VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AUTOMATICKÁ ANOTACE SÍŤOVÉHO PROVOZU NA ZÁKLADĚ SYSTÉMOVÝCH UDÁLOSTÍ

AUTOMATED ANNOTATION OF NETWORK TRAFFIC BASED ON SYSTEM EVENTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAN KALA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN ŽÁDNÍK, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Kala Jan, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Kybernetická bezpečnost
Název: **Automatická anotace síťového provozu na základě systémových událostí**
Automated Annotation of Network Traffic Based on System Events
Kategorie: Bezpečnost
Zadání:

1. Seznamte se s aktuálními metodami pro klasifikaci síťové komunikace pomocí IP Flows, které se používají pro reprezentaci síťové komunikace ve velkých sítích (např. pomocí open source nástroje ipfixprobe).
2. Zaměřte se na problematiku anotace datových sad pro strojové učení.
3. Sestavte množinu informací, kterou je možné získat z běžných koncových zařízení s operačními systémy Microsoft Windows, GNU/Linux, MAC OS. Vyberte takové informace, které je vhodné využít k anotaci souvisejícího síťového provozu.
4. Navrhněte nástroj pro získávání informací z vybraného operačního systému a nástroj pro párování těchto systémových informací se síťovými toky.
5. Navržené nástroje implementujte.
6. Vyvinuté nástroje otestujte a vytvořte pomocí nich datovou sadu obsahující vybrané typy síťového provozu (seznam bude vytvořen na základě dohody s vedoucím práce).
7. Diskutujte dosažené výsledky a navrhněte možná rozšíření.

Literatura:

- G. Aceto, D. Ciuonzo, A. Montieri, V. Persico and A. Pescapé, "MIRAGE: Mobile-app Traffic Capture and Ground-truth Creation," *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1-8, doi: 10.1109/CCCS.2019.8888137.

- https://github.com/CESNET/ipfixprobe

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 4.

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

Vedoucí práce: **Žádník Martin, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 29. října 2021

Abstrakt

Tato diplomová práce se zabývá tématem anotace síťových toků pomocí dat z webového provozu. Seznamuje s problematikou monitorování síťových toků, jejich analýzy a klasifikace a také s protokoly HTTP a HTTPS. Popisuje techniku sběru dat z webových prohlížečů a jejich párování se síťovými toky. Navrhuje anotační systém, který je schopný automaticky anotovat webový provoz. Implementace navrhnutého systému je též součástí této práce.

Abstract

This thesis adresses topic of network flow annotation using web traffic data. Introduces to problematics of network flow monitoring, analysis and classification and also to protocols HTTP and HTTPS. Describes technique of data collection from web browsers and their pairing with traffic flows. Proposes annotation system that is able to annotate web traffic in automated manner. Implementation of the proposed system is also part of this thesis

Klíčová slova

Monitorování datových toků, analýza síťového provozu, anotace webového provozu, HTTP, HTTPS, TLS, ipfixprobe, rozšíření prohlížeče.

Keywords

Network traffic monitoring, network traffic analysis, web traffic annotation, HTTP, HTTPS, TLS, ipfixprobe, browser extension.

Citace

KALA, Jan. *Automatická anotace síťového provozu na základě systémových událostí*. Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Žádník, Ph.D.

Automatická anotace síťového provozu na základě systémových událostí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Martina Žádníka PhD. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jan Kala
23. května 2022

Poděkování

Velmi děkuji svému vedoucímu panu Ing. Martinu Žádníkovi, PhD. za pravidelné konzultace a motivaci při vedení této diplomové práce. Dále bych rád poděkoval panu Ing. Dominiku Soukupovi, s nímž jsem diskutoval technické podrobnosti a jehož spolupráce si velmi vážím. V neposlední řadě bych chtěl poděkovat své rodině za morální podporu během celého studia.

Obsah

Kapitola 1

Úvod

Kybernetická bezpečnost nikdy nedosáhne 100 %[1]. Stále se vyvíjející technologie znamenají nové výzvy k zabezpečení uživatele na síti. Antivirové programy a systémy firewall dokáží chránit uživatele před již známými hrozbami. Pokud ale útočník využije neznámé kybernetické hrozby (hrozba nultého dne, angl. *zero-day attack*), je třeba využít dalších nástrojů, které nám pomohou nebezpečí detekovat předtím, než napáchá nějakou škodu. I k tomuto účelu lze využít data nasbíraná pomocí systému monitorujícího síťový provoz. Lze například pomocí analýzy modelovat běžné chování uživatelů sítě. V případě odchylky od normálního stavu během sledování lze dále řešit, zda se jedná o reálnou hrozbu, či validní komunikaci.

Výzvou těchto analytických nástrojů je právě klasifikace provozu. Se vzrůstajícím počtem způsobů anonymizace provozu je stále obtížnější klasifikovat, o jaký typ provozu se doopravdy jedná. Příkladem může být použití VPN (angl. *Virtual Private Network*) či proxy serverů, pomocí kterých dokážeme skrýt cílovou destinaci odesílaných paketů. Pokud ještě bereme v úvahu použití šifrování odesílaných dat, stává se analýza takového provozu téměř nemožnou. Proto se postupně vyvíjí i anotační nástroje, které mají za cíl poskytnout potřebné informace analytickým nástrojům pro zvýšení přesnosti detekce neznámých hrozeb.

Původním cílem této práce bylo vytvoření anotačního nástroje, který by vyhodnocoval systémové události jednotlivých klientů a tyto stavy poskytoval k analýze. Při zkoumání existujících řešení jsem však došel k tomu, že již existují anotační nástroje pro získávání systémových informací, které poskytnou přesnější pohled na komunikující procesy než samotná analýza systémových událostí. Nedostatkem těchto existujících nástrojů však byl chybějící pohled na jeden konkrétní typ provozu, a to webový provoz. Ten již od 90. let tvoří většinu globálního internetového provozu [2] a nabízí tedy širokou oblastí zkoumání. Rozhodl jsem se proto zaměřit převážně na anotaci webového provozu pro účely analýzy bezpečnostních hrozeb. Tato volba byla diskutována a schválena vedoucím práce Ing. Martinem Žádníkem, PhD.

Cílem této práce je tedy návrh a implementace anotačního nástroje, který poskytuje informace dostupné ohledně webového provozu na zařízení klienta sítě. Nástroj bude schopný anotovat datové toky zachycené pomocí sledování síťového provozu, které spadají do kategorie webového provozu. Další částí této práce bude vytvoření anotované sady dat, která bude obsahovat různé případy chování uživatele a která může dále sloužit k učení detekčních algoritmů analytických nástrojů.

[1] https://hbr.org/2017/12/you-cant-secure-100-of-your-data-100-of-the-time
[2] http://www.cs.unc.edu/~jeffay/papers/MASCOTS-03a.pdf

Kapitola 2

Monitorování síťového provozu

Ve středních a větších počítačových sítích je často zapotřebí zjišťovat stav jednotlivých klientů a monitorovat správnou funkčnost sítě. O to se stará systém pro správu sítě, jehož součástí bývá monitorování síťového provozu. Díky němu má systém správy sítě informace o dostupnosti jednotlivých poskytovaných služeb, probíhající komunikaci na síti a stavu jednotlivých linek. Na základě těchto informací může administrátor sítě řešit výpadky zařízení, plánovat rozšíření sítě nebo mít přehled o špičkách provozu v síti. Aktuálně existují dva přístupy k monitorování: **aktivní** monitorování založené na monitorovacím provozu a **pasivní** monitorování, které sleduje datové toky procházející určitým místem v počítačové síti. Pro celkový přehled stavu sítě a zařízení v ní se používá kombinace obou dvou přístupů.

V této kapitole si popíšeme oba dva přístupy zvlášť a porovnáme rozdíly mezi nimi. Zaměříme se především na pasivní monitorování, které bude dále v práci použito jako základ anotačního nástroje. Dále si popíšeme i využití nasbíraných dat pro účely analýzy datových toků a nakonec si představíme nástroje firmy CESNET, která se zabývá pasivním monitorováním a analýzou síťových toků a jejíž projekty síťové sondy *IPFIXprobe* a síťového kolektoru *IPFIXcol2* byly použity v řešení této práce.

2.1 Aktivní

Obecně se aktivním monitorováním rozumí dotazování se zařízení na jeho stav (aktivní komunikace se zařízením). Dotazování může probíhat buď periodicky, nebo neplánovaně[8], kdy každý způsob má svoje konkrétní použití. Periodické dotazování použijeme v případě stálého sledování stavu zařízení, které je pro nás středem zájmu. Získaná data můžeme využít k tvorbě grafů a přehledů vytíženosti sítě či služeb. Při periodickém dotazování je důležité zvolit správný interval mezi jednotlivými dotazy. Při nízkém intervalu dochází ke zbytečnému vytěžování pásma, čímž může docházet k rušení ostatní komunikace na síti. Naopak při nastavení vysokého intervalu můžeme přicházet o informace ze zařízení nebo je dostávat opožděně.

Oproti tomu neplánované dotazy odesíláme v tu chvíli, kdy se potřebujeme v určitou dobu dozvědět více informací z monitorovaného zařízení (například ve chvíli výpadku části sítě). Většinou se jedná o tzv. *ad-hoc* dotazy, které obsahují požadavky závislé na typu situace. V kombinaci s periodickým dotazováním se dají použít neplánované dotazy v případě překročení prahu sledovaných hodnot pro zjištění více informací o stavu zařízení.

Aktivním monitorováním můžeme získat nejen informace o stavu zařízení, ale i informace ohledně síťového provozu a dostupnosti služeb. Při takovém postupu se vysílá do sítě monitorovací provoz a na základě naměřených parametrů tohoto provozu se poté vyhodnocuje stav linky. Jedná se například o příkazy `ping` nebo `traceroute`. Tímto typem monitorování zjistíme nejen dostupnost síťových rozhraní, ale například i dobu, jak dlouho požadavek trval, nebo cestu, kterou byl požadavek směrován. Je však důležité brát v potaz, že samotný monitorovací provoz ovlivňuje naměřené výsledky, jelikož je sám součástí síťového provozu. [10]

2.1.1 Princip

Základní architektura aktivního monitorování se skládá z těchto zařízení[8]:

- **Spravovaná zařízení** Jedná se o konkrétní klienty sítě, jejichž stav systém sleduje.
- **Řídící jednotka** Hlavní prvek aktivního monitorovacího systému, který provádí dotazování. Zde se shromažďují nasbíraná data a zároveň se provádí i jejich vyhodnocování, jehož výsledkem může být například grafické znázornění získaných hodnot.
- **Komunikační Protokol** Komunikační standart používaný pro přenos dat mezi spravovanými zařízeními a řídící jednotkou. Komunikace může probíhat buď na základě výzvy (angl.*polling*), nebo na základě ohlašování stavu (angl. *event reporting*). Příkladem tohoto protokolu je SNMP (*Simple Network Monitoring Protocol*).

Při nasazení systému přidáme do sledované sítě zařízení, které bude sloužit jako řídící jednotka. Klienty sítě, jejichž stav chceme sledovat, nakonfigurujeme jako spravovaná zařízení, tedy umožníme jim komunikaci s řídící jednotkou pomocí zvoleného komunikačního protokolu (například zmíněný SNMP). Na řídícím zařízení poté přidáme tyto klienty sítě pomocí jejich IP adresy a poté můžeme volit různé styly dotazování. Při běžném používání monitorovacího systému se využívají prahy hodnot (angl. *treshold*) získaných periodickým monitorováním k vyvolání tzv. *alarmu*. Na základě alarmu lze pak upozornit administrátora na kritický stav nebo spustit automatické procesy pro zjištění příčiny.

2.2 Pasivní

Pasivní monitorování se zabývá sledováním probíhající komunikace na síti. Analýzou komunikace získáme především data ohledně stavu sítě bez nutnosti přídavného monitorovacího provozu. Tím ušetříme dostupné pásmo sítě a vyhneme se nepřesnostem, které nastávají u měření monitorovacího provozu. Nasbíraná data nám mohou posloužit k analýze jednotlivých komunikací nebo k celkové analýze chování uživatelů, což přispívá k bezpečnosti sledované počítačové sítě.[6]

2.2.1 Architektura systému

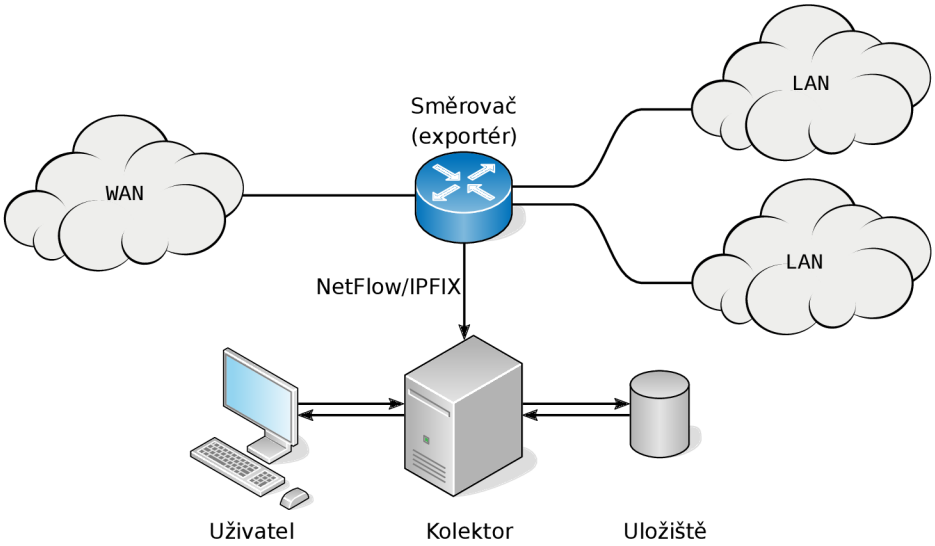
Pasivní monitorovací systém obsahuje několik hlavních kroků a to **pozorování**, **měření a export**, **kolekce dat** a následná **analýza dat**.[5]

1. Pozorování probíhá pomocí sondy umístěné v síti. Pozorovací bod sondy může být například síťové rozhraní směrovače.

2. Měření a export. Během měření se agregují pakety do záznamů toků, přičemž jeden tok je definován jako *Set IP paketů procházející určitým bodem v určitý čas, kde všechny pakety patřící k jednomu datovému toku mají stejné, předem definované parametry*[5]. Těmito parametry jsou informace jako IP adresa a port příjemce, IP adresa a port odesílatele a použitý komunikační protokol [5]. Při ukončení toku a naměření jeho parametrů je tok předán exportovacímu procesu, který tok odesílá kolektoru.

3. Kolekce dat se stará o příjem přenášených toků ze sond v síti a jejich uložení. Předzpracování přijatých informací před uložením obsahuje další operace jako třeba filtrování nebo kompresi.

4. Analýza dat se provádí po jejich uložení kolektorem. Data mohou být analyzována buď ručně, například při zkoumání anomálií, nebo automaticky, kdy lze využít korelaci a klasifikaci dat.



Obrázek 2.1: Architektura systému pro monitorování datových toků [6]

Obvykle se více kroků sdružuje na samostatném zařízení. Jedno zařízení běžně vykonává pozorování, měření a export. Proces, který tyto kroky vykonává, se nazývá *exportér* (angl. *exporter*). Zařízení, na kterém exportér běží, se pak nazývá *sonda* (angl. *probe*). [5]

Kolektor, jako samostané zařízení, pak vykonává především kolekci záznamů od sondy či exportéru. Záznamy jsou přenášeny mezi exportérem a kolektorem pomocí protokolu jako třeba NetFlow od firmy CISCO nebo protokolu IPFIX, jenž se stal standardem pro tento účel. Analýza záznamů buď může probíhat přímo v samotném kolektoru, nebo jsou data poskytována dalším zařízením.[6]

Exportér

Prvním krokem, který exportér vykonává, je pozorování paketů, což je proces, při kterém probíhá samotné zachytávání síťové komunikace ze síťového rozhraní. Zachytávání může být prováděno ve dvou módech. Tím prvním je *in-line* mód, kdy je sonda připojena do cesty mezi síťové klienty a komunikace prochází přímo sondou. Druhým módem je *mirroring* mód

(někdy také nazývané *port mirroring*), tedy zrcadlení, při kterém dochází k přeposílání komunikace z jednoho nebo více portů síťového zařízení (jako třeba směrovač nebo přepínač) na výstupní port, na kterém je připojená sonda. Tyto dva módy sledování provozu by měly pomoci k výběru správného místa umístění sondy v síti. Sonda by měla být nejlépe v části, kde jsou data přenášena pomocí kabelových spojení, a v místě, kde lze zjistit co nejvíce informací z procházející komunikace.[5]

Dále zachycená komunikace prochází vzorkováním a filtrováním. Díky vzorkování můžeme ušetřit výpočetní výkon sondy, a to tak, že sonda ignoruje některé procházející pakety. Taktika vzorkování může být různá, ale dvě základní taktiky jsou: výběr N-tého paketu a náhodný výběr paketu. Při filtrování je cílem dále zpracovávat pouze ty pakety, jež mají určitou vlastnost, a zahodit ty, které ji nemají. Filtrování může probíhat na základě porovnávání hodnot v paketech jako třeba IP adresa, číslo portu či použitý protokol. Nebo může být filtrování založeno na hašování, kdy se vytvoří otisk paketu nebo jeho částí a ten se pak porovnává s dopředu definovaným otiskem.[5]

V poslední fázi exportéru se provádí měření a export. Měřící proces agreguje zachycené pakety do záznamů toků. Agregované vlastnosti toku se v záznamu nazývají *informační elementy* (zkráceně IE). Ty mohou pocházet z různých vrstev zaobalení paketu. Existují proto například informační elementy pro MAC adresu z linkové vrstvy nebo IP adresu z vrstvy síťové. Dále lze do záznamů přidávat i informační elementy z aplikační vrstvy. Při získávání těchto elementů hovoříme o DPI (*Deep Packet Inspection*), při kterém exportér extrahuje nejen data z hlaviček paketu, ale i ze samotných přenášených dat. Musí mít proto přístup k těmto datům (problém se šifrováním) a musí znát jejich přesný formát. Záznamy toků se poté ukládají do mezipaměti *flow cache*, odkud je dále odesílá exportní proces. Záznamy jsou uložené ve flow cache po celou dobu trvání příslušného toku. Po ukončení toku může být záznam ihned označen za expirovaný nebo se může dále uchovávat v cache po delší dobu. [5]

Kolektor

Hlavní úlohou kolektoru je sbírání dat od jednoho a více exportérů a tato data ukládat a připravovat pro analýzu. První fází činnosti kolektoru je fáze příjmu dat. Každý z exportérů může používat nejen jiný transportní protokol (TCP, UDP), ale i aplikační protokol (Netflow, IPFIX), takže tuto prvotní fázi většinou vykonává více procesů kolektoru, které jsou rozděleny podle obsluhované kombinace transportního a aplikačního protokolu. Připojené exportéry ke kolektoru mezi sebou nekomunikují, takže navzájem neví o používaných šablonách jiných exportérů, proto se set používaných šablon v kolektoru udržuje pro každé spojení s exportérem zvlášť. [6]

Další fází je modifikace záznamů, kdy je možnost přijaté záznamy upravit před jejich uložením. Příkladem takovéto úpravy je anonymizace, kdy ze záznamů odstraňujeme informace, na základě kterých by bylo možné identifikovat komunikující klienty. Další úpravou pro potřebu analýzy může být naopak přidání informací, například ohledně času, kdy byl záznam kolektorem přijat.[6]

Poslední fází je ukládání dat a distribuce. Samotné úložiště může být řešeno ve volatilní paměti, která se vyznačuje svou rychlostí a náchylností na výpadek elektrického proudu,

nebo může být v perzistentní paměti, která není náchylná na výpadky proudu, ale je znatelně pomalejší. Při přímé analýze dat je vhodnější způsob ukládání do volatilní paměti, kdy nás zajímá samotný výsledek analýzy a ne data samotná. Při pozdější analýze je vhodnější perzistentní paměť, kde mohou být data uložena v relační databázi, ze které se později mohou získat pomocí dotazů. [5]

Protokol IPFIX

O největší rozvoj protokolů pro přenos dat mezi exportéry a kolektorem se nejvíce zasloužila firma Cisco, a to díky vývoji proprietárního protokolu NetFlow. Verze protokolu Netflow v5 nabízela přenos záznamů pomocí definovaných struktur informačních elementů, jejichž definice znal dopředu jak exportér, tak i kolektor. Právě pevně nadefinované struktury se staly omezujícími, což přivedlo firmu Cisco k vývoji nové verze protokolu, a to Netflow v9, na jehož definici je založen i protokol IPFIX. Hlavním rozdílem těchto dvou protokolů oproti verzi Netflow v5 je existence šablon, což jsou dynamické struktury informačních elementů. Exportér tedy může zasílat informační elementy v různém formátu, který ale musí nejdříve jako šablonu přenést na kolektor. Kolektor si ukládá přijaté šablony, které používá při interpretaci přijatých záznamů.

