. První fází modulu je sběr

informací a základní payload analýza. Během této fáze přicházejí pakety, které rozšiřují informace o toku jednotlivých IP adres. Již během přijímání paketů mohou být některé IP adresy označeny jako podezřelé z výskytu anomálie. Po ukončení první fáze modul přechází do fáze vyhodnocení. V této fázi se u každé IP adresy a každého typu detekce vyhodnocuje účast anomálie. Pokud se detekuje nějaká anomálie, vypíše se. Pokud po vyhodnocení všech IP adres modul přijme ukončovací signál, ukončí se, jinak pokračuje první fází. Tento princip je znázorněn na diagramu aktivit 3.3.



Obrázek 3.3: Diagram aktivit znázorňující činnost modulu.

Kapitola 4

Implementace modulu pro detekci tunelování přes DNS

Tato kapitola se věnuje implementaci detekčního modulu DNS anomálií, především tunelů. V první části je popsán výběr nástrojů a prostředí pro realizaci. Druhá část se zabývá zpracováním dat a jejich převedením do modulu. Třetí částí je diskuze nad výběrem vhodných datových struktur pro ukládání a vyhledávání IP adres a domén. Po výběru datových struktur, přecházíme k jejich implementaci. Zde je popsán základní princip struktur a hlavní funkce pro jejich ovládání a komunikaci s modulem. Čtvrtou částí je implementace samotného modulu, kde je popsána realizace následujících funkcí: hlavní cyklus modulu, sběr informací z paketů, základní payload analýza probíhající během sběru informací, přechod k vyhodnocení, detekční testy, výstup modulu, parametrizování modulu.

4.1 Nástroje a prostředí

Detekční modul jsem napsal v jazyce C. Především kvůli tomu, že modul v reálném čase analyzuje obrovské množství dat. Dalším důvodem je vytvoření univerzální datové struktury pro ukládání informací o IP adresách, kterou budou používat i ostatní moduly. Systém Nemea mimo jiné podporuje tento jazyk.

4.1.1 Předzpracování dat

Pro práci s daty, která přijdou do modulu, se používá knihovna UniRec [9] vyvinutá v Cesnetu. Modul je vytvořený tak, aby přijímal tento formát a zároveň, z důvodu testování, i data z externího souboru. Naměřená data určená pro testování jsou ve formátu PCAP. Tato data nejprve převádím pomocí nástroje Tshark [12] do textového souboru, který v modulu parsuji a data dále kopíruji do struktur, se kterými modul pracuje.

4.1.2 Grafický výstup

Jedním z výstupu modulů jsou histogram počtu žádostí v závislosti na velikosti paketu, histogram počtu odpovědí v závislosti na velikosti paketu a histogram počtu unikátních znaků v žádosti v závislosti na velikosti. Tyto informace je možné z textové podoby převést i do podoby grafické. Pro převedení dat histogramů do grafické podoby je použit program R [25].

Grafický výstup slouží pouze k informativním účelům pro uživatele, pro strojovou detekci anomálií není potřeba.

4.2 Zpracování PCAP souborů

Pro analýzu DNS provozu je třeba exportovat tyto informace: čas přijetí paketu na rozhraní, zdrojová IP adresa, cílová IP adresa, velikost paketu.

Pro DNS žádosti: hledané doménové jméno.

Pro DNS odpovědi: řetězec dotazu, TXT záznam, CNAME záznam, MX záznam, NS záznam.

Data pro analýzu jsou rozdělená do mnoha menších souborů, kde v názvu je pořadové číslo souboru. Pro export dat z PCAP formátu do textového souboru pomocí programu Tshark jsem vytvořil skript *data_parser.sh*, kterému zadáme parametry: vstupní soubor, index prvního souboru, index posledního souboru, výstupní soubor. Máme tak možnost soubory zpracovat dávkově. Konkrétní volání programu Tshark ve skriptu vypadá takto:

```
tshark -r file.pcap -T fields -E separator=\; \
-e frame.time_epoch  \
-e ip.addr \
-e ipv6.addr \
-e dns.flags.response \
-e frame.len \
-e dns.qry.name \
-e dns.txt \
-e dns.resp.primaryname \
-e dns.mx.mail_exchange  \
-e dns.resp.ns
```

Ve výsledném souboru máme uloženy informace: Čas ve formátu epoch (čas v sekundách od 1.1.1970), zdrojová IP adresa verze 4, cílová IP adresa verze 4, zdrojová IP adresa veze 6, cílová IP adresa verze 6, příznak zda se jedná o odpověď či o dotaz, velikost paketu v bajtech, text DNS žádosti, TXT záznam, CNAME záznam, MX záznam, NS záznam. Jednotlivé atributy jsou odděleny středníkem.

Příklad DNS žádosti s anonymizovanými IP adresami:

```
1388962692.523174000;4.17.161.214,115.2.119.16;;0;84;webmail.pec.
    kunika.de;;;;
```

Příklad DNS odpovědi s anonymizovanými IP adresami:

```
1388959816.752051000;2.1.14.19,115.31.181.15;;1;633;maimemotor.de
    ;;;;ns1.brok.de,ns2.brok.de
```

Výsledný soubor lze jednoduše zpracovat parserem. V detekčním modulu je parser implementovaný v souboru *parser_pcap_dns.c*. Nejprve parser musíme inicializovat funkcí *parser_initialize()*, která jako parametr požaduje jméno souboru, který chceme číst. Funkce načítá soubor pro čtení a vrací ukazatel na něj. Dále voláme v cyklu funkci *read_packet()*, jako parametry předáváme ukazatel souboru pro čtení, a adresu na strukturu pro uložení informací z paketu. Protože v tomto případě stačí pouze jedna instance této struktury, vytvoříme ji jen jednou v *main()* funkci modulu a přepisujeme ji v každé iteraci. Po načtení paketu, funkce vrací -1, pokud je soubor u konce nebo 0, pokud načtení paketu proběhlo v pořádku.

Struktura, kterou parser naplňuje obsahuje všechny položky, které exportujeme programem Tshark. Přičemž pro IP verze 6 je zde pole o dvou prvcích typu *uint64_t*, které odpovídá velikosti 16 B. Pro IP verze 4 stačí *uint32_t* o velikosti 4 B. Takto uložená data již můžeme lehce programově zpracovávat.

4.3 Výběr datové struktury pro ukládání a vyhledávání IP adres

Pro účely detekce bylo nutné provést výběr vhodné datové struktury, která splní následující požadavky pro práci s IP adresami:

- efektivní vkládání,
- efektivní vyhledávání,
- efektivní mazání,
- práce s prvky jako se spojovým seznamem,
- nízká paměťová náročnost,
- možnost uložit předem nespecifikované množství prvků.

Do datové struktury budou ukládány informace o komunikaci. Jako identifikátor komunikace lze jednoznačně použít IP adresu, která posílá žádost a u odpovědi IP adresu, která přijímá odpověď. Zajímají nás tedy klientské IP adresy, nikoliv IP adresy serverů.

Pro ukládání IP adres existuje mnoho různých způsobů, v této sekci uvádím pouze několik možností, které přicházejí v úvahu, jejich výhody a nevýhody.

4.3.1 Rozptylovací tabulka

Jednou z možností jak ukládat a vyhledávat IP adresy je tabulka. Můžeme vytvořit asociativní pole o velikosti adresního prostoru všech IP adres a adresy do pole mapovat přímo. Nevýhodou tohoto způsobu je paměťová náročnost. Pouze pro IP verze 4 bychom museli vytvořit pole o velikosti 2^{32} prvků .

Výhodnější je použit rozptylovací funkci [29]. Ta z klíče (IP adresy) vytvoří heš, který slouží jako index do pole prvků. Paměťová složitost je poté *O(m)*, kde m je velikost tabulky. Operační složitost poté závisí na rozptylovací funkci a mechanismu řešení kolizí.

Nevýhodou je, že dopředu nevíme kolik IP adres bude třeba ukládat. Pokud zvolíme m příliš nízké, bude docházet častěji ke kolizím a zvýší se operační složitost. Pokud zvolíme m naopak příliš vysoké, budeme alokovat zbytečně mnoho paměti.

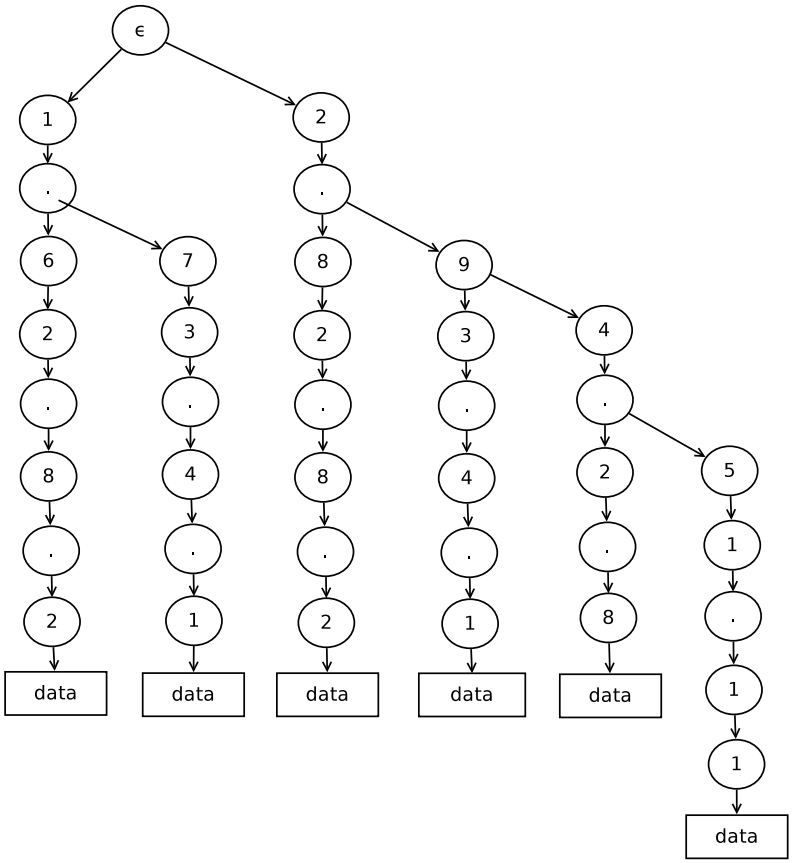
4.3.2 Vyhledávací stromy bez vyvažování

Jedná se o hierarchické stromové struktury. Procházením stromu shora dolů se v každém kroku zužuje prostor vyhledávání, pokračujeme dokud nenalezneme hledaný klíč nebo až do krajního uzlu, ve kterém se nevyskytuje požadovaný potomek.

Trie

Pokud bychom chtěli uchovávat IP adresu jako textový řetězec (v notaci oddělené tečkami), zřejmě by se nám vyplatilo použít jako datovou strukturu prefixový strom, známý také jako Trie [28]. Na rozdíl od binárního stromu, který má pouze dvě možnosti, kam z rodičovského uzlu zamířit, Trie má tolik možností, jak velká je abeceda definovaná nad tímto stromem. Každý symbol v řetězci určuje kam se dále ve stromu zanořit. Klíče se společným prefixem mají ve stromě stejnou cestu, která se začne lišit, jakmile jejich řetězce přestanou být stejné. Hodnoty, zde informace o IP adresách, jsou uloženy v listech, které se ve stromě nacházejí na konci každé IP adresy. Na obrázku 4.1 je struktura stromu.

