Zadání bakalářské práce

Název:	Detekce WireGuard provozu pomocí Active Learning
Student:	Štěpán Jílek
Vedoucí:	Ing. Dominik Soukup
Studijní program:	Informatika
Obor / specializace:	Bezpečnost a informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Prostudujte aktuální technologie pro monitorování síťového provozu a bezpečnostní analýzu. Zároveň se zaměřte na využití tzv. rozšířených IP toků.
Dále se seznamte s konceptem Active Learning a technologií Active Learning Framework (ALF) [1,2], která slouží k nasazení, automatickému přetrénování a vyhodnocení modelů strojového učení.
Navrhněte sadu možných způsobů anotace (indikátorů) pozorovaného WireGuard (WG) provozu.
Vytvořte konfigurovatelný modul Anotátor pro automatické vytváření datové sady WG na základě navržených indikátorů. Vytvořená datová sada bude sloužit k bezpečnostnímu monitoringu WG provozu.
Ve spolupráci s vedoucím práce k vytvořenému Anotátoru vytvořte klasifikátor a nasaďte v rámci ALF.
Dále ve spolupráci s vedoucím nasaďte vytvořený systém do provozu reálné sítě a vyhodnoťte úspěšnost natrénovaného klasifikátoru s vytvořeným anotátorem prostřednictvím ALF.

[1] Jaroslav Pešek: Framework pro automatické zlepšování klasifikace síťového provozu, Master thesis, Faculty of information technology, CTU in Prague, 2022.
[2] Pešek, J.; Soukup, D.; Čejka, T. Vision of Active Learning Framework Approach to Network Traffic Analysis Research In: Proceedings of the 10th Prague Embedded Systems Workshop. Praha: CTU. Faculty of Information Technology, 2022. ISBN 978-80-01-07015-4.

Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 30. září 2022 v Praze.

Bakalářská práce

DETEKCE WIREGUARD PROVOZU POMOCÍ ACTIVE LEARNING

Štěpán Jílek

Fakulta informačních technologií
Katedra informační bezpečnosti
Vedoucí: Ing. Dominik Soukup
11. května 2023

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2023 Štěpán Jílek. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Jílek Štěpán. *Detekce WireGuard provozu pomocí Active Learning.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Obsah

Poděkování vi

Prohlášení vii

Abstrakt viii

Seznam zkratek ix

Úvod 1

1 Rešerse 3
1.1 Počítačové sítě . . . 3
1.1.1 Principy počítačových sítí . . . 3
1.1.2 Monitorování počítačových sítí . . . 4
1.1.3 Síťové toky . . . 5
1.1.4 IPFIX . . . 5
1.1.5 UniRec . . . 7
1.1.6 NEMEA . . . 7
1.1.7 Analýza paketů . . . 8
1.1.8 Protokoly VPN . . . 9
1.1.9 Protokol WireGuard . . . 10
1.2 Strojové učení . . . 11
1.3 ALF framework . . . 12

2 Analýza 13
2.1 Obecný anotátor . . . 13
2.2 Metody anotace WireGuard . . . 13
2.2.1 Protokol . . . 14
2.2.2 Číslo portu . . . 14
2.2.3 Blocklist IP . . . 14
2.2.4 Data uvnitř paketu . . . 14
2.2.5 Aktivní dotaz . . . 15
2.2.6 Kontrola log souboru . . . 16
2.2.7 Párování paketů . . . 16

3 Návrh 17
3.1 Požadavky . . . 17
3.1.1 Nefunkční požadavky . . . 17
3.1.2 Funkční požadavky . . . 18
3.2 Architektura . . . 18

4 Implementace 21
4.1 Obecně . . . 21
4.2 Komunikační část . . . 22
4.3 Databáze . . . 23
4.4 Anotace . . . 23
4.4.1 BasicAnotator . . . 24
4.4.2 PayloadAnotator . . . 24
4.4.3 LogAnotator . . . 25
4.4.4 PairAnotatorDominantLabel . . . 25
4.4.5 RequestAnotator . . . 25
4.5 Třída Core . . . 25
4.5.1 Konfigurační soubor . . . 26
4.6 Soubor parser.py . . . 29
4.7 Anotator_ctl.py . . . 29
4.8 Komunikace s ALF . . . 30

5 Vyhodnocení 31
5.1 Testovací prostředí . . . 31
5.2 BasicAnotator . . . 31
5.3 PayloadAnotator . . . 32
5.4 LogAnotator . . . 32
5.5 PairAnotatorDominantLabel . . . 32
5.6 RequestAnotator . . . 32
5.7 Klasifikace . . . 32

6 Závěr 35
6.1 Budoucí práce . . . 35

A Instalační příručka 37

Obsah přiloženého média 43

Seznam obrázků

1.1 Zapouzdření WireGuard na jednotlivých vrstvách 4
1.2 Princip fungování NAT 4
1.3 Komplexní systém monitorování pomocí síťových toků [9] 8
1.4 Architektura systému NEMEA [26] 8
1.5 Diagram VPN 9

2.1 Navázání komunikace WireGuard [32] 15
2.2 Handshake Initiation WireGuard [32] 16
2.3 Handshake Response WireGuard [32] 16

3.1 Diagram zpracování 18
3.2 Diagram zpracování rozdělen do tříd 19
3.3 Architektura obecného anotátoru 19

4.1 Architektura obecného anotátoru s rozepsanými metodami 21

Seznam tabulek

Seznam výpisů kódu

1.1 Ukázka IPFIX zprávy [22] 6
1.2 Ukázka IPFIX hlavičky [22] 6
1.3 Ukázka UniRec šablony [3] 7
4.1 Konstruktor třídy Link 22
4.2 Ukázka konfiguračního souboru pro třídu Log Anotator 25
4.3 Ukázkový konfigurační soubor Basic Anotator 26
4.4 Konfigurační soubor Log Anotator 27
4.5 Konfigurační soubor pro DoH 28

Děkuji především svému vedoucímu práce Ing. Dominiku Soukupovi za pomoc a cenné rady při tvorbě této práce. Dále chci poděkovat za pomoc kolegům z laboratoře monitorování síťového provozu jmenovitě Ing. Danielu Uhříčkovi, Filipu Němcovi a Ing. Karlu Hynkovi. Také chci poděkovat své rodině za podporu při tvorbě této práce a během studia

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. května 2023 ....

Abstrakt

Tato bakalářská práce se zabývá problematikou detekce šifrovaného provozu protokolu WireGuard, který se používá pro šifrované VPN spojení. K detekci je využit framework Active Learning Framework, který využívá výhod strojového učení a který je vylepšen o obecný modul Anotátor. Využitím tohoto frameworku vznikne jednoduchá datová sada pro natrénování modelu. Vedlejším cílem práce je vytvořit takový modul Anotátor, který půjde snadno upravit i na jiné protokoly.

V první části je čtenář seznámen s principy monitorování sítí a protokolem WireGuard. V dalších částech pak návrhem a implementací. Na konci je vyhodnocena výkonnost a přesnost detekce. Výkonost klasifikace dosahuje v mnoha případech až 95%

Klíčová slova strojové učení, klasifikace síťového provozu, WireGuard, ALF, VPN, IPFIX, aktivní učení, anotace

Abstract

This bachelor's thesis deals with the detection of encrypted traffic of the WireGuard protocol, which is used for encrypted VPN connections. This protocol is only few years old and is rapidly growing in popularity. The ALF framework is used for detection which takes advantage of machine learning and is improved by the universal Annotator module.

In the first part, the reader is introduced to the principles of network monitoring and the WireGuard protocol. In the next parts is the implementation proposal. At the end, the performance and accuracy of the detection is evaluated.

Keywords machine learning, network traffic classification, WireGuard, VPN, IPFIX, active learning, anotation

Seznam zkratek

ALF	Active Learning Framework
WG	WireGuard
IP	Internet Protocol
MAC	Media Access Control address
UR	Unirec
VPN	Virtual Private Network
NEMEA	Network Measurements Analysis
CESNET	Czech Education and Scientific NETwork
DNS	Domain Name System
DoH	DNS over TLS
IPFIX	IP Flow Information Export
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
TLS	Transport Layer Security
SSD	Solid State Drive
Gbit/s	Gigabity za sekundu
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
NAT	Network Address Translation
API	Application Programming Interface
TRAP	Traffic Analysis Platform
XML	Extensible Markup Language
OS	Operační Systém
HW	Hardware
SW	Software
IFC	Communication Interface

Úvod

Doba, kdy byl šifrovaný síťový provoz vyjímečný či nedostatečně zabezpečený, je již pryč. Nyní je již většina provozu šifrovaná [1]. Stejně tak v posledních letech velmi roste obliba využívání virtálních sítí (VPN). Ať je to k firemním účelům a nakládání se vzdálenými daty, tak je to i anonymizace činností ve veřejném online světě. Společně s tím je v dnešní době maskována šifrováním i nelegální či škodlivá činnost [2]. Dostat se k původním datům, která byla zašifrována je z podstaty šifrování nemožné. Jsou zde ale stále možnosti, jak s určitou pravděpodobností zjistit, jaká činnost se za šifrováním skrývá.

Existuje mnoho způsobů, jak zpracovávat síťová data. Jeden z přístupů je deterministické zpracování. Tento způsob zpracovávání je výkonově i datově náročný. Tento přístup také komplikuje tunelování a šifrování originální síťové komunikace. V této práci se zaměřím na zpracování pomocí strojového učení. Strojové učení je podobor umělé inteligence. Je to soubor algoritmů, které dokáží předpovídat jevy a pomocí trénování svou předpověď zlepšovat. Ke své predikci využívají metody z oblastí matematické statistiky a statistické analýzy. Tyto algoritmy na počátku potřebují datovou sadu na natrénování. Sada musí být co nejobsáhlejší, ale příliš velká datová sada může mít vliv na rychlost a paměťové nároky. Proto je vhodné v datové sadě nemít duplicitní či velmi podobná data. Tvorbou datových sad se věnuje diplomová práce Jaroslava Peška [3], ve které byl vytvořen framework ALF se kterým v této bakalářské práci budu pracovat. Celý framework je funkční, ale je složité ho upravovat na jednotlivé protokoly a druhy provozu. Detekci síťových jevů zde provádí komponenta anotátor. Cílem této práce je navrhnout obecný konfigurovatelný modul anotátor, který bude snadno nastavitelný na různé sledované protokoly. Jako konkrétní příklad detekce bude použit protokol WireGuard [4].

Protokol WireGuard [4] se používá na vzdálené připojení VPN. Oproti ostatním protokolům popsaných blíže v této práci má mnoho výhod. Je jednoduchý, bezpečný a výkonný [5]. Je momentálně již široce podporován mezi různými operačními systémy. Mnoho anonymizačních služeb poskytujících skrytí opravdové polohy pomocí VPN již přešla na tento protokol (třeba společnost Surfshark [6] nebo NordVPN [7])

Vytvořený anotátor společně s již vytvořeným frameworkem je určen na místa, kde je potřeba rychle detekovat různé anomálie v síťovém provozu a kde jsou velké objemy přenášených dat. Jsou to datacentra, poskytovatelé internetového připojení či internetová přípojka pro střední a velké firmy. Do míst, kde je síťový provoz malý se tato technologie nehodí z důvodu malé různorodosti dat a také z důvodu režijních nákladů s celým frameworkem, sběrem a uchováváním dat. Pro společnost tato práce může mít přínos v podobě detekce různých různých útoků, které útočník chce skrýt šifrovaním

Téma této práce jsem zvolil z důvodu, že protokol WireGuard [4] je stále dost mladý a je zde velký prostor pro jeho průzkum. Zároveň je detekce tohoto protokolu těžší než u jiných VPN protokolů. Podobné téma, které by se zabývalo využitím strojového učení k detekování tohoto protokolu, jsem v Česku nenašel. V mezinárodních databázích jsem na jisté experimenty

narazil [8], které ale byly zaměřeny obecné na VPN protokoly a protokolu WireGuard nebyla věnována takové pozornost. Rozhodl jsem se proto zjistit, jaké mohou být reálné výsledky pomocí nástrojů vyvinutých sdružením CESNET [9, 10, 11].

Na začátku této práce se budu věnovat vysvětlení monitorování počítačových sítí, popisu základního fungování protokolu WireGuard [4] a frameworku z výše uvedené diplomové práce. V dalších kapitolách to bude softwarový návrh obecného anotátoru a komponent, ze kterých se skládá. V závěru práce se zaměřím na implementaci a vyhodnocení anotace a výkonosti strojového učení.

Kapitola 1
Rešerse

V této kapitole vysvětlím stávající fungování počítačových sítí a technologie pro monitorování síťového provozu. Dále se zaměřím na fungování protokolu WireGuard [4], strojové učení a framework pro vytváření datových sad Active Learning Framework (ALF) [3].

1.1 Počítačové sítě

Již s příchodem prvních počítačů vznikla i potřeba je mezi sebou propojovat. Pomocí různých protokolů a řešení začaly vznikat počítačové sítě. Sítě se postupem času rozšiřovaly a s tím vznikla i potřeba monitorování sítí a provozu v nich. Důvodů pro to bylo hned několik. Ať už to byla kontrola vytížení sítě, či správné fungování sítě, tak i případná detekce škodlivého či zbytečného provozu.

V následujících podkapitolách nejdříve vysvětlím základní principy fungování počítačových sítí. Následně se věnuji jednotlivým způsobům jejich monitorování a v závěru se věnuji tunelovacím protokolům se zaměřením na protokol WireGuard.

