VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ANONYMIZACE PCAP SOUBORŮ

ANONYMIZATION OF PCAP FILES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. PETR NAVRÁTIL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MARTIN HOLKOVIČ**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Navrátil Petr, Bc.**
Program: Informační technologie Obor: Informační systémy
Název: **Anonymizace PCAP souborů**
Anonymization of PCAP Files
Kategorie: Počítačové sítě
Zadání:

1. Nastudujte počítačové sítě a možnosti zpracování síťových dat programem TShark.
2. Popište různé anonymizační algoritmy aplikovatelné pro síťová data ve formě PCAP.
3. Identifikujte různé atributy v síťové komunikaci, které umožňují identifikaci uživatele.
4. Navrhněte různé anonymizační politiky pro síťový provoz a formát pravidel pro aplikaci těchto politik na jednotlivé síťové atributy.
5. Navrhněte a implementujte systém, který podle pravidel z předchozího bodu bude anonymizovat síťovou komunikace ve formátu PCAP. Systém musí být možné použit na data z libovolné síťové vrstvy a musí být jednoduše rozšiřitelný o nové síťové protokoly a atributy.
6. Otestujte vytvořený systém a zaměřte se také na porovnání anonymizovaných dat vůči původním datům. Pro porovnání zvolte vhodné metriky.
7. Zhodnoťte dosažené výsledky.

Literatura:

- Combs, Gerald. "Tshark - Dump and Analyze Network Traffic." Wireshark (2012).
- Yurcik, William, et al. "Privacy/Analysis Tradeoffs in Sharing Anonymized Packet Traces: Single-field Case." 2008 Third International Conference on Availability, Reliability and Security. IEEE, 2008.
- Pang, Ruoming, et al. "The Devil and Packet Trace Anonymization." ACM SIGCOMM Computer Communication Review 36.1 (2006): 29-38.
- Farah, Tanjila, et al. "Anonym: A Tool For Anonymization of the Internet Traffic." 2013 IEEE International Conference on Cybernetics (CYBCO). IEEE, 2013.

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

Vedoucí práce: **Holkovič Martin, Ing.**
Konzultant: Dudek Jindřich, Ing., Flowmon
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 20. května 2020
Datum schválení: 30. října 2019

Abstrakt

Diplomová práce se zabývá návrhem a implementací aplikace, která umožní anonymizaci síťové komunikace uložené ve formátu PCAP. Práce se zabývá jednotlivými vrstvami architektury TCP/IP, na jejíž vrstvách představuje významné identifikátory komunikujících stran. Pro potřeby anonymizace zmíněných identifikátorů a dalších citlivých údajů popisuje práce několik anonymizačních metod. Navržená a implementovaná aplikace užívá konzolového nástroje TShark pro převod formátu PCAP do formátu JSON, se kterým následně pracuje. Užití nástroje TShark odstraňuje nutnost ručně zpracovávat jednotlivé pakety a má kladný dopad, kvůli velkému množství podporovaných protokolů, na rozšířitelnost výsledné aplikace. Proces anonymizace je řízen anonymizační politikou, kterou je možné libovolně rozšiřovat o nové atributy či anonymizační metody.

Abstract

This diploma thesis deals with the design and implementation of an application suitable for the anonymization of PCAP files. The thesis presents TCP/IP model and for each layer highlights attributes that can be used to identify real people or organizations. Some of the anonymization methods suitable to modify highlighted attributes and sensitive data are described. The implemented application uses TShark tool to parse byte data of PCAP format to JSON format that is used in the application. TShark supports lots of network protocols which allows the application to anonymize various attributes. Anonymization process is controlled by anonymization politics that can be customized by adding new attributes or anonymization methods.

Klíčová slova

anonymizace, tshark, TCP/IP, IP adresa, MAC adresa, pcap, libpcap, anonymizační metody, anonymizační politika, TCP, anonymizační nástroj

Keywords

anonymization, tshark, TCP/IP, IP address, MAC address, pcap, libpcap, anonymization methods, anonymization politics, TCP, anonymization tool

Citace

NAVRÁTIL, Petr. *Anonymizace PCAP souborů*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Holkovič

Anonymizace PCAP souborů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Martina Holkoviče. Další informace mi poskytl konzultant Ing. Jindřich Dudek. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Navrátil
10. června 2020

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Martinu Holkovičovi za věcné rady a ochotu při vedení této diplomové práce. Velkou pomocí byly i připomínky od Ing. Jindřicha Dudka.

Obsah

Kapitola 1

Úvod

Internetová komunikace nám dává pocit anonymity. Pocit, že naše aktivita je těžce a nebo vůbec zjistitelná, že nelze spojit naše virtuální akce s osobami lidského světa. Ovšem každá akce, ač virtuální, zanechává stopy. Stopy ve formě komunikace jednotlivých protokolů, jenž nám využité internetové služby poskytly. Mezi služby lze zahrnout čtení internetových deníků, prohlížení fotografií či finanční transakce. O některých službách by se dalo říci, že nám zanechané stopy nevadí, ovšem nelze to tvrdit o všech.

Komunikace může být zaznamenávána pro další využití. Nejedná se ani tak o zjišťování kdo s kým byl v kontaktu, či jakou užil službu, ale spíš o přehled celkového dění na dané síti. Důvodů může být vícero – výzkumné účely sledující chování nových protokolů či hledání abnormalit a nestandardních chování nebo také odhalování útoků a narušení bezpečnosti sítě.

Zaznamenaná komunikace má tak velkou cenu, jelikož může nést celý průběh síťového útoku, který je možné zkoumat. Je tak vhodné zaznamenanou komunikaci déle sdílet. Ovšem kromě samotného útoku může obsahovat spoustu citlivých informací. Již zmíněnou komunikaci reálných osob, citlivé údaje vlastníka sítě – jako jsou různé služby dostupné pouze uživatelům na dané síti, užité technologie apod. Informace, pokud by se dostaly to nesprávných rukou je možné zneužít například pro plánování dalšího útoku.

Komunikaci je možné různě upravovat než dojde k jejímu zveřejnění či sdílení. Lze tak odstranit citlivé informace a skrýt události, které nechce majitel komunikace publikovat. Ovšem odstraněním či úpravou je možné zaznamenanou komunikaci poškodit až zcela zneplatnit. Je tak třeba najít rovnováhu mezi množstvím informací, které je možné odstranit či upravit a dopadem, které toto odstranění má. Tímto procesem se zabývá anonymizace síťové komunikace, kterou tato diplomová práce řeší.

Cílem práce je definovat citlivé informace komunikace, způsob jejich anonymizace a vytvoření snadno rozšiřitelné aplikace, která umožní vlastní anonymizaci. Práce postupně představuje síťovou architekturu TCP/IP, která umožňuje síťovou komunikaci. Popsány jsou jednotlivé vrstvy včetně některých protokolů a identifikátorů uživatelů. Architekturou se zabývá kapitola č. 2. Jak si představit anonymizaci a jaké způsoby je možné na síťová data aplikovat řeší kapitola 3. Návrhem aplikace se zabývá kapitola č. 4. Kapitola č. 5 řeší vlastní implementaci aplikace a v kapitole č. 6 je možné nalézt její testování.

Kapitola 2

Síťová komunikace

Komunikace dvou a více zařízení na společné síti může mít více podob. Způsob, kterým jednotlivá zařízení komunikují, včetně prostředí, které jim tuto komunikaci umožňuje, má velký vliv na obsah zpráv, jež je nutné přenášet, aby celá komunikace dávala smysl.

Pro účel anonymizace komunikace je nejprve nutné pochopit, jakým způsobem samotná zařízení komunikují, jaké informace a v jaké podobě jsou přenášeny, jak vypadá architektura dané komunikace a jaké jsou její zákonitosti. V následující sekci 2.1 je pozornost věnovaná architektuře TCP/IP. Představeny jsou její principy a vrstvy, které jsou zodpovědné za různé úrovně abstrakce komunikace. Definovány jsou také klíčové identifikátory těchto vrstev a jejich užití.

Sekce 2.2 se dále zaměřuje na způsob zpracování vlastní síťové komunikace a představuje informace o způsobu získání dat síťové komunikace včetně jednoho ze způsobů její možné souborové reprezentace pro potřeby ukládání. Jako souborová reprezentace je vybrán formát libpcap (pcap), který je detailně popsán.

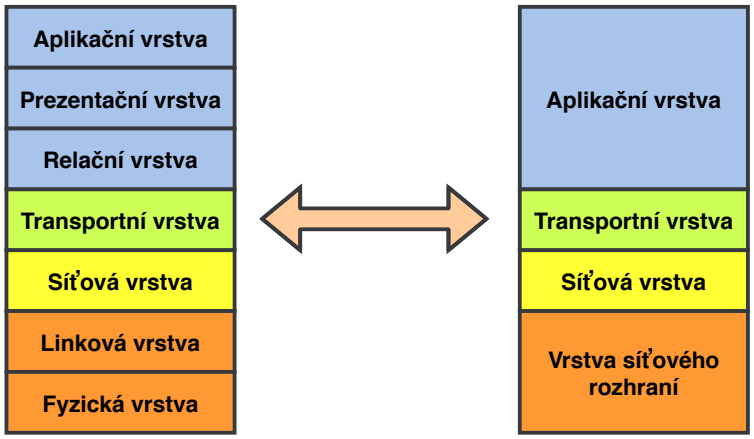
Příklady nástrojů umožňujících zpracování záznamů síťové komunikace jsou popsány v poslední sekci 2.3, jež představuje mimo jiné konzolový nástroj TShark hrající klíčovou roli v následujícím návrhu anonymizační aplikace, jíž se tato práce zabývá.

2.1 Architektura TCP/IP

Architektura síťového modelu TCP/IP je jednou z existujících architektur pro komunikaci zařízení po sdílené počítačové síti. Model se stal základem Internetu [13] a je tvořen pouze čtyřmi vrstvami, čímž se mírně liší od referenčního modelu ISO/OSI. Referenční model přestavuje celkem sedm vrstev, které popisují způsob komunikace pouze mezi vrstvami stejné úrovně. Uplatněn je hierarchický přístup, kde daná vrstva užívá pro komunikaci vrstev nižších, aniž by znala způsob jejich chování nebo detailní implementaci. Důvodem sníženého počtu vrstev modelu je sloučení některých jeho vrstev. Jedná se především o vrstvy fyzickou a linkovou, které jsou sloučeny ve vrstvu síťového rozhraní. Vrstvy relační, prezentační a aplikační modelu ISO/OSI jsou v modelu architektury TCP/IP zastoupeny pouze jednou vrstvou, a to vrstvou aplikační. Srovnání představené architektury vůči referenčnímu modelu je možné vidět na obrázku č. 2.1.

Architektura označuje tzv. **PDU** (Process Data Unit), které představují datové jednotky jednotlivých vrstev modelu. Dle vrstvy modelu hovoříme o **rámci** pro vrstvu síťového rozhraní, **IP datagramu** pro vrstvu síťovou, vrstva transportní užívá označení pro datové

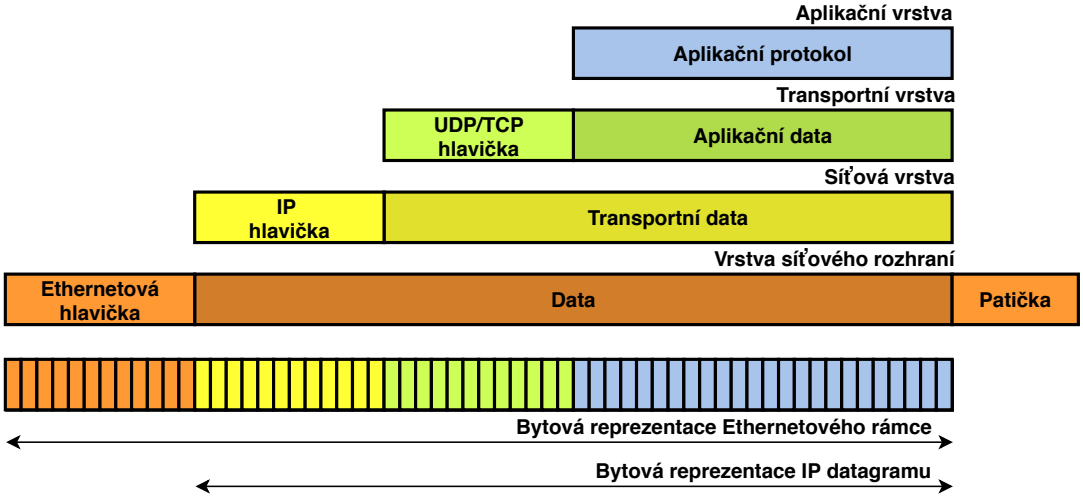
jednotky **TCP/UDP pakety** a jako **data** či **zprávu** označujeme datovou jednotku vrstvy nejvyšší – aplikační.



Obrázek 2.1: Referenční model ISO/OSI a TCP/IP.

Datové jednotky jednotlivých vrstev mají důležitou roli při realizaci fyzického přenosu, jelikož dochází k jejich skládání, přesněji k **zapouzdření**. Jednotky vyšších vrstev jsou zapouzdřeny jako součást datových jednotek vrstev nižších. Lze konstatovat, že celý protokol vyšší vrstvy je užitý jako datová část protokolu nižší vrstvy. Výsledek si lze představit jako souvislé pole bytů, které je po síti přenášeno, přičemž je zodpovědností jednotlivých protokolů definovat, kde začíná jiný – zapouzdřený – protokol. Tento proces probíhá na straně odesílatele a příklad je možné vidět na obrázku č. 2.2 včetně výběru protokolů uplatňovaných na jednotlivých vrstvách. Příjemce aplikuje proces opačný – **rozbalení** – čímž získá aplikační data.

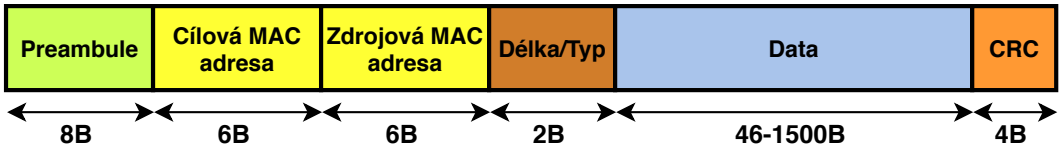
Každá z vrstev má na starost jinou funkcionalitu v rámci celé architektury a užité protokoly v různé míře přenášejí atributy, jež identifikují účastníky komunikace. Z tohoto důvodu jsou zodpovědnosti a některé přenášené atributy jednotlivých vrstev modelu detailněji popsány v následujících sekcích.



Obrázek 2.2: Architektura modelu TCP/IP.

2.1.1 Vrstva síťového rozhraní

Nejnižší vrstvou architektury TCP/IP je vrstva síťového rozhraní spojující vrstvy linkovou a fyzickou referenčního modelu ISO/OSI. Do této vrstvy řadíme technologie pro fyzický přenos komunikace i funkce těchto technologií [13]. Mezi užívané technologie lze zařadit například Token Ring[1] či Ethernet, který je užitý jako výchozí technologie v této práci.



Obrázek 2.3: Formát rámce technologie Ethernet IEEE 802.3

Na obrázku č. 2.3 lze vidět formát ethernetového rámce. Důležitou částí rámce jsou zdrojová a cílová fyzická adresa. Ty jsou 48bitové a slouží jako unikátní identifikátory síťového rozhraní na lokální síti, kde jsou užity při adresování samotného rámce jeho příjemci. Fyzickou adresu lze rozdělit na dvě části. Prvních 24 bitů adresy označuje výrobce síťového rozhraní. Kromě identifikace výrobce obsahuje tato část ještě další informace. Nejméně významný bit prvního oktetu značí, zda se jedná o skupinovou fyzickou adresu, či nikoliv. Druhý nejméně významný bit udává, zda je adresa spravována lokálně, nebo globálně. Tento bit určuje, jestli je celá část výrobce validním identifikátorem či nikoli. Zbylých 24 bitů je výrobcem přiděleno a mělo by být unikátních. Pro reprezentaci MAC adresy se užívá notace hex zápisu jednotlivých oktetů oddělených dvojtečkou, např. `ff:ff:ff:ff:ff:ff` označuje všesměrovou MAC adresu.

2.1.2 Síťová/Internetová vrstva

Následující vrstvou modelu je síťová vrstva neboli vrstva internetová, která odpovídá stejnojmenné vrstvě referenčního modelu. Přenášené datové jednotky nazýváme IP datagramy. Tato vrstva je zodpovědná za adresaci a směrování přenášených datagramů. Hovoříme zde o tzv. doručení s největším úsilím (best-effort delivery), tedy snaze doručit data nejvhodnější cestou [13]. Přenášená data na této vrstvě nejsou kontrolována na správnost přenosu a nejsou nikterak opravována.

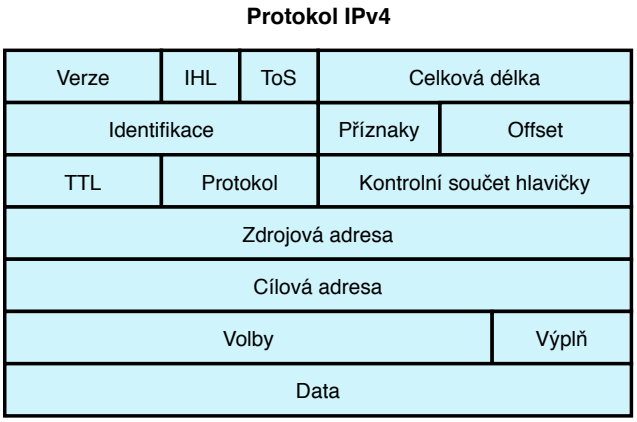
O komunikaci na úrovni této vrstvy se stará několik protokolů, mezi které lze zařadit protokol **IP**, jenž se stará o vlastní přenos. Příklad formátu IP protokolu verze 4 je možné vidět na obrázku č. 2.4. Ačkoliv struktura protokolu umožňuje přenášet datagramy až o velikost 64kB, nenabývají takových velikostí, jelikož nižší vrstva umožňuje zapouzdření maximálně 1500B dat.

Je-li velikost přenášených dat větší něž toto omezení, dochází k tzv. fragmentaci na úrovni protokolu IP, jejímž důsledkem je rozdělení jednoho IP datagramu na více datagramů, které jsou dále přenášeny individuálně. Jejich spojení pak řeší cílové zařízení. [10]

Dále zde najdeme protokol **ICMP** určený pro správu komunikačních zpráv, příkladem je ztráta dat či nedostupnost cílového zařízení. Nelze vynechat protokol **ARP** sloužící k mapování IP adresy na MAC adresy, což má pozitivní dopad na funkci předešlé – fyzické – vrstvy.

[1] https://www.sciencedirect.com/topics/computer-science/token-ring

Zmíněná IP adresa slouží jako identifikátor komunikujících zařízení na této vrstvě a je důležitým atributem užívaných protokolů. Její užití lze nicméně nalézt i v protokolech vyšších vrstev, příkladem aplikační protokol DNS. Oproti identifikátoru předchozí vrstvy je možné ji užít i při směrování mimo lokální síť. Síťové zařízení nemusí mít po celou dobu své existence stejnou IP adresu, jako je tomu u MAC adresy. Adresy jsou přidělovány staticky administrátorem počítačové sítě, případně dynamicky za pomoci DHCP serveru.



Obrázek 2.4: Formát protokolu IP verze 4

Rozlišujeme dvě verze IP protokolu, a tím i dvě verze jejich IP adres. Verzi 4, kde IP adresa čítá 32 bitů (například `255.255.255.255`), a verzi 6, kde je adresa tvořena 128 bity (např. `fe80::f623:4f3c:e0f2:6625/64`). Pozornost bude nadále věnována pouze verzi 4.

Třída	**Hodnota prvního oktetu**	**MSB prvního oktetu**	**Počet bitů sítě**
A	1 - 126	0	8
B	128 - 191	10	16
C	192 - 223	110	24
D	224 - 239	1110	-
D	240 - 254	1111	-

Tabulka 2.1: Rozdělení adresového prostoru IP verze 4 do tříd

IP adresa se skládá ze dvou částí – první definuje do jaké sítě dané síťové zařízení patří, druhá část označuje samotné síťové zařízení. Počet bitů, které patří síťové části, označujeme jako prefix sítě. Jelikož je počet IP adres omezen počtem bitů, existuje několik způsobů, jak tento adresní prostor rozdělit. Hovořit lze o rozdělení adresního prostoru do tříd. Rozdělení do tříd je uvedeno v tabulce č. 2.1, kde lze vidět, že počet bitů, které jsou užity pro síťovou část, je pevně daný. Ovšem tento přístup vede k neefektivnímu způsobu využití adresového prostoru [13]. Dalším způsobem rozdělení, který tento problém částečně řeší, je vytváření podsítí, tzv. subnetting. Ten slouží k rozdělení přiděleného adresového prostoru na více podsítí. Pro vytvoření podsítě je prodloužen prefix sítě užitím několika bitů z části síťového zařízení, čímž dojde ke snížení počtu adresovatelných zařízení. Pro směrování mimo lokální síť se nadále užívá původní délka prefixu přiděleného adresového prostoru. Dalším zmíněným způsobem je užití tzv. beztřídního adresování (CIDR) [8]. Princip je obdobný jako u vytváření podsítí, avšak na rozdíl od subnettingu dochází ke spojování adresového pro-

storu [13]. Podmínkou pro správné směrování je nutnost uchovávat na směrovačích kromě samotné sítě i její masku [13].