Za předpokladu, že odkazy na syny jsou alokované v asociativním poli, můžeme přistupovat do konkrétního potomka pomocí indexu. Poté operace přidání, vyhledání nebo mazání mají stejnou složitost $O(n)$, kde n je délka ukládané adresy. Protože je délka adresy konstantní, můžeme prohlásit složitost za konstantní ($\theta(1)$). Výhodou je, že čas vykonání je nezávislý na celkovém počtu uložených IP adres. Nevýhoda je v obrovském množství alokované paměti. Každý uzel by musel mít k odkazů na syny, kde k je velikost abecedy stromu. Pro IP verze 4 by bylo k maximálně velikosti 11 (čísla 0–9 a znak „.“). Pro IP verze 6 by k bylo maximálně 17 (symboly 0–F a znak „:“). Protože počet IP adres během analýzy stoupá velice rychle, tento způsob ukládání se jeví jako vysoce paměťově náročný. Navíc by při práci se stromem docházelo velice



Obrázek 4.1: Příklad prefixového stromu a ukládání IP adres v něm. V obrázku je znázorněno uložení těchto adres: 1.62.8.2, 1.73.4.1, 2.82.8.2, 2.93.4.1, 2.94.2.8, 2.94.51.11.

často k výpadkům v cache, protože by hodnoty nebyly alokovány současně (v paměti vedle sebe) a tak by byla částečně degradována i rychlost.

Možností jak ušetřit paměť, je ukládat v každém uzlu pouze spojový seznam potomků. To znamená, že by se využití paměti zvyšovalo mnohem pozvolněji. Nevýhodou tohoto řešení by bylo vyhledávání, kdy bychom v každém uzlu museli s lineární složitostí vyhledávat správného potomka. Tím bychom přišli o výhodu rychlosti procházení stromu, která by degradovala na $O(n*k)$.

Pokud bychom IP adresu ukládali jako 4 čísla oddělené tečkami pro IP verze 4 respektive 16 čísel pro IP verze 6, snížila by se hloubka stromu (4 respektive 16 uzlů) a vzrostl by počet možných potomků na jeden uzel (255 možných potomků). Pro ukazatele na potomky máme opět na výběr mezi asociativním polem a spojovým seznamem.

Tento způsob se z důvodu paměťové a výpočetní náročnosti nezdá jako vhodný pro práci s IP adresami.

Binární vyhledávací strom

Binární stromy mají vždy maximálně dva potomky levého a pravého. Každý uzel má klíč, podle kterého jsou jednotlivé uzly ve stromu uspořádány. Platí, že pokud je klíč pro provedení operace (vyhledání, vložení, ...) menší než klíč v uzlu, jdeme do levého potomka, pokud je větší pokračujeme pravým potomkem a pokud je roven, našli jsme správný uzel. Pokud chceme hodnotu vložit, procházíme strom až k uzlu, ve kterém narazíme na prázdný ukazatel požadovaného potomka. Na tento ukazatel napojíme novou hodnotu.

Pro mazání, vkládání i vyhledávání je operační složitost *O(h)*, kde h je výška stromu. Výška stromu je v nejlepším případě $h = \log_2 n$. Všechny klíče by nám museli přicházet v určitém pořadí, abychom takovéhoto ideálního případu dosáhli. Na druhou stranu se může stát, že prvky budou přicházet seřazené podle klíče, složitost by byla v takovém případě *O(n)*, kde n je počet hodnot. Všechny prvky by se řadily za sebe a binární strom by degradoval na spojový seznam.

4.3.3 Vyvažovací stromy

Nevýhody klasického binárního vyhledávacího stromu jsou minimalizovány vyvažovacími stromy (VS). VS se snaží, aby hloubka stromu byla minimální. Za tímto účelem, při přidávání a mazání hodnot, upravují strukturu stromu. Operace vyvažování o něco zpomalí operace vkládání a mazání, ale zabezpečí udržení logaritmické složitosti vyhledávání ve stromě.

Máme několik typů vyvažovacích stromů. Mezi nejznámější a nepoužívanější patří AVL a červeno-černý strom, které se liší se především algoritmem pro vyvažování.

AVL strom

Pro každý uzel musí platit, že hloubka jeho potomků se může lišit maximálně o jedna. To znamená, že operace vkládání, mazání a hledání mají vždy složitost *O(*$\log n$*)*. Složitost vyvažování počítáme jako konstantní *O(1)*.

Vkládání nového prvku probíhá stejně jako u binárního vyhledávacího stromu. Po vložení se vypočítá koeficient vyváženosti (*výškalevéhopodstromu* − *výškapravéhopodstromu*). Poté se směrem zpět ke kořeni kontroluje vyváženost potomků. Jakmile, se u některého z uzlu zjistí, že není vyvážený, zahájí se algoritmus vyvažování. Pokud dojde k vyvažování, nemusíme již procházet zbytek stromu a máme jistotu, že koeficienty vyváženosti jsou správně nastaveny.

Při operaci mazaní smažeme hodnotu stejně jako v binárním vyhledávacím stromu. Poté postupujeme podobně jako při operaci vkládání, na rozdíl od které musíme kontrolovat vyváženost celé cesty od smazané hodnoty až ke kořeni.

Červeno-černý strom

Červeno-černý strom má volnější pravidla pro vyvážení než AVL strom. Uzly se dělí na červené nebo černé. V závislosti na sobě musí uzly splňovat sadu pravidel [30]:

1. Každý vrchol je buď červený, nebo černý.
2. Kořen je černý.
3. Listy jsou pokládány za černé vrcholy.
4. Každý červený vrchol má dva černé syny.
5. Každá cesta z jednoho vrcholu do jeho podřízených listů obsahuje stejný počet černých vrcholů.

Algoritmus pro vkládání a mazání je blíže popsán v [32].

V porovnání s AVL, se zde vyrovnávací operace volají méně častěji, ale hloubka stromu může být v nejhorším případě až $O(2 \log n)$.

4.3.4 B stromy

B stromy se od předchozích liší především počtem potomků m a počtem uložených klíčů $m - 1$. Implementace by měla být nezávislá na hodnotě m, která může být větší nebo rovno dvěma.

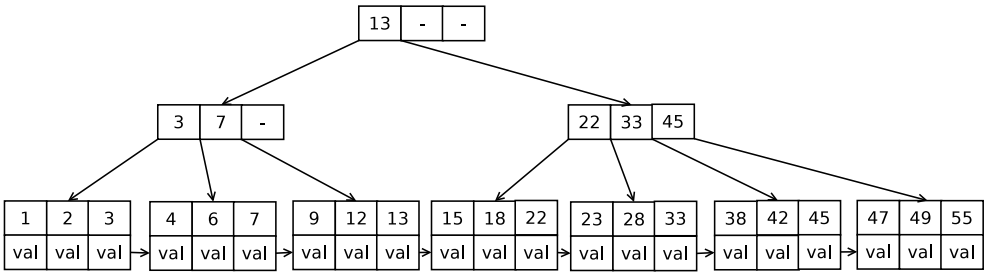
B stromy se používají tam, kde je třeba ukládat mnoho dat do pomalejší paměti, typicky na disk. Hodnoty uzlu se nacházejí v paměti vedle sebe, proto při jejich načítání dochází maximálně k několika málo výpadkům. Z paměti tedy celkově čteme méně dat, než u ostatních stromů.

B strom

Definice m-árního B stromu [27]:

1. Všechny listy mají stejnou hloubku.
2. Počet synů každého vnitřního uzlu je roven počtu jeho prvků +1.
3. Kořen obsahuje minimálně jeden prvek.
4. Pokud kořen není listem, pak má minimálně dva syny.
5. Vnitřní uzly různé od kořene musejí obsahovat minimálně $\lfloor \frac{m-1}{2} \rfloor$ prvků (a tudíž minimálně $\lfloor \frac{m-1}{2} \rfloor + 1$ synů).
6. Listy musejí obsahovat minimálně $\lfloor \frac{m-1}{2} \rfloor$ prvků.

Operace B stromu je blíže popsáné zde [27].



Obrázek 4.2: Příklad B+ stromu $m = 4$, v uzlech jsou uloženy klíče podle kterých se vyhledává. V listech se nacházejí hodnoty.

B+ strom

B+ strom je velice podobný B stromu, navíc musí splňovat tyto vlastnosti [27]:

1. Vnitřní uzly obsahují pouze klíče, nikoliv hodnoty.
2. Vlastní prvky (dvojice {klíč, hodnota}) jsou uloženy v listech.
3. Každý list obsahuje maximálně $m - 1$ prvků a odkaz na následující list.
4. Klíč na pozici i je poslední klíč v posledním listu podstromu na pozici $i + 1$.

Vkládání, mazání a vyhledávání v B+ stromu je detailně popsané v sekci 4.6 na straně 43. Všechny tyto operace mají složitost $O(log_m n)$, kde m je počet potomků uzlu a n je celkový počet prvků ve stromě.

Výhodou B+ stromu oproti B stromu je, existence spojového seznamu všech hodnot. Při potřebě práce s více po sobě seřazenými hodnotami, stačí nalézt první hodnotu a ke zbytku přistupovat pomocí spojového seznamu. Tato vlastnost se nám hodí při vyhodnocování dat na konci sekvence sběru dat, kdy postupně přistupujeme ke všem prvkům a ohodnocujeme jejich stav. Na obrázku 4.2 je znázorněná struktura stromu.

4.3.5 Vyhodnocení výběru datové struktury

Všechny uvedené datové struktury mají své klady a zápory. Nejdůležitější je zvážit k čemu přesně by datová struktura měla sloužit. V implementovaném modulu je po ní požadováno: efektivní vkládání, efektivní vyhledávání, efektivní mazání, práce s prvky jako se spojovým seznamem, nízká paměťová náročnost a možnost uložit předem nespecifikované množství prvků. Těmto kriteriím odpovídají vyvažovací stromy nebo B stromy.

Po zvážení požadavků jsem se rozhodl zúžit výběr na B+ a B strom. Ty splňují výše zmíněné požadavky, a navíc jsou efektivní, pokud vezmeme v potaz

práci s pamětí. Tyto struktury se používají především na uložištích s pomalým přístupem (například pevný disk). Při načtení dat ze zařízení, se vždy načte větší kus paměti (z disku například sektor). Protože B stromy ukládají více hodnot za sebou, tak v načtených datech využijeme více než jen jednu hodnotu. To má za následek méně výpadků, než u jiných stromových struktur. Paměť RAM, ve které jsou uloženy data běžícího programu, je považována za rychlou, ale i data z této paměti jsou kešovány v mezipaměti procesoru. Proto by měla být struktura B/B+ stromu rychlejší než AVL/červeno-černý strom.

Nyní zbývá rozhodnout mezi B+ a B stromem. Z důvodu potřeby zpracovávat data podobně jako ve spojovém seznamu, je výhodnější B+ strom, který spojový seznam z dat přímo vytváří.

Datová struktura B+ strom je použitelná i v ostatních modulech, které potřebují pracovat s datovou strukturou tabulka. Implementace této struktury bude přidána do knihovny libnemea-common, aby k ní měli přístup i ostatní vývojáři Cesnetu.

4.4 Implementace B+ stromu

Z popsaných datových struktur jsem vybral B+ strom. Tato sekce se věnuje především konkrétnímu popisu základních a rozšiřujících funkcí B+ stromu.

Základními funkcemi jsou: vyhledání prvku podle klíče (prvek = {hodnota, klíč}), přidání prvku a odebrání prvku podle klíče.

Rozšiřujícími funkcemi jsou: přidání nebo vyhledání prvku, předání seznam prvků, předání dalšího prvku ze seznamu a vymazání prvku ze seznamu. Tyto rozšiřující funkce slouží k efektivnější spolupráci s modulem.

