VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

NALEZENÍ VZORU ÚTOKU Z DAT NETFLOW

FINDING ATTACK PATTERN FROM NETFLOW DATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MICHAL JIREŠ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MATĚJ GRÉGR, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

147023

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Jireš Michal, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Počítačové sítě
Název:	**Nalezení vzoru útoku z dat NetFlow**
Kategorie:	Počítačové sítě
Akademický rok:	2022/23

Zadání:

1. Seznamte se s technologií NetFlow pro monitorování síťového provozu.
2. Prostudujte knihovnu libnf pro práci s daty NetFlow.
3. Navrhněte systém, který dokáže detekovat probíhající útok a nalézt vzor útoku pro co nejefektivnější filtraci. Zaměřte se na útoky typu DDoS.
4. Navržený systém implementujte a nasaďte v testovacím prostředí VUT.
5. Vyhodnoďte dosažené výsledky.

Literatura:

- Claise B., "Cisco Systems NetFlow Services Export Version 9", RFC 3954, DOI 10.17487/RFC3954, October 2004, https://www.rfc-editor.org/info/rfc3954>
- Hofstede R. et al., "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX," in IEEE Communications Surveys & Tutorials, vol. 16, no. 4, pp. 2037-2064, Fourthquarter 2014, doi: 10.1109/COMST.2014.2321898.
- Mirkovic J., and Reiher P. "A taxonomy of DDoS attack and DDoS defense mechanisms." *ACM SIGCOMM Computer Communication Review* 34.2 (2004): 39-53.
- Zargar S. T., Joshi J. and Tipper D. "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks." *IEEE communications surveys & tutorials* 15.4 (2013): 2046-2069.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Grégr Matěj, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	26.10.2022

Abstrakt

Tato práce se zabývá detekcí možných útoků v síti a následného vygenerování co nejpřesnějších pravidel pro filtraci nalezeného útoku. Cílem práce je vytvořit nástroj, který bude analyzovat příchozí NetFlow data se snahou detekovat právě probíhající útoky. Pokud program detekuje probíhající útok vygeneruje na základě NetFlow dat z blízké minulosti co nejpřesnější pravidlo pro možnou mitigaci.

Abstract

This thesis deals with detection of possible attacks in computer networks and subsequent generation of the most specific rules used for filtration. The goal of this thesis is to create tool, that will analyze incoming NetFlow data and will try to detect ongoing attacks. If there is ongoing attack detected, the tool will generate the most specific rules, based on recent NetFlow history, that can be used to filter packets associated with the attack.

Klíčová slova

NetFlow, DDoS, nalezení vzoru útoku, ochrana proti DDoS útoku, filtrační pravidla, Libnf, nfdump.

Keywords

NetFlow, DDoS, finding attack pattern, DDoS guard, filter rules, Libnf, nfdump.

Citace

JIREŠ, Michal. *Nalezení vzoru útoku z dat NetFlow.* Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Matěj Grégr, Ph.D.

Nalezení vzoru útoku z dat NetFlow

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Matěje Grégra a uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Jireš
15. května 2023

Poděkování

Rád bych poděkoval vedoucímu práce, panu Ing. Matěju Grégrovi, Ph.D., za cenné rady a poznatky.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Cílem této práce je vytvořit automatizovanou detekci síťových útoků a nalezení vzoru těchto útoků. K získání informací o provozu v síti, bude využit protokol NetFlow. Vytvořený program bude analyzovat příchozí NetFlow data se snahou nalézt anomálie. Pro regulaci, které NetFlow data se použijí k analýze, bude mít síťový administrátor možnost použít filtrační pravidla. Příkladem filtračního pravidla může být například specifické číslo portu transportního protokolu. Použitím těchto pravidel administrátor efektivně zúží rozsah analýzy, pouze na chtěnou část provozu v síti. Program při detekci útoku vytvoří co nejefektivnější filtrační pravidla, které by mohlo být použito pro mitigaci útoku. Aby byla pravidla co nejpřesnější využijí se NetFlow data i z nedávné minulosti. Další velkou částí práce je nasazení tohoto nástroje v testovacím prostředí VUT a následné testování.

Práce je strukturovaná takto: Kapitola 2 obsahuje seznámení s technologií NetFlow a její architekturou. Následující kapitola 3 popisuje získávání NetFlow dat a nástroje pro práci s těmito daty. Úvodní informace o DDoS útocích a rozdělení DDoS útoků popisuje kapitola 4. Kapitola 5 stručně vytyčuje rozsah a možnosti programu ve formě návrhu. Pokračuje kapitola 6, která podrobně popisuje implementaci navrhovaného programu. Kapitola 7 povídá o testování správnosti implementovaného programu. Na konci psané části práce je kapitola 8 obsahující zhodnocení učiněných kroků a jejich výsledků. Na úplném konci práce je vložena příloha A s informacemi o přiloženém paměťovém médiu.

Kapitola 2

Toky v síti

Každému administrátorovi sítě je užitečné, až nezbytné, zkoumat provoz sítě. Toto zkoumání může být potřebné nejen pro detekci anomálií, jak bude ukázáno v této práci, ale také pro získávání zajímavých statistik o množství, původu, cíli a cestách dat v různých částech sítě.

Toto zkoumání je usnadněno tím, že si je provoz v síti možné představit jako řadu toků (*Flow*) procházejících sítí. Jeden tento tok je definován jako sekvence paketů mající společnou vlastnost a procházející bodem pozorování[8]. Typicky se o toku ukládají záznamy obsahující informace o zdrojové a cílové IP adrese, zdrojovém a cílovém portu, typu protokolu, data a času zaznamenání a počtu přenesených dat. Na obrázku 2.1 je možné vidět data shromážděná a zobrazená pomocí nástroje nfdump, používaného v této práci.

Date first seen	Duration	Proto	Src IP Addr:Port		Dst IP Addr:Port	Flags	Tos	Packets	Bytes	Flows
2021-10-28 16:10:57.998	0.000	ICMP	192.168.100.3:0	->	10.0.222.101:0.0	.....	0	1	92	1
2021-10-28 16:10:57.998	0.000	UDP	172.16.1.105:54227	->	10.1.1.1:53	.....	0	1	92	1
2021-10-28 16:10:56.828	1.170	TCP	192.168.1.21:20462	->	10.0.100.5:3389	...AP...	0	10	1888	1

Obrázek 2.1: Ukázka informací ukládaných o tocích.

Ukládání toků lze také použít pro splnění požadavků na sběr provozních a lokalizačních údajů, neboli tzv. „Data Retention“, v České republice dle vyhlášky č. 357/2012 Sb.[34].

2.1 NetFlow

NetFlow je technologie vyvinutá firmou Cisco pro účely monitorování provozu v síti. Původně proprietární, poté standardizované autoritou IETF a popsané v RFC 3954[8]. Díky tomu je NetFlow implementované nejen na Cisco platformách. NetFlow popisuje architekturu a komunikační protokol, pro vytváření, kolekci a zobrazování toků v síti.

2.1.1 Architektura NetFlow

Architektura je tvořená těmito prvky:

- **Exportér** - Získává statistiky toků procházejících skrz.
- **Přenosový protokol** - Protokol pro přenos záznamů mezi exportérem a kolektorem.
- **Kolektor** - Ukládá záznamy o tocích.

2.1.2 Exportér

Exportér je síťový prvek, který generuje záznamy o tocích (*Flow Records*) na základě provozu jdoucím skrz tento prvek. Tyto statistiky jsou ukládány do vnitřní paměti prvku (*NetFlow Cache*) a následně pomocí přenosového protokolu (viz. 2.1.3) poslány do kolektoru (viz. 2.1.4) k možné analýze.

V tradiční Cisco architektuře je za exportér považován aktivní síťový prvek (router nebo L3 switch), který je hlavně zodpovědný za směrování. Generování záznamů o tocích je pouze nadstavbou činnosti prvku. Tento přístup má hlavní nevýhodu v tom, že samotné generování záznamů je výpočetně náročné a může tak ovlivnit i směrovací výkon. Levnější prvky kvůli tomu využívají tzv. vzorkování, díky kterému se při generování toků nebere v potaz každý paket, ale mohou využívat každý n-tý nebo vybírat pakety náhodně.

Dalším přístupem je využití pasivních NetFlow sond. Tyto sondy jsou zařízení specializovaná pro generování a export záznamů o tocích. Na rozdíl od tradiční architektury zmíněné výše se sondy nepodílí na směrování (proto pasivní). Sondy jsou tedy zcela transparentní vůči provozu.

Exportéry umožňují filtrování, díky kterému je na základě hodnot v hlavičce paketů možné určit, které pakety budou použity pro generování záznamů o tocích. Filtrování je možné použít zároveň se vzorkováním zmíněným výše. Toto použití vede k tomu, že si síťový administrátor může určit třídy provozu, pro které se budou vybírat pakety pro generování odlišně. Například si může vytvořit třídu s nejvyšší prioritou, u které se budou generovat záznamy z každého paketu a druhou třídu s nižší prioritou, u které se využije vzorkování.

Export záznamů o tocích

Existuje více scénářů, kdy bude exportér odesílat vygenerovaný záznam o toku uložený ve vnitřní paměti exportéru. Tyto scénáře jsou:

- **Neaktivita toku** - Překročení časového limitu neaktivity.
- **Příliš dlouhý tok** - Překročení časového limitu aktivity.
- **Detekce konce toku** - Když je zřejmé, že byl přijat poslední paket toku (např. TCP FIN).
- **Interní omezení** - Např. zaplnění paměti exportéru nebo přetečení čítačů.

Exportér si pro každý tok uchovává dva časovače, aktivní a neaktivní. Aktivní časovač značí čas od přijetí prvního paketu toku nebo od posledního exportu tohoto toku. Neaktivní časovač značí čas, jakou dobu je tok neaktivní, tedy čas od přijetí posledního paketu toku.

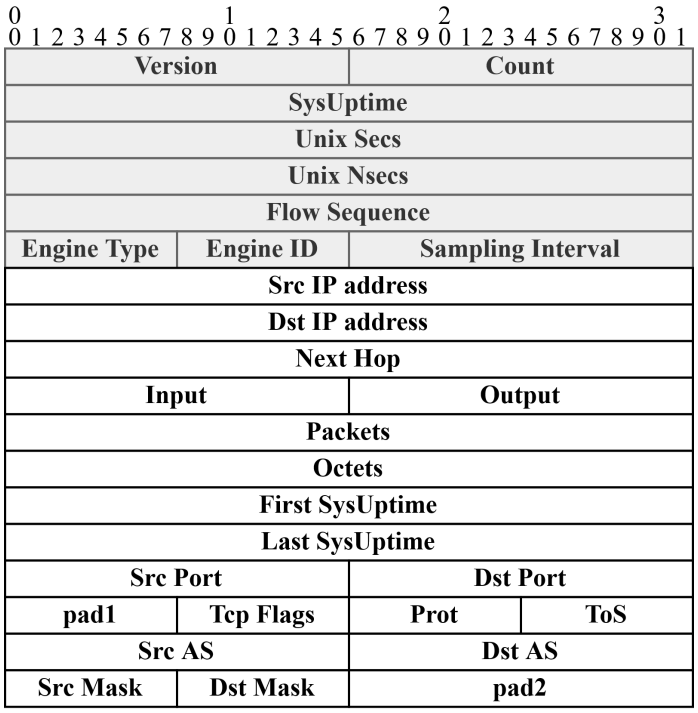
2.1.3 Přenosový protokol

Přenosový protokol NetFlow zajišťuje možnost exportéru efektivně odeslat své záznamy o tocích kolektoru. Tento protokol má několik verzí, kde mezi nejvýznamnější patří v5 a v9.

NetFlow v5

Paket této verze[5] je možné vidět na obrázku 2.2. Paket je rozdělen na hlavičku (šedě) a tělo (bíle). Velikosti jednotlivých polí je možné odvodit z čísel znázorňujících počet bitů.

Hlavička obsahuje informace o počtu záznamů NetFlow (*Count*), čas odeslání tohoto paketu (*Unix time*) a identifikaci exportéru (*Engine type, Engine ID*). Tělo paketu obsahuje informace o samotném toku. Mezi odeslané informace patří zdrojová a cílová IP adresa (*Src IP address, Dst IP address*), zdrojový a cílový port (*Src Port, Dst Port*), počet paketů v toku (*Packets*), celkový počet bajtů v toku (*Octets*), typ transportního protokolu (*Prot*) a type of service (*ToS*).



Obrázek 2.2: Ukázka paketu NetFlow v5.

Z paketu NetFlow v5 jsou zřejmé některé nevýhody/nedostatky. Jedním z hlavních nedostatků je fakt, že pole *Src IP address* a *Dst IP address* jsou pevné velikosti, což znamená, že tato verze dokáže zasílat pouze informace o IPv4 tocích. Paket má také pouze dvě nevyužitá místa (*pad1, pad2*), neumožňuje tedy žádné veliké rozšiřování o nové přenášené hodnoty.

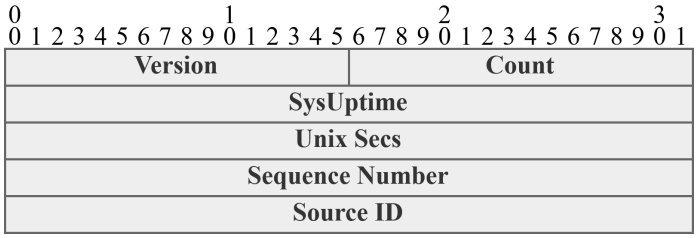
NetFlow v9

Verze protokolu NetFlow v9[8] je navržena zásadně jinak, než verze 5. Využívá se zde šablon a obecného formátu TLV (*Type-Length-Value*), které umožňují mnohem snazší rozšiřitelnost a větší flexibilitu. Díky tomu se možné přenášené hodnoty rozšířily o další položky, kde mezi hlavní patří např. VLAN, IPv6, MPLS, Multicast nebo čísla AS (*Autonomous System*).

Paket NetFlow verze 9 zažil zásadní změny. Výjimkou je hlavička paketu, která se příliš neliší od starší verze. Došlo pouze k přesunu některých hodnot z hlavičky do těla paketu. Ukázku hlavičky lze vidět na obrázku 2.3.

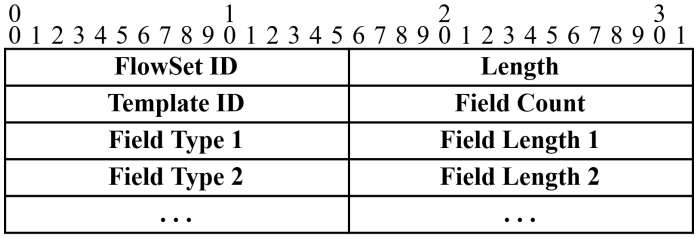
Změny jsou na první pohled viditelné v těle paketu. Došlo zde k vytvoření více možných formátů. Definované formáty jsou:

- **Template FlowSet** - Formát je možné vidět na obrázku 2.4. Tato šablona obsahuje informace o typu (*Field Type N*) a délce (*Field Length N*) v bajtech, pro jednotlivé



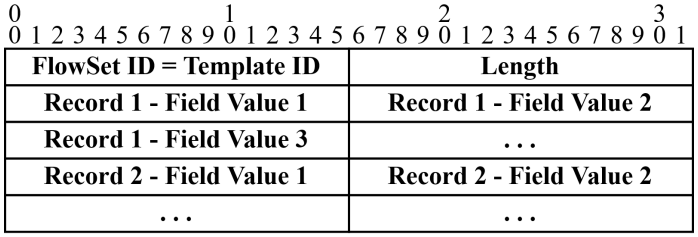
Obrázek 2.3: Ukázka hlavičky paketu NetFlow v9.

hodnoty obsažené v **Data Flowset**. Do jednoho těla paketu, lze těchto šablon vložit více za sebe. Při tomto vložení mají šablony stejné *FlowSet ID* a liší se v *Template ID*.



Obrázek 2.4: Ukázka NetFlow v9 formátu Template FlowSet.

- **Data FlowSet** - Formát je možné vidět na obrázku 2.5. Kolektor nemůže interpretovat data, pokud dříve nepřijal příslušný **Template FlowSet**, což pozná na základě stejných ID těchto flowsetů (*Template ID*). Tento formát obsahuje již samotné užitečné hodnoty (*Record N - Field Value M*) pro záznamy popsané v jeho **Template FlowSetu**. Pro každý záznam (*Field Type N*) může obsahovat sadu hodnot (*Record N - Field Value 1, Record N - Field Value 2, ...*). Tyto hodnoty popisují informace o jednotlivých tocích.



