FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Detekce VPN provozu pomocí automatu
Student:	Bc. Jan Jirák
Vedoucí:	Ing. Karel Hynek
Studijní program:	Informatika
Obor / specializace:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Nastudujte si používané VPN protokoly a zaměřte se na možnosti jejich detekce. Prozkoumejte charakteristické vzory VPN komunikace a navrhněte algoritmus na principu konečného, zásobníkového nebo jiného automatu, který dokáže tyto vzory rozpoznávat v reálném síťovém provozu a detekovat tak přítomnost VPN. Integrujte navržený algoritmus do exportéru síťových toků ipfixprobe [1]. Navržené řešení otestujte a vyhodnoťte jeho propustnost a přesnost detekce na datových sadách poskytnutých vedoucím práce.

[1] https://github.com/CESNET/ipfixprobe

Elektronicky schválil/a doc. Ing. Jan Janoušek, Ph.D. dne 24. listopadu 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Detekce VPN provozu pomocí automatu

Bc. Jan Jirák

Katedra teoretické informatiky
Vedoucí práce: Karel Hynek

3. května 2023

Poděkování

Děkuji skvělému vedoucímu Ing. Karlu Hynkovi za veškeré rady a pomoc při vypracování této diplomové práce. Dále je nutné poděkovat své rodinně bez které by nemohla tato práce vůbec vzniknout.

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. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 3. května 2023 .....

Odkaz na tuto práci

Jirák, Jan. *Detekce VPN provozu pomocí automatu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Tato práce se zabývá problematikou detekce VPN provozu z pohledu monitoringu sítě a návrhem automatu, který tuto detekci provádí. Po provedení řešerše nastiňuje teoretické řešení založené na automatu zachycujícím typický obraz TCP handshaku v reálném provozu. Toto řešení je následně prozkoumáno detailněji v prototypu v jazyce Python na datech zachycených vedoucím práce. Po zjištění teoretické úspěšnosti na těchto datech je prototyp naimplementován do exportéru IPFIXprobe a tato implementace je opět otestována na reálných datech.

Klíčová slova VPN, klasifikace síťového provozu, automat, detekce, IDS, vysokorychlostní sítě, Python, IPFIXprobe, C++

Abstract

This work deals with the issue of VPN traffic detection from the view of network monitoring. An automaton that will perform this detection is designed. After researching other solutions for this problems, new solution based on an automaton capturing a typical representation of a TCP handshake in real traffic is proposed. This solution is then explored in more detail in a Python prototype based on the data captured by the supervisor. After establishing the theoretical performance on these data, the prototype is implemented into the IPFIXprobe exporter and this implementation is tested again on real data.

Keywords VPN, classification of network traffic, automaton, detection, IDS, highspeed networks, Python, IPFIXprobe, C++

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době kdy provoz na síti roste každým rokem a očekává se, že dvě třetiny celkové populace budou připojeny k internetu už v roce 2023 (podle odhadů Cisca [1]), je téma monitoringu provozu sítě stále aktuálnější. Monitoring slouží jak k tvorbě statistik nad postavenou síti, tak i k zachycení a případně zastavení nechtěného provozu. Může tedy sloužit i jako forma cenzury.

Stále se vede nekonečný souboj jak tento monitoring obejít a jak se obejití bránit. Velice často se používá technika, kdy je zakázaný provoz zabalen do legitimního provozu a tedy my musíme vyvinout úsilí navíc abychom toto rozpoznali. Příklady takového legitimního provozu jsou HTTP, HTTPS, DNS, SSH a moderní VPN technologie které mohou být k maskování škodlivého provozu použity.

Toto je i případ malwaru, kde se infikovaný program snaží vynést informace z privátní sítě vytvořením takzvaného backdooru neboli zadních vrátek. Provoz v tomto backdooru se schová do tunelu, který je maskovaný legitimním provozem.

Tato diplomová si klade za ambici vyvinutí nové techniky pro detekci takového tunelování se zaměřením na VPN technologie. Využívá hlavně jedné vlastnosti těchto tunelů a to, že paket jdoucí do VPN tunelu se zobrazí právě na jeden paket viditelný na síti a zároveň při zachování šifrování (obecně nastavení tunelovacího protokolu) bude vždy zobrazen na paket stejné délky. My budeme hledat častou sekvenci paketů při navazování TCP spojení. Tomuto navazování TCP spojení se říká TCP handshake a ten obsahuje sekvenci tří malých paketů s alternujícími směry.

Práce je rozdělena do dvou částí a to teoretické a praktické. Teoretická část nejdříve představí čtenáři základní pojmy a technologie. Konkrétně bude vysvětlena problematika VPN a monitoringu sítě pomocí flow technologie. Dále budou představeny známé techniky, jak řeší detekci tunelů další autoři a do tohoto kontextu si zasadíme naše teoretické řešení. Uvedeme také přehledné

srovnání výhod a nevýhod konkrétních postupů.

V praktické části nejdříve ukážeme implementaci základního algoritmu odladěném na zachycených datech ze sítě CESNET2 a určíme jeho přesnost. Následně naší implementaci přeneseme i do reálného prostředí vysokorychlostních sítí pomocí flow sondy IPFIXprobe a experimentálně vyhodnotíme i úspěšnost zde.

V úplném závěru ještě jednou shrneme výsledky našeho algoritmu ve srovnáni s předchozími přístupy a pokusíme se nastínit směry, které tato práce otevírá a možné vylepšení.

Kapitola 1

Teoreticka příprava

V této kapitole se budeme zabývat teoretickou přípravou, která bude potřeba v praktické části. Začneme nejdříve popsáním problematikou obecných tunelů, poté probereme detailněji konkrétní VPN protokoly. Také si řekneme něco obecného k monitoringu sítě a zaměříme se na monitoring pomocí flow záznamů, který se používá na vysokorychlostních sítích. V závěru kapitoly si nastíníme metody detekce tunelů a jejich výhody a nevýhody.

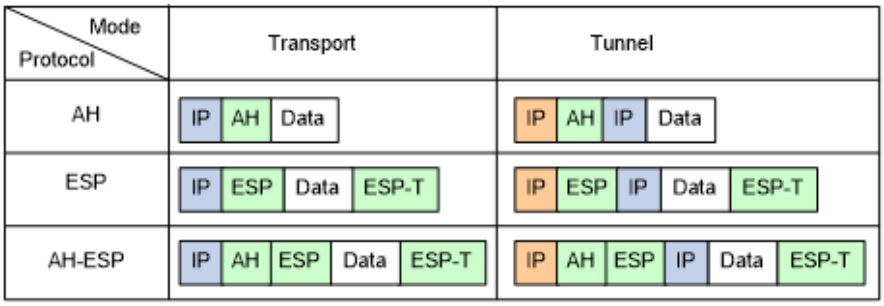
1.1 Síťové tunely

Z [2] víme, že tunelování sítě je technika, která se používá k obalení jednoho síťového protokolu (například nepodporovaného) uvnitř druhého. Toho se dosáhne tak, že se vytvoří dvoubodové spojení mezi koncovými body a pakety prvního protokolu jdoucí z jednoho konce se zabalí do protokolu druhého a na přijímacím konci se opět rozbalí do paketů původního protokolu. Takto lze poskytnout třeba vyšší zabezpečení komunikace nebo obejít bezpečnostní pravidla sítě. Většina síťových zařízení jako jsou routery pracuje pouze s hlavičkami obalujícího paketu.

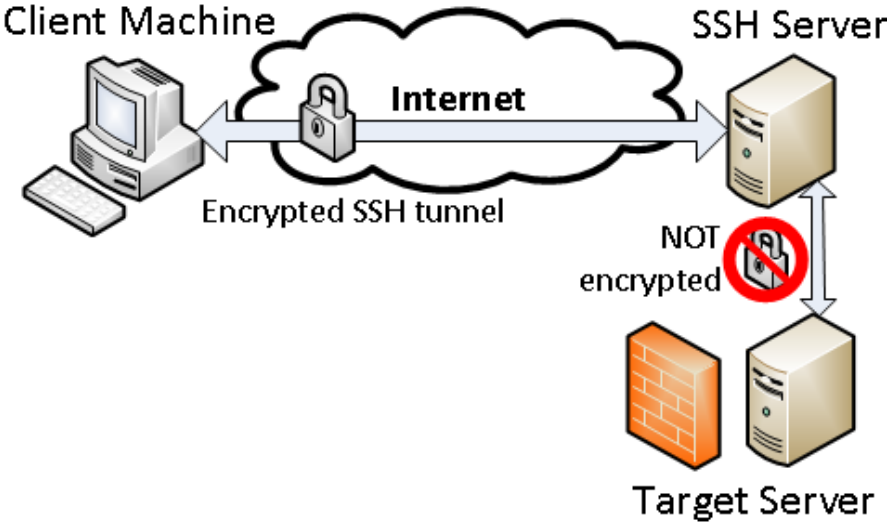
Příklady tunelovacích protokolů jsou

- IPv6 over IPv4
- Secure Shell (SSH)
- Point-to-Point Tunneling (PPTP)
- Internet Protocol Security (IPsec)

Tunelovací protokoly mohou pracovat ve dvou módech, detailní vysvětlení je k nalezení v [3]. „Tunnel mode“ funguje tak, že zabalí celý paket včetně hlavičky. Tento mód je použit v případě, že je potřeba skrýt všechny informace včetně IP adres. Zajišťuje plnou ochranu. Naproti tomu „transport mode“ IP hlavičku nechá původní. Tedy v transport módu je chráněný pouze obsah



Obrázek 1.1: Ukázka fungování IPsec z [3]



Obrázek 1.2: Ukázka fungování SSH tunelu z [4]

paketu a hlavička není, paket je nadále odeslán přímo na místo doručení. U „tunnel mode“ je chráněna i IP hlavička, z toho plyne, že paket nemůže být odeslán přímo na koncovou stanici, ale musí být po cestě rozbalen a přeposlán s dešifrovanou IP hlavičkou.

Na obrázku 1.1 je znázorněno fungování protokolu IPsec. IPsec má dva módy a to AH (Authentication Header) a ESP (Encapsulating Security Payload), ty mohou být zároveň kombinovány (IPsec bude podrobněji rozebrán v další sekci). Tento obrázek ilustruje jednoduchost vytvoření tunelu. Opravdová data jsou pouze zabalena do ESP nebo AH hlavičky a v závislosti na Transport/Tunnel módu je přidána nová IP hlavička.

Na obrázku 1.2 můžeme vidět praktickou ukázku použití tunelu. klient naváže SSH tunel s serverem v privátní síti za firewallem. Přes tento tunel může následně bezpečně posílat data, které patří cílovému serveru a ví, že útočník jeho data nemůže pozměnit ani přečíst. Tyto data jsou rozbalena až v bezpečném prostředí privátní sítě a jsou přeposlána SSH serverem na cílový

server.

1.2 VPN tunely

VPN neboli „Virtual private network“ slouží k vytvoření bezpečného spojení do vzdálené privátní sítě přes nezabezpečenou síť veřejnou. Funguje na principu vytvoření tunelu k VPN serveru, který dále přeposílá všechny požadavky na cílový server. VPN služba splňuje nutně tyto body [5].

1. Útočník, který je schopen odposlechu našich paketů, stále není schopný je dešifrovat.
2. Útočník není schopen pozměnit paket bez toho aniž bychom to zjistili.
3. Útočník není schopen podvrhnout reálný provoz.
4. Útočník je schopen zahodit náš paket na cestě, ale už není cíleně schopen zahazovat jen určité pakety. (protože je není schopen rozeznat od zbytku).

Hlavní výhoda této služby je, že nikdo kromě poskytovatele VPN služby nemůže vidět naší aktivitu na síti. Dokonce ani náš ISP (Internet Service Provider) ne. Na veřejné síti je vidět pouze jedno šifrované spojení s VPN poskytovatelem. Pokud VPN poskytovatel nedrží žádné logy o vašem provozu jste prakticky nepostřehnutelní za své aktivity na síti, protože neexistují žádné její záznamy a váš provoz je nečitelný.

Toto ale může vést snadno k zneužití a tímto způsobem lze obcházet nastavená bezpečnostní pravidla. Většina firewallů si s VPN neporadí.

Mnohé firemní sítě mají například nastaveny přísnější pravidla, aby nebylo možné třeba do firemního intranetu stáhnout malware z vnějšku. Totéž platí dvojnásob i pro armádní sítě atd. VPN jsou zakázány i v cenzuru podporujících státech jako jsou Bělorusko, Čína, Irák, Severní Korea, Omán, Rusko [6].

Mezi další výhody VPN patří změna regionu odkud se dotazujete na různé zdroje. Hodně zdrojů na internetu je právě podmíněné geolokací odkud přistupujeme k internetu (např. různé streamovací platformy jako je Netflix nabízejí obsah podle národnosti). Tím, že použijeme VPN server třeba v USA vytváříme iluzi, že jsme odtud a maskujeme naši skutečnou geolokaci. Cílový server to nemá jak rozeznat a na základě IP adresy serveru v USA si opravdu bude myslet, že jsme odtamtud.

Mezi nejčastější použití VPN služby patří stále ale firemní použití [7], kdy se pracovník potřebuje vzdáleně připojit k intranetu a je mu tedy zřízen VPN přístupový bod a přes něj bezpečně přistupuje k zdrojům na intranetu jako by byl fyzicky připojen v síti.

1.3 Síťové protokoly

Pro další odstavce této kapitoly je potřeba ukázat referenční model OSI obrázek (1.3) vyvinutý roku 1984. Z [8] víme, že OSI (Open System Interconnection) je konceptuální model, který popisuje, jak se informace dostává ze softwarové aplikace přes fyzické médium po síti do další aplikace. OSI model se skládá ze sedmi modulárních vrstev. Těchto sedm vrstev je navržených tak, aby na sobě byly nezávislé a tedy mohla být vyměněna jen část celého komunikačního protokolu.

Průběh poslání paketu z jedné aplikace do aplikace druhé běžící na síti probíhá tak, že aplikace na sedmé vrstvě vytvoří obsah, který chce poslat, a ten projde postupně každou vrstvou OSI modelu. V každé vrstvě dojde k přidání nové hlavičky s novými informacemi. Takto se paket zabalí až do fyzické vrstvy, odkud je poslán přes síť k cíli.

Na cestě k cíli mu ještě mohou být změněny hlavičky nižších vrstev. Pokud například paket musí přes gateway do jiné sítě bude mu určitě změněna zdrojová MAC adresa na druhé vrstvě na MAC adresu routeru na této gateway. Nebo při port-forwardingu u NATu [9] bude měnit router i zdrojovou IP adresu ležící na třetí vrstvě na svoji IP adresu.

Jakmile paket dorazí na cílovou stanici projde všemi vrstvami OSI modelu od fyzické vrstvy až po aplikační a v každé vrstvě se zpracuje hlavička a poté se odstraní. Po průchodem všech sedmi vrstev odspodu navrch tedy cílová aplikace dostane přesně ten obsah, který byl ze zdrojové aplikace odeslán.

1.3.1 TCP (Transmission Control Protocol)

TCP je protokol čtvrté vrstvy OSI modelu, který je zabalen do IP protokolu. Je to asi nejpoužívanější protokol internetu vůbec.

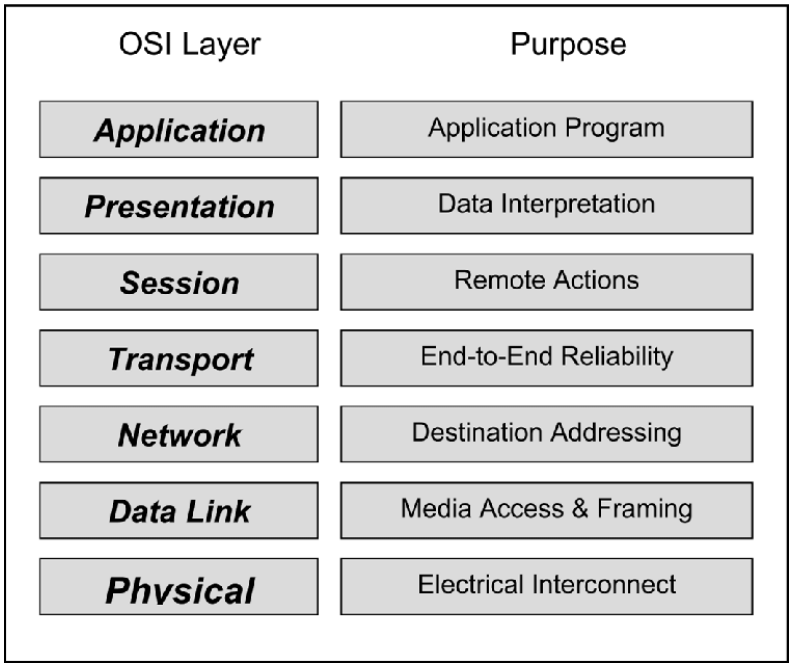
Hlavní funkcí tohoto protokolu je spolehlivé doručení všech seřazených paketů k cíly [11]. K tomuto účelu TCP vytvoří spojení a pomocí mechanismů jako je potvrzování zpráv (acknoledgement), timeoutů a znovu posílání nedoručených paketů (retransmission) nám garantuje doručení. Další funkcí je kontrola zahlcení sítě.