Tuto strukturu jsem již implementoval jako semestrální práci v předmětu BI-EFA. Strom byl napsaný v jazyce C++ a pro práci s klíči a hodnotami obsahoval dvě speciální třídy *CKey* a *CValue*. První verze, kterou jsem propojil s modulem byla tato. Stačilo přidat několik funkcí na volání C++ z C jazyka. Do *Ckey* přidat proměnnou pro uložení a porovnávání IP adresy a do *CValue* přidat atributy ukládaných informací o IP adresách. Tento způsob omezoval použití stromu pouze pro konkrétní modul. Navíc pro podporu IP verze 6 musela být proměnná pro uložení adresy velká 16 B a protože většina komunikace je zatím IP verze 4, strom využíval zbytečně mnoho paměti. Z důvodu přidání implementace B+ stromu do knihovny libnemea-common, která je v jazyce C, jsem přepsal datovou strukturu do jazyka C a upravil ji, aby byla nezávislá na místě použití. Lze ji tedy použít v jakémkoliv jiném modulu a ukládat v ní jakékoliv prvky. Protože je strom nezávislý na hodnotě i klíči, je třeba mu předat informace, jakým způsobem má porovnávat klíče, velikost klíčů a velikost hodnot. Předáme mu tedy ukazatel na funkci porovnání, která vyžaduje za parametry dva klíče. Například *int compare(A, B)*. Pokud $A = B$ funkce vrátí hodnotu 1, pokud $A < B$ vrací hodnotu 1, jinak vrací 2.

4.4.1 Vytvoření a odstranění stromu

K vytvoření stromu slouží funkce *b_plus_tree_initialize()*, které jako parametry předáváme: počet potomků uzlu (hodnota m), ukazatel na funkci porovnání, velikost hodnoty a velikost klíče. Návratová hodnota je typu ukazatel na void, který slouží pro identifikaci stromu.

Pro smazání všech hodnot stromu a uvolnění alokované paměti voláme funkci *b_plus_tree_destroy()*, které předáváme ukazatel na strom.

4.4.2 Vyhledání prvku podle klíče

Vyhledání prvku zajišťuje funkce *c_b_tree_plus_search()*, které jako parametry předáváme: ukazatel na strom a ukazatel na klíč, podle kterého chceme vyhledávat. Funkce rekurzivně prochází vnitřními uzly, dokud nedorazí do listu. Pokud se hledaný klíč v listu nachází, vrátí jeho hodnotu, jinak vrací *NULL*.
Pseudokód operace vyhledání prvku podle klíče je popsán v příloze B.1.

4.4.3 Vložení prvku

Vložení prvku zajišťuje funkce *b_plus_tree_insert_item()*, která vyžaduje jako parametry odkaz na strom a odkaz na vkládaný klíč. Návratovou hodnotou je ukazatel na alokovanou paměť pro hodnotu. Pokud se klíč ve stromu již nachází, vrací *NULL*. Funkce nejprve vyhledá vhodný list pro vložení, na toto místo vloží klíč a alokuje potřebný prostor. Poté začíná kontrola velikosti uzlů. Pokud je překročen maximální počet prvků, které můžeme uložit do listu, vytvoří se nový list, polovinu položek přesune do nového listu, nastaví ukazatele na levý a pravý sousední list a odkaz na nový list vloží do rodičovského uzlu. Tím se zvětší počet hodnot v rodičovském uzlu. Pokud je tento počet menší než stanovené maximum, funkce končí, jinak dochází ke štěpení rodičovského (vnitřního) uzlu, které je velice podobné štěpení listu. Vytvoří se nový uzel, z původního uzlu se polovina položek a potomků přesune do nového uzlu a do rodiče se vloží ukazatel na nový prvek. Pokud rodič neexistuje, vytvoří se nový uzel, který bude zároveň kořenem stromu. Takto se pokračuje dokud hodnota v uzlu není menší než stanovené maximum.

Pseudokód operace vložení prvku je uveden v příloze B.2.

4.4.4 Odebrání prvku podle klíče

Odebrání prvku podle klíče zajišťuje funkce *b_plus_tree_delete_item()*, která se volá s parametry: ukazatel na strom a ukazatel na klíč.

Nejprve se vyhledá list, ve kterém se nachází daný klíč. Pokud takový list neexistuje, metoda končí. Z listu se vymaže hodnota a zkontroluje se počet prvků, který musí být větší než stanovené minimum ($\lfloor \frac{m-1}{2} \rfloor$). Pokud tuto podmínku nesplňuje může dojít k několika možnostem, které se testují v pořadí

v jakém jsou popsány, provede se ta operace, která jako první splní podmínky. 1) Existuje-li levý bratr, se stejným rodičem, který má více než minimum prvků, vyjmeme jeho poslední prvek, vložíme ho jako první prvek upraveného uzlu a aktualizujeme hodnoty klíčů v rodiči. 2) Existuje pravý bratr, se stejným rodičem, který má více než minimum prvků, vyjmeme jeho první prvek, vložíme ho jako poslední prvek upraveného uzlu a aktualizujeme hodnoty klíčů v rodiči. 3) Existuje-li levý bratr se stejným rodičem, sloučíme ho s upravovaným uzlem, vymažeme ukazatel na smazaný prvek v rodiči a aktualizujeme ukazatele na sousední prvky 3) Existuje-li pravý bratr se stejným rodičem, sloučíme ho s upravovaným uzlem, vymažeme ukazatel na smazaný prvek v rodiči a aktualizujeme ukazatele na sousední prvky. U možností 3 a 4 musíme volat funkci na kontrolu počtu potomků v rodiči. Pokud klíčů a potomků bude méně než je stanovené minimum, bude probíhat úprava uzlů obdobně jako je tomu u listů. Pokud se nevybere ani jedna z možností, znamená to, že uzel ve kterém se nacházíme je kořenem.

Pseudokód operace odebrání prvku je uveden v příloze B.3.

4.4.5 Přidání nebo vyhledání prvku

Funkce *b_plus_tree_insert_or_find_item()* slouží k přidání záznamu. Pokud již klíč ve stromě existuje, nevrací *NULL* jako funkce *b_plus_tree_insert()*, ale ukazatel na nalezenou hodnotu prvku. Tuto funkci využívá modul ve fázi sběru informací, kdy potřebuje uložit informace o IP adrese. Jako parametry požaduje: ukazatel na strom a ukazatel na klíč. Vrací ukazatel na nově alokovanou nebo nalezenou hodnotu prvku.

4.4.6 Vytvoření a smazání seznamu prvků

Funkce *b_plus_tree_create_list_item()* vytvoří strukturu *b_plus_tree_item*, ve které se ukládají: hodnota, klíč, index v listu a odkaz na list. Poslední dva argumenty slouží pro identifikaci přesného místa prvku ve stromu. Díky nim můžeme v konstantním čase načíst další prvek. Jako parametr požaduje ukazatel na strom a vrací ukazatel na zmiňovanou strukturu

Funkce *b_plus_tree_get_list()* slouží ve fázi vyhodnocení, kdy potřebujeme postupně přistupovat ke všem prvkům. Jako parametry požaduje: ukazatel na strom a ukazatel na zmíněnou strukturu. Funkce nalezne nejlevější prvek (s nejnižší hodnotou klíče) a uloží jej do struktury *b_plus_tree_item*, kterou vrátí jako návratovou hodnotu

Funkce *b_plus_tree_destroy_list_item()* slouží pro uvolnění paměti alokované strukturou *b_plus_tree_item*. Typicky jí voláme při ukončování modulu. Jako parametr jí předáváme ukazatel na alokovanou strukturu.

4.4.7 Další prvek ze seznamu

Funkce *b_plus_tree_get_next_item_from_list()* slouží pro získání dalšího prvku ze seznamu IP adres. Jako parametry požaduje: ukazatel na strom a ukazatel na alokovanou strukturu *b_plus_tree_item*. Návratovou hodnotou je 1, pokud byl načten další prvek a 0, pokud jsme došli na konec seznamu a další hodnota neexistuje.

4.4.8 Vymazání prvku ze seznamu

Funkce *b_plus_tree_delete_item_from_list()* slouží pro odstranění prvku, bez nutnosti jej vyhledávat ve stromu, protože již přesně známe list i index na kterém se prvek nachází. Po odstranění prvku načte další položku ze seznamu. Jako parametry požaduje: ukazatel na strom a ukazatel na alokovanou strukturu *b_plus_tree_item*. Návratovou hodnotou je 1, pokud byl načten další prvek a 0, pokud jsme došli na konec seznamu a další hodnota neexistuje.

4.5 Výběr datové struktury pro ukládání a vyhledávání podle doménových jmen

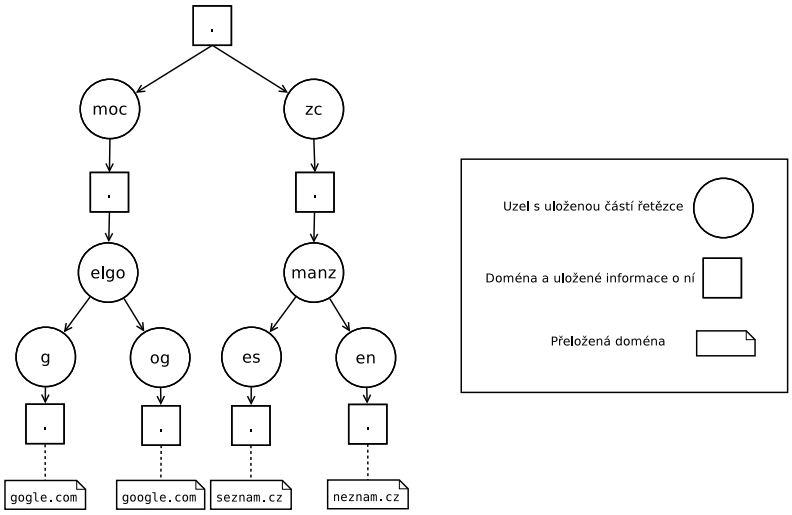
Modul ukládá podezřelá doménová jména DNS žádostí, u nichž poté sleduje: počet vyhledávání, počet poddomén, nejvíce vyhledávané domény a nejméně vyhledávané domény.

Doména se dělí na několik částí oddělených tečkou. Jedná se o stromové členění DNS záznamů. Pro ukládání těchto dat může být použita stromová struktura Trie, která je již popsaná v sekci 4.3.2 na straně 34.

Ve struktuře prefixového stromu bude potřeba provést pár změn. Záznamy se klasicky ukládají, tak jak je čteme a to zleva doprava. To znamená, že by se spojovaly domény, které mají totožný nejvyšší řád. My ale potřebujeme shlukovat domény od nejnižšího řádu, tak jak je to pro DNS přirozené. Potřebujeme tedy obrátit pořadí a například místo *www.translate.google.com* uložit doménu do stromu jako *com.google.translate.www*. Obracet pořadí poddomén by bylo zbytečně výpočetně složité. Jednodušší bude celou doménu ukládat zprava doleva. Například doména *www.translate.google.com* bude uložena jako *moc.elgoog.etalsnart.www*. Toto ukládání nás bude stát minimum času navíc.

Další změnou jsou ukládané informace. Informace o poddoméně budeme ukládat pokaždé, když se v řetězci objeví tečka. O poddoméně budeme ukládat počet potomků a počet vyhledávání.

Poslední změnou je relaxace stromu, tak aby byl méně paměťově náročný. Uzly, které mají pouze jednoho potomka, budeme shlukovat do jednoho uzlu. To znamená, že pokud existuje poddoména, která je v daném stromě unikátní, budeme ji mít uloženou pouze v jednom uzlu. Pokud dojde k přidání poddomény se stejným prefixem (v již obráceném pořadí), rozštěpíme uzel na místě změny a zbytek řetězců uložíme do nových uzlů. Například, máme uloženou



Obrázek 4.3: Příklad prefixového stromu pro ukládání a vyhledávání domén. Domény jsou ukládány pozpátku.

poddoménu *elgoog* a přijde požadavek na uložení *elgog*. Nová poddoména je až do čtvrtého písmene identická. Rozdělení tedy bude na uzel *elgo*, na který navazují dva nové uzly *og* a *g*. Strom v grafickém znázornění vypadá takto 4.3.