Informační elementy jsou základním prvkem pro stavbu šablon. Definují, jakým způsobem se mají přenášená data interpretovat. Informačním elementem je například IP adresa, časová značka počátku toku nebo počet paketů v toku. Definice informačního elementu musí obsahovat tyto části: název, datový typ, unikátní identifikátor informačního elementu, popis a stav platnosti. Seznam všech definovaných elementů je spravován organizací IANA, která registruje nová čísla informačních elementů a zveřejňuje ty stávající. Při vývoji informačních elementů a práci s nimi lze využít privátního rozsahu, který je přidělen stejnou organizací a je identifikován na základě přiděleného čísla PEN (*Private Enterprise Number*). Při použití informačních elementů v rámci privátního rozsahu není nutné nikde uvádět jejich definici.

Datové typy informačních elementů mohou být například základní jako znaménková i neznaménková celá čísla, čísla s plovoucí desetinnou čárkou, textové řetězce, pravdivostní hodnoty nebo časové značky. Dalšími datovými typy jsou typy síťových dat jako IPv4, IPv6 nebo typ MAC adresy. Příležitostně lze využít i typ bajtového pole pro případ, že přenášená data nelze převést pomocí existujícího typu nebo textové reprezentace.

Protokol je navrhnut tak, že se při přenosu dat snaží co nejvíce šetřit místo a tím i objem přenášených dat. Za tímto účelem tedy byly navrhnuty formáty definující různé velikosti pro celočíselné datové typy se znaménkem i bez znaménka. Konkrétně se jedná o velikosti 8, 16, 32 a 64 bitů. Tyto definované velikosti však neoznačují reálnou velikost dat ve zprávě IPFIX, ale jejich maximální velikost, tedy v případě datového typu `unsigned64` může být tento typ přenesen pomocí maximálně 64 bitů, ale také může být přenesen jen pomocí 8 bitů, pokud by nebylo využito zbylých 56 bitů. [6]

2.2.2 Analýza síťového provozu

Po úspěšném nasbírání záznamů datových toků se dostáváme k jejich analýze. Na základě požadovaného výsledku, který chceme analýzou získat, rozlišujeme **analýzu toků a re-**

portování, **monitorování služeb** a **detekci hrozeb**. Každá z těchto oblastí nám přináší jiný typ informací, přičemž i informační elementy, které se využívají při analýze, se od sebe liší v závislosti na zvolené analýze.[5]

Analýza toků a reportování

Při tomto typu analýzy využíváme faktu, že sonda je umístěna v takovém bodě, přes který prochází všechna komunikace většiny klientů sítě. Samotná analýza se zabývá filtrováním a vyhledáváním v záznamech datových toků, což může s předchozím předpokladem posloužit například při zpětném zkoumání výpadku v síti (který klient zrovna aktivně komunikoval v daný čas, jaké objemy dat přenášel, ...). Zároveň lze z nasbíraných dat sestavovat statistické přehledy, které nám vnesou větší vhled do chování uživatelů (se kterými IP adresami klienti nejčastěji komunikují, kteří klienti přenášejí největší objemy dat, ...). V neposlední řadě lze této analýzy využít při reportování nenadálých událostí na základě nastavených prahů (překročení prahu používaného pásma, překročení počtu aktivních připojení klienta, ...). [5] Vhodné je i zmínit, že na základě tohoto typu analýzy lze například i účtovat poplatky klientům za přenesená data, blokovat jim komunikaci při překročení datového limitu nebo účtovat zvláštní poplatky. Tedy praktiky, které většinou známe z oblasti mobilních sítí.

Monitorování služeb

Tento typ analýzy se zabývá především monitorováním poskytovaných služeb. Ze zachycené komunikace získáváme informace ohledně aktuálních připojení ke službě, velikosti obousměrného zpoždění (angl. *Round-Trip-Time*), časů odpovědi a využití pásma. Oproti předchozímu typu analýzy se při monitorování služeb více zaměřujeme na měřené vlastnosti toků než na obsah komunikace.[5]

Detekce hrozeb

V případě zkoumání hrozeb ze síťového provozu můžeme rozlišovat dvě využití. Prvním využitím záznamů o tocích je zkoumání, kteří klienti spolu vzájemně komunikují, na základě čehož lze odhalit hrozbu. Využívají se seznamy typu *blacklist*, které obsahují nebezpečné IP adresy a porty, jejichž výskyt v záznamech datových toků detekuje nebezpečnou aktivitu.

Druhým typem využití je analýza chování hrozeb v prostředí sítě a následovné modelování chování. V tomto typu analýzy nejprve zjišťujeme, jaký vzor komunikace mají různé hrozby. Uveďme si příklad na SSH brute-force útoku. U něj lze rozeznat 1. fázi skenování portů, kdy se útočník snaží připojit souběžně na velký počet různých portů. Poté nastává 2. fáze útoku, kdy vzroste počet paketů v jednotlivých tocích mezi útočníkem a obětí. Při této fázi útočník v jednom toku pošle určitý počet hesel, po kterém SSH oběti zavře TCP relaci a tím ukončí i datový tok. Vzápětí však útočník započíná novou TCP relaci a tím i nový datový tok, ve kterém opět zkouší zadávat hesla.

2.3 IPFIXprobe a IPFIXcol

Mezi existující nástroje v oblasti pasivního monitorování patří *IPFIXprobe* a *IPFIXcol2*, které vyvíjí firma CESNET. Oproti ostatním řešením, jako je třeba systém Netflow[1] od firmy Cisco nebo nástroje od firmy Flowmon[2], jsou oba dva projekty CESNETu dostupné jako *open-source*. Navíc nabízí širokou škálu použití díky existujícím modulům, které nástroje rozšiřují o různé přídavné funkce. Především nástroj *IPFIXprobe* bude využit k výslednému řešení této práce, proto si jej popíšeme včetně kolektoru *IPFIXcol2*, se kterým tvoří monitorovací systém. Popis těchto nástrojů vychází z diplomové práce[6] Ing. Lukáše Hutáka a z online dostupné dokumentace obou nástrojů.

2.3.1 IPFIXprobe

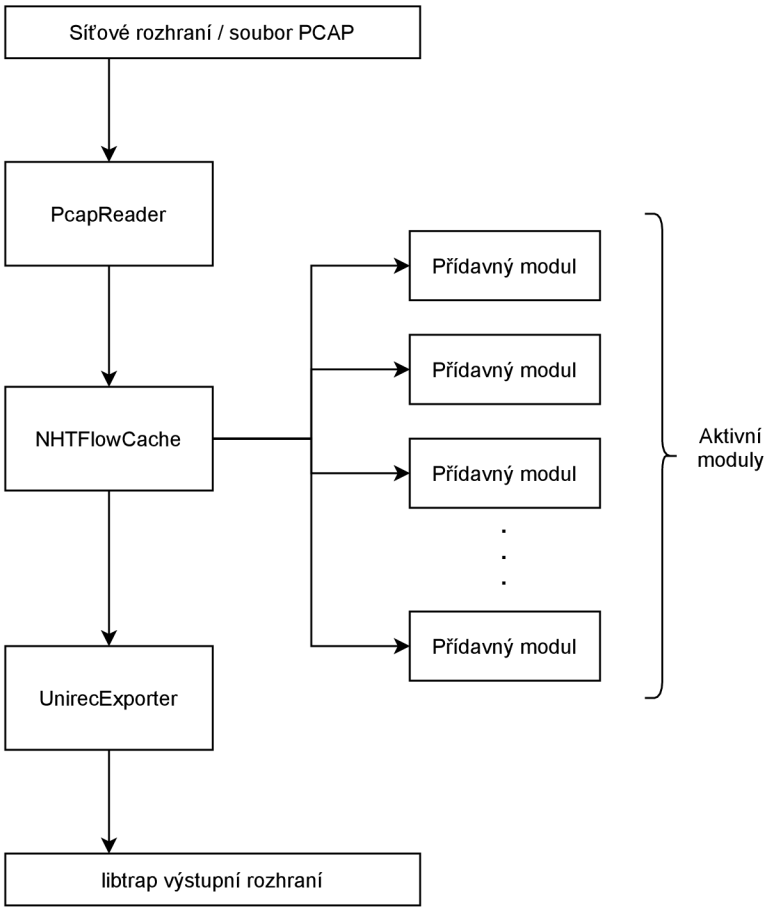
Je to exportovací proces v architektuře pasivního monitorování pro operační systémy UNIX. Jedná se o modulární exportér, jehož hlavní výhody jsou vícevláknové zpracování paketů a vysokorychlostní zpracování dat pomocí FPGA karty. Zároveň nabízí velké množství použitelných modulů (pluginů), které se dělí na **vstupní**, **zpracovávající**, **úložné** a výstupní.

Vstupní pluginy se starají o příjem dat v různých formátech. Na vstupu může být například paket přeposlaný vstupním síťovým rozhraním nebo i například soubor ve formátu PCAP. Na obrázku 2.2 je použitý jako vstupní modul `PcapReader`. Po úspěšném zpracování paketu se začne s vkládáním příslušného toku do tzv. *Flow Cache*.

Flow Cache při přijetí toku notifikuje všechny spuštěné moduly o akci, kterou se chystá vykonat nebo už vykonal. Konkrétně se jedná o akce `pre_create`, `post_create`, `pre_update`, `post_update` nebo `pre_export`, které mají všechny moduly implementovány. Zároveň Flow Cache uchovává aktivní datové toky, které posílá dále exportéru, pokud toky expirují, cache je plná nebo je přinucena jedním z modulů.

Výstupní formát dat IPFIXprobe může být buď IPFIX zpráva, textová reprezentace, nebo mohou být data ve formátu Unirec, který se používá k přeposílání a ukládání dat v rámci firmy CESNET.

[1] https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html
[2] https://www.flowmon.com/



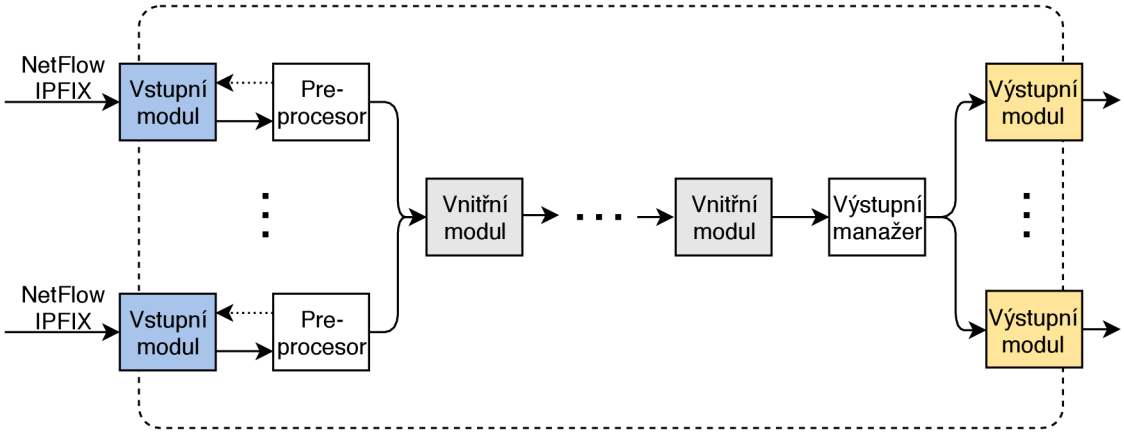
Obrázek 2.2: Architektura IPFIXprobe dle dokumentace https://github.com/cesnet/ipfixprobe

2.3.2 IPFIXcol

Vysokorychlostní kolektor záznamů datových toků. Aktuální verze je IPFIXcol2, která oproti předchozí verzi nabízí modulární architekturu. Příjem dat je řešen pomocí vstupních modulů, jejichž typy odráží existující transportní protokoly, tedy TCP, UDP a SCTP. Vstupní moduly pracují paralelně a nezávisle na sobě, tedy lze mít pro jeden typ transportního protokolu více vstupních modulů. Jakmile vstupní moduly zpracují přijatou IPFIX zprávu, převedou přijatá data do vnitřních struktur a odesílají dalším modulům ve formě zpráv.

Zprávy jsou hlavním interně komunikačním nástrojem kolektoru. Pomocí nich se vnitřní moduly navzájem informují o nastálých událostech, jako třeba vytvoření a zánik spojení s exportérem, nová datová zpráva či ukončení činnosti kolektoru.

Další v pořadí jsou pokročilé moduly, které se starají především o úpravu a práci s daty. Oproti vstupním a výstupním modulům jsou tyto moduly zřetězeny za sebou. Všechny příchozí zprávy tedy projdou stejnou sekvencí pokročilých modulů, než se dostanou k výstupním modulům. Pokročilé moduly jsou tedy úzkým hrdlem kolektoru, se kterým je třeba při nasazení a používání počítat.



Obrázek 2.3: Architektura IPFIXcol2 [6]

Jako poslední se záznamy datových toků dostávají k výstupním modulům. Ty stejně jako vstupní moduly pracují paralelně. Výstupní moduly mohou interpretovat data v různých datových formátech a ty buď odesílat dalším procesům, nebo je lokálně ukládat.

Kapitola 3

Anotace a klasifikace síťového provozu

Se stále narůstajícími požadavky na bezpečnost a anonymitu se zvyšuje i míra šifrovaného provozu na síti. Například podle Google Transparency Record[1] míra šifrovaného provozu v rámci Google stabilně stoupá. V období leden 2014 až leden 2022 stoupla míra šifrovaného provozu z 50 % na 95 %. Tento trend však činí analýzu síťového provozu stále obtížnější.

Je tedy třeba přistupovat na nové techniky v oblasti anotace síťového provozu. Takovými technikami anotace může být značkování - *tagování* síťových paketů na zařízení uživatele, či značkování zachycených toků na síťové sondě pomocí dotazování. V této kapitole si popíšeme obě techniky anotace a představíme si existující nástroje. Dále si popíšeme techniky klasifikace anotovaného síťového provozu.

3.1 Klasifikace

Za základě korektní klasifikace síťových dat můžeme rozeznávat komunikující aplikace v reálném čase, což nám poté umožňuje například prioritizovat určitou část provozu, vytvářet modely chování uživatelů nebo využít tyto informace jako statistické ukazatele.

Triviální klasifikací je rozeznávání komunikujících aplikací na základě známých IP adres a portů. Určité systémové procesy a protokoly využívají rezervovaná čísla portů (například SSH používá port číslo 22). V kombinaci se známými IP adresami fungoval tento způsob tak, že zachycené pakety byly agregovány do toků a přímým porovnáním IP adresy a portu byl tok klasifikován. Pokud však tok používal neznámou IP adresu a číslo portu, nastal problém s klasifikací tohoto toku. Postupem času se objevovalo stále více aplikací, které spíše než rezervované porty používaly dynamickou alokaci portů a tím se tato metoda stala pro klasifikaci nedostatečnou.[14]

Pokročilejší klasifikací je identifikace na základě přenášených dat (angl. *payload classification*). Při této klasifikaci jsou nejdůležitější částí paketu samotná přenášená data. Pro každý zkoumaný protokol poté musíme předem znát formát přenášených dat, která zkoumáme (například JSON formát pro webové aplikace). Tato metoda později přinesla hned dvě úskalí: nástroje pro klasifikaci jsou velmi náchylné na změnu formátu, kdy by se s každou novou verzí musely aktualizovat i klasifikační nástroje, a se stále se rozšiřujícím šifrováním provozu se stal tento typ klasifikace nepoužitelným. [14]

[1] https://transparencyreport.google.com/https/overview?hl=en

Zatím nejvhodnější technikou je klasifikace pomocí strojového učení. Při tomto rozeznávání se zaměřujeme na statistické informace toků jako třeba délka paketů nebo doba trvání toku. Základem je klasifikátor naučený na anotované sadě datových toků, kde u každého toku je navíc informace, které aplikaci daný datový tok náležel. Takto naučený klasifikátor je poté zapojený do monitorovacího systému a v reálném čase rozpoznává komunikující aplikace. [14]

3.1.1 Klasifikace strojovým učením

Algoritmy strojového učení pro klasifikaci síťových toků spadají do dvou kategorií: učené bez učitele (angl. *unsupervised learning*) a učené s učitelem (angl. *supervised learning*). Učené algoritmy bez učitele využívají shlukování, kdy na základě statistických informací datových toků shlukují toky, které mají podobné parametry. Učené algoritmy s učitelem mají trénovací sadu příkladů, které určují parametry jednotlivých shluků. Na základě těchto příkladů poté naučený algoritmus klasifikuje neznámé toky.

Jednotlivé parametry pro klasifikaci jsou statistické vlastnosti toků, jak již bylo zmíněno dříve. Tyto vlastnosti jsou získávány pomocí měření, během kterého získáváme například průměrnou délku paketu, dobu trvání toku nebo dobu mezi odpovědmi v toku. Těchto informací se dá vyvodit spousta, proto je důležité se zaměřit pouze na ty, které nejvíce ovlivňují chování algoritmu strojového učení, jelikož nadbytečné informace mají negativní dopad na výsledky algoritmu. Pro tento účel vznikly algoritmy pro redukci dostupných parametrů, které automatizují proces volby dostupných parametrů.