O správu IP adres se stará organizace IANA[2] a přidělování adresového prostoru má hierarchickou podobou. Lze definovat regionální registrátory, kteří jsou zodpovědní za rozdělení podle světových regionů, a lokální registrátory, kteří přidělené adresové bloky dále distribuují koncovým uživatelům.

Některé adresy mají speciální význam, a proto nejsou užity pro fyzické přidělovány koncovým zařízením či sítím. Příkladem lze uvést adresu `255.2555.255.255`, jež představuje všesměrovou (broadcast) adresu lokální sítě, které umožňuje směrovat všem zařízením na síti daný datagram. Adresy sítě `127.0.0.0/8`, tzv. loopback, představují adresy v režii operačních systému. Pro soukromé účely je vyhrazen blok adres `192.168.0.0/16`. Více vyhrazených IP rozsahů lze nalézt na stránkách organizace IANA.

2.1.3 Transportní vrstva

Třetí vrstvou architektury TCP/IP je vrstva transportní, která vytváří logické spojení mezi procesy komunikujících stran [13]. Datové jednotky užívané na této úrovní označujeme za pakety, jež zapouzdřují aplikační data. Data mohou být větší než je maximální kapacita, kterou jsou schopny transportní protokoly přenést, a tak může docházet k segmentaci aplikačních dat do více paketů.

Identifikátorem pro adresování je tzv. port identifikující službu, se kterou je komunikace vyžadována. Rozlišujeme zpravidla dva porty, zdrojový pro iniciátora komunikace a cílový pro dotazovanou službu. Číslo portu je 16bitové číslo, což umožňuje adresovat celkem 65 535 služeb. Tento prostor se dělí na tři skupiny, které spravuje již zmíněná organizace IANA. Porty z intervalu 0–1 023 nazýváme rezervované a jsou jako jediné přidělovány zmíněnou organizací, příkladem port 53 definující službu DNS. Další skupinou jsou porty registrované pro rozmezí 1 024–49 151, zbylé porty označujeme za dynamické.

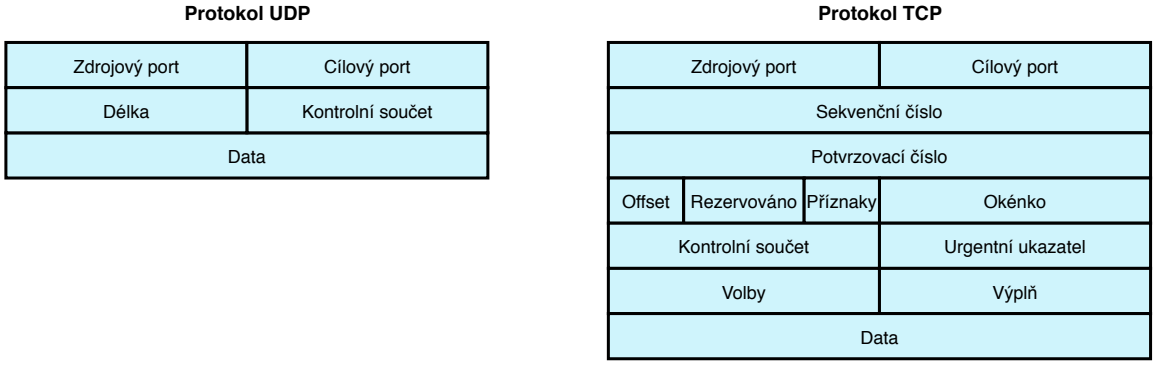
Při využití informací z nižších vrstev lze definovat přenosový kanál (stream) dvou komunikujících procesů. Kanál tvoří transportní protokol, zdrojová a cílová IP adresa a zdrojový a cílový port služeb.

Jedním z protokolů uplatněných na transportní vrstvě je **UDP**, viz RFC 768 [18]. Protokol je užívaný pro rychlý přenos paketů bez vytváření spojení, čímž neposkytuje žádný mechanismus pro ověření úspěšnosti doručení odchozích paketů. Jeho uplatnění lze nalézt v aplikacích kladoucích důraz na rychlost komunikace i za cenu občasného výpadku zasílaných dat, příkladem může být přenos multimediálních souborů [13].

Vyžaduje-li aplikace zajištění spolehlivého přenosu, lze užít protokol **TCP**, viz RFC 793 [19]. Protokol komunikujícím stranám poskytuje kromě spolehlivosti doručení přenášených aplikačních dat také zajištění doručení ve stejném pořadí, v jakém byly odeslány. Užívá k tomu sekvenčních čísel jednotlivých paketů. Pakety, které se po cestě ztratily a nebyly doručeny, jsou díky užití tohoto protokolu opakovaně zaslány. Protokol automaticky zajišťuje segmentaci dat v případě, že je jejich velikost větší, než dovolují nižší vrstvy přenést. Data jsou rozdělena do segmentů a ty jsou jako samostatné pakety odeslány příjemci, kde jsou ve správném pořadí opět spojeny a předány cílovému procesu. Tímto se liší zpracování velkých dat oproti UDP, kde není rozdělení ani zajištění pořadí řešeno samotným protokolem. Zajištění těchto výhod ovšem přináší režii, která se projevuje v rychlosti komunikace, jelikož před vlastním přenosem aplikačních dat, je nutné ustanovit komunikační kanál.

[2] www.iana.org

Podle vývoje komunikace je protokol schopen řídit tok, a tak jej, v případě zahlcení sítě, cíleně zpomalovat. Formát obou používaných protokolů je možné vidět na obrázku č. 2.5.



Obrázek 2.5: Formáty protokolů UDP(vlevo) a TCP(vpravo)

2.1.4 Aplikační vrstva

Aplikační vrstva představuje nejvyšší vrstvu architektury TCP/IP a tvoří ji aplikace komunikující po síti [13]. Oproti předchozím vrstvám nelze jednoznačně určit primární identifikátor adresování komunikace této vrstvy, jelikož škála aplikačních protokolů je velmi různorodá. Příkladem identifikátoru může být tzv. doménová adresa (www.google.com) užívaná k identifikaci počítačů v síti [13]. V elektronické poště lze nalézt jako identifikátor emailovou adresu (john@doe.com), či adresa klienta protokolu SIP (sip:john@doe.com) pro internetovou telefonii. Některé identifikátory aplikační vrstvy jsou vázané na jiné vrstvy, a tak je nutné zajistit mezi nimi překlad. Jako příklad lze uvést doménové adresy, jež jsou vázané na IP adresy síťové vrstvy, a proto je nutné je mezi sebou překládat např. pomocí protokolu **DNS**.

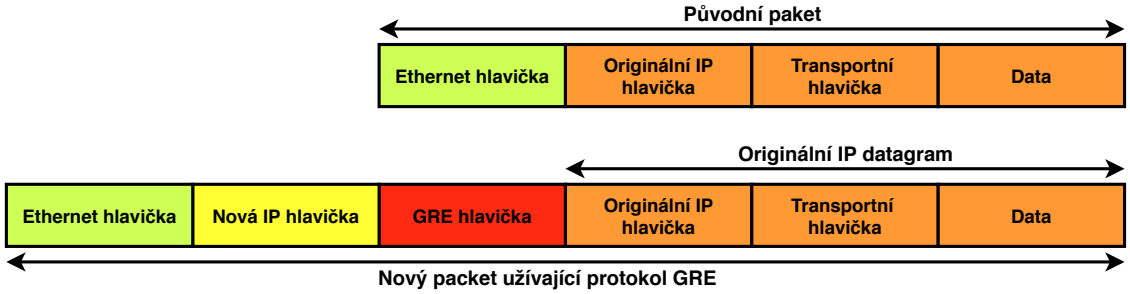
Identifikátory komunikace nemusí být jedinou důležitou přenášenou informací. Na této vrstvě lze nalézt citlivá uživatelská data, obsah elektronických zpráv apod. Proto je vhodné tuto komunikaci zabezpečit proti nechtěnému získání informací. Avšak i zabezpečená komunikace či její ustanovení, může nést zajímavá data. Příkladem je protokol **TLS**, během jehož komunikace mohou být přenášeny certifikáty komunikujících stran či informace pro ustanovení šifrované komunikace — např. parametry pro výměnu klíčů metody Deffie-Hellman.

2.1.5 Tunelovaní komunikace

Předchozí sekce představily jednotlivé vrstvy architektury TCP/IP a princip hierarchického zapouzdřování. Ovšem ne vždy je tato architektura dostatečná. Příkladem může být komunikace mezi dvěma nekompatibilními sítěmi z důvodu rozdílných architektur. Možným řešením je tzv. tunelování, které umožňuje zapouzdření protokolu do jiného protokolu, ovšem způsobem, který nemusí odpovídat představené architektuře TCP/IP, jelikož může docházet k zapouzdření nižší vrstvy do vrstvy vyšší. Tunelování nachází mimo jiné své uplatnění v oblasti virtuálních počítačových sítí, kde umožňuje uživateli na veřejné, případně nezabezpečené, síti přístup do sítě soukromé.

Jedním z možných způsobů implementace tunelování je užití protokolu **GRE** (Generic Routing Encapsulation), viz RFC 2784 [7]. Protokol umožňuje zapouzdření různých protokolů síťové vrstvy do jiných protokolů této vrstvy. Jinými slovy, celý datagram je uložen jako

datová část nového datagramu, který je následně použitý při směrování. Hlavička nového datagramu je upravena tak, aby zajistila správné směrování a také nese informaci o užití protokolu GRE, jehož hlavička je do datagramu přidána. Tato skutečnost způsobí, že se může některá z vrstev TCP/IP modelu objevit v zasílané komunikaci vícekrát, což vede k většímu přenosu informací o komunikujících stranách za předpokladu, že nebylo užito šifrování. Princip modifikace originálního datagramu je možné vidět na obrázku č. 2.6.



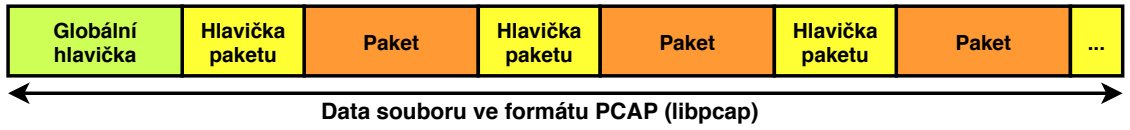
Obrázek 2.6: Modifikace IP datagramu protokolem GRE

Kromě výše zmíněného protokolu GRE existují i další alternativy, které lze užít. Uplatnění nachází např. protokoly **L2TP** či **PPTP** pracující stejně jako předešlý protokol na síťové vrstvě.

2.2 Zpracování síťového záznamu

Jedním ze způsobů získání síťového záznamu je poslouchání veškeré komunikace na některém ze síťových zařízení v dané síti, případně umístěním dedikované sondy sloužící k tomuto účelu. Existuje velké množství softwarových řešení, které lze k záznamu síťové komunikace použít. Uvést lze například program **Tcpdump**[3] či knihovnu **Libpcap**[4], která umožňuje čtení i zápis komunikace. Stejným jménem je také označován jeden z formátů pro ukládání záznamů o síťové komunikaci.

Soubory formátu libpcap jsou binárními soubory a je doporučeno je označovat příponou `.pcap`. Formát souboru obsahuje globální hlavičku, která nese informace týkající se samotného souboru. Globální hlavička je následována jednotlivými pakety zaznamenané komunikace. Každý paket je předstoupen vlastní hlavičkou, která obsahuje dodatečné informace zaznamenané při přijmutí paketu, a nejedná se o fyzickou součást přenášeného paketu. Popsaný formát je možné nalézt na obrázku č. 2.7.

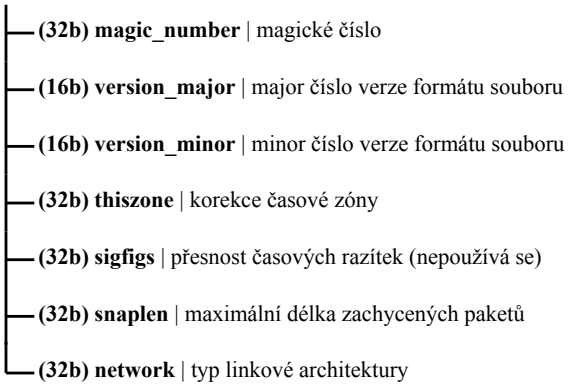


Obrázek 2.7: Formát souboru PCAP (libpcap)

[3] https://www.tcpdump.org/
[4] https://www.tcpdump.org/

Globální hlavička

Globální hlavička, obrázek č. 2.8, nese informace o samotném souboru, které mají vliv na jeho zpracování. První informací je tzv. magické číslo. Tato informace označuje, že je soubor ve formátu libpcap, a také je užita při určování endianity (byte order), která je užita pro aktuální soubor. Pořadí bytů se týká pouze hlaviček formátu a netýká se samotných paketů. Princip určování je následující. Aplikace při vytváření souboru zapíše hodnotu `0xa1b2c3d4` v nativním pořadí bytů systému. Aplikace, jež existující soubor zpracovává, tuto hodnotu načte podle nativního pořadí bytů a porovná, zda je načtená hodnota shodná s očekávanou hodnotou či nikoli. V případě shody je pořadí bytů identické, v opačném případě je načtené číslo `0xd4c3b2a1`, což je signálem jiného pořadí bytů, které aplikace následně musí řešit. Magické číslo je také užito pro zjištění přesnosti časových razítek souboru. Pro tuto informaci existuje v globální hlavičce samostatný atribut, ovšem v praxi není využitý. Již uvedené číslo `0xa1b2c3d4` je užito pro soubory, jichž časová razítka mají přesnost v řádu mikrosekund. Užívá-li soubor přesnost v rámci mikrosekund, je použité magické číslo rovno `0xa1b23c4d`, přičemž princip detekce endianity je identický. Mezi další informace patří verze formátu, kterou soubor užívá, časová zóna či typ protokolu síťového rozhraní. [9]



Obrázek 2.8: Struktura globální hlavičky formátu libpcap

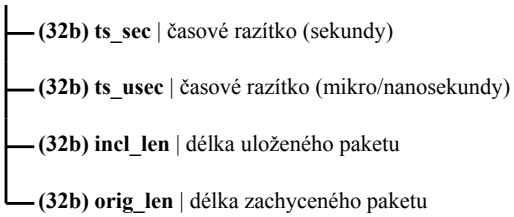
Hlavička paketu

Hlavička paketu představuje dodatečné informace o každém zaznamenaném paketu a její formát lze vidět na obrázku č. 2.9. Mezi významné informace patří unixové časové razítko představující čas přijetí paketu. Na základě přesnosti časových razítek souboru definovaných v globální hlavičce obsahuje hlavička paketu i čas přijmutí paketu v mikrosekundách, případně nanosekundách. Uvedena je i celková délka přijatého paketu a uloženého paketu. Hodnoty se od sebe mohou lišit, pokud byl paket zmenšen z důvodu omezené velikosti ukládaných paketů. [9]

2.3 TShark

Pro zpracování záznamu síťové komunikace nejen formátu libpcap lze užít populární grafický nástroj Wireshark[5], který poskytuje přehledně komunikaci zobrazit a analyzovat. Nástroj umožňuje kromě zpracování vstupního souboru sledovat vybrané komunikační toky

[5] https://www.wireshark.org/



Obrázek 2.9: Struktura hlavičky paketu formátu libpcap

různých protokolů, např. TCP stream, a adekvátně je zpracovat. Lze tak získat spojitá data segmentované TCP komunikace apod. Samotné zachytávání síťové komunikace je také podporováno, a to za pomocí knihovny libpcap. Komunikaci je možné nativně ukládat do souborového formátu **pcapng**[6], jehož snahou je řešit některá omezení formátu libpcap. Konverze mezi formáty je možná nativně pomocí zmíněného nástroje za cenu ztráty informací dostupných pouze v novějším formátu.

Ke grafickému nástroji existuje i jeho konzolová verze TShark, která může nalézt uplatnění např. při zpracovávání záznamu síťové komunikace další aplikací. Pomineme-li chybějící grafické prostředí, představují obě aplikace obdobnou funkcionalitu. Důležitým aspektem obou aplikací je shodný způsob zápisu jmen identifikátorů jednotlivých protokolů, které lze užít při přístupu k samotným atributům. Definice jednotlivých atributů protokolů lze nalézt na stránkách podporovaných protokolů[7]. Konzolový nástroj navíc disponuje velkým množstvím přepínačů sloužících např. k filtrování příchozí komunikace. Seznam možných přepínačů včetně dokumentace nástroje lze nalézt v manuálových stránkách [5].

Jak již bylo zmíněno, nástroj lze užít i ke zpracování uložené síťové komunikace. Pomocí přepínačů lze definovat filtry pro výběr specifických paketů či získat statistiky o dané komunikaci. Nástroj také poskytuje možnost parsovaní[8] jednotlivých paketů. Volbou přepínačů lze manipulovat s informacemi, které jsou parsováním získány. Kromě hodnot jednotlivých atributů protokolů lze získat i jejich bytové vzory a umístění v zapouzdřeném bytovém rámci. Výstup tohoto procesu lze uložit do několika možných formátů, např. **PDML**, **TEXT** či **JSON**, což je vhodné pro další zpracování, jelikož odpadá nutnost pracovat s jednotlivými pakety v binární podobě.

Struktura výstupních dat, pro formát JSON, je zobrazena na obrázku č. 2.10. Lze vidět, že jednotlivé protokoly nejsou po rozbalení vnořeny do sebe tak, jak definuje architektura TCP/IP, ale jsou na stejné úrovni. Jednotlivé atributy protokolů hierarchické zanoření již dodržují. Při užití přepínače `-x` případně `-T jsonraw`, je každý atribut protokolu obohacen o svoji **__raw** verzi, která nese informace o umístění v bytovém rámci včetně nezpracované hodnoty v hexadecimální podobě, viz obrázek č. 2.11. Každý atribut je definovaný pěticí informací:

1. originální hodnota v hexadecimální podobě,
2. začátek atributu v bytovém rámci,
3. délka atributu v bytech,
4. bitová maska, sdílí-li více atributů jeden byte,

[6] https://wiki.wireshark.org/Development/PcapNg
[7] https://www.wireshark.org/docs/dfref/
[8] Parsovat - provést rozbalení paketu na jednotlivé protokoly s přiřazením hodnot jednotlivým atributům.

```
{
    "layers": {
        "frame_raw": [],
        "frame": {},
        "eth_raw" : [],
        "eth": {},
        "ip_raw": [],
        "ip": {},
        "tcp_raw": [],
        "tcp": {},
        "tls_raw": [],
        "tls": {}
    }
}
```

Obrázek 2.10: Část formátu výstupního souboru JSON nástroje TShark. Hodnoty vlastních polí jsou vynechány a zastoupeny pouze užitým datovým typem.

5. datový typ atributu.

```
{
    "eth": {
        "eth.dst_raw": [
            "ffffffffffff",
            0,
            6,
            0,
            29
        ],
        "eth.dst": "ff:ff:ff:ff:ff:ff"
    }
}
```

Obrázek 2.11: Formát jednotlivých atributů výstupního souboru JSON nástroje TShark.