Obrázek 2.5: Ukázka NetFlow v9 formátu Data FlowSet.

- **Options Template FlowSet** - Formát je možné vidět na obrázku 2.6. Tato šablona obsahuje informace o typu (*Scope N Field Type / Option N Field Type*) a délce (*Scope N Field Length / Option N Field Length*) v bajtech, pro jednotlivé hodnoty obsažené v **Options Data FlowSet**. Na rozdíl od flowsetů zmíněných výše options flowsety neposílají informace o tocích, ale o NetFlow procesu samotném. Scope typ obsahuje informaci o tom, ke které části procesu NetFlow náleží další přijaté options fieldy. Scope typ nabývá hodnot např. *System*, *Line Card*, atd...

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1	
FlowSet ID	**Length**
Template ID	**Option Scope Length**
Option Length	**Scope 1 Field Type**
Scope 1 Field Length	**. . .**
Scope N Field Length	**Option 1 Field Type**
Option 1 Field Length	**. . .**
. . .	**. . .**

Obrázek 2.6: Ukázka NetFlow v9 formátu Options Template FlowSet.

- **Options Data Record** - Formát je možné vidět na obrázku 2.7. Každý **Options Data Record** má svojí šablonu **Options Template FlowSet**, která popisuje význam jeho hodnot. Příslušná šablona se pozná na základě stejného ID (*Template ID*). Může obsahovat několik scope hodnot (*Record N - Scope M Value*), určujících kontext pro další přijaté záznamy (*Record N - Option Field K Value*).

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1	
FlowSet ID = Template ID	**Length**
Record 1 - Scope 1 Value	**Record 1 - Option Field 1 Val**
Record 1 - Option Field 2 Val	**. . .**
Record 2 - Scope 1 Value	**Record 2 - Option Field 1 Val**
Record 2 - Option Field 2 Val	**. . .**
. . .	**. . .**

Obrázek 2.7: Ukázka NetFlow v9 formátu Options Data Record.

Existuje tabulka[6][14] popisující všechny možné použitelné typy (*Field Type*) zmíněné výše. Tato tabulka se do budoucna může zvětšovat, což umožňuje onu snadnou rozšiřitelnost. Přidání nového typu pak znamená pouze aktualizování tabulek na exportéru a kolektoru a případná reoptimalizace ukládaných struktur na kolektoru.

Flexibilita těla paketu je pak zajištěna tím, že formáty zmíněné výše lze libovolně skládat za sebe do jednoho paketu. Aby bylo možné tyto formáty od sebe rozeznat jsou předepsané rozsahy ID (*FlowSet ID*), díky kterým je snadné zjistit, jaký formát následuje. Každý z těchto formátů také obsahuje pole s hodnotou jeho délky (*Length*), takže je jasné, kde tento formát končí a další může začínat.

NetFlow verze 9 byl použit jako základ při tvorbě nového IETF protokolu s názvem IPFIX[1].

2.1.4 Kolektor

Zařízení určené pro efektivní přijímání a ukládání záznamů o tocích. Jeden kolektor může přijímat záznamy z jednoho či více exportérů. Jako implementace kolektoru existuje několik technologií. Tyto technologie obecně zařizují určité dotazování nad NetFlow daty a následné zobrazování výsledků. V této práci je jako technologie kolektoru použit nástroj nfdump. Použití tohoto nástroje je popsáno níže v kapitole 3.1.

Kapitola 3

Získávání NetFlow dat

V této sekci jsou popsány technologie použité k efektivnímu shromažďování a načítání NetFlow dat.

3.1 Nfdump

Nfdump[13] je sada nástrojů určená pro shromažďování, ukládání a zpracování NetFlow dat. Je distribuován pod BSD licencí. Podporuje protokol IPv4 i IPv6, NetFlow verze 1,5/7,9 a IPFIX. Základní sada se skládá konkrétně z těchto nástrojů:

- **nfcapd** - Daemon pro shromáždění a ukládání NetFlow dat přijatých od exportérů.
- **nfdump** - Nástroj pro zpracování již uložených NetFlow dat s podporou filtrování a agregace.
- **nfanon** - Nástroj pro anonymizování IP adres uložených NetFlow dat.
- **nfexpire** - Obstarává expiraci NetFlow dat.
- **nfreplay** - Umožňuje již uložená NetFlow data poslat dalšímu zařízení.

Snaha této sady je umožnit analýzu NetFlow dat z minulosti a pozorování nějakého vzoru chování provozu. Přijatá NetFlow data se ukládají na disk do souborů podle času přijetí těchto dat. Jeden soubor bývá typicky určen pro časové rozmezí pěti minut. Časové rozmezí souboru je zřejmé přímo z jeho názvu, který je ve formátu: *nfcapd.YYYYMMddhhmm*. Data jsou v souborech uloženy v binárním formátu optimalizovaném k vyhledávání.

Jak daleko do minulosti jsou data uchovávána záleží na velikosti volného místa na disku. Pro účely efektivnějšího využití místa na disku umožňuje nfdump kompresi uložených NetFlow dat. Komprese může být provedena třemi různými algoritmy *LZO1X-1*, *LZ4* anebo *bzip2*. Ke kompresi dat přímo při jejich přijetí se doporučují rychlé algoritmy *LZO1X-1* a *LZ4*. Algoritmus *bzip2* je pomalejší a značně výkonově náročnější, ale je nejvíce efektivní, doporučuje se tedy při archivaci NetFlow dat.

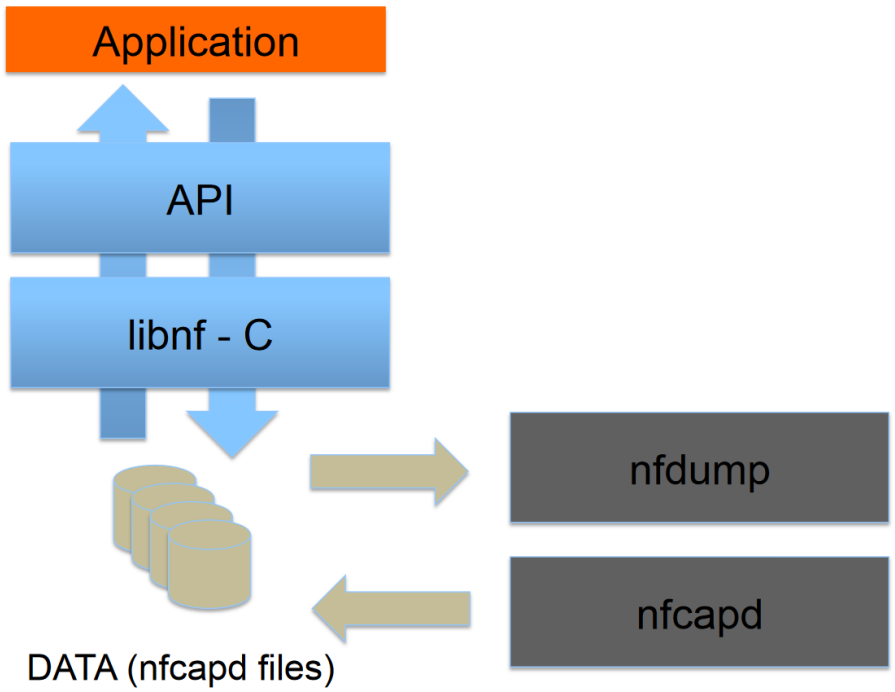
3.2 Libnf

Při tvorbě aplikací pracujících s NetFlow daty za použití kolektoru nfdump, je prvním krokem nastavení nfcapd daemona pro shromažďování těchto NetFlow dat. Vzniká ale otázka, jak k těmto datům přistupovat.

Prvním způsobem je dotazování nad daty pomocí nástroje nfdump. Těmito dotazy dostaneme textový řetězec, který je potřeba parsovat pro zisk užitečných dat. Tento přístup je příliš pomalý.

Druhým způsobem je přistupování přímo k souborům vytvořeným nfcapd daemonem. Tento přístup je mnohem efektivnější. Problém ovšem může nastat při aktualizaci sady nástrojů nfdump, může se totiž stát, že nová verze změní formát ukládání dat. V tomto případě je pak nutné aplikaci, která přistupovala přímo k souborům, předělat, což může být velice časově náročné, zvlášť pokud se takových aplikací provozuje více.

Proto vnikl návrh na třetí způsob získání NetFlow dat a to pomocí knihovny Libnf[21] a s ní spojeného API (*Application Programming Interface*). Aplikace napsané s využitím tohoto API mají výhodu v tom, že v případě aktualizace sady nástroje nfdump a změny formátu uložených dat, není nutné všechny tyto aplikace předělávat a stačí upravit pouze část libnf, která přistupuje k souborům. Vizualizace tohoto přístupu je na obrázku 3.1.



Obrázek 3.1: Ukázka vztahu Libnf a Nfdump. (Převzato z materiálů Libnf)

3.2.1 Vlastnosti knihovny

Knihovna libnf vznikla jako součást projektu VG20102015022 podporovaného Ministerstvem vnitra České republiky na Vysokém učení technickém v Brně, Fakultě informačních technologií.

Při implementaci[27] byly použity zdrojové soubory samotného nfdump. Díky tomu může libnf používat stejné funkce jako nfdump. Toto vede ke snížení rizik přivádění nových chyb do kódu a jednoduší úpravě při aktualizaci nfdump. Tento přístup je umožněn faktem, že zdrojové soubory nástroje nfdump jsou naprogramované srozumitelným způsobem. Přináší ale také určité limitace, některé funkce nástroje používají globální proměnné, což nijak neovlivňuje funkcionalitu samotného nástroje, jelikož při spuštění několika instancí

dochází k jejich izolaci operačním systémem. V případě knihovny můžou nastat potíže. Je umožněno pomocí objektu vytvořit více instancí knihovny v jednom procesu a je zde očekáváno, že více instancí nfdump bude pracovat nezávisle, což použití globálních proměnných znemožňuje.

Dobrou zprávou je, že funkce nástroje, které používají globální proměnné jsou operace jako agregace a třídění dat. Zato základní operace jsou implementovány funkcemi, které používají striktně pouze své parametry. Základní operace jako otevírání a zavírání datových souborů, čtení a zapisování záznamů. Stále to ale vede k faktu, že knihovna podporuje pouze základní operace, jelikož implementace složitějších operací jako agregace a třídění by znamenala změnu zdrojových souborů samotného nfdump, což by značně zkomplikovalo přechody na novější verze nfdump. Pokud chce aplikace používat tyto složitější operace, musí si je implementovat sama nad knihovnou.

Architektura libnf, lze rozdělit na tři části. Tyto části je možné vidět na obrázku 3.1. Zde jsou stručně popsány role těchto částí:

- **API** - Rozhraní pro aplikaci, která pracuje s NetFlow daty. Byla zde snaha o co nejlepší návrh, aby se minimalizovala potřeba aktualizace **API**, a tím i potřeba úprav samotné aplikace.
- **libnf - C** - Kód, který převádí vnitřní nfdump struktury do vyšších jazyků a umožnit **API** přístup k funkcím, které jsou podporované ve zdrojových souborech nfdump.
- **nfcapd soubory** - Soubory s uloženými NetFlow daty v originálním formátu sady nfdump.

3.2.2 Optimalizace při používání knihovny

Při tvorbě knihovny nebyl hlavním cílem maximální výkon. Je ale možné, za použití několika zásad, značně tento výkon vylepšit pro některé operace.

První optimalizací je použití korektní metody pro načítání a ukládání záznamů. Porovnání mezi metodami je možné vidět v tabulce 3.1. Nejrychlejší z metod jsou metody, které používají reprezentaci záznamu jako referenci na pole (**_arrayref*), což je nejspíše způsobené faktem, že je knihovna napsaná v jazyce C, který je velice efektivní v používání ukazatelů (*pointer*). Rozdílem mezi reprezentací záznamu jako reference na pole a jako pole je v tom, že při použití reprezentace jako pole musí každá funkce navíc alokovat paměť pro uložení onoho pole jako lokální proměnnou.

	fetchrow_*	storerow_*
*_arrayref	384 615 záznamů/s	689 655 záznamů/s
*_array	297 345 záznamů/s	455 616 záznamů/s
*_hashref	50 847 záznamů/s	166 666 záznamů/s

Tabulka 3.1: Porovnání použití různých metod pro načítání a ukládání záznamů. Při použití API v jazyce perl. (Převzato z materiálů libnf)

Druhou optimalizací je přenášení co nejméně dat mezi libnf a aplikací. Toto může být dosaženo specifikováním pouze určitých částí záznamů, které se budou přenášet. Například je možné specifikovat, že se budou přenášet pouze informace o zdrojové a cílové IP adrese (*srcip, dstip*).

3.2.3 Použití pro rozšiřování informací NetFlow dat

Jak je zmíněno v předchozí kapitole 2.1.3, novější verze NetFlow podporují zasílání informací o čísle zdrojového a cílového AS. V reálných použitích se tyto informace zasílají velmi zřídka. Může to být způsobeno umístěním exportéru v místě, kde nemá přístup ke globální BGP tabulce.

Knihovna si při tomto použití stáhne globální BGP tabulku. Tato tabulka je k dispozici ke stažení díky společnosti RIPE[29], která má celosvětově rozšířené kolektory a pravidelně ukládá momentální podoby BGP tabulky. Díky této tabulce může knihovna provádět vyhledávání na základě co nejdelší shody IP adresy (*longest prefix match*) s očekávaným výsledkem čísel patřičných AS.

Podobný princip lze využít i při přidávání zdrojového a cílového kódů země. V tomto případě je použita databáze MaxMind[19].

Kapitola 4

DDoS útoky

DDoS (*Distributed Denial of Service*) je typ útoku jehož cílem je znepřístupnit a nebo omezit funkčnost nějaké služby. Na rozdíl od DoS (*Denial of Service*) je DDoS prováděn za pomoci většího počtu zařízení, které mohou být propojeny a spravovány.

DDoS útoky jsou obecně prováděny v několika fázích[20]. Útočník se nejprve snaží najít co největší počet zranitelných zařízení, u kterých může využít bezpečnostních děr pro zavedení svého kódu. Další fází je samotný útok, kde útočník využije již nakažené zařízení k zasílání paketů na jednotný cíl.

Na rozdíl od mnoha jiných útoků, které se snaží získat neoprávněný přístup do systému za účelem získání dat, je cílem DDoS útoků způsobit oběti škodu. DDoS útoky mají často postranní motiv ať už to jsou osobní záležitosti (osobní pomsta), snaha o získání nepřímého zisku (poškození konkurence) a nebo politické motivy.

4.1 Typy DDoS útoků

DDoS útoky lze rozdělit do několika základních typů[10]. Jednotlivé typy se obecně liší v použitém protokolu pro útok. Tyto typy lze také rozdělit do dvou skupin[30] a to útoky pomocí záplav (*flooding attacks*) a logické útoky (*logical attacks*).

Útoky pomocí záplavy (*flooding attacks*) se snaží oběť zaplavit co největším počtem paketů a způsobit přetížení např. komunikační linky, nebo procesoru či paměti zařízení oběti. Mezi nejznámější útoky této skupiny patří:

- **SYN flood** - Tento útok využívá navazování spojení u protokolu TCP. U navazování spojení se využívá 3-way handshake (SYN, SYN/ACK, ACK). Útočník pošle velké množství SYN paketů s pozměněnou zdrojovou IP adresou na zařízení oběti. Zařízení oběti reaguje na každý SYN paket alokací systémových zdrojů a odeslání SYN/ACK paketu. Jelikož zařízení oběti nikdy nedostane odpověď ACK, alokované zdroje po čase uvolní. Cílem útočníka je tedy poslat dostatečné množství SYN paketů pro přetížení zařízení oběti, což vede k následnému znepřístupnění tohoto zařízení legitimním uživatelům.
- **UDP flood** - Útočník posílá velké množství UDP paketů s náhodným číslem portu na zařízení oběti. Zařízení oběti při přijetí UDP paketu musí zkontrolovat zda některá aplikace naslouchá na zadaném portu, pokud žádnou aplikaci nenajde zařízení vygeneruje ICMP paket se zprávou *Destination unreachable*. Dostatečné množství paketu vede k přetížení zařízení oběti.

- **ICMP flood** - Útočník posílá ICMP dotaz (*echo request*) s pozměněnou zdrojovou adresou na IP adresu oběti. Tento paket pošle na nějakou multicastovou adresu. Toto vede k tomu, že každé zařízení, které naslouchá na této multicastové adrese odešle ICMP odpověď (*echo response*) na IP adresu oběti. Při dostatečném množství zařízení odesílajícím odpověď může vést k přetížení zařízení oběti.