Konkrétní příklady tvorby klasifikačních algoritmů pomocí algoritmů jako například *Naïve Bayes*, *Bayesovská síť* nebo *C4.5 rozhodovací strom* jsou podrobně popsány v [14].

3.2 Anotace

Anotování síťového provozu je úkon, při kterém automatizovaně přidáváme tokům vlastnosti, které poté slouží k jejich přesnější klasifikaci. Jedná se většinou o vlastnosti, které nelze získat pouhým měřením. Jsou to například systémové informace (název procesu, číslo procesu, práva uživatele, ...) nebo informace o přenášených datech (aplikační protokol, způsob šifrování dat, formát dat, ...). Takto získané vlastnosti ale nemusí být vždy přínosné pro klasifikační algoritmus, proto musí stejně jako ostatní vlastnosti toků projít redukcí, jak bylo popsáno v 3.1.1.

3.2.1 Značkování paketů

První technikou anotace je anotace na úrovni síťových paketů. Tímto typem anotace se zabývá práce [15], ze které jsem čerpal informace pro tuto podsekci. Principem je přidávání systémových vlastností toku do hlavičky IP paketu na výstupním rozhraní klientského zařízení. Takto obohacený IP paket je poté zachycen síťovou sondou a buď může být přímo analyzován, nebo může být agregován do patřičného datového toku, který bude mít navíc vlastnost z hlavičky IP paketu.

Informace, které jsou použity v práci [15] jako přídavné systémové informace toků jsou:

- **Identifikátor uživatele** Obsahuje informaci o roli uživatele (či se jedná o administrátora, nebo ne) a dále heš identity z operačního systému.
- **Identifikátor aplikace** Název procesu odesílající paket.

Tyto informace jsou procesem běžícím v operačním systému posbírány v tu chvíli, kdy se na rozhraní zachytí odchozí paket. Následně jsou informace zašifrovány a vloženy do hlavičky paketu do pole *IP Options*. Následně je paket odeslán.

Zmíněné šifrování je velmi důležitou součástí tohoto typu anotace. Pokud bychom informace přenášeli v nešifrované podobě, vystavovali bychom uživatele bezpečnostnímu riziku, jelikož by potencionální útočník mohl zachytit takovýto paket a informace z hlavičky si bez problému přečíst a využít k útoku.

Hlavní nevýhodou anotace paketů je redundance informací. V případě, že síťová sonda agreguje pakety do toků, každý z těchto paketů obsahuje tytéž informace v poli *IP Options v hlavičce paketu* a tím zbytečně zvětšujeme hlavičku pro každý paket. V případě nasazení této technologie u všech klientů sítě by mohlo značně stoupnout využití pásma.

Zajímavým řešením zmíněného nedostatku je práce [1], ve které anotace paketů neprobíhá na zařízení uživatele, ale na síťových sondách, které se nazývají *iBox*. Tyto iBoxy provádí *deep packet inspection* (DPI), při které získavají informace z různých vrstev TCP/IP modelu, některé agragují do síťových toků, některé ponechávají spjaté s pakety a všechny tyto informace si poté mezi sebou posílají pomocí průchozích paketů. Informace však neduplikují, a to díky zprávám, které si mezi sebou posílají. Zajímavou částí této práce je právě přístup ke komunikaci a zasílání dat mezi iBoxy. Autoři se snažili o to, aby se s použitím iBoxů nemusel přidávat do sítě provoz navíc. Proto si iBoxy zasílají informace tak, že je vloží do průchozích paketů, které směřují k dalším iBoxům. Systém tímto získává výhodu v tom, že i při plném vytížení sítě jsou iBoxy schopné mezi sebou komunikovat.

3.2.2 Dotazování

Způsobem, jakým lze předejít redundanci dat, je použitím dotazování. V tomto případě by se jednalo o přímou anotaci toků v síťové sondě, přičemž informace potřebné k anotaci by z klientského zařízení získával lokálně běžící proces. Postup anotace je následující:

1. Sonda agreguje pakety v datový tok. Při vytvoření záznamu o toku, o kterém potřebuje znát více vlastností, odesílá dotaz na IP adresu klienta, která je obsažena v záznamu datového toku. V dotazu uvádí i číslo portu používaného klientem v komunikaci.

2. Proces běžící na zařízení klienta přijímá dotaz a sesbírá požadované informace. Ty pak odesílá v odpovědi zpět sondě.

3. Sonda přijímá odpověď a přidává informace mezi vlastnosti toku.

Kritickou částí dotazování je způsob sběru dat na klientském zařízení. Informace jsou získávány na základě IP adresy a portu klienta a IP adresy a portu vzdáleného zařízení, se kterým klient komunikuje, což jsou informace pro rozlišování jednotlivých datových toků. Úskalím tohoto přístupu je právě párování toku s požadovanými informacemi. V případě krátkých datových toků může být takový tok ukončen ještě před tím, než požadavek doputuje ke klientskému procesu. Při této situaci by klientský proces nemohl získat požadované

informace, neboť by port z požadavku byl již neaktivní, tedy nelze dále zjistit, která aplikace ho používala. Proto se tento přístup hodí převážně v případě déle trvajících datových toků, jejichž doba trvání je delší než průměrná doba komunikace mezi klientským procesem a sondou.

Jednou z možností komunikačního protokolu pro přenos požadovaných informací je protokol SNMP, pomocí kterého lze získávat informace ze systému na základě MIB (*Mangement Information Base*) tabulek. V těchto tabulkách jsou jednotlivé dostupné informace ze systému označeny hierarchickým identifikátorem, takže dotazující se sonda může zaslat v požadavku, o co přesně má zájem.

Dalším řešením může být i například dotazovací jazyk SQL, jehož použití realizuje nástroj OSQuery[2], který k anotaci datových toků využívá exportér *IPFIXprobe*.

OSQuery

Jedná se o multiplatformní *open-source* nástroj, který využívá SQL dotazy k poskytování informací o systému. Při popisu této kapitoly čerpám především z veřejně dostupné dokumentace[3]. Cílem OSQuery je přístup k systémovým informacím tak, jako by byly uloženy v relační databázi. K jejich získávání se využívá jazyk SQL, takže i dotazy se podobají dotazům do relační databáze. Ve skutečnosti tento nástroj obsahuje schémata, což jsou popisy tabulek, ve kterých se mapují názvy sloupců s funkcemi, které získávají informace. Uveďme si příklad na dotazu o získání jména procesu s PID=567:

SELECT name FROM processes WHERE pid=567

OSQuery přijme takový dotaz a vyhledá schéma `processes`. V jeho mapování hledá funkci, která odpovídá sloupci `name`.

Výhodou OSQuery je jeho multiplatformní použití, jehož jediná limitace je v oblasti dostupných schémat. Dále je to rozhodně poskytovaná abstrakce nad systémem, kterou nástroj poskytuje pro vývojáře a umožňuje mu využití již existujícího jazyka SQL k vytváření dotazů.

V současné době existuje anotační modul exportéru *IPFIXprobe*, který využívá OSQuery k získávání anotačních dat z klientského zařízení. Tento dotazovací systém byl mou inspirací při návrhu výsledného anotačního systému.

[2] https://osquery.io
[3] https://osquery.readthedocs.io

Kapitola 4

Získávání informací o webovém provozu

V předchozích kapitolách jsme si uvedli výhody monitorování síťových toků a jejich následné analýzy. Uvedli jsme si různé způsoby anotace toků pomocí systémových informací, které pozitivně ovlivňují následnou klasifikaci. Jak jsem již zmínil v úvodu, velká část síťových toků však skrývá daleko více informací, které i při aplikaci anotace zůstávají skryté klasifikačním nástrojům. Jedná se o webový provoz, jenž je v případě existujících anotačních nástrojů anotován na základě systémového procesu, který jej vyvolal, přičemž se už dále nepracuje s aplikačními metadaty a daty obsaženými v tomto typu provozu. Tyto informace mohou pomoci při hlubším zkoumání chování uživatele v relaci k jeho aktivitám na internetu, a to i v případě použití proxy serverů a virtuálních tunelů uživatelem. V této kapitole si tedy popíšeme způsob získávání informací z aplikační vrstvy webového provozu.

4.1 Prostředí

Pokud bychom pro získávání informací o webovém provozu uvažovali pouze prostředí počítačové sítě, narazíme na několik problémů:

- **Šifrování** Při použití šifrované komunikace nejsou aplikační data v zachycených tocích čitelná.
- **HTTP Proxy** Pokud uživatel nastaví ve svém webovém prohlížeči použití HTTP proxy, je třeba provádět DPI k získání informací o serveru, se kterým chce komunikovat. Při použití šifrování jsou však i tyto informace nedostupné. Podrobnější popis HTTP proxy je v 4.2.3.
- **VPN (angl. *Virtual Private Network*)** Při použití VPN je tunelována veškerá komunikace odcházející ze zařízení uživatele, tím pádem nelze rozeznat webový provoz od ostatního provozu a zároveň u něj nelze použít DPI k získání informací.

Uvažujeme-li například prostředí firemní sítě, kde lze omezit používání proxy serverů a VPN, stále nám zůstává problém se šifrováním, které je stále více používáno k přenosu webových dat. K překonání této limitace a zároveň k rozšíření použitelnosti získaných informací je tedy zapotřebí rozšířit prostředí o operační systém uživatelova zařízení. K tomu potřebujeme jeho spolupráci, tedy počítáme s proaktivním přístupem uživatele.

Pokud bude mít anotační systém přístup k datům v rámci zařízení uživatele, dokáže obejít problém se šifrováním tak, že si aplikační data a metadata získá ještě před jejich šifrováním. Bod šifrování aplikačních dat z webového prohlížeče probíhá na rozhraní mezi aplikací prohlížeče a operačním systémem. Ideálním prostředím pro získávání informací při použití šifrování je tedy webový prohlížeč.

Vraťme se ale zpět k dalším dvěma zmíněným problémům, a to proxy a VPN. Při použití HTTP proxy nebo tunelu je původní komunikace nejdříve směrována na proxy server, odkud poté putuje na webový server. V rámci počítačové sítě mohou nastat dvě situace, ve kterých je zachycena probíhající komunikace, a to při cestě požadavku na proxy server nebo při cestě požadavku z proxy serveru na webový server. V prvním případě lze takovou komunikaci zachytit a získat z ní požadovaná data, pouze pokud se bude jednat o nešifrovaný přenos (paket obsahuje IP adresu klienta a přenášená zpráva obsahuje doménové jméno webového serveru včetně požadavku). Při druhé situaci lze jen stěží zjistit, od kterého uživatele požadavek pocházel. Stále ale musíme brát v potaz šifrování, které nás vrací k prvnímu problému. Tedy i v případě proxy je nejlepším řešením získávat data z dotazů už přímo v prohlížeči.

Situace při použití VPN je podobná, jen s tím rozdílem, že všechna odchozí komunikace prochází šifrovaným tunelem. Při zachycení takové komunikace v počítačové síti nelze rozlišit webový provoz od ostatního provozu generovaného stejným klientem. Pokud se přesuneme do prostředí operačního systému klienta, pak zjistíme, že pakety vstupují do VPN tunelu na rozhraní operačního systému a počítačové sítě. To se týká jak nešifrovaných, tak i šifrovaných HTTP zpráv, takže se opět vracíme k problému se šifrováním a opět můžeme usoudit, že i zde je vhodným prostředím webový prohlížeč.

Ideálním místem pro sbírání aplikačních metadatad a dat webového provozu se tedy jeví webový prohlížeč na zařízení uživatele, jelikož se v tomto prostředí dokážeme vyhnout všem třem zmíněným problémům. Pro použití tohoto prostředí však potřebujeme proaktivní přístup uživatele, který ho musí zpřístupnit anotačnímu nástroji. Výzvou této volby je možnost používání více webových prohlížečů jedním uživatelem, tedy je třeba pokrýt i takovou situaci.

4.2 HTTP(S)

Zaměřme se tedy na prostředí prohlížeče, ve kterém je webový provoz ještě nezaobalen v IP paketu. V tomto prostředí se nacházíme v aplikační vrstvě TCP/IP modelu. Použitým protokolem je zde primárně protokol **HTTP** (angl. *Hyper Text Transfer Protocol*) a jeho bezpečnější verze **HTTPS**, kde S stojí za *Secure*, tedy bezpečný.

4.2.1 Popis

Hlavní protokol aplikační vrstvy webového provozu. Jeho princip je na základě požadavků (angl. **requests**) a odpovědí (angl. **responses**), které si mezi sebou vyměňuje klient a server. Při nejjednodušší situaci uvažujeme klienta, který nejprve započíná spojení se serverem, přes které odesílá požadavek na server za účelem získání webových zdrojů (HTML, CSS, Javascript). V tomto požadavku uvádí v hlavičce požadavku především identifikátor cesty

k požadovaném zdroji, který se nazývá *Uniform Resource Identifier* (**URI**). Dále může uvést přídavné informace v hlavičkách včetně těla požadavku. Server při přijetí požadavku vyhledá požadovaný zdroj a v HTTP odpovědi posílá zpět uživatelovi přes stejné spojení. O něco komplikovanější situace nastává při použití proxy, kterou si popíšeme v 4.2.3.

Protokol HTTP je bezstavový, tedy server si neukládá žádná data mezi separátními požadavky.[4]

4.2.2 Formát zpráv

Samotná HTTP zpráva má čitelnou podobu, tedy při zachycení nezašifrované zprávy lze bez většího úsilí přečíst obsah komunikace mezi klientem a serverem. Formát zprávy se liší podle toho, zda se jedná o požadavek, či odpověď. Formát požadavku je následující:

1. První řádek požadavku obsahuje název použité metody požadavku, URL (angl. *Uniform Resource Locator*) požadovaného zdroje a verze protokolu HTTP. Metod požadavku existuje několik a jejich použití závisí na typu zdroje, o který klient žádá. Konkrétně to jsou například metody: GET, POST, DELETE, PUT. URL je ve spojení s doménovým jménem serveru jedinečným identifikátorem zdrojového souboru na internetu.

2. Dále následuje dobrovolný seznam hlaviček, které jsou vzájemně oddělené novým řádkem. Seznam končí prázdným řádkem.

3. Poslední částí požadavku může být tělo požadavku. To je obsažené ve zprávě, pokud společně s požadavkem zasíláme data potřebná k získání zdroje.

```
GET / HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Upgrade-Insecure-Requests: 1
Host: example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.2 Safari/605.1.15
Accept-Language: en-GB,en;q=0.9
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Výpis 4.1: Příklad HTTP požadavku o webovou stránku http://example.com

Odpověď od serveru je zaslána přes stejné spojení, jaké vytvořil klient. Formát odpovědi se liší hlavně v prvním řádku, jinak je totožný s formátem požadavku. Popis formátu odpovědi je tedy následující:

1. První řádek odpovědi obsahuje verzi protokolu HTTP, status odpovědi vyjádřený kódem a textovou reprezentací.

2. Stejně jako u požadavku následuje dobrovolný seznam hlaviček ukončený prázdným řádkem.

3. Tělo odpovědi, které obsahuje požadované zdroje. Pouze v případě požadavku typu HEAD se nikdy v odpovědi nenachází tělo, jelikož požadavek HEAD slouží pouze k získání metadat ze serveru.

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Vary: Accept-Encoding
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Age: 421846
Content-Encoding: gzip
Expires: Tue, 01 Feb 2022 12:19:55 GMT
Cache-Control: max-age=604800
Date: Tue, 25 Jan 2022 12:19:55 GMT
Content-Length: 648
ETag: "3147526947"
Accept-Ranges: bytes
Server: ECS (nyb/1D15)
X-Cache: HIT
```

Výpis 4.2: Příklad HTTP odpovědi od serveru poskytující webovou stránku `http://example.com` (kromě těla odpovědi)

Hlavičky

Slouží k přenosu metadat požadavků a odpovědí v komunikaci mezi klientem a serverem. V požadavcích se objevují hlavičky, které přesněji specifikují požadovaný zdroj ze serveru nebo poskytují informace o klientovi, který požadavek odesílá. V odpovědi se též mohou vyskytovat hlavičky, které nesou informace o poskytovaném zdroji nebo o samotném serveru, který zdroje poskytuje. Přehled dostupných hlaviček je k nalezení například v dokumentaci organizace Mozilla[1]. V tabulce 4.1 jsou popsány některé z hlaviček, které lze v HTTP komunikaci nalézt.

Seznam hlaviček začíná v HTTP zprávě na druhém řádku a končí prázdným řádkem. Formát jednotlivých hlaviček je `id: hodnota`, kde *id* je identifikátor hlavičky a `hodnota` je přenášená hodnota hlavičky. Identifikátory jsou v čitelné textové podobě a odpovídají názvu, který popisuje přenášená data.

Identifikátor hlavičky	Popis
Host	Specifikuje doménové jméno serveru.
Accept	Udává formát dat, o které prohlížeč žádá.
Accept-Encoding	Určuje způsob komprese dat, která má být použita při přenosu požadovaných dat.
Content-Type	Formát dat, která jsou zaslána v odpovědi.
Content-Encoding	Použitá komprese dat v odpovědi.
Cookie	Obsahuje cookie přijaté od serveru, které se nachází v požadavku.
Set-Cookie	Obsahuje cookie zasílané serverem, které se nachází v odpovědi.
Connection	Určuje co se stane se spojením po vykonání požadavku.
Keep-Alive	Umožňuje udržet spojení aktivní i po vykonání požadavku. Spojení může být využito k více požadavkům

Tabulka 4.1: Popis některých z HTTP hlaviček

[1] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers`

4.2.3 HTTP proxy a tunel

Proxy server se chová jako klient i server zároveň, a to hlavně za účelem zastoupení odesílání požadavků od klienta směrem k serveru. Díky proxy je IP adresa původního klienta serveru skryta a tím se zvyšuje klientovo soukromí. Princip proxy spočívá v tom, že klient neodesílá požadavky přímo na server, ale místo toho je posílá na proxy server. Ten požadavky přeposílá dále webovému serveru, se kterým si žádá klient komunikovat, a jeho odpovědi přeposílá zpět klientovi. Existují dva typy proxy, a to transparentní a netransparentní. Transparentní proxy nijak neupravuje požadavky ani odpovědi od serveru, kromě požadavků pro autentizaci a identifikaci proxy serveru. Netransparentní proxy upravuje průchozí požadavky i odpovědi za cílem přidání služby, jako je třeba převod dat do jiného formátu. [4]

HTTP tunel se oproti proxy nezapojuje do komunikace mezi klientem a serverem. Tunel je tvořen proxy serverem na základě požadavku od klienta. Jedinou HTTP zprávou si klient požádá o vytvoření tunelu a dále už proxy server pouze přeposílá TCP tok mezi klientem a serverem.[4]

4.2.4 HTTPS

Jedná se o bezpečnostní rozšíření HTTP protokolu, ve kterém se využívá šifrování komunikace pomocí TLS (angl. *Transport Layer Security*). Proto je toto rozšíření známé i pod názvem *HTTP over TLS*. Samotné rozšíření nijak nemění formát HTTP protokolu, komunikace probíhá stejným způsobem, jako by probíhala pouze přes TCP protokol. Při popisu tohoto rozšíření a protokolu TLS vycházím z publikace [12].