Průběh přenosu má typicky tři fáze, které jsou detailněji vysvětlené v [11].

1. TCP establish (3-way handshake)
2. TCP transmission
3. TCP release

Nejdřív je potřeba navázat spojení mezi klientem a serverem. K tomu právě dojde ve fázi TCP establish. Tato fáze, zobrazená na obrázku 1.5, probíhá následovně.

Klient nejříve pošle serveru TCP paket s flagem SYN a tento paket signalizuje žádost o navázání spojení. Detail paketu můžeme vidět na obrázku



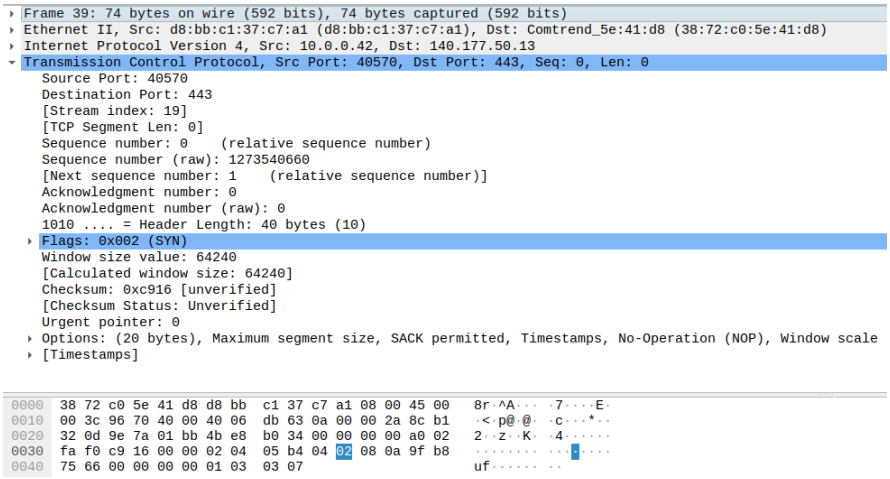
Obrázek 1.3: OSI model z [10]

1.4, ale pro nás to hlavní jsou Source Port (zdrojový port), Destination Port (cílový port), Flags a Options. Číslo SEQ slouží k bezpečnostním účelům, je to náhodně vygenerované číslo, které je nemožné pro útočníka uhodnout.

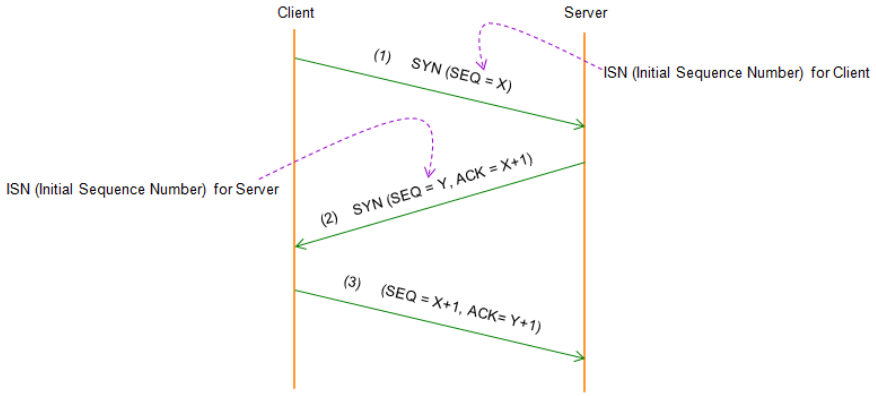
Do pole Flags, které je složeno z 12 bitů, je zapsán typ a vlastnosti TCP paketu. Do pole Options je možné přidat různý přídavné parametry pro TCP spojení a pole Options tedy nabízí dodatečnou kontrolu nad protokolem, která je zapotřebí pro stále rychlejší linky. Mezi příklady uvedeme pouze Maximum Segment Size (MSS), který určuje maximální délku dat, které jsou zabaleny do TCP protokolu. Souhrnné vysvětlení se nachází například na [12]. Pro další kapitoly bude také potřeba znát délku TCP hlavičky. Vzhledem k tomu, že velikost pole Options se pohybuje v rozmezí 0-40 bytů (násobky 8 bytů), a bez toho sama hlavička má pevně 20 bytů, celková velikost se je 20-60 bytů. Na obrázku 1 je znázorněno i Acknoledgement number (ACK), které vznikne přičtením jedničky k SEQ. Pozor neplést s ACK flagem, zde je to něco jiného.

V druhém kroku server od klienta obdrží SYN paket a odpoví SYN-ACK paketem. Tedy jedním paketem s bity SYN a ACK nastavenými na 1. Pro nás je zde důležité, že i SYN-ACK obsahuje pole Options a z toho vyplývá, že i tento paket má velikost 20-60 bytů. Důležité je podotknout, že Options u SYN a u SYN-ACK paketů se teoreticky mohou lišit, tedy i jejich velikosti se mohou zásadně lišit.

V třetím kroku odešle klient serveru potvrzení ACK paketem s flagem ACK nastaveným na 1. Teoreticky může i ACK paket obsahovat větší množství



Obrázek 1.4: TCP syn paket (Wireshark)



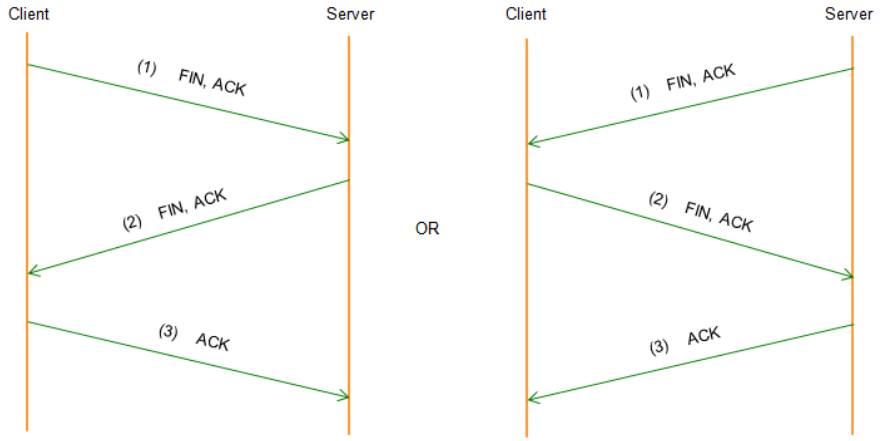
Obrázek 1.5: TCP handshake z [11]

Options, ale v praxi většinou jich posílá pouze pár a měl by tedy být většině případů menší než SYN a SYN-ACK pakety. Tuto vlastnost ověříme experimentálně v kapitole 2.

Po úspěšném TCP handshaku může klient a server normálně komunikovat. To probíhá do té doby než se rozhodne server nebo klient ukončit toto spojení. Standardní ukončení probíhá přesně podle obrázku 3. U nestandardního ukončení, kdy například server přestane odpovídat a vyprší timeout, klient pouze pošle RST paket a spojení ukončí.

1.3.2 IPsec

IPsec [13] neboli „Internet Protocol security“ je standartní sada protokolů, vyvíjena společností Internet Engineering Task Force (IETF), která slouží pro bezpečný přenos informací po síti. V dnešní době je IPSec standartně



Obrázek 1.6: TCP release z [11]

podporovaný většinou operačních systémů a síťových zařízení. Narozdíl od Wireguardu obsahuje kromě protokolů pro zašifrování a autentizaci paketu i protokoly pro management klíčů.

IPsec je implementován na třetí nebo čtvrté vrstvě OSI modelu. Má také dva rozdílné módy a to „Tunnel mode“ a „Transport mode“. „Tunnel mode“ standartní mód VPN služby, který šifruje nejen obsah paketu ale i IP hlavičku. Transport mode naopak zašifruje pouze obsah paketu a IP hlavička zůstává nezměněná, takže je na síti vidět skutečný cíl paketu.

IPSec obsahuje následující protokoly [13]:

- Authentication Header (AH) - zajišťuje, že přenášená data pochází z důvěryhodného zdroje a nebyla modifikována, toto je zajištěno pomocí takzvaného MAC kódu (Message authentication code)
- Encapsulating Security Payload (ESP) - zajišťuje utajenost dat pomocí šifrování paketu
- Security Association (SA) - definuje parametry, které budou použity pro konkrétní spojení, například šifrovací klíč, algoritmus pro výměnu klíče nebo autentizační metodu

1.3.3 OpenVPN

OpenVPN [14] je VPN implementace založená na SSL/TLS protokolech, což jsou nejpoužívanější protokoly na webu, které slouží pro autentizaci a šifrování komunikace. OpenVPN může fungovat jak v klient-server, tak v peer-to-peer režimu.

Oproti IPSec je OpenVPN flexibilnější a jednodušší na nasazení do provozu díky jednoduché konfiguraci, zato ale je také pomalejší kvůli závislosti na

SSL/TLS protokolech. Mezi její další výhody patří i to, že je implementována pouze v uživatelském prostoru a tedy je více platformově nezávislá.

1.3.4 Wireguard

Tento protokol [15] je moderní nástupce IPsecu a OpenVPN. Klade si za úkol hlavně to, aby byl rychlý a zároveň měl jednoduché rozhraní i implementaci (což se mu i daří, jeho implementace má pouze okolo 4000 řádků a je tedy snadno udržitelná). Původně byl vydán pro Linux, ale dnes už je podporován na většině platformách. Jeho infrastruktura je založená na výměně veřejných klíčů. Je i implementován v jádře Linuxu, což mu dovoluje dosáhnout větší výkonnosti než konkurentům. Další jeho výhodou je, že podporuje pouze malou podmnožinu kryptografických algoritmů (Curve25519, ChaCha20, Poly1305, BLAKE2, SipHash24, HKDF), které byli prověřeny moderními přístupy a nehrozí tedy, že by bezpečnost závisela na jeho konfiguraci. Tyto algoritmy byli zároveň vybrány pro svojí rychlost. V neposlední řadě je Wireguard implementován stylem, aby byl co nejméně odhalitelný. Tedy například Wireguard server vůbec neodpovídá na neautentizované pakety a je tedy odolný vůči aktivním přístupům detekce.

1.4 Monitoring sítě

Monitoring sítě je proces sběru dat ze síťových zařízení a jejich následná analýza. S rostoucí komplexitou počítačových sítí, která je způsobena jak většími počty uživatelů, většími počty nových útoků, tak i složitějšími síťovými službami a hardwarovými zařízeními, vzrůstá potřeba tohoto monitoringu. Vzhledem k tématu této diplomové práce se zde zaměříme pouze na monitoring bezpečnosti sítě a další aspekty jako jsou například výkon zanedbáme.

V této sekci si popíšeme různé přístupy k monitoringu sítě. Tato sekce je detailněji popsána v [16].

1.4.1 NIDS (Network Intrusion Detection System)

NIDS je systém, který slouží pro detekci a reakci na bezpečnostní hrozbu pro síťové zařízení. NIDS systémy řadíme do dvou kategorií [17].

1. Detekce signatur - U tohoto přístupu využíváme nějakou databázi signatur, která vzniká většinou tak, že v minulosti byl rozpoznán útok a my si definuje jeho signaturu. Pokud aplikujeme tuto signaturu na všechny zachycené pakety a nějaký této signatuře odpovídá, můžeme s velkou jistotou vydat varování.

2. Detekce anomálií - U tohoto přístupu se vytvoří statistický model validního provozu (klidně i složitější strojové učení) a každou signifikantní odchylku hlásíme jako hrozbu.

Kdybychom měli porovnat porovnat hlavní rozdíl v těchto dvou přístupech tak musíme říci, že detekce signatur je velice přesná ve smyslu, že pokud prohlásí něco za škodlivé, je toto tvrzení důvěryhodné. Oproti tomu řešení založené na detekci anomálií sice není tak přesné (může mít vysoký false positive rate) zato nepotřebuje ke svému fungování žádnou databázi a umí rozpoznat i zero-day útoky (nikdy nedetekované).

Dále rozlišujeme monitoring na dva typy a to monitoring paketů a flow monitoring. Toto je rozlišení podle toho v jakém formátu se na data ze sondy díváme.

1.4.2 Monitoring na úrovni paketů

Monitoring na úrovni paketů nebo také DPI (deep paket inspection) pracuje na principu sondy, která pro každý paket který zachytí prohlédne jeho obsah a porovnává ho s databází signatur [18]. Je zde také možnost brát do úvahy širší kontext a pamatovat si například statistiky posledních n paketů. Tento přístup bohužel nelze použít na rychlejších sítích s větší propustností a to z důvodu velkých výpočetních a paměťových nároků. V tu chvíli co máme databázi o tisících nebo dokonce milionech signatur a vysokorychlostní sít s rychlostí v jednotkách Gbps je nepředstavitelné, že by sonda stíhala zkusit analyzovat všechny pakety. Většina sond může pracovat maximálně v stovkách Mbps. Další nevýhodou je, že tato technika velice těžko analyzuje šifrovaný provoz.

Tradičním zástupcem tohoto přístupu je Snort[19]. Snort můžeme použít jako plnohodnotný IDS, paket sniffer nebo paket logger. Je založený na libpcap (Linux) nebo winpcap (Windows) knihovnách. Funguje tak, že mu poskytneme konfigurační soubory s pravidly a on je přehraje oproti reálnému provozu nebo například zachycenému pcapu. Tato pravidla mají následující formu:

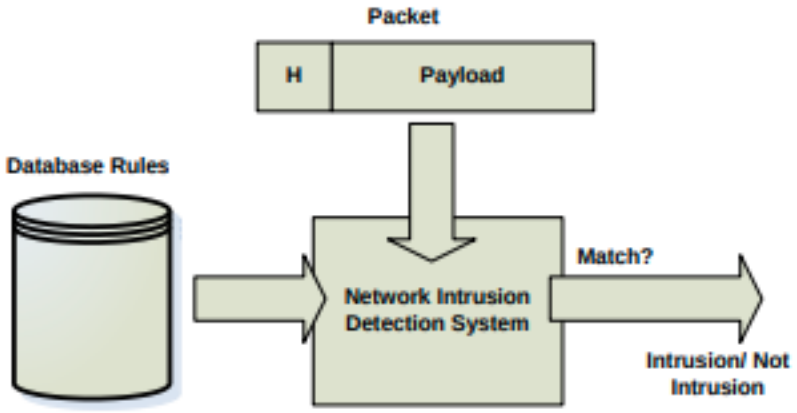
```
alert udp any 71 -> 10.255.177.1 1
(msg: "raise alert!", sid:10000)
-------------------------------------
akce protokol srcIP srcPort -> dstIP dstPort
hláška pro upozornění
```

Demonstraci principu můžeme vidět na obrázku 1.7.

1.4.3 Flow monitoring

Flow monitoring je technika monitoringu sítě pomocí flow záznamů [20]. Flow monitoring byl zaveden hlavně kvůli vysokorychlostním sítím. Zde místo toho abychom pracovali s každým paketem zvlášť, budeme uvažovat jen menší množství zvolených statistik ke každému flow záznamu. Toto nám výrazně zmenší časovou a paměťovou složitost analýzy.

Definice flow záznamu se často liší podle článku nebo autora. Podrobnější vysvětlení lze nalézt v [21]. My si zavedeme flow záznam následovně.



Obrázek 1.7: Ukázka monitoringu paketů z [16]

Flow je tok dat mezi dvěmi koncovými procesy na síti reprezentován čtveřicí (IP1, port1, IP2, port2), kde IP1 je IP adresa prvního procesu a port1 jeho příslušející port a obdobně IP2 a port2 pro druhý proces. S touto definicí je možné uvažovat za flow záznam i například komunikaci dvou procesů na stejném počítači. To ale žádná síťová sonda nezachytí, takže tento případ nemusíme uvažovat.