Ze základních operací je potřeba pouze operace vkládání. Rozšiřujícími funkcemi jsou přidání výjimky a testování výjimky (tyto operace jsou popsány v sekci 4.6). Operační náročnost přidání prvku je $O(m)$, kde m je počet písmen v doméně. Paměťová složitost je $O(k*n)$, kde k je počet písmen abecedy, které doména může obsahovat a n je celkový počet uzlů. Ten je v nejhorším možném případě a při největším možném zaplnění roven k^l, kde l je délka nejdelší domény. Díky relaxaci tento počet bude reálně mnohem menší. V nejlepším případě bude n rovno počtu poddomén.

Současně s vytvářením stromu si budeme ukládat seznam nejčastěji a nejméně vyhledávaných domén.

4.6 Implementace prefixového stromu

Datová struktura se skládá ze základny stromu a dvou typů uzlů.

Základna udržuje ukazatel na kořen stromu a obecné informace o stromu jako jsou:

- count_of_domain_inserted_just_ones - Počet domén vkládaných pouze jednou (z důvodu možnosti ztráty paketu na síti, „pouze jednou“ je bráno jako pouze n-krát, kde n je pevně stanovená konstanta).

- count_of_inserting - Celkový počet vkládání (počítají se i duplikace).

- count_of_different_domains - Počet uložených domén
- list_of_most_used_domains - Seznam domén seřazených podle počtu vkládání sestupně (domény vyhledávané vícekrát než určitá konstanta).
- list_of_most_unused_domains - Seznam domén vyhledávaných pouze jednou.
- list_of_most_subdomains - Seznam poddomén seřazených podle počtu vkládání sestupně (seznam je ukládán pouze do určité hloubky).

Prvním typem uzlu je *prefix_tree_domain_t*, který slouží pro uložení informací o doméně. Mimo dat, které tvoří architekturu datové struktury obsahuje:

- exception - Příznak zda doménové jméno slouží jako výjimka pro vyhledávání. Více o této proměnné v sekci 4.7.2.
- degree - Řád domény.
- count_of_insert - Počet vkládání.
- count_of_different_letters - Počet unikátních písmen, ze kterých se doména skládá.
- count_of_different_subdomains - Počet potomků.
- prefix_tree_domain_t - Ukazatel na doménu nižšího řádu, pokud neexistuje tak *NULL*.

Druhým typem uzlu je *prefix_tree_inner_node_t*, který zajišťuje spojení jednotlivých uzlů a uchovávají se v něm části řetězce domény.

4.6.1 Vytvoření a smazání stromu

Vytvoření stromu implementuje funkce *prefix_tree_initialize()*, která nevyžaduje žádné parametry a návratovou hodnotou je ukazatel na vytvořený strom. Funkce alokuje strukturu základny a kořenovou doménu nultého řádu.

Mazání stromu implementuje rekurzivní funkce *prefix_tree_destroy()*, jako parametr požaduje ukazatel na strom. Funkce postupně prochází a odstraňuje jednotlivé uzly.

4.6.2 Vkládání do stromu

Operace, která zajišťuje vložení domény do stromu a aktualizaci potřebných údajů, pokud se již doména ve stromě nachází, vrátí ukazatel na ni.

Vkládání implementuje funkce *prefix_tree_add_domain()*, jako vstupní parametry vyžaduje ukazatel na strom, délku domény a volitelně strukturu

char_stat, která obsahuje informace o řetězci domény. Pokud tato struktura nebude zadána, nebudou uloženy informace, které přenáší. Návratovou hodnotou je ukazatel na strukturu *prefix_tree_domain_t*.

Vkládání začíná zjištěním společné části řetězce, která se již ve stromu nachází. Pokud současně došlo k přečtení celé domény pro vložení (doména ve stromu již existuje), aktualizují se informace o doméně a funkce končí. V opačném případě mohou nastat následující možnosti. 1) Přečetl se celý řetězec uložený v příslušném uzlu, poté vytvoříme nový uzel, do kterého uložíme unikátní část poddomény a připojíme ho na příslušný ukazatel původního uzlu (indexem je následující znak v řetězci). Pokud ve vkládaném řetězci existuje další poddoména, tuto akci opakujeme. 2) Pokud se přečetla pouze část řetězce v uzlu, uzel se musí rozdělit, tak aby obsahoval pouze společnou část. Po rozdělení původního uzlu se pokračuje jako v prvním případě.

Pseudokód operace vložení domény do stromu je uveden v příloze C.1.

4.6.3 Výjimky z detekce

Operace přidání výjimky slouží pro označení domén, které nebudeme chtít detekovat, a tudíž o nich nebudeme shromažďovat žádné informace. Tato operace přidá do stromu doménu a označí ji jako výjimku. Na přidávání ostatních domén výjimka nemá vliv, slouží pouze funkci testování výjimky.

Funkce *prefix_tree_add_domain_exception()* přijímá za parametry: ukazatel na strom, řetězec domény a délku domény. Návratovou hodnotou je ukazatel na alokovanou strukturu domény. Tato funkce funguje obdobně jako *prefix_tree_add_domain()*, jedinou změnou je, že u vložené domény změní příznak exception na *true*.

Operace testování výjimky zkontroluje, zda se ve stromu nachází daná doména a zároveň patří mezi výjimky.

Funkce *prefix_tree_is_domain_in_exception()* přijímá za parametry: ukazatel na strom, řetězec domény a délku domény. Návratovou hodnotou je 1, pokud se doména ve stromu nachází a má aktivní příznak *exception*, jinak 0.

4.7 Implementace modulu

Modul se spouští z příkazové řádky příkazem *./tunnel_detection_dns* s přepínači. Použití přepínačů je popsáno v sekci 4.7.6.

Po spuštění modulu se odehrávají následující operace:

1. Načtou se přepínače a uloží se z nich potřebné informace.
2. Načtou se domény, které jsou v souboru s výjimkami a uloží se do datové struktury prefixového stromu.
3. Inicializují se dva B+ stromy, jeden pro ukládání informací o adresách IP verze 4 a druhý pro IP verze 6.

4. Inicializuje se parser dat, který načítá soubor s pakety nebo se pro příjem paketů nastaví knihovna UniRec.

5. Začíná hlavní cyklus modulu. Viz 4.7.1

6. Přijímání paketů a základní payload analýza. Viz 4.7.2 a 4.7.3.

7. Ukončení přijímání paketů

8. Vyhodnocení a výpis anomálií. Viz 4.7.4.

9. Pokud jsou ke zpracování další pakety a nebyl přijat signál pro ukončení, pokračování krokem 6.

10. Výpis výsledků detekce do souborů. Viz 4.7.5.

11. Uvolnění alokovaných struktur.

12. Konec programu

4.7.1 Hlavní cyklus modulu

Jedná se o smyčku, ve které se řídí přechod mezi načítáním paketů a vyhodnocením. Smyčka začíná kontrolou existence paketu k přečtení, pokud neexistuje, hlavní smyčka končí. V opačném případě se spouští vnitřní smyčka, která načítá pakety přesně stanovenou dobu (lze nastavit parametrem) nebo dokud nepřijme signál k ukončení. Ukončení za určitý interval je zajištěno uložením času prvního paketu a kontrolou času každého dalšího paketu. Pokud čteme data ze souboru, pakety do modulu přichází ihned po sobě, nemůžeme tedy počítat s reálným časem, ale časem uloženým v hlavičce rámce paketu. Ve vnitřním cyklu se kontroluje, zda paket není v seznamu výjimek z detekce. Poté vložíme paket do funkce, která zajišťuje uložení potřebných informací a základní payload analýzu. Po ukončení vnitřní smyčky získáme seznam IP adres a informací o tocích jimi tvořených. Tento seznam je odeslán k vyhodnocení. Po ukončení vyhodnocení začíná hlavní smyčka od začátku.

4.7.2 Sběr informací z paketů

Sběr informací je chápán jako vyhledání nebo přidání IP adresy do B+ stromu a aktualizace informací datové struktury, kterou obdržíme jako návratovou hodnotu funkce *b_plus_tree_insert_or_find_item()*. Dále probíhá základní payload analýza, založená na informacích získaných z paketu. Tyto operace zajišťuje funkce *collection_of_information_and_basic_payload_detection()*, které jako parametry předáváme ukazatel na B+ strom, IP adresu paketu, velikost paketu, příznak zda se jedná o žádost či odpověď a obsah paketu.

Pro každou IP adresu ukládáme struktury popsané v D.1.

Ve struktuře *ip_address_t* jsou uloženy čtyři prvky, jejichž jména začínají *ip_address_suspicion...* Jedná se o realizaci typů detekce anomálií popsané v sekci 3.3.1. Pokud se typ dostane do stavu *SUSPICION*, alokuje se struktura, která udržuje informace o podezřelých doménách, TXT, CNAME, MX nebo NS záznamech. Tyto informace jsou ukládány do prefixového stromu.

Realizace typu „request tunnel“
: Datový typ *ip_address_suspicion_request_tunnel_t* v sobě uchovává: pole, ve kterém určuje velikosti žádostí, které se budou ukládat pro hlubší analýzu; ukazatel na prefixový strom, kde jsou uloženy podezřelé domény; počet cyklů vyhodnocení, ve kterých byla struktura ve stavu *SUSPICION*.
Struktura je uvedena v příloze D.2.

Realizace typu „request other anomaly“
: Datový typ *ip_address_suspicion_request_other_anomaly_t* uchovává stejné informace jako realizace typu request tunnel.
Struktura je uvedena v příloze D.3.

Realizace typu „response tunnel“
: Datový typ *ip_address_suspicion_response_tunnel_t* v sobě uchovává: proměnnou určující typy záznamů, které se budou ukládat pro hlubší analýzu; ukazatele na prefixové stromy každého typu záznamu, ve kterých jsou uloženy podezřelé záznamy; počet cyklů vyhodnocení, ve kterých byla struktura ve stavu *SUSPICION*.
Struktura je uvedena v příloze D.4.

Realizace typu „response other anomaly“
: Datový typ *ip_address_suspicion_response_other_t* v sobě uchovává: ukazatel na prefixový strom, kde jsou uloženy podezřelé domény; počet cyklů vyhodnocení, které struktura absolvovala ve stavu *SUSPICION*; počet odpovědí bez doménového jména; celkový počet podezřelých odpovědí.
Struktura je uvedena v příloze D.5.

4.7.3 Základní payload analýza při sběru informací

Již během sběru informací a ukládání souhrnných údajů probíhá payload analýza řetězců. Vypočítává se počet unikátních písmen v řetězci a počet číslic v řetězci. Pokud některý z těchto dvou údajů přesáhne stanovené maximum, řetězec bude uložen do prefixového stromu a příslušný typ změní svůj stav na *SUSPICION*. Analýza se provádí u žádostí, MX, CNAME, NS a TXT záznamů.

Analýzu řetězce provádí funkce *calculate_character_statistic()*, která jako parametry požaduje řetězec a adresu struktury, do které uloží výsledky.

4.7.4 Vyhodnocení

Vyhodnocení probíhá pro každou IP adresu zvlášť. Z B+ stromu se načte seznam adres, které postupně procházejí jednotlivými testy. Posloupnost testování je popsána pseudokódem v příloze E.

Pokud jsou po testech všechny typy detekce ve stavu *OK* (komunikace IP adresy neobsahuje anomálie), vymažeme IP adresu ze seznamu (zároveň z B+ stromu). Naopak pokud některý typ detekce změní stav na *ATTACK*, nahlásíme nalezení anomálie.

Jednotlivé testy jsou implementovány funkcemi, které jsou popsány v tabulce 4.1. Pokud jsou splněny popsané podmínky, funkce vrací hodnotu 1, což znamená, že daná funkce neprokázala výskyt anomálie.