Transport Layer Security (TLS)

Primárním přínosem TLS protokolu mezi dvě komunikující strany je přidání soukromí a integrity dat. První vrstvou tohoto protokolu je TLS Record vrstva, která leží těsně nad spolehlivým transportním protkolem TCP. TLS Record se používá k zapouzdření vyšších protokolů, kterým přidává navíc tyto vlastnosti:

- Spojení je soukromé. Použitá je symetrická kryptografie (algoritmy jako AES nebo RC4). Klíče jsou unikátní pro každé nové spojení a jsou vyměněny pomocí vyšší vrstvy protokolu, což je TLS Handshake.
- Spojení je spolehlivé. Přenášené zprávy obsahují kontrolu integrity dat, kromě zpráv, které jsou přenášeny za účelem smluvení šifrovacích parametrů.

Pro ustanovení šifrovacích parametrů se používá TLS Handshake vrstva, která je přenášena pomocí TLS Record vrstvy. TLS Handshake umožňuje klientovi a serveru se autentizovat a vyměnit si vzájemně informace o šifrovacím algoritmu včetně klíče, který bude použit k šifrování přenášených dat. Vlastnosti TCP Handshake protokolu jsou:

- Identita obou stran může být ověřena pomocí asymetrické kryptografie (algoritmy jako například RSA nebo DSA). Je vyžadováno ověření alespoň jedné strany komunikace.
- Výměna šifrovacího klíče je bezpečná, odolná proti naslouchajícím útočníkům na síti.
- Zároveň je výměna šifrovacího klíče spolehlivá. Útočník nemůže změnit vyjednávání o šifrovacím algoritmu a klíči.

Běžná komunikace pomocí TLS pak probíhá tak, že za pomocí protokolu TLS Handshake si nejdříve klient a server vzájemně zašlou `hello` zprávy, ve kterých se domluví na šifrovacích algoritmech a vymění si náhodná čísla pro potřeby těchto algoritmů. Poté si v rámci „handshake“ dále vymění potřebné kryptografické parametry, certifikáty pro autentizaci, vygenerují a zašlou si šifrovací klíč (tzv. *master secret*) a ten následně používají při komunikaci na vrstvě TLS Record. V další části komunikace si zasílají aplikační data pomocí TLS Record vrstvy.

Server Name Indication (SNI)

Jedná se o rozšíření TLS, které se využívá ve zprávách TLS Handshake protokolu. Konkrétněji se toto rozšíření nachází v `hello` zprávě, kterou zasílá klient serveru a obsahuje jmenný název serveru, ke kterému se chce klient připojit. Před zavedením tohoto rozšíření totiž nastával problém v situaci, kdy na jednom fyzickém serveru běželo více virtuálních serverů, přičemž všechny komunikovaly přes společnou IP adresu. Klient v takovém případě zaslal požadavek na IP adresu tohoto serveru, ale ten nedokázal rozhodnout, ke kterému z virtuálních serverů se chce doopravdy připojit. [3]

4.3 Přístup k HTTP zprávám v prohlížeči

Ve zvoleném prostředí prohlížeče dostáváme přístup k jednotlivým HTTP požadavkům pomocí rozšíření webového prohlížeče (angl. *browser extensions*). Problémem by se mohl zdát velký počet prohlížečů na trhu, a tím pádem velká škála různých implementací pro jednotlivé prohlížeče. Většina moderních prohlížečů je však založena na projektu zvaném *Chromium* od společnosti Google, která s ním zaznamenala velké úspěchy. Přes 90 % trhu[2] dnes tvoří prohlížeče buďto založené na jádru Chromia, nebo podporující jeho API pro vývoj rozšíření. Mezi nejznámější prohlížeče podporující API Chromia patří Google Chrome, Mozilla Firefox, Microsoft Edge nebo Safari. Díky tomuto unifikovanému přístupu k datům lze zajistit širokou kompatibilitu vyvíjených rozšíření webových prohlížečů.

4.3.1 Vývoj

Prvním prohlížečem, který začal používat rozšíření, se stal Internet Explorer v4.0, a to v roce 1999[9]. V následujících letech začaly podporovat rozšíření i ostatní prohlížeče jako Mozilla Firefox nebo Opera. Největší úspěch v této oblasti však zanamenal Google, který se svým prohlížečem Chrome začal jako první nabízet API pro vývojáře, které nabízelo vytváření rozšíření pomocí HTML, CSS a Javascriptu[2]. Rychle stoupající trend oblíbenosti Google Chrome donutil vývojáře ostatních prohlížečů k podpoře stejného API od Google a tím začala být rozšíření použitelná i mimo prohlížeč Chrome.

4.3.2 Události požadavku

Pro přístup k HTTP požadavkům a odpovědím existuje v Chromiu API nesoucí název **webRequest**. Toto konkrétní API nabízí možnost čtení a úpravy hlaviček požadavků,

[2]Data ze stránky https://netmarketshare.com/browser-market-share.aspx ze dne 25.1.2022

které odchází z prohlížeče na webový server. Zároveň i nabízí přístup i k hlavičkám odpovědí, které k požadavkům náleží. Příkladem využití *webRequest* API mohou být blokátory reklam na webových stránkách, které na základě URL v požadavku blokují požadavky na servery poskytující reklamní sdělení.

Než se požadavek zaobalí do IP paketu a odešle přes síťové rozhraní, proběhne v prohlížeči několik událostí, během kterých můžeme pomocí webrequest API přistoupit k datům probíhající komunikace. Každá z událostí má svůj určitý význam a pouze v některých událostech jde komunikace ovlivnit. V ostatních případech slouží události pouze jako informační element. Následující popis událostí vychází z dokumentace Google Chrome[3].

1. **onBeforeRequest**
   První událost, která je vyvolána předtím, než je vytvořeno spojení se serverem. V této události lze požadavek zrušit nebo přesměrovat.

2. **onBeforeSendHeaders**
   Pokud žádné rozšíření nezruší nebo nepřesměruje požadavek v předchozí události, nastává druhá událost. V této fázi jsou od prohlížeče připraveny hlavičky požadavku k odeslání, které mohou jednotlivá rozšíření měnit, přidávat a odebírat. Během této události je stále možnost zrušení požadavku.

3. **onSendHeaders**
   Po tom, co všechna rozšíření dostala možnost upravit hlavičky požadavku, je vyvolána ještě jedna událost před jejich finálním odesláním. Při této události již nemají rozšíření možnost upravovat odesílané hlavičky. Jedná se pouze o informativní událost všem rozšířením.

4. **onHeadersRecieved**
   Tato událost je vyvolána při každém přijetí hlaviček odpovědi. Během jednoho požadavku může být tato událost vyvolána několikrát, zvláště při různých přesměrováních požadavku nebo potřebě autentizace. V této fázi lze upravovat, přidávat a odebírat příchozí hlavičky. Zároveň můžeme na základě příchozích hlaviček požadavek přesměrovat nebo zrušit.

   (a) **onAuthRequired**
   Pokud je v přijatých hlavičkách informace o potřebě autentizace, vyvolává se tato událost. Během ní lze upravovat zasílané autentizační údaje nebo zrušit celý požadavek. Po této události se přechází zpět do fáze 2.

   (b) **onBeforeRedirect**
   Událost, která je vyvolána v případě přesměrovaní. To může být na základě odpovědi serveru, nebo mohlo být vyvoláno jedním z rozšíření.

5. **onResponseStarted**
   Při přijetí prvního bajtu se vyvolává tato událost, která obsahuje finální hlavičky odpovědi včetně řádku s kódem a statusem odpovědi. Opět se jedná pouze o informativní událost, změna hlaviček nebo těla odpovědi není možná.

6. **onCompleted**
   Finální událost, jež je vyvolána po přijetí celé odpovědi. Tato událost však neimplikuje i ukončení spojení, pouze ukončení přenášení odpovědi na požadavek.

[3] https://developer.chrome.com/docs/extensions/reference/webRequest/

7. **onErrorOccured**
   Ve fázi této události se můžeme ocitnout po jakékoli z výše popsaných událostí. Jedná se o chybovou událost, která nastane, pokud se v nějaké fázi objeví problém, kvůli kterému nelze dále v požadavku pokračovat.

4.3.3 Alternativní přístup k HTTP komunikaci

Dalším způsobem, jak programově získat přístup k probíhajícím HTTP komunikacím ve webovém prohlížeči je rozšíření vývojářských nástrojů, jak je opět popsáno v dokumentaci od Google[4]. Jednou z možností je rozšíření síťového nástroje, který je běžně dodáván ve verzích prohlížeče ke zkoumání síťové aktivity. Při použití takového rozšíření má vývojář přístup ke komunikaci ve formátu HAR (angl. *HTTP Archive*). Tento formát je založen na formátu JSON a obsahuje informace o proběhlé HTTP komunikaci [11]. V prohlížeči se datová sada ve formátu HAR dá získat právě pomocí síťových nástrojů. Nevýhoda tohoto přístupu je však závislost na interakci s vývojovými nástroji. Při běžném používání prohlížeče nejsou data vývojářským rozšířením poskytována, pouze pokud uživatel otevře okno s vývojářskými nástroji. Další nevýhodou je variabilní význam dat pro některá pole. Například význam pole `connections` je doslovně popsán takto: *"Unique ID of the parent TCP/IP connection, can be the client or server port number. Note that a port number doesn't have to be unique identifier in cases where the port is shared for more connections. If the port isn't available for the application, any other unique connection ID can be used instead (e.g. connection index). Leave out this field if the application doesn't support this info."*[11]. Pokud tedy neznáme zamýšlený význam pole tvůrcem záznamu, tak je hodnota tohoto pole redundantní.

Dalším způsobem přístupu k síťovým aktivitám prohlížeče je přímo úprava zdrojového kódu prohlížeče. Projekt Chromium je nabízen jako *open-source* a lze tedy vytvořit vlastní verzi prohlížeče, kde by rozšíření bylo přímo součástí výsledného programu. Zde je však nevýhodou složitá údržba takového projektu a také bezpečnostní rizika spojená s přímou úpravou zdrojového kódu.

[4] https://developer.chrome.com/docs/extensions/mv3/devtools/

Kapitola 5

Návrh anotačního systému

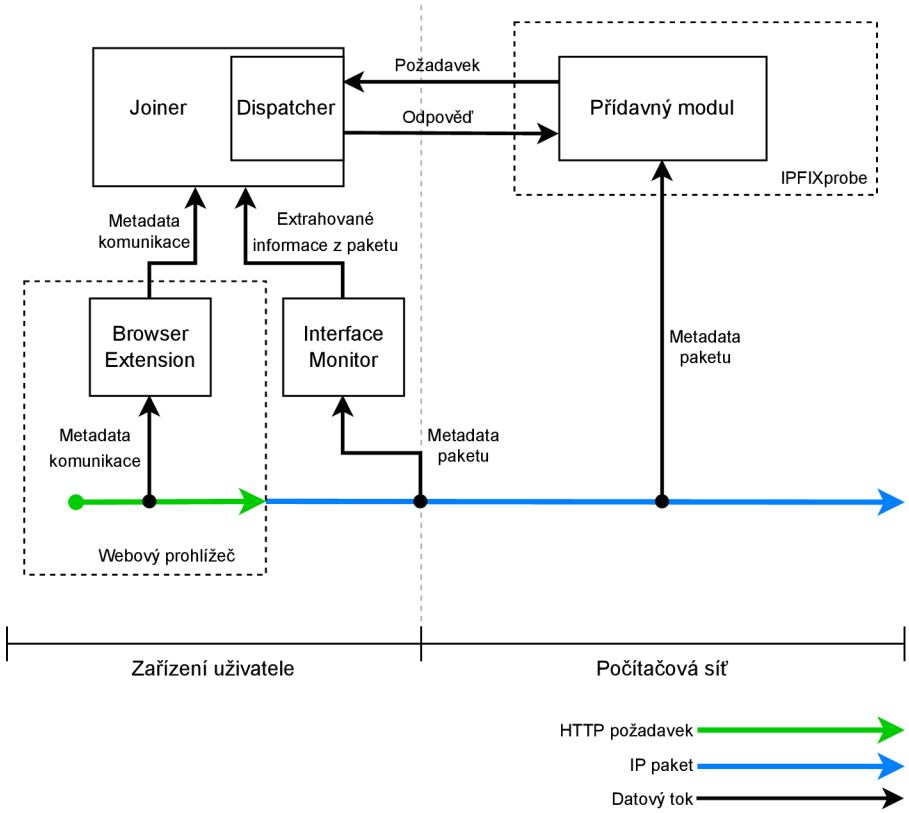
Prvním krokem k vytvoření anotované datové sady, je vytvoření anotačního systému, který bude z uživatelského zařízení sbírat informace o webovém provozu, které poskytne dále exportéru *IPFIXprobe*. Hlavní myšlenkou návrhu je neinvazivní sběr dat, při kterém bude cílem co nejméně ovlivnit dopad na aktivitu uživatele pro co nejpřesnější nasbírání vzorku. Při použití technik, které by zasahovaly do průběhu komunikace by se totiž mohlo lišit reálné chování od monitorovaného, a to z důvodů nezapříčiněných samotným uživatelem. Dále byly při návrhu brány v potaz možnosti vývoje, rozšíření a další údržby takového nástroje, neboť by v opačném případě nebyl použitelný s dalším vývojem webových prohlížečů a komunikačních protokolů.

V této kapitole navrhuji nástroje potřebné k sestavení takového systému a popisuji jejich jednotlivé komponenty. Popsáno je zde prostředí, ve kterých se nástroje nachází, způsob jejich činnosti a motivace pro zvolení daného návrhu. Dále jsou v této kapitole popsány i problémy, které výsledný návrh ovlivnily do výsledné podoby.

5.1 Architektura

U prvotního návrhu architektury jsem vycházel z již existujícího anotačního systému využívající nástroj *OSQuery* a síťovou sondu *IPFIXprobe*. Stejně jako OSQuery nástroj by i navrhovaný anotátor měl poskytovat sesbíraná data v reálném čase, přičemž přidanou hodnotou je i možnost uchování a archivace již skončených událostí. Inspirací však byl hlavně dotazovací mechanismus, díky kterému jsou data dostupná pro zpracování dalšími procesy, a to jak lokálními tak i vzdálenými.

Další podobností v obou architekturách je rozdělení na nástroje běžící na zařízení uživatele a přídavný modul sondy *IPFIXprobe*. Stejným způsobem je navržen i tento anotační systém, kde se nástroje na zařízení uživatele starají především o sběr dat z webového prohlížeče a přídavný modul v sondě se na ně dotazuje pomocí požadavků. Na obrázku 5.1 lze vidět vzájemnou komunikaci mezi jednotlivými nástroji. Pr účel sběru dat se na zařízení uživatele nachází rozšíření webového prohlížeče (angl. **Browser Extension**), monitor síťového rozhraní (angl. **Interface Monitor**) a párovací a komunikační modul (angl. **Joiner and Dispatcher**).



Obrázek 5.1: Architektura systému pro anotaci síťových toků

Druhá část anotačního systému se nachází v prostředí počítačové sítě. Jedná se o **přídavný modul** exportéru *IPFIXprobe*, jehož činností je anotace nasbíraných datových toků pomocí dat, která bude získávat pomocí dotazů na zařízení uživatele. Tyto dotazy zasílá před samotným exportem ukončeného datového toku ke zpracování na kolektor *IPFIXcol2*.

5.2 Zařízení uživatele

Jak jsme si již popsali v 4.1, pro získávání informací o webovém provozu se potřebujeme pohybovat v rámci webového prohlížeče, který webový provoz generuje. K tomu využíváme rozšíření prohlížeče (**Browser Extension**), které má od prohlížeče přístup k údajům o probíhajících HTTP komunikacích. Mezi těmito údaji však chybí podrobnější informace o lokálním síťovém spojení, které bylo využito k přenosu požadavku a odpovědi. Mezi poskytovanými daty tak nalezneme IP adresu a případně port (pokud se jedná o nestandardní port, tedy mimo port 80 a 443), ale chybí zde informace o lokální IP adrese a portu, které jsou potřebné pro párování s datovými toky.

Zde lze realizovat několik přístupů. Prvním z nich je vytvoření lokálního proxy serveru, přes který by webový provoz procházel a anotační nástroj by měl tady plnou kontrolu nad datovými toky. Tento přístup se však stává problematickým kvůli faktu, že se takto implementovaná lokální proxy stává přímým účastníkem komunikace. Mohla by se tedy stát i úzkým hrdlem a negativně tak ovlivnit nasbíraná data a následně i výsledný model chování,

který by byl naučený na takovém vzorku dat. Cílem nástroje tedy je pouze sledování, nikoli ovlivnění komunikace.

Další možností je využití systémových API k přístupu k datovým tokům či modelům. Jedno z takových API je využito na systému macOS od firmy Apple k implementaci VPN programů. Tento přístup je stejně jako v předchozím případě nevhodný, jelikož by se implementovaný proces opět stal součástí komunikace a navíc je zde velká závislost na platformě, na které má anotační nástroj fungovat.

Zvolenou možností k získávání informací o datových tocích je sledování průchozích paketů na síťovém rozhraní zařízení. V tomto případě se nástroj určený ke sledování nestává součástí komunikačního řetězce a svou činností neovlivňuje nasbíraná data do takové míry, jako předchozí dva případy. Výhodou tohoto přístupu je jednodušší přenositelnost na ostatní operační systémy. Nevýhodou je jeho náchylnost na použití HTTP proxy či VPN na zařízení uživatele. V takovém případě nemá sledovací proces dostatek informací k rozpoznání potřebných datových toků.

Komponentou, která provádí sledování síťového rozhraní je **Interface Monitor**. Ten má za úkol sledovat probíhající komunikaci uživatele a vyhledávat prvky HTTP a TLS komunikace. Tyto informace poté poskytuje poslední komponentě, a to párovacímu a komunikačnímu modulu (**Joiner and Dispatcher**). Tento proces přijímá informace jak od rozšíření prohlížeče, tak od monitoru síťového rozhraní a snaží se napárovat poskytnuté informace, které pak dále zpřístupňuje pomocí komunikačního modulu ostatním procesům.

5.2.1 Browser Extension

Zvoleným přístupem k implementaci sběru dat ve webovém prohlížeči je využití *browser extension* API, který splňuje především požadavky na jednoduchou udržovatelnost, širokou škálu využití a do budoucna i jednoduchou rozšiřitelnost o další funkce. Pro sledování probíhajících HTTP komunikací slouží *webRequest* API, které jako jediná část anotačního systému nabízí možnost vstoupit do cesty odchozím a příchozím datům. Tato schopnost by se při rozsáhlejším zpracování dat v prohlížeči mohla stát negativním vlivem, proto bude API využito pouze v režimu sledování.

V sekci 4.3 jsme si popsali několik druhů událostí, které nabízí *webRequest* API ke čtení metadat a dat požadavků. Pokud by rozšíření prohlížeče chtělo v některé z událostí rušit nebo přesměrovávat požadavek, potřebovali bychom při instalaci rozšíření povolení od uživatele. Ke sběru dat ale stačí pouhé čtení, čímž se vyhneme požadavkům o povolení.