Příklady statistik, co můžeme u flow záznamu evidovat jsou:

- Čas mezi prvním a posledním paketem
- Velikost přenesených dat
- Histogram rozdělení délek paketů
- Histogram mezi paketových mezer (v sekundách)

Výhody oproti tradičnímu paketovému řešení jsou velmi malá výpočetní a paměťová složitost. Zároveň nejsme schopni rozpoznat útoky, které jsou vidět pouze v obsahu paketu a nepromítnou se do flow záznamu. Flow monitoring má zároveň menší přesnost [22].

Autor článku [16] přináší hybridní řešení při kterém se použijí obě techniky. V první fázi označí flow technika podezřelé toky dat a následně takto označené toky jsou podrobené paketové inspekci. Touto technikou se autor snaží zredukovat false positive rate, který může být typicky u některých flow technik vysoký, a tedy i zlepšit přesnost.

Mezi další výhody tohoto přístupu patří bezesporu to, že mnoho routerů má už samo o sobě zaintegrovanou možnost pro export flow záznamů a tedy je jejich nasazení mnohem levnější než klasické DPI. Další výhodou je i to, že data jsou mnohem více anonymizovaná a z toho mohou těžit i mnozí poskytovatelé připojení, kteří jsou nuceni uschovávat záznamy komunikace i po roky.

Nejrozšířenější protokoly jsou NetFlow od společnosti Cisko a IPFix od společnosti IETF.

V [20] autoři podávají ucelený pohled na nasazení a implementaci flow monitoringu. My si vytáhneme z tohoto článku jen nutné minimum a to obecnou architekturu flow řešení.

Typická flow architektura se skládá ze čtyř částí a to konkrétně

1. Zachycení paketu
2. Export flow záznamu
3. Sběr dat na kolektoru
4. Analýza

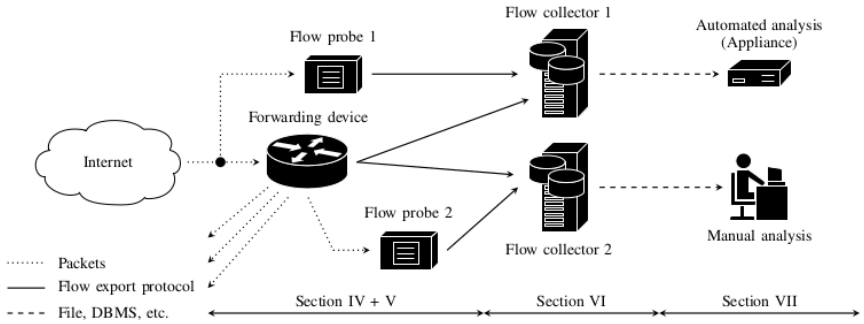
V fázi 1 jsou zachyceny sondou pakety a jsou zde také předzpracovány. Sonda zde může být například síťová karta nebo rozhraní zařízení, které pakety dále přeposílá, například rozhraní routeru.

V fázi 2 proběhnou dvě věci, nejdříve jsou pakety roztříděné do jednotlivých flow záznamů a po ukončení flow záznamu (například z důvodu ukončení TCP spojení nebo vypršení nějakého nastaveného timeoutu) jsou flow data předána flow export protokolu a odeslána na síťový uzel (kolektor), který slouží k jejich agregaci. Flow data která jsou exportována si můžeme přestavit jako tabulku databáze, kde jeden řádek odpovídá jedné sledované statistice flow záznamu (může to ale být i třeba zdrojový port).

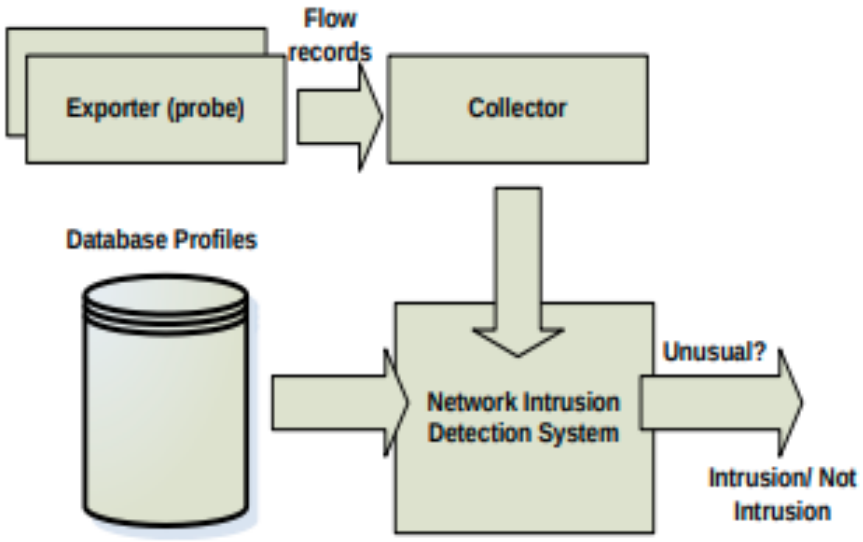
V fázi 3 dojde k příjmu dat kolektorem, který data uloží a začne je předzpracovávat. Nejběžnější formy předzpracování flow záznamu jsou agregace, filtrování, komprese a generování souhrnu.

V fázi 4 jsou předzpracovaná flow data podrobené buď manuální nebo automatické analýze. Záleží na konkrétních potřebách. Například mohou být ručně vyhodnocena data, které nám ukazují jak jsou zatěžovány servery v největším provozu. My se pak podle toho rozhodneme jak nastavit loadbalancer. Automaticky naopak mohou být vyhodnocena data pro analýzu, jaký provoz nám vůbec na síti běží nebo pro automatickou detekci hrozeb.

Pro větší názornost se podívejme na obrázek 1.8, kde v jednom případě přijdou pakety z Internetu do flow sondy 1, ta je po agregaci přepošle na flow kolektor 1 a z něj může tyto předzpracované data využít například aplikační server s automatickou analýzou. Je také důležité podotknout, že díky tomu, že máme proces flow monitoringu rozdělený do více fází je relativně jednoduché měnit naši architekturu. Například pokud máme více exportérů a více kolektorů, může konkrétní exportér posílat své data do více kolektorů a konkrétní kolektor může přijímat od více exportérů.



Obrázek 1.8: Ukázka flow architektury z [20]



Obrázek 1.9: Ukázka flow monitoringu z [16]

1.5 Detekce tunelů

Detekce tunelů je obecně obtížný problém, protože většina z tunelovacích protokolů je navržena tak, aby byla šifrovaná a zabezpečená. Dnes se detekce provádí mnoha způsoby. My se zaměříme na hlavní přístupy detekce a porovnáme jejich výhody a nevýhody.

Mezi tradiční přístupy patří následující:

- Deep packet inspection (DPI) - Pomocí DPI lze rozeznat některé tunelovací protokoly pouze na základě signatur paketů. Je také možné rozeznat například použití konkrétního šifrovacího algoritmu a toto použít jako jeden z indikátorů.
- Detekce na základě portů - Některé tunelovací protokoly mají často aso-

ciované porty, které používají. Toho lze vyžít a tyto porty blokovat (sice má většina tunelovacích protokolů možnost si port zvolit, ale to už někdy vyžaduje speciální nastavení, které ne všichni dělají).

- Reputace IP adresy - Existují algoritmy, které na internetu klasifikují reputaci IP adresy. Z [23] víme, že tyto algoritmy mohou například uvažovat škodlivé IP adresy, které jsou spojované se škodlivým provozem. Mnoho poskytovatelů VPN služeb používá neměnící se množinu IP adres a ta může získat špatnou reputaci. Nebo do reputace může být vzato v potaz, jestli IP adresa známá jako adresa patřící VPN poskytovateli.

Mezi pokročilejší metody detekce tunelů patří metody využívající strojové učení a fingerprinting síťového provozu. V následujících dvou sekcích si představíme zástupce těchto principů.

1.5.1 Detekce VPN pomocí strojového učení

Začněme nejdříve aktuálním článkem Detection of VPN Network Traffic [24] z roku 2022, který tvrdí, že má lepší výsledky než jeho předchůdci. Tento přístup je založený na strojovém učení nad flow záznamy.

Je zde použit dataset ze společnosti Canadian Institute for Cybersecurity s názvem „VPN-nonVPN Dataset“, který byl zachycen standardně pomocí nástroje Wireshark. Jedná se tedy o dataset s celými pakety. Pro jeho záchyt zde byl použit OpenVPN server (v UDP nastavení). Dataset se snaží být rozmanitý a obsahuje mnoho kategorií provozu jako je například Streaming, VoIP, Chat, Command & Control, File transfer. Tyto protokoly jsou v datasetu někdy tunelovány přes VPN a jindy se vyskytují jako běžný provoz. Jelikož je dataset ve formě pcapů, rozhodli se autoři transformovat ho pomocí Nfstream do csv formátu.

K tomuto datasetu autoři ještě přidali odchytávání reálného provozu s IPsec serverem běžícím na AWS (Amazon web services), což je placená platforma cloud computingu od Amazonu.

Pro prostor příznaků použili mnoho atributů jako například histogram velikostí paketů v toku, počet syn, fin paketů, počet přenesených bytů a vyzkoušeny byli dva ML algoritmy, konkrétně Multilayer Perceptron (MLP) model a Random Forest Model (RFM).

Pro vyhodnocení autoři jednoduše rozdělili dataset na testovací a trénovací data v poměru 20:80. Oba dva modely mají vysokou úspěšnost a to 1.1.

Na tomto článku je vidět, že i jednoduché použití pouze strojového učení vede k slušnému výsledku

Model	Accuracy	True positive rate	True negative rate
MLP	93.83%	96.93%	90.64%
RFM	95.43%	98.62%	92.26%

Tabulka 1.1: Úspěšnost modelů.

1.5.2 Detekce VPN pomocí fingerprintingu

V článku [25] se autoři zaměřili na detekci OpenVPN pomocí fingerprintingu provozu. Inspirovali se architekturou „The Great Firewall of China“ a ve svoji architekturu rozdělili na dvě komponenty, Filter a Prober. Filter má za účel pasivní filtrování v reálném čase, kde označuje podezřelé flow záznamy. Prober naopak má naopak aktivně ověřit ty flow záznamy, které předtím označil Filter. Dělá to tak, že na označenou IP adresu posílá provoz, který se tváří jako provoz od OpenVPN klienta a čeká jak server zareaguje.

Tento článek se zabývá pouze OpenVPN a jejími komerčními variantami, které obvykle používají pouze nějaké obfuskace navíc. Mezi nejtypičtější obfuskace patří

1. XOR patch - paket je celý přexorován se sdíleným klíčem, nebo se obrátí pořadí bytů a každý se xoruje s jeho novou pozicí v paketu, nebo se použije kombinace dvou předchozích postupů

2. OpenVPN přes šifrovaný tunel - pro obranu proti DPI zabalují někteří do nějakého šifrovaného tunelovacího protokolu jako je Obfsproxy, Websocket tunnel, STunnel.

3. Komerční protokoly - Principy této rodiny obfuskačním protokolů nejsou většinou známy a poskytují tedy další vrstvu bezpečnosti. Příklady jsou VyprVPN nebo Astrill.

Pro fingerprinting se používají následující tři vlastnosti sekvence paketů. Sled po sobě jdoucích bytů, délka paketu, chování serveru. Následují konkrétní příklady jeho využití.

- Opcode-based fingerprinting - V OpenVPN protokolu je na fixní pozici nešifrovaný opcode a key-id zprávy, kde opcode je číslo v rozmezí 1-10, které označuje typ zprávy. Toho využívají i autoři článku a ti se snaží najít sekvenci těchto opcodů, která náleží typickému OpenVPN navázání spojení. Autoři zároveň nehledají přesnou shodu, ale hledají sekvenci podobných bytů. Toho pomůže proti některým obfuskačním technikám obzvláště XOR, protože ty často mapují stejné opcody na stejnou hodnotu.

- Ack-based fingerprinting - OpenVPN implementuje svůj vlastní potvrzovací protokol pro navázání spojení, ten je zde potřeba z důvodu, že

potřebujeme navázat spojení spolehlivě. Jakmile se spojení naváže už toto potvrzování není zapotřebí a protokol může pokračovat standardně nespolehlivě přes UDP. Velmi vhodná vlastnost tohoto potvrzování je, že tyto pakety nenesou žádný obsah a budou tedy stejně veliké. Stejné malé pakety viděné pouze v určitém místě našeho toku nám tedy umožní provoz fingerprintovat. Toto zafunguje hlavně na provoz, který není obfuskován přidáváním náhodného paddingu.

- Active-server fingerprinting - Zde autoři článku využívají vlastnosti toho, že pokud OpenVPN serveru pošleme reset paket, odpoví server klientovi také reset paketem a tím vydá svojí totožnost. Mnohé komerční implementace se tomuto brání tak, že přidají další vrstvu autentizace, kde musíme naše pakety podepsat HMAC algoritmem z nějakého sdíleného klíče, aby nám server vůbec odpověděl. To je problém, ale autoři zjistili, že na rozdíl od obfs4 (obfs4 používá náhodné zpoždění) zařízne spojení server okamžitě. Toto lze také využít.

Celkově je kombinace těchto metod velice účinná. Obecný recall uvádějí autoři jako 76.24% a false positive rate jako 0.0039%, což je velice zanedbatelné číslo.

1.5.3 Porovnání přístupů

Dva články popsané v této kapitole reprezentují typické přístupy k problému detekce VPN tunelů. Článek z [24] reprezentuje přístup strojového učení a článek z [25] naopak princip fingerprintingu s kombinací s DPI.

Kdybychom měli zdůraznit pro nás důležité rozdíly, tak že přístup strojového učení je obecnější oproti fingerprintingu, kde je potřeba navrhnout fingerprintování každého nového VPN protokolu zvlášť. To sice není v prvním článku demonstrováno úplně obecně, ale věřím, že by to nějaký rozumně velký ML model zvládl. Nevýhoda ML je velký požadavek na výpočetní výkon.

Naopak výhodou principu založeném na fingerprintingu je hlavně jeho menší časová a paměťová náročnost.

Kapitola 2

Analýza a návrh

V této kapitole si ukážeme nový přístup k detekci VPN tunelů. Tento přístup je zároveň navržen tak, aby byl použitelný na vysokorychlostních sítích a také protokol agnostický, neboli nebude využívat žádné konkrétní implementační záležitosti jednoho VPN protokolu, ale bude na všechny VPN spojení nahlížet obecně. Bude zde proveden jeho rozbor i s rozborem VPN protokolů s ním související. Nadále ho naimplementujeme v Pythonu a zkusíme cvičně vyhodnotit jeho výkonnostní efektivitu a celkovou úspěšnost na anonymizovaných datech ze sítě CESNET2.

2.1 Návrh algoritmu

Na začátku návrhu stála domněnka, že délka paketu jdoucího do VPN tunelu se zobrazí vždy na stejnou délku, tedy že se VPN protokol chová jako jednoduchá funkce. Zároveň i pakety podobné velikosti budou mít ve VPN tunelu podobnou velikost. Pokud tedy nalezneme nějaký častý protokol, který zároveň bude mít nějak typickou sekvenci délek paketů, zobrazená sekvence si zachová nějaké vlastnosti. Ideálním kandidátem pro toto se nám jeví TCP protokol s jeho handshakem.

Sekvence paketů SYN, SYN-ACK, ACK (dále zkracováno na SSA sekvenci) je velice charakteristická a zároveň je TCP handshake (1) dnes jeden z nejrozšířenějších jevů na síti. Lze předpokládat, že se i ve většině VPN spojení bude v tunelu v nějaký čas vyskytovat. Budeme tedy hledat obrazy SSA sekvence.

Pro připomenutí vlastností SSA sekvence se podívejme znovu na obrázek 1.5. My předpokládáme následující.

- Tři malé pakety podobné velikosti - Jak bylo vysvětleno v teoretické části, tyto pakety mají všechny 20-60 bytů v závislosti na poli TCP Options
- Alternující směry mezi serverem a klientem

- Krátké časové okno ve kterém jsou viděny

Dále se vyplatí ještě uvažovat tzv. rozšířenou SSA sekvenci, což je klasická SSA sekvence a za ní následuje opět v krátkém časovém okně a směru ACK paketu paket, který je větší než všechny z SSA. Toto je založené na myšlence, že jakmile se naváže spojení se serverem, začne klient okamžitě posílat data, kvůli kterým vůbec toto spojení navazoval. Ty budou pravděpodobně větší než ty z SSA sekvence. Toto rozšíření by nám mohlo výrazně snížit false positive rate v případech, že se na síti bude vyskytovat protokol, který posílá dokola malý paket tam a zpět.