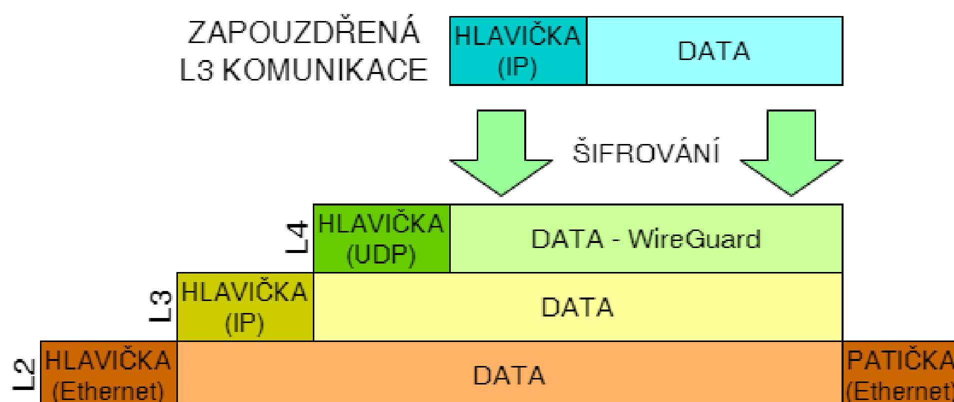
1.1.1 Principy počítačových sítí

Postupem času vznikly různé komunikační sítě a protokoly. Jednou ze základních koncepcí fungování sítí se stal model ISO/OSI, který rozděluje komunikaci do sedmi vrstev. Následně vznikly i další (třeba TCP/IP). Struktura ISO/OSI je následující:

- 7 – aplikační vrstva - doručování dat podle protokolu
- 6 – prezentační vrstva - doručování dat podle protokolu
- 5 – relační vrstva - doručování dat podle protokolu
- 4 – transportní vrstva - adresování portů - doručování segmentů
- 3 – síťová vrstva - IP adresování - doručování paketů
- 2 – linková vrstva - MAC adresování - doručování rámců
- 1 – fyzická vrstva - doručování bitů

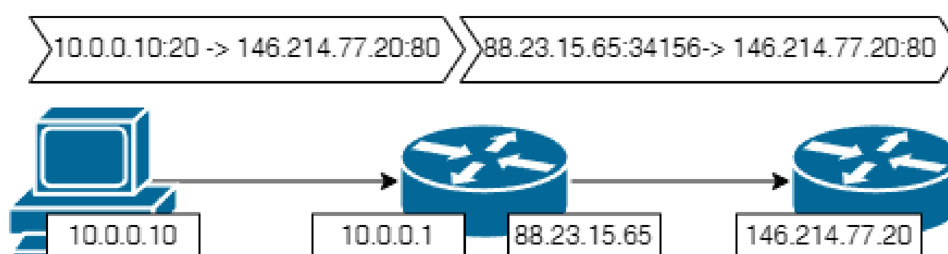
Na každé vrstvě mohou pracovat různé protokoly. V této práci se věnuji protokolu WireGuard, proto budu uvažovat, že na druhé vrstvě se používá Ethernet, na třetí IP protokol a na čtvrté UDP. WireGuard je tunelovací protokol pracující na třetí vrstvě. Znamená to, že je

schopen zapouzdřit data, která jsou na třetí vrstvě a výše a poslat je tunelem ke druhému bodu tunelu. Tunelovacím protokolům je věnována podkapitola 1.1.8. Zapouzdřená a zašifrovaná data posílá přes čtvrtou vrstvu pomocí protokolu UDP. TCP je vedle UDP jedním z nejpoužívanějších protokolů na čtvrté vrstvě. UDP oproti TCP nemá zajištěnou kontrolu doručení dat a jeho komunikace je primitivnější. Nákres fungovaní zapouzdření WireGuard na jednotlivých vrstvách je vidět na obrázku 1.1



Obrázek 1.1 Zapouzdření WireGuard na jednotlivých vrstvách

Důležitou prerekvizitou pro další zbatek práce je vysvětlení NAT. IPv4 pracující na druhé vrstvě používá adresaci pomocí 32 bitových adres. S rozšířením internetu po světě se po nějaké době narazilo na strop v podobě maximálního počtu adres, kterých může být $2^{32} = 4294967296$. Reálně použitelných je ve skutečnosti méně, protože různé rozsahy adres jsou na nějakou funkci rezervované. Začalo se tedy využívat překladu adres, protože na každé IP adrese může být použito $2^{16} = 65535$ portů a jen málokdy se používají všechny. Funkce NAT tedy vezme adresu na vstupu a přeloží jí na adresu na výstupu a pamatuje si výstupní port. Tato funkce si v čase uchovává aktivně používané porty, které odpovídají jednotlivým spojením a mají určitou platnost po které se informace zmaže. Páry adres a portů si funkce uchovává v překladové tabulce. Grafický nákres fungovaní NAT je na obrázku 1.2.



Obrázek 1.2 Princip fungování NAT

1.1.2 Monitorování počítačových sítí

Přístupů k monitorování sítí může být hned několik. Můžeme monitoring provádět jednoduchými nástroji pro kontrolu vytížení linky, či pouze dostupností nějakého bodu (ping). Tomuto monitoringu v této práci se nebudu věnovat

Pozornost zde zaměřím na sofistikovanější monitoring. Takový monitoring většinou slouží jako obrana před síťovými útoky a hromadně se nazývají IDS/IPS nástroje (Intrusion Detection System, Intrusion Prevention System). Pro potřeby monitoringu je často potřeba přenášená data zachytávat a dále s nimi pracovat. Zachytání se může provádět na aktivních prvcích v síti, či na koncových zařízeních nebo pomocí sond. Sonda je zařízení (HW či SW), které sbírá data a dále je posílá ke zpracování do jiného zařízení. Sonda nejčastěji bývá na nějakém důležitém místě v síti, kde je velký provoz a zachycených dat bude co nejvíce.

Ukládat a zpracovávat všechna sesbíraná data je výpočetně a hlavně paměťově velice náročné. Páteřní linky dnes běžně dosahují rychlostí desítek až stovek Gbit/s. Navíc mnoho zachycených dat nemusí být pro potřeby monitoringu důležité a jen zabírají místo na úložišti. Zmínit mohu třeba velké množství duplicitních dat z hlaviček paketů či nějaké dodržení syntaxe. Dodržení syntaxe může znamenat třeba používání fixní délky paketů nebo částí v něm. Z tohoto důvodu se začala nasbíraná data slučovat. Data můžeme slučovat jen na základě počtu přenesených dat, či časového rozmezí, ale mnohem efektivnější je slučovat na základě síťových spojení.

Přesto existují nástroje, které pracují s jednotlivými pakety a provádí paketovou analýzu. Mohou například hledat některé známé příznaky škodlivého provozu pomocí datábází signatur. Mezi známé systémy ze světa open-source mohu vyjmenovat Suricata [12], Snort [13] nebo Zenarmor [14]. V komerčním světě jsou to pak produkty od společností Cisco [15], Fortinet [16], Juniper [17]. Komerční řešení většinou kombinují více přístupu a nelze je tedy jednoznačně zařadit. Navíc i detekce probíhá jinak než softwarově. Jako příklad zmíním využivaní vlastních ASIC čipů od společnosti FortiNet, které jsou vyvíjeny jen pro potřeby hloubkové detekce v paketech [18].

1.1.3 Síťové toky

Síťový tok popisuje síťové spojení. [19] To vzniká z jednoduché myšlenky, že v počítačových sítích na nějaký dotaz existuje odpověď. Tato komunikace je složena z rámců, nicméně pro lepší práci s daty se zaměříme na třetí a čtvrtou vrstvu počítačových sítí podle modelu ISO/OSI 1.1.1. To znamená na úroveň paketů. Pakety tvořící síťové spojení považujeme za IP toky (IP-flows). IP toky jsou založeny na následujících shodných informacích:

- IP adresa zdroje
- IP adresa cíle
- Port zdroje
- Port cíle
- Časová razítka
- Počet přenesených paketů
- Velikost přenesených paketů

Prvním standardem v oblasti síťových toků byl NetFlow [20] od společnosti Cisco [15] v roce 1996. Ten byl časem rozvíjen až na NetFlow v9 [21], které je založeno na šablonách políček. NetFlowv9 lze také dohledat pod RFC 3954. Následně byl organizací IETF definován standart IPFIX (RFC 7011) [22]. Někdy je také označovaný jako NetFlow v10.

1.1.4 IPFIX

IP Flow Information Exchange (IPFIX) [22] je standard pro popis síťových toků, který má dva typy zpráv. Tyto typy jsou šablony a data. Vše je uloženo v binárním formátu. Syntaxe IPFIX začíná hlavičkou, za kterou následuje alespoň jedna množina:

- Data Set
- Template Set
- Options Template Set

Konkrétní ukázka je vidět na 1.1

Výpis kódu 1.1 Ukázka IPFIX zprávy [22]

```
+--------+-------------------------------------------------------+
|        | +----------+ +---------+     +-----------+ +---------+ |
|Message | | Template | | Data    |     | Options   | | Data    | |
| Header | | Set      | | Set     | ... | Template  | | Set     | |
|        | |          | |         |     | Set       | |         | |
|        | +----------+ +---------+     +-----------+ +---------+ |
+--------+-------------------------------------------------------+
```

IPFIX může kromě základních informací obsahovat některé další. Můžeme si tak základní data o IP tocích obohatit o další užitečné informace, se kterými budeme dále pracovat. Rozšířené informace musí být jedním z datových typů:

- adresa IP (v4, v6)
- adresa MAC
- celé číslo
- reálné číslo
- boolean
- časové značka
- řetězec znaků

Formát hlavičky IPFIX zprávy je přesně definovaný počtem bitů, jak je vidět na ukázce1.2