Registrace k jednotlivým událostem probíhá pomocí handlerů, které prohlížeč při vyvolání události volá. Zde je zapotřebí zvolit správnou událost k tomu účelu, aby párování proběhlo s co největší přesností (podrobnosti o průběhu párování jsou popsány v podkapitole 5.2.3). Vhodnou událostí se tedy jeví `onResponseStarted`, která je vyvolána při přijmu prvního bajtu odpovědi a navíc poskytnutá data k této události již obsahují IP adresu a port komunikujícího serveru, což je velmi cennou informací k párování. Před samotným odesláním dat párovacímu procesu je ještě zapotřebí kontrola, zda-li data odpovědi nepocházejí z cache prohlížeče. Pokud ano, tak takový požadavek ignorujeme, a to z toho důvodu že taková data neputovala počítačovou sítí. Po této kontrole se metadata odesílají párovacímu procesu.

5.2.2 Interface Monitor

Komponenta sledující síťové rozhraní běžící jako samostatný proces slouží především k monitorování průchozích HTTP a TLS Handshake zpráv. Dále monitoruje i zprávy TCP protokolu s příznaky oznamující ukončení spojení. Důvodem této volby je dále popsaný párovací proces.

HTTP dotazy lze rozeznat v zachycených paketech jednoduše, a to tak, že v TCP datech vyhledáváme, zda-li se první 3 bajty rovnají slovu GET nebo zda-li se první 4 bajty rovnají slovu POST, což jsou v obou případech metody protokolu HTTP. Jelikož se jedná o nejvíce používané metody protokolu HTTP, budeme se soustředit především na tyto dvě. Pokud tomu tak je, tak se s velkou pravděpodobností jedná právě o zprávu protokolu HTTP.

Z protokolu TLS Handshake se zajímáme pouze o jediný typ zprávy, a to konkrétně *ClientHello*. Tento typ zprávy dokážeme také rozeznat porovnáním určitých bajtů v datech TCP paketu. Konkrétně se jedná o první bajt s hodnotou `0x16`, která odpovídá označení zprávy protokolu TLS Handshake. Dále to je pátý bajt, ve kterém hodnota `0x01` odpovídá označení konkrétní zprávy *ClientHello*. Pokud bychom brali v úvahu porovnání pouze těchto dvou hodnot, mohli bychom nesprávně označit zachycený paket jako zprávu protokolu TLS Handshake, a to v případě výskytu stejných hodnot na stejných pozicích což je možné riziko. Pro co největší eliminaci této situace tedy ještě můžeme přidat porovnání dvou hodnot v 1. a 9. bajtu, kde obě hodnoty odpovídají prvnímu ze dvou bajtů označující verzi protokolu TLS, která má ve všech aktuálních verzích na prvním zmíněném bajtu hodnotu `0x03`.

Oba protokoly HTTP i TLS sdílejí transportní protokol TCP. Protokol UDP v této práci není brán v úvahu kvůli jeho ojedinělému využití. A právě náležitá TCP spojení jsou chybějící informací, kterou se snažíme pomocí tohoto procesu zjistit. Proto také dochází k extrakci informací převážně z této vrstvy. Jsou to informace jako IP adresa a port příjemce a odesilatele, ale také je zde předmětem sledování i pole `Flags`, obsahující označení příznaků TCP zprávy, ve kterém se objevují příznaky TCP Fin (odpovídá hodnotě 1 prvního bitu) a TCP Rst (odpovídá nastavení hodnoty 1 třetího bitu).

5.2.3 Joiner and Dispatcher

Posledním dílem ke kompletaci nástroje běžícím na zařízení uživatele je párovací a komunikační proces zvaný **Joiner and Dispatcher**. Tento proces běží samostatně v operačním systému uživatele, stejně jako Interface Monitor, a jeho hlavní činností je příjem a párování informací od Browser Extension a Interface Monitoru. Po úspěšném párování poskytuje HTTP metadata napárovaná na datové toky pomocí komunikačního modulu zvaném **Dispatcher**, a to jak k ukončeným datovým tokům, tak i k těm aktivním.

Párování

Pokud by *webRequest* API od Google poskytovalo v prohlížeči informace o lokální IP adrese a portu, nebylo by potřeba sledování síťového rozhraní a tedy ani párování informací. Jelikož tomu tak není, je zapotřebí párovat datové toky na HTTP požadavky, tedy přiřazovat jim informace z prostředí počítačové sítě.

Před popisem párovací techniky je vhodné popsat problémy, se kterými se musí párovací proces vypořádat. Začněme tedy tím nejjasnějším problémem, kterým je šifrování.

Při použití nešifrovaného protokolu HTTP lze použít přímočarou techniku párování, kdy si ze zachyceného paketu na síťovém rozhraní dokážeme přečíst informace z aplikační vrstvy, které obsahují samotná data požadavku. Ty pak lze jednoduše spojit s metadaty požadavku z prohlížeče, a to na základě pouhého porovnání. V případě více totožných požadavků odeslaných ve stejnou dobu by pak teoreticky mohla nastat situace, kdy by jeden požadavek „předběhl“ jiný požadavek a byl by mu přiřazen nesprávný datový tok (pokud uvažujeme že každý požadavek využívá oddělený datový tok). V takovém případě by bylo na zváženou, zda li je tato chyba přijatelná, jelikož data požadavku jsou totožná, tedy i data odpovědi by měla být stejná. Dále stojí za zvážení, zda-li je potřebné párování u HTTP komunikace používat, jelikož sonda, která zachytí data na síti, má stejný přístup k datům v paketu, stejně jako Interface Monitor, tedy může si data sama přečíst. Hlouběji se však tímto problémem zabývat nebudeme, jelikož použitý styl párování vychází z techniky navrženou pro TLS, a to především kvůli zmíněnému problému s nekorektním napárováním.

Při použití protokolu TLS by se mohlo zdát, že zcela ztrácíme přehled o tom, který datový tok patří jaké komunikaci, jelikož jsou data požadavku šifrovaná, a tedy je nemůžeme porovnávat s metadaty požadavku z prohlížeče. V samotné TLS komunikaci se však objevuje prvek, který nám k tomuto párování může napomoci. Tímto prvkem je identifikátor SNI, který jsme si popsali v sekci 4.2.4, a který se objevuje ve zprávě ClientHello protokolu TLS Handshake. Jelikož identifikátor SNI odpovídá části *hostname* v požadovaném URL, které nalezneme v metadatech požadavku, vybízí se využití tohoto identifikátoru jako základem pro párování. Další informací, která je naopak dostupná z API webového prohlížeče, je IP adresa a port serveru, na který požadavek putoval, což jsou velmi cenné informace, které vylepšují přesnost párovacího procesu. Před dalším popisem si uveďme posloupnost událostí, které jsou z hlediska párování zajímavé:

1. Požadavek je vygenerován ve webovém prohlížeči uživatele a předtím než ho prohlížeč odešle do počítačové sítě, tak je vyvolána událost *onSendHeaders*, při které si rozšíření prohlížeče ukládá data HTTP požadavku, ale nikam je dále neposílá.

2. Následně může požadavek odcházet již ustanoveným spojením, nebo započít nové, při kterém se na síťovém rozhraní uživatele zachytává zpráva *ClientHello* pomocí Interface Monitoru.

3. Dále mohou nastat mezikroky ke správnému přesměrování a autentizaci uživatele, kde při každém obdržení hlaviček nastává v prohlížeči událost *onHeadersReceived*. Na tuto událost opět Browser Extension nijak nereaguje.

4. Jakmile je spojení k serveru správně nastavené tak započíná příjem dat odpovědi a v prohlížeči nastává událost *onResponseStarted*, na kterou již reaguje Browser Extension. V tuto chvíli lze konstatovat, že všechna potřebná spojení k přenosu požadavku jsou vytvořena a ustálena.

5. Po přijetí odpovědi si může prohlížeč vytvořená spojení stále udržovat aktivní dokud je sám nebo druhá strana neukončí. Spojení lze znovu využít k přenosu dalších požadavků (například při prohlížení a navigování se po stejné webové stránce). Konec spojení je poté indikován koncem TCP spojení, které TLS využívá.

Názornou sekvenci událostí lze vidět v sekvenčním diagramu 5.2.3, kde je naznačena komunikace mezi webovým prohlížečem, serverem a znázorněny jsou události, které lze zachytit

na síťovém rozhraní. V tomto názorném případě se jedná o vytvoření nového spojení, zaslání jednoho požadavku a příjem dat odpovědi. Po konci komunikace klient uzavírá spojení pomocí zpráv TCP Fin. V diagramu lze vidět, v jakých fázích probírá vyvolání dvou událostí ve webovém prohlížeči, které jsou součástí navrženého sběru dat pro párování a anotaci.

Zde se konečně dostáváme k dalšímu problému, který je potřeba při párování řešit. Jak je zmíněno v 2. bodě popsané posloupnosti událostí, odchozí požadavek může buďto navázat nové spojení (stejně jak je znázorněno na 5.2.3) nebo může využít již existující spojení. Například u prohlížeče Chromium je ze zdrojového kódu patrné, že aktivní spojení jsou uchovávána v *connection pool*, které mohou být znovu využity k přenosu HTTP komunikace. V takovém případě tedy přicházíme o zprávu *ClientHello*, kterou bychom mohli zachytit na síťovém rozhraní a získat z ní potřebné informace. Řešením takové situace je tedy vytvoření podobné struktury jakou disponuje samotný prohlížeč, ve které si budeme uchovávat záznamy o aktivních spojení. Při sledování HTTP komunikace pak můžeme k jednotlivým zprávám přiřazovat všechna spojení, která byla v době vyřizování požadavku aktivní a naopak k jednotlivým spojením můžeme přiřazovat všechny zprávy, které mohly být spojením přeneseny. Bohužel se tedy nejedná o párování 1:1, které by bylo tím nejideálnějším případem, ale zároveň se jedná o vylepšení stávajícího nedostatku.

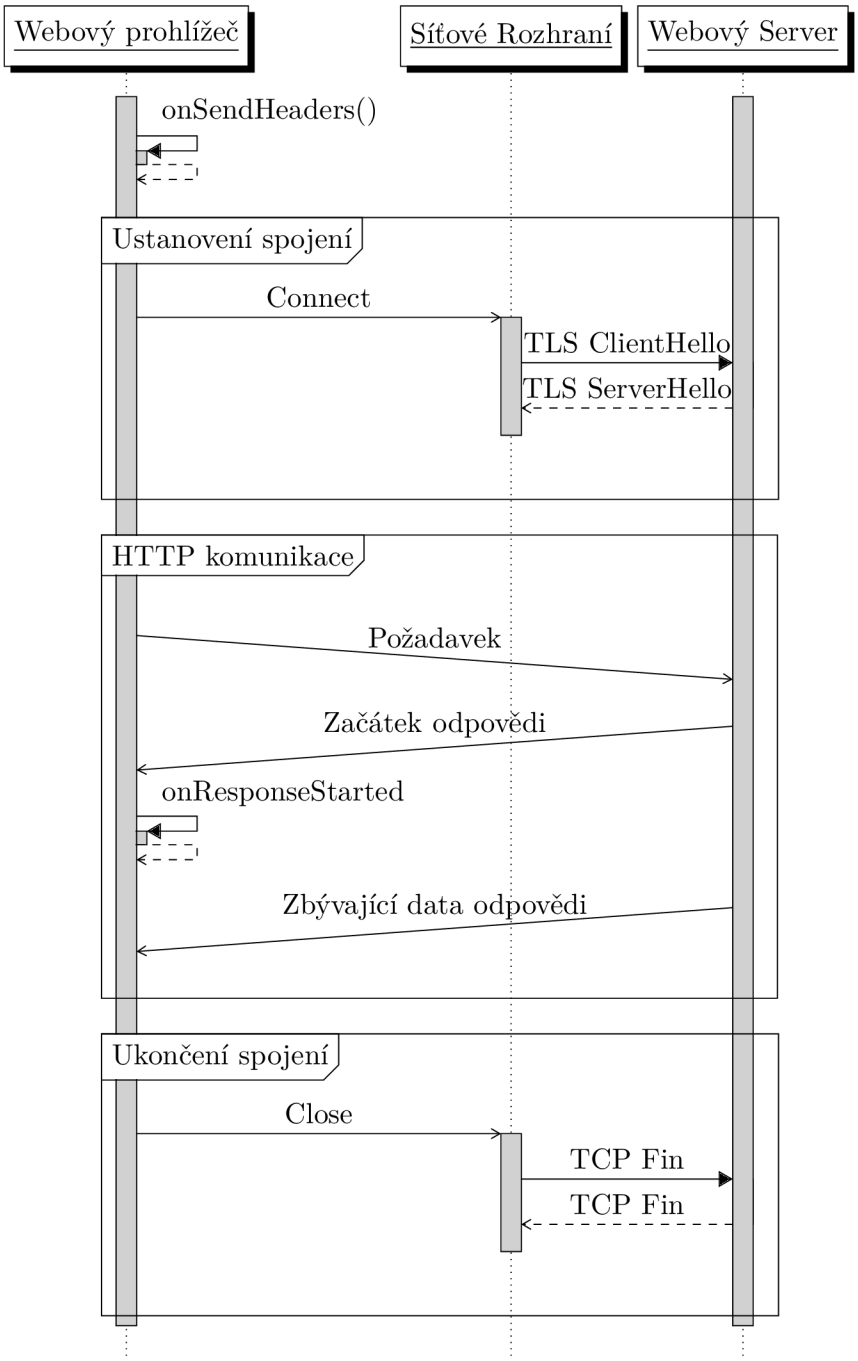
Jak jsem se již zmínil dříve v této sekci, stejný způsob párování je použit i v případě protokolu HTTP, pro který není tento typ párování nejvíce vhodný, ale jak již bylo poznamenáno, síťová sonda má přístup k samotným datům, která byla skrze spojení přenášena a proto nebyl pro nešifrovaný protokol HTTP navrhnut další, odlišný styl párování.

Joiner

K vykonání zmíněného párování slouží **Joiner**, kam Interface Monitor i Browser Extension zasílají své zprávy, obsahující potřebné informace k párování. I přesto že události, které jsou impulsem k odeslání zpráv jsou vyvolány v postupném pořadí, nedá se zcela spoléhat že zprávy budou ve stejném pořadí i přijaty párovacím procesem. Kupříkladu uvažujme zpoždění vyvolané nenadálou aktivitou operačního systému, kvůli kterému by byla zpráva od Browser Extension přijata dříve, než zpráva od Interface Monitoru, přičemž vyvolány byly v opačném pořadí. K řešení tohoto problému je zavedení alespoň základní synchronizace, která může být v případě navrhovaného nástroje pomocí časových značek.

O značení zpráv časovou informací se nástroj nemusí starat, jelikož Google API poskytuje metadata požadavků již označená a stejně tak při sledování průchozích paketů na síťovém rozhraní je nám tato informace k dispozici. Pokud porovnáme tyto časové značky, tak můžeme určit správné pořadí zpráv.

Vraťme se ale zpět k problému se záměnou pořadí zpráv. Pokud bychom data sbírali k odloženému zkoumání, bylo by seřazení zpráv podle časových značek triviální záležitostí. Pokud však chceme poskytovat informace v reálném čase, je zapotřebí navrhnout takovou synchronizaci, která by bez zbytečně dlouhých prodlev zajistila správné pořadí událostí. Při zkoumání, jak častý je výskyt špatných posloupností událostí bez použití synchronizace jsem velmi často narážel na „předbíhání“ zpráv od Browser Extension zprávami z Interface monitoru, což by z pohledu párování jednoho požadavku nebyl problém. Při více požadavcích a více spojeních však bylo patrné, že se požadavku z prohlížeče v situaci vícenásobného



Obrázek 5.2: Příklad sekvence událostí, při které klient navazuje nové spojení a po konci HTTP komunikace jej uzavírá.

spojení se serverem přiřazují taková spojení, která započala až po jeho vyřízení. Tento jev je pro anotační systém nežádoucí, jelikož by pak anotovaná data neodpovídala realitě.

Navrženým řešením je synchronizace pomocí fronty zpráv od Interface Monitoru. Joiner tyto zprávy přijímá, avšak nezpracovává ihned, pouze si je ukládá v pořadí, v jakém je přijal, což odpovídá pořadí zachycených paketů. Vyprazdňování fronty započne až v případě přijetí zprávy od Browser Extension. V takovém případě se nejprve porovná časová značka příchozí zprávy s časovou značkou zprávy na začátku fronty a začne se zpracovávat ta zpráva, která dle tohoto porovnání má následovat. Nedostatkem navrženého mechanizmu je náchylnost na opoždění zpráv z Interface Monitoru, což by muselo být řešeno čekáním při přijetí zprávy od Browser Extension. Jelikož jsem však při zkoumání dopadu tohoto nedostatku zjistil, že k takové situaci nenastává ve větší míře, rozhodl jsem se upřednostnit rychlejší dostupnost dat před variantou s čekáním.

Dalším prvkem potřebným pro párování, který již byl nastíněn v předchozí sekci, je sledování aktivních TCP spojení, která souvisí s HTTP a TLS provozem. To probíhá právě na základě zpráv od Interface Monitoru, které jsou vyhodnocovány následovně (význam zpráv neodpovídá přesnému významu zpráv v protokolu TCP, ale jedná se o jejich interpretaci):

- Zpráva o zachyceném *ClientHello* má význam počátku nového spojení.
- Zpráva o zachycené TCP FIN zprávě má význam, že jeden z účastníků spojení ukončil odesílání dat.
- Zpráva o zachycené TCP RST znamená okamžitý konec spojení.

Joiner si vnitřně udržuje vyhledávací strukturu tak, aby při příchozí zprávě od Browser Extension mohl korektně poskytnout informace o aktivních spojeních. Je třeba ještě poznamenat způsob zpracování TCP FIN zpráv. U nich nelze reagovat pouze na jedinou zprávu, ve které se objeví příznak FIN, jelikož tím oznamuje pouze jedna strana úmysl o ukončení spojení[7]. Pro správné označení spojení jako ukončeného je tedy potřeba sledovat zprávy FIN zprávy od obou odesilatelů. V

A v neposlední řadě má Joiner za úkol provádět zmíněné párování a spárovaná data ukládat do vnitřních struktur k pozdějšímu zpracování. V aktuálním návrhu uvažujeme dva typy úložiště. Prvním je operační paměť, ve které se uchovává kompletní seznam všech dat sesbíraných Joinerem a která slouží jako zdroj dat pro exportní proces Dispatcher. Druhým úložištěm je externí soubor, do kterého se ukládají informace o napárovaných datech, jejichž náležité spojení se serverem bylo již kompletně ukončeno. Tento soubor slouží pro zpětné zkoumání po ukončení nástroje.