Kapitola 3

Teorie anonymizace

Probíhající síťovou komunikaci je vhodné monitorovat a případně zaznamenávat hned z několika důvodů. Sledováním vývoje komunikace lze definovat charakteristiky sledované sítě, objevit závislosti či získat podklady pro optimalizaci síťových zdrojů. Síťové záznamy lze také uplatnit při síťovém inženýrství, mapování internetových topologií, návrhu nových protokolů a aplikací či při analýze síťové bezpečnosti využívající abnormalit v komunikaci pro detekování síťových útoků. [6]

Síťový útok může probíhat napříč několika sítěmi, a je tak vhodné pro zvýšení bezpečnosti sítě použít informace o komunikaci okolních sítí, které byly útokem postiženy, k analýze útoku a zvýšení odolnosti dané sítě vůči odhalenému útoku. Aby mohlo dojít ke zkoumání komunikace mimo hranice sítě náležící oběti útoku, je vhodné sdílet informace mezi jednotlivými postiženými stranami. Ovšem ke sdílení záznamů síťové komunikace mezi jednotlivými organizacemi či vlastníky daných sítí dochází málo až vůbec. [22]

Mezi hlavní úskalí poskytování síťových záznamů třetím stranám patří strach z jejich zneužití. Data síťové komunikace mohou být zneužita samotnou třetí stranou, případně se data mohou dostat do nepovolených rukou – například mezi útočníky. Síťová komunikace, obzvlášť nešifrovaná, může obsahovat tajné informace dané organizace či citlivé informace uživatelů sítě, mezi které lze zařadit obsah elektronické pošty či jiné osobní komunikace [12]. Útočník tak může získat např. citlivé soukromé informace nebo přihlašovací údaje uživatelů. Mezi citlivé informace lze zařadit kromě samotných přenášených informací i znalost o existující komunikaci mezi dvěma stranami [16].

Ze záznamu síťové komunikace lze mimo citlivé informace uživatelů sítě získat představu o užité topologii, technologii a konfiguraci sledované sítě. Tyto informace je možné zneužít při plánování budoucího útoku na danou síť, jelikož mohou odhalovat zranitelná místa, příkladem je identifikace míst citlivých na **bottleneck**[1], které mohou zvýšit sílu útoku typu **DoS** (Denial of Service) [20].

Sekce 3.1 se zabývá principy anonymizace a jejími problémy. Metody, které lze uplatnit pro anonymizaci síťových dat, představuje sekce 3.2. V sekci 3.3 jsou uvedeny některé z možných atributů protokolů, které jsou vhodné k anonymizaci.

3.1 Anonymizace

Rozhodne-li se organizace ke sdílení záznamů své síťové komunikace třetím stranám, ať už přímo konkrétní organizaci či široké veřejnosti, většinou se uchýlí ke zpracování daných

[1] https://www.techopedia.com/definition/24819/network-bottleneck

záznamů před jejich zveřejněním z výše uvedených důvodů – tento proces nazývejme **anonymizace**. Jako příklady sdílené komunikace lze uvést například záznamy DNS komunikace univerzity Thapar[2] či datasety organizace RIPE[3].

Souběžně s aplikováním všech anonymizačních prostředků na záznam síťové komunikace dochází ke snižování jeho výzkumné hodnoty, jelikož proces anonymizace může odstraňovat části či celá data anonymizovaných atributů. Na anonymizaci se tak dá nahlížet jako na způsob hledání kompromisu mezi těmito dvěma stavy, přičemž způsob, jak kompromisu dosáhnout, není přesně daný. [15]

V případě anonymizovaného síťového provozu je vhodné znát účel, za kterým jsou data poskytována. Tato znalost může napomoci minimalizovat dopady způsobené samotnou anonymizací, a zvýšit tak vhodnost poskytovaných dat pro daný případ užití. Bude-li případný výzkum zaměřený na přesná časová razítka, je vhodné je zachovat či použít anonymizační metodu, která zachová vztahy mezi jednotlivými razítky i přes případnou anonymizaci. Data určená k anonymizaci – vzory – tak mohou mít více anonymizovaných podob – obrazů na základě jejich budoucího použití. [14].

Nejjednodušším způsobem anonymizace je odstranění všech citlivých a identifikujících údajů. Touto myšlenkou je inspirovaný například nástroj **tcpdpriv**[4], který odstraňuje např. informace o datových zprávách transportní vrstvy, čímž činí aplikační vrstvu zcela nepoužitelnou pro případnou analýzu síťového provozu. Kromě aplikační vrstvy odstraňuje i některé informace o TCP tocích, což vede k zneplatnění i této vrstvy, čímž se snižuje použitelnost síťového záznamu pro výzkum [15], je-li předmětem zkoumání tato síťová vrstva. Výstup takové anonymizace tak sice může zabránit zneužití poskytnutých informací, nicméně hodnota anonymizovaných dat pro případné výzkumné užití je značně degradována.

Následující sekce 3.2 představuje metody, které lze užít pro anonymizaci různých datových typů protokolových atributů, včetně případných dopadů na anonymizované data. Jejich vhodnou kombinací lze anonymizovat záznamy síťové komunikace takovým způsobem, aby byl dopad samotné anonymizace co nejmenší.

Aplikace anonymizace na síťový záznam komunikace nicméně nezaručuje, že data nelze zneužít. Některé z uvedených technik užívají vstupní data anonymizovaných atributů a transformují je na jiná anonymizovaná data, přičemž zachovávají některé z vlastností původních dat. Jako příklad lze uvést zachování síťové části IP adresy.

Data, která jsou v anonymizovaném souboru ponechána, tak mohou být zneužita pro prolomení anonymizačních technik, které byly užity pro vlastní anonymizaci. Pomocí charakteristik webových serverů lze získat identitu webového serveru, s nímž uživatel komunikoval, i z anonymizovaných dat [12]. Rozlišujeme dva hlavní typy útoků na anonymizovaná data [4]:

- Útok **inspection** (inspekce), který užívá k prolomení anonymizovaných dat volně dostupné informace (např. znalost o majiteli anonymizovaných dat, jeho rozsah IP adres, informace o společnosti či data z databází DNS a WHOIS). Užít lze např. i informace získané skenováním sítě.

- Útok **injection** (otisk) může užívat všech dostupných informací, jako útok inspection. Na rozdíl od něho má ale k dispozici i další informace, které již veřejně dostupné nejsou. V anonymizovaném záznamu síťové komunikace dokáže lokalizovat vzory komunikace, jenž mu napomohou k prolomení anonymizace. Útočník může charakteris-

[2] https://data.mendeley.com/datasets/zh3wnddzxy/1
[3] https://labs.ripe.net/
[4] http://fly.isti.cnr.it/software/tcpdpriv/

tickou komunikaci do záznamu vložit sám, což ale vyžaduje jeho přístup k dané síti. Z důvodu znalosti charakteristické komunikace před její anonymizací dokáže stejnou komunikaci nalézt i v datech anonymizovaných. Útočník tak dostává do ruky anonymizovaná a neanonymizovaná data, čímž lze útok považovat za **known-plaintext attack** [1].

3.2 Anonymizační metody

Pro úspěšné provedení anonymizace síťového záznamu je nutné zaznamenaná data nějakým způsobem modifikovat. Způsoby anonymizace můžeme označit za anonymizační metody či algoritmy. Lze definovat několik základních metod, které se odlišují principy anonymizací vstupních dat. Některé z metod jsou pouze speciálními případy metod jiných, případně kombinují více metod naráz. Některé anonymizační metody je možné použít pouze na vybrané protokolové atributy, jiné je možné užít pro všechny.

Anonymizační metody lze rozdělit na dvě základní kategorie, podle způsobu modifikace vstupních dat [3]:

- **Anonymizace** – v množině možných neanonymizovaných hodnot atributu nelze identifikovat anonymizovanou hodnotu [17]. Mezi metody lze zařadit např. **black marker** či **clear**.
- **Pseudoanonymizace** – neanonymizovaná hodnota atributu je nahrazena pseudoanonymizovanou hodnotou. Mezi hodnotami existuje nějaký vztah, např. **prefix preserving** či **hash**.

Přehled některých anonymizačních metod lze nalézt níže. Tabulka 3.1 demonstruje použití tří anonymizačních metod na IP adresy.

IP adresa	Black marker	Reverse truncation (8)	Prefix preserving
192.168.0.1	10.10.10.10	192.168.0.0	192.172.131.229
192.168.0.65	10.10.10.10	192.168.0.0	192.172.131.140
147.229.2.90	10.10.10.10	147.229.2.0	146.122.129.165
216.58.201.110	10.10.10.10	216.58.201.0	215.201.54.161
147.229.9.26	10.10.10.10	147.229.9.0	146.122.141.26

Tabulka 3.1: Srovnání metod Black marker, Reverse truncation a Prefix preserving aplikovaných na IP adresy.

Black marker

Název metody **black marker** (lze volně přeložit jako černý fix) má analogii v reálném světě. Průběh anonymizace si lze představit jako začerňování informací na vytištěném záznamu komunikace [21]. Pro anonymizaci síťové komunikace nedochází k reálnému začerňování, ale k nahrazování neanonymizované hodnoty statickou (fixní) hodnotou.

Metody je možné aplikovat na všechny atributy síťové komunikace, ovšem značně snižuje výslednou použitelnost anonymizavaného souboru. Dosazení nevalidní statické hodnoty může poškodit následné zpracování paketu parsovacím nástrojem, případně může dojít k nesprávné interpretaci nové hodnoty — např. změna typu protokolu v hlavičce síťového protokolu IP.

Clear

Anonymizační metodu **clear** lze označit za speciální případ metody **black marker**. Zapisovanou statickou hodnotou je nulová hodnota datového typu anonymizovaného atributu. U číselných datových typů lze hovořit o hodnotě `0`, pro textové řetězce lze užít prázdný řetězec. Algoritmus je možné aplikovat i na celé zanořené protokoly a anonymizovat tak např. celou aplikační zprávu přenášenou transportními protokoly. Aplikace algoritmu **clear** má velký dopad na data, jelikož dochází k jejich úplné ztrátě ve výsledném anonymizovaném souboru. [6]

Permutation

Metodou **permutation** označujeme anonymizační proces, kdy dochází k nahrazení neanonymizované hodnoty hodnotou anonymizovanou, která je platná pro daný atribut. Není-li uvedeno jinak existuje mezi neanonymizovanými a anonymizovanými daty vztah jedna ku jedné. Pro implementaci tohoto algoritmu lze užít např. kryptografické šifry,čímž lze zajistit stejný způsob mapování neanonymizované a anonymizované hodnoty mezi více soubory, kdy anonymizace neprobíhá ve stejných časech. Užívá-li šifra kryptografického klíče, je nutné jej pro následující proces anonymizace uchovat. Užít kryptografických přístupů ovšem nelze vždy. Kvůli délce dat anonymizovaného atributu nemusí existovat dostupná kryptografická šifra či způsob, jak nativně zajistit mapování jedna ku jedné. Data by musela být doplněna výplní a následně anonymizovaná, přičemž anonymizovaná data by nemuselo být možné použít kvůli rozdílné délce. [6]

Random permutation

Za metodu **random permutation** lze uvažovat zmíněnou metodu **permutation**, která může, ale také nemusí užívat kryptografických šifer. Rozdílem mezi oběma metodami je způsob získávání anonymizovaných hodnot tak, aby splňovaly mapování jedna ku jedné vůči hodnotám anonymizovaných atributů. V případě klasické metody **permutation** je mapování zajištěno kryptografickou šifrou. Pro **random permutation** je mapování možné zajistit např. tabulkami, jež toto mapování uchovávají. Anonymizované hodnoty tak mohou být získány náhodně, avšak unikátně. [21]

Mezi metody **permutation** lze zařadit i metody, které zachovávají určitou část vstupních dat a zbývající část mutují. Jako příklad lze uvést např. **prefix preservation** metodu, jež lze užít pro anonymizaci IP adres.

Random

Princip metody **random** je podobný metodě **permutation**. Nová, anonymizovaná hodnota je platná, ale nemusí platit vztah jedna ku jedné mezi neanonymizovanou a anonymizovanou hodnotou. Metodu lze užít na všechny atributy síťové komunikace.

Hash

Pro anonymizaci lze užít také hashovacích funkcí, metodu následně označujeme za **hash**. Metodu může poskytovat anonymizovaná data rozdílné délky než data vstupní, a tak je nutné délku výstupu upravovat. Hashovací funkce mohou způsobovat kolize, což má za následek mapování více neanonymizovaných dat na jednu hodnotu dat anonymizovaných. Hash tak lze užít pro atributy, kde není výskyt kolizí problematický. Nutné je také zvážit

bezpečnost tohoto přístupu. Hashovací funkce jsou náchylné na slovníkové útoky, které kvůli omezenému definičnímu oboru některých atributů (např. IPv4 adresy) není problém vypočítat. [21]

Prefix preservation

Metodu **prefix preservation** lze vyložit dvěma způsoby. První způsob výkladu metody spočívá v zachování určité délky (prefixu) neanonymizovaných dat a anonymizací pouze dat zbývajících. Jako příklad lze uvést např. anonymizaci atributu `host` protokolu `HTTP`, kde je vhodné zachovat prefix atributu `Host:`, aby nedošlo porušení formátu protokolu. Samotná informace atributu `host` je anonymizovaná libovolně dostupnými anonynimizačními metodami. Tuto myšlenku metody lze aplikovat na libovolné atributy protokolů. Druhý výklad metody se zaměřuje na anonymizaci IP adres.

V kontextu IP adres se metoda **prefix preservation** chová obdobně jako v prvním výkladu, ale anonymizaci je podroben i prefix dat. Prefix je transformovaný na novou hodnotu, která nese informace neanonymizovaného prefixu. Dojde tak k jeho změně, ale informace o něm jsou zachovány. Neanonymizované hodnoty se shodným prefixem budou mít po anonymizaci opět shodný prefix.

Sdílí-li dvě neanonymizované IP adresy stejný prefix (např. část adresy označující síť), budou jej sdílet i po anonymizaci, avšak samotná hodnota prefixu bude rozdílná. Dochází tak k aplikaci metody **permutation**, která zachovává prefix sítě. Užívá se funkce `F()`, kterou označujeme za tzv. **prefix-preserving** funkci, pokud pro dvě IP adresy `A` a `B` sdílející prefix `k` jsou výsledkem anonymizace `F(A)` a `F(B)` opět IP adresy, které sdílejí stejný prefix `k`. [11]

Samotný algoritmus metody je označován jako **Crypto-PAn**. Bezpečnosti algoritmu se věnuje [3], který diskutuje jeho odolnost vůči injection útokům.

Suffix preservation

Zachovává-li metoda posledních `k` informací anonymizovaných dat, lze ji označit za metodu **suffix preservation**. Metodu lze užít např. pro anonymizaci části IP adresy označující vlastní síť (netid) a část zařízení (hostid) ponechat beze změn.

Truncation

Metodu **truncation** lze označit za upravené metody **black marker** či **clear**. Metodu považujeme za **truncation**, pokud jejím užitím dochází k odstranění `k` informací zprava. Pro IP adresu si lze představit odstranění několika (`k`) nejméně významných bitů (část obsahující netid). Anonymizováno (nahrazeno defaultní hodnotou) je pouze definovaných `k` dat, ostatní data jsou zachována a dále nejsou zpracována. Lze tak např. úplně odstranit celou část označující síťové zařízení (hostid) z IP adresy a ponechat pouze informace o vlastní síti – tím dochází ke agregaci všech IP adres dané sítě v jednu IP adresu. Její užití tak není zcela destruktivní pro vstupní data, jako tomu je např. u **black marker**. [2]

Reverse truncation

Metoda **reverse truncation** je komplementem metody **truncation**. Opět zde dochází k odstranění `k` informací anonymizovaných dat, avšak zleva. Kromě opačného směru odstraňování informací se metody chovají stejně. Jako příklad užití lze uvést odstranění síťové

části (hostid) z IP adresy. Tento způsob anonymizace může být nedostatečný, je-li znám původ anonymizvaného souboru, jelikož neanonymizované IP adresy mohou být zjistitelné. Metoda také může spojovat více síťových zařízení různých sítí v jedno, což může způsobovat problémy při analýze anonymizovaného souboru. [2]

Enumeration

Metodu **enumeration** lze užít pro seřazená data. Jako příklad lze uvést např. časová razítka příchodu paketu. Metoda zachovává pořadí seřazených dat, ale mění jejich hodnotu. První hodnota seřazených dat může uvádět počáteční hodnotu anonymizovaných dat, přičemž ke každé následující hodnotě je přičten předem definovaný přírůstek. Metoda tak zahazuje originální hodnoty, ale zachovává informace o jejich pořadí. Data atributů zpracovaných touto metodou mohou být zcela zničena. Nelze ji tak použít pro atributy, kde je zapotřebí přesných dat pro následné zpracování (např. časová razítka). [2]

Implementací metody může existovat více. Jejím hlavním cílem ze zatemnit informace, ale zachovat pořadí. Užití metody dává smysl jen pro některé atributy protokolů, převážně pro ty, které definují nějakou sekvenci. Metodu lze také v některých případech označit za **precision degradation** metodu (např. pro časová razítka), jelikož snižuje přesnost anonymizovaných dat. [2]

Partitioning

Metoda **partitioning** nahrazuje skupiny hodnot určených k anonymizaci jednou hodnotou. Všechny dostupné hodnoty anonymizovaného atributu jsou rozděleny do skupin a pro každou skupinu je vybrána hodnota, jež skupinu zastupuje. Během anonymizace dochází k nahrazování hodnot atributů stanovenou hodnotou podle skupiny, do které hodnota atributu spadá. [2]

Metodu lze použít na všechny atributy síťových protokolů. Použitelnost anonymizovaných dat se odvíjí od počtu definovaných skupin. Při užití metody na čísla aplikačních portů tak může docházet např. k zachování skupiny portů – rezervované apod.

Random noise addition

Metoda **random noise addition** vnáší do anonymizovaných dat šum. Metoda je týká převážně čítačů či časových razítek, u kterých se snaží zachovat jejich informaci, ale mírně je pozměnit. Důvodem může být snaha minimalizovat možnosti tzv. **fingerprint** útok[5]. Metoda může modifikovat původní pořadí neanonymizovaných dat. [2]

3.3 Zdroje citlivých dat

Některé protokoly vyskytující se na ve vrstvách TCP/IP modelu přenášejí informace, které se dají v různé míře využít k identifikaci komunikujících stran. Jsou to právě tyto informace, na které klade proces anonymizace důraz. K identifikaci lze užít informace přímo, např. IP adresy určené ke směrování IP datagramů v síti, či nepřímo – specifické charakteristiky jednotlivých operačních systému extrahovatelné z transportních či aplikačních protokolů. Kromě identifikačních atributů může být vhodné anonymizovat i další přenášené informace, jako jsou například certifikáty, technologie užité při ustanovení šifrované komunikace apod.

[5] https://securitytrails.com/blog/cybersecurity-fingerprinting

Při anonymizování jednotlivých atributů je nutné zvážit dopad anonymizace na výsledný paket. Modifikací některých atributů může dojít k porušení komunikačních toků – např. příznaky protokolu TCP. Paket se také může stát nezpracovatelným – např. změna verze IP adresy v protokolu IP.

MAC adresa

MAC adresa se nachází např. v protokolech `Ethernet`, `ARP` či `DHCP`. Adresu je vhodné anonymizovat, jelikož přímo označuje koncové zařízení komunikujících stran, resp. síťové zařízení komunikujících stran.

Každá MAC adresa navíc obsahuje informace o svém výrobci, který by mohl mít dostatek informací k propojení MAC adresy s komunikujícím uživatelem [2]. MAC adresa může být také součástí IPv6 adresy. Adresy lze anonymizovat např. metodami black marker, clear, random permutation či tzv. **structured permutation**, kdy jsou části označující výrobce a zařízení anonymizovány zvlášť.

IP adresa

IP adresu můžeme nalézt hlavně v protokolu `IP`, kde slouží ke směrování jednotlivých IP datagramů. Vyskytuje se i ale např. v protokolech `ARP`, `DNS`. Její anonymizace je vhodná, jelikož je možné ji spojit s určitou organizací či jednotlivými osobami. Jedná se o nejběžnější identifikátor užitý v síťové komunikaci a na prolomení anonymizace tohoto atributu cílí nejvíce útoků. Pro anonymizaci IP adresy existuje velká škála anonymizačních metod. IP adresu lze anonymizovat např. metodami black marker, clear, prefix preservation, permutation, truncation či partitioning.

Time to Live, Hop limit (počet skoků)

Čítače protokolu `IP` verze 4 a 6 označují dobu, kterou paket může strávit v síti, než je odstraněn. Počáteční hodnoty těchto atributů mohou být užity k detekci operačních systémů, určení vzdálenosti (počtu hopů) komunikujících stran a dalších informací. Atributy mohou být použity jako postranní kanály při určování identity komunikujících stran. [22]

Porty

Porty definující aplikace užité na transportní vrstvě modelu TCP/IP je možné anonymizovat. Očekává se, že pokud port paketu obsahuje číslo nějaké známé aplikace, patří tento paket do dané komunikace aplikace [22]. Ve snaze anonymizovat užití aplikace je možné anonymizovat příslušné porty, vhodné je to v případech, kdy aplikace užívá nestandardních portů.