- **Reflexní útoky** - Při tomto útoku využije útočník nějakou legitimní službu, která splňuje požadavky útočníka, například DNS. Tyto požadavky jsou, že služba odpovídá na každý paket a odpověď této služby je obecně několikanásobně větší než dotaz útočníka. Útočník poté posílá dotazy s pozměněnou zdrojovou IP adresou na IP adresu oběti. Legitimní služba poté posílá odpověď na každý dotaz na IP adresu oběti. Například u služby DNS je možné se dostat na poměr přibližně 1:6, kde na každý 1 bajt dotazu útočníka odešle DNS služba přibližně 6 bajtů odpovědi oběti.

Druhou skupinou jsou logické útoky. Tyto útoky se snaží využít zranitelnosti aplikace či softwaru na zařízení oběti. Cíl je stejný jako u předešlé skupiny a to zabránit funkčnosti služby. Na rozdíl od předešlé skupiny, útoky v této skupině používají pro dosažení cíle mnohem menší počet paketů. Mezi nejznámější útoky této skupiny patří:

- **Ping of death** - Útočník pošle oběti ICMP paket s velikostí větší jak 65535 bajtů, což je největší možná velikost paketu definovaná v RFC 791[15]. Přijetí takového paketu může znamenat přetečení vnitřních čítačů a pád systému zařízení oběti.

- **Teardrop útok**- Útočník pošle oběti fragmentované pakety se špatně nastavenými hodnoty offsetu. Přijetí může vést k pádu systému zařízení oběti.

- **Land útok**- Útočník pošle oběti SYN paket se zdrojovou IP adresou a zdrojovým portem oběti. Při přijetí se oběť může snažit navázat TCP spojení sám se sebou, což může vést k pádu systému.

4.2 Provádění DDoS útoků

Uskutečnit útok za účelem znepřístupnění služby v dnešní době není žádný velký problém. Dokonce i amatérský programátor lehce dokáže naprogramovat program, který bude generovat a posílat co největší množství paketů. Při využití funkcí Linuxového kernelu pro generování nových paketů, může mít tento program pouze několik desítek řádků a přitom možnost generovat pakety v rychlostech gigabitů a vyšších.

Existuje také velké množství aplikací, kde k uskutečnění útoku stačí pár stisknutí. Jednou z těchto aplikací je například LOIC[9], open source program pro stress testing a DoS útoky, který dokáže generovat jednoduché útoky za použití protokolů TCP, UDP anebo HTTP. Dalším příkladem je aplikace Yersinia[31]. Tato aplikace umožňuje DoS útoky zaměřené na lokální síť, konkrétně na protokoly jako DHCP, STP, CDP a další.

Pro DDoS útoky je také možné využít jednu z nespočetně webových stránek[17], které nabízí realizace útoků ke koupi. Tyto stránky samozřejmě tvrdí, že nejsou určené k provádění útoků za účelem způsobení škody, ale že slouží pouze k testování obranyschopnosti systému kupujícího. Při koupi, ale nikdy nedochází ke kontrole zda je DDoS útok aplikován na systém kupujícího či systém někoho jiného. Samotný útok ani není nijak drahý, ceny se mohou pohybovat v nízkých desítkách dolarů za hodinu.

4.3 Detekce DDoS útoků

Obecně je při detekci DDoS útoků snaha o použití takového řešení, které je co nejpřesnější a nejrychlejší. Nejpřesnější řešení je takové řešení, které úspěšně detekuje a mitiguje co největší procento útoků a zároveň zablokuje co nejméně legitimního provozu. Nejrychlejší řešení je pak řešení, které co nejrychleji detekuje a mitiguje útok.

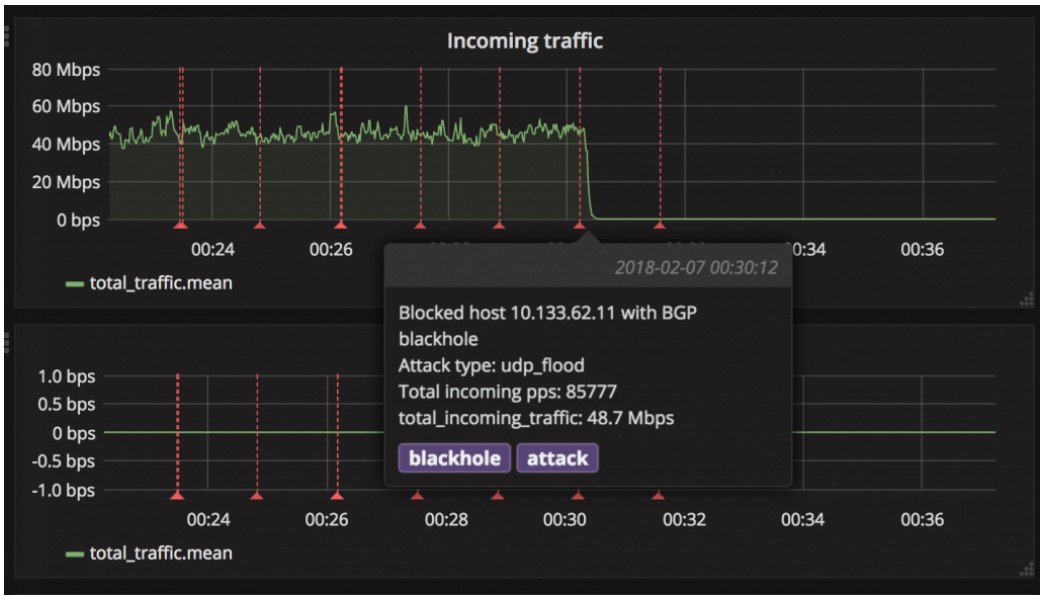
Řešení pro detekci DDoS útoků je možné rozdělit na dvě skupiny podle různých přístupů. Tyto přístupy jsou:

- **Out-of-band** - Zařízení provádějící detekci je umístěno mimo cestu provozu. Detekce je realizovaná analýzou dat získaných ze záznamů o tocích. Pro získání těchto záznamů se zde využívá technologie generující záznamy o tocích, jako například NetFlow. Příkladem aplikace využívající tento přístup je FastNetMon (viz. 4.3.1).
- **In-line** - Zařízení, které provádí detekci je umístěno přímo v cestě provozu. Toto zařízení pak může být k provozu čistě transparentní anebo může provádět routing. Detekce se provádí analýzou samotného provozu. Příkladem aplikace využívající tento přístup je nScrub (viz. 4.3.2).

4.3.1 FastNetMon

FastNetMon[12] je software pro obranu proti DDoS útokům. Tyto útoky detekuje analýzou informací získaných ze záznamů toků. Podporuje toky ve formátech sFlow, NetFlow, IPFIX a další. Lze škálovat až k rychlostem v řádech Terabitů/s. K vizualizaci dat využívá aplikaci Grafana. Umožňuje použití BGP Flowspec. Konfigurovat lze pomocí příkazového řádku či API, podporuje také přidání skriptů, které se vykonají při detekci útoku.

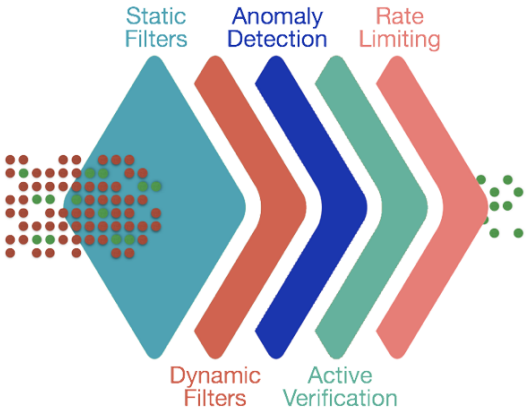
Ukázka oznámení detekovaného útoku je možná vidět na obrázku 4.1. Mimo jiné je na obrázku vidět identifikace typu útoku jeho rozsahu a způsob mitigace.



Obrázek 4.1: Ukázka oznámení útoku. (Převzato z materiálů FlowNetMon)

4.3.2 nScrub

nScrub[25] je software pro obranu proti DDoS útokům pracující na bázi PF_RING ZC (Zero Copy). Je schopen operovat při rychlostech 10 Gigabitů/s za použití nízko výkonového hardwaru. Podporuje modulární architekturu, díky které umožňuje operovat při rychlostech v řádech Terabitů/s. Konfigurovat lze za pomocí REST API anebo příkazového řádku. Na obrázku 4.2 je možné vidět znázornění architektury tohoto softwaru.



Obrázek 4.2: Znázornění architektury nScrub. (Převzato z materiálů nScrub)

Mezi hlavní podporované funkce patří verifikace aktivní relace protokolů jako TCP a DNS, tvorba blacklistů, filtrace na bázi vzorů z aplikačního protokolu, omezování rychlosti přenášených dat na základě informací z paketu a další. nScrub je navržen pro možnou rozšiřitelnost, což umožňuje uživatelům přidání nových funkcí detekce a mitigace specifických protokolů. Podporuje rozdělení příchozího provozu do více skupin na základě cílové IP adresy. Každá skupina má nastavený profil určující jaké funkce se s ní použijí. Lze tedy pro některé skupiny provozu nastavit přísnější detekci útoků.

nScrub může pracovat ve dvou možných módech. Tyto módy jsou:

- **Transparent Bridge Mode** - Výchozí mód, který je transparentní vůči provozu v síti, pouze filtruje zjištěné útoky.
- **Routing Mode**- Tento mód vyžaduje svou vlastní routovací tabulku. Umožňuje použití s protokolem BGP pro odklánění provozu.

4.4 Mitigace DDoS útoků

Mitigace obvykle následuje po detekci útoku. Mitigace útoku je akce, která se snaží plně zastavit, případně zmírnit, dopad právě probíhajícího DDoS útoku. Je zde snaha o co nejefektivnější zastavení útoku s co nejmenším dopadem na legitimní provoz.

Jedním možným způsob mitigace je přímé blokování IP adres hostů, nebo blokování celých prefixů. Toto blokování lze uskutečnit například použitím ACL (*Access Control List*) anebo používáním určitých funkcionalit směrovacího protokolu BGP (*Border Gateway Protocol*).

4.4.1 ACL - Access Control List

Access control listy jsou používány jako filtrační mechanismus na aktivních síťových prvcích. Umožňují filtraci pomocí informací ze síťové a transportní vrstvy.

Další popis bude primárně podle terminologie společnosti Cisco[7]. Mají podobu pravidel, kde každé pravidlo obsahuje akci (*permit* / *deny*), popis zdroje provozu a případně popis cíle provozu. Access control listy jsou očíslované unikátním identifikátorem. Každý tento list obsahuje implicitní pravidlo zahazující veškerý provoz. Tyto pravidla se aplikují na určitý směr vybraného síťového rozhraní a způsobí, že se veškerý provoz zkontroluje vůči těmto pravidlům a je buďto zahozen, nebo poslán dál.

Access control listy je možné rozdělit na dvě skupiny, standardní a rozšířené. Tyto skupiny se liší rozsahem svých možností u popisování pravidel.

Standardní access control listy

Je možné je poznat tak, že jejich identifikační číslo je v rozmezích 1 až 99 a 1300 až 1999. Standardní pravidla jsou limitovaná na filtraci pouze pomocí zdrojové IP adresy. Zadávaná pravidla mají tento tvar:

```
access-list 10 [deny / permit] <ip-adresa> <wildcard-maska>
```

Kde `<wildcard-maska>` je maska sítě v invertované formě. Konkrétní příklad pravidla, kde je povolena komunikace pouze ze sítě `192.168.0.0/24`, by vypadal následovně:

```
access-list 10 permit 192.168.0.0 0.0.0.255
```

V případě tvorby pravidla pro jednu konkrétní IP adresu uživatele je možné použít klíčové slovo `host`, které umožňuje vynechání políčka `<wildcard-maska>`

Rozšířené access control listy

Jejich identifikační číslo je v rozmezích 100 až 199 a 2000 až 2699. Oproti standardním access control listům mají rozšířené možnosti pro filtraci paketů. Filtrovat pakety dokáží navíc pomocí cílové adresy a informací o použitém transportním protokolu. Pravidla se zadávají ve tvaru:

```
access-list 100 [deny / permit] <protokol> <zroj> <cíl>
```

Políčko `<protokol>` slouží k vybrání konkrétního transportního protokolu. V případě, že je zapotřebí, aby pravidlo zahrnovalo všechny transportní protokoly, použije se klíčové slovo `IP`. Políčka `<zdroj>` a `<cíl>` obsahují informace o adrese a případně i portu zdroje/cíle komunikace. Tyto informace mají tvar:

```
<ip-adresa> <wildcard-maska> {<operator> <port>}
```

Konvence zadávání IP adresy a masky je stejná jako u standardních access control listů a to i možnost použití klíčového slova `host`. Část ve složených závorkách je nepovinná a obsahuje klíčové slovo operátoru a používané číslo portu transportního protokolu. Klíčové slovo operátoru `<operator>` je jedním z těchto možných: `eq` značící rovnost, `neq` značící nerovnost, `gt` značící je vetší než a `lt` značící je menší než. Je také možné použit klíčové slovo `range` a zadat rozsah portů.

Konkrétní příklad pravidla, které povolí pouze komunikace, kde je použit protokol TCP a je směřován na cílovou adresu `192.168.0.1` a port `179` je možné vidět zde:

```
access-list 100 permit TCP any host 192.168.0.1 eq 179
```

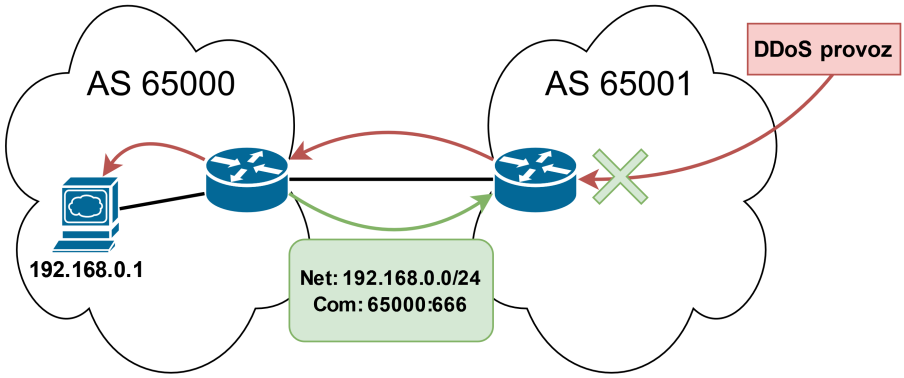
4.4.2 BGP - Border Gateway Protocol

BGP je směrovací protokol používaný v páteři internetu. BGP propojuje autonomní systémy internetu a distribuuje mezi nimi směrovací informace.

Jedna funkcionalit BGP je použití tzv. komunit. Tyto komunity slouží k označení směrovací informace posílané sousednímu autonomnímu systému. Komunity jsou pouze specifické číslo přidané ke směrovací informaci sloužící příjemci k lepší identifikaci této informace, případně může komunita napovídat jakou akci má příjemce s touto směrovací informací podniknout. Existuje pouze pár *well-known* komunit, tedy komunit, které musí každý účastník BGP procesu rozpoznávat. Používaných komunit je, ale značně více. Obvykle si velká entita určí svoje komunity a informuje o nich své sousedy. Jako například Peering.cz a jejich používané komunity[26].

RTBH - Remote Triggered Black Hole

Jedna z *well-known* komunit je komunita 666, blackhole[16]. Tato komunita může sloužit jako mitigace DDoS útoku. Tomuto použití se říká RTBH (*Remote Triggered Black Hole*). Používá se tak, že oběť DDoS útoku pošle svým BGP sousedům síť, která je napadána a přidá k této informaci blackhole komunity. Pro sousedy oběti je přijatá směrovací informace s blackhole komunitou zpráva o tom, že mají veškerý provoz, mířící do sítě získané z této směrovací informace, zahodit. Tímto je DDoS útok úspěšně mitigován, ale současně je zahazována i veškerá legitimní komunikace. Toto vede k tomu, že napadaná síť je nedostupná ze směru od autonomních systémů, které obdržely blackhole komunitu.



Obrázek 4.3: Znázornění využití mechanismu RTBH.