Tabulka 4.1: Funkce reprezentující testy. Funkce jsou rozděleny do dvou kategorií *is_payload_on_ip_ok_* a *is_traffic_on_ip_ok_*

Funkce	**Podmínky**
is_payload_on_ip_ok_	
request_other	procento opakovaných doménových jmen < MAX a procento různých doménových jmen > MIN
request_tunnel	procento vyhledávání různých domén < MAX
response_other	procento opakovaných doménových jmen < MAX a procento různých doménových jmen > MIN a procento odpovědí bez doménového jména < MAX
response_tunnel	procento různých záznamů TXT, CNAME, MX, NS < MAX
is_traffic_on_ip_ok_	
request_other	střední hodnota a rozptyl jsou v daném intervalu
request_tunnel	střední hodnota a rozptyl jsou v daném intervalu
response_other	střední hodnota a rozptyl jsou v daném intervalu

.

4.7.5 Výstup modulu

Modul zapisuje výsledné soubory do předem specifikované složky. Pokud tato složka neexistuje, vytvoří ji. Volitelně můžeme spustit skript *./create_graphs [složka s daty]*, který vytvoří soubory pdf s histogramy (viz příloha F).

Ve složce se nacházejí soubory:

- found_anomaly.txt
  Uložené výsledky detekce, kde jsou za sebou uložené IP adresy, ve kterých byla nalezena nějaká anomálie. Výsledky jsou uložené ve tvaru:

```
...
[IP address]
    Request tunnel found:
            Domains searched just once: xxx
            Count of different domains: xxx
            Percent of subdomain in most used domain: xxx
            All recorded requests: xxx
        [domain] [count of searching]
        [domain] [count of searching]
        ...
    Request traffic anomaly found:
            Domains searched just once: xxx
            Count of different domains: xxx
            All recorded requests: xxx
            Count of malformed requests: xxx
        Found in sizes: xx-xx
        [domain] [count of searching]
        [domain] [count of searching]
        ...
    Response tunnel found:
            Strings searched just once: xxx
            Count of different domains: xxx
            All recorded requests: xxx
        [string] [count of searching]
        [string] [count of searching]
        ...
    Response anomaly found:
            EX: xxx
            VAR: xxx
            Percent without request string: xxx
            Count of responses xxx
        [domain] [count of searching]
        [domain] [count of searching]
...
```

Prvky v hranatých závorkách jsou ve výstupu nahrazeny konkrétní hodnotou. Například místo [IP address] je 192.168.0.1.

- suspicion_list.txt
  Seznam s IP adresami, které byly v době výpisu ve stavu *SUSPICION*.

- request_letters_count.dat
  Data histogramu počtu unikátních písmen v závislosti na velikosti paketu pro každou IP adresu zvlášť. Grafický výstup se nachází v *graphs/request_letters_count.pdf*.

- requests.dat
  Data histogramu počtu žádostí v závislosti na velikosti paketu pro každou IP adresu zvlášť. Grafický výstup je v *graphs/requests.pdf*.

- responses.dat
  Data histogramu počtu odpovědí v závislosti na velikosti paketu pro každou IP adresu zvlášť. Grafický výstup je v *graphs/responses.pdf*.

- summary_requests.dat
  Data histogramu, který zobrazuje počet žádostí v závislosti na velikosti paketu. Grafický výstup je v *graphs/summary.pdf*.

- summary_responses.dat
  Data histogramu, který zobrazuje počet odpovědí v závislosti na velikosti paketu. Grafický výstup je v *graphs/summary.pdf*.

4.7.6 Parametry pro spuštění modulu

Modul je možné nastavit přidáním přepínačů při spuštění. Můžeme upravovat limitní hodnoty v testech, dobu vykonání cyklu, cestu ke vstupnímu souboru a cestu k adresáři pro výstup. Povinné přepínače jsou: *-f* vstupní soubor s pakety nebo *-i* vstupní rozhraní UniRec. Mezi nejdůležitější volitelné přepínače patří: *-s* adresář pro výpis výsledků; *-d* soubor pro průběžný výpis anomálií; *-e* soubor s domény, které nebudeme detekovat (výjimky z detekce viz 4.6.3) a *-h* vypíše nápovědu. Pokud nejsou specifikované hodnoty v limitních testech, jsou načteny defaultní hodnoty, které jsou zapsané jako konstanty v hlavičkovém souboru modulu.

Kapitola 5

Testování modulu

V této kapitole je nejprve popsáno testování detekčního modulu na připravených modelových datech a poté testování na datech ze skutečné živé sítě CESNET2. Jsou zde popsány nalezené anomálie a diskuze nad nimi. Dále výpočetní a paměťová složitost detekce. Nakonec je zhodnocena úspěšnost detekce a možnost využití na síti.

5.1 Dynamická analýza kódu

Detekční modul jsem otestoval v programu Valgrind [14]. Tento program neidentifikoval žádné chyby a ani neuvolněnou paměť po ukončení běhu modulu.

5.2 Testování v simulovaném prostředí

Pro ověření funkčnosti modulu jsem modul nejdříve testoval v simulovaném prostředí virtuální sítě, stejném jako pro účely analýzy. Modul při tomto testování uspěl na 100 procent. Identifikoval komunikaci tunelu jak v žádostech, tak v odpovědích. V případě vstupního souboru bez anomálií, modul správně neidentifikoval žádnou hrozbu, dokonce žádnou komunikaci neoznačil jako podezřelou.

5.3 Testování na datech sítě CESNET2

Testovací data jsem dostal od svého vedoucího v rámci spolupráce s organizací Cesnet. Jedná se o 16 hodin komunikace a to od neděle 23:58 do pondělí 15:05. Za tuto dobu bylo uloženo zhruba 88 GB dat v *PCAP* formátu, které mají zdrojový nebo cílový port roven 53 (předpokládáme DNS data). Data byla zachycena sondou nacházející se na hranici s ACOnetem [1]. Před započetím práce byly všechny IP adresy anonymizovány. Toto opatření je v zájmu zachování anonymity uživatelů.

5.3.1 Nalezené tunely

V této sekci jsou popsány domény, které byly modulem ohlášeny jako potencionální tunel. Tyto domény jsem rozdělil do podsekcí podle typu jejich komunikace: tunelovaná komunikace, content delivery network & load balancing a reverzní překlad.

Tunelovaná komunikace

Tyto domény pravděpodobně obsahují ve svých poddoménách tunelovanou komunikaci.

- domena1.com
  Domény typu:

  1. `C2C627B3B3114188892DBE301E68901A0184692F.B17FCEDBE71A644A95283B0480DD2CBB3B523DDE.c27d.ko-24107.v3.url. domena1.com`
  2. `0.r4---sn-xoxgbvuxa-cune.googlevideo.com_-.videoplayback.8bcc.ko-24107.v3.url. domena1.com`
  3. `0.http_--_-._-.www.youtube.com_-.video_-.h-t0Kd-rRlU.ca4a.ko-24107.v3.url. domena1.com`
  4. `0.s2.youtube.com_-.s.b97a.ko-24107.v3.url. domena1.com`
  5. `0.csi.gstatic.com_-.csi.9906.ko-24107.v3.url. domena1.com`

  První doména z ukázky je zjevně kódovaná. Zbytek je nestandardní, protože domény nižšího řádu jsou zakódované adresy pravděpodobně jiné komunikace. Dochází zde k přenášení informací v jiné struktuře, než je klasické DNS vyhledávání, tento způsob komunikace by mohl být považován za tunel.
  Firma, které patří tato doména se zaobírá bezpečností na internetu (detekce botnetu, phishingu a dalších) a analýzou informací ze sítě.

- domena2.net
  Domény typu:

  1. `20368f25f518172a55329ff2ebfbe0b4.ebay.com.dnsbl7.domena2.net`
  2. `57b6a974b96a28f2582336e8a9b31687.ebay.com.dnsbl7.domena2.net`
  3. `05019301a15cbca581cc4eff8420249d.ebay.com.dnsbl7.domena2.net`
  4. `4232eaa395c10151835526c4a8e40cd6.ebay.com.dnsbl7.domena2.net`
  5. `c374c7955cbbc7e3b8bc113184b69195.ebay.com.dnsbl7.domena2.net`

  Tato doménová jména se vyskytují u několika IP adres. Na první pohled je vidět, že se zde posílají kódovaná data.

Podle oficiálních stránek majitele domény se jedná o službu pro bezpečný překlad IP adres. Tuto službu používá několik antivirových programů.

- domena3.ac.at
  Domény typu:

  1. `5304dbfe17de8ed7cff8d26a1e45e2e5.generic.ixhash.net.domena3.ac.at`
  2. `5304dbfe17de8ed7cff8d26a1e45e2e5.ix.dnsbl.manitu.net.domena3.ac.at`
  3. `2898bad18ca8396453c8c289e98a897d.generic.ixhash.net.domena3.ac.at`
  4. `15a5b2da9106d0bcdd7b76f0c9e64e1a.ix.dnsbl.manitu.net.domena3.ac.at`
  5. `2898bad18ca8396453c8c289e98a897d.ix.dnsbl.manitu.net.domena3.ac.at`

  Podle doménových jmen lze říci, že se skoro jistě jedná o tunelovanou komunikaci.
  Doména je registrována vysokou školou v Rakousku.

- domena4.sk
  Domény typu:

  1. `ltxcuaacaakjngaaaeaaaagsaqaabhh6ovo2cp3kedmps3ieaeaaaaa3h.nsxyacaaiaaaqakaof5eyrvaanrcsdxmbhe45jjcb3lcjr7y.domena4.sk`
  2. `ghrsuaacaakjngaaaeaaaagsaqaabhh6ovo2cp3kedmpt7ieaeaaaaa3y.gyy5acaaiaaaqbiuelfhfvu5lqjoyounajv6ccokusrmqj7y.domena4.sk`
  3. `fp4suaacaakjngaaaeaaaagsaqaabhh6ovo2cp3kedmpt7ieaeaaaaa3m.wvxuacaaiaaaqbuikomje3t7uab3vmf55ydjxg47jlyphl7y.domena4.sk`
  4. `u57cuaacaakjngaaaeaaaagsaqaabhh6ovo2cp3kedmps3ieaeaaaaa3j.tdcqqcaaiaaaqalt2ln7gh7kagyz4yqk4knfgddlbh2uyv7y.domena4.sk`
  5. `kp2suaacaakjngaaaeaaaagsaqaabhh6ovo2cp3kedmps3ieaeaaaaa3e.yqq7acaaiaaaqa264suy4s7kznygsztu5dc5ulsqbh73td7y.domena4.sk`

  Tato doménová jména se vyskytují u několika IP adres. Na první pohled je vidět, že se jedná o tunelovanou komunikaci.
  Doména je registrována známou antivirovou společností.

Content delivery network & Load balancing

Domény, které zprostředkovávají distribuční systém doručující data uživatelům například na základě jejich geografické pozice.

- metric.gstatic.com
  Domény typu:

```
1. p4-cww5vaz3clggm-74h5vsvms7ufdznj-if-v6exp3-v4.metric.
   gstatic.com
2. p5-cdj3bvlvuzd4o-lwdo732xsgbgbknl-633995-i2-v6exp3-ds.
   metric.gstatic.com
3. p5-cdj3bvlvuzd4o-lwdo732xsgbgbknl-633995-i1-v6exp3-v4.
   metric.gstatic.com
4. p4-e3lrtfsmixjh4-nlfug7i432cptucw-if-v6exp3-v4.metric.
   gstatic.com
5. p4-c3jfxc55u5zyo-43a6vpmizz7aikle-249350-i1-v6exp3-ds.
   metric.gstatic.com
```

  Tento typ domény se vyskytuje u mnoha IP adres, kde za minutu přijde zhruba 70 žádostí.
  Je vidět, že v doménách se přenášejí zakódovaná data.
  Doména druhého řádu napovídá, že se jedná o službu od společnosti Google. Podle [11] se jedná o službu pro urychlení prohlížení webu. Uživatelé místo klasických DNS dotazů zakódují dotaz do speciální žádosti zakončené *metric.gstatic.com*, tento dotaz přijde na Google server, který by měl odeslat zpět nejvhodnější IP adresu požadovaného serveru (nejbližší k lokaci uživatele). O této službě se mi nepodařilo, v době psaní tohoto textu, najít další informace.
  Tuto doménu budeme považovat za bezpečnou a pro příští detekci ji můžeme zahrnout mezi výjimky z detekce.