Náš algoritmus je navržený pro využití na vysokorychlostních sítích s flow monitoringem. Nemůžeme si tedy dovolit DPI a budeme pracovat pouze s flow záznamy. Hlavní statistiky pro nás budou délka obsahu paketu a čas záchytu paketu.

Podotkněme, že kvůli tomu, že hledáme SSA sekvenci, která se vyskytuje na čtvrté vrstvě OSI modelu, je možný použít náš detekční algoritmus pouze pro ty VPN služby, které balí i tuto vrstvu. Z tohoto důvodu byl z této práce vynechán SSH protokol, protože ten čtvrtou vrstvu s TCP neobaluje.

2.2 Datasety

K ladění a ověřování našich hypotéz byly zachyceny následující datasety.

- `ipsec.csv` - IPsec protokol s 64M pakety (45.6GB)
- `wireguard.csv` - Wireguard protokol s 51.3M pakety (45.8GB)
- `openvpn.csv` - OpenVPN protokol s 67.9M pakety (45.6GB)
- `novpn_tcp{1-3}.csv` - běžný TCP provoz, tři datasety o souhrnné velikosti 87.5M paketů (76.5GB)
- `novpn_udp{1-3}.csv` - běžný UDP provoz, tři datasety o souhrnné velikosti 28.5M paketů (16.6GB)

Ty byly všechny zachyceny na síti CESNET2 pomocí filtrování portů vedoucím práce a následně anonymizovány. Toho se dalo využít, protože většina VPN serverů běží na známých portech. Tyto datasety máme ve formě csv, kde uvnitř jsou klasické sloupce z analýzy Wiresharku. Jeden řádek má následující formu.

```
Number,Time,Source,Destination,SrcPort,DstPort,
Protocol,PayloadLength,FrameLength,Info
"120","1.660492","162.142.125.132","147.32.112.223",
"26845","161","SNMP","51","89","get-request 1.3.6.1.2.1.1.5.0"
```

Number je číslo paketu v datasetu, Time je čas v sekundách od počátku zachytávání, Source je anonymizovaná zdrojová IP adresa, Destination je anonymizovaná cílová IP adresa, SrcPort je zdrojový port, DstPort je cílový port, Protocol je klasická protokolová detekce Wiresharku, PayloadLength je velikost obsahu paketu, FrameLength je velikost celého paketu i s hlavičkami, a info je opět klasické pole, co nám doplní Wireshark.

Běžný provoz byl chytán odděleně na různých podsítích sítě CESNET2 a abychom názorně viděli, která data nám dělají problémy, je i rozdělen na UDP a TCP data.

Mimo tyto datasety hlavně v počáteční fázi výzkumu byli použity data z nachytaná z lokální sítě. K tomu slouží skript `capture.sh`, kterému se na vstup dá jméno VPN rozhraní a on začne souběžně chytat běžné rozhraní a rozhraní VPN. Poté se pomocí sjednoceného času můžeme dívat na paket jdoucí do VPN tunelu a jeho obraz už v něm.

2.3 Analýza chování VPN

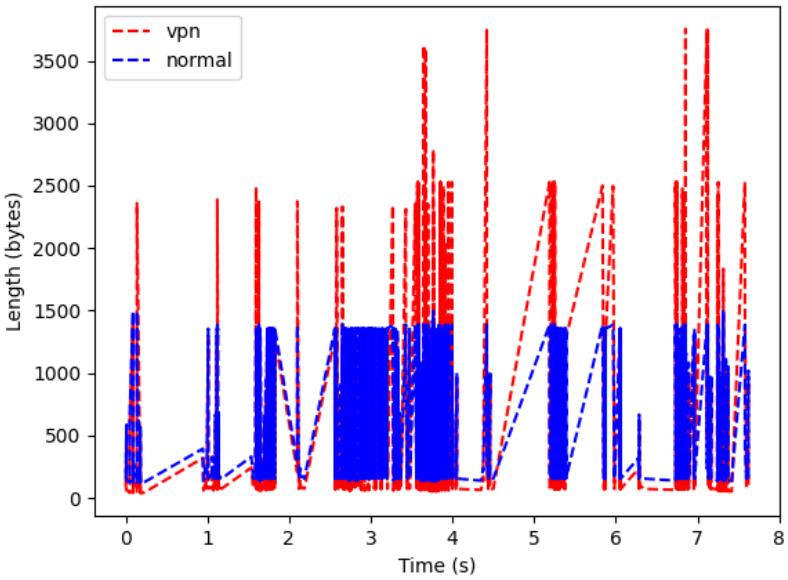
V první řadě je potřeba zjistit jak se chová VPN k paketům, které jdou do VPN tunelu. Je potřeba zjistit, jestli platí, že se opravdu VPN protokoly chovají jako funkce.

K tomuto účelu jsme použili Wireguard, IPsec a OpenVPN připojení na server poskytnutý společností Cesnet. Data byla nejdříve zachycena pomocí skriptu `track_data.py` a následně zobrazena pomocí `plot_from_csv.py`. Bylo i zkoumáno konkrétně, jak se zobrazí SSA sekvekce. Ve skriptu `generate.sh` byla SSA sekvence generováná linuxovou utilitou Netcat, která naváže TCP spojení s zadaným serverem.

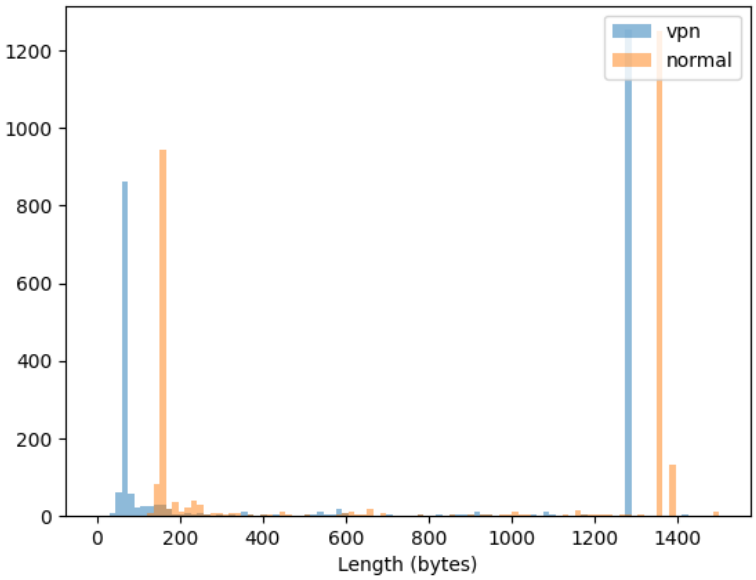
Podívejme se nejdříve na Wireguard protokol. Na obrázku 2.1a jsou vidět délky paketů než jsou obaleny Wireguard protokolem a po zabalení. Je tu vidět, že za prvé Wireguard zobrazuje pakety jedna ku jedné a za druhé Wireguard nepoužívá žádný náhodný padding. Je to vidět i na obrázku 2.1b, kde je většina velikostí paketů jen posunutá o nějaký konstantní počet bytů. Neplatí to samozřejmě ve všech případech. Například některé velké pakety jsou Wireguardem rozloženy do více menších. Poté určitě Wireguard má nějaké své řídící pakety. To nám ale nevadí. Pro nás je hlavní, že paket stejné délky se vždy zobrazí na větší paket a vždy na stejnou délku.

Na obrázku 2.2 je ještě potvrzení, že tyto vlastnosti platí pro SSA sekvenci. SSA sekvence je zde vyznačena a vždy se zobrazí na stejnou velikost. Je také zajímavé, že ačkoli je v tomto případě nezabalený SYN větší než nezabalený ACK zobrazí se na stejnou velikost. Je zde v platnosti tedy nějaký padding. Protože je Wireguard opensource dalo se i zjistit, že konkrétně se používá padding na 16 bytů.

Nyní se podívejme na OpenVPN. Jak obrázek 2.3a, tak i obrázek 2.3b naznačují, že všechny vlastnosti popsané v předchozím odstavci platí i pro

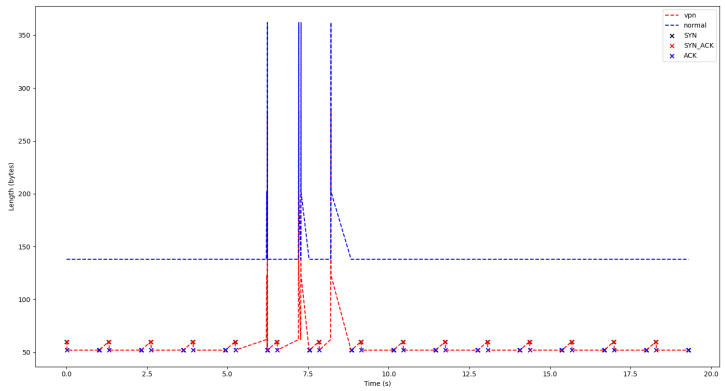


(a) Zobrazení délek paketů



(b) Histogram délek paketů

Obrázek 2.1: Zobrazení Wireguardu



Obrázek 2.2: Histogram délek generovaných paketů

OpenVPN. Jediný rozdíl je vidět na obrázku 2.4, kde zjišťujeme, že OpenVPN nepoužívá padding jako Wireguard.

U OpenVPN jsme vyzkoušeli i zapnout kompresi, konkrétně `lz4-v2`, ale u SSA sekvence jsme žádné rozdíly nezpozorovali. Wireguard možnost komprese nepodporuje. U VPN služeb se komprese nedoporučuje a my tedy můžeme předpokládat, že ani většina běžících VPN služeb jí nepoužije. Není doporučena z důvodu, že otevírá možnost útoku na VPN službu, útok se jmenuje VORACLE a detailněji je rozebrán v [26].

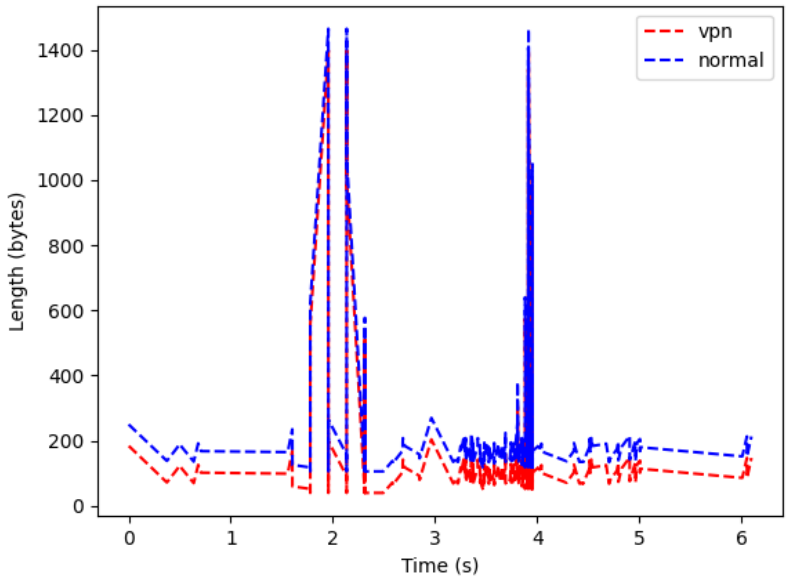
Podotkněme, že IPsec nebyl ověřen z toho důvodu, že ačkoli nám byl poskytnut přístup přes IPsec k síti CESNET2, nepodařilo se nám ho zprovoznit. To ale nevadí, protože víme že u Wireguardu a OpenVPN analýza vypadá nadějně. IPsec bude stejně testován v následujících sekcích.

Podle této analýzy to vypadá, že SSA sekvence půjde snadno rozeznat i zabalené v klasických VPN službách. Mapování jedna na jednu a to, že relativní velikost vůči ostatním paketům zůstává zachována, tomu dost napovídají.

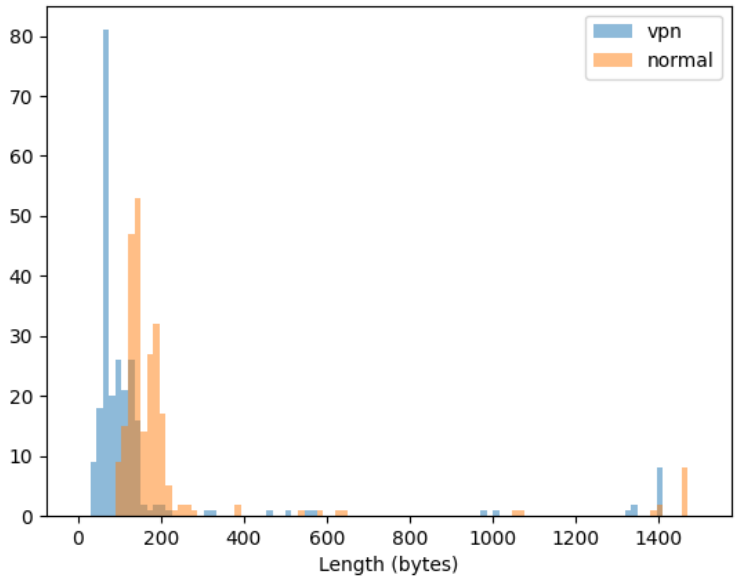
2.4 Implementace v Pythonu

V této sekci se budeme věnovat tomu, jak naimplementovat náš algoritmus v jazyce Python pro testovací a ladící účely. Nejdříve navrhneme základní automat, který bude hledat podezřelé SSA sekvence a pak ho budeme inkrementálně vylepšovat na datech popsaných v sekci 2.2. V této sekci si ale nebudeme ukazovat přímou implementaci, ale pouze principy na jakých funguje. Tato implementace totiž slouží pouze jako prototyp k odladění parametrů algoritmu a reálná implementace v jazyce C++ nás čeká v další kapitole. Implementace z této kapitoly je k nalezení na přiloženém médiu.

Navrhněme nejdříve tedy testovací implementaci. Tu uděláme v Pythonu,

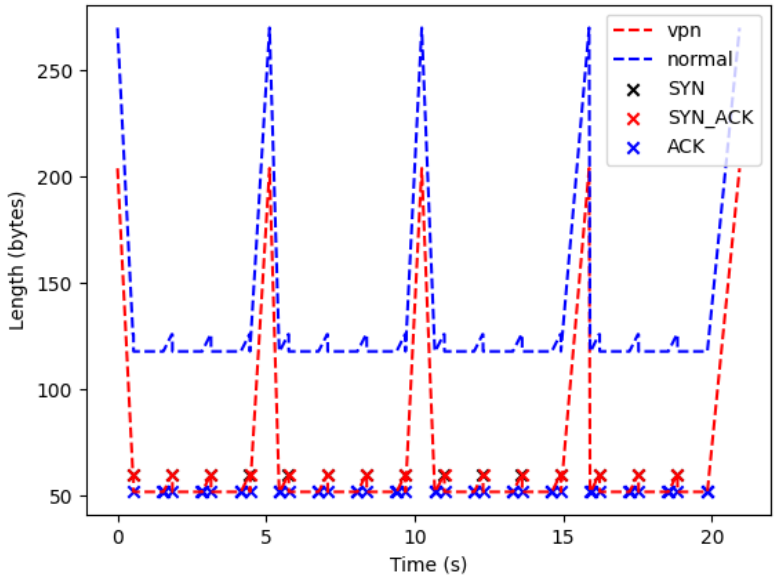


(a) Zobrazení délek paketů



(b) Histogram délek paketů

Obrázek 2.3: Zobrazení OpenVPN



Obrázek 2.4: Histogram délek generovaných paketů

protože nám zatím nejde o rychlost algoritmu a Python má tu výhodu, že pro něj existuje mnoho vizualizačních a statistických nástrojů a zároveň je rychlý na implementaci prvotního řešení. Toto řešení lze nalézt v souboru `detection.py`.

Naše testovací řešení bude obsahovat nedeterministický automat s dále popsanými stavy a přechody mezi nimi. Na obrázku 2.5 vidíme jeho stavy a přechody. Tento obrázek ale sám o sobě nestačí, protože stavy ani přechody zde nejsou vysvětlené. Nejsou zde vysvětlené z toho důvodu, že jejich parametrizací se teprve budeme zabývat v dalších sekcích.