Flowspec

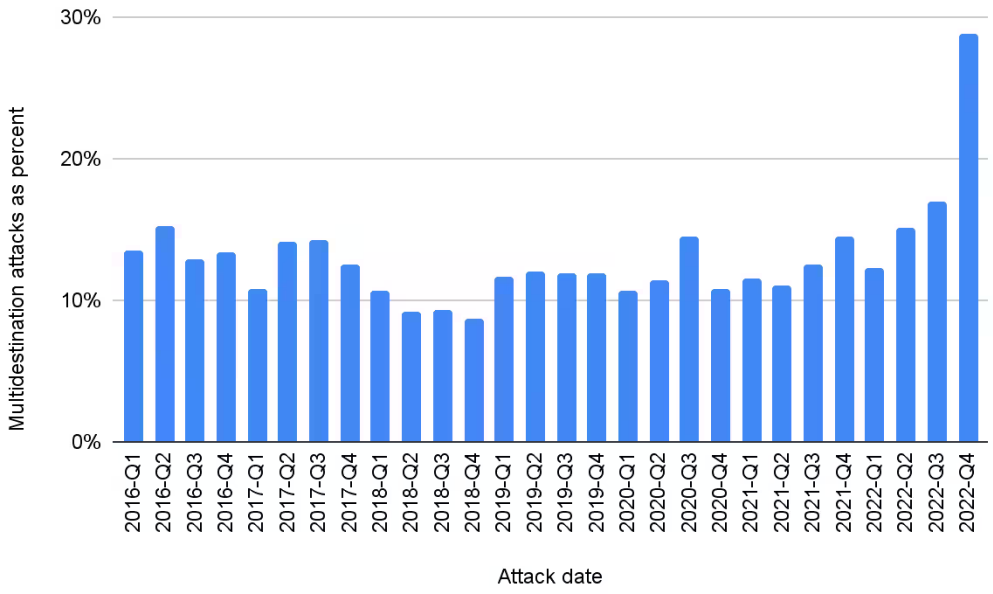
BGP Flowspec[18] je rozšíření BGP, které přidává nový typ BGP zpráv. Tyto zprávy slouží k zasílání směrovacích informací společně s akcemi popisující co se má stát s provozem cíleným na sítě z těchto informací.

Na rozdíl od RTBH, které při odeslání komunity způsobí zahazování všeho, vytváří Flowspec pravidla podobná access control listům, které umožňují přesnější filtraci provozu. Flowspec podporuje filtrování na základě cílové a zdrojové adresy, parametrů transportního protokolu a vlastností paketů, jako délka paketu. Při využití těchto vlastností, lze tedy zaslat sousednímu autonomnímu systému informaci o specifickém toku popisující provoz DDoS útoku a úspěšně tento útok mitigovat. Mitigace v tomto případě spočívá pouze v zahazování provozu spadajícího do popsaného toku, bez ovlivnění ostatního provozu. Flowspec také umožňuje více akcí než jen zahodit/poslat. Mezi tyto akce patří limitování rychlosti provozu, přesměrování anebo označení paketu DSCP hodnotou, kterou je možné zohlednit u QoS.

4.5 Dnešní trendy DDoS útoků

Na začátku roku 2020 byl zjištěn doposud největší útok proti společnosti Amazon[2]. Tento útok byl směřován na platformu AWS a dosahoval síly 2,3 Tbps. Využíval protokol UDP k provedení CLDAP reflexního[3] útoku. Tento útok byl o 44 % větší než předešlý největší útok na AWS. Útoky na AWS podobné síly za použití stejného provedení trvaly tři dny v kuse.

Statistiky společnosti Akamai za rok 2022 upozorňují na zvyšování zastoupení útoků na více cílů[4]. Obvyklé DDoS útoky se zaměřují na jeden cíl/jednu IP adresu, například adresu webové stránky. Útoky na více cílů neprioritizují jeden cíl, ale několik za účelem zvýšení obtížnosti mitigace pro bezpečnostní týmy. Toto zvyšování je možné vidět na obrázku 4.4.



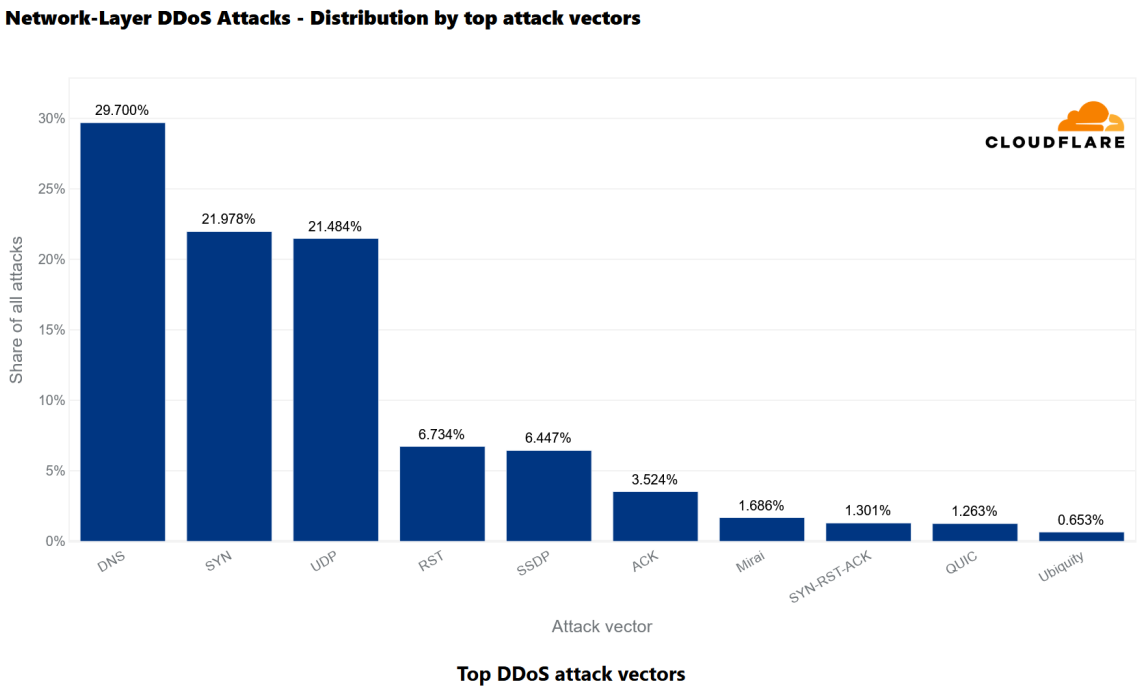
Obrázek 4.4: Ukázka procentuálního zastoupení více cílových DDoS útoků v posledních letech. (Převzato z materiálů Akamai)

Tyto statistiky společnosti Akamai také popisují nejpoužívanější útoky na nejvíce útočených portech. U portu 80 (HTTP) se značnou převahou vítězí útok SYN flood, který má 51,3 % zastoupení všech útoků na tento port. Port 53 (DNS) má několik vysoce používaných útoků, o první příčku se dělí útoky DNS flood, NTP flood a DNS reflection, kde tyto útoky dohromady mají zastoupení 75 % ze všech útoků na port DNS. Pro port

443 (HTTPS) existuje velké množství používaných útoků, mezi nejpoužívanější patří SYN flood, NTP reflection, UDP flood a TCP anomaly, kde se všechny tyto jednotlivé útoky pohybují kolem 15 % zastoupení. U všech portů s číslem 1000 a výše jsou nejdominantnější útoky UDP flood a NTP reflection.

V roce 2020 zaznamenala společnost Akamai největší útok v rámci počtu poslaných paketů[11]. Tento útok o síle 809 Mpps (milión paketů za sekundu) byl směřován na velkou evropskou banku. Byl použit protokol UDP s cílovým portem 80 (HTTP), kde každý paket přenášel data o velikosti pouhého jednoho bajtu.

Další velkou společností vedoucí si statistiky je Cloudflare[33]. Tyto statistiky dávají dohromady všechny útoky síťové vrstvy a vyznačují jejich zastoupení. Zastoupení útoků síťové vrstvy za první čtvrtletí roku 2023 je možné vidět na obrázku 4.5.

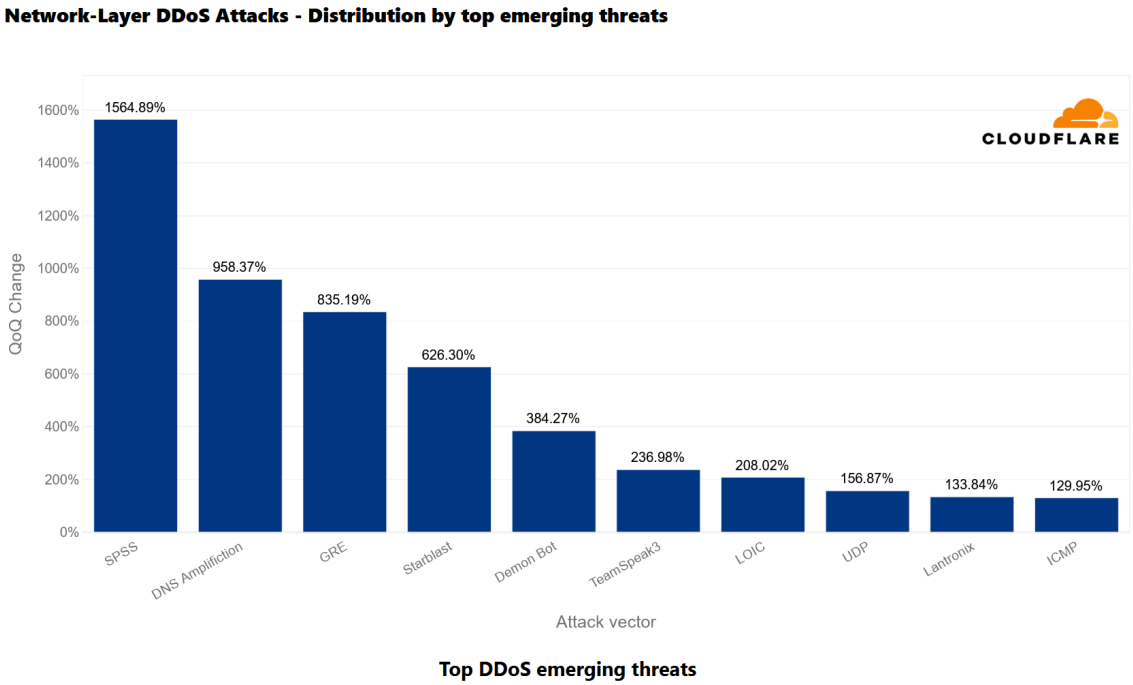


Obrázek 4.5: Útoky s největším zastoupením. (Převzato z materiálů Cloudflare)

Oproti předešlým čtvrtletím jsou toto čtvrtletí na prvním místě útoky DNS. Skoro třetina všech útoků byla na bázi DNS jako útoky DNS flood a reflexní DNS útoky. Obvykle nejčastější útok SYN flood se posunul na druhé místo, těsně následovaný UDP útoky.

Při zkoumání útoků s největším čtvrtletním nárůstem, je možné vidět důvod, proč se DNS útoky posunuly na první místo u předešlého obrázku. Útoky s největším nárůstem je možné vidět na obrázku 4.6. Je zde vidět 958 % zvýšení u reflexních útoků DNS využívající fakt, že požadavek útočníka je menší než odpověď DNS serveru, síla útočníka je tedy zesílena podle poměru velikosti těchto odpovědí. Nejvíce vzrůstající útok je na bázi SPSS (*Statistical Product and Service Solutions*). SPSS je soubor programů pro firemní použití vyvíjený společností IBM. Útočník může využít server spravující licence těchto programů pro uskutečnění reflexního útoku. Útočník pošle vytvořené požadavky na licenční server, který vygeneruje odpověď oproti požadavku mnohem větší a pošle jí na adresu oběti. Další útok, který zaznamenal značný čtvrtletní nárůst je útok s použitím protokolu GRE (*Generic Routing Encapsulation*). GRE je protokol pro vytváření tunelů v síti. Útočník vytvoří něko-

lik GRE tunelů ke kompromitované oběti, přes které posílá data do sítě oběti. Tento útok je obtížný detekovat, protože pro síť, přes kterou je tunel vytvořen, vypadá jako legitimní provoz.



Obrázek 4.6: Útoky s největším čtvrtletním vzrůstem. (Převzato z materiálů Cloudflare)

V druhé polovině roku 2021 se pro Cloudflare staly normou útoky přesahující sílu přes 1 Tbps, kde nejsilnější z nich měl sílu přes 2 Tbps[32], čímž se stal nejsilnějším útokem v historii Cloudflare. Tento útok byl multi-vektorový, tedy kombinoval více DDoS technik. Konkrétně využíval kombinaci UDP flood a reflexního útoku pomocí DNS. Tento útok trval jednu minutu.

Kapitola 5

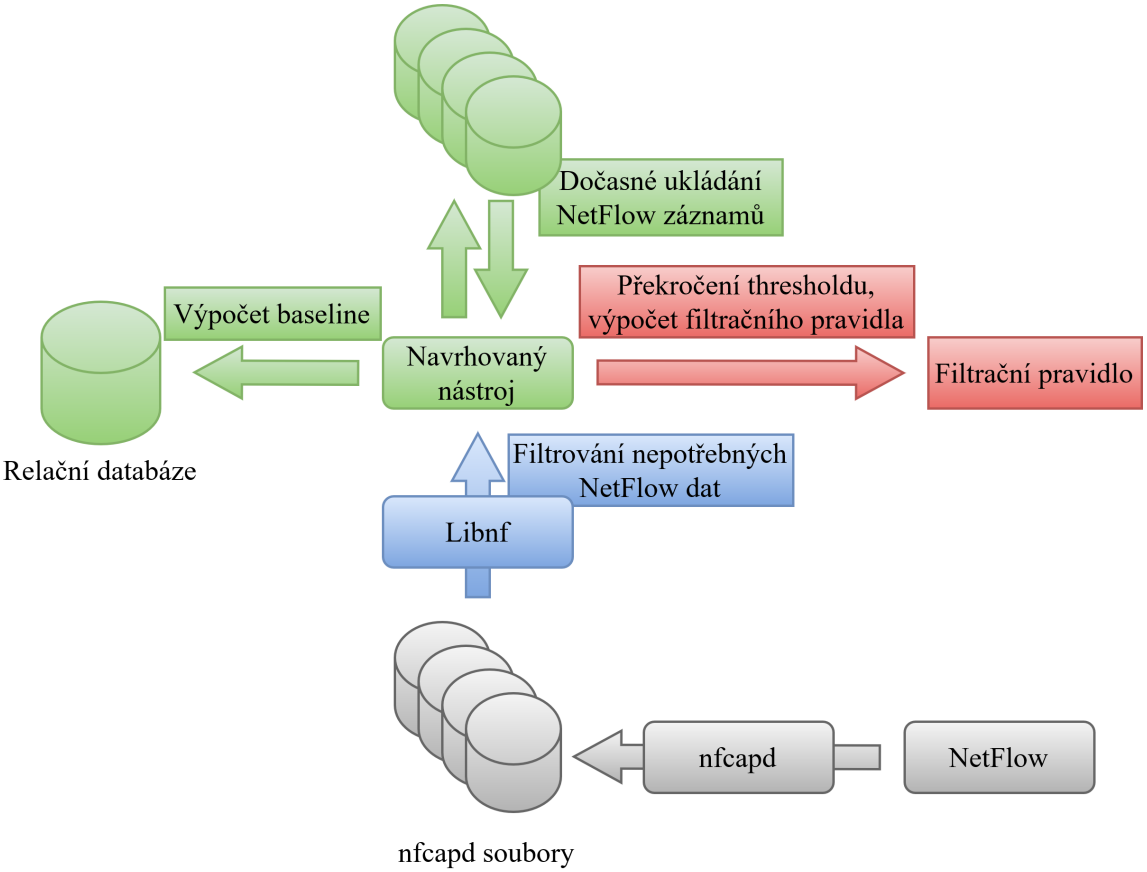
Návrh

Cílem této práce je vytvořit program pro automatickou detekci síťových útoků a následného nalezení jejich vzoru. Tento program musí analyzovat nfcapd soubory, zjistit momentální trend komunikace v síti a být schopen detekovat prudké zvýšení množství přenášených dat. Po detekci musí na základě dat z blízké minulosti najít viníky tohoto prudkého zvýšení a tyto viníky obsáhnout ve filtračním pravidlu.