Výpis kódu 1.2 Ukázka IPFIX hlavičky [22]

```
 0                   1                   2                   3
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Version Number          |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Export Time                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Sequence Number                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Observation Domain ID                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Pro zachytávání dat existuje open-source řešení ipfixprobe [11] od sdružení CESNET. Jedná se o softwarovou sondu, která je schopna zachycená data pomocí různých dostupných pluginů zpracovat a poslat dále v různých formátech pro práci se síťovými toky. Pro kolekci nasbíraných toků existuje open-source software IPFIXcol2 [10], který přijímá data ve formátu IPFIX a umožňuje sbírat data z několika sond a dále je transformovat. Tento kolektor může data dále posílat ke zpracování třeba do frameworku NEMEA [9] nebo je ukládat a to v různých formátech včetně UniRec [23].

1.1.5 UniRec

UniRec je dalším možným formátem pro práci se síťovými toky. Částečně je podobný formátu IPFIX. Data jsou také ukládána v binárním formátu a také se jedná o formát, který je tvořen šablonami a daty. Rozdílem je, že se šablona posílá jen na začátku a dále se neposílá. Z toho plyne, že přes jedno komunikační rozhraní se stále posílá jen jedna posloupnost dat daná šablonou na začátku. Oproti IPFIX je tedy rychlejší a není potřeba pokaždé data složitě parsovat. Rychlost přístupu ke konkrétním datům v záznamu lze přirovnat ke struktuře v jazyce C, které je také částečně podobný. Délka jednoho toku v UniRec formátu je 65534 bytů s čím je potřeba dále v implementaci počítat. Formát UniRec1.3 vznikl přímo pro potřeby systému NEMEA ve sdružení CESNET.

Výpis kódu 1.3 Ukázka UniRec šablony [3]

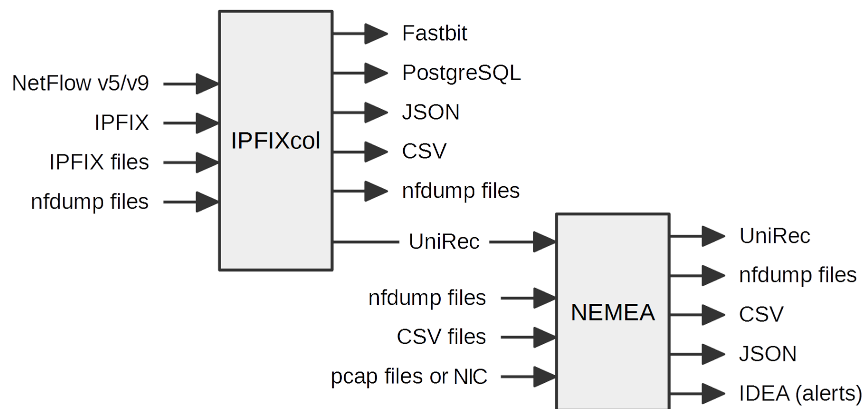
```
byte      0        1        2        3
       +-------+-------+-------+-------+
 0     |                               |
 4     |            DST_IP             |
 8     |                               |
12     |                               |
       +-------+-------+-------+-------+
16     |                               |
20     |            SRC_IP             |
24     |                               |
28     |                               |
       +-------+-------+-------+-------+
32     |             BYTES             |
       +-------+-------+-------+-------+
36     |            PACKETS            |
       +-------+-------+-------+-------+
40     |   DST_PORT    | HTTP_RSP_CODE |
       +-------+-------+-------+-------+
44     |   SRC_PORT    | PROTO | TCP_F |
       +-------+-------+-------+-------+
48     | HTTP_URL(off) | HTTP_URL(len) |
       +-------+-------+-------+-------+
52     | HTTP_USER(off)| HTTP_USER(len)|
       +-------+-------+-------+-------+
```

1.1.6 NEMEA

Systém Network Measurements Analysis (NEMEA) [9] je modulární systém určený pro síťovou analýzu a detekci anomálií - útoků. Byl vyvinut sdružením CESNET. Celý systém se skládá ze tří hlavních částí - Moduly, Framework, Supervisor, které popíši dále. Systém může být používán spolu s nástroji ipfixprobe a ipficol2, jako je to vidět na obrázku 1.3

Systém zpracování dat je složen z modulů, které mohou být dvojího typu. První typ jsou detektory, které provádí samotnou analýzu a případně detekují nějaké chování v síti. Druhou částí jsou podpůrné moduly, které mají na starost práci s daty - např. export, anonymizace, čtení, ukládání, předzpracování, přehrání, opakování.

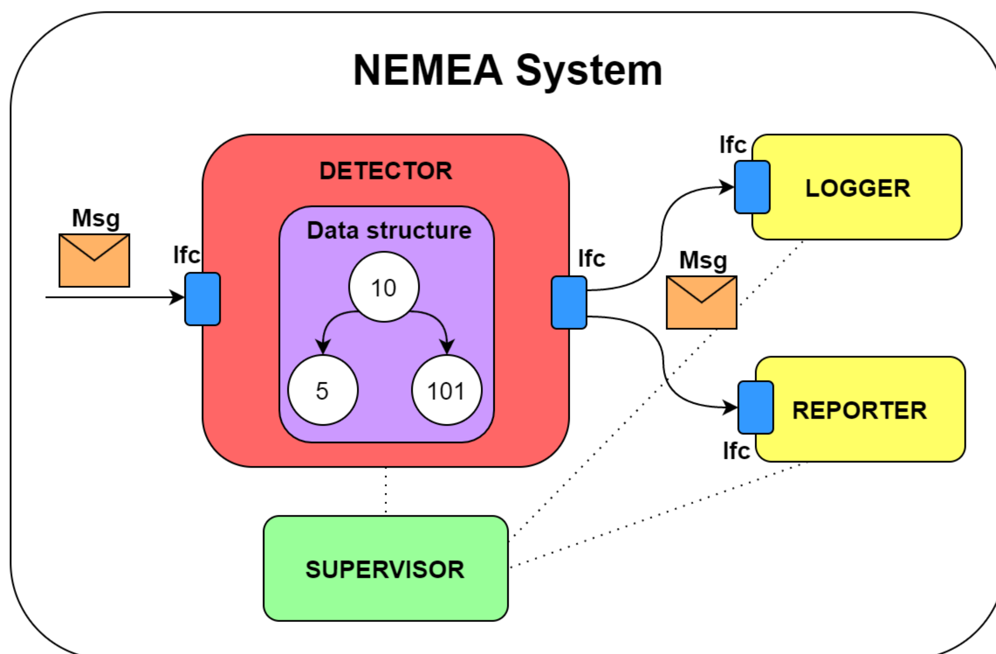
Moduly využívají k fungování soubor knihoven, který se hromadně nazývá NEMEA-Framework [24] a který je nedílnou součástí celého systému. Moduly spolu komunikují přes rozhraní Traffic Analysis Platform (TRAP) [25] a posílají si mezi sebou data ve formátu UniRec, jehož implementace je jednou z knihoven ve frameworku. TRAP může využívat několik komunikačních kanálů, kterými jsou UNIX socket, TCP/IP port, TLS a práce se soubory. Framework mimo



Obrázek 1.3 Komplexní systém monitorování pomocí síťových toků [9]

předchozí obsahuje knihovny common, pytrap a pycommon. Jak z názvů vyplývá moduly mohou být napsány jazycích C, C++ a Python.

Poslední částí je Supervisor, který zajišťuje správné fungování celého systému a dohlíží nad jednotlivými moduly



Obrázek 1.4 Architektura systému NEMEA [26]

Celý Systém je graficky znázorněn na obrázku 1.4. Moduly jsou červenou barvou, moduly, žlutou, TRAP rozhraní modrou, UniRec zprávy oranžovou, knihovny fialovou a supervisor zelenou

1.1.7 Analýza paketů

Výše zmíněným dalším způsobem pro monitoring a analýzu síťového provozu je paketová analýza. Tento způsob bývá často pomalejší, protože musí zpracovávat stejné množství dat, jako pro-

teklo monitorovaným bodem v síti. Naopak tento způsob je nejpřesnější, protože máme o datech všechny informace. Problémem paketové analýzy je i práce s nasbíranými daty. Potřebujeme je někde ukládat a v sítích, které dosahují stovek gigabitu za sekundu, potřebujeme rychlé drahé SSD disky, které navíc musí mít velikou kapacitu.

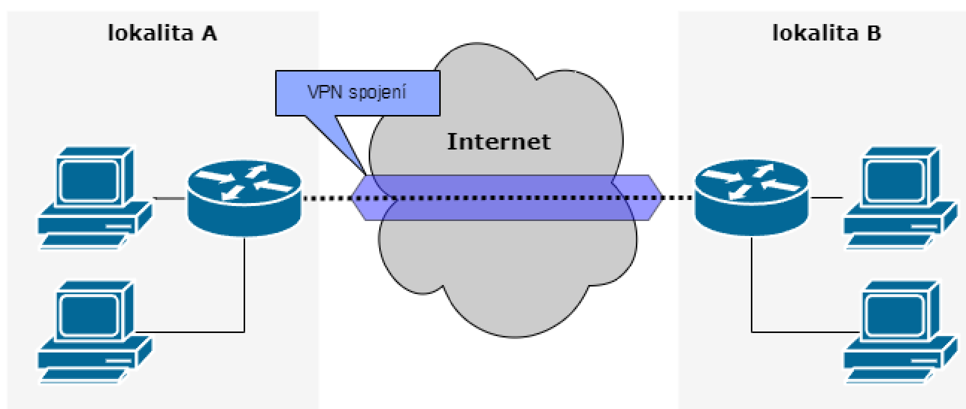
Ze základního principu fungování OSI/ISO modelu nasbírané pakety postupně rozdělíme podle vrstev a podle informací v jednotlivých vrstvách dále zpracováváme. Tímto procesem opět spotřebováváme značné množství výkonu a po rozpouzdření dat můžeme také dojít k závěru, že celý proces byl zbytečný, protože se na nejnižší vrstvě data před odesláním zašifrovala a my analyzujeme jen náhodné byty a vidíme jen nezašifrované hlavičky. Z tohoto důvodu se také používají síťové toky, kde informace z hlaviček necháváme a nemusíme potenciálně zašifrovaná data zpracovávat.

Existují samozřejmě nástroje, které dokáží paketovou analýzu provádět a jsou jimi třeba Suricata [12] či Snort [13], které používají databázi signatur pro bezpečnostní analýzu. Dalším široce populárním je program Wireshark [27]. Tento program je jeden z nejpoužívanějších a má mnoho funkcí. K jeho popularitě přispívá i to, že je široce podporován mezi operačnímy systémy a má grafické rozhraní. Může fungovat, jak v offline režimu, kdy zpracovává již nasbíraná data, tak i v online režimu, kdy využívá knihovnu pcap [28] a nasbíraná data dokáže dále uložit a dále s nimi pracovat. Velkou výhodu má také v množství dostupných pluginů a je distribuovaný pod licencí GPLv2 [29], takže ho lze dále použít a upravovat jeho zdrojový kód

Kromě paketové analýzy dokáže pracovat i se síťovými toky. Dle mého názoru ale na to není moc vhodný, protože pro tento typ dat nemá k dispozici mnoho nástrojů a umí data jen zobrazit, případně použít primitivní filtry. V této práci jsem tento nástroj také používal pro záchyt některých dat, třeba pro práci v anotátoru PayloadAnotator.

1.1.8 Protokoly VPN

Pojem Virtual Private Network (VPN) značí propojení dvou a více bodů ve veřejných sítích tak, že se chovají, jak kdyby byly ve stejné privátní síti. Pro lepší představu si představme veřejný internet, ve kterém chceme, aby počítač v práci komunikoval přímo s počítačem doma a tiskárnou, jako kdyby byly vedle sebe fyzicky. Uvedený případ je jen jedna z možností, k čemu se VPN používá. Další možností může být propojení více poboček firemní sítě nebo skrytí skutečné identity na internetu za nějakou fiktivní. Principu fungování VPN se také říká tunelování, protože se chová, jak kdyby na jednom konci síťový provoz vešel do tunelu a na druhém konci beze změn vyšel ven. Nákres fungování je na obrázku 1.5



Obrázek 1.5 Diagram VPN

První VPN protokoly se objevily již koncem 90.let. Trpěly mnoha nedostatky, mezi největší

patřilo zabezpečení. Postupem času vzniklo mnoho dalších [30]. Mezi aktuálně nejpoužívanější patří:

- L2TP
- IKEv2
- SSTP
- IPSec
- OpenVPN
- WireGuard

Poslední vyjmenovaný protokol je také nejmladší a oproti ostatním má několik odlišností. Má mnoho výhod, ale nehodí se na každé místo, kde VPN potřebujeme. Ve srovnání s ostatními protokoly umí fungovat pouze v režimu bod-bod (P2P). Naopak některé protokoly podporují i další typy, jako je síť-síť nebo bod-síť. Mezi další nevýhody patří také absence kontroly doručení dat, protože využívá protokol UDP a tak strana odesílající pakety neví, zda druhé pakety došly. Určitou kontrolu spojení protokol má, ale ne na úrovni jednotlivých paketů, které se mohou proto ztrácet. Oproti tomu má i mnoho výhod. Mezi ně patří podpora IP roamingu, jednoduché nastavení nebo výpočetní nenáročnost spojená s krátkou implementací. Zdrojový kód totiž dosahuje délky pouze jednotek tisíc řádků, zatímco třeba OpenVPN přes sto tisíc a IPsec až čtyřista tisíc [31]. Podrobnému popisu protokolu se budu věnovat v další kapitole.

1.1.9 Protokol WireGuard

Protokol je k dispozici teprve krátce. Byl vydán v roce 2018 jako opensource Jason A. Donenfeldem [4]. Toto je první odlišností oproti ostatním, prože ho nevyvinula žádná velké společnost ani tým programátorů. Další odlišností je využívání pouze protokolu UDP bez ověření doručení. Důvodem pro vytvoření bylo pro Jasona absence dostatečně rychlých protokolů VPN. Existující řešení byla pomalá a i z pohledu zdrojového kódu značně náročnější [5]. OpenVPN má pro linux přes sto tisíc řádků zatímco WireGuard pouze kolem čtyř tisíc [31] a i méně znalý programátor se v něm proto lépe orientuje. Rychlost ostatních protokolů využívajících TCP je značně ovlivněna i tím, že v TCP tunelu přenášíme TCP provoz a navazování spojení a kontrola doručení je tak prováděna dvakrát na různých vrstvách.

Jak jsem již zmínil, tak protokol funguje na třetí vrstvě ISO/OSI modelu a využívá UDP protokol. Pro tunelování podporuje jak IPv4, tak IPv6 protokoly. Jeho konfigurace je velmi snadná a zabere jen přibližně čtyři jednoduché příkazy pro správné nastavení. Se správnou konfigurací umí překonat NAT a umí plynule provádět roaming. Roaming je změna IP adresy jedné ze stran. Roaming proběhne správně jen pokud je změna IP adresy jen na jedné straně spojení a pro správné navázání WireGuard spojení je potřeba aby alespoň jedna strana měla určenou IP adresu či DNS záznam. Při provozování skrz NAT je také potřeba, aby jedna strana měla veřejnou adresu nebo z jedné strany byla route na druhou.

Spojení se navazuje tak, že jedna strana pošle paket "Handshake Initiation". Druhá strana na něj odpoví paketem "Handshake Response"a velmi často pošle rovnou paket "Keepalive". Protokol má níže uvedené konstanty, které nejčastěji definují časovou platnost (timeout) po které se mažou klíče, navazuje nové spojení, případně čeká na odpověď. Struktura jednotlivých paketů je vysvětlena v sekci 2.2.5.

Protokol WireGuard využívá následující kombinaci algoritmů šifrování [32]. K symetrickému šifrování využívá proudovou šifru ChaCha20 [33] s ověřením integrity pomocí Message Authentication Code (MAC) Poly1305 [34], dohromady jsou definované v RFC7539 [35]. K výměně klíčů využívá protokol Noise [36] a klíče generuje pomocí Diffie-Hellman (ECDH) algoritmu nad

eliptickými křivkami, konkrétně nad křivkou Curve25519 [37]. K funkčnímu šifrování potřebuje také hashovací funkci BLAKE2s [38]. Oproti ostatním hashovacím funkcím by měla být rychlejší, což se podepisuje i na rychlosti WireGuard komunikace. Dále v šifrování ještě využívá algoritmus během hashování SipHash24 a HKDF [39] pro odvození klíče. Aby byl protokol odolný i proti útokům typu reply, tak v šifrování používá časové značky typu TAI64N. Celé šifrování je tedy postaveno na protokolu Noise. Tento protokol přijímá pro WG parametr: `Noise_IKpsk2_25519_ChaChaPoly_BLAKE2s`. V implementaci se vyskytují konstanty [32]. Zmíním nejdůležitější:

- `REKEY_AFTER_MESSAGES` = 2^{60} sekund
- `REJECT_AFTER_MESSAGES` = $2^{64} - 2^{13} - 1$ sekund
- `REKEY_AFTER_TIME` = 120 sekund
- `REJECT_AFTER_TIME` = 180 sekund
- `REKEY_ATTEMPT_TIME` = 90 sekund
- `REKEY_TIMEOUT` = 5 sekund
- `KEEPALIVE_TIMEOUT` = 10 sekund
- `KEEPALIVE` – nastavuje se v konfiguračním souboru WG v celých sekundách

Konstanta `REKEY_AFTER_TIME` určuje po jaké době je navázáno nové spojení (Handshake). Ta je nastavena na 2 minuty. Každé dvě minuty se tedy pomocí Curve25519 vygeneruje dvojice privátní–veřejný klíč a nazývají se ephemerální a slouží k šifrování jen po určenou dobu. Ty se po uplynutí `REJECT_AFTER_TIME * 3` smažou. Tato hodnota je nastavena schválně, protože program si vždy pamatuje tři klíče. Jeden je aktuálně používaný k šifrování a pak předchozí a budoucí, aby při konci platnosti bylo zaručeno, že šifrovaná data plynule přejdou na nový šifrovací klíč a data předchozího klíče nebudou zahozena. Aby byla bezpečnost ještě zvýšena jako ochrana před zpětným dešifrování dat pomocí nastupujících kvantových počítačů, může být konfigurace obohacena o před-sdílený klíč, který se zamíchá do veřejného klíče. Výchozí hodnota je nastavena na 32 bitové pole nul. Pokud jsme během intervalu `KEEPALIVE` neposlali žádný paket, tak se posílá ”Keepalive”paket. Je to paket který má prázdná data. Pokud pošleme paket a nedostaneme na něj odpověď během `KEEPALIVE + REKEY_TIMEOUT` pokusíme se navázat nové spojení

1.2 Strojové učení

Strojové učení je podoblast umělé inteligence. Jeho podstatou je studium algoritmů, které se dokáží v čase učit a předpovídat budoucí jevy. Algoritmy jsou založené na matematické statistice, statistické analýze a data mining. V dnešní době existuje již mnoho algoritmů strojového učení i samotných metod učení. Jsou to metody jako učení s učitelem, učení bez učitele, učení se zpětnou vazbou. Mezi nejvíce známé algoritmy strojového učení patří rozhodovací stromy, metoda nejbližších sousedů nebo RandomForest.

Strojové učení umí provádět tři typy úloh. Je to klasifikace, regrese a shlukování. V této práci se budu věnovat učení s učitelem a klasifikaci. Pro správné fungování strojového učení je potřeba nejdříve zajistit datovou sadu. Pomocí ní natrénovat klasifikátor a následně vyhodnotit klasifikaci. Ta by měla mít, co nejobsáhlejší data, ale naopak nesmí být až příliš velká. Navíc se datová sada může v čase měnit a měla by i reagovat na změny. V doméně počítačových sítí to může být změna adres či portů nebo změna komunikace. Problematikou postupného vylepšování a změnou datových sad se zabývá diplomová práce Jaroslava Peška[3], ze které vycházím. V této mojí práci se věnuji tvorbou datové sady protokolu WireGuard pomocí obecného anotátoru.

Část anotátor říká klasifikátoru přesný výsledek jevu. V našem případě je to zařazení síťového toku do nějaké skupiny pomocí nastavení *labelu*. Je tedy nutné, aby anotátor vždy dával správný výsledek. Výsledky mnou implementovaného anotátoru budou popsány v závěru práce. Výsledky modelu řadíme do čtyř skupin:

- TP – správně předpovězena pravda
- FP – nesprávně předpovězena pravda
- TN – správně předpovězena nepravda
- FN – nesprávně předpovězena nepravda

Z počtu hodnot v těchno skupinách se dále počítají další hodnoty pro určení výkonnosti strojového učení. Jsou to hodnoty přesnost, recall nebo f1 skóre, které je pro nás důležité a dál s jím budeme pracovat.

1.3 ALF framework

Klasifikace a tvorba datových sad síťových toků naráží na již zmíněný problém měnících se dat. Navíc anotování může být hodně drahé a to jak časově tak výkonově a paměťově. Je proto potřeba anotovat jen některé toky, které podle nějaké strategie zvolíme. Této metodě se říká aktivní učení[40]. Vybírat můžeme hned několika způsoby, jako jsou náhodné .... a podle zvolené metody výběru se poté liší i výkonost prediktivního modelu. Vše zmíněné implementuje framework ALF[3], který umí různé metody výběru toků k anotaci, iterování datové sady ze které může i ubírat záznamy. Dále automaticky přetrénovává model a vyhodnocuje výkonost predikce.

Celý framework funguje na příkladu klasifikace síťového provozu DoH. Pro libovolný jiný provoz je potřeba framework upravovat. Cílem této práce je vytvořit konfigurovatelný anotátor, který přijme konfigurační soubor s parametry a podle toho bude schopen snadno měnit třídy klasifikace anotovaných toků. Fungování framworku je graficky znázorněno na obrázku XY. Pro moji práci je ještě potřeba modul anotátor upravit tak, aby fungoval s mojí implementací.

Kapitola 2

Analýza

V této kapitole vysvětlím, jaké jsou možné způsoby detekování WireGuard provozu a doporučení do další kapitoly k návrhu a implementaci. Problém detekce WireGuard umí řešit některé nástroje. Detekce je přímo implementovaná v nástroji Wireshark [27]. Všechny nalezené práce ale spojuje jedno a to je paketová analýza na základě obsahu jednotlivých paketů, které mají přesně dané stavy a sekvenci. Sekvence a obsah paketů je dále vysvětlen dále v 2.2.4 a 2.2.5. V mezinárodních databázích jsem našel jednu práci [8], která se zabývala i zapojením strojového učení, ale pouze k detekci výskytu spojení. Nenašel jsem žádné řešení, které by se zabývalo párováním paketů, či aktivním dotazem.

2.1 Obecný anotátor

Jeden z hlavních cílů této bakalářské práce je vytvořit konfigurovatelný modul anotátor. Nevýhoda frameworku ALF je, že pro každý případ užití je potřeba napsat samostatný anotátor. Existuje několik přístupů, podle čeho data anotovat. Obecně jsem je rozdělil na následující části