Velikost TCP okénka

Atribut transportního protokolu `TCP` definující maximální počet bytů, které je možné odeslat před získáním potvrzení o jeho doručení. Atribut může být užitý pro získání informací o použitém operačním zařízení na koncovém zařízení, a je tak vhodné jej anonymizovat jakoukoliv dostupnou metodou pro anonymizaci číselných atributů. [22]

Sekvenční čísla

Transportní protokol `TCP` užívá sekvenčních čísel pro zajištění spolehlivé přenosu. Hodnota tohoto atributu se může opět použít pro zjištění operačního systému komunikujících stran. Roli hrají funkce generující náhodná čísla při zahájení komunikace. [22]

Časová razítka

Časová razítka samotných PCAP souborů nebo aplikačních protokolů je vhodné anonymizovat, jelikož mohou prozrazovat informace o době, kdy jsou využívané specifické služby či kdy došlo k útoku na danou síť. Časové období komunikace může také sloužit jako postranní kanál pro identifikaci jednotlivých osob.

Kapitola 4

Návrh

Ačkoliv nástrojů vhodných pro anonymizaci záznamů síťové komunikace existuje vícero, jen některé z nich dávají uživateli možnost ovlivňovat průběh samotné komunikace. Absence možnosti přizpůsobit si anonymizační proces má dopad na výslednou použitelnost anonymizovaných dat pro další zpracování. Nástroje mohou odstranit ze záznamu většinu klíčových dat, a tak mohou být pro některé případy nevhodné.

Následující kapitola se zabývá návrhem aplikace, jejíž cílem je co největší možnost definice anonymizačního procesu, čímž se snaží eliminovat problém nevhodného – předem definovaného – procesu, jenž může vést k úplnému znehodnocení síťových záznamů dat. Aplikace se tohoto cíle snaží dosáhnout pomocí tzv. **anonymizačních politik**, jež definují, na jaké atributy protokolů má být anonymizace aplikována. Anonymizační politika také umožňuje definovat způsob anonymizace, jakým mají být jednotlivé atributy zpracovány. Rozšiřitelnost anonymizačních algoritmů také patří mezi klíčové požadavky.

Navrhovaná aplikace se snaží o co největší podporu síťových protokolů, aby bylo možné anonymizovat co nejširší škálu síťové komunikace, čímž se vymezuje od některých existujících řešení, které pracují s předem definovanou množinou protokolů. S velkou škálou protokolů také úzce souvisí podpora anonymizace na všech vrstvách architektury TCP/IP.

Přehled nutných požadavků navrhované aplikace shrnuje následující seznam:

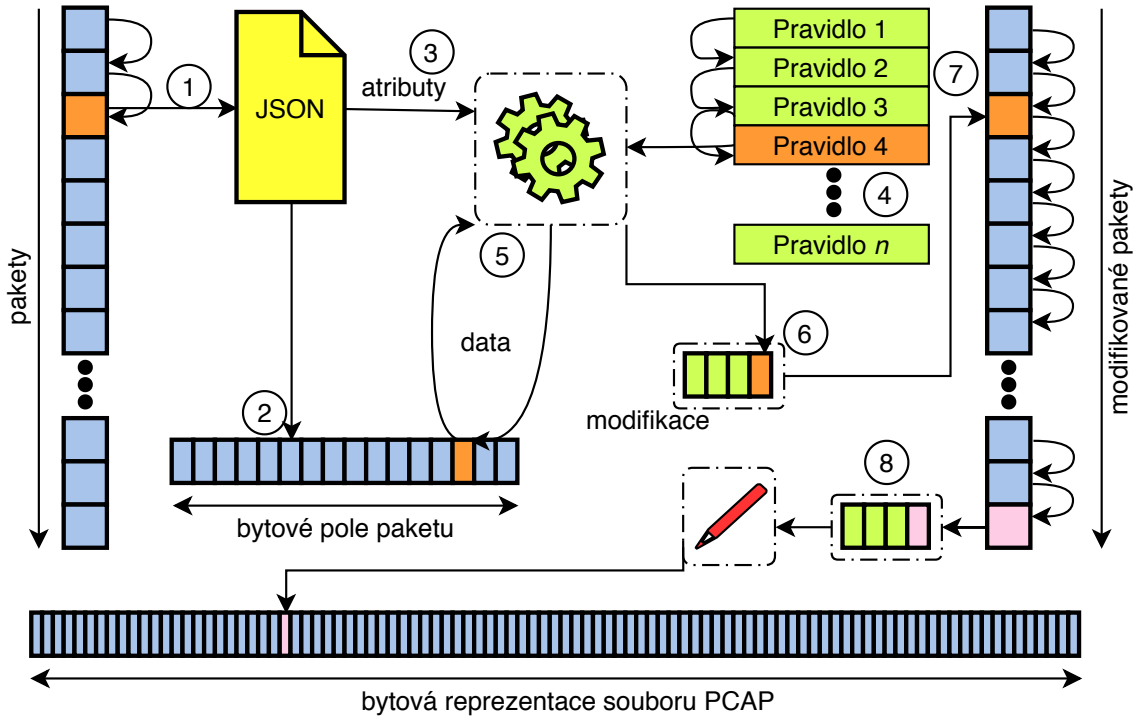
- PCAP jako vstupní i výstupní formát dat,
- anonymizace na všech vrstvách TCP/IP modelu,
- podpora velkého množství protokolů,
- rozšiřitelnost anonymizačních algoritmů,
- vlastní anonymizační politika,
- anonymizace nekompletní TCP komunikace.

Obrázek č. 4.1 představuje schéma navrhované aplikace, na kterém lze představit jednotlivé části aplikace a fáze, kterými pakety během anonymizace procházejí.

Před vlastní anonymizací je nutné získat jednotlivé pakety ze vstupního souboru. Procesem získávání paketů se zabývá sekce 4.2 a na obrázku zahrnuje body **1, 2** a **3**. **Bod 1** představuje seznam paketů, které jsou postupně zpracovávány. Pro každý paket je dostupný JSON objekt, jenž poskytuje informace o daném paketu, viz. sekce 4.3. Z tohoto objektu jsou získána **raw** data paketu, **bod 2**, označovaná za tzv. **bytové pole paketu**. **Bod 3** představuje získané atributy, které jsou anonymizovány.

Bod 4 představuje seznam anonymizačních pravidel, která jsou získána z anonymizační politiky a definují, co a jak bude anonymizováno. Zpracováním politiky a jednotlivých pravidel se zabývá sekce 4.1. Dochází k vytvoření tzv. **struktury seznamu modifikací** (**bod 5**). Seznamu modifikací se věnuje sekce 4.4. Po zpracování anonymizační politiky a vstupního paketu nastává fáze anonymizace pro daný paket, **bod 5**. Jednotlivé atributy (bod 3) jsou anonymizovány pomocí pravidel (bod 4), přičemž hodnota atributů je získávána i nazpět zapisována do bytového pole paketu (bod 2). Anonymizací se zabývá podsekce 4.3.1. Výstupem každého anonymizace pravidla je tzv. **modifikace** (**bod 6**), jenž obsahuje informace o atributu a jeho nové hodnotě. Jednotlivé modifikace jsou ukládány do struktury seznamu modifikací zpracovaných paketů (**bod 7**).

Po anonymizaci **všech** paketů dochází k validaci modifikací, jejich seřazení a následné aplikaci změn do výstupního souboru (**bod 8**). Samotnému zápisu změn se věnuje sekce 4.5.



Obrázek 4.1: Schéma architektury navrhované aplikace

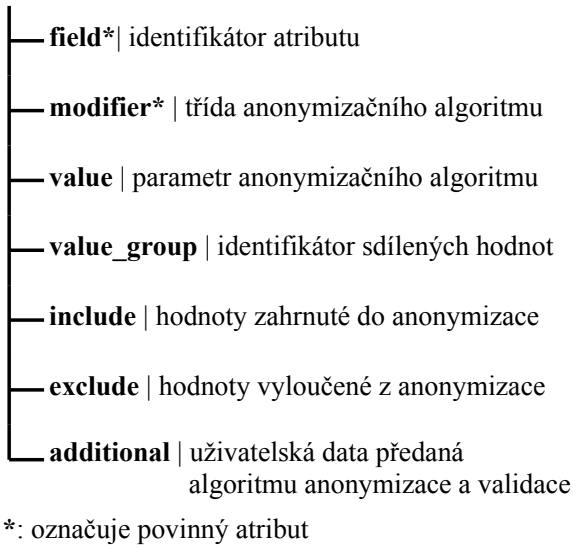
Sekce 4.7 se věnuje anonymizaci transportního protokolu **TCP**, jelikož zajištění spolehlivého přenosu přináší z pohledu anonymizace jisté problémy, které je nutné zahrnout do jednotlivých fází anonymizace. Problémům anonymizace časových razítek se věnuje sekce 4.6. Sekce 4.8 se zabývá globálním nastavením anonymizace a formátem konfiguračního souboru.

4.1 Anonymizační politika

Jedním z hlavních požadavků na výslednou aplikaci je možnost definovat jaké atributy protokolů se mají anonymizovat a také jaká technika má být k anonymizaci použita. Tento požadavek aplikaci značně odlišuje od existujících řešení, která umožňují malou až nulovou

možnost vlastní konfigurace. Existuje-li možnost konfigurace, vyskytuje se většinou ve výběru anonymizační techniky pro předem definované atributy protokolů, přičemž možnost definovat nové atributy k anonymizaci většinou chybí.

Navrhovaná aplikace se tento problém snaží řešit implementací tzv. **anonymizačních politik**. Sémanticky představuje anonymizační politika seznam **anonymizačních pravidel**, která jsou postupně aplikovaná na vstupní soubor pcap. Vytvoření anonymizační politiky je plně v režii uživatele aplikace, čímž mu dává možnost anonymizovat vstupní soubory takovým způsobem, aby docházelo k co nejmenšímu znehodnocení anonymizovaných dat vzhledem k jejich budoucímu užití.



Obrázek 4.2: Struktura anonymizačního pravidla

Pro zajištění co největší možnosti konfigurace anonymizační politiky jsou na její části, pravidla, kladeny jisté požadavky. Každé anonymizační pravidlo musí definovat, kterého protokolového atributu se týká. Jméno atributu se uvádí v položce `field` a nabývá hodnot užívaných nástroji TShark či Wireshark při tvorbě filtrovacích pravidel.

Jelikož neexistuje žádná, předem definovaná vazba mezi jednotlivými atributy a anonymizujícími algoritmy, je nutné definovat, jakým způsobem bude vybraný atribut protokolu zpracován. Pro definici algoritmu slouží položka `modifier`, jež nabývá hodnot z množiny anonymizačních modifikátorů, které jsou v aplikaci dostupné. Od volby anonymizačního modifikátoru se odvíjí následující položka pravidla a to `value`. Položka `value` již nepatří mezi povinné části pravidla, jako tomu bylo u předešlých dvou položek, a její obsah je přímo závislý na užitém modifikátoru. Jedná se o parametr nutný pro správnou funkci anonymizačního algoritmu zvoleného modifikátoru. Jako příklad lze uvést IP adressu `255.255.255.255` při užití algoritmu **IP black marker**, kde položka `value` zastupuje statickou hodnotu, kterou jsou nahrazeny atributy protokolu definované položkou `field`. Při užití algoritmu **k-prefix preservation** hodnota `value` naopak udává počet bitů, které mají zůstat zachovány.

Uplatnění pravidla na zvolený atribut lze také specifikovat pomocí dvou nepovinných položek `exclude` a `include`. `Exclude` slouží k definování hodnot, které nemají být připuštěny k anonymizaci. Položka `include` má funkci opačnou, a to definici hodnot, které jsou k anonymizaci připuštěny. Uvést lze jednotlivé hodnoty, případně intervaly. Mezi položkami

`exclude` a `include` platí následující vztahy. Je-li definovaná položka `include`, je anonymizace vykonaná pouze nad hodnotami atributu, které patří do dané definice, nevyhovující hodnoty jsou vynechány. Není-li položka definovaná, anonymizace probíhá nad všemi hodnotami atributu. Položka `exclude` naopak definuje hodnoty, které nejsou k anonymizaci připuštěny. Položky `include` a `exclude` je možné použít současně, a existuje-li mezi nimi průnik, hodnoty atributů tohoto průniku nejsou k anonymizaci připuštěny. Příklad užití obou položek je možné vidět na obrázku č. 4.3, kde je definované pravidlo pro atribut `ip.src`. Položka `include` pravidla definuje, že anonymizace má proběhnout pouze pro síť `192.168.1.0/24`. Ovšem z anonymizace je vyloučena IP adresa `192.168.1.255` díky definici položky `exclude`. Anonymizované tak budou všechny IP adresy uvedené sítě, kromě adresy `192.168.1.255`.

```
-
 field: ip.src
 modifier: IPRandom
 value_group: IP
 include: ['192.168.1.0/24']
 exclude: ['192.168.1.255']
-
 field: ip.dst
 modifier: IPRandom
 value_group: IP
```

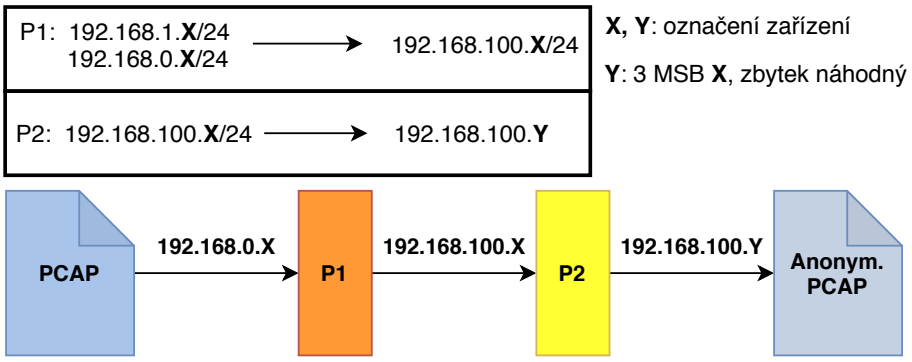
Obrázek 4.3: Definice dvou pravidel demonstrujících užití atributů `include`, `exclude` a `value_group`.

Způsob definice umožňuje rozdílný způsob anonymizace stejných protokolových atributů, jelikož pro jeden atribut může existovat více než jedno pravidlo. Lze tak např. definovat anonymizaci IP adres interní sítě jiným způsobem než IP adres externích sítí. Je třeba dbát pozornosti při definování položek `exclude` a `include`, jelikož všechna pravidla jsou postupně aplikována na vstupní soubor a jejich vyhodnocení má mutující efekt. Výstup jednoho anonymizačního algoritmu je vstupem pro následující anonymizační algoritmus, jsou-li oba definované pro stejný atribut.

Příklad mutujícího efektu lze najít na obrázku č. 4.4, který demonstruje následující scénář. Chceme anonymizovat IP adresy dvou sítí tak, že je nejdříve převedeme na jednu společnou síť, přičemž část značící zařízení zůstane nezměněna (pravidlo P1). Následně chceme adresy sítě, do které byly dvě předešlé sítě sloučeny, anonymizovat tak, aby zůstala zachovaná celá část sítě včetně prvních tří bitů části adresy označující zařízení zařízení (`hostid`), které mají pro budoucí užití dat strategický význam (pravidlo P2), přičemž ostatní bity jsou náhodně změněny. Hodnoty `X` a `Y` představují adresu síťového zařízení před, resp. po anonymizaci jednotlivými pravidly.

Poslední položkou definující anonymizační pravidlo je `value_group`, která je opět nepovinná. Položku lze užít pro zajištění stejných anonymizovaných hodnot atributu, který se vyskytuje ve stejném, případně jiném protokolu vícekrát. Dochází tak ke sdílení anonymizovaných hodnot mezi více atributy. Jako příklad lze uvést anonymizaci zdrojové a cílové IP adresy protokolu IP za užití protokolu **IP Random**, viz obrázek č. 4.3. Užití položky `value_group` způsobí sdílení anonymizovaných hodnot mezi atributy `ip.src` a `ip.dst`. Je-li anonymizovaná zdrojová adresa (`ip.src`) `A` hodnotou `A'`, pak je stejná anonymizovaná

hodnota `A'` užita pro cílovou adresu (`ip.dst`) `A`. Bez užití položky `value_group` není toto chování zajištěno, a IP adresa `A` tak může mít dvě anonymizované hodnoty – jednu pro `ip.src` a druhou pro `ip.dst`. Pro správnou funkci položky `value_group` je nutné použít pro všechny propojené atributy stejný `modifikátor`, aby nedocházelo k porušení definice anonymizační politiky.



Obrázek 4.4: Demonstrace pořadí vyhodnocování anonymizačních pravidel. Pravidlo **P1** spojuje dvě sítě v jednu a zachovává adresu síťového zařízení. Pravidlo **P2** zachovává první 3 MSB zařízení, zbytek doplní náhodně.

Položka `additional` významově nepatří mezi definici anonymizačního pravidla, nicméně je možné ji specifikovat pro každé pravidlo. Její účel je definice uživatelných dat, která jsou předána anonymizačnímu a validačnímu algoritmu.

Zpracování anonymizační politiky je první fází zpracování vstupních dat. Jednotlivá pravidla jsou převedena do interní struktury a dochází k ověření platnosti pravidel. Validuje se přítomnost definovaných modifikátorů v položkách `modifier` a také jejich případná shoda u pravidel, které sdílejí anonymizované hodnoty pomocí `value_group`. Objeví-li se v této fázi chyba, je anonymizace předčasně ukončena, jelikož definovaná anonymizační politika není platná.

4.2 Zpracování vstupních dat

Předchozí sekce řešila způsob definice anonymizační politiky, která je jedním ze dvou nutných vstupů aplikace. Druhým vstupem jsou vlastní soubory ve formátu **libpcap** dále jen **pcap**, na které je definovaná anonymizační politika aplikována. Samotnému formátu pcap byla věnována vlastní sekce 2.2.

Anonymizační politika, viz sekce 4.1, umožňuje vybrat k anonymizaci libovolný atribut různých protokolů, přičemž je jedno, na které vrstvě síťového modelu se protokol nachází. Z pohledu bytového pole, které tvoří samotný paket, se tak atribut může nacházet kdekoliv. Ačkoliv se pozice některých atributů jeví jako fixní, např. zdrojová IP adresa, může být její reálná pozice v bytovém poli ovlivněna např. zapouzdřením v tunelovacím protokolu.

Nabízí se tak použití existujícího nástroje, který dokáže bytové pole paketu zpracovat a zajistit informace o použitých protokolech a jejich atributech. Tímto přístupem lze zajistit velkou podporu protokolů, které navrhovaná aplikace zvládne anonymizovat. Zpracování vstupního souboru externím nástrojem také může napomoci při zpracování TCP komunikace, která může být rozprostřena přes více paketů.

Nástrojů vhodných pro zpracování existuje vícero a zváženy byly tři. Tabulka č. 4.1 představuje jejich výhody a nevýhody. Při výběru nástroje byly zváženy následující parametry:

- Počet podporovaných protokolů – Počet protokolů, které je nástroj schopný zpracovat, přímo definuje počet protokolů, které bude možné pomocí aplikace anonymizovat.
- Streamové zpracování – Schopnost daného nástroje zpracovávat a předkládat informace o jednotlivých paketech postupně, bez nutnosti zpracování celého souboru naráz, aby došlo k minimalizování paměťové složitosti aplikace. Je hodnoceno, zda nástroj přímo poskytuje rozhraní, které lze užít přímo v navrhované aplikaci. Implementace této funkcionality, např. pomocí přesměrování standardního výstupu nástroje, zde není zvážena.
- Stabilita – Zkoumá, zda je nástroj stabilní. Jelikož případné pády nástroje způsobí pád navrhované anonymizační aplikace.
- Integrace nástroje – Jak náročné bude integrovat nástroj do navrhované aplikace. Zkoumáno je, zda je nástroj pouze knihovnou či např. konzolovou aplikací.
- Udržovatelnost – Udržovatelnost zkoumá, zda je nástroj aktivně udržovaný. Zda jsou opravovány chyby a přidávány nové protokoly, což má vliv na počet podporovaných protokolů a stabilitu.

Z výběru byl vyloučen nástroj Scapy kvůli nízké podpoře protokolů navzdory dobré stabilitě a streamovému zpracování. Nástroj PyShark se jeví jako nejlepší varianta pro předzpracování paketů, jelikož se jedná o knihovnu v jazyce Python nad nástrojem TShark, čímž získává stejnou podporu protokolů jako nástroj samotný. Ovšem knihovna je málo aktualizovaná a během experimentování způsobovala neočekávané pády.