Je třeba také vysvětlit, že stavy na obrázku 2.5 nejsou ve skutečnosti stavy, ale třída stavů. Protože totiž neznáme exaktní délku SYN paketu, do stavu Syn můžeme přijít s pakety různé velikosti a z dvou směrů, je tedy nutné si toto ve stavu pamatovat. Stav bude tedy například uspořádaná trojice (třída, směr, délku paketu). Toto se ještě bude nadále v této sekci měnit s tím, jaké všechny optimalizace použijeme a bude potřeba si pamatovat i jiné atributy.

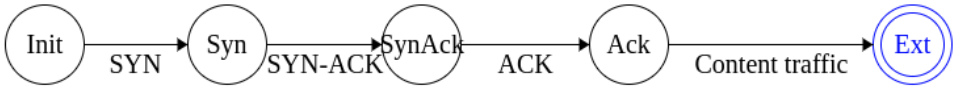
Následuje podrobnější popis stavů v jejich základní verzi.

1. Init - Toto je pouze počáteční stav, žádné atributy nepotřebujeme.

2. Syn - Zde je potřeba znát velikost SYN paketu, směr paketu (ve smyslu od serveru nebo od klienta), čas přechodu

3. SynAck - Stejné atributy jako u Syn stavu

Obrázek 2.5: Automat pro SSA sekvenci



Obrázek 2.6: Automat pro rozšířenou SSA sekvenci

4. Ack - Toto je koncový stav, tedy žádné atributy nepotřebujeme.

Ještě je možné uvažovat automat pro rozšířenou SSA sekvenci který bude mít navíc další stav pojmenovaný jako Ext a ten bude reprezentovat první poslaný paket po navázání spojení od klienta na server. Tento automat je znázorněn na obrázku 2.6. V případě tohoto rozšířeného automatu zároveň musíme pozměnit stav Ack. Ten nadále nebude koncový (protože roli koncového stavu za něj přejímá Ext) a také je potřeba u něj evidovat směr posledního paketu.

Jak dále zavést přechody? Následuje shrnutí, které je stále nepřesné. Specifické parametry budou odladěny dále.

1. Přechod SYN - Očekáváme pouze malý paket.

2. Přechod SYN-ACK - Očekáváme malý paket, který je opačného směru než předchozí SYN. Zároveň jeho velikost musí být teoreticky schopná následovat velikost předchozího SYN paketu v SSA sekvenci.

3. Přechod ACK - Očekáváme malý paket, opačného směru než předchozí SYN-ACk a také správné velikosti.

4. Přechod Content traffic - Tento přechod se vyskytuje pouze u automatu pro rozšířenou SSA sekvenci. Zde očekáváme paket ve směru ACK paketu, který bude větší než všechny předchozí pakety z SSA sekvence.

Klíčovým místem v naší implementaci je třída `FlowAutomaton` a metoda `make_transition`, které právě náš nedeterministický automat implementuje. Toto už je třída, která pracuje s konkrétním flow záznamem. Podívejme se na kód z 2.1 (kód je zde zjednodušený oproti skutečné podobě).

Ze začátku proměnná `self.states` obsahuje pouze INIT stav. Pokaždé když přijde nový paket patřící do tohoto flow záznamu, zavolá se metoda `make_transition` a za parametry vezme t – čas záchytu paketu, l – délka paketu, direction – směr paketu (v naší implementace se směr bere relativně ke směru prvního zachyceného paketu náležícího do flow záznamu). Poté se projdou všechny aktuálně platné stavy a metoda `state.make_transition` se pokusí vygenerovat nový platný stav. Pokud není koncový pouze se nově vygenerovaný stav přidá do množiny všech platných stavů a pokud je tak se uloží informace o validní SSA sekvenci a zahodí se všechny stavy. Toto zahození je nutné z důvodu jak časové optimalizace, tak z důvodu, že kdybychom nevyhodili platné stavy, brzo by se mohl vygenerovat další konečný stav a tedy bychom jednu SSA sekvenci měli označenou vícekrát. Toto nastávalo často v testovacích datech a SSA sekvence se často lišily pouze délkou posledního paketu. Toto znemožnilo používat efektivně nějaké statistiky založené na rozptylu nebo počtu SSA sekvencí.

Listing 2.1: Main implementation

```
def make_transition(self, t, l, direction):
        new_states = set()
        for state in self.states:
            new_state =
            state.make_transition(t, l, direction)
        for new_state in new_states:
            if new_state.is_end_state():
                self.flow_context.suspects += 1
                self.states.clear(); break;
            else: self.states.add(new_state)
```

2.4.1 Ladění algoritmu

Náš algoritmus obsahuje mnoho parametrů, u kterých je potřeba rozhodnout, jak by měli vůbec vypadat, abychom byli co nejvíce efektivní z hlediska celkových výsledků. Tato implementace v Pythonu si neklade za cíl i být maximálně efektivní z hlediska časové náročnosti, to bude až záležitost realizace v C++ v další kapitole až při reálném nasazení do IPFIXprobe.

Začneme nejdříve tím, že si tyto hlavní parametry vyjmenujeme a zkusíme je rozumně odhadnout na startovací pozici ze které budeme vycházet. Hlavní parametry a jejich počáteční hodnoty jsou následující.

- Velikost malého paketu - `<60, 200>` bytů
- Maximální časový rozestup mezi dvěma pakety SSA sekvence - 0.4s
- Přijatelné velikostní rozdíly mezi pakety SSA sekvence - toto je popsané v subsekci 2.4.1.1

Poznamenejme, že v následujících úvahách můžeme tyto parametry uvažovat jako nezávislé a můžeme je ladit aniž by se nějak ovlivňovali.

Tyto parametry nebyly zvoleny náhodou, ale rozhodli jsme se pro ně po opakovaném spouštění prvotní implementace. Tento postup byl spíše experimentální a vypadal tak, že se zvolily parametry podle prvotního odhadu a poté jsme hodnoty upravily a porovnali FPR a TPR s předchozími výsledky. V tento moment se neměnily pouze parametry, ale i celkově implementace prototypu a proto tento postup není přesně zdokumentován.

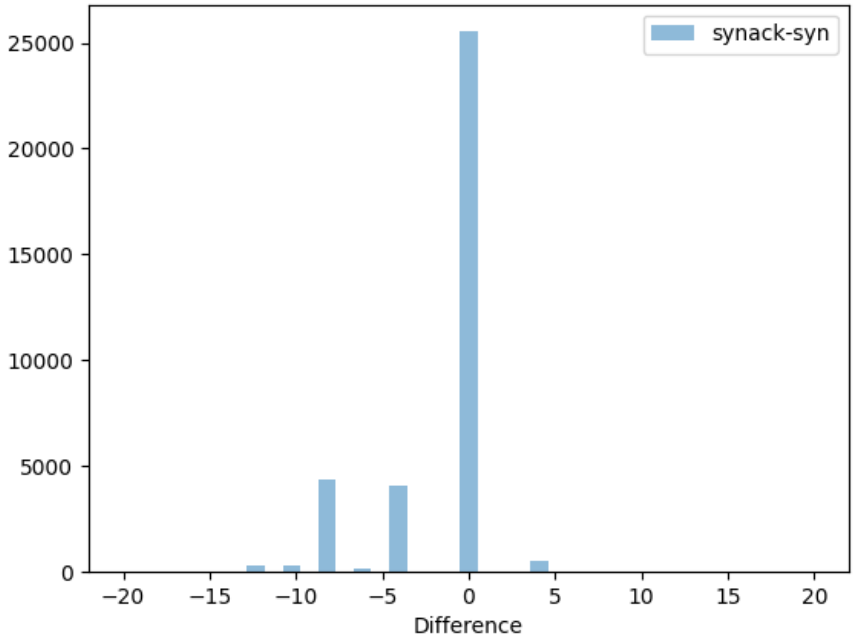
Všechny následující běhy popsané v následujících odstavcích byli z časových důvody spouštěné na všech datasetech, ale byli omezeny na šest milionů paketů z každého. Trénování probíhalo na zapůjčeném serveru od společnosti Cesnet.

Pro naše počáteční nastavení máme tuto úspěšnost.

```
Rate ipsec.csv: 487/656 (0.7423780487804879)
Rate wireguard.csv: 84/141 (0.5957446808510638)
Rate openvpn_udp.csv: 523/630 (0.8301587301587302)
Rate openvpn_tcp.csv: 167/341 (0.4897360703812317)
Rate novpn_tcp1.csv: 26/2759 (0.009423704240666908)
Rate novpn_udp1.csv: 120/211 (0.5687203791469194)
Rate novpn_tcp2.csv: 10/1676 (0.0059665871121718375)
Rate novpn_udp2.csv: 293/980 (0.29897959183673467)
Rate novpn_tcp3.csv: 11/237 (0.046413502109704644)
Rate novpn_udp3.csv: 282/904 (0.31194690265486724)
========================================
Rate true_positive: 1261/1768 (0.7132352941176471)
Rate false_positive: 742/6767 (0.10964977094724397)
Accuracy: 7286/8535 (0.8536613942589338)
```

Tyto výsledky jsou docela slibné, ale vypadá to, že u UDP dat má náš algoritmus velké problémy s FPR. Po nahlédnutí do dat jsme zjistili, že toto vysoké FPR je způsobené tím, že v těchto datech se nachází často sekvence malých paketů tam a zpět pořád dokola. Po konzultaci s vedoucím práce jsme zjistili, že se jedná hlavně o komunikaci s herními servery.

Ideální jako výstup z této kapitoly by byli dvě verze našeho algoritmu. První verze by byla optimalizována tak, aby ji bylo možné nasadit samotnou. To znamené, že u ní bude kladen důraz i na malé FPR. Druhá verze by měla hlavně maximalizovat TPR a při jejím nasazení se využijí nějaké přídavné mechanizmy, které dodatečně budou kontrolovat falešné pozitiva. Na tuto myšlenku bude ještě navázáno v závěru práce, kde se zmíníme teoreticky právě o těchto přídavných mechanizmech.



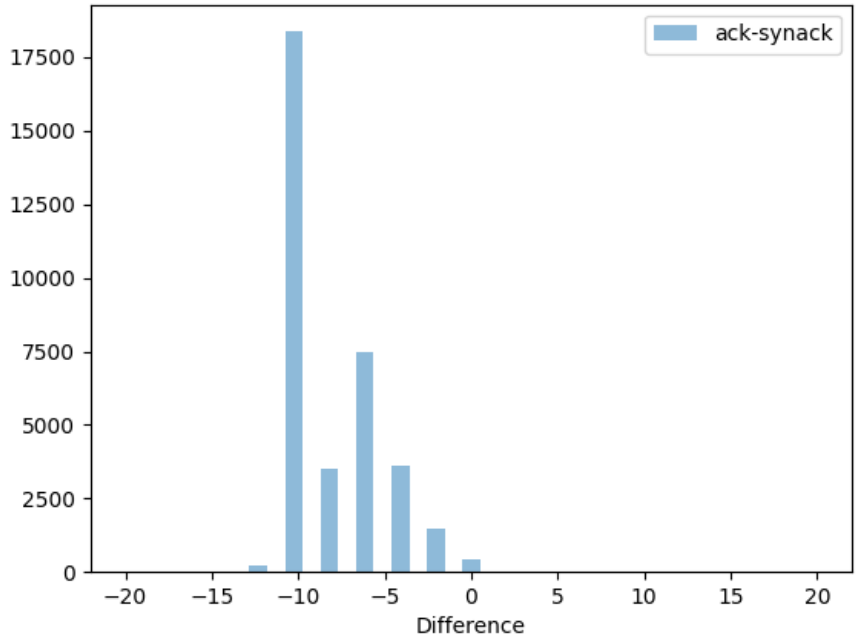
Obrázek 2.7: Histogram rozdílu mezi SYN-ACK a SYN pakety

2.4.1.1 Analýza rozdílů

Ze všeho nejdříve je potřeba navrhnout jaké velikostní rozdíly mezi sebou mohou mít jednotlivé pakety SSA sekvence. K tomu použijeme datasety `novpn_tcp{1-3}.csv`, které obsahují mnoho nešifrovaných SSA sekvencí. Toto je implementováno v souboru `analyze_ssa.py` pomocí jednoduchého automatu, který hledá v krátkém časovém okně (0.3 s) SSA sekvenci. Z těchto datasetů lze získat informaci o řídících paketech TCP.

Na obrázku 2.7, který znázorňuje histogram rozdílů mezi SYN-ACK a SYN pakety v SSA sekvenci, je krásně vidět, že v naprosté většině případů je rozdíl těchto dvou paketů v rozmezí $(-10, 0>$, což znamená, že SYN paket je větší nebo roven SYN-ACK. Většina těchto rozdílů je nula. Z tohoto intervalu se dostane jen v zanedbatelných číslech. V naší testovací implementaci použijeme tedy tento interval.

Na obrázku 2.8, který znázorňuje histogram rozdílů mezi ACK a SYN-ACK pakety v SSA sekvenci, je vidět podobná tendence jako mezi SYN-ACK A SYN pakety. Opět je zde podobný interval $< -12, 0 >$, ale tentokrát jsou hodnoty více rozloženy. Naopak zde není nejčastější hodnota nula, ale -10, což znamená, že v naprosté většině případů bude SYN-ACK větší než ACK. Tento interval opět využijeme v testovací implementaci.



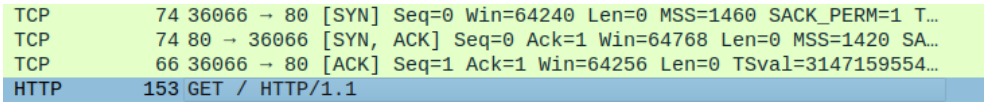
Obrázek 2.8: Histogram rozdílu mezi ACK a SYN-ACK pakety

2.4.1.2 Analýza času mezi pakety v SSA sekvenci

Maximální časové okno mezi pakety v SSA sekvenci je v kódu označeno jako parametr `time_window`. Tento parametr byl zvolen pomocí typického přístupu ze strojového učení známým jako grid search [27]. Byl prohledán prostor s parametry `0.05, 0.1, 0.3, 0.4, 0.5, 0.7, 1` sekund. Výsledky jsou k nalezení ve složce `testing/time_window`. My zvolili parametr 0.3, protože má nejslibnější výsledky.

2.4.1.3 Analýza malého paketu

Další parametr, který bylo potřeba zjistil jsou možné velikosti paketů v SSA sekvenci, to znamená horní a spodní mez velikosti. Zde byl opět použit přístup hledání optimálního parametru pomocí grid search. Výsledky jsou k nalezení ve složce `testing/paket_size`. My zvolili parametr s spodní hranicí za 60 bytů a s horní hranicí za 150 bytů. Sice to vypadá, že kdybych posunuli spodní hranici na 80 bytů, snížíme tím FPR, ale bohužel si toto dovolit nemůžeme, protože v OpenVPN datasetech je nejčastější malá velikost 74 bytů a lze očekávat, že v této velikosti budou schované i některé pakety SSA sekvence.



Obrázek 2.9: Pakety SSA sekvence rozšířené o Ext stav

2.4.1.4 Analýza ext stavu

Myšlenka rozšířit náš automat o jeden stav je postavená na pozorování z obrázku 2.9. Zde je vidět typický příklad toho, že po úspěšném TCP handshaku, tedy zachycení celé SSA sekvence, pošle klient serveru ihned větší paket. Vyzkoušíme tedy rozšířit náš automat o Ext stav 2.6 s očekáváním, že snížíme falešné pozitiva na UDP datech.

Pro tento přechod do ext stavu budeme očekávat následující tři podmínky. Za prvé očekáváme větší paket (parametr ext size) než je ACK paket, za druhé ve velice krátkém časovém okně (parametr ext time) a za třetí paket musí přijít ve směru od klienta. Nejdříve si tedy uděláme průzkum na našich TCP datech jak ve standardní komunikaci vypadá tento teoretický přechod.

Na obrázku 2.11 jsou zachyceny maximální velikosti z tří zachycených paketů následujících SSA sekvenci. Jelikož jsou data tak rozprostřená nemůžeme parametr ext size určit pouze z těchto statistik.

Na obrázku 2.10 jsou naopak zachyceny časy záchytu prvního paketu ve směru ACK paketu po SSA sekvenci. Tento obrázek už nám docela napoví a parametr ext time bude nastaven určitě do 0.2s.