- podle hlaviček (IP, port, protokol)
- podle rozšířených hlaviček (byty, časové značky, pakety)
- podle samotných dat uvnitř paketů (je potřeba vhodně zvolit vstupní toky)
- aktivním dotazem
- kontrolou souboru - logu

Pokud by se způsob anotování nehodil ani do jednoho výše uvedeného způsobu, je možné si napsat vlastní, který bude dědit již z existujícího řešení, případně napsat kompletně nový. Tato varianta není zcela efektivní, ale může využít níže navrženou architekturu mého řešení.

2.2 Metody anotace WireGuard

Podle mnou zkoumaného WireGuard provozu jsem navrhnul několik možných způsobů anotace - detekce WireGuard provozu. WireGuard je poměrně mladý komunikační protokol. Není vydán žádný oficiální standard definující tento protokol. Jediné z čeho je možné vycházet je oficiální dokumentaci [32], případně zdroj kódu na *https://www.wireguard.com/* a *https://git.zx2c4.com/wireguard.*

2.2.1 Protokol

WireGuard používá pro komunikaci UDP protkol, proto primitivní Anotace může anotovat všechny pakety, které využívají protkol UDP.

2.2.2 Číslo portu

Často používaným portem pro komunikaci je 51820. Tento port je i několikrát zmíněn v ukázkových konfiguracích. Podle zkoumání tento port používá i několik známých VPN koncentrátorů. Je vhodné se zaměřit i na porty kolem toho čísla. Předpokládejme, že ve sledovaném provozu je již port 51820 používán, se tedy použije port 51821. Proto je dle mne možné anotovat v tomto přístupu porty v rozsahu 51810 až 51830, pokud nemáme žádné další informace. Tento přístup může generovat nepravdivě anotované toky, protože může být anotovaný port využíván jinou službou. Také je možné že nesprávně anotovaný tok z tohoto rozsahu portů může být přeložená IP adresa při průchodu NAT.

2.2.3 Blocklist IP

Dalším přístupem může být blacklist WireGuard serverů. Ten může obsahovat námi vytvořený WireGuard server, ale můžeme sem zařadit i známé a hojně používané známé WireGuard servery. Ty můžeme zjistit z veřejně dostupných seznamů komerčních VPN koncentrátorů. Pro příklad uvedu servery společnosti SurfShark [6] a NordVPN [7], které protokol WireGuard používají.

Zjistit konkrétní IP adresy serverů společností bylo poměrně obtížné, protože si často výpočetní výkon u někoho pronajímají a podle dat z databází WHOIS [41] nic nezjistíme. Obtížné je to i proto, že se IP adresy často mění. K těmto závěrům jsem dospěl po tom, kdy jsem si službu u společnosti SurfShark zaplatil a pomocí nástroje Wireshark [27] analyzoval síťový provoz. Další postup byl projít soubory programu Surfshark na mém počítači se systémem Microsoft Windows. Tam jsem objevil nějaké log soubory a konfigurační soubory, ze kterých jsem zjistil, že společnost má veřejně dostupné API [42] bez ověření a ze kterého se dají zjistit doménová jména serverů (odkaz). Po přeložení těchto doménových jmen jsem se dostal k IP adresám, které jsou v příloze. Při nasazení do reálné sítě je důležité domény překládat velmi často, protože mají parametr time to live (TTL) jen 5 sekund a často se během mého testování měnily.

IP adresy serverů společnosti NordVPN jsem zjistil obdobně. Stejně jako společnost Surfshark má veřejně dostupné API [43] a dokonce přímo z něj lze zjistit IP adresy a další užitečné informace, jako třeba polohu serveru. Během průzkumu dalších poskytovatelů anonymních VPN jsem narazil na zajímavý projekt na stránce Github[1], který se právě věnuje zjišťování doménových jmen VPN serverů. Jejich překlad už je potom primitivní úkon. Seznamy IP adres jsou také na přiloženém mediu.

2.2.4 Data uvnitř paketu

WireGuard podle oficiální dokumentace [32] a implementace používá zašifrované UDP pakety pro všechnu komunikaci. Komunikace na paketové úrovni má strukturu čtyř počátečních bytů a pak zašifrovaná data. Počáteční čtyři byty určují jeden ze čtyř možných typů komunikace. Důležitý je jen první byte, zbylé tři jsou vždy nulové hodnoty. První nabývá hodnot:

- 1 – ”Handshake Initiation”
- 2 – ”Handshake Response”
- 3 – ”Cookie Reply”

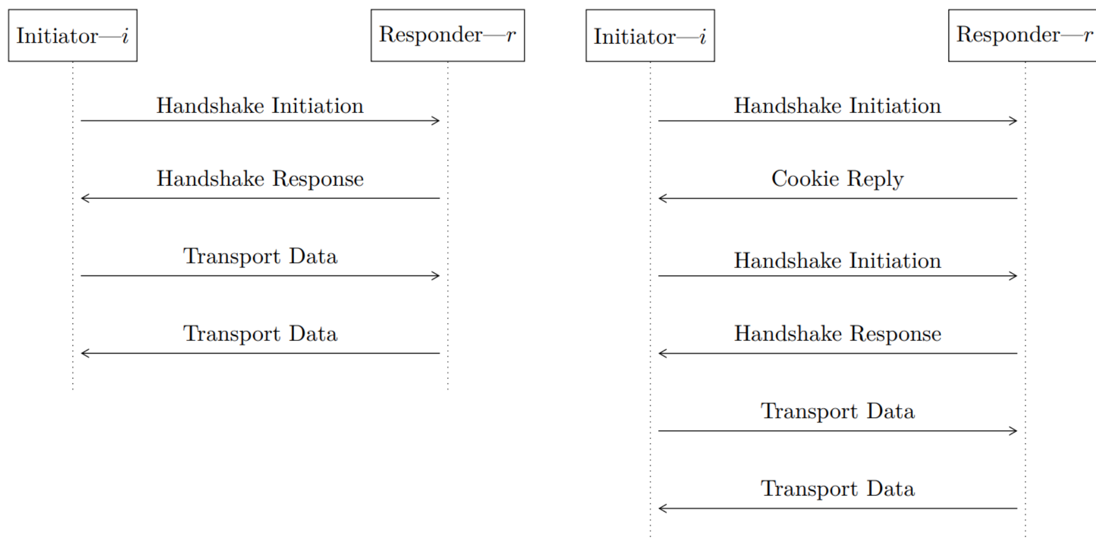
[1] https://github.com/passepartoutvpn/api-source

- 4 – ”Transport Data”

Tento přístup bude velmi přesný pro anotaci komunikace, která používá oficiální a neupravenou implementaci WireGuard. Bohužel se ovšem dostáváme na paketovou analýzu, která není tak rychlá a optimální. Jako mnou považované efektivní řešení je ze sondy posílat u UDP paketů i sekvenci prvních čtyř bytů dále ke zpracování. Tato funkce je již implementovaná v používaném nástroji ipfixprobe

2.2.5 Aktivní dotaz

Dále jsem se zaměřil na strukturu, posloupnost a velikost paketů. Protože WireGuard používá UDP, tak pro navázání spojení je potřeba vlastní řešení. Jako protiklad uvedu TCP, které navázání spojení samotné definuje. Podle oficiální dokumentace navazování spojení vypadá následovně 2.1 . Jednotlivé pakety pak vypadají pro Handshake Initiation 2.2 a pro Handshake Response 2.3. Pokud máme k dispozici klíče obou peerů, tak jsme schopni navázat spojení a podle odpovědi provoz anotovat. Anotovat můžeme pouze, pokud na navázání komunikace dostaneme odpověď, protože nějaký jednoduchý sken je nedostatečný a cílová adresa vůbec nemusí být WireGuard server. WireGuard odpoví na navázání komunikace pouze pokud jsou v pořádku klíče a kontrolní součty. Podle dokumentace je zde limit na dobu odpovědi pěti sekund, který je nutný pro tento typ anotace. To může aktivní dotaz výrazně prodloužit. Pokud ovšem využíváme síť, kde jsou odezvy v řádu milisekund, což je standart. Můžeme výchozí dobu pro odpověď výrazně snížit za cenu, že v některých případech můžeme anotaci provádět s mírnou chybou. Ze své zkušenosti bych výchozích 5 sekund snížil na dvě stě milisekund. Je to z důvodu, že pokud budeme chtít anotovat třeba tisíc síťových toků, anotace v nejhorším případě zabere více jak hodinu, snížením na dvě stě milisekund stále nevytvoříme velikou chybu a anotaci tisíce toků snížíme na tři minuty v nejhorším případě.



Obrázek 2.1 Navázání komunikace WireGuard [32]

type := 0x1 (1 byte)	reserved := 0^3 (3 bytes)
sender := I_i (4 bytes)	
ephemeral (32 bytes)	
static ($\widehat{32}$ bytes)	
timestamp ($\widehat{12}$ bytes)	
mac1 (16 bytes)	mac2 (16 bytes)

Obrázek 2.2 Handshake Initiation WireGuard [32]

type := 0x2 (1 byte)	reserved := 0^3 (3 bytes)
sender := I_r (4 bytes)	receiver := I_i (4 bytes)
ephemeral (32 bytes)	
empty ($\widehat{0}$ bytes)	
mac1 (16 bytes)	mac2 (16 bytes)

Obrázek 2.3 Handshake Response WireGuard [32]

2.2.6 Kontrola log souboru

WireGuard může být u Linuxu součástí jádra a může mnoho informací logovat do logu kernelu. Přečíst ho můžeme buď journalctl nebo dmesg. Velmi mnoho informací loguje i v systému Windows. V jiných systémech jsem toto nezkoumal. Pokud máme k dispozici log soubor, tak jsme schopni příchozí IP toky velmi přesně anotovat. Podle dokumentace je jednou za dvě minuty navázáno nové handshake s novými ephemeral klíči. Proto se log souboru každých 120 sekund objeví nové navázání komunikace a pokud anotovaný tok spadá do intervalu 120 sekund od navázání, tak můžeme přesně rozhodnout.. Dokumentace také říká, že afemerální klíče jsou platné maximálně 180 sekund, po této době jsou neplatné a druhá strana je nepřijme. Tento interval můžeme brát jako horní limit pro anotování IP toků.

2.2.7 Párování paketů

Jako rozšíření klasického anotování toků, zda se jedná o WireGuard provoz či ne, jsem určil, že můžeme i anotovat, co bylo obsahem WireGuard provozu. Potřebujeme k tomu na jedné straně šifrovaný WireGuard provoz a k tomu dešifrovaná data na druhé straně. K těmto údajům je potřeba také párovací tabulky s veřejnými a soukromými adresami. Do návrhu jsem zvolil, že se toky v této části budou anotovat, jako číslo majoritního portu v dešifrovaném provozu. Pokud víme, jaké služby se používají, tak čísla portu snadno zjistíme jaká majoritní služba byla v toku WireGuard provozu.

Kapitola 3
Návrh

V této kapitole se budu věnovat návrhem architektury softwarového prototypu obecného anotátoru a jeho konfigurace pro anotování WireGuard provozu.

3.1 Požadavky

Mezi důležitou část v softwarovém inženýrství patří i vytyčení základních požadavků na funkčnost. Ty jsou zmíněny ve dvou následujících podsekcích

3.1.1 Nefunkční požadavky

1. **Nefunkční požadavek** – Jako hlavní nefunkční požadavek je určena univerzálnost a obecnost. Je to požadavek na to, aby byl anotátor snadno upravitelný na jakýkoli jiný provoz s minimálními změnami. Je to jednak používání konfiguračních souborů, které lze snadno upravit v textovém editoru. Dále je to využití jazyka a struktur, které budou podporované na velké škále systémů. Univerzálnost by měla být dodržena i dodržením architektury, která umožňuje snadné doimplementování dalších způsobů anotace. Obecnost by dále měla být i ve velkém množství parametrů u tříd a metod. Kdy je třeba možné určovat zpoždění dat u toků nebo prioritu zpracování.

2. **Nefunkční požadavek** – Další požadavek související s obecností je univerzální rozhraní. Data může anotátor přijímat, jak z databáze, kam byla již dříve vložena, tak z rozhraní TRAP [25], kterým může být soubor, tak i TCP, UDP port nebo UNIX socket. TRAP rozhraní může anotátor používat jako výstup. Z tohoto důvodu je i samotný anotátor použitelný v systému NEMEA, případně nemusí nutně anotovat pouze data pro framework ALF. Anotátor musí umět pracovat s jakýmikoliv toky ve formátu UniRec – nezáleží, která políčka se používají – jediná podmínka je, aby byly obsaženy povinná políčka.

3–4. **Nefunkční požadavek** – Mezi nefunkční požadavky, které souvisí s obecností lze zařadit i škálovatelnost a schopnost distribuce. Anotátor by měl být připraven na vícevláknové předělání. Distribuce v tomto požadavku znamená, že jednotlivé části mohou běžet odděleně na různých strojích. Například na jednom poběží komunikační část pro TRAP rozhraní a na druhém se budou toky anotovat. Tyto požadavky bude anotátor splňovat i z pohledu paralelního běhu několika anotátorů.

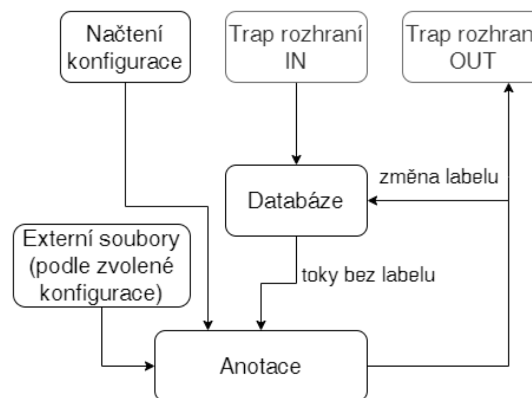
3.1.2 Funkční požadavky

1. **Funkční požadavek** – Bude umět přesně anotovat WG provoz podle navrhnutých indikátorů a metod. Bude garantovat přesnou anotaci.
2. **Funkční požadavek** – Bude korektně pracovat s daty a bude používat databázi. Požadavek potřeby databáze je vysvětlen dále.
3. **Funkční požadavek** – Parametry pro metody anotace bude načítat v konfiguračním souboru široce používaného formátu.
4. **Funkční požadavek** – Bude umět komunikovat s frameworkem ALF, ke kterému je přímo vyvíjen.
5. **Funkční požadavek** – Bude umět vytvářet datovou sadu.

3.2 Architektura

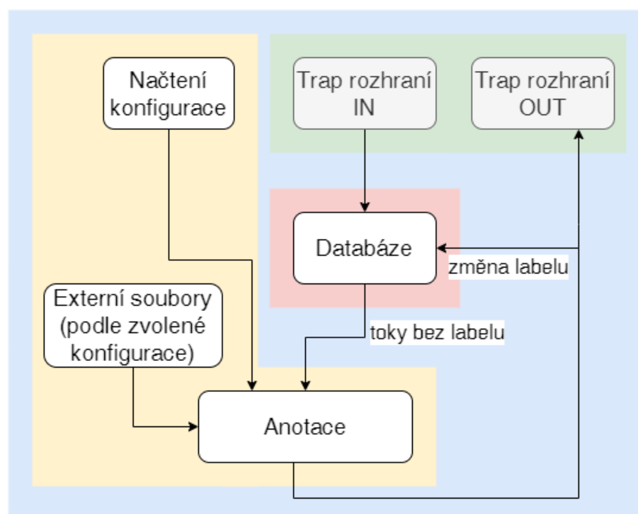
Při návrhu architektury jsem se snažil, co nejvíce dodržet obecnost, díky níž bude anotátor možné použít pro libovolný zvolený síťový provoz. Také jsem se zaměřil na komunikaci s pracovníky z laboratoře monitorování síťového provozu (CESNET), abych zjistil, zda můj návrh je dobré vylepšit nebo v něm něco neschází. Navrhl jsem proto po konzultacích využívat pro správu síťových toků databázi.

Základní princip anotátoru diagram anotace, je znázorněn na obrázku 3.1. Na začátku jsou data načtena z rozhraní TRAP. Ty jsou vloženy do databáze s políčkem *label*, který určuje, v jakém stavu jsou pozorované toky, případně hodnotu jejich anotace. Dále jsou z databáze vybrány záznamy k anotaci tj. ty které ještě anotovány nebyly. Následně probíhá anotace, ke které potřebujeme konfigurační soubory. V případě některých typů anotace i další potřebné soubory. Těmi jsou třeba párovací tabulky či log soubory. Po provedení anotace se aktualizují záznamy v databázi s konkrétním *labelem* a pokud se používá rozhraní TRAP, tak se data pošlou i na toto rozhraní dále.



Obrázek 3.1 Diagram zpracování

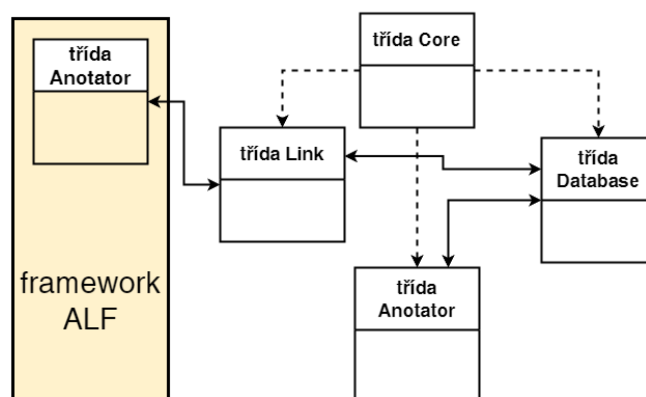
Z diagramu zpracování jsem určil, jak bude program rozdělen do základních tříd. To je graficky znázorněno na obrázku 3.2. Rozhraní jsem dal do jedné třídy Link se zelenou barvou. Databázi zajišťuje samostatná třída Database, která je označena červeně. Vše ostatní je ve třídě Anotator, protože se to podle zvoleného způsobu anotace může lišit a vše je závislé na konkrétní konfiguraci. Tato třída je znázorněna žlutou barvou. Celý systém pak řídí třída Core, která je na obrázku modrou barvou.



Obrázek 3.2 Diagram zpracování rozdělen do tříd

Části Database, Anotator, Core jsou reprezentovány jako abstraktní třídy. Třída Database zajišťuje komunikaci se zvolenou databází. anotátor řeší samotné anotování dat v databázi. Link je třída, která zajišťuje komunikaci s frameworkem ALF, případně jiným modulem pomocí rozhraní TRAP využívající formát UniRec. Třída Core ovládá všechny části, aby spolu dobře komunikovaly a celý systém fungoval korektně. Graficky je návrh znázorněn na obrázku 3.3. Na tomto obrázku jsou znázorněné i konkrétní třídy, které dědí z abstraktních tříd a implementují konkrétní použití. Mnou zvolená implementace popsaná dále je lehce odlišná od zmíněného obrázku. Odlišnost je způsobena tím, že v průběhu implementace jsem načítání konfiguračního souboru vyčlenil z třídy Anotator a vytvořil nový soubor parser.py obsahující třídy, které načítají a parsují konfigurační soubor. Pro funkční řešení je potřeba vytvořit třídy, které budou z abstraktních dědit rozhraní a budou implementovat správný případ užití.

Kromě výše zmíněných částí je na obrázku 3.3 vidět i část Anotator ve frameworku ALF. Tato část je důležitá, abych dodržel zadání, že anotátor bude fungovat s již fungujícím frameworkem. Pro komunikaci mezi frameworkem a mým systémem jsem se rozhodl použít rozhraní TRAP, které už je navrženo ve třídě Link.

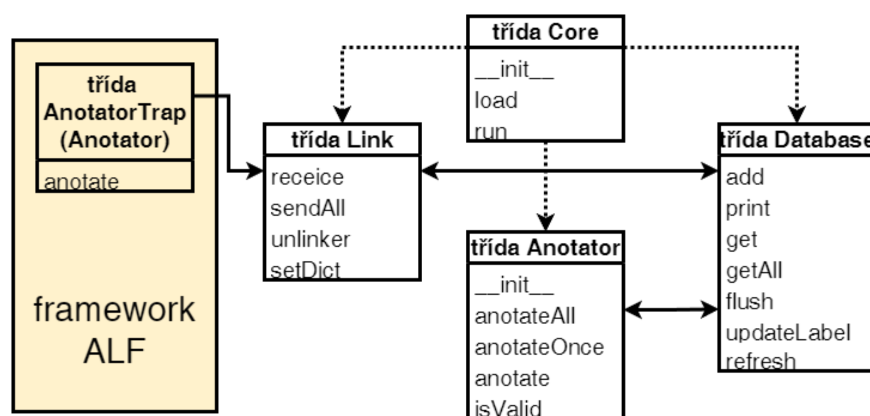


Obrázek 3.3 Architektura obecného anotátoru

Kapitola 4

Implementace

V této kapitole vysvětlím, jak jsem celý systém, vymyšlený v předchozí kapitole implementoval. Popíši jednotlivé části systému a jejich funkce. Graficky je systém i s metodami vidět na 4.1. V některých modech se nepoužívají všechny znázorněné části.



Obrázek 4.1 Architektura obecného anotátoru s rozepsanými metodami

4.1 Obecně

Pro implementaci jsem zvolil programovací jazyk Python [44]. Je to z důvodu rychlosti implementace a dostupným knihovnám pro práci s daty. Dále jsou to schopnosti pro používání v doméně strojového učení. V neposlední řadě jsem se pro tento jazyk rozhodl z důvodu, že v něm je napsaný framework ALF [3] a bude zachována vysoká kompatibilita. Kromě samotného systému pro anotování je totiž potřeba upravit implementaci frameworku, aby s mnou navrženým systémem komunikoval. Tato skutečnost byla uvažována už v návrhu. Pro konfigurační soubor jsem zvolil formát XML [45], který je široce používaný a jeho otevírání a úprava je jednoduchá.

Během vývoje jsem došel k závěru, že bude lepší, když bude program fungovat i samostatně bez frameworku ALF či jiných NEMEA [9] komponent, ale bude možné ho použít i pro jiné účely. Z tohoto důvodu umí program pracovat ve čtyřem operačních modech. Operační mody jsou:

- **mod 1** – Jen se anotují toky z databáze

- **mod 2** – Program přijímá toky a ty následně oanotované posílá na výstup. V tomto modu se nepoužívá databáze
- **mod 3** – Program přijímá toky, ukládá je do databáze a následně je anotuje v databázi.
- **mod 4** – Program přijímá toky, oanotované je posílá na výstup a navíc je ještě ukládá do databáze

Anotace toků se provádí změnou proměnné *label*, kterou má v mém programu každý tok se kterým se pracuje. Na počátku je *label* prázdný nebo nějaký výchozí. Pokud tok splňuje podmínky pro anotaci, tak se *label* podle konfigurační souborů změní na určený název.

Skoro ve všech částech zdrojového kódu používám knihovnu nemea-pytrap, která je dostupná skrz databázi PyPI [46]. Během implementace jsem ji používal v nejnovější dostupné verzi a to 0.15.0. Zdrojový kód je součástí NEMEA-Framework [24]. Knihovna je vydaná pod licencí nová BSD [47].

4.2 Komunikační část

Komunikaci zajišťuje třída Link. Ta má jako vstupní parametry databázi, šablonu vstupu pro TRAP rozhraní a definici vstupního IFC TRAP rozhraní. V případě, že chceme toky dále posílat na výstup, tak v metodě `sendAll` pak předáváme parametr pro výstupní TRAP rozhraní. Ukázka konstruktoru třídy Link je v rámečku 4.1.

Třída Link má metody:

- `receive` – čeká na příchozí data na TRAP rozhraní a ukládá je do databáze, která byla určena parametrem v konstruktoru
- `sendAll` – pošle všechny data z databáze na výstupní TRAP rozhraní
- `unlinker` – ukončí TRAP rozhraní a uvolní prostředky
- `setDict` – nastaví překladovou tabulku pro *label* u posílaných toků na výstupní TRAP rozhraní. Překladová tabulka jsou záznamy *label* - číslo oddělené středníkem. Překladová tabulka je zapotřebí, protože *label* se předává jako celé číslo.

Metoda `sendAll` má parametr `outputsrc_prm` typu string ve kterém je definováno IFC pro výstupní TRAP rozhraní. Metoda `setDict` má parametr `dictpath_prm` typu string, ve kterém je cesta k souboru s překladovou tabulku pro *label* u posílaných toků. Výstupní UniRec formát totiž předává *label* jako číslo a je proto potřeba jednotlivé *label*y na výstupu přeložit. Tabulka má formát dvojice *label*–číslo oddělených středníkem

Výpis kódu 4.1 Konstruktor třídy Link