Dispatcher

Výstupní součástí nástroje na uživatelském zařízení je **Dispatcher**. Ten se stará o obsluhu požadavků od ostatních procesů, kterým naslouchá pomocí komunikačního síťového socketu. Příchozí požadavky jsou ve formátu *JSON*, stejně jako odchozí odpovědi. Požadavek musí obsahovat 5 povinných informací k úspěšnému vyhledání požadovaných informací:

1. Zdrojová IP adresa toku
2. Zdrojový port toku
3. Cílová IP adresa toku

4. Cílový port toku

5. Časová značka, která označuje jakýkoli čas aktivity toku.

Po přijetí správně zformátovaného požadavku vyhledává Dispatcher ve vnitřních strukturách párovacího procesu a snaží se získat data ke konkrétnímu datovému toku. V případě aktuálního návrhu však nelze vyhledávat pouze pomocí IP adres a portů, jelikož může nastat situace, kdy port může být využit několikrát pro spojení se stejnými parametry, avšak v jiné časové období. Proto je posledním potřebným parametrem časová značka, která může označovat jakýkoli moment, který spadá do období, během kterého byl port přiřazen pouze jednomu spojení.

Výsledkem vyhledávání je pole obsahující metadata o HTTP komunikaci, ke které bylo spojení využito. Při interpretaci získaných dat je však dávat pozor na nesprávnou implikaci, že by všechny požadavky a odpovědi, které náleží získaným metadatům, byly přeneseny pouze pomocí jednoho spojení, ke kterému byly záznamy získány. Taková implikace může být pravdivá pouze v situaci, kdy nebyla navázána žádná další spojení s webovým serverem. V opačném případě musíme brát v potaz i ostatní spojení.

5.3 IPFIXprobe modul

Jediná část anotačního nástroje, která se nemusí nacházet na zařízení uživatele je přídavný modul sondy *IPFIXprobe*. Činnost modulu spočívá v dotazovaní se na probíhající či ukončené datové toky. K těmto dotazům využívá potřebná data, přičemž volí vhodnou časovou značku, kterou obsáhne v dotazu. Takovou značkou se myslí moment, ve kterém je datový tok s největší pravděpodobností aktivní jak na zařízení uživatele, tak i na místě síťové sondy. Jako méně vhodné se jeví časové značky začátku či konce datového toku. Začáteční časová značka totiž může předbíhat moment, kdy je na zařízení uživatele Interface Monitorem označen datový tok jako aktivní, a to z toho důvodu, že Interface Monitor reaguje až na zprávy ClientHello protokolu TLS Handshake, který funguje na již ustanoveném TCP spojení. Koncová časová značka je nevhodná kvůli zpoždění sítě, kdy může nastat situace, ve které klientské zařízení uživatele potvrzuje ukončení spojení, ale toto potvrzení ještě nedoputovalo k síťové sondě.

Vhodnou časovou značkou může být například střed mezi začátkem a koncem toku v případě ukončeného spojení nebo střed mezi začátkem a aktuálním momentem v případě probíhajícího spojení. Aktuální návrh přídavného modulu počítá s dotazováním se na ukončené datové toky, a to především kvůli tomu, že se v takových tocích neobjeví další záznamy HTTP komunikace, na které by se modul musel znovu dotazovat.

Kapitola 6

Implementace nástrojů

Zvoleným prostředím pro zprovoznění implementovaného anotačního systému se staly operační systémy založené na jádře UNIX. Tím anotační systém pokrývá nejen různé distribuce Linuxu, jako například Ubuntu, ale i operační systém macOS. Jediným nekompatibilním operačním systémem je Microsoft Windows, a to z důvodu potřeby větších úprav přístupu k souborům a procesům, které mají UNIXové systémy společné. Zároveň se však ani budoucí kompatibilita nevylučuje, a to díky použití sestavovacího systému *CMake*, jenž je použitelný na všech zmíněných operačních systémech a tím umožňuje multiplatformní vývoj.

Zvoleným jazykem pro implementaci většiny nástrojů se stal jazyk C++. Důvodů k této volbě bylo několik. Hlavními dvěma důvody byla rychlost běhu výsledných binárních souborů a také kompatibilita při implementaci přídavného modulu sondy *IPFIXprobe*. Jediným nástrojem, který nemohl být v jazyce C++ implementovaný je rozšíření webového prohlížeče Browser Extension, a to z důvodu kompatibility s Extension API.

Většina nástrojů je implementována jako samostatně běžící proces (s výjimkou komponenty Dispatcher, který běží jako součást Joineru). Důvodem této volby byla jednodušší správa jednotlivých komponent na různých systémech. Pokud bychom například měli na nějaké z Linuxových distribucí problém s funkcemi pro monitorování síťového rozhraní, není potřeba měnit celý anotační systém, ale stačí pouze upravit a vyměnit komponentu Interface Monitor. Tím získáváme možnost vytvářet různé kombinace upravených nástrojů pro širší pokrytí kompatibilních systémů. Pro budoucí vývoj anotačního systému je výhodou, že v případě změny požadavků na jeden z nástrojů můžeme též vyměnit jednu z komponent a ne celý anotační systém.

6.1 Zařízení uživatele

Implementovaná architektura se od návrhu liší v jednom bodě, a to v komunikaci mezi Browser Extension a Joinerem. Aktuálně existují dvě možnosti k realizaci této komunikace. První je *webSocket* API, při kterém si rozšíření prohlížeče vytváří komunikační socket, přes který komunikuje buď s lokálními nebo i se vzdálenými hosty. Jelikož je možnost vzdálené komunikace redundantní v aktuálním návrhu, přistoupil jsem k druhé dostupné metodě, která se nazývá *Native Messaging*. Při použití tohoto stylu komunikace je zapotřebí prohlí-

žeči poskytnout cestu k binárnímu souboru, který prohlížeč spustí a na vstup mu předkládá data odeslaná z rozšíření prohlížeče. V případě zápisu na výstup binárním souborem interpretuje prohlížeč zapsaná data jako odpověď v této komunikaci.

Jelikož se prohlížeč sám stará o spuštění binárního souboru, tak nemůžeme jakožto příjemce zpráv uvést přímo proces komponenty Joiner. Znamenalo by to totiž, že nasbíraná data by byla dostupná pouze v případě že je spuštěný prohlížeč, což je v aktuálním návrhu nežádoucí efekt. K řešení tohoto problému je tedy zapotřebí vytvoření prostředníka mezi rozšířením prohlížeče a Joinerem, který by pouze interpretoval příchozí zprávy od rozšíření Joineru. Tento proces je nazván HttpDataReSender a k takovému účelu slouží.

6.1.1 Browser Extension

Jediná část projektu implementována v jazyce Javascript. Využívá browser extension API od Google, díky kterému se stává použitelným na všech prohlížečích založených na jádru Chromium. Základem takového rozšíření je soubor `manifest.json`, který popisuje dané rozšíření prohlížeče, specifikuje používané soubory nebo určuje přístupová práva k různým funkcím prohlížeče, o kterých je uživatel informován při instalaci rozšíření. Aktuálně nejpoužívanější verzí souboru `manifest.json` je verze *v2*, která se během vypracovávání této práce začala stávat zastaralou, kdy ji začala nahrazovat verze *v3*. Hlavními rozdíly je nový přístup ke zdrojovým souborům rozšíření prohlížeče a k celkové bezpečnosti. Nově se tak místo skriptů běžících v pozadí (angl. *background scripts*) vytvářejí tzv. *service workers*, které reagují na události v prohlížeči. Bezpečnostním omezením nové verze je přístup k menšímu rozsahu dat, kdy rozšíření prohlížeče už například nemá přístup k výslednému DOM aktuální stránky a ani k datům HTTP požadavků a odpovědí. Nutno podotknouti, že celé API prochází neustálými změnami, které jsou většinou reakcemi na podněty od vývojářů používající toto API.

Implementovaný *Service Worker* tedy při svém spuštění registruje příslušné funkce k událostem *onSendHeaders* a *onResponseStarted*, při kterých provádí sběr dat a jejich odeslání tak, jak již bylo popsáno v návrhu. Pomocí *Native Messaging* odesílá nasbíraná data ve formátu JSON procesu HttpDataReSender.

Prvotní sběr dat probíhá při události **onSendHeaders**, při které jsou webovému rozšíření dostupná data požadavku. Mezi těmito daty se nachází `requestId`, které odpovídá pořadí požadavku v rámci běhu aktuální instance prohlížeče. Nemůže se tedy stát, že by dva různé požadavky měly stejné `requestId`. I přesto, že se jedná o zdánlivě číselný identifikátor, u kterého by mohlo dojít k přetečení, tak je `requestId` poskytováno jako textový řetězec. Příchozí odpověď má poté stejné `requestId` jako požadavek, na který je odpovězeno, takže tento identifikátor lze využít k párování požadavků a odpovědí, k čemuž také dochází v rámci prohlížeče. Při vyvolání události dochází k ukládání poskytnutých dat požadavku do asociativního pole, kde klíčem je zmiňované `requestId`.

Druhá část sběru dat probíhá při události **onResponseStarted**, kdy jsou k dispozici data odpovědi. Při této události dochází nejdříve ke kontrole, zda-li požadavek nepochází z datové cache prohlížeče a pokud ne, tak dochází k extrakci dat z asociativního pole na základě `requestId` a vytvoření datové zprávy pro Joiner. Hlavními informacemi v této

datové zprávě jsou: IP adresa a port webového serveru, hostname, časová značka vyvolání události a poskytnutá data požadavku a odpovědi.

6.1.2 HttpDataReSender

Jedná se již o proces v prostředí operačního systému, který slouží k interpretaci zpráv od Browser Extension. Zprávy přijímá jako data na standardním vstupu `stdin`, přičemž má i možnost odpovídat zapsáním dat na standardní výstup `stdout`, kterou však nevyužívá. Při použití *Native Messaging* je využito jednoduchého komunikačního protokolu pro interpretaci dat, který určuje, že v prvních 4 bajtech je zapsána celková velikost odesílané zprávy a po jejím přečtení je zapsána celá zpráva. Stejný komunikační protokol je použitý i při přenosu zpráv mezi ostatními procesy, jak si popíšeme dále.

Jakmile HttpDataReSender obdrží zprávu od Browser Extension, interpretuje tyto zprávy z formátu JSON do zpráv *Protocol Buffers*, které si popíšeme v podsekci 6.1.3. Výslednou zprávu zapisuje do svého textového logu a odesílá procesu Joiner ke zpracování.

6.1.3 Meziprocesová komunikace

Jako mezi procesovou komunikaci jsem zvolil zasílání zpráv přes UNIX doménové sockety. Jejich využití je oproti síťovým socketům bezpečnější, protože se nachází pouze lokálně v systému a nejsou exponovány pro připojení přes počítačovou síť. Zároveň by nebylo složité je v budoucnu nahradit síťovými sockety v případě distribuované architektury a potřeby komunikace s jednotlivými komponenty přes počítačovou síť.

Jako formát zpráv meziprocesové komunikace jsem zvolil *Protocol Buffers* (zkráceně **Protobuf**), jejichž vývojem se rovněž zabývá firma Google. Protocol Buffers se využívají k uchovávání a přenosu dat mezi různými platformami a architekturami, což zajišťuje jejich konzistenci. Pro anotační systém je to opět výhoda v cestě k jeho širšímu využití. V aktuální implementaci existují dva typy zpráv: datová zpráva **HTTPMessage** a monitorovací zpráva **IFMessage**.

Komunikační protokol pro přenos Protobuf zpráv přes UNIX doménové sockety je stejný protokol, který jsem již popsal v komunikaci Browser Extensionu a HttpDataReSender, tedy že formát zprávy je následující:

1. 4 bajty, které obsahují velikost přenášených dat
2. Přenášená data

6.1.4 Interface Monitor

Jedná se o proces běžící v prostředí operačního systému, jehož spuštění má na starosti uživatel. Po spuštění sleduje procházející pakety na aktivním síťovém rozhraní pomocí knihovny *libpcap*. Sledování probíhá v bezprostředním módu, tedy že reaguje na každý požadovaný paket ihned co je k dispozici.

Jelikož má Interface Monitor zájem pouze o určité typy paketů (TLS ClientHello, HTTP, TCP se značkou FIN a TCP se značkou RST), používá k filtrování paketů tzv. BPF filtr

(angl. *Berkeley Packet Filter*). Pomocí něj se procesu dostávají jenom pakety, které projdou definovaným filtrem.

Po zachycení požadovaného paketu dochází k jeho vyhodnocení. Nejdříve se extrahují informace, které jsou pro všechny druhy stejné, jako časová značka záchytu paketu nebo IP adresa a port příjemce a odesilatele z TCP a IP vrstvy. Během zkoumání TCP vrstvy se kontroluje, zda-li jsou nastavené příznaky FIN nebo RST. Pokud ano, jedná se o jednu z požadovaných zpráv TCP vrstvy. Pokud ne, přechází se ke zkoumání přenášených dat, tzv. *payload*.

Nejdříve se rozhodne, zda se jedná o HTTP payload či TLS payload, a to pomocí jednoduchého testu na výskyt slova GET či POST v prvních třech nebo čtyřech bajtech. Pokud ano, jedná se o HTTP payload. V opačném případě zbývá už jen TLS payload. Při parsování HTTP payloadu je důležitá extrakce celkového URL z požadavku. Ta probíhá konkatenací doménového jména dotazovaného serveru z hlavičky `Host` a cesty k požadovanému zdroji z prvního řádku požadavku. Získaná URL se poté vloží spolu s ostatními daty do Protobuf zprávy.

Při parsování TLS mohou nastat dvě situace. V obou případech platí, že v zachycené zprávě se nachází pole rozšíření SNI ve zprávě ClientHello. Obsah tohoto pole však může být i prázdný. V takovém případě se jedná o nežádanou zprávu, a Protobuf zpráva oznamující její výskyt není odeslána Joineru. Pokud však pole SNI obsahuje nějaká data, tak se tyto extrahují a vloží do příslušné Protobuf zprávy, která je určena k odeslání.

Vedlejším efektem aktuální implementace je fakt, že monitorovány jsou všechny TLS ClientHello zprávy, nejen ty odcházející z prohlížeče. Proto jsou Joineru oznamovány i zprávy pocházející z programů jako je například pošta, či online úložiště. Tato nadbytečná data však nijak neovlivňují nasbírané výsledky, což si vysvětlíme při popisu Joineru.

6.1.5 Joiner

Hlavním jádrem anotačního nástroje je Joiner, který přijímá data pocházející od Browser Extension, tak od Interface monitoru a tato se snaží deterministicky párovat a ukládat pro jejich pozdější export. Na naslouchání nových zpráv od obou procesů používá separátní vlákna, přičemž zprávy od Interface Monitoru ukládá do fronty ke zpracování, kdežto zprávy od Browser Extension zpracovává ihned, a to na základě navrženého párovacího procesu. Obě dvě naslouchací přistupují ke frontě zpráv ke zpracování, proto je k ní zřízen výlučný přístup pomocí prvku *mutex*.

Důležitou součástí Joineru pro párování TLS datových toků je tzv. *Active Connection Pool*, což je struktura pro sledování aktivních TLS spojení. Impulsem k zavedení této struktury bylo zkoumání samotného prohlížeče Google Chrome, který si podobně udržuje přehled o aktivních spojeních, které znovu využívá při zasílání požadavků, které tyto spojení vyžadují. Struktura Active Conection Pool je implementována jako asociativní pole následovně:

- Klíčem pro vyhledávání v v celém Pool je IP adresa a port vzdáleného serveru
- Hodnotou klíče je asociativní pole obsahující jména serverů patřící pod stejnou IP adresu a port:
  - Klíčem je jmenný identifikátor serveru SNI

– Hodnotou je pole aktivních lokálních socketů, které jsou identifikovány IP adresou a portem.

Ke přidáni záznamu do Active Connection Pool dojde při přijetí TLS ClientHello zprávy od Interface Monitoru, která obsahuje všechny potřebné údaje (včetně identifikátoru SNI). K vymazání záznamu může dojít v následujících případech:

1. Příchozí zpráva od Interface Monitoru oznamující výskyt TCP FIN zprávy v daném spojení. V tomto případě je u každého záznamu lokálního socketu ještě veden údaj, zda-li FIN zprávu odeslal klient nebo host. K vymazání záznamu dochází až ve chvíli kdy ji zaslali oba. Při vyhledávání správného spojení je zapotřebí projít všechny záznamy o socketech, které jsou vedeny pod klíčem celého Pool. Je to z toho důvodu, že v TCP zprávě není žádný jmenný identifikátor serveru, tedy nezbývá než projít všechny možnosti.

2. Příchozí zpráva od Interface Monitoru oznamující výskyt TCP RST zprávy v daném spojení. Takový případ nastává při okamžitém ukončení TCP spojení, tedy i záznam je z Active Connection Pool ihned odstraněn. Opět je zapotřebí projít všechny záznamy patřící pod stejnou IP adresu a port hosta v Active Connection Pool. Opět totiž není k dispozici identifikátor SNI.

K vyhledávání dochází pří zpracování zprávy od Browser Extension, respektive HttpDataReSenderu. Při párování je k dispozici IP adresa a port serveru, se kterým probíhala komunikace z browseru (součást webRequest API) a také identifikátor SNI, což je hostname daného serveru. Při úspěšném vyhledání záznamu podle těchto informací dostáváme seznam lokálních socketů (spojení), které jsou v dané chvíli aktivní.

Způsob ukládání dat je realizován pomocí obousměrně vázáného seznamu. Záznamy v tomto seznamu jsou propojeny přes reference s Active Connection Pool, takže při práci se záznamem aktivního spojení je přímý přístup k záznamu v tomto listu. Tím se zvýší efektivita vyhledávání, především kvůli tomu, že struktura *std::list* má pomalejší vyhledávání než struktura `std::map`, pomocí které je Active Connection Pool implementovaný. Navíc jsou reference na záznamy ve `std::list` platné po celou dobu běhu programu, oproti ostatním strukturám. Konkrétněji v nástroji existují dva seznamy, a to seznam všech spojení se servery, který obsahuje metadata HTTP komunikace a dále seznam všech identifikátorů datových toků s časovými známkami začátku a konce, který slouží primárně pro Dispatcher, jehož činnost bude popsána dále. Oba dva seznamy jsou vzájemně propojené, tedy záznam v seznamu serverových spojení má referenci na patřičné záznamy datových toků a naopak datový tok má záznam na serverové spojení, ke kterému patří. Přístup k těmto seznamům je opět opatřen výlučným přístupem.

Dispatcher

Součást procesu Joiner, která však běží v samostatném vlákně pro nezávislou obsluhu požadavků na export dat. Dispatcher si na uživatelem definovaném portě otevře socket, na kterém naslouchá pro příchozí spojení. Příchozí zpráva se opět řídí komunikačním protokolem popsaném v 6.1.3, ale liší se tím, že dotaz má formát JSON. Dotaz musí obsahovat tato pole:

- `srcIP`, `srcPort` označující klientovu stranu spojení
- `dstIP`, `dstPort` označující vzdálenou stranu spojení
- `timeStamp` označující časový moment, podle kterého se vyhledávají náležitá spojení

Přijetím korektního dotazu započíná Dispatcher svou úlohu, tedy nejprve vyhledává v seznamu identifikátor datového toku, který odpovídá tomu v požadavku, a jehož aktivní doba obsahuje časový moment v požadavku. Při nalezení tohoto záznamu přistupuje přes referenci do seznamu serverových spojení, ze kterého si získá všechny záznamy o HTTP komunikaci, která probíhala během aktivity daného spojení. Z těchto záznamů vytvoří Dispatcher zprávu ve formátu JSON a odesílá zpět jako odpověď. V případě špatného formátu požadavku odesílá chybovou hlášku a v případě nenalezeného záznamu odesílá prázdnou zprávu.

```
{
  "dstIp": "185.33.220.240",
  "dstPort": 443,
  "srcIp": "192.168.0.199",
  "srcPort": 59513,
  "timestamp": 1652478515177790
}
```

Výpis 6.1: Příklad požadavku na export nasbíraných dat.