Zkusíme tedy prohledat prostor těchto dvou parametrů, kde uvažujeme ext size $30 - 210$ a ext time $0.1 - 0.2$. Toto prohledávání naimplementujeme standardními for cykly a tedy tento prostor bude rozřezán rovnoměrně.

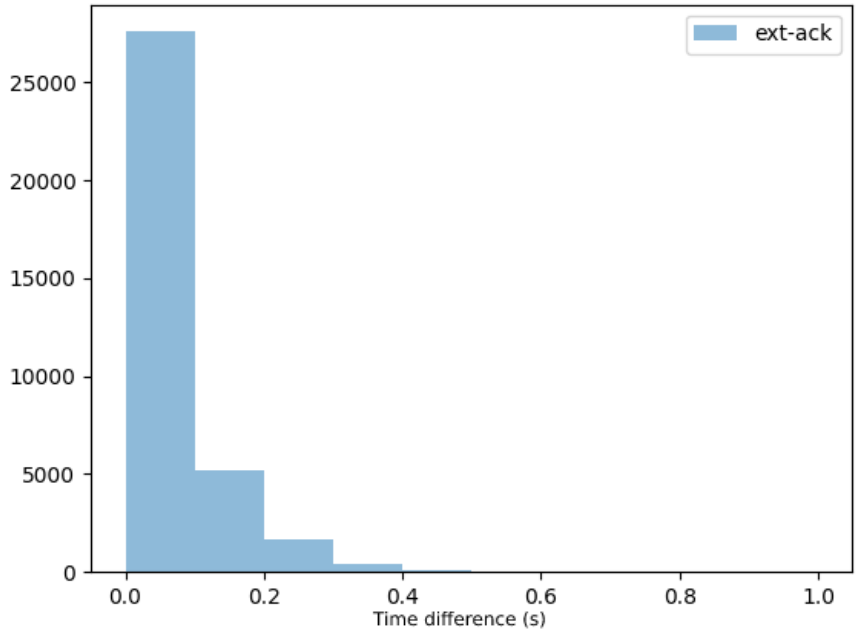
Výsledky hledání v tomto prostoru dvou parametrů je možno nalézt v souboru `testing/ext_state`. Bohužel je z výsledků vidět, že nám s redukcí FPR moc nepomohou a navíc by tento stav zkomplikoval náš algoritmus. Proto od tohoto rozšíření raději upustíme.

2.4.1.5 Analýza času flow záznamu a objemu přenesených dat

Celková doba trvání flow záznamu (rozdíl času prvního a posledního paketu náležících do flow záznamu) a objem přenesených dat jsou sice základní charakteristiky, ale mohli by nám pomoci. Pro tuto analýzu jsme použili všechny datasety obsahující pouze VPN provoz. Tyto datasety jsme použili celé. Podívejme se nejdříve na histogram délek jednotlivých flow v sekundách 2.12.

Z tohoto histogramu nemůžeme dedukovat bohužel žádné pravidlo, protože délky jsou moc rovnoměrně rozprostřené a jsou zde zástupci jak krátkých flow záznamů, tak i dlouhých.

Histogram přenesených bytů vypadá podobně (byty a čas budou s největší pravděpodobností korelovat), takže ani toto nám bohužel nepomůže.



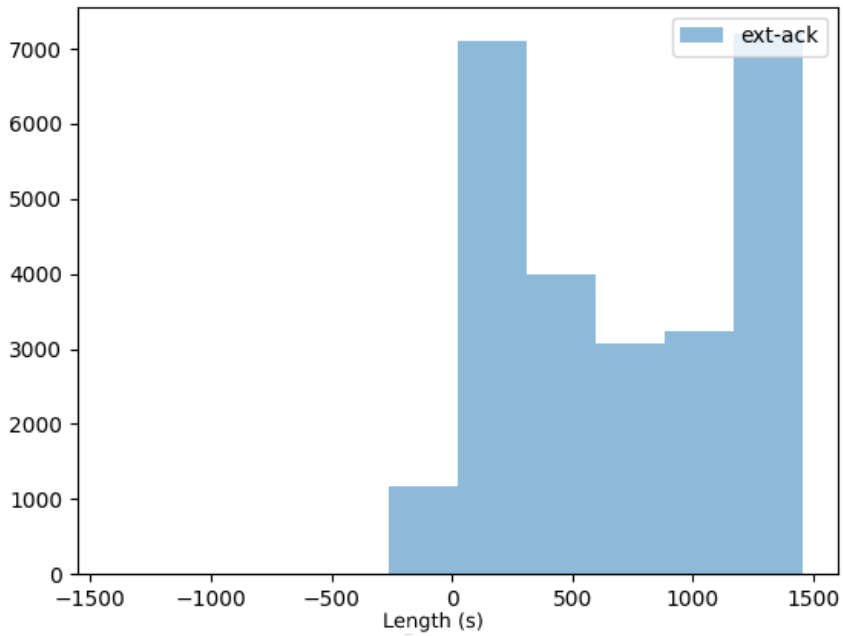
Obrázek 2.10: Histogram časových rozdílů mezi ACK stavem a Ext stavem

2.4.1.6 Analýza historie rozdílu na flow záznamu

Další nápad, co jsme dostali po nahlédnutí do falešně pozitivních dat, bylo, že bychom mohli nějak kvantifikovat rozptyl suspectů. Suspect je podle naší definice trojice délek paketů, kterou náš algoritmus označil jako potenciální pakety SSA sekvence. Teorie je taková, že pokud VPN spojení nemění za běhu své parametry tak například komunikace s jedním serverem vygeneruje vždy stejné suspecty. Tedy v ideálním případě budou všechny suspecty jednoho flow stejné trojice. Když se právě podíváme do suspectů falešných pozitiv, tak většinou je jen málo stejných trojic. Zkusme to tedy nějak kvantifikovat.

My zkusili pár jednoduchých metrik. Nejdříve jsme začali jednoduchou směrodatnou odchylku, ale ta , jak se ukázalo, špatně pracuje s principem unikátnosti tripletů, protože například falešné suspecty s délkami SYN paketů 80, 81, 82, bude mít také malý rozptyl, ale unikátnost vůbec nezohleďnuje.

Rozhodli jsme se tedy použít jednoduchý poměr unikátních suspectů k celkovému počtu suspectů. Tyto poměry se spočítají tři (přes SYN, SYN-ACK, ACK pakety) a z nich se spočítá aritmetický průměr. Dále bylo potřeba tuto charakteristu nějak ještě vztáhnout k počtu suspectů. To jsme provedli tím, že jsme si rozdělily flows na malé, středně velké, velké a velmi velké. Malé mají 3 až 14 suspectů, středně velké mají 15 až 39 suspectů, velké mají 40 až 99 suspectů a velmi velké mají 100 a více suspectů. Následně byla přes



Obrázek 2.11: Histogram velikostí potenciálních paketů pro přechod do Ext stavu

všechny tyto třídy spočítána naše charakteristika.

Grafy rozložení poměru unikátních suspectů k celkovému počtu suspectů jsou následující. Na obrázku 2.13 je vidět rozložení pro malé, na obrázku 2.14 pro středně velké, na obrázku 2.15 pro velké a na obrázku 2.16 pro velmi velké.

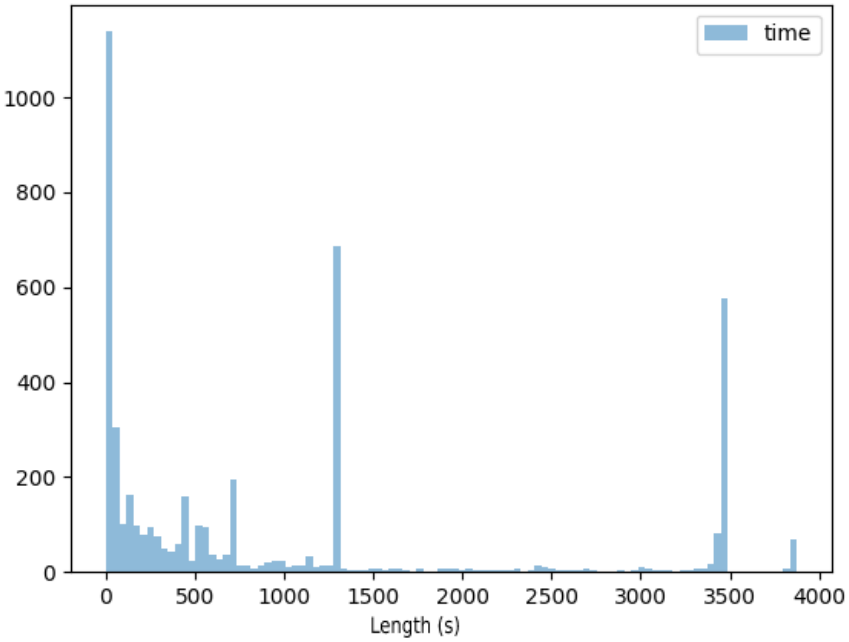
Z těchto výsledků jsem se rozhodl pro horní mez naší charakteristiky jako 0.6 pro malé, 0.4 pro střední, 0.2 pro velké, 0.1 pro velmi velké.

2.4.1.7 Analýza poměru počtu suspectů k počtu paketů

Další věc co by nám mohla pomoci je poměr počtu paketů k počtu suspectů v flow záznamu. Lze očekávat že u falešných pozitiv bude menší počet paketů potřebných na jeden viděný suspect v porovnání s validním VPN provozem. Na obrázku 2.17 je vidět, že na validním VPN provozu je tento poměr do 2500 a na falešných případech z obrázku 2.18 je ten poměr velice často větší. Je tedy dobrý nápad tento poměr omezit ze shora právě na zmíněných 2500.

2.4.2 Vyhodnocení na trénovacích a testovacích datech

Výsledkem této kapitoly budou dvě verze našeho algoritmu. Jedna jednoduchá která se soustředí pouze na označování podezřelých flow záznamů (dále verze

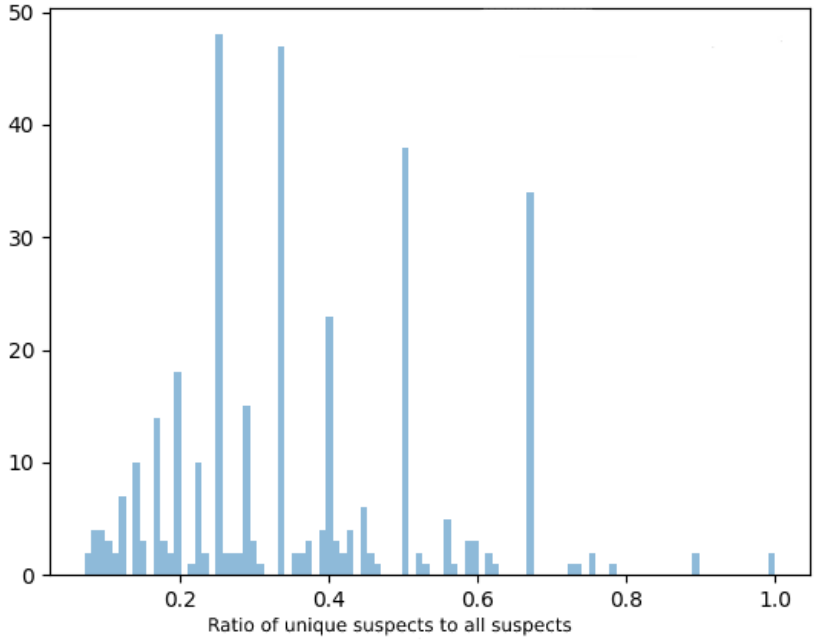


Obrázek 2.12: Histogram délek flow záznamů

1) a druhá která i pracuje s věrohodností těchto označených záznamů a tím se snaží zredukovat FPR (dále verze 2). Obě tyto verze vyhodnotíme zvlášť.

Pojďme se podívat na vyhodnocení úspěšnosti našeho algoritmu. Ze všeho nejdříve se podíváme na úspěšnost u trénovacích dat. Vzhledem k tomu, že jsme většinou těchto dat k trénování nepoužili z důvodu časové úspory neměli by být výsledky až tak zaujaté.

```
TRAINING MAX TRP
Rate ipsec.csv: 894/1308 (0.6834862385321101)
Rate wireguard.csv: 147/215 (0.6837209302325581)
Rate openvpn_udp.csv: 685/1030 (0.6650485436893204)
Rate openvpn_tcp.csv: 287/2226 (0.1289308176100629)
Rate novpn_tcp1.csv: 25/3270 (0.00764525993883792)
Rate novpn_udp1.csv: 33/211 (0.15639810426540285)
Rate novpn_tcp2.csv: 74/7560 (0.009788359788359789)
Rate novpn_udp2.csv: 187/980 (0.19081632653061226)
Rate novpn_tcp3.csv: 58/3193 (0.01816473535859693)
Rate novpn_udp3.csv: 1371/3019 (0.45412388208015897)
========================================
Rate true_positive: 2013/4779 (0.421217827997489)
Rate false_positive: 1748/18233 (0.095870125596446)
```

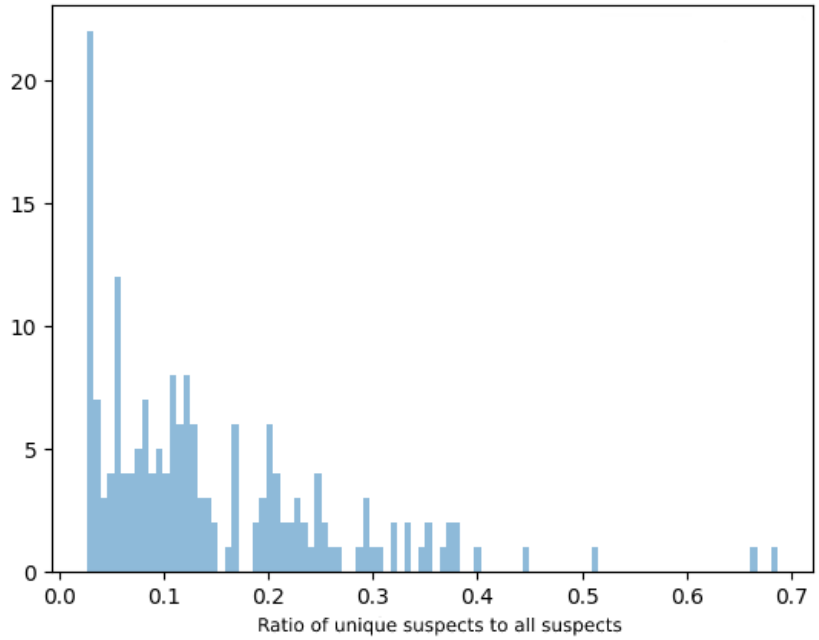


Obrázek 2.13: Histogram rozložení poměru unikátních suspektů pro malé flow záznamy

```
Accuracy: 18498/23012 (0.8038414740135581)

TRAINING MIN FPR
Rate ipsec.csv: 723/1308 (0.5527522935779816)
Rate wireguard.csv: 129/215 (0.6)
Rate openvpn_udp.csv: 632/1030 (0.6135922330097088)
Rate openvpn_tcp.csv: 238/2226 (0.1069182389937107)
Rate novpn_tcp1.csv: 2/3270 (0.0006116207951070336)
Rate novpn_udp1.csv: 2/211 (0.009478672985781991)
Rate novpn_tcp2.csv: 8/7560 (0.0010582010582010583)
Rate novpn_udp2.csv: 31/980 (0.03163265306122449)
Rate novpn_tcp3.csv: 2/3193 (0.0006263701847792045)
Rate novpn_udp3.csv: 353/3019 (0.11692613448161643)
========================================
Rate true_positive: 1722/4779 (0.3603264281230383)
Rate false_positive: 398/18233 (0.02182855262436242)
Accuracy: 19557/23012 (0.8498609421171562)
```

Z těchto výsledků je vidět, že verze 1 má na většině VPN provozu úspěšnost kolem 0.66 s výjimkou OpenVPN přes TCP. Toto je zvláštní a může to zname-



Obrázek 2.14: Histogram rozložení poměru unikátních suspektů pro středně velké flow záznamy

nat i například skutečnost, že v datasetu se vyskytuje velké množství provozu, který není VPN provoz. Skutečně po té, co jsme z datasetu vyfiltrovali pouze provoz, který wireshark neoznačil jako OpenVPN stoupne úspěšnost na 79%.