Jako finální nástroj je tak zvolen nástroj TShark, který podporuje přibližně 3000 protokolů, je aktivně udržovaný a během experimentování se nevyskytly problémy se stabilitou. Nedostatkem nástroje je chybějící rozhraní pro streamové zpracování. Jelikož se jedná o konzolový nástroj, není integrace do navrhované aplikace tak snadná jako v případě knihoven Scapy nebo PyShark. Užití nástroje také přidává režii nutnou k manipulaci s výstupními daty, jež knihovny řešily díky dostupnému API, jako příklad lze uvést vyhledávání jednotlivých atributů nebo zápis zpracovaných paketů. Zmíněné nedostatky je tak nutné řešit ve vlastní aplikaci dodatečně, což snižuje pohodlnost jeho užití.

	Scapy	**PyShark**	**TShark**
Podporované protokoly	47[1]	cca. 3000[2]	cca. 3000[3]
Streamové zpracování	Ano	Ano	Ne
Stabilita	Stabilní	Nestabilní	Stabilní
Integrace nástroje	Knihovna	Knihovna	CLI nástroj
Udržovanost nástroje	Dobrá	Špatná	Dobrá

Tabulka 4.1: Srovnání nástrojů ke zpracování souborů pcap. Jako parametr pro udržovatelnost nástroje byla zvolena udržovanost repozitáře zdrojových kódů.

[1] https://scapy.readthedocs.io/en/latest/api/scapy.layers.html
[2] PyShark podporuje stejné množství protokolu jako TShark, jelikož jej interně používá
[3] https://www.wireshark.org/docs/dfref/

Vybraný nástroj TShark, popsaný v sekci 2.2, umožňuje zpracovaný pcap soubor exportovat do formátu JSON, který byl zvolen pro návrh aplikace. Ovšem exportovaný soubor dosahuje pro větší pcap soubor obřích rozměrů. Pro pcap soubor o velikost 477 MB získáme po zpracování soubor JSON o velikosti 20 GB. Z tabulky 4.2 je patrné, že závislost mezi vstupním a výstupním problémem je lineární a pro uvedené hodnoty je výsledný soubor průměrně 43krát větší. Velikost výstupního souboru je značně ovlivněna obsahem zpracovávaného vstupního souboru, a tak se uvedené hodnoty mohou lišit pro jiný vstupní soubor. Například reprezentace segmentované TCP komunikace, jejíž data jsou ve výstupním souboru značně duplikována, má velký vliv na velikost výstupního souboru.

PCAP [MB]	**JSON [MB]**	**Koeficient nárůstu velikosti**
447	20 480	42,9
246	10 752	43,7
120	5 324	44,4
62	2 662	42,9
31	1 331	42,9
15	593	39,6
8	307	38,4
Průměr		43,1

Tabulka 4.2: Srovnání velikostí vstupních (PCAP) a výstupních (JSON) souborů po zpracování nástrojem TShark.

Velikost souboru přináší dva problémy. Prvním problémem je nutnost dostatku volného místa na disku pro předzpracování pcap souboru ještě před začátkem vlastní anonymizace. Druhý problém tvoří zpracování tak velkého souboru JSON. Jelikož je nutné nástroj TShark volat z programovacího jazyka výsledné aplikace, je možné vyhnout se ukládání souboru JSON na disk a namísto toho jej celý načíst do paměti aplikace, což jen přesouvá problém dostupné paměti na jiné místo.

K řešení problémů napomůže fakt, že nástroj TShark během zpracovávání souboru zapisuje výsledný JSON soubor na svůj standardní výstup. Výstup lze přesměrovat do navrhované aplikace a zde jej zpracovat. Aby nedošlo opět k problému ukládání celého souboru do operační paměti, je nutné soubor JSON zpracovávat postupně, nejlépe paket po paketu. Je tedy vhodné zavolat nástroj TShark z výsledné aplikace neblokujícím způsobem a následně přesměrovat jeho standardní výstup. V jazyce Python lze zvolit knihovnu **asyncio**[4], jež zmíněné požadavky splňuje. Namísto celého JSON souboru, který lze načíst a zpracovat, nyní pracujeme s proudem (streamem) znaků, které soubor tvoří.

Nastává tak další problém, jak z proudu znaků získat jednotlivé pakety, které jsou ve formátu JSON. K jeho řešení je užita Python knihovna **JsonSlicer**[5] podporující proudové zpracování souborů JSON, interně užívající knihovnu **YAJL**[6] napsanou v jazyce C.

Knihovna JsonSlicer umožňuje definovat vzor (pattern), který je ve vstupním proudu znaků vyhledáván. Narazí-li knihovna na objekt odpovídající vzoru, je celý zpracován a vrácen volající aplikaci. Definujeme-li vzor tak, aby odpovídal jednotlivým paketům souboru JSON, lze získat z proudu dat produkovaných nástrojem TShark postupně jednotlivé pakety a ty sekvenčně zpracovávat, aniž by docházelo k ukládání celého souboru JSON na disk

[4] https://docs.python.org/3/library/asyncio.html
[5] https://pypi.org/project/jsonslicer/
[6] https://lloyd.github.io/yajl/

či do operační paměti. Na obrázku č. 2.10 můžeme vidět, že paket je definovaný objektem s názvem `layers`, užitý vzor pro knihovnu JsonSlicer je tak `(None, None, 'layers')`.

Kromě samotných paketů je nutné v této části zpracování získat informace o samotném pcap souboru. Jedná se především o informace z globální hlavičky pcap souboru, viz sekce 2.2. Extrahované informace, endianita souboru a přesnost časových razítek, jsou nutné pro anonymizaci časových razítek jednotlivých paketů.

Během zpracovávání jednotlivých paketů je nutné si udržovat informace o aktuální pozici vůči originálnímu pcap souboru pro pozdější zápis změn. Umístění paketu lze iterativně počítat na základě velikosti paketové hlavičky `16B` a velikosti zpracovávaného paketu, kterou lze nalézt v již zmíněné paketové hlavičce, případně ve vrstvě `frame_raw`, která představuje hexadecimální reprezentaci celého paketu. Umístění první paketu je ovlivněno globální hlavičkou souboru, jejíž velikost je `24B`.

4.3 Zpracování paketu

Po úspěšném zpracování anonymizační politiky a vytvoření komunikačního kanálu mezi aplikací a nástrojem TShark zpracovávajícím anonymizovaný soubor, lze přistoupit ke zpracovávání jednotlivých paketů komunikace. Cílem této fáze je vyhledání protokolových atributů určených k anonymizaci a extrakce informací nutných v dalších fázích zpracování. Následující seznam představuje některé z extrahovaných informací. Důvody k extrakci jednotlivých informací jsou postupně popsány v následujících sekcích. V této sekci se důraz klade na vyhledávání protokolových atributů:

- užité protokoly,
- atributy určené k anonymizaci,
- paket v bytové formě (data nezpracovaného paketu),
- bytová pcap hlavička paketu,
- bytová délka,
- všechny vrstvy byly zpracovány,
- informace specifické pro TCP protokol.

Jak je již možné vidět na obrázku č. 2.8, jednotlivé protokoly zpracovaného paketu jsou rozděleny do vlastních objektů. Ke každému objektu protokolu i atributu existuje objekt obohacený o příponu `_raw`, který nese informace o protokolu s ohledem na nezpracovaný paket (ten si lze představit jako pole jednotlivých bytů). Definuje-li položka `field` anonymizačního pravidla atribut `eth.dst`, je nutné tento atribut najít ve zpracovaném JSON objektu a získat informace o jeho pozici v bytovém poli. Jména hledaného atributu a JSON objektu jsou vždy identické. Informace ohledně umístění obsahuje objekt s příponou `_raw`. Objekt, který aplikace hledá, tak není `eth.dst`, nýbrž `eth.dst_raw`.

V ideálním případě definuje název protokolového atributu, rozdělený podle tečky na jednotlivé úrovně zanoření, průchod objektem JSON, na jehož konci se nachází objekt hledaného atributu. Ovšem na tento předpoklad nelze u všech atributů spoléhat, jelikož některé protokolové atributy jsou v poskytnutém JSON objektu zanořeny hlouběji či naopak vynechávají několik úrovní zanoření. Jako příklad lze uvést atribut `tls.handshake.type`,

který je v objektu JSON zanořený o jednu úroveň níže, a to v objektu `record`, čili je jeho cesta JSON objektem `tls.record.handshake.type`, což již neodpovídá definovanému atributu `tls.handshake.type`. Přidaná úroveň zanoření se ale ve jménu objektu neprojeví, a tak je hledaný objekt stále `tls.handshake.type_raw`.

Části jména atributu tak nelze přímo užít k lokalizaci stejnojmenného objektu JSON. Na každé úrovni zanoření je nutné ověřit, zda se zde hledaný atribut již nenachází (může být vynecháno několik úrovní zanoření). Pokud se zde nenachází, je nutné ověřit, jestli se zde nachází objekt představující další úroveň zanoření (pro `tls.handshake.type` a aktuální zanoření `tls` zjišťujeme, zda lze dále pokračovat s klíčem `tls.handshake`). Pokud ano, postoupíme v hledané cestě a zanoříme se do další vrstvy. Pokud ne, je nutné navštívit všechny objekty daného zanoření, jelikož atribut může být zanořený v některém z nich, ač to nedefinuje cesta (pro `tls.handshake.type` a zanoření `tls` neexistuje objekt `tls.handshake`, jelikož je zanořený až v objektu `record` vyskytující se ve vrstvě `tls`).

Užití první části názvu protokolového atributu jako označení protokolu nemusí být vždy dostatečné. Pro atribut s názvem `http.response.line` se usuzuje, že se bude modifikovat atribut protokolu **HTTP**. Ovšem protokol HTTP může být užitý i jako součást jiných protokolů, např. protokolu **SSDP**. V takovém případě lze ve zpracovaném JSON objektu nalézt mezi objekty zpracovaných protokolů pouze protokol SSDP, ve kterém je protokol HTTP zapouzdřen. Řešením je neuvažovat první část jména atributu za definici protokolu, a tím pádem procházet všechny dostupné protokoly. Vyhledávání atributu je tak aplikováno na všechny dostupné protokoly, což vede ke zvyšování doby nutné k jejich nalezení. Existuje tedy možnost, zda tuto funkcionalitu pro daný běh anonymizace souboru povolit či nikoli, a lze ji nalézt v konfiguračním souboru.

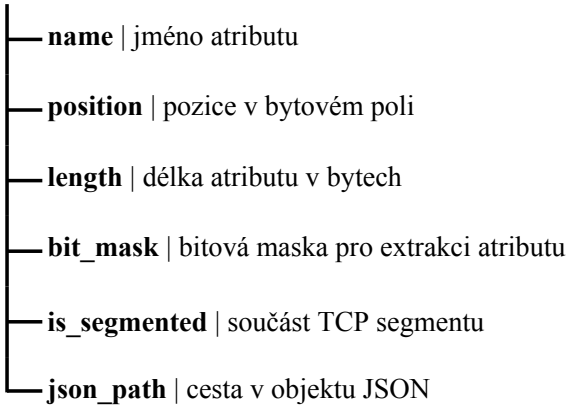
Prohledávání objektů, které nejsou uvedené ve jménu atributu, je označeno za **wildcard**. Užití wildcard značně zpomaluje proces vyhledávání, nicméně je jediným způsobem, jak vyhledat atributy, které jsou zanořené jinak, než definuje jejich název. Příklady možných odlišností, mezi cestou definovanou jménem atributu a jeho skutečným umístěním, demonstruje tabulka č. 4.3.

Varianta	**Jméno atributu**	**Očekávaná cesta**	**Reálná cesta**
Stejně zanořený	eth.dst	eth	eth
Více zanořený	tls.handshake.type	tls>handshake	tls>handshake>record
Méně zanořený	dhcp.ip.your	dhcp>ip	dhcp
HTTP v SSDP	http.host	http	ssdp>http

Tabulka 4.3: Přehled možných odlišností jména atributu od jeho reálného výskytu v objektu JSON.

Obrázek č. 2.11 představuje jednotlivé atributy jako struktury, ovšem v některých případech je pod jedním jménem atributu zahrnuto více objektů v podobě pole struktur. Toto chování je explicitně aktivováno pomocí přepínače `-no-duplicate-keys` nástroje TShark, jinak dochází k zahazování více hodnot pro jeden atribut. Tato situace může nastat během vyhledávání atributů na jakékoli úrovni zanoření. Příkladem je atribut `http.response.line`, který zahrnuje všechny řádky odpovědi protokolu HTTP. Jev lze také pozorovat u segmentované TCP komunikace, kde je v rámci zpracování jednoho paketu obsaženo více aplikačních protokolů. Obdobné chování přináší i tunelovaná komunikace, kde lze nalézt více protokolů např. síťové vrstvy – dva protokoly IP apod.

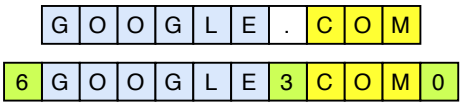
Nalezené atributy určené k modifikaci jsou převedeny do interní struktury. Strukturu lze vidět na obrázku č. 4.5, kde si lze všimnout, že interní struktura obsahuje informace o délce



Obrázek 4.5: Interní reprezentace anonymizovaného atributu

či pozici daného atributu, ale neobsahuje vlastní hodnotu. Hodnota atributu není ukládaná z důvodu možného výskytu více anonymizačních pravidel pro jeden atribut, viz sekce 4.1, které postupně mutují hodnotu. Namísto vlastní hodnoty atributu je díky informacím o jeho umístění v bytovém poli užita při každém dotazu na hodnotu aktuální hodnota z bytového pole. Bytové pole představující celý rámec paketu lze nalézt v JSON objektu s názvem `frame_raw`.

Je možné si všimnout, že informace o datovém typu atributu není též ukládána. Důvodem je nespolehlivost této informace pro účely převodu bytové reprezentace na zpracovanou hodnotu a naopak. Jako příklad lze uvést atribut `dns.qry.name`, který nabývá datového typu řetězec. Budeme-li předpokládat uložení řetězce v bytovém poli jako posloupnost bytů jednotlivých znaků, získáme při zpracování bytové reprezentace neplatnou hodnotu, jelikož zmíněný atribut užívá jiné bytové reprezentace. Obrázek č. 4.6 demonstruje způsob bytové reprezentace atributu `dns.qry.name`, kde lze vidět, že každá část doménového jména je předcházena délkou této části.



Obrázek 4.6: Bytová reprezentace atributu `dns.qry.name`. Každá část doménového jména je předcházena její délkou.

Po ukončení fáze zpracování paketu jsou již dostupné všechny informace uvedené na začátku této sekce a je možné přistoupit k samotné anonymizaci jednotlivých atributů.

4.3.1 Anonymizace atributu

Po extrahování informací o paketu do interní reprezentace a zpracování anonymizační politiky, má aplikace dostupná všechna data k samotné anonymizaci. Jednotlivá anonymizační pravidla jsou sekvenčně procházena a podle jejich identifikátoru `field` jsou ve zpracovaném paketu vyhledávány příslušné atributy. Jsou-li atributy nalezeny, dochází k vlastní anonymizaci, jinak se pokračuje následujícím pravidlem.

Jak již bylo zmíněno v předešlé sekci 4.3, interní reprezentace atributu neobsahuje hodnotu daného atributu, ale pouze informace o jeho pozici v bytovém poli celého rámce paketu.

K tomuto přístupu existují dva důvody. První důvod byl již uveden a jedná se o sekvenční zpracování pravidel, které se mohou týkat stejného atributu a je nutné zajistit, aby výstup jednoho anonymizačního algoritmu byl vstupem pro druhý. Druhý důvod přináší anonymizace atributů, jejichž délka je menší než 1 B a užívají tak bitových masek pro sdílení tohoto bitového prostoru s dalšími atributy. V obou případech by došlo k porušení očekávaného toku dat atributu skrz pravidla. Je tedy nutné nejprve získat hodnotu atributu určenou k anonymizaci, a to načtením příslušného počtu bytů `length` z pozice `position` bytového pole paketu. Jedná-li se o atribut s bitovou maskou, je provedena extrakce vlastní hodnoty pomocí dané masky. Po získání hodnoty následuje fáze validace atributu, která je popsána v podsekci 4.3.2. Prozatím předpokládejme, že validace proběhla úspěšně.

Každé anonymizační pravidlo obsahuje modifikátor, který je zodpovědný za samotnou anonymizaci atributu. Modifikátor by měl implementovat předem definované rozhraní, aby jej byla aplikace schopna užít. Aplikace očekává implementaci anonymizační metody.

Od metody se očekává, že má její návratová hodnota stejnou délku jako vstupní data atributu a že představuje anonymizovaná data ve formátu, který je možné okamžitě zapsat nazpět do bytového pole – tedy pole bytů. Předpokládá se i druhý typ návratové hodnoty (např. `None` pro Python 3), který říká, že hodnota atributu nebyla připuštěna k anonymizaci. Této funkcionality lze užít v případě, že definice položek `include` a `exclude` zcela nepokrývají požadavky na validaci hodnot atributů. Jako příklad lze uvést, že chceme anonymizaci aplikovat pouze na pakety číslo 10 a 20. Anonymizační metoda má tak možnost vyloučit z anonymizace hodnoty atributů i přesto, že úspěšně prošly fází validace, viz podsekce 4.3.2 .

Po získání anonymizované hodnoty dochází k jejímu zapsání nazpět do bytového pole paketu. Pro zajištění zapsání správné hodnoty je původní hodnota v bytovém poli nejdříve vynulována a až následně dochází k uložení hodnoty nové. Důvodem je např. zápis číselných hodnot atributů, jichž délka je 2 B, ale nová hodnota zabírá pouze 1 B. Naopak anonymizovaná data delší než data původní jsou zkrácena. Tímto je ošetřen stav, kdy anonymizační funkce nevrací data správné délky. Hodnota je následně uložena do tzv. **poolu**, kde jsou mapovány originální hodnoty na hodnoty anonymizované. Důvodem je např. zajištění užití stejných anonymizovaných hodnot pro již anonymizované atributy při užití algoritmu **random** či vygenerování anonymizačního logu. Každé pravidlo má svůj vlastní pool. Výjimkou jsou pravidla, která sdílí hodnoty pomocí položky `value_group`. Posledním krokem anonymizace atributu je vytvoření tzv. **modifikace**. Modifikace obsahuje informace nutné k zápisu anonymizované hodnoty do anonymizovaného souboru a je jí věnována sekce 4.4.

4.3.2 Validace atributu

Před samotnou anonymizací nalezených atributů dochází k validaci jejich hodnot z důvodů splnění podmínek definovaných anonymizačním pravidlem v položkách `exclude` a `include`. Sekce 4.3 uvádí, že datový typ jednotlivých atributů není důvěryhodný, a tak nelze jednotlivé atributy validovat automaticky bez znalosti jejich bytové reprezentace. Způsob validace jednotlivých atributů se může sémanticky lišit i pro atributy stejných datových typů. Validaci číselných atributů či IP adres lze zobecnit na příslušnost do definovaných intervalů či sítí, případně na rovnost definovaných a validovaných hodnot. Textové řetězce ale přidávají další způsoby validace, a to shodnost pouze částí hodnot – prefix či sufix.

Jelikož je validace atributu úzce spjata s jeho anonymizací, je způsob validace definovaný v modifikátoru. Aplikace pro validaci hodnoty atributu užívá předem definovanou

metodu rozhraní modifikátoru. Výběrem modifikátoru pro anonymizovaný atribut v rámci anonymizačního pravidla tak dochází i k výběru způsobu validace.

Obrázek č. 4.7 představuje způsob definice položek `include` a `exclude`, které demonstrují různé konfigurace validace. Je-li uvedeno pouze pole hodnot, atribut `exclude`, dochází k validaci, zda hodnota atributu spadá mezi definované hodnoty. Ovšem pokud to validační metoda umožňuje, je možné definovat i položku `validation`, případ definice `include`, která ovlivňuje způsob validace. V případě položky `include` obrázku č. 4.7 dochází ke kontrole, zda hodnota atributu začíná některým z řetězců uvedených v `value`.

```
field: http.host
modifier: HttpTextRandom
include:
   value: ['Host: www.foo.com']
   validation: prefix
exclude: ['Host: www.foo.com']
```

Obrázek 4.7: Definice různých způsobů užití atributů `include` a `exclude`.