V JSON odpovědi od Dispatcheru se nachází informace o jmenném názvu serveru (SNI) a dvě pole. První z nich obsahuje seznam lokálních socketů, které byly využity v komunikaci s daným serverem. U záznamů v tomto poli je specifikován začátek a konec spojení a informace k identifikaci datového toku. Druhé pole obsahuje seznam HTTP dat, kde každý záznam v poli obsahuje část s metadaty požadavku a patřičnou odpovědí.

6.1.6 Konfigurace

Jednotlivé procesy mohou být konfigurovány pomocí konfiguračního souboru ve formátu JSON. To umožňuje různé nastavení modulů pro běh v odlišných operačních systémech. Příkladem takového konfiguračního souboru je příloha B. V té můžeme vidět, že je soubor rozdělen na tyto části:

- **Společná část** obsahující pouze informaci o nastavení cesty, kam budou jednotlivé procesy ukládat své logy.
- **Specifická část** pro každý z procesů.
  - **HttpDataReSender** obsahuje nastavení cesty k souboru doménového socketu pro komunikaci s Joinerem.
  - **InterfaceMonitor** též obsahuje nastavení cesty k souboru doménového socketu.

– **Joiner** obsahuje nastavení IP adresy a portu, na kterém bude naslouchat novým požadavkům pro export dat. A dále také obsahuje nastavení cesty k souboru, do kterého bude ukládat záznamy anotovaných toků, které již skončily.

6.2 IPFIXprobe modul

Implementace přídavného modulu je provedena v jazyce C++, a to kvůli implementaci sondy v tomto jazyce. Aktuální architektura sondy umožňuje jednoduše přidávat přídavné moduly a komunikovat s nimi pomocí předem definovaných funkcí, které musí moduly implementovat. Z návrhu přídavného modulu pro tento anotační systém vyplývá, že hlavním úkolem je zaslání požadavku na poskytnutí dat ohledně datového toku po jeho ukončení. Hlavní úkony modulu tedy jsou implementovány ve funkci `pre_export`. Zmíněnými úkony je pouhé odeslání požadavku směrem k Dispatcheru, vyhodnocení odpovědi a případné přidání získaných dat do záznamu datového toku.

Pro účel této práce je aktivita přídavného modulu o něco chudší, a to o krok přidání dat k záznamu síťového toku. Výsledná podoba potřebných dat je úzce spjata s procesem vyhodnocování užitečnosti informací ke klasifikaci, který však není součástí této práce. Zároveň je neefektivní přidávat všechna získaná data k záznamu datového toku z toho důvodu, že by se výsledný záznam několikrát zvětšil na své velikosti a s tím by vznikaly další problémy s přenášením dat a ukládáním na disk. Aktuální implementace tedy obsahuje potřebný mechanizmus pro dotazování a zpracování dat, ale dále nechává prostor pro implementaci extrakce a uložení dat k záznamům datových toků.

6.3 Budoucí vývoj

Výsledná implementace trpí několika nedostatky, které dávají prostor k budoucím rozšířením anotačního systému. Tím hlavním nedostatkem je podpora operačního systému Windows, díky které by se dal anotační systém využít na větším počtu zařízení. Při implementaci byl tento nedostatek uvažován a proto jsou některé jeho části navrženy tak, aby případná úprava nástrojů byla možná a co nejméně závislá na externích knihovnách. Dalším kompatibilním nedostatkem je aktuální použitelnost pouze na prohlížeči Google Chrome nebo prohlížečích vytvořených přímo z projektu Chromium. Tato práce bohužel vznikala v období, při kterém začala firma Google měnit své požadavky na bezpečnost a soukromí uživatelů používajících rozšíření prohlížečů a podle toho představila nový formát rozšíření prohlížečů, který ještě nebyl podporován prohlížeči jako je Mozilla Firefox nebo Safari. V návrhu je tato skutečnost uvažována a zvolen byl právě onen nový formát rozšíření prohlížeče. Podpora aktuálně nekompatibilních prohlížečů tedy nejvíce závisí na firmách, které tyto prohlížeče spravují.

Kapitola 7

Spuštění a testování

Spuštění anotačního systému probíhá v několika krocích, jejichž posloupnost je třeba dodržet ke správné funkcionalitě celého systému a také k co největší přesnosti nasbíraných dat. Kroky jsou následující:

1. Jako prvním procese, který je potřeba spustit je proces Joineru. Tento vytvoří potřebné komunikační sockety jak pro meziprocesovou komunikaci, tak pro funkcionalitu Dispatcheru. V případě různých omezení (nedostatečná práva uživatele, nedostupnost portu z důvodu jeho využití jiným procesem) tuto skutečnost nahlásí uživateli, který podle něj může upravit konfiguraci systému.

2. Dalším doporučeným krokem je spuštění Interface Monitor, a to pokud možno ještě před samotným spuštěním webového prohlížeče. Tento krok je doporučený z toho důvodu, že při otevření prohlížeče mohou vznikat nová spojení (opakované načtení stránek z předchozí relace), která by pak mohla být využita k přenosu HTTP komunikace.

3. Dalším doporučeným nástrojem v pořadí je IPFIXprobe s modulem pro anotaci webového provozu. Tento krok by mohl být zaměněn s druhým krokem, ovšem v tomto pořadí máme opět jistotu, že monitorované datové toky sondou již mohou mít záznam v Joineru.

4. Posledním krokem je načtení a spuštění rozšíření prohlížeče Browser Extension. Po této akci se rozšíření prohlížeče připojí k Joineru a začne mu zasílat metadata o probíhající HTTP komunikaci. V tomto bodě máme jistotu, že probíhající komunikace je zachycena jak Interface Monitorem, tak sondou IPFIXprobe. V případě, že se na zařízení uživatele nachází více webových prohlížečů, je zapotřebí tento krok opakovat pro každý prohlížeč, abychom zajistili co největší pokrytí webového provozu.

Po zavedení a spuštění anotačního systému lze dále zařízení používat k běžným úkonům, přičemž se v reálném čase vytváří anotovaná datová sada na zařízení, na kterém je spuštěna sonda IPFIXprobe (může se jednat o stejné zařízení jako to, kde je spuštěn anotační systém).

7.0.1 Kompatibilita a systémové vytížení

Testování přeložitelnosti a použitelnosti anotačního nástroje bylo testováno v následujících prostředích:

- Zařízení s operačním systémem macOS 12.3.1 s 10 jádrovým procesorem Apple M1 Pro, architektura ARM64 a operační paměť 32 GB.
- Virtuální operační systém Ubuntu 20.04 se 2 jádry, architektura ARM64 a operační paměť 4 GB
- Virtuální operační systém Ubuntu 20.04 se 4 jádry, architektura x86_64 a operační paměť 8 GB.

V každém zmíněném prostředí platilo, že anotační nástroje určené pro běh na zařízení uživatele lze přeložit a spustit. To však přímo neplatilo pro sondu IPFIXprobe, u které se vyskytly problémy s přeložitelností v prvním případě, tedy v operačním systému macOS s použitou architekturou ARM64. Pro využití anotačního systému pro tento případ je tedy zapotřebí spustit sondu IPFIXprobe mimo zařízení uživatele.
V druhém případě se opět vyskytly potíže s použitou architekturou ARM64, ale tentokrát to byla nekompatibilita webového prohlížeče Google Chrome, který pro kombinaci Linuxu s ARM64 není nabízen. Pokus o zprovoznění systému s prohlížečem Chromium, který je dostupný pro tuto kombinaci, však neskončil úspěšně. Ukázalo se, že prohlížeč Chromium měl v takovém prostředí problém se spuštěním přeposílacího procesu HttpDataReSender kvůli použitým knihovnám, jejichž instalaci nedokázal naleznout. V druhém případě se tedy jedná o nemožnost použití anotačního systému, a to z důvodu závislosti na plně funkčním jádru Chromium.
V posledním případě se úspěšné podařilo jak přeložit všechny části anotačního systému, tak i jejich spuštění. Zde se tedy jedná o plnou podporu celého anotačního systému.

7.0.2 Ovlivnění webového provozu

I přesto, že navržený anotační systém není přímou součástí komunikačního řetězce, může jeho aktivita na zařízení uživatele ovlivňovat nasbíraná data tím, že při aktivitě zatěžuje procesor a operační paměť a tím zpožďuje i zpracování požadavků a načítání webových stránek. Pro ověření, že implementace netrpí těmito vlastnostmi jsem provedl test, při kterém jsem využil webovou stránku www.httpvshttps.com. Tato stránka slouží primárně k porovnání rychlosti při použití HTTP a HTTPS, přičemž měří čas, za který bylo přeneseno 360 ikon o celkové velikosti 0,62 MB bez využití vnitřní cache prohlížeče. Pro tento test bylo testovací zařízení připojeno rychlostí 1 Gb/s pomocí ethernetového kabelu pro vytvoření co nejstabilnějšího spojení.

Celkový test tedy probíhal tak, že jsem celou proceduru nahrávání a měření opakoval 20x pro spojení pomocí HTTP a HTTPS, a to za běhu anotačního systému a bez něj. Výsledky měření jsou zaznamenány v tabulce 7.1.

Z naměřených hodnot je patrné, že k žádnému výraznému vyvolání zpoždění nedochází. Průměrná hodnota rychlosti přenosu pomocí protokolu HTTP byla při použití anotátoru o $0,3\%$ menší než bez jeho použití. Zatímco průměrná hodnota při použití HTTPS byla o $1,4\%$ vyšší při spuštěném anotátoru. Tyto výsledky se tedy pohybují v přijatelných mezích, o kterých lze tvrdit, že nemají větší vliv na nasbíraná data.

Bez anotačního systému		S anotačním systémem	
HTTP	HTTPS	HTTP	HTTPS
8,333	0,696	8,349	0,691
8,242	0,701	8,241	0,819
8,167	1,519	8,181	1,533
8,282	0,818	8,270	0,829
8,179	0,806	8,179	0,711
8,287	1,614	8,249	1,646
8,225	0,697	8,241	0,707
8,288	0,799	8,067	0,813
8,282	1,622	8,280	1,630
8,311	0,711	8,133	0,705
8,139	0,706	8,106	0,817
8,090	1,525	8,140	1,351
8,091	0,720	8,177	0,813
8,279	1,393	8,202	0,695
8,196	0,818	8,200	0,700
8,215	0,707	8,152	0,706
8,177	1,636	8,309	1,624
8,220	0,696	8,152	0,711
8,233	0,699	8,181	0,710
8,167	0,419	8,218	1,372
$\varnothing$ 8,220	$\varnothing$ 0,9651	$\varnothing$ 8,201	$\varnothing$ 0,97915

Tabulka 7.1: Naměřené hodnoty rychlosti (v jednotkách sekund) přenesení 360 ikon.

7.0.3 Míra úspěšnosti anotace

Hlavním údajem pro určení využitelnosti tohoto typu anotačního systému je míra úspěšnosti anotace, tedy k jak velké části síťových toků dokáže anotační systém poskytnout přídavné informace k jeho následné klasifikaci. Míru úspěšnosti anotace můžeme vyjádřit dvěma způsoby:

1. Počet úspěšně a korektně napárovaných požadavků z celkového počtu všech odchozích požadavků vyvolaných monitorovanými instancemi webového prohlížeče.

2. Počet úspěšně anotovaných datových toků z celkového počtu datových toků, které náleží k monitorovanému zařízení.

Pro zjištění první míry anotace tedy potřebujeme sledovat počet korektně napárovaných HTTP komunikací k síťovým spojením. Při párování totiž může docházet k situacím, které nejsou zcela pokryté párovacím procesem. Zároveň je však párovací proces navržený tak, aby takové případy neoznačil nesprávně, ale místo toho data k žádným síťovým spojením nepřiřazoval. Při testování také bylo zjištěno, že míru úspěšné anotace ovlivňuje použití nového protokolu QUIC, který se jeví jako nástupce protokolu TLS. Při použití tohoto protokolu nemáme možnost z poskytnutých metadat od prohlížeče zjistit, že daná HTTP komunikace tento protokol používala. Metadata jsou tedy stejně jako ostatní komunikace odeslány k párování, kde však pro ně není nalezené žádné monitorované spojení. Podpora

protokolu QUIC nebyla v aktuálním návrhu ani implementaci brána v potaz, a to především z důvodu existence přídavného modulu do sondy IPFIXprobe, který slouží ke čtení dané komunikace.

Druhá míra úspěšnosti anotace vychází ze sledování všech datových toků monitorovaného zařízení, aby mohlo být určeno, jaké procento z těchto toků tvoří webový provoz, ke kterému může anotační systém poskytnout informace. Tato míra je poté hlavním ukazatelem toho, zda-li má smysl daný typ anotace uplatňovat či nikoli.

Měření obou údajů je však velmi spojené s aktivitami uživatele na zařízení, jaké stránky navštěvuje a celkově jaký provoz generuje. Pro každého uživatele může být tento údaj odlišný, a to až v řádech desítek procent. Pokud si uvedeme příklad uživatele generující webový provoz, který dokážeme 99 % anotovat, ale zároveň toto množství tvoří pouze 10 % celkového provozu a oproti tomu uživatele, jehož webový provoz tvoří 60 % celkového provozu s mírou úspěšnosti anotace 80 %, tak dojdeme ke dvěma odlišným závěrům, zda-li je vhodné využití takové anotace. Nelze tedy jednoznačně určit údaj, který by obecně vyjadřoval využití tohoto anotačního systému.

Pro účel testování této práce jsem tedy zvolil následující taktiku: Uvažujme uživatele, jehož primární aktivitou na zařízení bude využívání webového prohlížeče k prohlížení stránek, sledování videí či využívání sociálních sítí. U takového uživatele poté určíme míru úspěšnosti anotace pro různé kategorie vybraných webových stránek, které vycházejí jak z žebříčku nejpoužívanějších webových stránek v roce 2021 celosvětově[1], tak především z žebříčku nejnavštěvovanějších webových stránek v České republice, který lze získat díky online datům výzkumného projektu NetMonitor[2]. Zvolené kategorie pro testování této práce poté vycházejí z výstupu výzkumu sociodemografie návštěvníků v z června roku 2017[13]. Zvolené webové stránky ke každé kategorii však vycházejí z aktuálních dat dostupných na stránkách projektu NetMonitor a jejich pořadí je zvoleno náhodně. Seznam vybraných kategorií a vybraných webových stránek se nachází v tabulce 7.2.

Sledování videí, online přenosy videí a TV	iprima.cz, nova.cz, irozhlas.cz, youtube.com, twitch.tv, o2tv.cz, netflix.com, ceskatelevize.cz
Použití vyhledávače, zpravodajství	seznam.cz, novinky.cz, idnes.cz, seznamzpravy.cz, sport.cz, denik.cz, aktualne.cz, reuters.com, bbc.com, yahoo.com, google.com, bing.com
Sociální sítě	facebok.com, twitter.com, reddit.com, linkedin.com, tiktok.com, instagram.com
Online nákupy, E-shopy	bazos.cz, sreality.cz, sbazar.cz, sauto.cz, amazon.com, heureka.cz, ebay.com

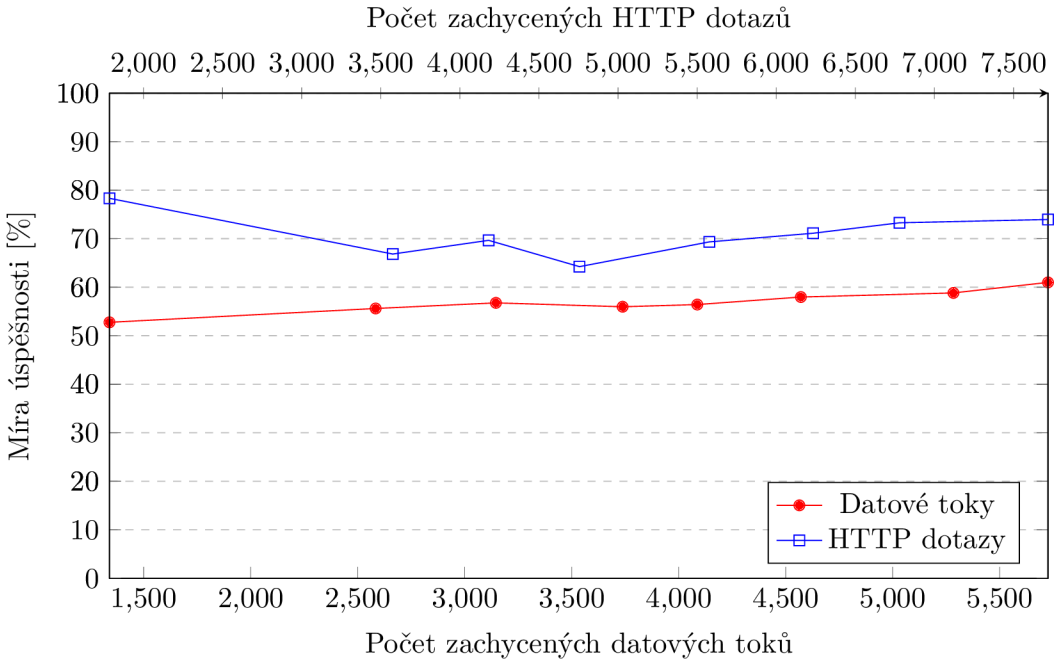
Tabulka 7.2: Vybrané kategorie aktivit na internetu a k nim přiřazené nejnavštěvovanější webové stránky dle aktuálních dat projektu NetMonitor z 1.5.2022.