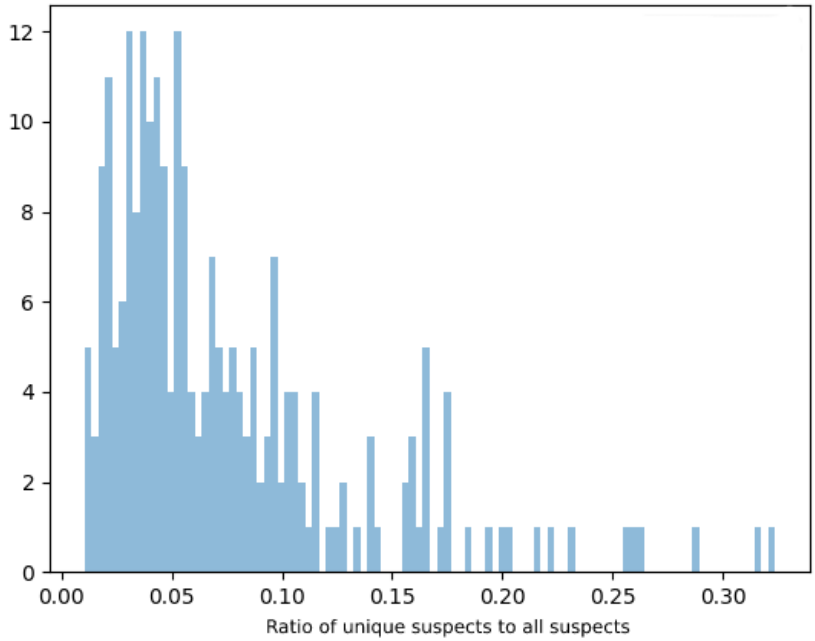
```
Rate openvpn_filtered.csv: 280/354 (0.7909604519774012)
```

Je tedy vidět že v datasetu `Rate openvpn_tcp.csv` je hodně flow záznamů, které vlastně VPN provoz neobsahují. Verze 1 má bohužel problém s UDP daty, protože ty často falešně označí za pozitivní. Zde je to krásně vidět na datasetu `novpn_udp3.csv` s 0.45 FPR, což je opravdu hodně.

Pokud se podíváme na algoritmus verze 2, tak je očividné, že bohužel sníží trp. Když vynecháme z úvahy OpenVPN data tak přibližně o 5 procent. Na druhou stranu zas rapidně sníží FPR u UDP dat, což je opravdu potřeba. Když se podíváme na `novpn_udp3.csv` tak FPR je zde 0.11, což je oproti 0.45 opravdu rozdíl.

Podívejme se ale na testovací data a to pouze na datasety s validním VPN provozem, protože z datasetů pro běžný provoz byl použit v této sekci pouze dataset `novpn_udp3.csv`, který už byl diskutován společně s trénovacími daty.

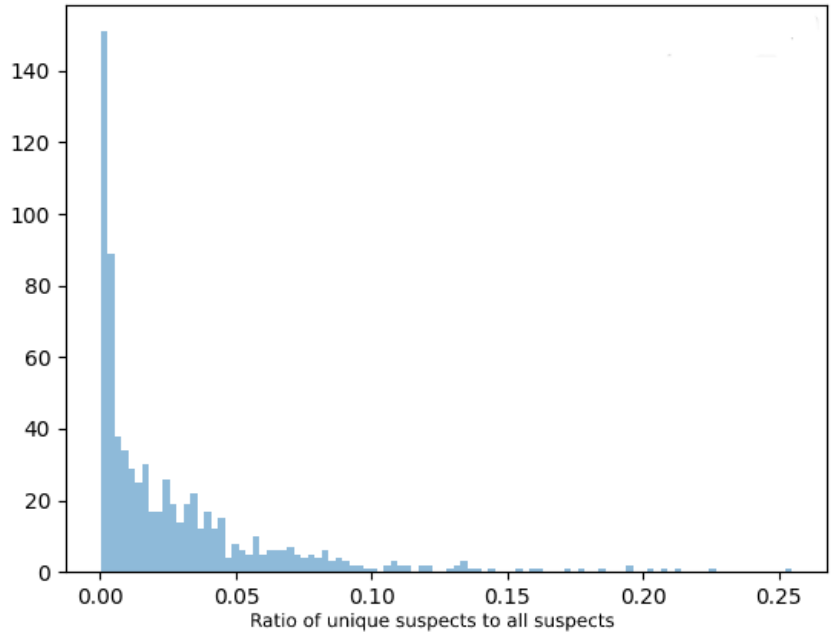
```
TEST MAX TPR
```



Obrázek 2.15: Histogram rozložení poměru unikátních suspektů pro velké flow záznamy

```
Rate ipsec-test.csv.udp: 1796/2050 (0.8760975609756098)
Rate openvpntcp-test.csv.tcp: 230/896 (0.25669642857142855)
Rate openvpn-test.csv.udp: 732/3433 (0.21322458491115642)
Rate wireguard-test.csv.udp: 351/421 (0.833729216152019)
Rate novpn_tcp3.csv: 58/3193 (0.01816473535859693)
Rate novpn_udp3.csv: 357/3490 (0.10229226361031518)
========================================
Rate true_positive: 3109/6800 (0.4572058823529412)
Rate false_positive: 415/6683 (0.06209786024240611)
Accuracy: 9377/13483 (0.695468367573982)
Time: 7025.787120819092

TEST MIN FPR
Rate ipsec-test.csv.udp: 1284/2050 (0.6263414634146341)
Rate openvpntcp-test.csv.tcp: 203/896 (0.2265625)
Rate openvpn-test.csv.udp: 643/3433 (0.1872997378386251)
Rate wireguard-test.csv.udp: 316/421 (0.7505938242280285)
Rate novpn_tcp3.csv: 2/3193 (0.0006263701847792045)
Rate novpn_udp3.csv.udp: 235/3490 (0.0673352435530086)
========================================
```



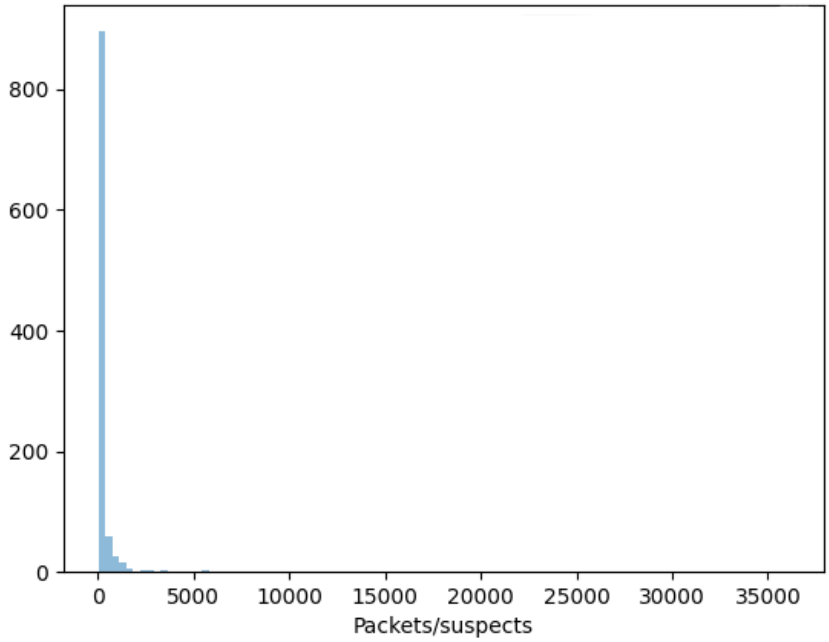
Obrázek 2.16: Histogram rozložení poměru unikátních suspektů pro velmi velké flow záznamy

```
Rate true_positive: 2446/6800 (0.35970588235294115)
Rate false_positive: 237/6683 (0.035463115367349994)
Accuracy: 8892/13483 (0.6594971445523993)
```

Když se podíváme na ipsec a wireguard jsou výsledky opravdu slibné. Verze 1 má na těchto datech přes 80 procent u obou a verze 2 sice má 0.62 a 0.75, ale i to je opravdu dobré. Bohužel u OpenVPN máme TPR okolo 0.2. U OpenVPN přes TCP to opět bude způsobené tím, že mnoho flow záznamů nebude vůbec VPN provoz. Po profiltrování to vypadá takto.

```
Rate openvpntcp-test_filtered.csv.tcp:
224/382 (0.5863874345549738)
```

U OpenVPN přes UDP se nám bohužel nepodařilo zjistit, proč máme tak nízkou úspěšnost. Vzhledem k tomu, že u trénovacích dat ji máme vysokou, je možné, že tento záchyt z této podsítě je nějakým způsobem silně zaujat. Například tím, že tam ve skutečnosti neběží VPN nebo přes VPN komunikuje nějaký protokol, který nevyužívá SSA sekvenci.



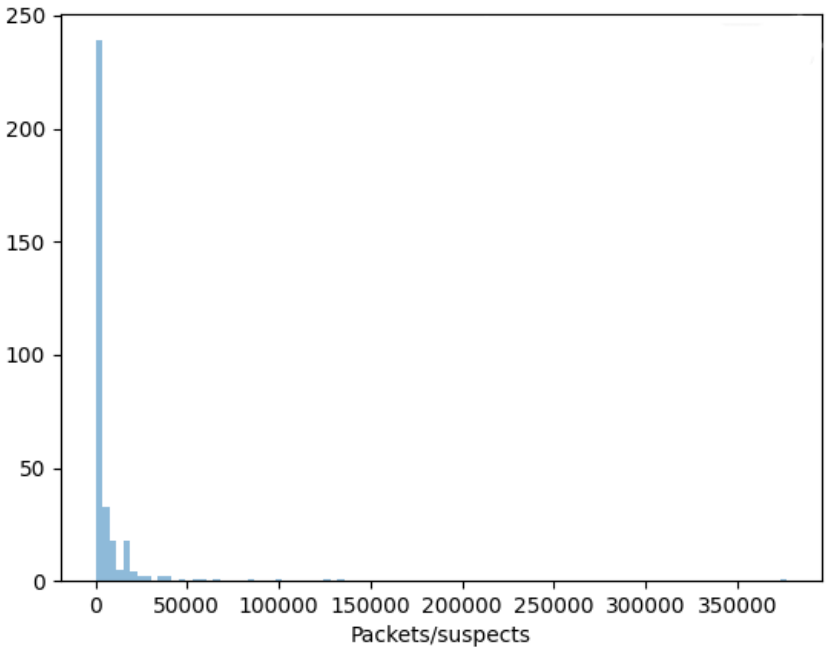
Obrázek 2.17: Histogram počtu paketů na počet suspectů u VPN záznamů

2.4.3 Shrnutí a možné navázání

Výsledky našeho algoritmu jsou tedy slibné, ale jako takový ho lze nasadit jen ve verzi 2, kde je redukováno FPR. To ale neznamená, že by verze 1 neměla použití. Pokud vezmeme výstup našeho algoritmu a zkombinujeme ho s dalšími indikátory jako je například princip založený na active probingu z článku [25], tak by se mohlo jednat o spolehlivou implementaci.

Naše algoritmy mají jednu skvělou vlastnost a to tu, že z gigabitových dat produkují pouze záznamy o podezřelých SSA sekvencích a tyto záznamy jsou v řádech kilobytů. Toto nám otevírá nové možnosti. Tím, že dokážeme objem informací takto zredukovat, dokázali bychom si představit ML řešení fungující nad těmito daty. Problém by byl samozřejmě nasbírat dostatečně velký dataset, abychom vůbec byli schopni ML algoritmus natrénovat, ale pokud by se toto podařilo, tak by tento ML přístup nemusel být výpočetně náročný, protože by jako vstup bral pouze pár záznamů. V této práci jsou tyto záznamy použity jen s jednoduchými algoritmy a ML přístup by určitě našel mnoho statistik, které my nevidíme.

Velkou nevýhodou našeho algoritmu je jeho malá robustnost detekce. Pokud by se nějaká implementace VPN protokolu rozhodla, že nechce být naším algoritmem detekována, stačí zvolit náhodné zarovnání paketů a to náš algoritmus dokonale zmate. Další možností by bylo možné i zarovnat všechny



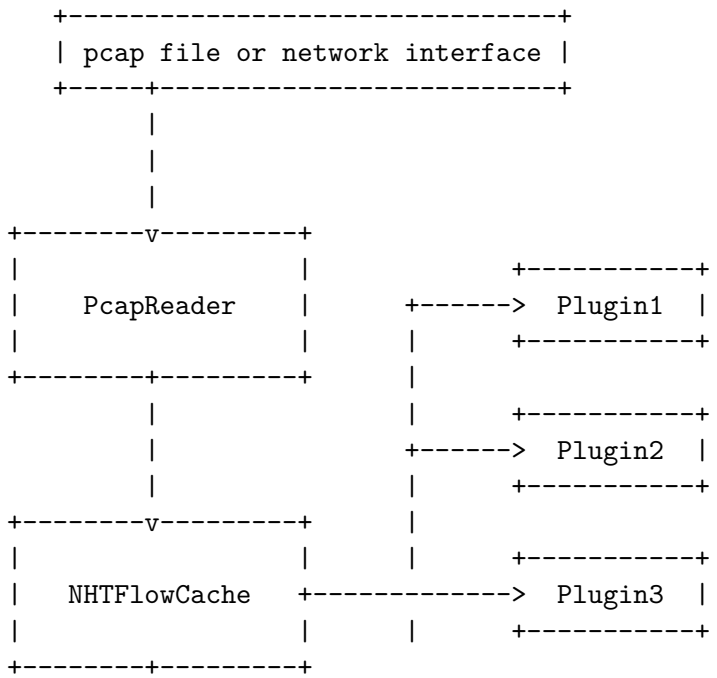
Obrázek 2.18: Histogram počtu paketů na počet suspectů u falešných VPN záznamů

pakety na velikost větší než je 160 bytů a ty my vůbec nedetekujeme.

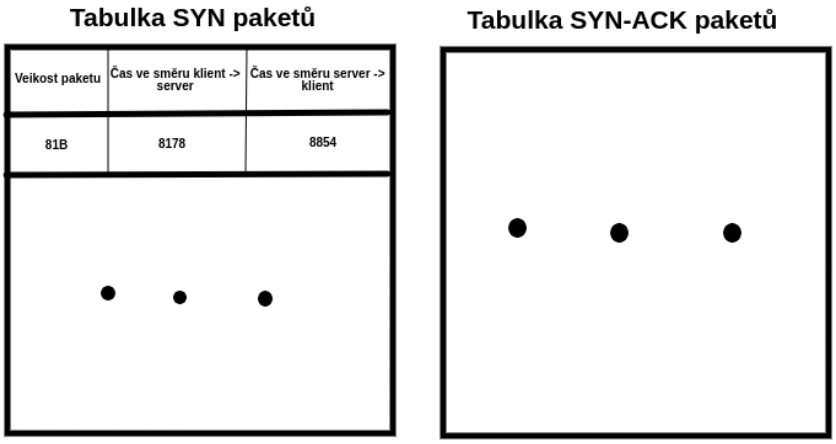
Kapitola 3

Realizace

Pro realizaci jsme vybrali verzi 2 našeho algoritmu s redukovaným FPR. Tento algoritmus bude implementován do exportéru síťových toků IPFIXprobe [28].



Tento software od společnosti Cesnet funguje následovně. Paket je přečten z pcap souboru nebo je zachycen na nastaveném síťovém rozhraní a je předán do flow cache. Flow cache aktualizuje flow záznam nebo v případě že neexistuje ho vytvoří. Následně také projede každý připojený plugin a zavolá na něm metody `pre_create`, `post_create`, `pre_update`, `post_update`, `pre_export`. Pokud je flow záznam označen jako expirovaný je poslán dále ke zpracování do exportéru a ten ho odešle na cílový výstup.



Obrázek 3.1: Tabulka s časy viděných paketů

Tento software jak můžeme vidět je modulární a nám tedy stačí do něho zaimplementovat jeden plugin, který se bude starat pouze o detekci VPN provozu. O ostatní implementační detaily jako je čtení paketu z fyzického rozhraní sondy nebo odeslání výsledku na exportér se nemusíme starat, protože jsou do IPFIXprobe softwaru už zaiplementovány.

Naše implementace je k nalezení buď na přiloženém CD nebo ve forknutém projektu na adrese `https://github.com/honzov/ipfixprobe` ve větvi `vpn_automaton`. Zásadní implementace je k nalezení ve zdrojových souborech `vpn_automaton.hpp` a `vpn_automaton.cpp`.