Obrázek č. 4.8 demonstruje zápis intervalu číselných hodnot, které mají být připuštěny k anonymizaci. Tento formát zápisu je nutné převést do interní reprezentace, aby s ním mohla validační metoda pracovat. Důvodem je, kromě transformace dat, také zajištění, že transformace proběhne při fázi zpracování anonymizační politiky ještě před vlastní anonymizací, čímž je vykonána pouze jednou. Pokud by byla transformace součástí validační metody, docházelo by ke zpracovávání položek `include` a `exclude` při každém volání této metody, což je vzhledem k počtu možných validací neefektivní.

```
include: [[0, 1024], 8080]
```

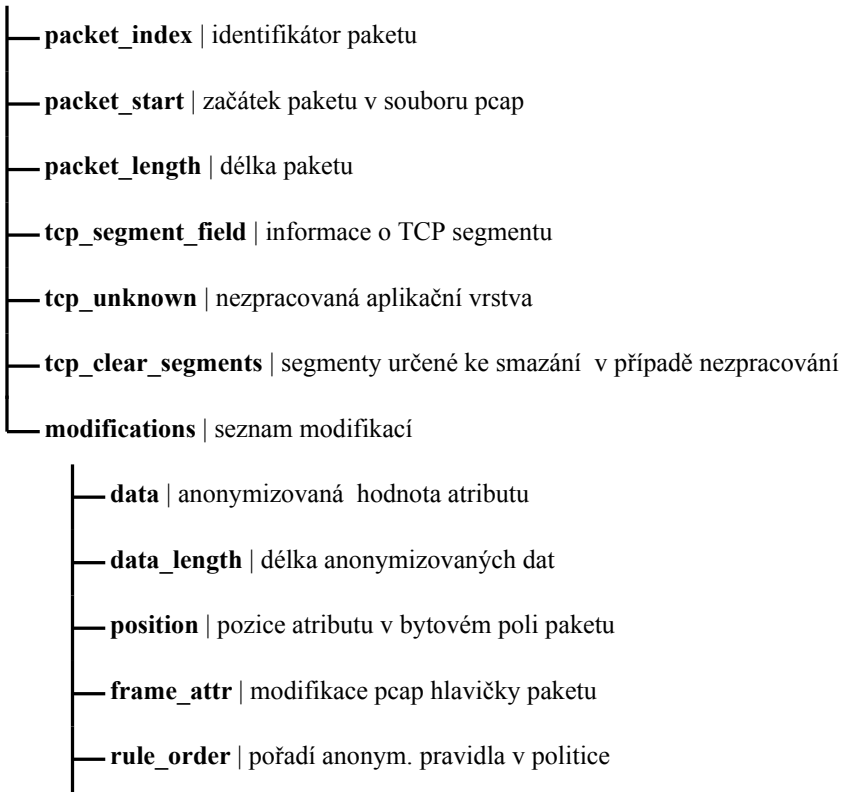
Obrázek 4.8: Příklad zápisu intervalu číselných hodnot pro položky `include` a `exclude`.

Podsekce 4.3.1 uvádí, že každý atribut, který byl anonymizovaný, je uložen do poolu daného pravidla. Ukládá se informace o mapování vstupní (původní) hodnoty na výstupní (anonymizovanou) hodnotu. Tato skutečnost přináší další úroveň validace atributu před vlastní anonymizací – dochází k ověření, zda byla vstupní hodnota atributu již anonymizována. Pokud byla anonymizace vstupní hodnoty již provedena, je odpovídající anonymizovaná hodnota nalezena v poolu a dochází pouze k vytvoření nové modifikace, aniž by došlo k opakované (zbytečné) anonymizaci.

4.4 Seznam modifikací

Během fáze anonymizace jednotlivých atributů paketu dochází k vytváření tzv. seznamu modifikací, viz obrázek č. 4.9. Seznam modifikací je struktura, která obsahuje nutné informace o paketu a anonymizovaných atributech nutných k zápisu zpět do souboru pcap. Některé informace jsou pouze zkopírovány z původní interní reprezentace zpracovávaného paketu – např. začátek paketu a jeho délka – jiné jsou postupně dynamicky doplňované během anonymizace – samotné modifikace. Důvodů k vytvoření seznamu modifikací namísto okamžitého zápisu anonymizovaných hodnot existuje několik.

Prvním způsobem, jak aplikovat změny způsobené anonymizací, je ukládat výsledky anonymizačních pravidel hned po jejích vyhodnocení do souboru pcap. Nebylo by tak nutné



Obrázek 4.9: Struktura seznamu modifikací

vytvářet seznam modifikací. Ovšem tento způsob přináší problémy. Jednotlivé atributy nemusí být seřazeny podle jejich umístění v bytovém poli paketu, a tak by při zápisu změn mohlo docházet ke zbytečným přesunům souborového ukazatele. Další problémy přináší anonymizace atributů, které sdílí 1B, jelikož by mohlo docházet k přepisu anonymizovaného atributu neanonymizovaným.

Další způsobem aplikace změn je zápis celého modifikovaného pole bytů paketu po skončení jeho anonymizace. Po aplikaci daného pravidla je modifikované pole bytů, které slouží jako zdroj dat, tudíž je jej možné zapsat do souboru, čímž jsou všechny anonymizace aplikované.

Oba zmíněné způsoby značně komplikuje návrat k již anonymizovaným paketům, který je nutný např. u paketů komunikace TCP. Zápis změn navíc probíhá ještě před koncem anonymizace celého souboru, která nemusí skončit úspěšně, a tak je zbytečné aplikovat pouze částečné změny.

Hlavním důvodem k užití odložené modifikace zpracovávaného souboru po skončení fáze anonymizace jsou problémy spojené s protokolem TCP, kterým se věnuje samostatná sekce 4.7. Uvést lze například anonymizaci segmentované TCP komunikace, jelikož nástroj TShark zpracuje aplikační protokol až ve chvíli, kdy má k dispozici všechny segmenty. Může se tak stát, že modifikovaný atribut není fyzicky přítomen v aktuálně zpracovávaném paketu, ale v paketu, jenž byl již zpracován. Seznam modifikací v tomto případě umožní zpětně vložit modifikaci daného atributu do správného paketu. Případné odstranění modifikací, které je nutné při řešení ztráty paketů TCP komunikace, je tímto přístupem také umožněno.

4.5 Zápis změn

Poslední fází anonymizace souboru pcap je samotné zapsání změn do souboru. Tato fáze nastává po zpracování všech dostupných paketů, čímž je finalizovaný seznam modifikací. Získaný seznam modifikací je nutné před samotným zápisem zvalidovat a případně upravit. Zmíněná validace se týká paketů patřících do TCP komunikace, jelikož jednotlivé TCP pakety je nutné prověřit, zda byly adekvátně anonymizované. Jaké aspekty je nutné zkoumat a proč, řeší vlastní sekce 4.7, a proto se jí následující text nebude věnovat.

Pravidla anonymizační politiky určují pořadí, ve kterém jsou aplikována na jednotlivé pakety. Jejich pořadí ale nemusí odpovídat fyzickému umístění anonymizovaných atributů v daných paketech. Aby bylo při zápisu užito pouze jednoho směru posunu souborového ukazatele, jsou jednotlivé modifikace paketů před zápisem seřazeny. Každá modifikace obsahuje pozici anonymizovaného atributu v rámci paketu, podle níž jsou jednotlivé modifikace seřazeny. Zápis tak probíhá jedním směrem. Výjimku tvoří pravidla, která modifikují data na stejné pozici, např. atributy užívající bitovou masku. Modifikace jsou řazeny podle dvou úrovní. První úroveň řazení je podle pozice v bytovém poli paketu, druhá úroveň je podle pořadí anonymizačního pravidla atributu.

Zápis změn začíná vytvořením kopie anonymizovaného souboru, aby nedošlo k poškození originálního souboru v případě selhání anonymizace. Soubor je otevřen pro binární zápis a ukazatel nastaven za globální hlavičku souboru, tedy na první byte paketové hlavičky prvního paketu. Následně dochází k aplikaci jednotlivých modifikací paketů. Díky informaci o začátku paketu daného souboru je pro každou modifikaci vypočten příslušný offset atributu v rámci celého souboru. Offset je následně užitý k navigaci po souboru a zapsání příslušných dat.

4.6 Modifikace časových razítek paketu

Časové razítko značící dobu příchodu paketu může vyzrazovat důležité informace o komunikaci na sledované síti — např. vytížení sítě či dobu výskytu podezřelé komunikace. Anonymizační politika tak může definovat tento atribut jako jeden z atributů určených k anonymizaci. Navrhovaná aplikace musí k tomuto atributu přistupovat specifickým způsobem, jelikož není fyzickou součástí zpracovávaného paketu, ale je umístěn v paketové hlavičce jednotlivých paketů. Z tohoto důvodu nelze atribut zpracovat stejným způsobem jako atributy ostatní, protože informace poskytnuté nástrojem TShark v tomto případě neobsahují informace o pozici ani délce fyzických dat. Bytová data časových razítek také nejsou dostupná v bytovém poli celého paketu.

Užitím přepínače `-x` nástroje TShark lze extrahovat informace o atributech paketové hlavičky alespoň ve formě zpracovaných hodnot. Všechny atributy jsou v tomto případě reprezentované jako textové řetězce, které lze, díky znalosti formátu paketové hlavičky, převést na bytové pole tak, aby s ním mohla aplikace pracovat stejným způsobem jako s ostatními atributy. Při převodu jednotlivých atributů je nutné užít správné edianity souboru a časové přesnosti, aby nedošlo k vytvoření nevalidního pole. Obě informace jsou během anonymizace poskytovány i všem modifikátorům, čímž je umožněno vracet z anonymizačních funkcí správně naformátovaná data.

Časové razítko v podobě textového řetězce spojuje dva atributy paketové hlavičky, a tak je nutné je adekvátně zpracovat. Příkladem je razítko `1577912212.304622000`, jehož celá část představuje počet sekund, které je nutné zapsat samostatně na své místo v bytovém poli. Desetinná část představuje mikrosekundy nebo nanosekundy dle přesnosti souboru

pcap a je nutné je opět umístit na správné místo výsledného pole. Pro desetinnou část také platí omezení velikosti hodnoty viz sekce 2.2, a proto je nutné odstranit veškeré přebytečné `'0'` zprava, jelikož na celou desetinnou část je nahlíženo jako na celé číslo, které by bez odstranění nepotřebných nul mohlo mít větší hodnotu. Stejného výsledku lze dosáhnout i načítáním bytového pole hlavičky přímo z originálního souboru, čímž by nebylo nutné jednotlivé atributy zpracovávat. Nicméně tento přístup není užitý z důvodu nutné režie pro práci se souborem, který by tak byl užívaný jak samotnou aplikací, tak nástrojem TShark. Bytové pole pro paketovou hlavičku souboru je získáváno během fáze zpracování paketu, viz sekce 4.3.

Aplikace v tomto případě musí zajistit, aby při anonymizaci časových razítek bylo užito správného bytového pole. Pro rozhodování o výběru bytového pole lze užít jméno modifikovaného atributu, jelikož atributy paketové hlavičky obsahují předponu `frame`, např. `frame.epoch_time` označuje zmíněné časové razítko.

K aplikaci modifikací tykajících se paketové hlavičky je nutné přistupovat odlišně. Za výchozí pozici všech modifikací je uvažován začátek paketu, ovšem v tomto případě je to začátek paketové hlavičky. Pro výpočet správné pozice v souboru je tak nutné odečíst délku paketové hlavičky od začátku paketu.

Při anonymizaci časových razítek je nutné mít na paměti, že dochází pouze ke změně hodnoty času příchodu paketu. Pořadí paketu v seznamu uložených paketů souboru anonymizace nijak neovlivňuje.

4.7 Anonymizece TCP komunikace

Pakety spadající do TCP komunikace je nutné z důvodu specifického chování protokolu TCP zpracovávat mírně odlišně od představených postupů anonymizace. Uvedené změny se týkají pouze transportní a aplikační vrstvy architektury TCP/IP. Pro nižší vrstvy probíhá anonymizace beze změn.

Na aplikační data přenášená pomocí protokolu UDP lze nahlížet jako samostatné pakety, které lze zpracovávat nezávisle na předcházejících či následujících paketech patřících do stejného komunikačního toku. Avšak na data přenášená pomocí transportního protokolu TCP takto nahlížet nelze. Protokol TCP vytváří před vlastní komunikací komunikační kanál a jednotlivé pakety patřící do komunikace tohoto kanálu mohou ovlivňovat své v čase sousední pakety či být jimi ovlivněny. Příkladem je automatické rozdělení aplikačních dat, kdy dochází na úrovni transportní vrstvy k segmentaci. Data tak nejsou z pohledu této vrstvy nezávislá, jako je tomu u protokolu UDP[7]. Budeme-li uvažovat, že data určená k anonymizaci obsahují užití TCP protokolu, je nutné aplikačně řešit následující problémy:

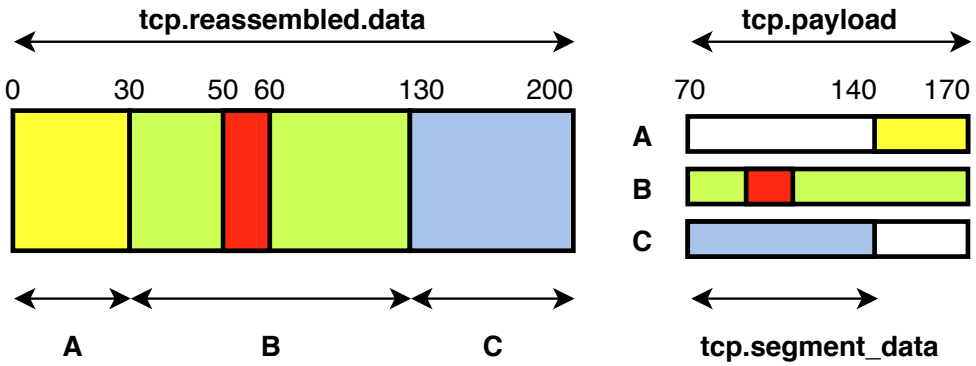
- segmentace aplikačních dat,
- znovu zaslané pakety,
- chybějící pakety,
- nezpracovatelné pakety.

Pro představení řešení jednotlivých problému je nejprve nutné uvést, jakým způsobem nástroj TShark přistupuje ke komunikaci obsahující protokol TCP. Zpracovává-li nástroj paket

[7] Z pohledu aplikační vrstvy mohou být data jednotlivých paketů na sobě závislá nehledě na užitý transportní protokol.

obsahující TCP protokol, který nese nějaká aplikační data (payload), pokusí se tento payload zpracovat a získat aplikační protokol. Obsahuje-li payload dostatek dat – celý aplikační protokol – provede jeho zpracování. Ve výstupním objektu JSON tak lze nalézt zpracovaný aplikační protokol. V opačném případě zůstává prozatím aplikační protokol nezpracovaný. TShark si jej ale zapamatuje. Pokud během zpracovávání dalších paketů narazí na následující segment předešlého TCP toku, pokusí se opět zpracovat aplikační protokol. Ovšem v tomto případě užije nezpracovaná data předešlého segmentu a data aktuální. Pokud se mu z dostupných dat podaří získat celistvý aplikační protokol, přidá jej do výstupního JSON objektu **aktuálně** zpracovávaného paketu.

Nástroj TShark dokáže zpracovat více záznamů protokolů aplikační vrstvy naráz, má-li dostupná data. Příkladem může být paket obsahující zbytek dat aplikačního protokolu předchozího paketu a celý zpracovatelný protokol, jev je častý u protokolu `TLS`. Data anonymizovaného atributu se tak mohou nacházet v jiném než aktuálně zpracovávaném paketu. Segmentaci aplikačního protokolu demonstruje obrázek č. 4.10.



Obrázek 4.10: Příklad segmentace aplikační vrstvy a její složení v celistvé pole. **A, B** a **C** označují jednotlivé pakety obsahující segmenty. Červená část představuje anonymizovaný atribut.

4.7.1 Segmentace aplikačních dat

Nutnou informací o paketu TCP komunikace je, zda bylo k zpracování aplikačního protokolu užito segmentů. Jestliže použil nástroj TShark segmenty jiných paketů, obsahuje tuto informaci poskytnutý JSON objekt v přidaném objektu `tcp.segments`, který lze nalézt mezi jednotlivými zpracovanými protokoly. V objektu je možné nalézt např. počet užitých segmentů, délku spojených dat, bytovou reprezentaci spojených dat a také identifikátory paketů, v nichž jsou segmenty obsaženy. Pro každý segment TShark poskytuje informace o jeho pozici a délce ve spojených datech segmentů. Informace o pozici segmentů v jejich paketech však dostupná není.

Pro každý segment paketu TCP komunikace je nutné zjistit a uchovat informace o jeho pozici a délce v paketu, aby bylo možné anonymizovat atributy nacházející se v jiných paketech. Zda paket TCP komunikace obsahuje ve svém payloadu segment, tedy data, která nebyla užita pro zpracování aplikačního protokolu, lze zjistit z objektu JSON. Segment definuje objekt `tcp.segment_data`, který nese informace o jeho pozici a délce v rámci bytového pole paketu. Informace o segmentu jsou získány během fáze zpracování paketu a následně překopírovány do struktury seznamu modifikací, kde jsou zpětně snadno dostupné.

Pro každý anonymizovaný atribut je nyní nutné zjistit, zda je součástí segmentovaných dat či nikoliv. Během fáze zpracování paketu 4.3 dochází k vyhledávání jednotlivých atributů postupných procházením objektu JSON. Při vyhledávání dochází k záznamu cesty objektem, která vedla k nalezení hledaného atributu, jelikož reálné zanoření atributu se může lišit od úrovně zanoření definované jeho jménem. Pomocí cesty objektem JSON lze přesně určit, který aplikační protokol (např. `TLS`) je vlastníkem tohoto atributu – stejných aplikačních protokolů může v jednom objektu více kvůli zpětnému zpracování dat.

Příkladem vyhledání vlastníka (protokolu) segmentovaného atributu je atribut protokolu TLS `tls.record.content_type` obsažený v objektu JSON se dvěma TLS protokoly. Jeho cesta je `tls.0.record.content_type`, kde `0` značí, že byl navštíven první `tls` protokol v řadě. Objekt definující protokol má stejnou strukturu jako jeho atributy, a tak lze získat informaci o jeho pozici v bytovém poli, která je klíčová pro detekci segmentovaného atributu.

Začíná-li celý protokol na pozici `0`, lze tvrdit, že byl celý protokol zpracovaný spojením více segmentů, jelikož všechny pozice atributů jsou vždy vztažené k užívanému bytovému poli. V případě nesegmentovaných protokolů je užité bytové pole shodné s bytovým polem zpracovávaného paketu. Segmentované protokoly používají bytové pole, jež vzniklo spojením jednotlivých segmentů. Výjimku tvoří pouze protokol Ethernet, který jako protokol nejnižší vrstvy zapouzdřuje všechna data, a jeho začátek je shodný se začátkem celého paketu.

Následující seznam představuje některé z objektů užívaných pro zpracování segmentované TCP komunikace:

- `tcp.segments` nese informaci o užitých segmentech pro zpracování segmentovaného aplikačního protokolu. Obsahuje identifikátory paketů jednotlivých segmentů, pole `tcp.segment`, a pro každý segment definuje, který úsek tvoří ve spojeném bytovém poli – příslušný objekt na indexu segmentu objektu `tcp.segment_raw`.

- `tcp.reassembled.data` představuje sloučené bytové pole segmentovaných dat. Využívá se k vlastní anonymizaci segmentovaných atributů, obdobně jako `frame_raw` pro nesegmentované atributy, ovšem v tomto případě představuje pouze jeden aplikační protokol.

- `tcp.segment_data` se v objektu vyskytuje, ppokud nebyla užita všechna data TCP payloadu ke zpracování aplikačního protokolu; lze tak určit, že paket má vazbu na předcházející čí následující pakety.

Dojde-li při fázi anonymizace k anonymizaci segmentovaného atributu, je třeba tento fakt zohlednit před i po vlastní anonymizací atributu. Zmíněná sekce uvádí, že aktuální hodnota modifikovaného atributu je získána načtením příslušného počtu bytů z bytového pole paketu. V případě segmentovaného atributu je toto pole nevalidní, jelikož nemusí obsahovat daný atribut. Informace, jež jsou o umístění atributu dostupné, jsou vztaženy k bytovému poli spojených segmentů. Je tedy nutné před získáním hodnoty ověřit o jaký typ atributu se jedná a na základě toho použít bytové pole – v tomto případě obsah `tcp.reassembled.data`. Jedná se tak o třetí typ bytového pole, se kterým aplikace může pracovat – pole vlastního paketu, pole hlavičky paketu a spojené pole segmentů. Zápis anonymizovaného hodnoty nazpět do bytového pole je shodný s nesegmentovanými atributy, ovšem liší se způsob vytvoření modifikace.