- d.akamai.net
  Domény typu:

```
1. a543.d.akamai.net
2. a548.d.akamai.net
3. a595.d.akamai.net
4. a676.d.akamai.net
5. a634.d.akamai.net
```

  U této domény bylo za minutu naměřeno 234 unikátních žádostí, které se lišili poddoménou čtvrtého řádu. Je vidět, že nedochází k odesílání kódovaných dat, pouze se posílají unikátní žádosti. Mohlo by jít o potvrzování streamu.
  Podle oficiálních stránek [2] se jedná o službu, která pomáhá zrychlit stahování dat z internetu.

- vk.me
  Domény typu:

```
1. login.vk.com
2. cs403231.vk.me
3. cs412822.vk.me
4. cs425728.vk.me
5. cs6077.vk.me
```

U této domény bylo za minutu naměřeno 216 unikátních žádostí, které se lišily poddoménou třetího řádu. Protože *vk.me* je známá ruská sociální síť, jedná se zřejmě o legitimní komunikaci.
Tuto doménu budeme považovat za bezpečnou a pro příští detekci ji můžeme zahrnout mezi výjimky z detekce.

Reverzní překlad

Překlad IP adresy na doménové jméno. Dotazy obsahují IP adresu a jsou zakončeny řetězcem „.in-addr.arpa“.

- in-addr.arpa
  Domény typu:

```
1. xxx.xxx.xxx.xxx.in-addr.arpa
2. xxx.xxx.xxx.xxx.in-addr.arpa
```

  Podle doménových jmen se jedná o reverzní překlad. Tento typ není tunelem, ale protože jeho struktura je podobná tunelu, byl také tak analyzován. Podobnost s tunelem je: mnoho číslic v doméně, mnoho potomků pro doménu druhého řádu, žádosti se nevyskytují opakovaně.
  Tyto dotazy budeme považovat za bezpečné a pro příští detekci je můžeme zahrnout mezi výjimky z detekce.

5.3.2 Ostatní anomálie

V této sekci jsou popsány ostatní nalezené anomálie v DNS komunikaci. Identifikované domény se v tomto případě vyznačují velmi častým vyhledáváním stejných záznamů. Většinou se zřejmě jedná o DDOS útoky nebo DNS amplification [20].

- isc.org
  Tato doména patří u mnoho IP adres mezi nejvíce vyhledávané. Počet vyhledání za jednu minutu u jedné IP adresy je průměrně 350. Za několik hodin komunikace je počet vyhledávání této domény desetitisíce.
  Podle [7] vlastní tuto doménu společnost zabývající se vývojem v oblasti síťových technologií. Podle internetových fór se jedná o DDOS útok na servery této společnosti.

- domena5.sk
  Tato doména se vyskytuje podezřele často u několika IP adres. Počet vyhledání za jednu minutu u jedné IP adresy je průměrně 60.
  Více informací, než že je doména registrovaná na Slovensku se mi nepodařilo získat.
  V tomto případě se může jednat o útok, nebo špatně napsaný program.

- domena6.pl
  Tato doména se vyskytuje podezřele často u několika IP adres. Počet vyhledání za jednu minutu u jedné IP adresy je průměrně 100.
  Doména patří polské vývojářské společnosti.

- domena7.com
  Domény typu:

  ```
  1. ksn-url-geo.domena7.com
  2. ksn4-12.domena7.com
  3. ksn-verdict-geo.domena7.com
  4. ksn-url-geo.domena7.com
  5. ksn-tpcert-1.domena7.com
  ```

  Tyto domény jsou velice často vyhledávány jednou IP adresou. Počet vyhledání první domény za jednu minutu je 218.
  Tyto domény vlastní společnost, která se zaměřuje na software pro počítačovou bezpečnost.
  Podle informací z fór se jedná o program, který neustále komunikuje se servery společnosti.

- Malformed packets
  Pakety, které se nepodařilo přečíst. Některé IP adresy těchto paketů mají pouze pár a proto nebyly detekované. Jsou zde ale i adresy, které mají těchto dotazů stovky za minutu. Může jít o tunel, nebo o spatně komunikující program.

- fkfkfkfa.com
  U této domény nebyly zachyceny žádosti, ale pouze odpovědi. Ty se vyskytují u mnoha IP adres a dosahují zhruba počtu 5000 paketů za minutu na jednu IP adresu. Velikost paketů se pohybuje kolem 1400 B. V odpovědích se většinou posílají desítky A záznamů.
  Tato doména je registrovaná v Americe a nejsou k ní žádné detailnější informace. S největší pravděpodobností se jedná o DDOS útok.

- qww1.ru
  Stejné příznaky jako u *fkfkfkfa.com*. Vyskytují se u mnoha IP adres a dosahují zhruba počtu 2000 paketů za minutu na jednu IP adresu.

- forgy5307.cloudns.info
  Stejné příznaky jako u *fkfkfkfa.com*. Vyskytují se u několika IP adres a dosahují zhruba počtu 8000 paketů za minutu na jednu IP adresu.

- ns.unitra.sk
  Stejné příznaky jako u *fkfkfkfa.com*. Vyskytují se u jedné IP adresy a dosahují zhruba počtu 28000 paketů za minutu. Rozdíl je zde ve velikosti paketů, která je průměrně 440 B.

- saveroads.ru
  Stejné příznaky jako u *fkfkfkfa.com*. Vyskytují se u několika IP adres a dosahují zhruba počtu 6000 paketů za minutu na jednu IP adresu.

5.4 Zhodnocení

V testovacích datech bylo nalezeno mnoho anomálií většinou stejného typu, zřejmě se jedná o DOS, DDOS nebo DNS amplification útoky. Některé z nahlášených anomálií byly legitimní a proto byly doporučeny k zahrnutí mezi výjimky z detekce.

Z celkového počtu 15 hodin komunikace, 95 655 781 paketů a 4 286 927 IP adres bylo ve stavu SUSPICION 1 672 IP adres a nahlášených anomálií bylo 66. Z toho 45 DDOS útoků, 6 CDN & load balancing, 5 IP adres s malformed pakety, 4 reverzní vyhledávání, 4 DNS tunely a 2 chyby (doménová jména, která neměla být nahlášena jako anomálie).

Modul byl spuštěn na počítači s procesorem Intel Core2 Duo E8500 s frekvencí $3,16\ Ghz$. Celková doba vykonání programu, s výchozím nastavením, pro testovací data trvá průměrně $176\ s$ (počítáno bez čtení dat z disku a zpracování parserem). Během této doby se vykoná 953 cyklů, ve kterých dochází k 10 233 995 přidání, 10 224 820 odstranění, 85 421 786 vyhledání IP adresy v B+ stromu a 16 942 580 přidání domény do prefixových stromů. V průběhu vykonávání je v B+ stromu před vyhodnocením průměrně 9 776 IP adres a po vyhodnocení průměrně 581 IP adres.

Pro sběr informací o délce intervalu $60\ s$ a zhruba 100 000 paketů, zaberou operace modulu průměrně $185\ ms$. Modul tak splňuje požadavky pro nasazení do současného provozu na páteřní síti CESNET2 a je již v testovacím režimu nasazen.

Závěr

Zadáním bakalářské práce bylo analyzovat, navrhnout a poté implementovat detekci tunelování přes protokoly DNS nebo HTTP. Z těchto dvou protokolů jsem si vybral DNS, pro HTTP jsem provedl analýzu, ve které jsem porovnal známé metody detekce.

V částech „Analýza tunelování přes HTTP“ a „Analýza tunelování přes DNS“ jsem popsal známé metody pro detekci tunelování přes HTTP a DNS, dále jsem provedl analýzu reálného provozu a provozu tunelované sítě v simulovaném prostředí. Popsal jsem rozdíly obou typů komunikací.

V části „Návrh modulu pro detekci tunelování přes DNS“ jsem vybral několik metod detekce tunelování přes DNS, popsal jsem jak metody použiji v modulu a zdůvodnil jsem jejich výběr. Popsal jsem propojení jednotlivých metod, informace potřebné pro detekci a navrhl strukturu detekčního modulu.

V části „Implementace modulu pro detekci tunelování přes DNS“ jsem nejprve diskutoval výběr datové struktury pro efektivní ukládání, vyhledávání a mazání IP adres. Z popsaných struktur jsem vybral datovou strukturu B+ strom. Datovou strukturu jsem rozšířil o několik funkcí, které snižují celkovou časovou složitost modulu. Popsal jsem konkrétní implementaci této struktury pro použití v detekčním modulu a možnosti využití v jiných aplikacích. Dále jsem diskutoval výběr datové struktury pro efektivní ukládání, vyhledávání a mazání doménových jmen. Vybral jsem prefixový strom a popsal jsem jeho potřebné úpravy pro efektivní využití v detekčním modulu. Ve zbytku kapitoly jsem se věnoval popisu konkrétní implementace detekčního modulu.

V části „Testování modulu“ jsem popsal nalezené anomálie v datech ze sítě CESNET2. Vyhodnotil jsem dobu zpracování dat a popsal úspěšnost modulu.

Modul kromě detekce tunelování přes DNS detekuje i některé ostatní DNS anomálie. Doba vykonávání díky využití efektivních datových struktur nebrání detekci anomálií v reálném čase na páteřní síti. Modul byl vyvíjený v rámci spolupráce s organizací Cesnet a bude sloužit pro detekci DNS anomálií v systému Nemea na síti CESNET2.

Literatura

[1] ACOnet. [Citováno 2014-03-30]. Dostupné z: http://www.aco.net/

[2] Akamai.net. [Citováno 2014-03-7]. Dostupné z: http://www.akamai.com/html/solutions/client_faq.html

[3] Dns2tcp. [Citováno 2014-02-26]. Dostupné z: http://www.aldeid.com/wiki/Dns2tcp

[4] DNScapy. [Citováno 2014-02-26]. Dostupné z: https://code.google.com/p/dnscapy/

[5] Dnscat. [Citováno 2014-02-26]. Dostupné z: https://wiki.skullsecurity.org/Dnscat

[6] Httptunnel. [Citováno 2014-03-12]. Dostupné z: http://www.nocrew.org/software/httptunnel.html

[7] Internet Systems Consortium, Inc. [Citováno 2014-03-06]. Dostupné z: http://www.isc.org/

[8] Iodine. [Citováno 2014-02-26]. Dostupné z: http://code.kryo.se/iodine/

[9] Nemea. [Citováno 2014-03-16]. Dostupné z: https://www.liberouter.org/technologies/nemea/

[10] Super Network Tunnel. [Citováno 2014-03-11]. Dostupné z: http://www.networktunnel.net/

[11] superuser-gstatic.com. [Citováno 2014-04-26]. Dostupné z: http://superuser.com/a/64724

[12] Tshark. [Citováno 2014-03-7]. Dostupné z: http://www.wireshark.org/docs/man-pages/tshark.html

[13] Universal HTTP Tunnel client-server. [Citováno 2014-03-12]. Dostupné z: http://www.umediaserver.net/networksoft

[14] Valgrind. [Citováno 2014-04-26]. Dostupné z: http://valgrind.org/

[15] Wireshark. [Citováno 2014-02-28]. Dostupné z: http://www.wireshark.org/

[16] Dusi, M.; Crotti, M.; Gringoli, F.; aj.: Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, ročník 53, č. 1, 2009: s. 81–97.