```
def __init__(self,
             database_prm: database.Database,
             inputspec_prm: str,
             inputsrc_prm: str
             ):
    """Init the Link"""

    self._trap = pytrap.TrapCtx()
    self._trap.setVerboseLevel(-1)
    self._trap.init(["-i", inputsrc_prm], 1, 0)
    self._trap.setRequiredFmt(0, pytrap.FMT_UNIREC, inputspec_prm)
    self._databs = database_prm
    self._inputspec = inputspec_prm
```

4.3 Databáze

Pro správu toků v paměti je podle návrhu zvolena databáze. Reprezentuje ji abstraktní třída Database. V této práci jsou dvě implementace databáze. Jedna je použití databáze MySQL [48] a druhá je jen simulace databáze na úrovni programovacího jazyka. Tato simulace je ve třídě FakeDB. Tato třída se používá v případě, kdy v druhém modu nechceme používat databázi, ale stále musíme toky nějak uchovávat a nějak s nimy pracovat. Jakákoliv další implementace jiné databáze je možná. V budoucnu plánuji implementovat použití databáze Apache Cassandra [49] a PostgreSQL [50]. Použitá databáze MySQL komunikuje s mým zdrojovým kódem pomocí knihovny mysql-connector-python. V implementaci byla použita její nejnovější dostupná verze 8.0.33. Je dostupná z databáze PyPI [51] či z oficiálních stránek vývojáře MySQL společnosti Oracle [52]. Knihovna je vydaná pod licencí GPLv2 [29].

Zvolená databáze musí povinně uchovávat informace jako unikátní číslo toku, data toku, prioritu zpracování, životnost toku a *label*. Primárním klíčem v databázi je unikátní číslo toku typu int a názvem `flowid`. Aktuální implementace je navrržena tak, že unikátní číslo je nastaveno jako auto increment a s každým přidaným tokem se číslo samo nastaví na číslo o jedna větší než předchozí přidaný tok. Data toku jsou v databázi uchována jako bytearray, tudíž pro MySQL je to typ VARBINARY s názvem `flowkey` a musí umět uchovat délku 65534 bytů, což je maximální délka UniRec záznamu. Dále musí uchovávat *label* ve formátu string, pro MySQL je to TINYTEXT a dále životnost `ttl` a prioritu `prio`, obě typu int.

Abstraktní třída Database má metody:

- `add` – přidá tok do databáze.
- `print` – vypíše data z databáze na konzoli
- `get` – vrátí nějaký neanotovaný tok s nejvyšší prioritou
- `getAll` – vrátí všechny toky s určenou proměnnou *label*
- `flush` – smaže data z databáze
- `updateLabel` – aktualizuje tok s předaným parametrem *label*
- `refresh` – toky, které nebyly anotované se označí jako určené znovu k anotaci

Metoda `add` má povinný parametr `flowkey_prm` samotný tok ve formátu UniRec uložený jako pole bytů, dále pak volitelné parametry `label_prm` počáteční *label* s výchozí hodnotou "toanotate", `prio_prm` prioritu zpracování toku v rozmezí 0 až 255, `ttl_prm` životnost toku, životnost udává v jakou dobu přestane být tok platný a už se nebude anotovat. Pokud je životnost nastavena na 0, je nekonečná. Tato hodnota je v implementaci výchozí

4.4 Anotace

Nejdůležitější částí programu je třída Anotator. Ta má pět implemntací, jsou jimi:

- `BasicAnotator` – Třída, která anotuje podle políček v toku. Políčka se taky nají nazvat hlavičkami
- `PayloadAnotator` – Třída, která provádí anotaci podle bytů v paketu přečteného z pcap [28] souboru
- `LogAnotator` – Toky se anotují v této třídě podle informací z log souboru

- PairAnotatorDominantLabel – Implementace zajišťující párování paketů a toky anotuje jako číslo služby, která se v toku vyskytuje většinově. Mírnou změnou se dají toky anotovat i jako počet spojení v toku, či něco jiného, co souvisí s párováním paketů
- RequestAnotator – Třída, která posílá aktivní dotaz na zdroj z toku

Abstraktní třída je koncipovaná, že potřebné soubory budou v implementaci předány v konstruktoru a dále se již volá jen metoda pro anotaci. Jsou zde dvě metody pro anotaci. První anotateAll o anotuje všechny dosud neoanotované toky a metoda anotateOnce, která anotuje pouze jeden tok. V konstruktoru jsou povinné dva parametry, kterými jsou objekt Database a šablona UniRec typu string.

Abstraktní třída Anotator má již zmíněné metody:

- __init__ – konstruktor – každá implementace vyžaduje jiné povinné parametry, takže konstruktor se přes implementace výrazně liší. Většina informací je předána v konfiguračním souboru xml [45]
- anotateAll – provádí anotaci všech toků. Je to pouze obecné zpracování, dále se pro každý tok volá metoda anotate
- anotateOnce – provádí anotaci jednoho toku, jinak je stejná jako AnotateAll
- anotate – abstraktní metoda, kterou je potřeba pro každý anotátor implementovat
- isValid – kontroluje, zda jsou ještě nějaké neanotované toky

Většina parametrů je předána v konfiguračním souboru ve formátu xml [45], který bude podrobněji rozebrán dále v 4.5.1. Jednotlivá data ze souboru jsou konstruktoru parsována a jsou nastaveny příslušné parametry.

4.4.1 BasicAnotator

Jde o základní způsob anotace podle políček v UniRec. K fungování potřebuje jen korektní XML konfigurační soubor, který je popsán níže. V konstruktoru je načten konfigurační soubor, který je dále parsován v souboru parser.py a podle konfigurace je nastavena proměnná self._labels typu dict. Tato proměnná obsahuje obekty typu Combi, které provádí samotné filtrování toků. V metodě anotate se pak postupně přes proměnnou self._labels volá metoda process z objektu Combi. Tento objekt je popsán abstraktní třídou Combi v souboru parser.py. V souboru jsou dvě implementace třídy Combi. Je to třída Logic a Filter. Filter provádí filtraci toků a Logic umí obsahuje v proměnných objekty typu Combi, které pak při anotaci dále volá a můžeme proto filtrování toků různě kombinovat

4.4.2 PayloadAnotator

Třída si načte soubor ve formátu pcap [28], se kterým pak dále pracuje. Doporučuji používat menší pcap soubory, protože soubory vyžadují mnoho prostředků. Dále vstupuje do hry i volitelný parametr delta, kterým můžeme určovat větší časový úsek ve kterém budeme jednotlivé pakety z pcap souboru hledat. Tento parametr nastavuje čas v sekundách a je zde z důvodu, že pcap soubor a tok mohou mít určité zpoždění při jejich ukládání. Pro různá časová pásma je zde ještě volitelný parametr timezone chovající se stejně jako delta, ale nastavuje rozdíl v hodinách. Všechny výše uvedené parametry se předávají v konfiguračním souboru ve formátu xml.

Implementovaná třída používá knihovnu scapy [53]. Tato knihovna umí pracovat se soubory obsahující záchyt síťové komunikace. V mé práci se používá pouze pro čtení pcap souboru a

dále rozpouzdřením paketů za účelem zjištění sekvence bytů, které na začátku dat v UDP určují stav WireGuard spojení. Použitá knihovna byla ve verzi 2.5.0 (vydáno 2022) a je vydávána pod licencí GPLv2 [29].

4.4.3 LogAnotator

Třída parsuje log soubor. Ten je potřeba mít ve formátu, který se shoduje s regulárním výrazem předaným v parametru `regex`. Jako povinné části výrazu jsou čas [54] `<time>`, IP adresa `<ip>` a port `<port>`. Přesný formát logu se na jednotlivých OS může lišit, proto je přijímán regulární výraz, ve kterém se specifikuje, jak log vypadá. Ukázkový konfigurační soubor s parametry je níže na 4.4 a log soubor vypadá například takto: `"2023-04-14T14:11:43,219648+02:00 wireguard: wg0: Receiving handshake response from peer 1 (123.10.10.10:51820)"`. Je zde i konstanta `REKEY_AFTER_TIME` určující maximální platnost klíčů [32].

4.4.4 PairAnotatorDominantLabel

Třída potřebuje párovací tabulky z důvodu zmíněného v kapitole věnující se principu fungování protokolu WireGuard. Párovací tabulky musí být ve formátu řádků obsahující dvojici klíč–adresa. Adresa musí v případě veřejných adres obsahovat i port. Dvojice je od sebe oddělena středníkem. K anotování WireGuard provozu jsou nutné dvě párovací tabulky. Jedna nám říká, která soukromá adresa patří jakému klíči. Tato tabulka je exportována z konfiguračního souboru programu WireGuard a v čase se výrazně nemění. Druhá tabulka nám říká, jaká veřejná adresa patří klíči. Tato tabulka se v čase často mění, protože se mohou měnit adresy koncových bodů spojení a pokud spojení prochází přes překlad NAT, tak se mění i porty. Ukázka veřejné párovací tabulky je na 4.2.

Výpis kódu 4.2 Ukázka konfiguračního souboru pro třídu Log Anotator