5.1 Popis navrhovaného řešení

Vizualizaci navrhovaného řešení je možné vidět na obrázku 5.1. Je zde barevně rozděleno na více částí. Tyto části jsou:

- **Šedá část** znázorňuje použití Nfdump sady nástrojů, konkrétně nástroj nfcapd, který bude použit jako NetFlow kolektor. Bude příchozí NetFlow záznamy ukládat do souboru *nfcapd.current* a po uplynutí určité doby, je rozdělí do příslušných pětiminutových souborů.
- **Modrá část** ukazuje uplatnění libnf knihovny. Knihovna bude napojena na soubor *nfcapd.current*, který bude neustále, opakovaně číst a propagovat navrhovanému nástroji veškeré informace o jednotlivých záznamech.
- **Zelená část** vyobrazuje samotné jádro navrhovaného nástroje. Tento nástroj bude od knihovny libnf dostávat informace o záznamech, které použije pro výpočet *baseline*, neboli průměrné množství aktuálního provozu v síti. Hodnotu *baseline* bude periodicky ukládat do relační databáze, k trvalému uložení, aby administrátorovi umožňoval lepší monitoring sítě. Detekci bude provádět na základě porovnávání momentální a předešlé hodnoty *baseline*, když dojde k většímu zvýšení, než hranice určená administrátorem, je detekován útok. Při přebírání dat od knihovny libnf si bude nástroj tyto data ještě ukládat a to po určitou dobu.
- **Červená část** obsahuje část nástroje pro nalezení viníků a výpočet filtračního pravidla. Pracovat bude nad sadou dočasně uložených informací o záznamech zmíněné výše. Po detekci se bude v této sadě snažit najít vzor útoku.



Obrázek 5.1: Navrhované řešení

5.2 Možnosti administrátora

Administrátor bude schopen ovlivňovat funkčnost programu pomocí nastavitelných parametrů. Mezi plánované parametry patří: frekvence ukládání *baseline* do relační databáze, míra zvýšení *baseline* k detekci, časové okno pro dočasné ukládání záznamů a časové okno, ze kterého se počítá *baseline*.

Administrátor bude také moci vložit filtry, které se použijí pro vyfiltrování záznamů z nfcapd souborů. Tímto bude schopen umožnit nástroji analyzovat pouze určitou podmnožinu záznamů. Těchto filtrů bude možno použít i více. Každý takový filtr bude fungovat odděleně od ostatních, bude mít vlastní *baseline*, tabulku v relační databázi, dočasně uložená data a bude nezávisle provádět detekce. Parametry zmíněné výše bude mít každý takový filtr vlastní.

Kapitola 6

Implementace

Program byl implementován na školním serveru s přístupem k nfcapd NetFlow kolektoru s daty z páteřní sítě VUT. Jako databázový server pro relační databázi byl použit server PostgreSQL[28]. Zdrojové soubory jsou naprogramovány v jazyce C s použitím vícevláknového zpracování a možností spuštění programu jako daemon. Mimo standardní knihovny byly použity knihovny:

- **libnf** - Pro práci se záznamy a pro přístup k souboru *nfcapd.current*.
- **libpq** - Pro napojení na PostgreSQL databázový server.
- **pthread** - Pro vícevláknové zpracování.

6.1 Získávání a propagace záznamů

Při spuštění programu, dojde k načtení parametrů z konfiguračního souboru 6.5. Mezi tyto parametry mimo jiné patří slovní popis jednotlivých *input-filtrů*. Tyto *input-filtry* slouží administrátorovi k zaměření programu na určitý síťový provoz. Jakmile jsou získané parametry a je jasný pevně daný počet *input-filtrů*, rozdělí se program na více vláken. Každému nastavenému *input-filtru* se vytvoří jedno vlákno speciálně pro něj. Vlákno, které provádí vytváření ostatních je označeno jako hlavní vlákno. Pokud je nastavena tvorba logů, vytvoří se proto ještě jedno vlákno navíc.

Účel hlavního vlákna je získat nově příchozí záznamy a rozdistribuovat je příslušným *input-filtrům*. Využívá knihovnu libnf pro napojení na soubor *nfcapd.current* a periodické čtení z něj. Při přečtení každého nového záznamu se aktualizuje libnf struktura *lnf_rec_t*, která obsahuje veškeré informace o záznamu. Jelikož tato struktura obsahuje spoustu, pro tento program, nepotřebných informací, přečtou se z ní jenom potřebné informace, které se vloží do jiné, nové struktury. Tato nová struktura *Ndd_rec_t* 6.1 vytváří oboustranně vázaný seznam, kde každý prvek obsahuje informace o právě jednom záznamu. Z *lnf_rec_t* do *Ndd_rec_t* se přesouvá celá struktura *lnf_brec1_t* 6.1 naráz. Struktura *lnf_brec1_t* obsahuje informace o času první a poslední komunikace (first, last), zdrojové a cílové IP adresy (srcaddr, dstaddr), číslo protokolu (prot), zdrojové a cílové porty (srcport, dstport), sumu bytů celé komunikace, počet přenesených paketů komunikace a informaci o tom, jestli byl záznam agregován a z kolika původních záznamů. *Ndd_rec_t* obsahuje celou strukturu *lnf_brec1_t*, hodnotu tcp_flags, informaci o tom, jestli byl tento záznam již zpracován a ukazatele na předešlý / následující záznam, zajišťující ono oboustranné provázání.

Ndd_rec_t	
lnf_brec1_t	brec
uint8_t	tcp_flags
int	processed
Ndd_rec_t *	next
Ndd_rec_t *	prev

lnf_brec1_t	
uint64_t	first
uint64_t	last
lnf_ip_t	srcaddr
lnf_ip_t	dstaddr
uint8_t	prot
uint16_t	srcport
uint16_t	dstport
uint64_t	bytes
uint64_t	pkts
uint64_t	flows

Tabulka 6.1: Znázornění formátu struktur ***Ndd_rec_t*** a ***lnf_brec1_t***

Hlavní vlákno při získání nového záznamu, tento záznam musí porovnat vůči všem *input-filtrům*. Záznamu je poté vytvořena ona struktura *Ndd_rec_t* a to pro každý *input-filtr*, kterému tento záznam odpovídá. Vlákno *input-filtru* periodicky kontroluje, jestli do jeho seznamu nepřibyl nový záznam ke zpracování.

6.2 Zpracování záznamů

Účelem každého vlákna jednotlivých *input-filtrů* je převzetí záznamu, počítání *baseline* bajtů a paketů, ukládání do databáze, detekce útoku a nalezení vzoru tohoto útoku. Práci těchto vláken, může administrátor ovlivnit několika parametry v konfiguračním souboru a to pro každé vlákno zvlášť. Každý *input-filtr* má své interní označení ve tvaru:

```
ndd{TIMESTAMP}f{ORDER}
```

Kde `TIMESTAMP` je časové razítko spuštění programu a `ORDER` je pořadí v jakém byl konkrétní *input-filtr* přečten z konfiguračního souboru.

6.2.1 Výpočet hodnot popisujících aktuální provoz

Na začátku si vlákno inicializuje dvě pole, jedno určené paketům a druhé bajtům. Do obou těchto polí se indexuje pomocí časového razítka získaného ze záznamu a to tak, že jeden index v poli odpovídá jedné sekundě časového razítka. Pole se indexuje od současnosti (nejnovějšího časového razítka jakéhokoliv záznamu) do minulosti. S tím, že časové omezení pole je jedním z parametrů v konfiguračním souboru (konkrétně *baseline_window*). Při klasifikaci jednotlivých záznamů do příslušných indexů, také dochází k počítání celkového počtu bajtů a paketů ze záznamů, za časové omezení pole. Hodnoty celkových počtů se poté použijí pro výpočet *packet_baseline* a *byte_baseline*. Obě hodnoty *baseline* se vypočítají s pomocí vzorce 6.1 jako vážený aritmetický průměr s ponížením váhy nejnovějších hodnot.

$$NewBaseline = \frac{OldBaseline + Traffic * C}{1 + C} \tag{6.1}$$

Kde *NewBaseline* je nově vypočítaná *baseline*, *OldBaseline* je *baseline* předchozího záznamu, C je koeficient získaný jako parametr v konfiguračním souboru (konkrétně *coeffici-*

ent) a *Traffic* je hodnota *pps* (pakety za sekundu) nebo *Bps* (bajty za sekundu), vypočítaná podle vzorce 6.2.

$$Traffic = \frac{Total}{baseline\_window} \tag{6.2}$$

Kde *Traffic* je průměrná hodnota *pps* nebo *Bps* za poslední časové omezení pole, *Total* je celkový počat bajtů nebo paketů, za časové omezení pole a *baseline_window* značí hodnotu časového omezení pole v sekundách.

Když je vypočítaná aktuální *baseline*, dochází ke kontrole zda nedošlo k příliš náhlému a příliš velkému zvětšení. Aktuální *baseline* je porovnávána s *thresholdem* 6.3, který značí největší možné, přípustné zvětšení za určitou dobu. Pokud je *threshold* větší, je vše v pořádku a pokračuje se dál. Pokud je *baseline* větší znamená to možný útok. V tomto případě dochází k vložení informace o přílišném nárůstu do databáze 6.3.2 a provedení algoritmu hledání vzoru útoku 6.4.

$$Threshold = Baseline * MaxIncrease \tag{6.3}$$

Kde *Baseline* je zrovna aktuální a *MaxIncrease* je konstanta získaná jako parametr v konfiguračním souboru (konkrétně *max_baseline_increase*).

Při aktualizaci současnosti (nejnovějšího časového razítka jakéhokoliv záznamu), dochází ke kontrole, zda není čas pro vložení aktuální *baseline* do databáze 6.3.1. K vložení dochází periodicky v pevných intervalech podle hodnoty parametru získaného z konfiguračního souboru (konkrétně *db_insert_interval*). V této příležitosti, také nastává aktualizace hodnoty *threshold* za pomocí právě uložené *baseline*. Tato hodnota *threshold* bude použita pro celý nadcházející interval trvající, až do dalšího ukládání *baseline* do databáze.

6.2.2 Dočasné ukládání hodnot záznamů

Při běžném běhu programu je zapotřebí si ukládat všechny zpracované záznamy po nějakou dobu. Důvodem takového ukládání je, že v případě detekce útoku je ihned k dispozici seznam záznamů ve kterých se bude hledat vzor tohoto útoku. Seznam záznamů pro hledání útoků, dále jen *dataset*, se periodicky ukládá do souborů pomocí knihovny libnf a v případě detekce útoku dochází ke čtení těchto souborů.

Dočasné ukládání může administrátor ovlivnit použitím dvou parametrů v konfiguračním souboru. Jedním z těchto parametrů (konkrétně *dataset_window*) je hodnota, určující čas v sekundách, po jakou dobu se budou záznamy dočasně uchovávat. Druhý parametr (konkrétně *dataset_chunks*) ovlivňuje množství souborů, které budou v jednu chvíli obsahovat záznamy.

Součástí získání a zpracování jednotlivých záznamů není jejich odstranění z paměti. K jejich odstranění dochází současně s uložením do souboru *datasetu*. Program díky parametrům z konfiguračního souboru zná informace o tom kolik souborů *datasetu* má v jednu chvíli existovat a jaké časové okno má tento *dataset* brát v úvahu. Počas běhu program se záznamy hromadí v paměti a kontroluje se zda nenastal čas vytvořit nový soubor *datasetu*. Pokud nastal čas, vytvoří se nový soubor s aktuálním časovým razítkem a vloží se do něj všechny záznamy z paměti, které již byly zpracovány. Během tvorby nového souboru, také dochází ke smazání nejstaršího souboru *datasetu*, který je v tuto dobu starší než časové okno zapsané v konfiguračním souboru.

Soubory *datasetu* mají pevně danou strukturu názvů:

```
ndd-{INDEX}.{TIMESTAMP}
```

Kde `INDEX` je číselné označení souboru v rozsahu od nuly do o jedna méně než maximální počet souborů a `TIMESTAMP` je časové razítko v dobu tvorby souboru. Při tvorbě nového souboru platí, že dojde ke smazání starého souboru a to takového, kde `INDEX` nového souboru a `INDEX` starého souboru jsou stejná čísla.

Tyto soubory *datasetu* se ukládají do specifické složky, která se vytváří při startu programu. Cesta k umístění souborů z kontextu programu:

```
./datasets/{FILTER}/{FILE}
```

Kde `FILTER` je interní označení konkrétního *input-filtru* a `FILE` je název souboru *datasetu* popsaný výše.

Jelikož je ukládání souboru *datasetu* periodické, může se stát, že dojde k detekci v nějakém čase po posledním ukládání. Toto vede k tomu, že program má v paměti několik zpracovaných záznamů, které ještě nejsou uloženy v souborech *datasetu*. Algoritmus hledání vzoru útoku proto čte nejen záznamy ze souborů *datasetu*, ale získává i zpracované, ještě neuložené záznamy přímo z paměti.

6.3 Databáze

Jako databázový server se používá server PostgreSQL a ke komunikaci s ním je použita knihovna libpq. Pro propojení programu s SQL server je zapotřebí zadat přihlašovací údaje do parametru v konfiguračního souboru (konkrétně *connection_string*). Přihlašovací údaje musí obsahovat název již existující databáze a jméno uživatele pod kterým se k databázi bude přistupovat.

Všechny potřebné tabulky databáze k chodu programu se vytváří na začátku běhu programu. Tabulky jsou dvě obecné určené všem *input-filtrům* a jedna speciálně pro každý existující *input-filtr*.

6.3.1 Tabulky filtrů

Každému *input-filtru* se při jeho detekci v konfiguračním souboru vytvoří tabulka speciálně pro něj. Tato tabulka bude mít stejný název jako je interní označení jejího *input-filtru*. Tyto tabulky nemají pevnou strukturu a mohou se od sebe lišit počtem i významem jejich sloupců. Jaké sloupce bude určitá tabulka mít může administrátor částečně ovlivnit použitím parametru v konfiguračním souboru (konkrétně *columns*). Do tohoto parametru administrátor vypíše všechny sloupce, které chce, aby tabulka měla. Všechny tabulky mají jeden pevně daný sloupce a to je `time`, do kterého se ukládá časové razítko. Struktura tabulky *input-filtru* `ndd1679694791f1` vypadá takto:

Tato ukázka je největší možná tabulka vytvořena zadáním parametru v konfiguračním souboru takto:

```
columns = "byte_baseline bps packet_baseline pps"
```

ndd1679694791f1	
Sloupec	**Datový typ**
time	timestamp wihout time zone
byte_baseline	bigint
bps	bigint
packet_baseline	bigint
pps	bigint

Tabulka 6.2: Ukázka struktury tabulky konkrétního *input-filtru*.

Tabulka *input-filtru* slouží pro ukládání hodnot *baseline* a naměřených hodnot aktuálního provozu v síti. Tyto hodnoty program nikdy dále nepoužívá, jejich smyslem je umožnění administrátorovi pohled do minulosti a snazší vytváření statistik minulého provozu.

6.3.2 Ostatní tabulky

Zbylé dvě tabulky jsou určeny pro informace od všech *input-filtrů* a proto mají pevnou strukturu. Jsou to tabulky `detected`, `filters` a `active_filters`.

Filters

Jelikož se berou data uložená v databázi jako perzistentní, je zapotřebí uchovat informaci o textovém zadání *input-filtru* i po případném ukončení běhu programu. Implementace tohoto programu, také nevylučuje možnost použití jedné databáze více instancemi programu.

filters	
Sloupec	**Datový typ**
id	character varying(20)
filter	text
active	boolean

Tabulka 6.3: Struktura tabulky **filters**.

Tato tabulka `filters` obsahuje spojení vnitřního označení *input-filtru* a textového zadání tohoto *input-filtru* v konfiguračním souboru. Vnitřní označení je zde uloženo ve sloupci `id`, textové zadání ve sloupci `filter` a sloupec `active` obsahuje informaci, zda je onen *input-filtr* aktivní, tedy má aktivní vlákno, které sleduje nově příchozí záznamy a počítá hodnoty popisující provoz.

Detected

Do této tabulky se vkládají informace o zjištěném útoku. Tedy při každém překročení *thresholdu*. Tabulka je určená všem *input-filtrům* zároveň.

Sloupec `id` obsahuje interní označení *input-filtru*, který detekoval překročení *thresholdu*, `time` obsahuje časové razítko momentu, kdy k překročení došlo, `baseline` obsahuje hodnotu sledované *baseline* v momentu překročení, `prev_baseline` obsahuje hodnotu *baseline*, ze které byl vypočítán aktuální *threshold* a sloupec `type` obsahuje informaci zda byla překročena *threshold* počítaná pro pakety, nebo bajty.

detected	
Sloupec	**Datový typ**
id	character varying(20)
time	timestamp without time zone
baseline	bigint
prev_baseline	bigint
type	character varying(10)

Tabulka 6.4: Struktura tabulky **detected**.

Hodnoty již vložené do tabulky `detected` se v programu nikde nepoužívají a slouží pouze jako dodatečná informace administrátorovi.