[17] Farnham, G.: Detecting DNS Tunneling. 2013. Dostupné z: https://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152

[18] Guy, J. J.: A study of DNS. 2009, online. Dostupné z: http://armatum.com/blog/2009/dns-part-ii/

[19] Iana: Domain Name System (DNS) Parameters. February 2014, [Citováno 2014-03-11]. Dostupné z: https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml

[20] Kambourakis, G.; Moschos, T.; Geneiatakis, D.; aj.: A fair solution to DNS amplification attacks. In *Digital Forensics and Incident Analysis, 2007. WDFIA 2007. Second International Workshop on*, IEEE, 2007, s. 38–47.

[21] Luotonen, A.: Tunneling TCP based protocols through Web proxy servers. Srpen 1998, [Citováno 2014-04-26]. Dostupné z: http://www.web-cache.com/Writings/Internet-Drafts/draft-luotonen-web-proxy-tunneling-01.txt

[22] Merlo, A.; Papaleo, G.; Veneziano, S.; aj.: A comparative performance evaluation of DNS tunneling tools. In *Computational Intelligence in Security for Information Systems*, Springer, 2011, s. 84–91.

[23] Mockapetris, P. V.: Domain names: Implementation specification. 1983.

[24] Pack, D. J.; Streilein, W.; Webster, S.; aj.: Detecting HTTP tunneling activities. Technická zpráva, DTIC Document, 2002.

[25] R Development Core Team: *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0. Dostupné z: http://www.R-project.org

[26] Tanase, M.: IP spoofing: an introduction. *Security Focus*, ročník 11, 2003.

[27] Tvrdík, P.: B-stromy a B+-stromy. [online prezentace]. 2013, [Citováno 2014-02-26]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-EFA/_media/lectures/efa2013prednaska10-btrees-handout.pdf

[28] Tvrdík, P.: Binární vyhledávací stromy a tries [online prezentace]. 2013, [Citováno 2014-03-18]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-EFA/_media/lectures/efa2013prednaska8-bvs-handout.pdf

[29] Tvrdík, P.: Datové struktury 2: Rozptylovací tabulky [online prezentace]. 2013, [Citováno 2014-02-25]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-EFA/_media/lectures/efa2013prednaska2-hash-anim.pdf

[30] Tvrdík, P.: Vyvažované BVS – AVL a RB stromy [online prezentace]. 2013, [Citováno 2014-03-18]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-EFA/_media/lectures/efa2013prednaska11-vvs-handout.pdf

[31] W3C/MIT: HTTP protocol. Červen 1999, [Citováno 2014-03-11]. Dostupné z: https://tools.ietf.org/html/rfc2616#section-9.9

[32] Walker, J.: Red Black Trees[online]. [Citováno 2014-02-26]. Dostupné z: http://www.eternallyconfuzzled.com/tuts/datastructures/jsw_tut_rbtree.aspx

[33] Wi-Free-Limited: Wired & Wireless Freedom. [Citováno 2014-02-28]. Dostupné z: https://secure.wi-free.com/

Příloha **A**

Seznam použitých zkratek

CNAME Canonical Name record

DNS Domain Name System

HTTP Hypertext Transfer Protocol

IP Internet Protocol

MTU Maximum Transmission Unit

MX Mail eXchange

NS Name Server

Příloha **B**

Pseudokódy operací B+ stromu

B.1 Vyhledání prvku podle klíče

```
function C_B_TREE_PLUS_SEARCH(tree, key)
    if (C_B_TREE_PLUS_SEARCH_RECURSIVE(key, tree.root) = −1) then
        return NULL
    end if
                                                    ▷ Vrátí hodnotu prvku
    return leaf.value[index]
end function

function C_B_TREE_PLUS_SEARCH_RECURSIVE(k, node))
     i ← 0
                        ▷ Najdi potomka pro pokračování nebo hodnotu v listu
    while (i < key.size and k < node.key[i]) do
        i ← i + 1
    end while
    if (node is leaf) then
        if (i < key.size and k = node.key[i]) then
                                                    ▷ Klíč je nalezen
            return (i, node)
        else
                                                    ▷ Klíč je nenalezen
            return (-1)
        end if
    else
        C_B_TREE_PLUS_SEARCH_RECURSIVE(k, node.child[i])
    end if
end function
```

B.2 Přidání prvku do stromu

```
function B_PLUS_TREE_INSERT_ITEM(tree, key)
    ▷ Nalezení listu pro vložení prvku leaf ← FIND_LEAF_TO_INSERT(k, node.child[i])
    if (key is in leaf) then
        return NULL
    end if
                                        ▷ Vložení prvku na správnou pozici
    INSERT_TO_LEAF(leaf, key)
    if (leaf.size ≤ MAX_M) then
        return leaf.val[index]
    end if
    new_leaf ← NEW_LEAF( )
     ▷ Zkopíruje pravou polovinu prvků a aktualizuje klíč v rodiči listu leaf
    COPY_HALF_OF_ITEMS(leaf, new_leaf)
    if (leaf.parent ≠ NULL) then
          ▷ Přidá odkaz na list i s klíčem do rodiče listu leaf pokud existuje
        ADD_TO_PARENT(leaf.parent, new_leaf)
    else
              ▷ Vytvoří nového rodiče, vloží do něj potomky a příslušné klíče
        root ← NEW_NODE( )
        ADD_TO_PARENT(root, new_leaf)
        ADD_TO_PARENT(root, leaf)
        tree.root ← root
        return leaf.val[index]
    end if
                                        ▷ Změna ukazatelů na sousední prvky
    new_leaf.right ← leaf.right
    new_leaf.left ← leaf
    leaf.right.left ← new_leaf
    leaf.right ← new_leaf
    if (leaf.parent.size ≤ MAX_M) then
        CHECK_INNER_NODE(leaf.parent)
    end if
    return leaf.val[index]
end function

function CHECK_INNER_NODE(node)
    new_node ← NEW_NODE( )
     ▷ Zkopíruje pravou polovinu klíčů a potomků, aktualizuje klíč v rodiči uzlu node
    COPY_HALF_OF_ITEMS(node, new_node)
                        ▷ Přidá odkaz na uzel i s klíčem do rodiče uzlu node
```

```
    ADD_TO_PARENT(leaf.parent, new_leaf)
    if (node.parent ≠ NULL) then
        ▷ Přidá odkaz na uzel i s klíčem do rodiče uzlu node pokud existuje
        ADD_TO_PARENT(node.parent, new_node)
    else
            ▷ Vytvoří nového rodiče, vloží do něj potomky a příslušné klíče
        root ← NEW_NODE( )
        ADD_TO_PARENT(root, new_leaf)
        ADD_TO_PARENT(root, leaf)
        tree.root ← root
        return
    end if
    if (node.parent ≠ NULL and node.parent ≤ MAX_M) then
        CHECK_INNER_NODE(leaf.parent)
    end if
    return
end function
```

B.3 Odstranění prvku ze stromu

```
function B_PLUS_TREE_DELETE_ITEM(tree, key)
     leaf ← FIND_LEAF_TO_INSERT(k, node.child[i])
    if (key is not in leaf) then
        return 0
    end if
                                                ▷ Vymaž prvek z listu
    DELETE_KEY_IN_LEAF(key, leaf)
    if (leaf.size ≥ MIN_M or leaf = tree.root) then
        return 1
    end if
    CHECK_AFTER_DELETE(tree, leaf)
end function

function CHECK_AFTER_DELETE(tree, node)
    if (GET_LEFT_BROTHER(node) ≠ NULL and left_brother.size ≥
MIN_m) then
            ▷ Vyjmout poslední prvek z levého bratra a vložit ho jako první
prvek uzlu
        MOVE_FROM_LEFT_BROTHER(left_brother, node)
                            ▷ Aktualizace hodnoty levého bratra v rodiči
        UPDATE_KEY_IN_PARENT(left_brother)
```

```
        return 1
    else if (GET_RIGHT_BROTHER(node) ≠ NULL and right_brother.size ≥
MIN_m) then
            ▷ Vyjmout první prvek z pravého bratra a vložit ho jako poslední
prvek uzlu
        MOVE_FROM_RIGHT_BROTHER(left_brother, node)
                                        ▷ Aktualizace hodnoty uzlu v rodiči
        UPDATE_KEY_IN_PARENT(node)
        return 1
    else if (GET_LEFT_BROTHER(node) ≠ NULL) then
            ▷ Sloučit levého bratra a uzel, hodnoty z levého bratra budou
nakopírovány do uzlu
        MERGE(node, left_brother)
                                                ▷ vymazat prvek v rodiči
        DELETE_NODE_IN_PARENT(left_brother)
        if (node is leaf) then
                                        ▷ Změna ukazatelů na sousední prvky
            left_brother.left.right ← node
            node.left ← left_brother.left
        end if
    else if (GET_RIGHT_BROTHER(node) ≠ NULL) then
            ▷ Sloučit pravého bratra a uzel, hodnoty z pravého bratra budou
nakopírovány do uzlu
        MERGE(node, right_brother)
                                                ▷ vymazat prvek v rodiči
        DELETE_NODE_IN_PARENT(right_brother)
                                        ▷ Aktualizace hodnoty uzlu v rodiči
        UPDATE_KEY_IN_PARENT(node)
        if (node is leaf) then
                                        ▷ Změna ukazatelů na sousední prvky
            right_brother.right.left ← node
            node.right ← right_brother.right
        end if
    end if
    if (node.parent.size = 1) then
        tree.root ← node
    else if (node.parent.size ≤ MIN_M) then
        return CHECK_AFTER_DELETE(tree, node.parent)
    end if
    return 1
end function
```

Příloha C

Pseudokódy operací prefixového stromu

C.1 Přidání domény do prefixového stromu

function PREFIX_TREE_ADD_DOMAIN(tree, string, length, char_stat)
node ← *tree.root*
▷ V cyklu projít všechny uzly, které jsou identické, pokud uzel není identický, cyklus končí.
while (COMPARE_STRING(string, node)) **do**
node ← *node.child*[*string*[*read_index*]]
end while
▷ Řetězec v uzlu nebyl přečten celý, uzel se musí rozdělit na dva. První uzel bude obsahovat identickou část s vkládanou doménou.
if (string in node was not read to the end) **then**
SPLIT(node, index_of_change)
ADD_NEW_DOMAIN(tree, node, string, read_index)
ACTUALIZE_INFORMATION(tree, domain)
return domain
▷ Uzel byl přečten celý, ale následující potomek neexistuje, musíme ho vytvořit a vložit do něj zbytek řetězce.
else if (read_index > 0 and node.child[string[read_index]] = NULL) **then**
ADD_NEW_DOMAIN(tree, node, string, read_index)
ACTUALIZE_INFORMATION(tree, domain)
return domain
▷ Uzel byl přečten celý a zároveň jsme přečetli i celý řetězec domény.
else
ACTUALIZE_INFORMATION(tree, domain)

```
        return domain
    end if
end function

function ADD_NEW_DOMAIN(tree, node, string, read_index)
                    ▷ Vytvoří nový vnitřní uzel a do něj vloží poddoménu.
    node.child[string[read_index]] ← NEW_NODE(string, read_index)
                                        ▷ Vytvoří ke vnitřnímu uzlu doménu
    node.child[string[read_index]].domain ← NEW_DOMAIN( )
    ACTUALIZE_INFORMATION(tree, domain)
            ▷ Vytvoří nový vnitřní uzel a do něj vloží zbytek řetězce domény
    ▷ Pokud v řetězci existuje další poddoména, volá tuto funkci rekurzivně
    if (read_index > 0) then
        ADD_NEW_DOMAIN(tree, node.child[string[read_index]].domain, string,
read_index)
    end if
end function
```