```
qRCwZSKInrMAq5sepfCdaCsRJaoLe5jhtzfiw7CjbwM=;123.10.10.10:48265
FpCyhws9cxwWoV4xELtfJvjJN+zQVRPISllRWgeopVE=;211.20.20.20:21514
```

4.4.5 RequestAnotator

Třída funguje tak a tak, potřebuje klíče, případně volitelné klíče. Je zde i proměnná REKEY +doplnit

Třída RequestAnotator používá knihovnu noiseprotocol [55]. Knihovnu vytvořil Piotr Lizończyk. Knihovna není primárně určena pro WireGruard, ale s protokolem WireGuard má podobný základ v kryptografii. Tuto knihovnu jsem použil vzhledem k jejímu licencování a jednoduchosti. Část mého kódu pochází z ukázek použití od uvedeného autora z portálu Github [56]. Knihovna neprošla žádným bezpečnostním auditem a vzhledem k používání privátních klíčů je do budoucna vhodné ji nahradit jinou knihovnou, která podobný audit bude mít. Knihovna je vydávána pod licencí MIT [57]. Knihovnu jsem použil ve verzi v0.3.1 z roku 2020.

4.5 Třída Core

Třída Core zajišťuje správné pořadí spouštění ostatních částí programu. Třída je abstraktní a má čtyři implementace. Ty jsou si velmi podobné s výše uvedenými operačními mody, kdy mod určuje která třída bude zavolána. Jsou to třídy:

- **CoreDbOnly** – odpovídá modu 1, kdy se pracuje pouze s databází

- **CoreTrapIn** – odpovídá druhému modu, kde se používá pouze vstupní trap rozhraní a databáze
- **CoreTrap** – zajišťuje fungování programu ve třetím modu, kdy se databáze vůbec nepoužívá. Místo toho je vytvořena instance fiktivní databáze FakeDB.
- **CoreTrapMem** – velmi podobní třída té předchozí. Odpovídá čtvrtému modu, kdy se kromě vstupního a výstupního rozhraní ještě toky ukládají do databáze

Třídy, které používají TRAP rozhraní potřebují předávat záznamy UniRec s políčkem `class` typu int64 a políčko předat i v šabloně. V tomto políčku se předává *label*, který je zapsán v podobě čísla. Překlad mezi číslem a string je definován překládací tabulkou ve třídě Link v metodě `setDict`.

Abstraktní třída Core má metody:

- `__init__` – konstruktoru se předávají povinné parametry, které se liší napříč implementacemi
- `load` – provádí načtení konfiguračního souboru a zavolání správného anotátoru. není to abstraktní metody, je pro všechny implementace schodná - není závislá na modu
- `run` – spouští se po provedení předchozích metod a zajišťuje chod programu. Nejčastěji se jedná o nekonečnou smyčku, kterou lze klávesovou zkratkou ukončit, či násilně ukončit příkazem `kill`

4.5.1 Konfigurační soubor

Třídě Core se předává konfigurační soubor 4.3 ve formátu xml. Ten má následující strukturu:

Výpis kódu 4.3 Ukázkový konfigurační soubor Basic Anotator

```
<?xml version="1.0" encoding="UTF-8"?>
<anotator name="BasicAnotator">
        <label name="WG">
                <logic name="or">
                        <logic name="and">
                                <filter>
                                        <urfield>DST_IP</urfield>
                                        <comp>==</comp>
                                        <value>146.70.123.187</value>
                                </filter>
                                <filter>
                                        <urfield>DST_PORT</urfield>
                                        <comp>==</comp>
                                        <value>51820</value>
                                </filter>
                        </logic>
                        <logic name="and">
                                <filter>
                                        <urfield>SRC_IP</urfield>
                                        <comp>==</comp>
                                        <value>146.70.123.187</value>
                                </filter>
                                <filter>
                                        <urfield>SRC_PORT</urfield>
                                        <comp>==</comp>
                                        <value>51820</value>
```

```
                                </filter>
                        </logic>
                </logic>
        </label>
</anotator>
```

XML konfigurační soubor musí začínat xml sekvencí a následuje xml tag ”anotator”s atributem name, který určuje použitý způsob anotace. Za ním následují tagy určující parametry. Níže je 4.4 ukázka jednoduché konfigurace pro anotaci pomocí logu. Nejzajímavejší je konfigurační soubor pro BasicAnotator. Tam lze totiž určit jakékoliv platné políčko z ÜniRec šablony a napsat znak porovnání. Aktuálně jsou povolené porovnání rovná se, nerovná se. Políčka lze libovolně kombinovat pomocí tagů ”logic”a atributů ”and”, ”or”. Níže na 4.3 je vidět ukázkový konfigurační soubor pro Basic anotator, který nejdříve porovná zdrojovou ip adresu a port a dále pak cílovou adresu a port. Framework ALF jako ukázku používá anotaci DoH provozu. Pro tento příklad jsem vytvořil i konfiguraci v mém anotátoru která je vidět na 4.5.

Všechny anotátory přijímají společné parametry v konfiguračním souboru. Jsou to tagy start, end, labelout, labelother. Start a end určují, že chceme anotovat pouze ty toky, které jsou z určeného časového intervalu. Oba parametry se zadávají v iso formátu [54]. Výchozí hodnoty jsou pro start 1.1.1970 a pro end 1.1.2030. Tag labelout značí, jak se má anotovat tok, který do intervalu nespadá a jeho výchozí hodnota je prázdný řetězec. Parametr labelother značí, jak se mají anotovat toky, které odpovídají intervalu, ale nejsou anotovány a výchozí hodnota je other.

Basic Anotator má parametry konfiguračního souboru:

- label – povinný tag, za kterým následuje název *labelu* v anotaci
- logic – volitelný tag, pokud chceme kombinovat více filtrů
- filter – povinný tag označující filtr. Uvnitř musí být tři následující tagy
- urfield – název políčka v UniRec šabloně
- comp – typ porovnání. Aktuálně je možné uřit rovná se -=”a nerovná se ”!=”
- value – hodnota pro porovnání. Můžeme místo value použít i tag file, kde lze napsat seznam hodnot pod sebe. Soubor se hodí třeba pro seznam IP adres, pokud jich je mnoho
- společné tagy – start, end, labelout, labelother

Výpis kódu 4.4 Konfigurační soubor Log Anotator