Pojďme ale nyní k implementaci. Ta bude velice odlišná od testovací implementaci v pythonu, kde jsme nebrali v potaz výpočetní rychlost. Jelikož my potřebujeme aby náš plugin byl schopen pracovat na vysokorychlostních sítích, tak ho musíme navrhnout trochu jinak. Nám se jevilo jako nejoptimálnější následující implementace.

Místo toho abychom generovali do paměti nové stavy automatu budeme využívat tabulky, která si bude pamatovat poslední čas viděného paketu dané velikosti a směru. Tyto časy jsou v jednotkách mikrosekund od začátku Unixové epochy. Toto si můžeme dovolit, protože my počítáme s tím, že každý paket v SSA sekvenci spadá do rozmezí 60 až 150 bytů. Teoreticky tedy bude taková tabulka velká pouze 2.8 kB na jeden flow záznam. Tato tabulka je znázorněna na obrázku 3.1.

Tato tabulka je v kódu naimplementována pomocí struktury

```
struct pkt_table
{
   pkt_entry table_[PKT_TABLE_SIZE];
   ...
```

```
}
```

, která zároveň řeší přepočet délky paketu na index do této tabulky. Tyto tabulky máme pro jeden flow záznam dvě a to konkrétně jednu pro SYN pakety a jednu pro SYN-ACK pakety. Pro ACK pakety si nic pamatovat nemusíme, protože ty reprezentují koncový stav automatu.

Pokud nám tedy například přijde paket s velikostí 130 bytů, aktualizujeme tabulku se SYN pakety (konkrétně zapíšeme do řádku 130 ve směru příchozího paketu jeho čas záchytu), zkusíme najít příslušný paket v SYN tabulce pokud by tento příchozí paket reprezentoval SYN-ACK a to stejné pro ACK. Poznamenejme, že najití příslušného předchozího paketu trvá vždy jen pár iterací přes indexy předchozí tabulky, protože počítáme s tím, že předchozí paket v SSA sekvenci je maximálně 12 bytů větší.

V naší implementaci si tedy plugin pamatuje pouze následující proměnné.

```
uint8_t possible_vpn {0};
uint64_t suspects {0};
uint8_t syn_pkts_idx {0};
uint8_t syn_pkts[SYN_RECORDS_NUM];

pkt_table syn_table{};
pkt_table syn_ack_table{};
```

`syn_table` a `syn_ack_table` jsou tabulky už v předchozím odstavci popsané, `possible_vpn` je výstupní proměnná nabývají hodnot 0 - flow záznam není VPN a 1 - flow záznam je VPN. `syn_pkts` je pole o 40 prvcích, kam si ukládáme velikosti označených SYN paketů pro další analýzu popsanou v sekci 2.4.1.6. `syn_pkts_idx` je index do tohoto pole.

Celý algoritmus je naimplementován v následujících funkcích.

1. `VPN_AUTOMATONPlugin::update_record`
2. `VPN_AUTOMATONPlugin::post_create`
3. `VPN_AUTOMATONPlugin::post_update`
4. `VPN_AUTOMATONPlugin::pre_export`

Ve funkci 1 je schovaná prakticky celá implementace. Tato funkce bere na vstup ukazatel na strukturu dat flow záznamu a příchozí paket. Následně provede všechny nutné přechody a zapíše potřebné změny do paměti. Je také dobré podotknout, že tato funkce počítá už s tím, že paket jako vstupní parametr je v rozmezí velikosti paketu SSA sekvence. Tato kontrola není zahrnuta do této funkce z důvodu, aby kontrola proběhla co nejdříve a tím jsme ušetřili drahý výpočetní čas.

Funkce 2 je volána po zachycení prvního paketu z toku a vytvoření flow záznamu. V této funkci tedy dochází k alokaci všech potřebných dat a zpracování paketu pomocí funkce `update_record`.

Funkce 3 je volána po záchytu každého nového paketu (krom prvního). V ní se načtou data z paměti a opět se pouze zavolá metoda `update_record`.

Funkce 4 je volána v případě, že je potřeba exportovat flow záznam. Ta si načte doposud spočítané charakteristiky toku a rozhodne na základě statistik z sekce 2.4.1, jestli je tok opravdu VPN tok. Toto pak zapíše do výstupní proměnné `possible_vpn`.

U této metody je také třeba podotknout, že oproti implementaci v Pythonu se zde náš algoritmus trochu liší. Z důvodů paměťové efektivity jsme se rozhodli nepamatovat si všechny pakety označených SSA sekvencí, ale pouze velikosti SYN paketů. To si můžeme dovolit, protože statistika měřená na těchto paketech byla průměrována. Tato změna mohla náš algoritmus ještě vylepšit. Platí totiž, že velikost každého SYN paketu bude vždy stejná (první inicializační paket od klienta posílá stejné parametry) a naopak velikosti SYN-ACK a ACK se v závislosti na konkrétních navazování spojení už mohou lišit (zde už hraje roli i server, který bude použit). Toto by tedy mohlo zmenšit statistiku použitou v sekci 2.4.1.6.

Toto je tedy zhruba kostra naší implementace. Je potřeba říci, že v kódu probíhá více věcí. Například registrace pluginu, specifikace výstupních formátů. To ale zde už probírat nebudeme, z důvodu, že vše je k nalezení na přiloženém CD.

Kapitola 4

Testování

V testování naší implementace se bylo potřeba zaměřit dvě věci. Za prvé, že implementace je korektní a odpovídá kvalitou výsledků implementaci v jazyce Python z minulé kapitoly a za druhé, že po nasazení do reálného provozu na vysokorychlostní síti bude plugin stíhat a nezpomalí žádné další části exportéru ani žádný síťový prvek.

Dále jsme také otestovali implementaci valgrindem, abychom se přesvědčily, že nedochází ke špatné práci s pamětí. Toto testování proběhlo úspěšně a valgrind žádné problémy s prací s pamětí nezaznamenal.

4.1 Porovnání výsledků s implementací v Pythonu

To, že je implementace korektní jsme ověřili tak, že jsme spustili plugin pro datasety z předchozí kapitoly. Výsledky jsou následující.

Dataset	IPFix	Python
ipsec	0.89	0.55
ipsec-test	0.91	0.63
openvpn_tcp-test	0.59	0.58
openvpn_udp-test	0.45	0.19
wireguard	0.75	0.6
wireguard-test	0.87	0.75
SH1-novpn	0.006	0.001
SH2-novpn	0.02	0.005
SH3-novpn	0.07	0.06

Když tyto výsledky zprůměrujeme dostaneme 74% úspěšnost detekce pozitivních dat a 3.2% falešně pozitivních.

Celkově můžeme vidět, že výsledky jsou trochu jiné. IPFiax implementace oproti implementaci v Pythonu má tendenci více označovat toky za pozitivní a její TPR se výrazně zlepšil. Toto bude způsobeno tím, že naše implementace

v C++ si pamatuje pouze velikosti označených SYN paketů z SSA sekvence. Toto bylo diskutováno v odstavci 3 předchozí kapitoly.

Toto porovnání nám tedy ukazuje, že naše implementace do IPFix je dokonce úspěšnější než jsme čekali po výsledcích z analýzy z předchozí kapitoly. Bohužel nás myšlenka použít pro snížení FPR pouze SYN pakety místo všech paketů SSA sekvence jako je to děláno v sekci 2.4.1.6 nenapadla v předchozí kapitole a teoreticky po provedení analýzy z 2.4.1.6 bychom mohli dostat ještě slibnější výsledky.

4.2 Vyhodnocení časové efektivity

Časová efektivita byla zhodnocena na základě porovnání s ostatními pluginy do IPFIXprobe. Na testovacím pcap souboru se 109 M pakety a o celkové velikosti 101 GB byli postupně pouštěny tyto IPFIXprobe pluginy. Tls plugin, pstats plugin, náš vpn_automaton plugin a poslední byl spuštěn IPFIXprobe bez jakéhokoliv pluginu. Všechny tyto běhy byli spuštěny desetkrát abychom mohli uvažovat průměrný běh. Výsledky byli zachyceny pomocí linuxového nástroje „time“ [29].

Následuje tabulka výsledků. Real je reálný čas od spuštění měření do jeho konce. User je čas který proces vykonávající měření stráví v user mode a sys je čas který proces stráví v kernel mode.

Plugin	Real	User	Sys
Tls	2m13.8594s	1m34.995s	41.4132s
Pstats	2m12.9525s	1m34.2592s	42.1605s
None	2m8.7018s	1m30.4386s	40.5435s
Vpn automaton	2m15.4793s	1m36.3125s	41.77s

Z těchto výsledků plyne, že náš vpn automaton plugin má cca 2s zpoždění oproti ostatním pluginům. To je zanedbatelná hodnota, při standartním běhu trvajícím okolo 2m10s je to 1.5% zpomalení. S výslednou rychlostí 0.8 M (789855) paketů za sekundu jsme bez problému schopni tento plugin nasadit do reálného provozu.

Závěr

Tato práce se zabývá možnostmi detekce VPN provozu. Konkrétně se zaměřuje na analýzu VPN provozu pomocí charakteristických obrazů paketů TCP handshaku. Na základě teoretické i praktické analýzy se snaží navrhnout a naimplementovat takový algoritmus, který rozpozná VPN spojení na základě informací z toku paketů a který zároveň bude možno nasadit i na vysokorychlostních sítích.

Teoretická část se zabývá rešerší současných přístupů k detekci VPN provozu a poté co představí tyto přístupy a základní fungování nejběžnějších VPN protokolů se pokusí navrhnout nový přístup založený na charakteristických vlastností TCP handshaku.

Na začátku praktické části je navržen prototyp našeho algoritmu v Pythonu. Tento prototyp je inkrementálně vylepšován na velkých datasetech a poté je jeho úspěšnost vyhodnocena na nezávislých datech. V této části se uplatní jak ML přístup, tak i vybrané statistiky z náhledu do poskytnutých datasetů. Výsledkem jsou dvě verze algoritmu. Jedna která maximalizuje TPR a druhá která minimalizuje FPR.

Dále byla v praktické části implementována verze algoritmu s minimalizací FPR do IPFIXprobe exportéru v C++. U této implementace byl na rozdíl od prototypové implementace kladen velký důraz na časovou efektivitu kvůli nasazení do exportéru na vysokorychlostních sítích. Tato implementace byla následně porovnána s implementací v Pythonu pro ověření korektnosti a pro ověření časové efektivity byla implementace porovnána oproti časové efektivitě jiných algoritmů implementovaných do IPFIXprobe knihovny.

Ve výsledku se ukázalo, že náš navržený postup a jeho následná implementace funguje s přesností 74% TPR a 3.2% FPR, kde hlavní problém falešných pozitiv je na některých typech UDP dat, například herních protokolů. Toto je velmi slušný výsledek a naši implementaci je možné nasadit do reálného prostředí vysokorychlostních sítí. To je možné i hlavně z toho důvodu, že naše implementace není náročná na výpočetní prostředky sondy.

Literatura

[1] Cisco Systems, Inc.: Cisco Annual Internet Report (2018–2023). 2020. Dostupné z: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] Cloudflare, Inc.: Tunneling in networking. 2023. Dostupné z: https://www.cloudflare.com/en-gb/learning/network-layer/what-is-tunneling/

[3] Hewlett Packard Enterprise Development LP: IPSec Encapsulation modes. 2017, cit. 10.03.2023 [online]. Dostupné z: https://techhub.hpe.com/eginfolib/networking/docs/switches/5130ei/5200-3946_security_cg/content/485048403.htm

[4] UC Regents: Securing Network Traffic With SSH Tunnels. 2023. Dostupné z: https://security.berkeley.edu/education-awareness/securing-network-traffic-ssh-tunnels

[5] Buchovecká, S.; Čejka, T.: Lecture notes in Computer Assisted Diagnosis. February 2013.

[6] Kim Porter: Are VPNs legal or illegal? 2023. Dostupné z: https://au.norton.com/internetsecurity-privacy-are-vpns-legal.html

[7] Cisco Systems, Inc.: Intranet VPN. Dostupné z: https://www.cisco.com/c/dam/global/da_dk/solutions/small-business/ivpn.pdf

[8] www.javatpoint.com.: OSI Model. 2021. Dostupné z: https://www.javatpoint.com/osi-model

[9] CompTIA, Inc.: What Is NAT? 2023. Dostupné z: https://www.comptia.org/content/guides/what-is-network-address-translation

[10] Grela, J.; Ożadowicz, A.: The Street Lighting Control System Application and Case Study. 06 2015, doi:10.1109/EBCCSP.2015.7300701.

[11] ShareTechnote: IP/Network. 2023. Dostupné z: https://www.sharetechnote.com/html/IP_Network_TCP.html

[12] Firewall.cx: Analyzing TCP header options. 2022. Dostupné z: https://www.firewall.cx/networking-topics/protocols/tcp/138-tcp-options.html

[13] Cloudflare, Inc.: How IPsec VPNs work. 2023. Dostupné z: https://www.cloudflare.com/learning/network-layer/what-is-ipsec/

[14] CactusVPN, Inc.: How Does OpenVPN Work? 2022. Dostupné z: https://www.cactusvpn.com/beginners-guide-to-vpn/what-is-openvpn/

[15] Jason A. Donenfeld: Wireguard, fast, modern, secure VPN tunnel. 2022. Dostupné z: https://www.wireguard.com/

[16] Alaidaros, H. M.; Mahmuddin, M.; Al Mazari, A.: From Packet-based Towards Hybrid Packet-based and Flow-based Monitoring for Efficient Intrusion Detection: An overview. 2012.

[17] Kaushik Sen: Top 6 Free Network Intrusion Detection Systems (NIDS) Software in 2023. 2023. Dostupné z: https://www.upguard.com/blog/top-free-network-based-intrusion-detection-systems-ids-for-the-enterprise

[18] Rahul Awati and Jessica Scarpati: Deep packet inspection (DPI). 2021. Dostupné z: https://www.techtarget.com/searchnetworking/definition/deep-packet-inspection-DPI

[19] Fortinet, Inc.: What Is SNORT? 2023. Dostupné z: https://www.fortinet.com/resources/cyberglossary/snort

[20] Hofstede, R.; Čeleda, P.; Trammell, B.; aj.: Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*, ročník 16, č. 4, 2014: s. 2037–2064.

[21] Hayes, M.: What is a Network Traffic Flow? 09 2018. Dostupné z: https://mattjhayes.com/2018/09/26/what-is-a-network-traffic-flow/

[22] LiveAction: Packet Vs. Flow: A Look at Network Traffic Analysis Techniques. 2022. Dostupné z: https://www.liveaction.com/wp-content/uploads/2022/03/2022-White-paper_-Packet-vs-Flow_v2-_1_.pdf

[23] Minnie J. Hamilton: What is IP Reputation And How To Check It? 2022. Dostupné z: https://www.vpnranks.com/ip-address/ip-reputation/

[24] Goel, A.; Kashyap, A.; Reddy, B. D.; aj.: Detection of VPN Network Traffic. In *2022 IEEE Delhi Section Conference (DELCON)*, 2022, s. 1–9, doi:10.1109/DELCON54057.2022.9753621.

[25] Xue, D.; Ramesh, R.; Jain, A.; aj.: OpenVPN is Open to VPN Fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Srpen 2022, ISBN 978-1-939133-31-1, s. 483–500. Dostupné z: `https://www.usenix.org/conference/usenixsecurity22/presentation/xue-diwen`

[26] OpenVPN Community: VORACLE attack and OpenVPN. 2018. Dostupné z: `https://community.openvpn.net/openvpn/wiki/VORACLE`

[27] Rohan Joseph: Grid Search for model tuning. 2018. Dostupné z: `https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e`

[28] CESNET: IPFIXprobe. 2023. Dostupné z: `https://github.com/CESNET/ipfixprobe`

[29] Andries Brouwer: time — Linux manual page. 2023. Dostupné z: `https://man7.org/linux/man-pages/man1/time.1.html`

Příloha **A**

Seznam použitých zkratek

TPR True positive rate

FPR False positive rate

SSA sekvence Sekvence paketů SYN, SYN-ACK, ACK

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `DP.pdf` ..... Text práce
- `ipfixprobe .2 process`
  - `vpn_automaton.hpp` ..... Header soubor pro VPN automaton implementaci
  - `vpn_automaton.cpp` ..... Source soubor pro VPN automaton implementaci
- `src` ..... Adresář se soubory z kapitoly 2
- `testing` ..... Adresář s výsledky ze sekce 2.4.1