Pro vytvoření modifikace nové hodnoty je nejprve nutné zjistit, ve kterém paketu se anonymizovaný atribut nachází. Jelikož je známá jeho počáteční pozice ve spojeném bytovém

poli, lze paket určit na základě informací z `tcp.segments`, který definuje úseky spojeného pole a identifikátor paketu, jemuž úsek patří. Kromě paketu, ve kterém se atribut nachází, je vhodné zjistit i pakety následující, jelikož samotný atribut může ležet na pomezí dvou a více segmentů. Identifikátor paketu je užitý k nalezení struktury seznamu modifikací paketu, jelikož je do něho třeba přidat nově vytvářenou modifikaci. Struktura seznamu modifikací obsahuje dostatečné informace pro výpočet fyzického umístění atributu v paketu.

Pokud jsou data atributu rozprostřena přes více segmentů, je nutné data rozdělit na jednotlivé modifikace a ty přiřadit do patřičných seznamů modifikací užitých paketů. Segmentovaný atribut tak může být anonymizovaný vícero modifikacemi, čímž se výrazně liší od nesegmentovaných atributů, kde jedna modifikace zastupuje právě jeden atribut.

4.7.2 Znovu zaslané pakety

Jednotlivé pakety TCP komunikace lze přiřadit do příslušných TCP toků. Jelikož jednotlivé toky zaručují spolehlivý přenos dat, může se stát, že je jeden TCP paket doručen vícekrát. Stát se tak mohlo z vícero důvodů, které vyplývají z principu daného transportního protokolu. Navrhovaná aplikace musí s tímto scénářem počítat a zajistit anonymizaci kopií (znovu zaslaných) paketů stejným způsobem, jako jejich originálů (první doručené). Pokud by byl anonymizovaný pouze vzorový paket, bude ve výsledné anonymizované komunikaci jak hodnota anonymizovaná, tak neanonymizovaná. Případný útočník na anonymizovaný soubor tak dostává do rukou obě verze dat, které lze užít pro útok typu **known-plaintext attack**.

Nástroj TShark ve zpracovaném JSON objektu poskytuje informace o příslušnosti paketu do TCP toku. Každý tok je definovaný číselným identifikátorem, který lze nalézt pod klíčem `tcp.stream`. Aby bylo možné vyhledávat originály jednotlivých kopií, je nutné si pro jednotlivé TCP toky poznačit pakety, které do nich patří.

Pro detekci kopie paketu lze opět užít informace poskytnuté nástrojem TShark. Nástroj do výstupního objektu JSON přidává objekt `tcp.analysis`, který není součástí protokolu TCP, ale nese dodatečné informace přidané nástrojem. Objekt není přítomný vždy, a tak je nutné validovat jeho přítomnost. V uvedeném objektu lze nalézt informace ohledně anomálií zpracovávaného protokolu TCP či zvláštností vůči příslušnému TCP toku. Obsahuje-li zmíněný objekt klíč `tcp.analysis.retransmission` (objekt je značně zanořený v dalších objektech), je nástrojem označený za kopii.

Je-li TCP paket označený za kopii, je třeba nalézt jeho originál. Budeme-li vycházet z předpokladu, že kopie je identický (na úrovni transportní vrstvy) paket, který byl zaslaný vícekrát, lze užít informace TCP protokolu pro nalezení jeho originálu v příslušném TCP toku, jelikož nástroj TShark informaci o originálním paketu neposkytuje. Jako identifikátory byly vybrány následující atributy – sekvenční číslo (`tcp.seq`) a následující sekvenční číslo (`tcp.nxtseq`), které není součástí protokolu TCP, ale uměle přidané nástrojem.

Pokud byl nalezený originální paket, lze přistoupit k anonymizaci. Jelikož jsou pakety shodné, je možné překopírovat seznam modifikací originálního paketu do seznamu modifikací kopie. Je však nutné překopírovat pouze modifikace transportní a aplikační vrstvy (lze odlišit užitím informací o začátku protokolu TCP a modifikovaných atributů v bytovém poli). Nižší vrstvy se mohou lišit a případná změna časového razítka by byla pro kopii nevalidní. Seznam modifikací také není finální – následující paket TCP toku může ovlivnit aktuální seznam. Z tohoto důvodu je v seznamu modifikací kopie paketu pouze poznačen identifikátor originálu a samotný seznam modifikací je překopírován až po fázi validace TCP

toků, viz 4.7.3. Nicméně kopie paketu je podrobena anonymizaci mimo atributy transportní a aplikační vrstvy.

4.7.3 Chybějící a neznámé TCP pakety

I přesto, že TCP protokol zajišťuje spolehlivý přenos dat, se může stát, že v uloženém síťovém záznamu některé pakety daného toku chybí. Situace může nastat fyzickým nedoručením paketu, případně je způsobena samotným záznamem komunikace, který mohl začít nebo skončit během existence TCP toku – chybí několik segmentů začátku nebo konce komunikace.

Je-li v TCP toku výpadek alespoň jednoho paketu, může se stát, že výslednou komunikaci nelze plně zpracovat, jelikož ztracený paket mohl být segmentovaný, což přináší problémy se zpracováním aplikačního protokolu. Navrženy jsou dva způsoby, jak se s výpadky paketů vypořádat, přičemž oba v různé míře odstraňují informace daných protokolů.

První způsobem řešení výpadku paketu je odstranění aplikačních dat celého TCP toku. Tento přístup má velký dopad na použitelnost anonymizovaných dat, jsou-li odstraněná data klíčová pro budoucí zpracování souboru. K odstranění aplikačních dat celého TCP toku dochází, pokud je tok prohlášený za nevalidní – během jeho existence došlo k výpadku alespoň jednoho paketu. TShark v případě chybějícího paketu poskytne očekávanému následovníkovi ztraceného paketu informaci o ztrátě. Informace o ztrátě je přítomna v objektu `tcp.analysis` pod klíčem `tcp.analysis.lost_segment`. Po skočení fáze anonymizace dochází k ověření validity všech TCP toků a data paketů nevalidních TCP toků jsou vymazána. Ze struktury seznamu modifikací jsou odstraněny všechny modifikace, které jsou anonymizují aplikační protokol (lze zjistit podle začátku atributu). Následně je do seznamu vložena modifikace, která maže všechna aplikační data protokolu TCP.

Druhým přístupem ošetření výpadku paketů je tzv. **chytré odstranění**. Aplikace se v tomto případě snaží odstranit pouze ty části segmentů, které nebyly užity a na niž měl výpadek paketu dopad. Obrázek č. 4.11 demonstruje čtyři možné situace zpracovaného TCP payloadu. `Segment` představuje nezpracovaná data, `aplikace` data zpracovaná. Při zpracování paketu, viz 4.3, dochází k vytvoření záznamu o segmentech, které nebyly zatím užity, aby je bylo možné případně smazat, pokud nebudou dodatečně referencovaná následujícími pakety. Identickým způsobem jsou anonymizované osamoceného segmenty komunikace, které vznikly z důvodu nepřítomnosti začátku či konce dané komunikace. Způsoby detekce jednotlivých segmentů demonstruje následující seznam.

Nástroj TShark představuje v objektu `frame.protocols` seznam protokolů, které paket obsahuje, ale nemusí být nutně zpracované. Protokol zapouzdřený přímo v TCP označíme za **AP (aplikační protokol)**. Pokud je AP zpracovaný, označíme jej za **ZAP (zpracovaný aplikační protokol)**. Následující seznam představuje možné situace výskytu segmentů aplikačních dat v jednom TCP payloadu:

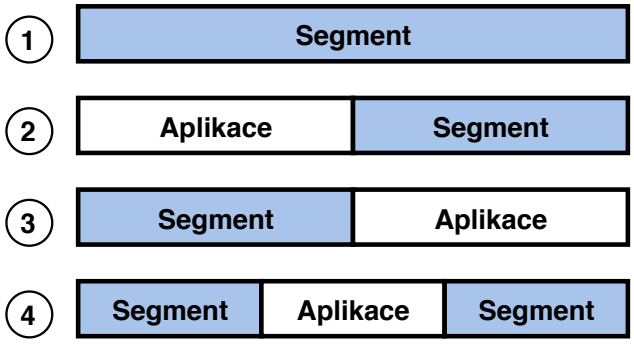
1. Celý `tcp.payload` představuje nezpracovaný segment. Jeho detekci lze provést ověřením, že TCP paket má `tcp.segment_data` a že `AP` není zpracovaný.

2. Segment obsažený v koncové části `tcp.payload` detekujeme na základě rozdílného začátku objektů `tcp.payload` a `tcp.segment_data`. Víme, že segment zabírá zbytek payloadu, jelikož zbytek aplikačních dat se nepodařilo zpracovat.

3. Nejproblematičtějším scénářem je segment obsažený na začátku payloadu následovaný zpracovaným aplikačním protokolem. Začíná-li payload segmentem, nástroj TShark

ve většině případů prohlásí celý `tcp.payload` za `tcp.segment_data`, ve kterém se tak ocitá i celý `ZAP`. Je tak nutné najít první `ZAP` (může jich být zpracováno více). Obsahuje-li paket segmentovaný `ZAP`, víme, že data segmentu byla užita pro jeho zpracování, a celá část segmentu je tak zpracována a není ji třeba uchovávat pro případné smazání. Nalezneme-li nesegmentovaný `ZAP`, víme, že nutně následuje po skončení segmentu. Díky informaci o jeho začátku, lze určit, kde končí nezpracovaný segment v `tcp.segment_data`. Segmentu tak lze opravit počáteční pozici a uchovat jej pro případné smazání.

4. Případ dvou segmentů v jednom `tcp.payload`, lze detekovat díky dvoum výskytum `tcp.segment_data`. Druhý segment lze automaticky uvažovat za koncový a není jej dále třeba zpracovávat. Ovšem první segment představuje stejné problémy jako bod. 3, a je tak nutné jej zpracovat stejným způsobem.



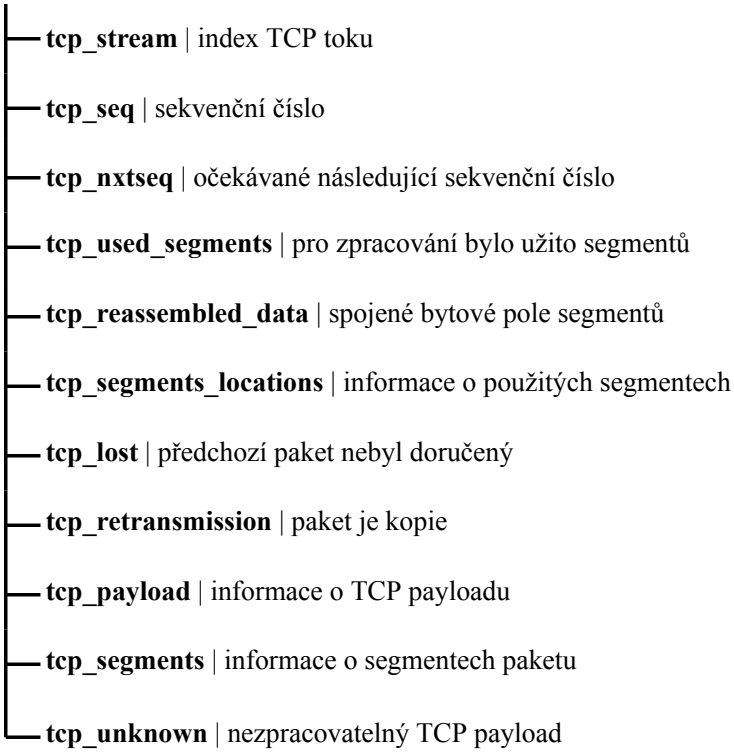
Obrázek 4.11: Přehled možných výskytů segmentů aplikačních dat v rámci TCP payloadu

Zpracovává-li aplikace segmentovaný protokol, poznačí pro všechny užité segmenty (struktura seznamu modifikací příslušného paketu), že byly použité. Po skončení fáze anonymizace jsou všechny segmenty testovány na případ užití. Pokud nebyly užity jsou vytvořeny modifikace, které jsou zodpovědné za smazání dat, na základě informací o těchto segmentech. Uvedené způsoby detekce nezpracovaných segmentů jsou silně závislé na informacích, které poskytuje nástroj TShark, jenž v některých situacích neposkytuje validní data. Například duplikované pakety přináší velký šum při zpětném referencování užitých segmentů, či jsou segmenty referencované vícekrát, pokaždé s jinou užitou délkou. Je tak nutné ověřit, jak se uvedený postup projeví po anonymizaci souboru.

Posledním problémem jsou neznámé pakety TCP komunikace. Do této kategorie jsou zařazené pakety, jež nemají `tcp.segment_data` čili jsou nezávislé na svých sousedech, ale mají `tcp.payload`, což znamená, že nesou nějaká aplikační data. Není-li v tomto případě zpracovaný aplikační protokol, je celý payload vymazán.

4.7.4 Nutné informace TCP komunikace

Předchozí sekce definovaly problémy způsobené transportním protokolem TCP. Pro řešení představených problému je nutné ze zpracovaných paketů extrahovat dodatečné informace, které jsou specifické pouze pro zpracování tohoto protokolu. Každá sekce postupně přidávala informace, které je nutné extrahovat a jejich souhrn lze nalézt na obrázku č. 4.12. Struktura rozšiřuje informace, které jsou získány ve fázi zpracování paketu, 4.3.



Obrázek 4.12: Přehled atributů, které je nutné extrahovat z paketu pro zpracování TCP komunikace.

4.8 Globální nastavení anonymizace

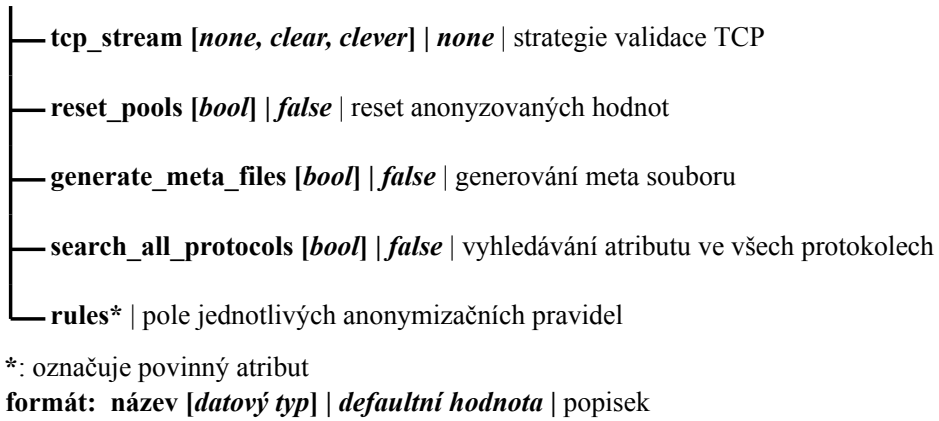
Definice průběhu anonymizace se skládá ze dvou částí. První část tvoří samotná anonymizační politika, kterou tvoří jednotlivá anonymizační pravidla, jež definují jaké atributy se budou anonymizovat a jakým anonymizačním algoritmem. Pořadí jednotlivých pravidel je důležité, jelikož jsou vyhodnocována sekvenčně a mají mutující efekt. Druhou část definice průběhu anonymizace tvoří globální nastavení, která ovlivňují její průběh. Obě části jsou definované pomocí konfiguračního souboru, jehož formát lze vidět na obrázku 4.13.

Položka `tcp_stream` slouží k nastavení strategie validace toků TCP komunikace, o kterých hovoří sekce 4.7. Její možné hodnoty jsou:

- `none` – žádná validace TCP toků neprobíhá,
- `clear` – probíhá validace porušených TCP toků, validace osamělých segmentů či neznámé komunikace zůstává nezměněna,
- `clever` – probíhá validace veškeré TCP komunikace, včetně osamělých segmntů a neznámé komunikace.

Vyhledávání atributu ve všech protokolech obsažených v paketu je možné pomocí položky `search_all_protocols`, která ovlivňuje průběh vyhledávání atributů. Je-li nastavena, jsou prohledávány všechny dostupné protokoly.

Roli, při anonymizaci více souborů síťové komunikace naráz, hraje položka `reset_pools` rozhodující o poolech jednotlivých pravidel. Je-li její hodnota booleovská pravda, dochází



Obrázek 4.13: Struktura konfiguračního souboru

mezi jednotlivými soubory k vymazání hodnot všech poolů. V opačném případě jsou hodnoy sdíleny mezi anonymizacemi všech souborů.

Definovat lze také užitím položky `generate_meta_files` zda mají být hodnoty všech poolu zaznamenány do souboru včetně jejich původních hodnot, což může být vhodné při zpětném dohledání originálních hodnot.

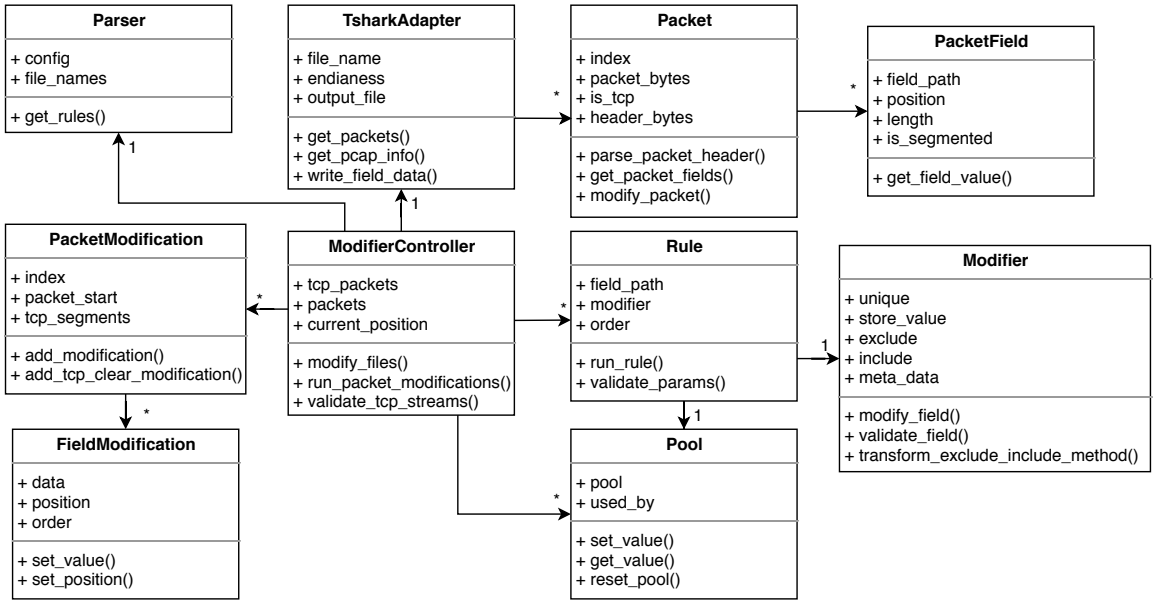
Kapitola 5

Implementace

Předchozí kapitola představila návrh aplikace pro anonymizaci síťové komunikace. Představeny byly jednotlivé fáze, kterými anonymizovaná data musí projít, včetně definice anonymizační politiky, jež celý anonymizačních průběh definuje. Tato kapitola se zabývá vlastní implementací navržené aplikace včetně řešení problémů propojení nástroje TShark s aplikací tak, aby splňovala definované podmínky. Představeny také jsou některé klíčové algoritmy, které aplikace užívá např. vyhledávání anonymizovaného atributu v objektu JSON nástroje TShark.

Pro správné fungování aplikace je nutné poskytnout dva vstupního argumenty. Prvním argumentem je cesta ke konfiguračnímu souboru. Druhým argumentem aplikace jsou cesty jednotlivých souborů, které jsou určeny k anonymizaci. Formátem konfiguračního souboru i oběma vstupními argumenty se zabývá sekce 5.1.

Aplikace je implementovaná jako konzolová aplikace v programovacím jazyce Python 3.8. Užito je objektového paradigmatu a jednotlivé třídy aplikace lze nalézt na obrázku č. 5.1.



Obrázek 5.1: Třídní diagram implementovaných tříd aplikace. Jednotlivé třídy zobrazují pouze některé atributy a metody.

Aplikace je plně závislá na použitém nástroji **TShark**, jenž dodává implementované aplikaci detailní informace o zpracovávaném síťovém záznamu. Pro správnou funkci aplikace je také nutné zajistit dostupnost knihovny **YAJL**, která zpracovává výstupní data nástroje TShark a předkládá je výsledné aplikaci. Způsobem propojení zmíněných nástrojů a knihoven se zabývá sekce 5.2.