Active filters

Do této tabulky se vkládají informace o nově vytvořených *active-filtrech*. Vytvářejí se při úspěšném nalezení vzoru útoku a obsahují informace o nalezeném vzoru. Tabulka je společná pro všechny *input-filtry*.

active_filters	
Sloupec	**Datový typ**
id	character varying(20)
filter	text
start	timestamp without time zone
stop	timestamp without time zone
filtered_bytes	bigint
filtered_packets	bigint

Tabulka 6.5: Struktura tabulky **active_filters**.

Sloupec `id` obsahuje interní označení *input-filtru*, který detekoval útok a úspěšně nalezl vzor tohoto útoku, `filter` obsahuje výsledné textové znění vytvořeného filtru z nalezeného vzor, ukázka je možná vidět níže 6.4.4, `start` obsahuje časové razítko momentu, kdy došlo k nalezení vzoru útoku, `stop` obsahuje časové razítko momentu, kdy přestane být vytvořený filtr z nalezeného vzoru aktivní, `filtered_bytes` obsahuje množství vyfiltrovaných bajtů tímto konkrétním filtrem, za celé jeho aktivní období a `filtered_packets` stejné jako počítání jako pro bajty, ale obsahuje množství vyfiltrovaných paketů

6.4 Hledání vzoru útoku

Tato část programu se provádí při detekci náhlého zvýšení *baseline*. Cílem této části je prozkoumat hodnoty z *datasetu* a pokusit se najít vzor útoku. Algoritmus hledání vzoru funguje na principu agregování záznamů podle určitých klíčů a hledání viníků, tedy agregované hodnoty s největším provozem. Pokud byl výsledný vzor nalezen, tvoří ho použité agregované klíče a hodnoty viníků u patřičných klíčů. Pseudo kód tohoto algoritmu je možné vidět níže 1.

Algorithm 1: Pseudo kód algoritmu hledání vzoru útoku.

```
1  readDataset()
2  threshold2 = threshold
3  while current > threshold do
4      for Každý item ∈ eval_items do
5          filterDataset()
6          current = getCurrentFromFilteredDataset()
7          if current <= threshold then
8              break
9          aggregateDataset()
10         if top1 > threshold2 then
11             filter_candidate.add(top1)
12     if filter_candidate contains required_items then
13         create active_filter
14         threshold2 = threshold
15     else
16         thstep − −
17         if thstep <= 0 then
18             break
19         threshold2 = (threshold/thsteps) ∗ thstep
20 return active_filter
```

Program přečte všechny soubory *datasetu* a nastaví si hodnotu *threshold2*. Pro každý prvek *eval_items* vyfiltruje a agreguje podle aktuálního prvku *eval_items dataset*. Přitom počítá sumu přenesených bajtů, ze které zjistí průměrný provoz za sekundu a porovná ho s hodnotou *threshold*, pokud je hodnota průměrného provozu větší, pokračuje, jinak končí algoritmus. Vezme si záznam s největší sumou přenesených bajtů z agregovaného *datasetu*. Hodnoty tohoto záznamu porovná s hodnotou *threshold2*. Pokud je hodnota záznamu větší, je vložen do *filter_candidate*.

Po zpracování všech prvků *eval_items* následuje kontrola obsahu *filter_candidate* vůči *required_items*. Pokud jsou všechny prvky *required_items* obsaženy v *filter_candidate*, byl nalezen vzor útoku. Vytvoří se nový filtr skupiny *active-filtrů*, resetuje se hodnota *threshold2* a pokusí se najít více vzorů. Jestliže nebyly nalezeny všechny prvky *required_items* v *filter_candidate*, dojde k ponížení hodnoty *threshold2* a znovu se pokusí najít vzor útoku. Pokud byla hodnota *threshold2* ponížena mockrát, je algoritmus ukončen i v případě nenalezení vzoru.

Následuje podrobné popsání jednotlivých částí algoritmu hledání vzoru útoku.

6.4.1 Získání dočasně uložených záznamů

Vlákno *input-filtru*, které detekovalo náhlé zvýšení *baseline*, poskytne umístění všech svých souborů *datasetu* a hodnoty zpracovaných, ještě neuložených záznamů z paměti. Všechny tyto soubory jsou následně přečteny pomocí libnf a shromážděny do paměti. Uložení v paměti je zprostředkováno použitím struktury libnf *lnf_mem_t*. Tato struktura byla použita z důvodu její efektivní agregace. Každý vložený záznam do této struktury je okamžitě agregován podle zvolených klíčů.

Pro tyto záznamy z původního *datasetu* jsou použity jako klíče všechny podporované hodnoty z *eval_items*. Pro hodnoty počtu přenesených bajtů a paketů je nastavena tvorba sumy a pro časové údaje je nastaveno zapamatování nejstaršího a nejmladšího časového údaje. Tyto dva časové údaje nám udávají časové okno, ve kterém byl alespoň jeden z původních záznamů, agregovaných do patřičného agregovaného záznamu aktivní.

6.4.2 Filtrace a agregace záznamů v paměti

Změna uložených záznamů je prováděna pouze při kopírování záznamů z jedné *lnf_mem_t* struktury do druhé. Tímto vznikají dvě *lnf_mem_t* stuktury, jedna původní se všemi záznamy z *datasetu* a druhá, filtrovaná a více agregovaná.

Filtrace

Záznamy jsou při kopírovaní mezi *lnf_mem_t* strukturami kontrolovány vůči nastaveným filtrům a na základě výsledků této kontroly je rozhodnuto, zda bude záznam zkopírován anebo zahozen. Při kontrole jsou použity dvě skupiny filtrů. Jedna skupina *aggr-filtrů* a druhá skupina *active-filtrů*. Skupina *active-filtrů* obsahuje filtry, které jsou výsledky nalezeného vzoru útoku z *datasetu* a slouží k vyfiltrování nepotřebných záznamů. Skupina *aggr-filtrů* obsahuje filtry pro specifikaci při hledání vzoru útoku.

Pro *active-filtry* platí, že pokud se při kontrole záznam shodne s jakýmkoliv filtrem z *active-filtrů*, je tento záznam zahozen. Důvodem je, že pokud pro tento záznam existuje *active-filtr*, tak už byl původce provozu ze záznamu zjištěn jako viník nějakého útoku a je zbytečné, aby byl tento viník obsažen ve více jak jednom *active-filtru*.

Pro *aggr-filtry* platí, že se každý záznam, který má být zkopírován, musí shodnout se všemi *aggr-filtry*. Tyto *aggr-filtry* zajišťují možnost zaměření algoritmu na určité typy záznamů s největším provozem.

Agregace

Agregace je prováděna knihovnou libnf, při zkopírování záznamu do struktury *lnf_mem_t*. Před kopírováním se nastaví klíč agregace jako aktuálně zkoumaný prvek *eval_items*.

6.4.3 Ponížení hodnoty *threshold2*

Nová hodnota *threshold2* je vypočítaná podle vzorce 6.4. Před výpočtem je vždy dekrementovaná hodnota *thstep*.

$$\begin{aligned} thstep &= thstep - 1 \\ threshold2 &= \frac{threshold}{thsteps} * thstep \end{aligned} \tag{6.4}$$

Smyslem ponížení hodnoty *threshold2* je snížení nároků výběru nového prvku do skupiny *filter_candidate*. Hodnoty *thsteps* a *thstep* jsou parametry získané z konfiguračního souboru.

6.4.4 Tvorba filtru

Při nalezení všech prvků *required_items* v *filter_candidate* dojde k vytvoření pravidla, které popisuje vzor útoku. Toto pravidlo je tvořeno z prvků *required_items* a jejich patřičných, nalezených hodnot z prvků *filter_candidate*. Při tvorbě je použita stejná syntax jako

u nástroje nfdump. Výsledný filtr je tvořen konjunkcí textového zadání *input-filtru*, který detekoval útok a všech prvků *required_items* s jejich hodnotami. Ukázkový příklad:

Ukázkové hodnoty z konfiguračního souboru:

```
filter = "src port 53"
required_items = "srcip dstport"
```

Hodnoty získané z algoritmu:

```
filter_candidate = {"srcip 172.16.1.105", "dstport 54227"}
```

Výsledný filtr:

```
src port 53 AND src ip 172.16.1.105 AND dst port 54227
```

Po vytvoření výsledného textového znění filtru jsou informace o něm uložené do databáze 6.3.2. Takto vytvořený filtr je vložen do skupiny *active-filtrů*, kde se použije k filtraci *datasetu*. Po určité době se všechny jednotlivé filtry skupiny *active-filtrů* stanou neaktivními a jsou z této skupiny odstraněny.

6.5 Konfigurační soubor

Konfigurační soubor se vyskytuje ve stejné složce jako zdrojové soubory programu a má název `ndd.conf`. Konfigurační soubor může obsahovat komentáře označené znakem `#`, tyto komentáře jsou programem ignorovány. Níže jsou popsány všechny parametry:

6.5.1 Obecné parametry

Tyto parametry nemají výchozí hodnoty a jsou povinné pro korektní spuštění programu.

- *connection_string*
  - Obsahuje údaje potřebné pro připojení k PostgreSQL serveru.
  - Ukázka zadání: `connection_string = "dbname={NAZEV} user={UZIV}"`
    Kde `NAZEV` je název databáze, ke které se program má připojit a `UZIV` je jméno uživatele, pod kterým se má program přihlásit k serveru.
- *nfcapd_file*
  - Obsahuje cestu k souboru *nfcapd.current*, nebo jinému nfcapd souboru.
  - Ukázka zadání: `nfcapd_current = "/netflow/nfcapd.current.1234"`

6.5.2 Zadání *input-filtru*

Zadání parametru *filter* mění kontext konfiguračního souboru, více popsáno zde 6.5.4.

- *filter*
  - Obsahuje textové zadání *input-filtru*.
  - Ukázka zadání: `filter = "src port 123"`
  - Při tvorbě filtru se používá libnf funkce *lnf_filter_init_v1* [22] a očekává se stejná syntax zadání filtru jako u nfdump.

6.5.3 Parametry pro konkrétní *input-filtr*

Tyto parametry mají pevně dané výchozí hodnoty. Pokud bude nějaký z těchto parametrů v konfiguračním souboru chybět, program pokračuje s dosazenou výchozí hodnotou. Jak funguje dosazování výchozích hodnot je popsáno více zde 6.5.4.

- *baseline_window*
  - Vytyčuje časové okno v sekundách, které určuje jak moc staré záznamy se budou používat při výpočtu *baseline*.
  - Ukázka zadání a výchozí hodnota programu: `baseline_window = 300`
- *max_newest_cutoff*
  - Počet sekund o kolik může být nový nejnovější záznam novější oproti aktuálnímu nejnovějšímu záznamu a stále se brát jako korektní. Pokud přijde záznam novější o více sekund než je hodnota tohoto parametru, je čas tohoto nového záznamu ignorován.
  - Ukázka zadání a výchozí hodnota programu: `max_newest_cutoff = 20`
- *coefficient*
  - Konstanta použitá ve výpočtu *baseline* ve vzorci 6.1. Čím větší je hodnota této konstanty, tím větší je reakce na nové hodnoty.
  - Ukázka zadání a výchozí hodnota programu: `coefficient = 300`
- *max_baseline_increase*
  - Konstanta použitá při výpočtu *threshold* ve vzorci 6.3. Čím menší hodnota, tím menší tolerance náhlých zvětšení *baseline*.
  - Ukázka zadání a výchozí hodnota programu: `max_baseline_increase = 3`
- *dataset_window*
  - Čas v sekundách značící po jakou dobu se budou dočasně ukládat záznamy pro případ útoku.
  - Ukázka zadání a výchozí hodnota programu: `dataset_window = 30`
- *dataset_chunks*
  - Počet souborů, který může *dataset* v jednu chvíli nabývat. Za účelem správného fungování by měla být hodnota *dataset_window* bezezbytku dělitelná hodnotou *dataset_chunks*.
  - Ukázka zadání a výchozí hodnota programu: `dataset_chunks = 6`
- *thstep*
  - Počáteční hodnota proměnné *thstep* použité při ponižování hodnoty *threshold2* u hledání vzoru útoku. Proměnná použitá ve vzorci 6.4.
  - Ukázka zadání a výchozí hodnota programu: `thstep = 4`

- *thsteps*
  - Konstanta použitá při ponižování hodnoty *threshold2* u hledání vzoru útoku. Hodnota použitá ve vzorci 6.4.
  - Ukázka zadání a výchozí hodnota programu: `thsteps = 4`
- *eval_items*
  - Hodnoty podle, kterých bude docházet k agregaci a případné filtraci, při hledání vzoru útoku.
  - Podporované hodnoty: `srcip`, `dstip`, `prot`, `srcport`, `dstport`, `tcp_flags`
  - Ukázka zadání a výchozí hodnota programu:
    `eval_items = "dstip srcip"`
- *required_items*
  - Hodnoty, které se mají použít při tvorbě filtru z nalezeného vzoru útoku.
  - Podporované hodnoty: `srcip`, `dstip`, `prot`, `srcport`, `dstport`, `tcp_flags`
  - Ukázka zadání a výchozí hodnota programu:
    `required_items = "dstip srcip"`
- *db_insert_interval*
  - Čas v sekundách po jehož uplynutí vždy dojde ke vložení hodnot popisujících aktuální provoz do databáze.
  - Ukázka zadání a výchozí hodnota programu: `db_insert_interval = 60`
- *columns*
  - Výpis sloupců, které se použijí při tvorbě tabulky konkrétního *input-filtru* a zároveň výpis hodnot, které se budou do této tabulky vkládat. Hodnota *bps* má jednotku Bps (bajtů za sekundu).
  - Podporované hodnoty: `byte_baseline`, `packet_baseline`, `bps`, `pps`
  - Ukázka zadání a výchozí hodnota programu:
    `columns = "byte_baseline bps packet_baseline pps"`
- *active_filter_duration*
  - Určuje dobu v sekundách, po jakou budou aktivní vytvoření *active-filtry*.
  - Ukázka zadání a výchozí hodnota programu: `active_filter_duration = 300`
- *max_top_x*
  - Určuje maximální hodnotu *topX* použitou v algoritmu 1 hledání vzoru útoku.
  - Ukázka zadání a výchozí hodnota programu: `max_top_x = 1`

6.5.4 Systém určování výchozích hodnot

Kontext konfiguračního souboru se dá rozdělit na dva druhy. Jeden druh se vztahuje parametry ke konkrétnímu filtru a druhý dělá parametry globální. Názorná ukázka změny kontextu je možná vidět na obrázku 6.1. Na začátku čtení konfiguračního souboru je kontext nastaven na globální. Kontext se poté mění při každém nalezení nového textového zadání *input-filtru*.

Program rozeznává tři typy zadání hodnot parametrů pro filtry. Jeden typ jsou hodnoty zadány přímo v kontextu daného filtru, druhý typ jsou výchozí hodnoty z konfiguračního souboru a třetí typ jsou výchozí hodnoty programu.

Tyto typy mají takovouto prioritu:

- 1. Hodnota z kontextu filtru
- 2. Výchozí hodnota z konfiguračního souboru
- 3. Výchozí hodnota programu

```
#Povinné hodnoty
connection_string = "dbname=ndd user=ndd"
nfcapd_current = "/data/netflow/DDG/nfcapd.current.29629"

#Globální kontext
coefficient = 300

#Filter #1
filter = "src port 123"
#Kontext filtru 1
coefficient = 200

#Filter #2
filter = "src port 53"
#Kontext filtru 2
```

Obrázek 6.1: Názorná ukázka změny kontextu v konfiguračním souboru.

Na obrázku 6.1 je hodnota z kontextu filtru možná vidět u `Filter #1` a to konkrétně u parametru `coefficient = 200`. Výchozí hodnota z konfiguračního souboru je hodnota takového parametru, který byl zadán do globálního kontextu. Výchozí hodnota programu, je pevně daná hodnota použita pro parametry, které nebyly zadání v globálním kontextu ani v kontextu právě řešeného filtru.

Program po přečtení konfiguračního souboru z obrázku 6.1 by použil pro `Filter #1` hodnotu z kontextu filtru, tedy `coefficient = 200` a pro `Filter #2` by použil výchozí hodnotu z konfiguračního souboru, tedy `coefficient = 300`. Jelikož se v ukázce nenachází například parametr *baseline_window*, použila by se jak pro `Filter #1`, tak pro `Filter #2` výchozí hodnota z programu. Každý z parametrů doplňujících *input-filtr* má výchozí hodnotu programu. Tyto hodnoty je možné vidět u výpisu všech parametrů zde 6.5.3.