[1] https://www.statista.com/statistics/1201880/most-visited-websites-worldwide/
[2] https://www.netmonitor.cz

Postup testování každé z kategorií byl následující:

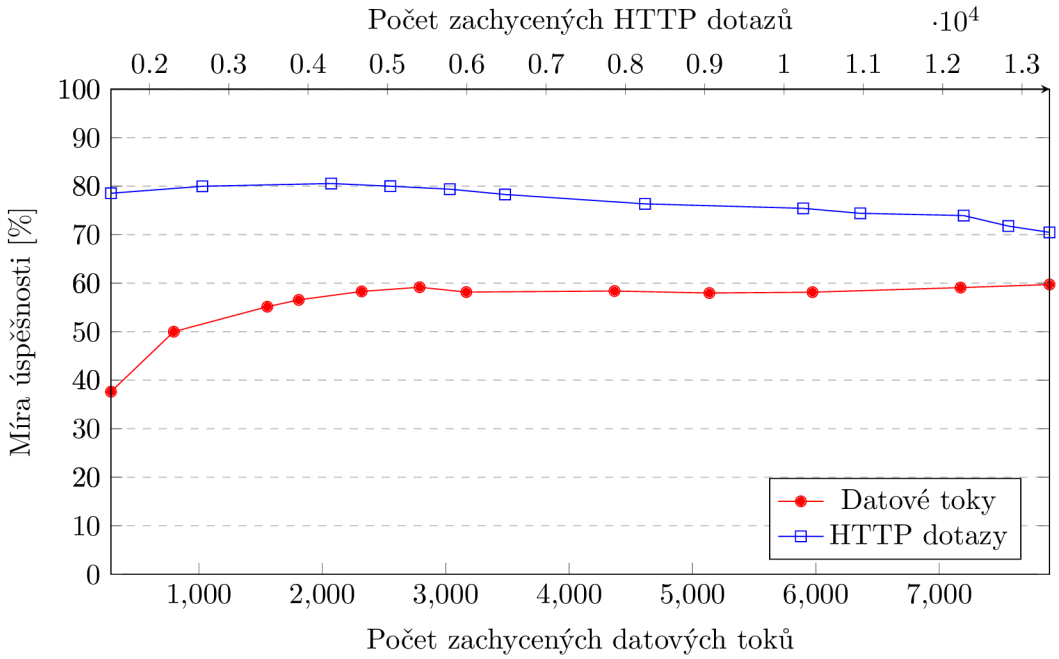
1. Spuštění anotačního systému podle doporučeného postupu pospaného v této práci.
2. Navigace v prohlížeči na stránku, která je v pořadí testovaných stránek podle tabulky 7.2.
3. Navigace po webové stránce (bez otevírání externích odkazů) po náhodně zvolenou časovou dobu.
4. Ukončení interakce se stránkou (zavření panelu v prohlížeči).
5. Zapsání aktuálních údajů míry úspěšnosti.

Výsledky provedeného měření jsou zaneseny do grafů 7.1, 7.2, 7.3 a 7.4, ve kterých jsou znázorněny obě míry úspěšnosti anotace.



Obrázek 7.1: Výsledky měření pro kategorii „Sledování videí, online přenosy videí a TV“

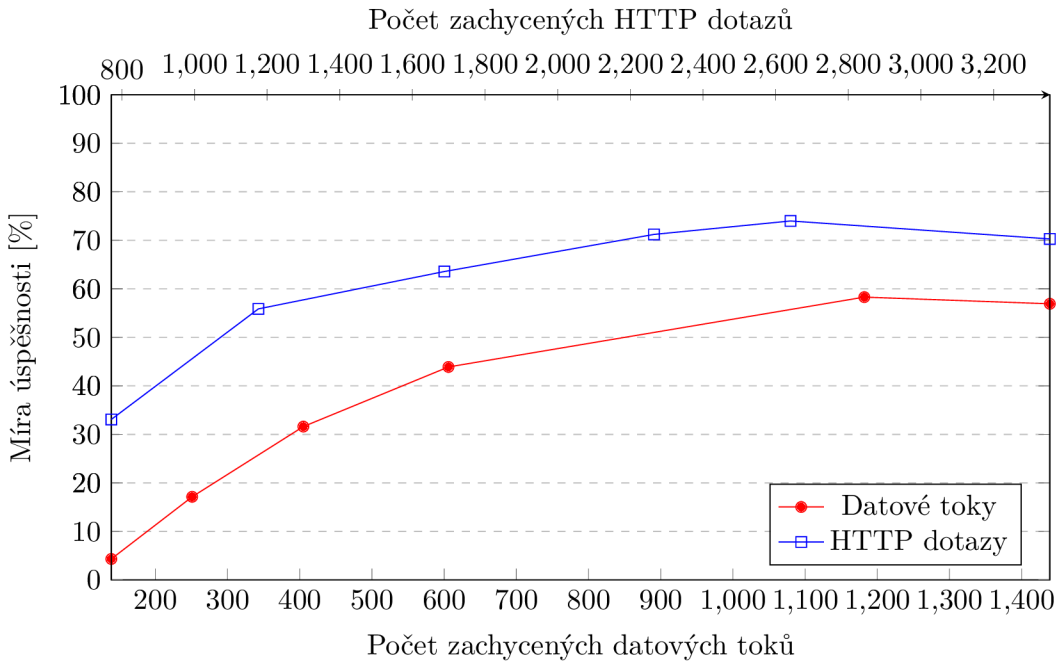
Při měření první kategorie „Sledování videí, online přenosy videí a TV“ byla naměřena míra úspěšné anotace datových toků mezi 50-60 %, která je viditelná z grafu 7.1. Při zkoumání anotačního logu jsem zjistil, že dotazy z prohlížeče se převážně týkají různých elementů na stránce, ale ne samotného videa, o jehož přenosu prohlížeč neinformuje rozšíření prohlížeče pomocí *webRequest* API. Dále jsem zaznamenal, že přenos videí nevyužívá tolik komunikačních portů, jako přenos různých elementů nacházejících se na stránce, což vysvětluje vysokou míru anotace datových toků i přesto, že datové toky s větším obsahem přenášených dat zůstaly bez anotace.



Obrázek 7.2: Výsledky měření pro kategorii „Použití vyhledávače, zpravodajství“

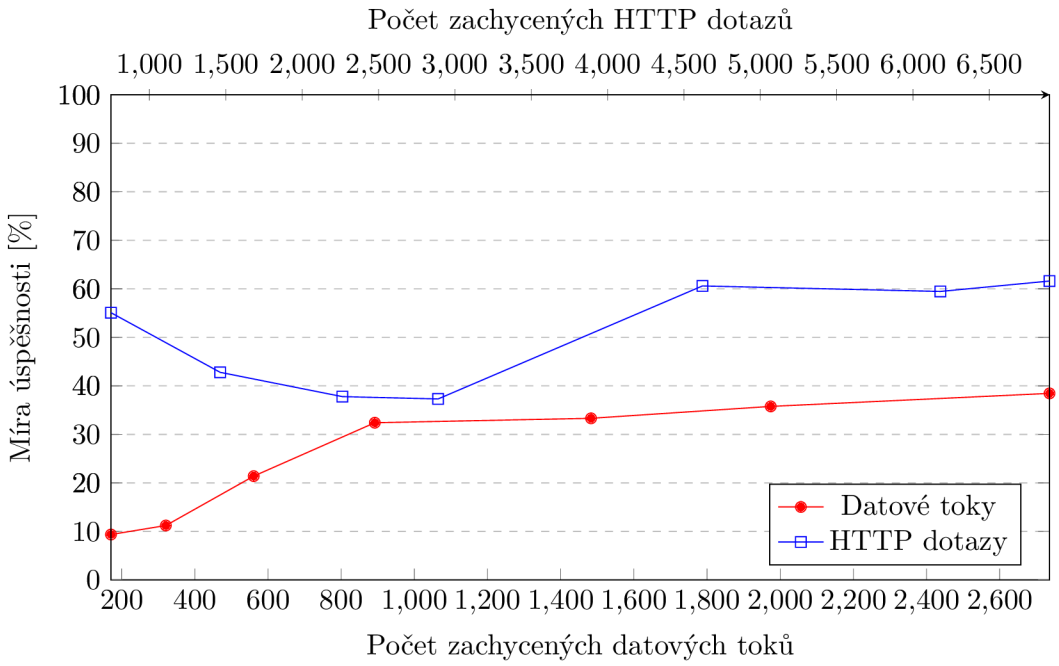
Výsledky měření druhé kategorie „Použití vyhledávače, zpravodajství“ mají podobnou konečnou míru anotace datových toků, která se též pohybuje mezi 50-60 %, jak lze vidět v grafu 7.2. U této kategorie jsem se setkal s vysokým počtem dotazů od každé z navštěvovaných stránek, což může být způsobeno velkou rozmanitostí vyhledávacích webů, které navíc nabízejí vedlejší informace jako zpravodajství nebo reklamní plochu.

Při měření výsledků kategorie „Sociální sítě“, které jsou znázorněné v grafu 7.3, jsem zpočátku narazil na velmi odlišný výsledek naměřených hodnot při návštěvě první stránky, kterou byl facebook.com. Při zkoumání anotačního logu jsem zjistil, že webové stránky facebooku hojně využívají protokol QUIC, který je jednou ze slabin anotačního systému. Při návštěvě dalších sociálních sítí se začaly obě míry zvyšovat až skončily v podobných mezích, jako u předchozích kategorií.



Obrázek 7.3: Výsledky měření pro kategorii „Sociální sítě“

U poslední kategorie „Online nákupy, E-shopy“ se objevilo zajímavé chování, které je vidět v grafu 7.4. Po navštívení první stránky začal míra úspěšnosti anotace webových dotazů klesat, zatímco míra úspěšnosti anotace datových toků stoupala. Po zkoumání anotačního logu jsem zjistil, že toto chování bylo způsobeno jedním důvodem. Tím byla situace, kdy se dotazy na webový server odesílaly přes TLS spojení navázané při použití jiného identifikátoru SNI. Stejné chování jsem zaznamenal i na stránce seznam.cz, kde bylo k zahájení komunikace použito SNI `d32-a.sdn.cz`, ale z pozorovaných dotazů prohlížeče pak bylo patrné, že takové spojení využívají i například dotazy obsahující hostname `d39-a.sdn.cz` (patrné to bylo z toho, že pro takový hostname nebylo otevřeno žádné spojení se stejným SNI za celou dobu načítání stránky a doba vyřízení požadavku spadala do doby aktivity spojení). Tato situace způsobila pokles míry úspěšnosti anotace webových dotazů, protože velké množství dotazů končilo nespárovaných, ale naopak zvyšovalo míru úspěšné anotace datových toků, jelikož k takovýmto datovým toků existovaly záznamy komunikace, ovšem byly nekompletní. Po navigaci na další stránky jsem takové chování nezaznamenal v takové míře, jako tomu bylo u stránek patřící firmě Seznam.



Obrázek 7.4: Výsledky měření pro kategorii „Online nákupy, E-shopy“

U různých stránek napříč všemi kategoriemi byl ještě odhalen jeden společný jev, a to vytváření spojení, kterým však po ukončení nebyly přiřazeny žádné HTTP komunikace. Tento jev nastával na stránkách, které nabízely možnost zasílání notifikací. Taková stránka vytváří TLS spojení pomocí *webSocket* API pro oboustrannou komunikaci se serverem, která však také není dostupná při použití *webRequest* API.

Provedené testování ukázalo, že existují kategorie webového provozu, ve kterých je vhodné nasadit anotaci síťových toků pomocí dat z webového prohlížeče, jelikož by míra takové anotace mohla přesáhnout až 50 % zachyceného provozu. Zároveň však bylo zjištěno, že je aktuálně implementovaný anotátor citlivý na použití protokolu QUIC, který snižuje výslednou míru anotace.

Kontrola korektního párování

Pro ověření správnosti párování byl použit přístup zpětné kontroly, kdy byla při testování nasbírána výsledná data, u kterých poté proběhla kontrola, zda-li dávají syntakticky smysl. Tato kontrola se odehrávala ve dvou krocích. Nejdříve to byla kontrola výsledného souboru s nasbíranými datovými toky, který obsahuje záznamy, ve kterých jsou sdruženy HTTP komunikace se všemi síťovými spojeními, které byly využity k relaci s určitým serverem, který je označen identifikátorem SNI. Kontrola poté proběhla takto:

- Každý ze záznamů HTTP komunikace má stejnou část*hostname*, která odpovídá identifikátoru SNI.
- Každá časová značka ze záznamů HTTP komunikace patří alespoň do jednoho z rozmezí, které jsou určeny dobou trvání síťových spojení, která k záznamu patří.
- IP adresa a port komunikujícího serveru je stejná pro všechny záznamy o spojení.

Pokud by jedna z těchto podmínek nebyla pravdivá. Nemohl by být anotační systém považován za funkční. Jelikož však při testování k žádnému takovému případu nedošlo, lze tvrdit že přiřazená data jsou spárována korektně.

Druhou fází je kontrola dat, která však nebyla spárována vůbec, a to kvůli nenalezení záznamu v Connection Pool. Tyto zprávy proces Joineru ukládá do speciálního *Error* logu, který slouží pro zpětnou kontrolu párovacího procesu. Při testování byl společně s párovacím procesem spuštěn i program *Wireshark*, který je určen pro sledování síťového provozu na rozhraní zařízení. Soubor s chybovými zprávami pak byl zkoumán společně s programem *Wireshark* pro objevení situace, která vyústila k neúspěšnému párování. Díky tomuto přístupu byl například odhalen případ, který je popsán u testování kategorie „Online nákupy, E-shopy“.

Kapitola 8

Závěr

Původním tématem této diplomové práce byla automatická anotace síťového provozu na základě systémových událostí. Při zkoumání existujících řešení byl však objeven větší nedostatek v oblasti anotace jednoho konkrétního typu síťového provozu, čímž je webový provoz. Po konzultaci s vedoucím diplomové práce se tento nedostatek stal hlavním tématem diplomové práce.

V prvních kapitolách diplomová práce popisuje problematiku monitorování síťového provozu, vysvětluje existující způsoby monitorování a popisuje jejich architektury. Hlouběji se však zabývá pasivním monitorováním a využitím nasbíraných síťových toků, při kterých vysvětluje potřebu anotace pro jejich následnou klasifikaci. Ve třetí kapitole se práce dostává k problematice webového provozu a také popisuje stěžejní části získávání informací o tomto typu provozu, na kterých je poté postaven návrh anotačního systému.

Cílem diplomové práce bylo jak navržení, tak i implementace automatického anotačního systému, který by poskytoval informace o webovém provozu, a to v reálném čase i zpětně. Při návrhu byla brána v potaz omezení vycházející z použití protokolů HTTP a TLS a dále i omezení použití *webRequest* API od firmy Google. Kapitola návrhu také popisuje problémy, které formovaly výslednou podobnu navrženého systému.

V kapitole Implementace práce popisuje použité prostředky pro vytvoření navrženého systému a také zde naráží na problémy, které se při implementaci vyskytly a jak byly tyto problémy řešeny. V poslední sekci stejné kapitoly jsou popsány další kroky pro rozvoj anotačního systému pro ještě větší zvýšení jeho použitelnosti..

Závěrečná kapitola popisuje způsob používání a testování implementovaného systému a zjišťuje, zda-li se jedná o přínos k aktuálně existujícím řešením anotace síťového provozu. Ze závěrů měření lze konstatovat, že i přes některé nedostatky anotačního systému se jedná o nástroj, který dokáže poskytnout informace k nemalé části síťového provozu a tím může být nápomocen při jeho klasifikaci. Zároveň se jedná o nástroj, který otevírá novou cestu k informacím poskytnutých webovým prohlížečem pro účel anotace síťových toků.

Literatura

[1] Baste, A., Chu, D., Joseph, D. a Porter, G. Network Management through Packet Annotation. Citeseer. 2005.

[2] Boodman, A. *Extensions Status: On the Runway, Getting Ready for Take-Off.* Září 2009. [Online; navštíveno 20.05.2022]. Dostupné z: https://blog.chromium.org/2009/09/extensions-status-on-runway-getting.html.

[3] Eastlake, D. et al. *Transport layer security (TLS) extensions: Extension definitions.* RFC 6066, January, 2011.

[4] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L. et al. *Hypertext transfer protocol–HTTP/1.1.* RFC 2616, june, 1999.

[5] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R. et al. Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE Communications Surveys Tutorials.* Fourthquarter 2014, sv. 16, č. 4, s. 2037–2064. DOI: 10.1109/COMST.2014.2321898. ISSN 1553-877X.

[6] Huták, L. *Nová generace IPFIX kolektoru.* Brno, CZ, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: http://www.fit.vutbr.cz/study/DP/DP.php?id=20939.

[7] Information Sciences Institute University, S. C. *Transmission Control Protocol.* RFC 793, September, 1981.

[8] Matoušek, P. *Síťové služby a jejich architektura.* Publishing house of Brno University of Technology VUTIUM, 2014. 396 s. ISBN 978-80-214-3766-1. Dostupné z: http://www.fit.vutbr.cz/research/view_pub.php.cs.iso-8859-2?id=10567.

[9] Microsoft. *About Browser Extensions.* Srpen 2017. [Online; navštíveno 20.05.2022]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/aa753620(v=vs.85).

[10] Nucci, A. a Papagiannaki, K. *Design, measurement and management of large-scale IP networks: Bridging the gap between theory and practice.* Cambridge University Press, 2009.

[11] Odvarko, J. *HAR 1.2 Spec* [http://www.softwareishard.com/blog/har-12-spec/]. Online; navštíveno: 2022-05-20.

[12] Rescorla, E. et al. *Http over tls.* RFC 2818, May, 2000.

[13] STEM/MARK a Gemius. *SPIR NetMonitor, Výzkum sociodemografie návštěvníků internetu v České Republice*. Červen 2017. Dostupné z: https://www.spir.cz/sites/default/files/verejne-vystupy/2017_06_TOTAL_PC.pdf.

[14] Williams, N., Zander, S. a Armitage, G. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *ACM SIGCOMM Computer Communication Review*. ACM New York, NY, USA. 2006, sv. 36, č. 5, s. 5–16.

[15] Zúquete, A., Correia, P. a Shamalizadeh, H. Packet tagging system for enhanced traffic profiling. In: IEEE. *2011 IEEE 5th International Conference on Internet Multimedia Systems Architecture and Application*. 2011, s. 1–6.

Příloha A

Obsah přiloženého paměťového média

Návod na instalaci nezbytných balíčků a překlad anotačního systému lze nalézt ve složce `source` v souboru `README.rst`. Složka `tools` obsahuje pomocný skript, který lze využít při ověřování funkčnosti samotného modulu bez potřeby použití sondy IPFIXprobe. Popis souborů a jejich využití lze nalézt ve stejné složce v souboru `README.txt`.

```
Kořenový adresář
├── source/.........Zdrojové kódy společně s návodem pro jejich přeložení
├── tools/..........Pomocný skript, který lze použít pro lokální test
├── doc/............Zdrojové texty diplomové práce
└── thesis.pdf......Text diplomové práce
```

Příloha B

Konfigurační soubor

Příklad konfiguračního souboru pro úspěšné spuštění anotačního systému. Uvedený konfigurační soubor určuje, že se komunikační doménové sockety vytváří ve složce `/tmp`, a historie anotačního systému se ukládá do složky `/tmp/data`. Dále je v konfiguračním souboru specifikována IP adresa a port, na kterém bude Dispatcher naslouchat pro příchozí požadavky, která je v tomto ukázkovém příkladě nastavena na IP adresu `127.0.0.1` a port `50555`.

```
{
  "logFilePath" : "default",
  "InterfaceMonitor":{
    "domainSocketPath" : "/tmp/IFMonitor"
  },
  "HttpDataReSender" : {
    "domainSocketPath" : "/tmp/HttpDataReSender"
  },
  "Joiner" : {
    "dispatcherIp" : "127.0.0.1",
    "dispatcherPort" : 50555,
    "archivePath" : "/tmp/data/archive.json"
  }
}
```