```
<?xml version="1.0" encoding="UTF-8"?>
<anotator name="LogAnotator">
        <label>WG</label>
        <logfile>/home/sjbakalarka2/new_ex/delta_e.txt</logfile>
        <regex>(?P&lt;time&gt;.*?),\d*[+]\d*:\d*\s*wireguard:\s*wg0:\s*
            Receiving\s*(handshake\s*response\s*from\s*peer|handshake\s*
            initiation\s*from\s*peer|keepalive\s*packet\s*from\s*peer)\s
            *\d*\s*[(](?P&lt;ip&gt;.*?):(?P&lt;port&gt;.*?)[)]</regex>
        <delta>0</delta>
</anotator>
```

Payload Anotator má parametry konfiguračního souboru:

- label – povinný tag, za kterým následuje název *labelu* v anotaci
- pcap – povinný údaj určující cestu k souboru pcap ve kterém budeme hledat pakety související s tokem

- delta – volitelný časový údaj uvedený v celých sekundách. Pomocí tečky se dá uvést desetinné číslo nastavovat milisekundy. Rozšiřuje časový úsek ve kterém budeme hledat v pcap souboru.
- timezone – volitelný časový údaj uvedený v celých hodinách sloužící k posunu hledání v pcap souboru o hodnotu časového pásma
- společné volitelné tagy – start, end, labelout, labelother

Log Anotator má parametry konfiguračního souboru:

- label – povinný tag, za kterým následuje název *labelu* v anotaci
- logfile – povinný údaj určující cestu k souboru, který obsahuje log informace
- regex – povinný tag označující regulární výraz, který se shoduje se záznamy v log souboru. Regulární výraz musí mít tři parametry, které filtrují z logu časovou značku, IP adresu a port. [58]
- delta – volitelný časový údaj uvedený v celých sekundách. Pomocí tečky se dá uvést desetinné číslo a nastavovat milisekundy. Rozšiřuje časový úsek ve kterém budeme jednotlivé logy hledat.
- timezone – volitelný časový údaj uvedený v celých hodinách sloužící k posunu hledání v pcap souboru o hodnotu časového pásma
- společné volitelné tagy – start, end, labelout, labelother

Pair Anotator má parametry konfiguračního souboru:

- pubpairfile – povinný tag, který určuje cestu k souboru, který obsahuje párovací tabulku s klíči a veřejnými adresami
- privpairfile – povinný tag určující cestu k souboru, který obsahuje párovací tabulku s klíči a soukromými adresami
- pairflows – povinný údaj určující cestu k souboru obsahující síťové toky se kterými budeme přijmutý tok párovat
- inputspec – povinný údaj určující šablonu pro UniRec formát toků
- delta – volitelný časový údaj uvedený v celých sekundách. Pomocí tečky se dá uvést desetinné číslo a nastavovat milisekundy. Rozšiřuje časový úsek ve kterém budeme toky párovat.
- společné volitelné tagy – start, end, labelout, labelother

Výpis kódu 4.5 Konfigurační soubor pro DoH

```
<?xml version="1.0" encoding="UTF-8"?>
<anotator name="BasicAnotator">
        <label name="DoH">
                <logic name="or">
                        <filter>
                                <urfield>SRC_IP</urfield>
                                <comp>==</comp>
                                <file>~/alf/conf/blacklist.txt</file>
                        </filter>
                        <filter>
```

```
                                <urfield>DST_IP</urfield>
                                <comp>==</comp>
                                <file>~/alf/conf/blacklist.txt</file>
                        </filter>
                </logic>
        </label>
</anotator>
```

Request Anotator má parametry konfiguračního souboru:

- label – povinný tag, za kterým následuje název *labelu* v anotaci
- publicpeerkey – povinný parametr definující veřejný klíč druhého bodu, kte kterému se připojujeme
- privatekey – povinný parametr, ve kterém se předává náš soukromý klíč
- publickey – volitelný parametr, ve kterém se předává náš veřejný klíč
- presharedkey – volitelný parametr definující předsdílený klíč
- společné volitelné tagy – start, end, labelout, labelother

4.6 Soubor parser.py

Během implementace vznikla potřeba informace z konfiguračního souboru efetivně parsovat. Z tohoto důvodu byl vytvořen soubor parser.py, který kombinuje třídy s metodami a funkce, pro parsování xml souboru předepsané struktury. Nejsložitější je pro třídu BasicAnotator, kde lze libovolně kombinovat parametry a proto může být konfigurační soubor značně komplexní a složitý. Dále také umožňuje načítat externí soubory se seznamem informací. Pro příklad uvedu třeba soubor s IP adresami serverů, které se mají anotovat.

4.7 Anotator_ctl.py

Pro snadnou volbu módů a nastavování různých proměnných jsem navíc nad hotovou architekturou vytvořil ještě soubor anotator_ctl.py, který funguje jako jednoduché ovládání celého programu pomocí jednoho příkazu. Pomocí přepínačů lze nastavovat nejrůznější parametry, protože konfigurační soubor jen nastavuje anotátor, ale už ne třeba databázi a šablonu. Tento skript je tedy nejlepší na ukázkové spuštění programu a jeho přesné rozchození je popsáno v instalační příručce. Jako jeho přepínače patří:

- `--dbtype`
- `--dbhost`
- `--dbuser`
- `--dbpass`
- `--dbname`
- `--dbtable`
- `--urformat`
- `--mode`

- `--otlabel`
- `--confile`
- `--urin`
- `--urout`

4.8 Komunikace s ALF

Mnou implementovaný systém komunikuje s frameworkem pomocí TRAP rozhraní. Framework jsem musel upravit, aby tuto komunikaci podporoval. Udělal jsem to vytvořením nové třídy AnotatorTrap, která je dědicem abstraktní třídy Anotator. V konstruktoru této třídy je potřeba předat parametry vstupního a výstupního TRAP rozhraní. Dále si můžeme nastavit, zda chceme vidět podrobné debug zprávy. Před samotnou anotací je potřeba nastavit šablonu ve které se bude posílat UniRec přes TRAP rozhraní. Názvy políček musí sedět s názvz políček v dataframe, který ALF používá pro uchovávání toků. Dále musí šablona obsahovat políčko class, ve kterém se předává *label*, jako číselná hodnota. V opačném případě program skončí s chybou.

Kapitola 5
Vyhodnocení

V této kapitole se budu věnovat testování programu a vyhodnocením výkonnosti strojového učení. Nejprve se budu věnovat tomu, jak experiment probíhal a na konci kapitol shrnu výsledky

5.1 Testovací prostředí

Testování jsem prováděl na svém počítači doma. Nasbíraná data pocházela buď z mé sítě nebo pocházela ze serveru, který spravuje Filip Němec a na kterém je WG server s přibližně čtyřiceti nakonfigurovanými klienty. Tímto bych mu chtěl moc poděkovat, protože sám bych nebyl schopen nasbírat tolik síťové komunikace.

Testovací prostředí vypadalo následovně. Na počítači s procesorem AMD Ryzen 2600 a 32GB DDR4 paměti a operačním systémem Windows 10 běží virtualizační nástroj VirtalBox. V něm běží virtuální počítač s přidělenými čtyřmi virtuálními jádry a má přiděleno 8GB RAM. Na hlavním systému běžel nástroj Wireshark a ukládal nasbíraná data do pcap souborů. Na virtuálním stroji běželo vytvořený program a byla mu předávána data z hlavního počítače. Data byla cestou upravena. Především se pcap soubor transformoval do formátu síťových toků ipfix a následně do formátu UniRec. V nástroji Wireshark jsem u pcap souborů nastavoval filtr tak, abych měl jistotu, že se jedná o WireGuard provoz. Výsledky z Wireshark jsem porovnával s anotovanými toky ve virtuálním počítači. Takto byly otestovány všechny anotační metody až na párování paketů. U této metody jsem použil nasbíraná data od Filipa Němce, protože na zmiňovaném serveru běží upravená implementace ipfixprobe, která sbírá data o každém paketu a navíc sleduje i důležitou sekvenci bytů u UDP provozu, aby se dalo snadno zjistit, zda se jedná o WireGuard. Další výhodou bylo, že na sledovaném serveru běží záchyt jak na vstupním rozhraní, kde jsou data zašifrovaná, tak i na virtuálním rozhraní, kde se dešifrují a rozbalují WG pakety. S těmito daty a párovacími tabulkami, které se na serveru také ukládají, jsme schopni říct, který provoz byl zašifrován.

5.2 BasicAnotator

Tato metoda byla v porovnání s ostatními nejrychlejší. Je také nejprimitivnější a k funkci nepotřebuje žádné další soubory. Anotovat zhruba třicet tisíc toků s využitím simulace databáze – implementace FakeDB, aby nebyly výsledky ovlivněny rychlostí databáze, zabere kolem 30 sekund. Tato rychlost se odvíjí od složitosti konfiguračního souboru a množství parametrů. Při tomto experimentu se používá TRAP rozhraní, takže i to může výsledky ovlivnit.

Problém tohoto přístupu je, že pokud nemáme obecné informace o provozu, tak je nejméně přesná. Stačí změnit používaný port a celá anotace přestane fungovat.

5.3 PayloadAnotator

Tato metoda je časově velmi proměnná. Je to z důvodu, že anotátor si na začátku musí otevřít a rozbalit surová data z pcap souboru. Ty mohou být různě velké, ikdyž počet toků je stejný. Pro nižší stovky toků dosahují pcap soubory velikosti i stovek megabytů. Tato metoda je z tohoto důvodu vždy velice pomalá. Pro příklad uvedu hodnoty z experimentu. Anotovat přibližně šest set toků pomocí pcap souboru, který měl kolem sedmdesáti megabytů zabralo 20 minut.

Tato metoda je velmi přesná. Jediné úskalí je vtom, že WireGuard může kdokoliv měnit a při změně implementace přestane tento přístup fungovat. Navíc můžeme chybně anotovat tok, jako WG provoz, jen proto, že zrovna data v UDP seděla na první byty stejně.

5.4 LogAnotator

V této metodě potřebujeme na vstupu log soubor. Ten není obtížné získat. Stačí u WireGuard zapnout logování. U sytému Windows je zapnuté už jako výchozí hodnota. Tato metoda je v porovnání s ostatními metodami spíše rychlejší. Nejrychlejší zůstává základní metoda. U této zabralo anotovat tisíc toků přibližně půl minuty. Log soubor měl několik tisíc řádků řádků.

Tato metoda je velmi přesná. Máme prakticky všechny informace potřebné k přesnému určení zda se jedná o WG spojení.

5.5 PairAnotatorDominantLabel

Tato metoda je nejzajímavější. Umí narozdíl od ostatních nejen detekovat WG provoz, ale říct i co se v něm nachází. Během analýzy jsem byl rychlosti lehce skeptický, ale výsledek předčil mé očekávání. Anotovat zhruba třicet tisíc toků zabere kolem 30 minut, kdy vstupní toky byly dodány pomocí trapcap souboru, který měl velikost kolem osmnácti megabytů a rozbalené toky měli v souboru velikost kolem sedmnácti megabytů.

Tato metoda je od ostatních odlišná, ale i tak ji lze označit za velmi přesnou.

5.6 RequestAnotator

Od ostatních se tato metoda odlišuje tím, že je velice závislá na rychlostech sítě a toků, které se anotují. Zatímco u ostatních trvalo anotovat každý tok stejný čas, u této metody je to výrazně rozdílné. Způsobené se to nastavením proměnné pro odezvu (timeout). Jako příklad uvedu následující situaci. Máme síťové toky z nichž každý desátý je WireGuard a máme k druhému bodu klíče. Potom čas anotace zabere 9*timeout + zpoždění sítě. Timeout je ve výchozí hodnotě roven pěti sekundám a předpokládejme zpoždění sítě dvě stě millisekund, potom každých deset toků trvá anotovat kolem čtyřicetipěti sekund. Pokud by WireGuard provoz byl naopak každý druhý, tak anotace zabere dvacetšest sekund. V reálné síti ovšem WireGuard provozu, ke kterému máme klíče nebude mnoho a tak používání této metody nedává úplně smysl. Lepší výsledky přinese, pokud bychom snížili hodnotu timeout a případně před anotací provedli ještě nějaký filtr toků.

Tato metoda by se dala považovat za nejpřesnější, co se týčew detekce. Potřebujeme ale soukromé klíče a anotace trvá velmi dlouho.

5.7 Klasifikace

Při nasazení společně s frameworkem ALF jsem postupně vyzkoušel úspěšnost strojového učení na všech metodách. Testování jsem prováděl pomocí upravených metod pro detekci DoH ve

frameworku ALF. U BasicAnotator byla na nasbíraných datech úspěšnost vysoká. Ať to bylo na mnou nasbíraných datech, tak na datech od Filipa Němce. Hodnota f1 skóre často přesahovala devadesát procent. Používal jsem různé filtry„ jako IP adresyWireGuard serverů nebo čísla portů pro WireGuard. Ke konci testování jsem udělal experiment, kdy jsem získal pcap soubory ze serveru od Filipa N., které byly síťovým provozem za tři dny. Bylo to kolem tříset tisíc toků a frameworku ALF stačila datová sada o velikosti jen stovek toků, aby f1 skóre bylo kolem devadesáti pěti procent. Soubor s výsledky klasifikace je na přiloženém mediu.

Klasifikaci metodou párování toků jsem prováděl také. Výsledky ovšem aktuálně nebyly uspokojivé. Někdy úspěšnost vycházela dobře – kolem devadesáti procent. Při jiných datech, ale vykazovala i nižší než dvacetiprocentní úspěšnost. Je potřeba s s toky nějak více pracovat, abychom odstranili z anotace toky, které nám budou úspěšnost kazit. To vidím v podobě zašifrovaných toků, ve kterých bylo zapouzdřeno mnoho toků různých služeb a výsledná zvolená dominantní služba není úplně správná. Může to nastat i v situaci, kdy v toku nebyla žádná dominantní služba a všechny měli stejné zastoupení. Potom anotátor zvolí první, ke které se proiteruje.

Kapitola 6

Závěr

Cílem bakalářské práce bylo vytvořit konfigurovatelný modul anotátor pro automatické vytváření datové sady protokolu WireGuard. Pro vytvoření anotátoru byla navržena sada možných indikátorů a i možných přístupů k anotaci vzhledem k dostupným datům. Práce navazuje a využívá některé stávající části Active Learning Framework diplomové práce Jaroslava Peška [3]. Pro účely obecného anotátoru byl framework ALF na některých částech upraven pro lepší kompatibilitu.

Při studiu protokolu WireGuard bylo určeno pět možných způsobů anotace podle určených indikátorů. Tyto kategorie jsou dostatečně obecné a univerzální. Ke každému způsobu jsou potřeba jiná vstupní data a informace. U různých způsobů se liší i jejich přesnost a výkonnost. Nejméně přesný způsob je obecný přístup k datovým tokům, kdy byla prováděna anotace jen pomocí hlaviček. Naopak anotace pomocí čtení logů, provádění dotazu nebo párování paketů byla mnohem více přesná. V kategorii párování paketů je do budoucna velký potenciál pro částečné dešifrování dat protokolu WireGuard a jeho bezpečnostní monitoring, případně bezpečnostní analýzu.

Už během analýzy a návrhu byl kladen důraz na vysokou univerzálnost a kompatibilitu anotátoru pro jeho budoucí využití pro anotaci i jiných protokolů a služeb. Díky univerzálnosti je snadné anotovat i jiný druh síťového provozu než jen WireGuard a během testování bylo provedeno i na jiné služby. Výsledný anotátor je schopen přijímat data přes univerzální rozhraní TRAP, které je i jeho výstupem. Pro jeho funkčnost není potřeba ani existence frameworku ALF, který je zmíněn výše. Při návrhu bylo využíváno výhod objektově orientovaného programování a návrhových vzorů. Výsledný softwarový prototyp je tvořen pěti třídami a je napsán v jazyce Python. Během vývoje byla zjištěna potřeba využívat pro ukládání anotovaných dat databázi. Vytvořený anotátor je tedy schopen přijímat data z databáze a aktualizovat data v ní. Výhoda tohoto přístupu je ušetření výkonových nákladů na dvě TRAP rozhraní (vstup, výstup)

Ve spojení s ALF a vytvořeným klasifikátorem bylo provedeno měření úspěšnosti strojového učení. Výsledky byly slibné. Nejlepší přesnost predikce je u základní metody anotace, kde je kolem devadesáti pěti procent, v ostatních je přesnost velmi podobná až na metodu párování paketů. Měření bylo prováděno v poměrně malé síti, takže reálné výsledky mohou být odlišné

6.1 Budoucí práce

Během implementace jsem narazil na několik možných míst, které lze do budoucna rozvinout a mám v plánu se tomuto projektu i nadále věnovat. Jako nejzajímavější považuji anotování i jiného než WireGuard provozu. Dále pak rozšíření anotátorů o další. Už nyní vím, že někteří kolegové z laboratoře monitorování síťového provozu (CESNET) budou pracovat a případně upravovat navrhnutý párovací anotátor, kdy jeho základ lze použít i k jiným způsobům anotace. Jedním

z míst, kde vidím lehkou slabinu je databáze a stálé používání šablony UniRec. Ideálně bych to nahradil databází, která má již všechna UR políčka jako sloupce. Dále bych se pak chtěl zaměřit na dopracování programu na vícevláknový, aby třeba anotátor aktivního dotazu nezpožďoval následující úlohy. Další rozšíření také vidím na zautomatizování některých procesů, třeba práce s trapcap soubory před samotnou anotací, kdy pro párování toků je potřeba mnoho předchozích kroků. Možné zlepšení rychlosti a možností také vidím ve změně parametrů anotátoru i při běhu anotátoru - načtení nové konfigurace za běhu.

Příloha A

Instalační příručka

Pro zprovoznění programu je zapotřebí programovací jazyk Python, alespoň ve verzi 3.10 dále mít nainstalované knihovny scapy a noiseprotokoll. To se snadno udělá příkazy `pip3 install scapy noiseprotocol`. Poté je potřeba zprovoznit framework ALF, ke kterému je samostatná příručka v citované literatuře. Ve třídě Anotator je potřeba přidat moji implementaci `../code/alf/AnotatorALF.py` a nastavit si potřebné parametry pro komunikaci – především TRAP IFC. Také se hodí frameworku nastavit počáteční datovou sadu bez které budou výsledky náhodné. Datová sada se nachází v `../example_conf/wireguard_D0.csv` Poté už jen stačí pustit můj kód příkazem