6.6 Spouštění programu

Pro spuštění programu je zapotřebí funkční PostgreSQL server, nastavený nfdump kolektor s příchozími daty a vyplnění `ndd.conf` konfigurační soubor. Program podporuje použití několika parametrů při spuštění.

Tyto parametry jsou:

- Parametr **-d** | Program se spustí jako daemon.
- Parametr **-p** | Program bude vypisovat dodatečné informace na stdout.
- Parametr **-s *hodnota*** | Program se vypne po ***hodnota*** zápisech *baseline* jednoho vlákna do databáze. Při použití s více filtry se program vypne po dosažení ***hodnota*** zápisů jakýmkoliv vláknem, nečeká se na tolik zápisů ostatních vláken. Absence tohoto parametru znamená, že program poběží po dobu neurčitou.
- Parametr **-l** | Program bude vytvářet logy do složky `./logs/`. Vytvoří soubor `ndd.err` určený chybovým výpisům, soubor `ndd.logs` pro všechny výpisy a `ndd.stats` pro výpis všech aktuálních hodnot programu (počty filtrů, jejich hodnoty, atd . . .).
- Parametr **-t** | Program po skončení smaže své tabulky filtrů z databáze.
- Parametr **-r** | Použití pro čtení konečného zdrojového souboru. Program bude číst do doby nalezení konce souboru a následně se vypne. Když se tento parametr nepoužije, program předpokládá, že čte ze souboru *nfcapd.current*, kde dochází k neustálému přísunu nových záznamů.
- Parametr **-m *hodnota*** | Umožňuje spustit program s více vstupními soubory najednou. Počet vstupních souborů je ***hodnota***. Použití pouze současně s parametrem **-r**. Program očekává ***hodnota*** krát položku *nfcapd_file* v konfiguračním souboru.

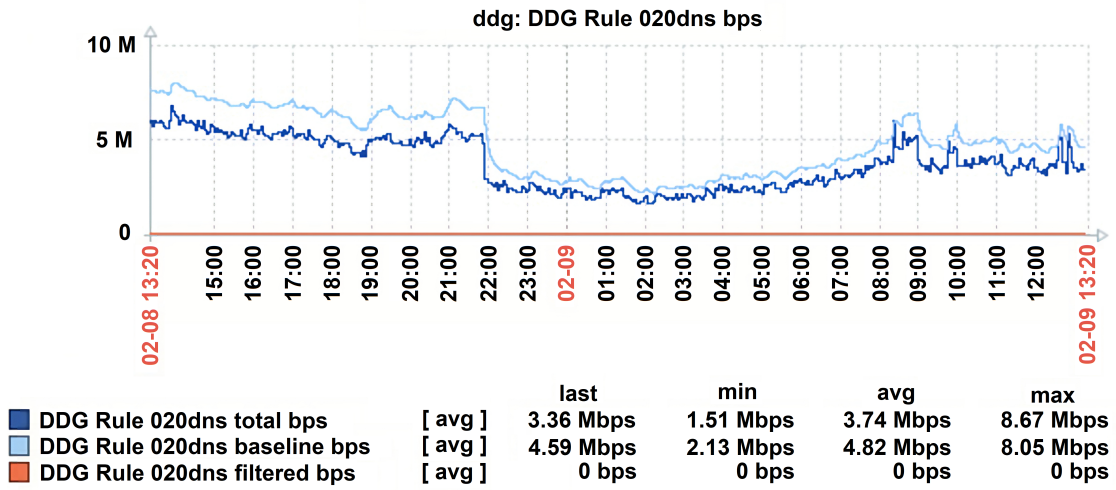
Kapitola 7

Testování

Testování proběhlo na školním serveru VUT s přístupem k NetFlow kolektoru získávajícím data z páteřní sítě VUT. V páteřní síti VUT se provoz pohybuje ve vysokých stovkách megabajtů za sekundu až jednotkách gigabajtů za sekundu, což vede ke generování desetitisíců NetFlow záznamů každou vteřinu.

7.1 Správnost hodnot popisujících provoz

Pro ujištění se o správnosti počítání hodnot popisujících provoz se tyto hodnoty porovnaly s hodnotami získanými z již aktivní ochrany sítě VUT. Tato ochrana zvaná DDoS Guard[24] je komponenta směrovačů NetX[23] umožňující detekci a následnou filtraci. Je to in-line ochrana fungující na stejném principu detekce jako vyvíjený program a to počítáním *baseline* a zkoumání její fluktuace. Proto je tato ochrana vhodným kandidátem k ujištění správnosti hodnot.



Obrázek 7.1: Ukázka získaných hodnot od DDoS Guard.

Na obrázku 7.1 je možné vidět graf znázorňující hodnoty *bps* (bity za sekundu) a *baseline* získané z ochrany DDoS Guard. Jelikož jsou hodnoty vyvíjeného programu počítány jako průměr za určitý čas, tak se porovnávané hodnoty nebudou přesně schodovat. Proto toto porovnání nezaručí naprostou správnost počítaných hodnot, ale velká podobnost po-

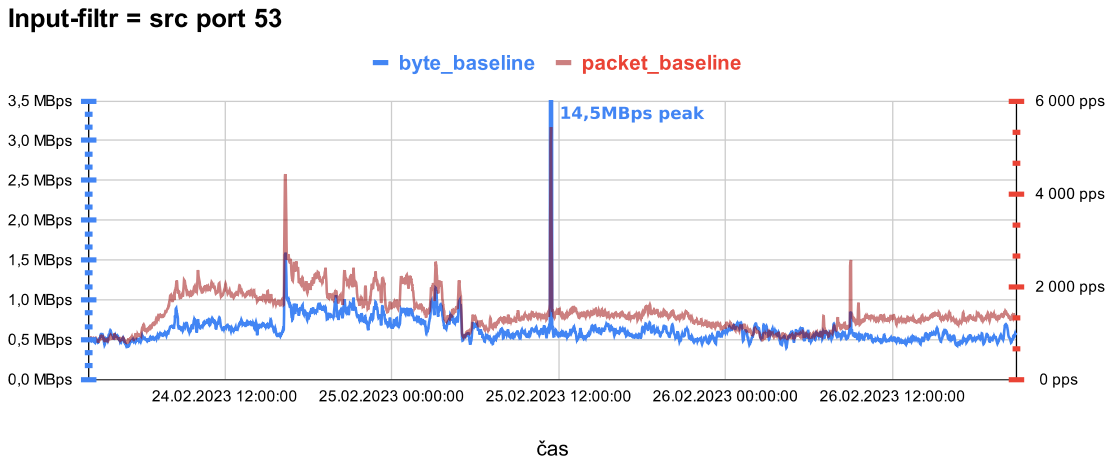
rovnávaných hodnot je dobrým znakem. Rozdílem mezi porovnávanými hodnotami je fakt, že vyvíjený program převážně pracuje s hodnotami v jednotkách bajtů oproti bitům. A to z důvodu, že záznamy předávané knihovnami obsahují informace o množství přenesených dat v bajtech.

Při konverzi na stejné jednotky a porovnání trendů hodnot vypočítaných vyvíjeným programem a získaných hodnot od DDoS Guard, byla zjištěna spojitost, jak v časech menšího tak i většího provozu.

7.2 Reakce *baseline* na zvýšení provozu

Při vývoji tohoto programu byla vždy nejnovější verze spuštěna na aktuálním provozu. Jelikož část programu počítající hodnoty popisující aktuální provoz byla jedna z prvních implementovaných, je z ní také nejvíce naměřených dat. Data se získávala v průběhu několika měsíců. Díky tomuto času a faktu, že je zkoumán provoz z větší sítě bylo zaznamenáno několik útoků. Vzhledem k dnešním trendům 4.5 byl k získávání dat vybrán konkrétní filtr zadaný takto: `src port 53`.

Největší programem zaznamenaný útok na tomto filtru byl zjištěn s naměřenou *baseline* pro bajty dosahující hodnoty 14,5 MBps (115 Mbps). Graf vývoje obou *baseline* kolem času útoku je znázorněn níže:



Obrázek 7.2: Ukázka zjištěného náhlého zvýšení *baseline*.

Konfigurace při zaznamenání tohoto útoku, byla následující: *Baseline* bajtů se počítala s použitým časovým oknem o velikosti 300 s, při ukládání hodnoty do databáze jednou za minutu a *coefficientem* s hodnotou 300.

Čas [hh:mm]	11:23	11:24	11:25	11:26	11:27	11:28	11:29	11:30	11:31
baseline [MBps]	6,7	9,7	13,1	14,0	14,5	11,4	7,5	5,3	3,0

Tabulka 7.1: Hodnoty *byte_baseline* v čase útoku.

V tabulce 7.1 jsou znázorněny přesné hodnoty *baseline* bajtů přímo v času útoku. Těsně před útokem se *baseline* pro bajty pohybovala kolem 0,7 MBps (5,3 Mbps) a zaznamenala

desetinásobné zvýšení oproti následující uložené hodnotě, která se zvýšila na 6,7 MBps (53,5 Mbps). Zvyšování pokračovalo i pro dalších pět uložených hodnot, poté následovalo zmenšení. Abnormální hodnoty *baseline* bylo možné vidět na devíti uložených hodnotách, u dalších přišlo navrácení k hodnotě kolem 0,7 MBps.

7.3 Úspěšnost nalezení vzoru útoku

Testování úspěšnosti nalezení vzoru útoků, bylo prováděno na principu volání programu se vstupními soubory, které obsahují nějaký útok. Testována byla schopnost nalezení korektního vzoru tohoto útoku.

7.3.1 Hledání vzoru útoku v ručně vytvořených souborech

Testování proběhlo spouštěním programu nad vytvořenými nfcapd soubory, které obsahují záznamy provozu přidané o simulovaný útok. Byl vytvořen nástroj, který kopíruje příchozí záznamy ze souboru *nfcapd.current* do nového souboru a po určitém času simuluje útok vložením určitého počtu nově vygenerovaných záznamů se specifickými zdrojovými IP adresami. Nový soubor obsahuje přibližně pět minut čistého provozu, po kterých je vložen krátký útok a dalších několik minut čistého provozu. Účelem je, aby program byl schopný během prvních minut vypočítat *baseline* čistého provozu a na konci se dokázal vrátit do normálního stavu. Takovýchto souborů bylo vytvořeno několik a to z důvodu otestování útoků s různým počtem zdrojových IP adres.

Bylo testováno, zda program detekuje útok a nalezne kompletní vzor tohoto útoku. Kompletním vzorem útoku, je v tomto případě myšleno nalezení všech zdrojových IP adres útoku vygenerovaných do vytvořeného souboru. Při testování docházelo také k měření času, za jakou dobu program nalezne kompletní vzor po první detekci útoku.

Počet ip	***Baseline* před útokem**	***Baseline* po útoku**	**Provoz z jedné ip**
1	862 966 Bps	15 479 278 Bps	1,4 GBps
5	715 248 Bps	15 425 070 Bps	279,7 MBps
10	622 860 Bps	15 283 823 Bps	139,9 MBps
50	713 398 Bps	15 340 196 Bps	28 MBps
100	672 588 Bps	15 397 722 Bps	14 MBps
250	490 170 Bps	15 135 334 Bps	5,6 MBps
250a	425 783 Bps	59 072 662 Bps	22,4 MBps

Tabulka 7.2: Hodnoty vygenerovaných útoků.

Všechny simulované útoky mají stejnou sílu, pouze se liší v počtu zdrojů. Čím větší počet zdrojů, tím je útok více rozprostřen a jednotlivý zdroj má tím menší samostatný provoz. K provedení simulace každého útoku bylo k čistému provozu přidáno 4,3 miliónů záznamů, kde každý záznam obsahoval informace o 1 paketu o velikosti 1024 bajtů. Pokud by toto byl reálný útok, bylo by záznamů daleko méně z důvodu jejich agregace exportérem před odesláním na kolektor. Takové velké množství záznamů bylo záměrné a určeno pro otestování práce s velkými počty záznamů programu. Výsledný simulovaný útok měl sílu přenesení přibližně 4 GB během tří sekund (přibližně 12 Gbps). Hodnoty těchto vygenerovaných útoků je možné vidět v tabulce 7.2. Tabulka obsahuje orientační hodnoty *baseline* před útokem a po útoku. Je důležité zmínit, že program po detekci útoku hodnotu *baseline*

nepočítá, aby zabránil jejímu nafouknutí a před pokračování počítání čeká, než se provoz znovu stabilizuje. Hodnoty ***Baseline* po útoku** znázorněné v tabulce jsou orientační, jak by *baseline* vypadala, kdyby jí program během útoku počítal.

V tabulce 7.2 je možné vidět i větší simulovaný útok, než které byly popisované výše. Tento útok byl vygenerován, aby poukázal na fakt, proč je útok s 250 zdroji příliš pomalý, tento fakt je popsán níže. Naměřené časy měřené od první detekce k nalezení všech zdrojů jednotlivých útoků je možné vidět v tabulce 7.3. Je patrné, že tyto naměřené časy s výjimkou malých počtů zdrojů jsou neakceptovatelné. Pro tří sekundový útok trvá nalezení vzoru tohoto útoku více jak šest minut. Je fakt, že simulovaný útok s takto velkým počtem záznamů, je výkonnostně náročnější, než reálný obdobný útok, i přesto, ale tento test značně znázorňuje nedostatky tohoto programu.

Počet ip	1	5	10	50	100	250	250a
Čas	0,927 s	1,808 s	8,574 s	17,673 s	62,827 s	408,440 s	153,974 s

Tabulka 7.3: Naměřené časy od první detekce do nalezení kompletního vzoru útoku.

Program byl následně podrobně zkoumán s cílem nalezení největších bodů zpomalujících hledání vzoru útoku. Jeden velký faktor byl nalezen u testování s počtem zdrojů 250. Bylo zde zjištěno, že pro naprostou většinu nalezených zdrojů útoku musel program při hledání vzoru zmenšit hodnotu *threshold2*, až na její minimum. Toto vede k tomu, že u hledání každého jednotlivého zdroje prochází algoritmem tou nejdelší možnou cestou. Míru zpomalení tohoto faktu bylo možné dokázat vygenerováním nového útoku s 250 zdroji, který se nazval 250a a byl čtyřikrát větší oproti ostatním útokům. Při větším provozu z jednotlivých zdrojů je možné u naměřené hodnoty vidět značné zlepšení o více jak polovinu měřeného času. K nalezení zdrojů i s menším provozem by se program mohl pokoušet o agregaci IP adres pomocí masek sítě, efektivně by pak program nehledal jednotlivé zdroje s jednou IP adresou, ale celou IP adresu sítě, která obsahuje značný provoz.

Byl nalezen největší bod zpomalení programu při hledání vzoru útoku. Tento bod byl u kopírování záznamů *datasetu* z jedné struktury *lnf_mem_t* do druhé. Hlavní problém byl u filtrace záznamů během tohoto procesu. Po úspěšném dokončení jedné iterace algoritmu se nalezený vzor skládá z jedné zdrojové IP adresy, pro kterou je následně vytvořen *active-filtr*. Po nalezení 249 zdrojových IP adres a hledání poslední adresy, je vytvořených 249 *active-filtrů*. U každého zkopírovaného záznamu z jedné struktury paměti do druhé platí, že se musí zkontrolovat vůči všem 249 *active-filtrům* a pár dalším *aggr-filtrům*. Přidání kontroly určující, podle které skupiny filtrů se bude filtrovat jako první, vedlo k okamžitému zlepšení naměřeného času u 250 zdrojů o přibližně 120 s. Tato změna vedla k tomu, že porovnávání všech 249 *active-filtrů* dochází pouze v případě, že aktuálně kopírovaný záznam již úspěšně prošel kontrolou všech *aggr-filtrů*. Jako úplné řešení tohoto problému byla navrhnuta možná tvorba *active-filtrů* obsahujících více zdrojových IP adres. Obecně řečeno: bylo navrhnuto použití *topX* místo *top1* v určitých situacích běhu algoritmu. Pro možnou změnu *topX* byl přidán nový parametr v konfiguračním souboru.

Ukázka tvaru *active-filtru* vytvořeného s $topX > 1$:

```
src port 53 AND dst ip 1.1.1.1 AND (src ip 2.2.2.2 OR src ip 3.3.3.3)
```

Po implementaci použití *topX* a několika dalších malých úprav, byly naměřeny nové hodnoty času od první detekce do nalezení všech zdrojových IP adres útoku. Tyto hodnoty

Počet ip	1	5	10	50	100	250	250a
Staré časy	0,927 s	1,808 s	8,574 s	17,673 s	62,827 s	408,440 s	153,974 s
Nové časy	0,273 s	1,401 s	0,316 s	0,642 s	1,438 s	8,941 s	1,987 s

Tabulka 7.4: Naměřené časy upraveného algoritmu od první detekce do nalezení kompletního vzoru útoku.

jsou zobrazeny v tabulce 7.4 Při tomto měření bylo použito využití *topX* pro poslední *eval_item* s hodnotou $x = 1000$, to znamená, že se všem vygenerovaným útokům vejdou všechny zdrojové IP adresy do jednoho jediného *active-filtru*.