Příloha D

Ukládané struktury detekčního modulu

D.1 Struktury ukládané pro každou IP adresu

```
struct ip_address_t {
    /*!< version of ip */
    unsigned char ip_version;
    /*!< counter struct for requests */
    counter_request_t counter_request;
    /*!< counter struct for responses */
    counter_response_t counter_response;
    /*!< suspicion struc - finding tunnel in requests */
    ip_address_suspicion_request_tunnel_t *
        suspicion_request_tunnel;
    /*!< state of finding other anomaly in requests */
    ip_address_suspicion_request_other_t *
        suspicion_request_other;
    /*!< state of finding tunnel in response */
    ip_address_suspicion_response_tunnel_t *
        suspicion_response_tunnel;
    /*!< state of finding other anomaly in requests */
    unsigned char state_request_other;
    /*!< state of finding tunnel in requests */
    unsigned char state_request_tunnel;
    /*!< state of finding other anomaly in response */
    unsigned char state_response_other;
    /*!< state of finding tunnel in response */
    unsigned char state_response_tunnel;
} ;

struct counter_request_t {
    /*!< histogram values, requests */
    unsigned long histogram_dns_requests [HISTOGRAM_SIZE_REQUESTS
        ];
    /*!< histogram values, sum for count of used letters */
```

```
    unsigned long
        histogram_dns_request_sum_for_cout_of_used_letter [
        HISTOGRAM_SIZE_REQUESTS];
    /*!< histogram values, ex of new letters, for size.  At first
         it is sum, but on the end it has to be devided by count
        of requests */
    unsigned long histogram_dns_request_ex_sum_of_used_letter [
        HISTOGRAM_SIZE_REQUESTS];
    /*!< count of requests */
    unsigned long dns_request_count;
    /*!< count of requests string */
    unsigned long dns_request_string_count;
    /*!< Sum of sizes request */
    unsigned long sum_Xi_request;
    /*!< Sum of sizes^2 request */
    unsigned long sum_Xi2_request;
    /*!< Sum of sizes^3 request */
    unsigned long sum_Xi3_request;
    /*!< Sum of sizes^4 request */
    unsigned long sum_Xi4_request;
    /*!< count of requests without string */
    unsigned int request_without_string;
    /*!< number of round which Ip is in suspicion */
    unsigned char round_in_suspicion_request;
};

 struct counter_response_t {
    /*!< histogram values, responses */
    unsigned long histogram_dns_response [HISTOGRAM_SIZE_RESPONSE
        ];
    /*!< count of responses */
    unsigned long dns_response_count;
    /*!< Sum of sizes respone */
    unsigned long sum_Xi_response;
    /*!< Sum of sizes^2 respone */
    unsigned long sum_Xi2_response;
    /*!< Sum of sizes^3 respone */
    unsigned long sum_Xi3_response;
    /*!< Sum of sizes^4 respone */
    unsigned long sum_Xi4_response;
    /*!< number of round which Ip is in suspicion */
    unsigned char round_in_suspicion_response;
};
```

D.2 Realizace typu request tunnel

```
 struct ip_address_suspicion_request_tunnel_t {
    /*!< state, for every size to store in prefix tree */
    unsigned char  state_request_size [HISTOGRAM_SIZE_REQUESTS];
    /*!< pointer to prefix tree */
    prefix_tree_t * tunnel_suspicion;
    /*!< count of round in SUSPICION state */
    unsigned int round_in_suspicion;
```

```
} ;
```

D.3 Realizace typu request other anomaly

```
struct ip_address_suspicion_request_other_t {
   /*!< state, for every size to store in prefix tree */
   unsigned char  state_request_size [HISTOGRAM_SIZE_REQUESTS];
   /*!< pointer to prefix tree */
   prefix_tree_t * other_suspicion;
   /*!< count of round in SUSPICION state */
   unsigned int round_in_suspicion;
} ;
```

D.4 Realizace typu response tunnel

```
struct ip_address_suspicion_response_tunnel_t {
   /*!< pointers to prefix trees */
   prefix_tree_t * txt_suspicion;
   prefix_tree_t * cname_suspicion;
   prefix_tree_t * mx_suspicion;
   prefix_tree_t * ns_suspicion;
   prefix_tree_t * request_suspicion;
   /*!< records to store */
   unsigned char state_type;
   /*!< count of round in SUSPICION state */
   unsigned int round_in_suspicion;
} ;
```

D.5 Realizace typu response other anomaly

```
struct ip_address_suspicion_response_other_t {
   /*!< state, for every size to store in prefix tree */
   unsigned char  state_response_size [HISTOGRAM_SIZE_RESPONSE];
   /*!< pointer to prefix tree */
   prefix_tree_t * other_suspicion;
   /*!< count of round in SUSPICION state */
   unsigned int round_in_suspicion;
   /*!< count of response without request string */
   unsigned int without_string;
   /*!< count of responses in suspicion */
   unsigned int packet_in_suspicion;
} ;
```

Příloha E

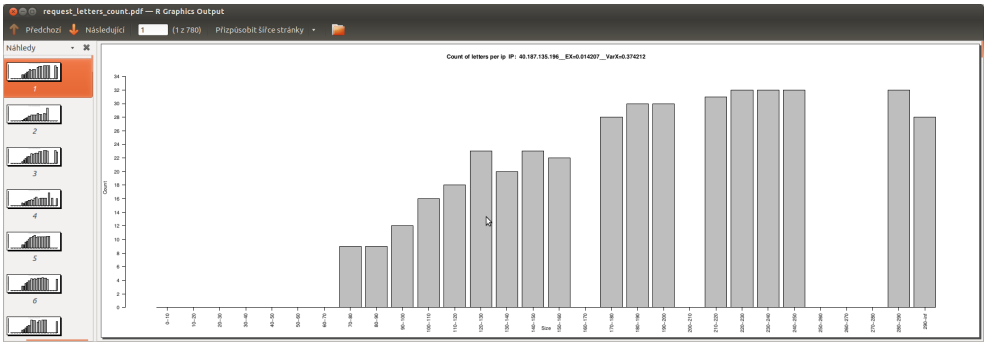
Testy anomálií

```
                                                    ▷ request other anomaly
if (item.state_request_other = STATE_SUSPICION) then
    if (IS_PAYLOAD_ON_IP_OK_REQUEST_OTHER(item)) then
        item.state_request_other ← STATE_ATTACK;
    else
        item.state_request_other ← STATE_OK;
    end if
else if (item.state_request_other = STATE_NEW) then
    if (IS_TRAFFIC_ON_IP_OK_REQUEST_OTHER(item)) then
        item.state_request_other ← STATE_SUSPICION;
    else
        item.state_request_other ← STATE_OK;
    end if
end if
                                                    ▷ request tunnel anomaly
if (item.state_request_tunnel = STATE_SUSPICION) then
    if (IS_PAYLOAD_ON_IP_OK_REQUEST_TUNNEL(item)) then
        item.state_request_tunnel ← STATE_ATTACK;
    else
        item.state_request_tunnel ← STATE_OK;
    end if
else if (item.state_request_tunnel = STATE_NEW) then
    if (IS_TRAFFIC_ON_IP_OK_REQUEST_TUNNEL(item)) then
        item.state_request_tunnel ← STATE_SUSPICION;
    else
        item.state_request_tunnel ← STATE_OK;
    end if
end if
                                                    ▷ response payload, tunnel anomaly
```

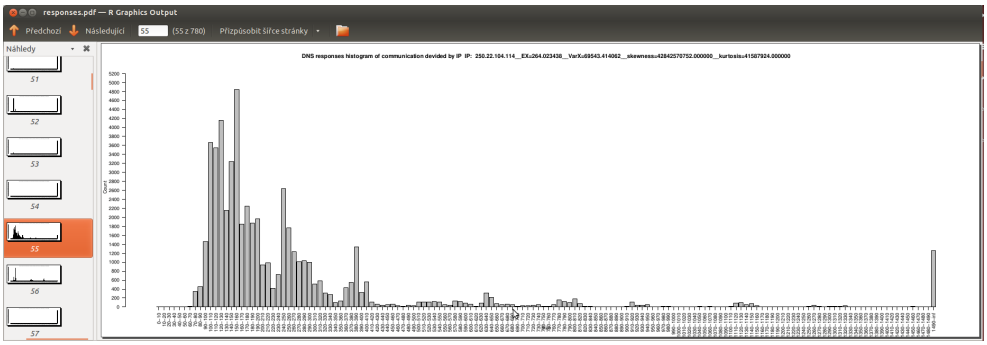
```
if (item.state_response_tunnel = STATE_NEW) then
    if (IS_PAYLOAD_ON_IP_OK_RESPONSE_TUNNEL(item)) then
        item.state_response_tunnel ← STATE_ATTACK;
    else
        item.state_response_tunnel ← STATE_OK;
    end if
end if
                                          ▷ response traffic, other anomaly
if (item.state_response_other = STATE_SUSPICION) then
    if (IS_PAYLOAD_ON_IP_OK_RESPONSE_OTHER(item)) then
        item.state_response_other ← STATE_ATTACK;
    else
        item.state_response_other ← STATE_OK;
    end if
else if (item.state_response_other = STATE_NEW) then
    if (IS_TRAFFIC_ON_IP_OK_RESPONSE_OTHER(item)) then
        item.state_response_other ← STATE_SUSPICION;
    else
        item.state_response_other ← STATE_OK;
    end if
end if
```

Příloha F

Ukázka grafického výstupu



Obrázek F.1: Histogram unikátních písmen v závislosti na velikosti paketu.



Obrázek F.2: Histogram počtu odpovědí v závislosti na velikosti paketu.

Příloha G

Instalační manuál

Instalační manuál byl sepsán a otestován na Linux Ubuntu 13.10. Podobnou instalaci je možné provést na některé jiné distribuci, na které je nainstalováno gcc.

Pro zkompilování a spuštění modulu je třeba doinstalovat knihovny Libtrap a UniRec dostupné z [9].

```
$ tar -zxvf tunnel_detection_dns-1.0.0.tar.gz -C [cílové umístění]
$ cd [cílové umístění]
$ ./configure
$ make install
```

G.1 Spuštění modulu

Modul spustíme zavoláním příkazu *tunnel_detection_dns*. Pro vypsání nápovědy použijeme přepínač *-h*.

G.2 Spuštění skriptu na převod PCAP do textové podoby

Na přiloženém CD ve složce *scripts* se nachází skript *data_parser.sh*. Pro spuštění skriptu potřebujeme program Tshark [12] (Otestováno na verzi 1.6.3).

```
$ ./data_parser.sh  [cesta a jméno vstupního souboru] [index prvního
    souboru] [index posledního souboru] [cesta a jméno výstupního textového
    souboru]
```

G.3 Spuštění skriptu pro vytvoření grafů z výsledných dat

Na přiloženém CD ve složce *scripts* se nachází skript *create_graphs.sh*. Pro spuštění skriptu potřebujeme program R [25] (Otestováno na verzi 2.14.1). Výsledné grafy se zapíší ve vstupním adresáři do adresáře *graphs*.

```
$ ./create\_graphs.sh  [cesta k adresáři s výsledky]
```

Příloha **H**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `tunnel_detection_dns-1.0.tar.gz` .... balík obsahující zdrojové kódy implementace modulu
- `doc` ..... dokumentace zdrojových kódů
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `scripts` ..... skripty
  - `data_parser.sh` ..... skript pro převod PCAP do textové podoby
  - `create_graphs.sh` ..... skript pro vytvoření grafů
  - `R` ..... rozšiřující skripty využívající program R