```
python3 code/project_ctl.py --dbtype mysql --dbhost localhost --dbuser
    uzivatel --dbpass second --dbname anotatortest --dbtable flowsn --
    urformat "ipaddr DST_IP,ipaddr SRC_IP,int64 BYTES,int64 BYTES_REV,
    int64 LINK_BIT_FIELD,time TIME_FIRST,time TIME_LAST,int64 PACKETS,
    int64 PACKETS_REV,int64 DST_PORT,int64 SRC_PORT,int64 FLOW_END_REASON
    ,int64 PROTOCOL,int64 TCP_FLAGS,int64 TCP_FLAGS_REV,bytes IDP_CONTENT
    ,bytes IDP_CONTENT_REV,int64* PPI_PKT_DIRECTIONS,int64* PPI_PKT_FLAGS
    ,int64* PPI_WG_PKT_SEGNUM,int64* PPI_WG_PKT_TYPE,bytes
    TLS_JA3_FINGERPRINT,string TLS_SNI,int64* PPI_PKT_LENGTHS,int64*
    DBI_BRST_BYTES,int64* DBI_BRST_PACKETS,int64* D_PHISTS_IPT,int64*
    D_PHISTS_SIZES,int64* SBI_BRST_BYTES,int64* SBI_BRST_PACKETS,int64*
    S_PHISTS_IPT,int64* S_PHISTS_SIZES,int64* DBI_BRST_TIME_START,int64*
    DBI_BRST_TIME_STOP,time* PPI_PKT_TIMES,int64* SBI_BRST_TIME_START,
    int64* SBI_BRST_TIME_STOP,int64 packets_sum,int64 bytes_rev,int64
    bytes,int64 dst_port,int64 src_port,int64 packets,int64 packets_rev,
    double bytes_ration,double num_pkts_ration,double time,double
    av_pkt_size,double av_pkt_size_rev,double var_pkt_size,double
    var_pkt_size_rev,double median_pkt_size,double median_pkt_size_rev,
    int64 stSum,int64 ndSum,int64 rdSum,double autocorr,double mindelay,
    double avgdelay,double maxdelay,int64 bursts,int64 fizzles,double
    time_leap_ration,int64 class" --mode 2 --confile ../example_conf/
    conf_uni_basic_wg.xml --urin u:socket1 --urout u:socket2 --urdict ../
    example_conf/dict_translate.txt
```

Bibliografie

1. GOOGLE, Inc. Šifrování HTTPS na webu. 2023. Dostupné také z: https://transparencyreport.google.com/https/overview.
2. CISCO. Cisco Encrypted Traffic Analytics. 2023. Dostupné také z: https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf.
3. PEŠEK, Jaroslav. Framework pro automatické zlepšování klasifikace síťového provozu. 2022. Dostupné také z: https://dspace.cvut.cz/handle/10467/101105.
4. DONENFELD, Jason A. Oficiální webové stránky softwaru WireGuard. 2023. Dostupné také z: https://www.wireguard.com.
5. DONENFELD, Jason A. WireGuard: Next Generation Kernel Network Tunnel. 2020, s. 18. Dostupné také z: https://www.wireguard.com/papers/wireguard.pdf.
6. SURFSHARK B.V., Amsterdam. Oficiální webové stránky společnosti Surfshark. 2023. Dostupné také z: https://surfshark.com.
7. S.A., Republic of Panama nordvpn. Oficiální webové stránky společnosti NordVPN. 2023. Dostupné také z: https://nordvpn.com.
8. ML-based tunnel detection and tunneled application classification. 2022. Dostupné také z: https://doi.org/10.48550/arXiv.2201.10371.
9. CESNET, z. s. p. o. NEMEA: System for network traffic analysis and anomaly detection. 2023. Dostupné také z: https://nemea.liberouter.org/.
10. CESNET, z. s. p. o. ipfixcol2. 2023. Dostupné také z: https://github.com/CESNET/ipfixcol2/tree/master.
11. CESNET, z. s. p. o. ipfixprobe. 2023. Dostupné také z: https://github.com/CESNET/ipfixprobe.
12. FOUNDATION, Open Information Security. Detekční software Suricata. 2023. Dostupné také z: https://suricata.io.
13. MARTY ROESCH Chris Green, Cisco. Detekční software Snort. 2023. Dostupné také z: https://www.snort.org.
14. INC., Sunny Valley Cyber Security. Zenarmor software. 2023. Dostupné také z: https://www.sunnyvalley.io/zenarmor-next-generation-firewall.
15. FREE SOFTWARE FOUNDATION, Inc. GNU General Public License verze 2. 2023. Dostupné také z: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.
16. FREE SOFTWARE FOUNDATION, Inc. GNU General Public License verze 2. 2023. Dostupné také z: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

17. FREE SOFTWARE FOUNDATION, Inc. GNU General Public License verze 2. 2023. Dostupné také z: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.
18. FORTINET, Inc. Forti ASIC. 2023. Dostupné také z: https://www.fortinet.com/products/fortigate/fortiasic.
19. TOMÁŠ ČEJKA, et al. Monitorování na L3+L4 - IP flow, NetFlow, IPFIX. 2022. Dostupné také z: https://courses.fit.cvut.cz/BI-HAM/lectures/bi-ham-p06.pdf.
20. CISCO SYSTEMS, Inc. Cisco IOS NetFlow. 2023. Dostupné také z: https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.
21. SYSTEMS, Cisco. Cisco Systems NetFlow Services Export Version 9. 2004. Dostupné také z: https://www.ietf.org/rfc/rfc3954.txt.
22. PAUL AITKEN Benoît Claise, Brian Trammell. RFC 7011–Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. 2013. Dostupné také z: https://datatracker.ietf.org/doc/html/rfc7011.
23. CESNET, z. s. p. o. UniRec. 2023. Dostupné také z: https://github.com/CESNET/Nemea-Framework/tree/master/unirec.
24. CESNET, z. s. p. o. Nemea-framework. 2023. Dostupné také z: https://github.com/CESNET/Nemea-Framework.
25. CESNET, z. s. p. o. Traffic Analysis Platform. 2023. Dostupné také z: https://nemea.liberouter.org/trap-ifcspec.
26. CESNET, z. s. p. o. Nemea-framework. 2023. Dostupné také z: https://github.com/CESNET/Nemea.
27. FOUNDATION, Wireshark. Oficiální webové stránky softwaru Wireshark. 2023. Dostupné také z: https://www.wireshark.org.
28. GROUP, The Tcpdump. Tcpdump a libpcap. 2023. Dostupné také z: https://www.tcpdump.org.
29. FREE SOFTWARE FOUNDATION, Inc. GNU General Public License verze 2. 2023. Dostupné také z: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.
30. DOROSHENKO, Dmitriy. Protokoly pro virtuální sítě. 2020. Dostupné také z: https://dspace.cvut.cz/handle/10467/88058.
31. ALEXANDER MASTER, Christina Garman. A WireGuard Exploration. 2021. Dostupné z DOI: 10.5703/1288284317610.
32. DONENFELD, Jason A. WireGuard: Next Generation Kernel Network Tunnel. 2020. Dostupné také z: https://www.wireguard.com/papers/wireguard.pdf.
33. BERNSTEIN, Daniel J. ChaCha, a variant of Salsa20. 2008. Dostupné také z: https://cr.yp.to/chacha/chacha-20080128.pdf.
34. BERNSTEIN, Daniel J. The Poly1305-AES message-authentication code. 2005. Dostupné také z: http://cr.yp.to/mac/poly1305-20050329.pdf.
35. YOAV NIR, Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. 2015. Dostupné také z: https://datatracker.ietf.org/doc/html/rfc7539.
36. PERRIN, Trevor. The Noise Protocol Framework. 2018. Dostupné také z: http://noiseprotocol.org/noise.pdf.
37. BERNSTEIN, Daniel J. Curve25519: new Diffie-Hellman speed records. 2006. Dostupné také z: https://cr.yp.to/ecdh/curve25519-20060209.pdf.
38. MARKKU-JUHANI O. SAARINEN, Jean-Philippe Aumasson. RFC7693–The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). 2015. Dostupné také z: https://datatracker.ietf.org/doc/html/rfc7693.

39. HUGO KRAWCZYK, Pasi Eronen. RFC5869–HMAC-based Extract-and-Expand Key Derivation Function (HKDF). 2010. Dostupné také z: https://datatracker.ietf.org/doc/html/rfc5869.
40. PEŠEK, Jaroslav; SOUKUP, Dominik; ČEJKA, Tomáš. Active Learning Framework For Long-term Network Traffic Classification. 2023. ISBN 979-8-3503-3286-5. Dostupné z DOI: 10.1109/CCWC57344.2023.10099065.
41. VERISIGN, Inc. WHOIS Protocol Specification. 2004. Dostupné také z: https://www.rfc-editor.org/rfc/rfc3912.
42. SURFSHARK B.V., Amsterdam. API společnosti Surfshark. 2023. Dostupné také z: https://api.surfshark.com/v4/server/clusters.
43. S.A., Republic of Panama nordvpn. API společnosti NordVPN. 2023. Dostupné také z: https://api.nordvpn.com/v1/servers.
44. FOUNDATION, Python Software. Python. 2023. Dostupné také z: https://www.python.org.
45. W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). 2008. Dostupné také z: https://www.w3.org/TR/xml/.
46. CESNET, z. s. p. o. Nemea pytrap. 2022. Dostupné také z: https://pypi.org/project/nemea-pytrap.
47. Nová licence BSD. [B.r.]. Dostupné také z: https://opensource.org/license/bsd-3-clause.
48. ORACLE. MySQL. 2023. Dostupné také z: https://www.mysql.com.
49. FOUNDATION, The Apache Software. Apache Cassandra. 2023. Dostupné také z: https://cassandra.apache.org.
50. GROUP, PostgreSQL Global Development. PostgreSQL. 2023. Dostupné také z: https://www.postgresql.org.
51. ORACLE. knihovna mysql-conn-oracle. 2023. Dostupné také z: https://pypi.org/project/mysql-connector-python/.
52. ORACLE. knihovna mysql-conn-oracle. 2023. Dostupné také z: https://dev.mysql.com/downloads/connector/python/.
53. BIONDI, Philippe. knihovna Scapy. 2023. Dostupné také z: https://scapy.net.
54. FREE SOFTWARE FOUNDATION, Inc. ISO 8601 Date and time format. 2019. Dostupné také z: https://www.iso.org/obp/ui/#iso:std:iso:8601:-1:ed-1:v1:en.
55. LIZOŃCZYK, Piotr. knihovna noiseprotocol. 2020. Dostupné také z: https://github.com/plizonczyk/noiseprotocol.
56. LIZOŃCZYK, Piotr. ukázky použití knihovny noiseprotocol. 2020. Dostupné také z: https://github.com/plizonczyk/noiseprotocol/tree/master/examples/wireguard.
57. BIONDI, Philippe. MIT licence knihovny noiseprotocol. 2017. Dostupné také z: https://github.com/plizonczyk/noiseprotocol/blob/master/LICENSE.
58. FOUNDATION, Python Software. Regular expression operations. 2023. Dostupné také z: https://docs.python.org/3/library/re.html.

Obsah přiloženého média

```
thesis.pdf ............................................ práce v pdf
thesis ....................... adresář se zdrojovými kódy LaTeX k práci
code ................................... adresář se zdrojovými kódy
  alf ........................................................ adresář
    anotatorALF.py ........... třída s implementací pro framework ALF
  src ........................................................ adresář
    anotator.py ........................................ třída Anotator
    core.py ................................................ třída Core
    database.py ........................................ třída Database
    link.py ................................................ třída Link
    parser.py .......................................... soubor parser
  project_ctl.py ............................ spouštění celého programu
example-conf ............................................. ukázkový běh
  thesis.pdf ........................... text práce ve formáe PDF
```