Po určitých úpravách programu dochází u nových časů mimo jiné také k dřívější detekci. U porovnávání útoků 250 a 250a je stále možné vidět značný rozdíl, tento rozdíl je způsoben tím, že u útoku 250 dochází k několika neúspěšným pokusům o nalezení vzoru, následovanými zpracováním více záznamů vlákna *input-filtru*, což vede ke zvětšení *datasetu* a patřičnému zjednodušení hledání vzoru útoku. U útoku 250a jsou provozy jednotlivých zdrojů větší, tudíž je zapotřebí menší *dataset* k jejich detekci.

7.3.2 Hledání vzoru útoku s pomocí testovacího prostředí

Dalším testem, bylo nalezení vzoru útoku ze souborů vytvořených přímo nfcapd kolektorem. Způsobem vložení útoku do těchto souborů, bylo exportování NetFlow záznamů z testovacího prostředí. Díky tomuto faktu, je možné shromažďovat záznamy o útocích uskutečněných v testovacím prostředí.

Byl vybrán útok s velkým počtem zdrojových IP adres s malým provozem jednotlivců. Při prvotních pokusech nalezení vzoru, bylo zjištěno, že jednotlivci mají moc malý provoz pro detekci i při maximálním ponížení hodnoty *threshold2*. Proto byla přidána implementace agregace podle IP adres sítí s různou délkou masky sítě. Agregace podle síťových IP adres se použije v případě, kdy aktuální *eval_item* `srcip` nebyl nalezen ani jeden *topX*. Při první agregaci se použije maska sítě `/24` pro IPv4 a `/64` pro IPv6, pokud ani v tomto kroku nebyl nalezen *topX* je maska ještě jednou snížena na hodnoty `/16` pro IPv4 a `/48` pro IPv6. Když se stane, že ani pro tyto masky sítě není nalezen *topX*, pokračuje se dál v algoritmu bez přidání `srcip` do *filter_candidate*.

Ukázka tvaru *active-filtru* vytvořeného s *topX* agregovaném podle IP adresy sítě:

```
src port 53 AND dst ip 1.1.1.1 AND src net 2.2.2.0 255.255.255.0
```

7.3.3 Hledání vzoru v reálném útoku

V průběhu tvorby této práce byl zaznamenán a mitigován útok na síť VUT. Ochrana sítě VUT DDoS Guard úspěšně útok detekovala a nalezla jeho vzor.

Nalezený vzor ochranou DDoS Guard:

Src IP	89.248.163.59
Dst IP	147.229. * . *
Src port	45312 - 45567
Dst port	....
TCPflags	SYN

Tento útok byl typu SYN flood a byl mířený na větší množství cílových IP adres z IP rozsahu VUT. Útok byl o velikosti přibližně 150 Mbps a 450 tisíc paketů za sekundu. Vytvořený filtr vůči tomuto útoku byl na ochraně DDoS Guard aktivní po dobu tří minut.

Pro otestování vytvořeného programu byl program spuštěn na nfcapd souborech obsahujících jednu hodinu provozu, kterého součástí je onen zmíněný útok. Důležité použité parametry *input-filtru*:

```
filter = "flags S and not flags AFRPU"
eval_items = "srcip dstip srcport dstport"
required_items = "srcip"
top_x = 10
```

Celkové trvání programu pro přezkoumání hodinového provozu trvalo lehce pod čtyři minuty, s časem od první detekce útoku do nalezení jeho vzoru přibližně čtyři vteřiny. Program nalezl tento vzor:

```
flags S and not flags AFRPU AND src ip ::89.248.163.59 AND -
- dst net 147.229.0.0 255.255.0.0 AND (src port 45441 OR src port 45444)
```

Po vytvoření *active-filtru* začalo hlavní vlákno vyfiltrovávat příslušný provoz útoku. Celkové množství takto vyfiltrovaných dat činilo 10,5 miliónů paketů o celkové velikosti 3,1 GB. Ostatní nevyfiltrovaná data útoku byly použity v rámci detekce a nalezení vzoru útoku, nebo byly již vloženy do paměti, ale ještě nebyly zpracovány vláknem *input-filtru*.

Po porovnání nalezeného vzoru programu se vzorem z ochrany DDoS Guard je možné vidět, že program úspěšně nalezl korektní vzor. Liší se pouze v hodnotách zdrojových portů. Program do svého výsledného vzoru zahrnul pouze dva zdrojové porty, které obsahovali většinu provozu, ostatní použité porty obsažené ve vzoru DDoS Guard neobsahovali pro program dostatečné množství přenesených dat pro jejich zahrnutí do výsledného vzoru.

Kapitola 8

Závěr

Práce se zabývala tématem detekce a nalezení vzoru útoku pomocí zkoumání NetFlow dat popisujících provoz sítě. Způsob zkoumání a nástroje pro něj použité byly teoreticky popsány a následně použity k vytvoření výsledného programu s interním označením `ndd`. Zdrojové soubory byly průběžně umísťovány na veřejné úložiště GitHub[1].

Program funguje na bázi počítání *baseline* z blízké minulosti provozu a porovnáváním ho s aktuálním provozem v síti. Potřebná data získává z NetFlow záznamů pomocí knihovny libnf, která je napojena na nfcapd NetFlow kolektor. Informace důležitého charakteru, nebo užitečné informace jsou ukládány do PostgreSQL databáze. Detekce útoku nastává při náhlém zvýšení aktuálního provozu vůči hodnotě vypočítané z předešlé *baseline*. Při detekci se provede algoritmus nalezení vzoru útoku z *datasetu*, kde *dataset* jsou dočasně ukládané záznamy, při běžném běhu provozu, k dispozici pro případnou detekci útoku. Algoritmus funguje na opakované agregaci *datasetu* podle hodnot zadaných administrátorem a hledání konkrétní část provozu překračující vypočítaný limit. Administrátor má k dispozici řadu měnitelných parametrů v konfiguračním souboru, které mu umožňují lépe používat tento program jako ochranu vůči specifickým útokům. Vzhledem ke komplexnosti problematiky síťových útoků se očekává znalost administrátora, vzhledem k velkému počtu měnitelných parametrů je jejich nastavení pro správné fungování programu klíčové.

Implementace a testování proběhla na školním serveru s dostupnými NetFlow daty z páteřní sítě VUT. Program byl porovnáván s již funkční ochranou VUT sítě s názvem DDoS Guard. Při implementaci programu byla zjištěna chyba v knihovně libnf, která způsobovala neustálé zvyšování velikosti alokované paměti. Tato chyba byla při nalezení reportována týmu libnf, který ji následně opravil ve verzi knihovny 1.32.

Testování detekce útoků bylo provedeno několika měsíců dlouhým spuštěním nad daty ze sítě VUT, jelikož je tato síť nemalá, došlo k několika detekcím útoků. Správnost nalezení vzoru útoku se testovala na základě vygenerovaných nfcapd souborů. Tyto soubory obsahovali několik minut normálního provozu, do kterého byly přidány vygenerované záznamy simulující útok. Souborů se vytvořilo několik, kde každý soubor měl jiný počet zdrojových IP adres útoku. Správnost se dokázala úspěšným nalezením všech simulovaných útoků. Dalším testem nalezení vzoru útoku bylo zkoumání nfcapd souborů, obsahujících informace exportované z testovacího prostředí. Toto umožnilo zkoušet hledání vzoru útoku nad daty o reálně provedených útocích v testovacím prostředí.

V první iteraci programu došlo k výkonnostním problémům. Tyto problémy byly pro konkrétní případy vyřešeny, ale vzhledem ke komplexnosti tématiky síťových útoků a neu-

[1] Odkaz: https://github.com/Methack/DP-Netflow-ddos

stálé tvorbě nových útoků, nelze zaručit naprostou neprůstřelnost tohoto programu. Stejně tak nelze zaručit, že výchozí hodnoty konfiguračního souboru jsou ideální v každém možném případě. Při používání tohoto programu se očekává znalost administrátora a jistá časová investice pro co nejlepší přizpůsobení programu konkrétnímu prostředí.

Literatura

[1] Aitken, P., Claise, B. a Trammell, B. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* [RFC 7011]. RFC Editor, září 2013. DOI: 10.17487/RFC7011. Dostupné z: https://rfc-editor.org/rfc/rfc7011.txt.

[2] Amazon. *Threat Landscape Report – Q1 2020* [online]. Duben 2020. Dostupné z: https://aws-shield-tlr.s3.amazonaws.com/2020-Q1_AWS_Shield_TLR.pdf.

[3] Arteaga, J. a Mejia, W. *CLDAP Reflection DDoS* [online]. Duben 2017. Dostupné z: https://www.akamai.com/our-thinking/threat-advisories/cldap-reflection-ddos.

[4] Birchard, D. *DDoS Attacks in 2022: Targeting Everything Online, All at Once* [online]. Březen 2023. Dostupné z: https://www.akamai.com/blog/security/ddos-attacks-in-2022-targeting-everything-online.

[5] Cisco. *NetFlow Export Datagram Format* [online]. Cisco, září 2007. Dostupné z: https://www.cisco.com/c/en/us/td/docs/net_mgmt/netflow_collection_engine/3-6/user/guide/format.html.

[6] Cisco. *NetFlow Version 9 Flow-Record Format* [online]. Cisco, duben 2011. Dostupné z: https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html.

[7] Cisco. *Security Configuration Guide: Access Control Lists* [online]. 2017. Dostupné z: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_data_acl/configuration/15-mt/sec-data-acl-15-mt-book.html.

[8] Claise, B. *Cisco Systems NetFlow Services Export Version 9* [RFC 3954]. RFC Editor, říjen 2004. DOI: 10.17487/RFC3954. Dostupné z: https://rfc-editor.org/rfc/rfc3954.txt.

[9] Darknet. *Low Orbit Ion Cannon DDoS Booter* [online]. Říjen 2017. Dostupné z: https://www.darknet.org.uk/2017/10/loic-download-low-orbit-ion-cannon-ddos-booter.

[10] Dzurenda, P., Martinasek, Z. a Malina, L. Network protection against DDoS attacks. *International Journal of Advances in Telecommunications, Electrotechnics, Signals and Systems*. 2015, sv. 4, č. 1, s. 8–14.

[11] Emmons, T. *Largest Ever Recorded Packet Per Second-Based DDoS Attack Mitigated by Akamai* [online]. Červen 2020. Dostupné z: https://www.akamai.com/blog/news/largest-ever-recorded-packet-per-secondbased-ddos-attack-mitigated-by-akamai.

[12] FastNetMon. *Product overview FastNetMon Advanced* [online]. 2018. Dostupné z: https://fastnetmon.com/product-overview/.

[13] Haag, P. *Nfdump* [online]. Říjen 2015. Dostupné z: https://github.com/phaag/nfdump.

[14] Iana. *IP Flow Information Export (IPFIX) Entities* [online]. Iana, květen 2007. Dostupné z: https://www.iana.org/assignments/ipfix/ipfix.xhtml.

[15] *Internet Protocol* [RFC 791]. RFC Editor, září 1981. DOI: 10.17487/RFC0791. Dostupné z: https://rfc-editor.org/rfc/rfc791.txt.

[16] King, T., Dietzel, C., Snijders, J., Döring, G. a Hankins, G. *BLACKHOLE Community* [RFC 7999]. RFC Editor, říjen 2016. DOI: 10.17487/RFC7999. Dostupné z: https://www.rfc-editor.org/info/rfc7999.

[17] Makrushin, D. *The cost of launching a DDoS attack* [online]. Březen 2017. Dostupné z: https://securelist.com/the-cost-of-launching-a-ddos-attack/77784/.

[18] Marques, P. R., Mauch, J., Sheth, N., Greene, B., Raszuk, R. et al. *Dissemination of Flow Specification Rules* [RFC 5575]. RFC Editor, srpen 2009. DOI: 10.17487/RFC5575. Dostupné z: https://www.rfc-editor.org/info/rfc5575.

[19] MaxMind. *GeoIP2 and GeoLite2 Database Documentation* [online]. Leden 2012. Dostupné z: https://dev.maxmind.com/geoip/docs/databases.

[20] Mirkovic, J. a Reiher, P. *A Taxonomy of DDoS Attack and DDoS Defense Mechanisms*. New York, NY, USA: Association for Computing Machinery, apr 2004. DOI: 10.1145/997150.997156. ISSN 0146-4833. Dostupné z: https://doi.org/10.1145/997150.997156.

[21] NETX AS. *Libnf* [online]. Červen 2014. Dostupné z: https://github.com/NETX-AS/libnf.

[22] NETX AS. *Libnf documentation* [online]. Červenec 2015. Dostupné z: http://libnf.net/doc/api/.

[23] NETX AS. *NETX Smart router series* [online]. Září 2018. Dostupné z: https://netx.as/.

[24] NETX AS. *DDoS Guard* [online]. Březen 2021. Dostupné z: https://docs.netx.as/docs/ddos/ddos-guard.html.

[25] ntop. *NScrub* [online]. 2018. Dostupné z: https://www.ntop.org/products/ddos-mitigation/nscrub/.

[26] Peering.cz. *Route Server Control* [online]. Leden 2020. Dostupné z: https://client.peering.cz/pdf/route-server_documentation.pdf.

[27] Podermański, T., Grégr, M. a Švéda, M. *Application Programming Interface for Advanced Processing of NetFlow Data*. 2013.

[28] PostgreSQL Global Development Group. *PostgreSQL: The World's Most Advanced Open Source Relational Database* [online]. 1996. Dostupné z: https://www.postgresql.org/.

[29] RIPE. *Routing Information Service (RIS)* [online]. Březen 2015. Dostupné z: https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris.

[30] Srivastava, A., Gupta, B., Tyagi, A., Sharma, A. a Mishra, A. A recent survey on DDoS attacks and defense mechanisms. In: Springer. *International Conference on Parallel Distributed Computing Technologies and Applications*. 2011, s. 570–580.

[31] Tomac a Slay. *Yersinia* [online]. 2017. Dostupné z: https://github.com/tomac/yersinia.

[32] Yoachimik, O. *Cloudflare blocks an almost 2 Tbps multi-vector DDoS attack* [online]. Listopad 2021. Dostupné z: https://blog.cloudflare.com/cloudflare-blocks-an-almost-2-tbps-multi-vector-ddos-attack/.

[33] Yoachimik, O. a Pacheco, J. *DDoS threat report for 2023 Q1* [online]. Duben 2023. Dostupné z: https://blog.cloudflare.com/ddos-threat-report-2023-q1/.

[34] Česko. *Vyhláška č. 357/2012 Sb. o uchovávání, předávání a likvidaci provozních a lokalizačních údajů*. 2012.

Příloha A

Obsah paměťového média

Součástí paměťového média, přiloženého k této diplomové práci, jsou zdrojové soubory LaTeXpro jeho přeložení, jíž výsledný text práce ve formátu PDF, zdrojové soubory vytvořeného programu s Makefile souborem pro přeložení a soubor README.txt obsahující doplňující informace.

Paměťové médium má tuto strukturu:

|- latex/	Složka obsahující zdrojové soubory latex.
|- ndd/	Obsahuje přeložený program a konfigurační soubor
|- ndd.conf	Konfigurační soubor.
|- ndd	Přeložený program, připraven pro spuštění.
|- src/	Obsahuje zdrojové soubory programu.
|- Makefile	Soubor pro překlad za použití příkazu *make*.
|- comm.c	Implementace logování programu.
|- comm.h	Hlavičkový soubor pro comm.c.
|- config.c	Scanner a parser konfiguračního souboru.
|- config.h	Hlavičkový soubor pro config.c
|- db.c	Funkce pro přístup k databázi.
|- db.h	Hlavičkový soubor pro db.c
|- main.c	Obsahuje funkci main.
|- ndd.c	Hlavní smyčky pro práci se záznamy.
|- ndd.h	Hlavičkový soubor pro ndd.c.
|- nddstruct.c	Základní funkce pro struktury.
|- nddstruct.h	Hlavičkový soubor, obsahuje definice struktur.
|- README.txt	Soubor obsahující doplňující informace.
|- xjires02.pdf	Text diplomové práce.