Jelikož anonymizační metody (modifikátory) tvoří jednu z nejdůležitějších částí aplikace, je jejich rozhraní věnována sekce 5.3. V sekci lze nalézt popis samotného rozhraní včetně vysvětlení nutnosti jednotlivých částí a jejich funkce. Sekce také popisuje způsob implementace a integrace vlastních modifikátorů do výsledné aplikace. Zbývajícími třídami aplikace se zabývá sekce 5.4, která popisuje jejich účel a zodpovědnosti.

5.1 Rozhraní aplikace

Aplikace vyžaduje dva povinné vstupní argumenty. Jedním z argumentů je `--files`, který slouží k definici cest souborů určených k anonymizaci. Definovat je nutné alespoň jeden soubor, přičemž jediným podporovaným formátem je formát pcap (libpcap).

Druhý argument `--config` je určený k definici cesty ke konfiguračnímu souboru, který obsahuje anonymizační politiku. Soubor se očekává ve formátu **YAML**[1]. Formát YAML byl vybrán pro možnost přehledného zápisu anonymizačnich pravidel, které jsou klíčové pro definici průběhu anonymizace. Formát, který konfigurační soubor musí splňovat, je zobrazen na obrázku č. 4.13.

Příklad anonymizační politiky je možné vidět na obrázku č. 5.2. Lze vidět nastavení validace TCP komunikace na možnost `clever` čili automatickou anonymizaci nevyužitých aplikačních segmentů. Mezi jednotlivými anonymizovanými soubory je vymazáno mapování neanonymizovaných a anonymizovaných hodnot – atribut `reset_pools`. Mapování jednotlivých hodnot není ukládáno do meta souborů a vyhledávání jednotlivých atributů je omezeno pouze na protokoly, které definuje první část jména atributu.

Obrázek č. 5.2 také demonstruje způsob, kterým lze definovat položky `include`, resp. `exclude`. První pravidlo (`http.host`) demonstruje definici položky `include`, kdy je užito kromě samotných hodnot i položky `validation`, která označuje způsob validace, jelikož je v tomto případě užitý validátor textového řetězce. Druhé pravidlo (`udp.srcport`) představuje způsob zápisu intervalů pro číselné hodnoty atributů. Samotná čísla lze uvést beze změn, pro intervaly je užitý následující formát `[0,1024]`. Nicméně způsob zpracování položek `include` a `exclude` je přímo závislý na položce `modifier`, a tak je možné zvolit zcela odlišný způsob zápisu, bude-li jej modifikátor podporovat.

5.2 Propojení externích nástrojů a knihoven

Pro získání informací o jednotlivých paketech je nutné propojit implementovanou aplikaci s nástrojem TShark. Jelikož je TShark konzolovým nástrojem, nelze jej přímo užít v implementované aplikaci, ale je jej nutné paralelně spustit a zpracovávat jeho výstup, o čemž hovoří sekce 4.2.

Nástroj TShark je nutné spustit s určitými přepínači, aby bylo zajištěno získání všech nutných informací pro chod implementované aplikace. Spuštění nástroje včetně užitých přepínačů je možné vidět na obrázku 5.3. Přepínač `-r` slouží pro zadání cesty k zpracovávanému souboru, typ výstupního formátu definuje přepínač `-T` s hodnotou `json`. Pro přepínač `-T`

[1] https://yaml.org/

```
tcp_stream: clever
reset_pools: true
generate_meta_files: false
search_all_protocols: false
rules:
  -
      field: http.host
      modifier: HttpTextRandom
      exclude:
        value: ['Host: www.foo.com']
        validation: 'prefix'
      include: []
      additional:
         foo: 'bar'
  -
      field: udp.srcport
      modifier: NumberMarker
      value_group: 'src_port'
      value: 123
      include: [[0,1024], 8080]
      exclude: []
```

Obrázek 5.2: Příklad anonymizační politiky ve formátu YAML

lze zvolit i hodnotu `jsonraw`, která poskytuje i binární data zpracovaných paketů, ovšem neposkytuje zpracovanou paketovou hlavičku ze souboru pcap. Způsobem, jak docílit zisku binárních dat včetně zpracované paketové hlavičky, je užít hodnoty `json` společně s přepínačem `-x`. Posledním přepínačem je hodnota `-no-duplicate-keys`, která slouží k zachování všech atributů se stejným jménem, např. `http.response.line`.

```
tshark -r {nazev_souboru} -T json -x --no-duplicate-keys
```

Obrázek 5.3: Příkaz pro správné spuštění nástroje TShark

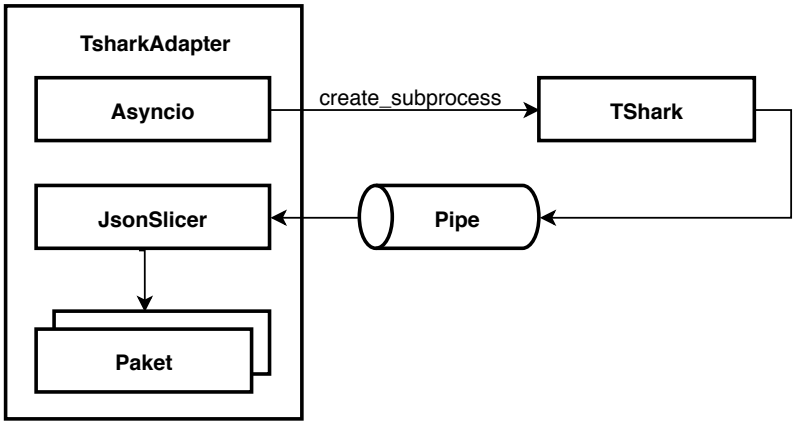
O integraci TSharku do aplikace se stará třída `TsharkAdapter` nacházející se v souboru `tshark_adapter.py`. Schéma integrace je možné videt na obrázku č. 5.4. Pro paralelní spuštění TSharku je využita python knihovna `asyncio`. Užití knihovny umožní přesměrovat standardní výstup spuštěného procesu (TShark) do předem definovaného souboru.

Jako soubor pro výstup (`stdout`) spuštěného procesu TSharku je zvolena roura systémová (`os.pipe`), přesněji soubor vytvořené roury určený pro binární zápis. Soubor pro zápis je nutné manuálně otevřít ze získaného souborového deskriptoru. Při spuštění procesu TSharku je možné použít přímo souborový deskriptor roury, avšak v tomto případě nedojde po skončení nástroje TShark k zaslání `EOF`[2] (End of file). Nezaslání `EOF` způsobí nezavření systémové roury a následující zpracování získaných dat by nikdy neskončilo.

Získaný proud znaků je následně zpracovaný pomocí python knihovny `JsonSlicer`, která umožňuje zpracování souborů ve formátu JSON. Soubory lze navíc knihovně předávat pomocí proudu dat. Je tak možné jako vstupní soubor předat knihovně druhý sou-

[2]EOF - Znak konce souboru

bor vytvořené roury, a to soubor určený ke čtení. Soubor je nutné opět ručně vytvořit ze získaného souborového deskriptoru. Knihovna umožňuje v získaném proudu znaků JSON souboru vyhledat jednotlivé pakety. Pakety jsou definované podle vzoru, který knihovna v datech vyhledává. Vzor je definovaný následovně `(None, None, 'layers')` a definuje, že hledaný objekt je zanořený na třetí úrovni s klíčem `layers`. Objekt pod klíčem `layers` je předaný aplikace pro další zpracování.



Obrázek 5.4: Schéma integrace nástroje TShark a knihovny JsonSlicer do výsledné aplikace.

5.3 Rozhraní modifikátorů

Jednotlivé modifikátory (anonymizační metody) tvoří klíčovou část anonymizační aplikace. Aby bylo možné modifikátory snadno upravovat, rozšiřovat či implementovat nové, implementují předem dané rozhraní. Rozhraní je možné nalézt v adresáři `interfaces` v souboru `modifier.py` a nese jméno `Modifier`. Samotné rozhraní je možné vidět na obrázku č. 5.1.

Jelikož rozhraní obsahuje několik atributů, které jsou pro správnou funkcionalitu modifikátoru důležité, jsou následně popsány:

- Atribut `unique` je typu `bool` lze použít pro sdělení aplikaci, že anonymizované hodnoty musí být unikátní. Lze tak očekávat, že modifikátor provádí permutaci nad vstupními (neanonymizovanými) daty. Aplikace následně kontroluje, zda anonymizovaná hodnota nebyla již přidělena jiné – neanonymizované – hodnotě.
- Atribut `store_value` je typu `bool` a slouží k nastavení, zda má být tvořeno mapování mezi neanonymizovanou a anonymizovanou hodnotou. V některých případech je tato informace zbytečná, např. při užití metody **clear**.
- Atributy `exclude` a `include` představují transformované hodnoty stejnojmenných položek anonymizačních pravidel. Jejich datovým typem je pojmenovaná struktura `tuple` nesoucí transformovanou hodnotu `value` a definovanou validaci `validation`, kterou některé validační funkce užívají – např. validace textového řetězce na shodu předpony, viz obrázek 5.2.
- Atribut `meta_data` je užitý pro exportování informací tykajících se samotné anonymizace. Obsah tohoto atributu je exportován v meta souboru u příslušného `poolu` hodnot. Lze jej tak užít např. pro exportování klíče užitého během pro hashovací funkci apod.

Rozhraní dále disponuje třemi abstraktními metodami, které jsou určené k implementaci tvůrcem modifikátoru. Ve všech metodách lze nalézt parametr `additional_parameters`, který obsahuje všechny informace z objektu `additional`, jenž je možné specifikovat v rámci anonymizačního pravidla. Parametr také obsahuje informace o anonymizovaném souboru a v některých případech číslo paketu. Funkci jednotlivých metod popisuje následující seznam:

- Metoda `modify_field` je zodpovědná za samotnou anonymizaci, resp. za získání anonymizované hodnoty. Jejími parametry jsou `original_value`, jenž představuje neanonymizovanou hodnotu, `value`, který odpovídá hodnotě `value` specifikované v anonymizačním pravidle a již zmíněný `additiona_parameters`. Očekávanou návratovou hodnotou je `bytearray` představující anonymizovanou hodnotu. Případně lze vrátit `None`, čímž lze aplikaci říci, že hodnota nebyla anonymizovaná a nemá být vytvořena modifikace.
- Metoda `validate_field` je zodpovědná za validaci hodnoty atributu před jeho anonymizací. Metoda má dva parametry `value`, jenž představuje neanonymizovanou hodnotu atributu a `additional_parameters`. Očekávanou návratovou hodnotou validační funkce je hodnota datového typu `bool`. V případě logické pravdy je atribut připuštěn k anonymizaci, v případě nepravdy je z anonymizace vyloučen.
- Metoda `transform_exclude_include_method` je užita aplikací pro transformaci položek `exclude` a `include` anonymizační pravidel do interní reprezentace vhodné pro validaci, viz. sekce 4.3.2. Očekávanou návratovou hodnotou je funkce, která je aplikovatelná na každou hodnotu zmíněných položek – jako příklad lze uvést např. transformaci definice intervalu číselných hodnot, viz obrázek č. 5.2 hodnota `include` druhého pravidla. Transformované hodnoty jsou následně dostupné v třídních atributech `exclude` a `include` zmíněných v předchozím seznamu.

5.3.1 Vytvoření nového modifikátoru

Nové modifikátory je možné vytvořit dvěma způsoby. Prvním způsob spočívá v rozšíření již existujícího modifikátoru, čímž dojde k zachování již implementovaných a definovaných funkcionalit, které je případně možné upravit dle vlastních potřeb. Nový modifikátor je také možné vytvořit z dostupného rozhraní jeho implementací. Aby aplikace dokázala s novým modifikátorem pracovat, je nutné splnit následující podmínky:

- Modifikátor implementuje rozhraní `Modifier` nebo rozšiřuje již existující modifikátor.
- Modifikátor je umístěn v adresáři `modifiers`.
- Jméno modifikátoru a název souboru, ve kterém je modifikátor obsažen jsou v následujícím vztahu. Jméno modifikátoru je v tzv. **PascalCase**[3], přičemž název souboru je v **snake_case**[4]. Důvodem je automatické odvozování názvu souboru modifikátoru z jeho jména, jelikož jméno modifikátoru se uvádí v položce `modifier` anonymizačního pravidla.

[3] https://techterms.com/definition/pascalcase

[4] https://medium.com/better-programming/string-case-styles-camel-pascal-snake-and-kebab-case-981407998841

Pro tvorbu nový modifikátoru jsou připraveny validační funkce, které lze nalézt ve třídě `Validator`, jež se nachází v adresáři `helpers`. Statické metody třídy je možné použít při implementaci metody `validate_field`. Pro implementaci jednotlivých anonymizačních metod `modify_field` je možné využít dostupných funkcí ze souboru `helpers.py` adresáře `helpers`. Jednotlivé funkce umožňují převod některých hodnot atributů z bytové reprezentace do datových typů a naopak. Dostupné jsou také např. funkce pro generování náhodných čísel zachovávající prefix či suffix, funkce pro modifikaci časových razítek apod.

5.4 Třídy anonymizační aplikace

Pro implementaci aplikace je užito objektového paradigmatu, a tak lze definovat jednotlivé třídy, které se podílejí na průběhu anonymizace. Významná třída, resp. rozhraní `Modifier` byla již detailně popsaná v sekci 5.3, a nebude jí tak dále věnována pozornost.

Parser

Třídu `Parser` je možné nalézt v adresáři `classes` v souboru s názvem `parser.py`. Třída je zodpovědná za zpracování vstupních argumentů – uložení cest souborů určených k anonymizaci a zpracování souboru s anonymizační politikou. Zpracování souboru s anonymizační politikou je implementováno pomocí python knihovny **yaml**, jež jej převede do python objektu. Nepovinné atributy anonymizační politiky jsou zde inicializované na své výchozí hodnoty.

TsharkAdapter

Třídu `TsharkAdapter` nacházející se v souboru `tshark_adapter.py` je možné nalézt v adresáři `classes`. Třída je zodpovědná za veškerou vstupní a výstupní práci s anonymizovanými soubory. Sekce 5.2 již tuto třídu zmínila a představila způsob integrace nástroje TShark do implementované aplikace.

Kromě zisku jednotlivých paketů je třída zodpovědná za zpracování globální hlavičky souboru pcap. Extrahované informace jsou endianita a přesnost časových razítek. Jelikož je tyto informace nutné získat ze samotného pcap souboru, musí jej třída pro tyto účely dočasně otevřít a načíst potřebný blok bytů.

Vytvoření kopie anonymizovaného souboru pro aplikaci jednotlivých modifikací včetně přesunu souborového ukazatele a zápisu nových dat je také v režii třídy `TsharkAdapter`.

Třída Rule

Třída `Rule` se nachází v adresáři `classes` v souboru `rule.py`. `Rule` představuje interní reprezentaci anonymizačního pravidla. Její součástí je atribut `modifier`, který obsahuje referenci na třídu `Modifier` daného pravidla. Důležitou metodou třídy `Rule` je `run_rule`, která je zodpovědná za validaci anonymizované hodnoty a samotnou anonymizovaci pomocí volání metody třídy `Modifier`.

Packet

Třída `Packet` představuje interní reprezentaci anonymizovaného paketu a je možné ji nalézt v souboru `packet.py` v adresáři `classes`. Třída zpracovává objekt JSON, který je získaný

z nástroje TShark. `Packet` obsahuje všechny atributy zmíněné v sekcích 4.3 a 4.7.4. Během zpracování paketu dochází k vytváření jednotlivých instancí třídy `PacketField`, která zastupuje jednotlivé atributy určené k anonymizaci.

Třída také obsahuje velké množství metod pro zpracování TCP komunikace, zahrnující např. určení pozic segmentů aplikačních dat pro případné smazání či extrahování spojeného segmentovaného bytového pole. Důležitou metodou je `modify_packet`, která je užita pro modifikaci bytového pole paketu po každé úspěšně provedené anonymizaci atributu.

PacketField

`PacketField` je třída představující interní reprezentaci jednotlivých atributů určených k anonymizaci. Kromě anonymizovaných atributů je užita i pro uchování informací např. o jednotlivých TCP segmentech. Nachází se v adresáři `classes` v souboru `packet_field.py`.

Mezi atributy třídy lze nalézt informace o pozici atributu v bytovém poli paketu, jeho délku či bitovou masku. Dynamicky jsou doplněné další atributy jako reálná cesta atributu v objektu JSON či informace, zda je atribut součástí segmentované komunikace.

PacketModification

Třída `PacketModification` představuje strukturu seznamu modifikací, o které se zmiňuje sekce 4.4. Třída se nachází v souboru `packet_modification.py` v adresáři `classes`. Ke každému zpracovanému paketu existuje jedna instance této třídy. Třída obsahuje informace o zpracovaném paketu, jako jsou identifikátor paketu, začátek paketu v souboru pcap, seznam nevyužitých TCP segmentů daného paketu a mnoho dalších informací.

Hlavním atributem třídy je pole jednotlivých instancí třídy `FieldModification`. Pole je v různých fázích anonymizace upravováno. Pomocí metod třídy `PacketModification` dochází k přidávání modifikací, jejich řazení či odstraňování během validace TCP toků.

FieldModification

Třída `FieldModification` se nachází v souboru `field_modification.py` v adresáři `classes` představují jednotlivé modifikace atributů. Obsahují novou hodnotu anonymizovaného atributu, její délku, pozici v rámci bytového pole či index pořadí anonymizačního pravidla pro pozdější seřazení. Instance třídy vznikají jako výsledek provedení anonymizačních metod a jsou vkládány do pole modifikací třídy `PacketModification`.

ModifierController

Hlavní třídou celé aplikace je třída `ModifierController`, která řídí celý průběh anonymizace. Třída se nachází v souboru `modifier_controller.py` v adresáři `classes`. Obsahuje referenci na třídu `TsharkAdapter`, díky které má přístup k anonymizovanému souboru a jednotlivým paketům souboru. Třída také uchovává mapování neanonymizovaných a anonymizovaných hodnot pomocí instancí třídy `Pool`. Dalšími atributy třídy jsou zpracovaná anonymizační pravidla `Rule` či seznam instancí třídy `PacketModification`.

Anonymizace je spuštěna pomocí metody `modify_files`, která v cyklu prochází všechny dostupné soubory k anonymizaci ze třídy `TsharkAdapter`. V rámci zpracovávaného souboru jsou postupně anonymizovány jednotlivé pakety, které jsou nejdříve převedeny do instance třídy `Packet`. Je-li paket součástí TCP komunikace, je poznačen ve struktuře patřičného toku pro případně pozdější vyhledání. Pokud paket obsahuje využité segmenty

jiných TCP paketů, jsou tyto segmenty poznačeny jako použité u patřičných instancí `PacketModification`. Následuje vytvoření instance třídy `PacketModification` a zavolání metody `run_packet_modification`, která spustí aplikaci jednotlivých pravidel na anonymizovaný paket.

Po anonymizaci všech paketů souboru následuje fáze validace TCP toků, která je spuštěna metodou `validate_tcp_streams`. Metoda projde všechny TCP toky a podle zvolené metody validace TCP toků provede případnou korekci. Je-li zvolena metoda **clever**, dochází k vytvoření instancí třídy `Modification` pro všechny nevyužité segmenty jednotlivých paketů za užití anonymizační metody **clear**.

Validaci toků následuje překopírování seznamu modifikací (instance tříd `Modification`) paketů, které představují originální pakety paketů, které byly opakovaně zaslány. Všechny seznamy modifikací jsou v tuto chvíli finální a lze je seřadit pro optimalizaci zápisu.

Pomocí instance třídy `TsharkAdapter` dochází k vytvoření kopie anonymizovaného souboru a jeho otevření pro zápis. Jednotlivé instance třídy `PacketModification` jsou sekvenčně procházeny a změny definované seznamem jejich modifikací (`Modification`) jsou aplikované na soubor.

Poslední částí metody `modify_files` je případné resetování mapování neanonymizovaných a anonymizovaných hodnot (třída `Pool`) a vytvoření meta souboru, jenž obsahuje dané mapování pro anonymizovaný soubor.

Pool

Třída `Pool` se nachází v adresáři `classes` v souboru `pool.py`. Její zodpovědností je udržovat mapování neanonymizovaných hodnot na anonymizované. Získané mapování je poté užito pro export do meta souboru.

Kapitola 6

Testování a zhodnocení

Tato kapitola se věnuje testování a zhodnocení navržené a implementované anonymizační aplikace. Z důvodu proměnných vstupních argumentů aplikace, mezi něž patří jednotlivé vstupní soubory či anonymizační politiky, nelze aplikaci testovat pro jednotlivé případy užití. Představeny jsou tak vybrané případy anonymizace a jejich dopady.

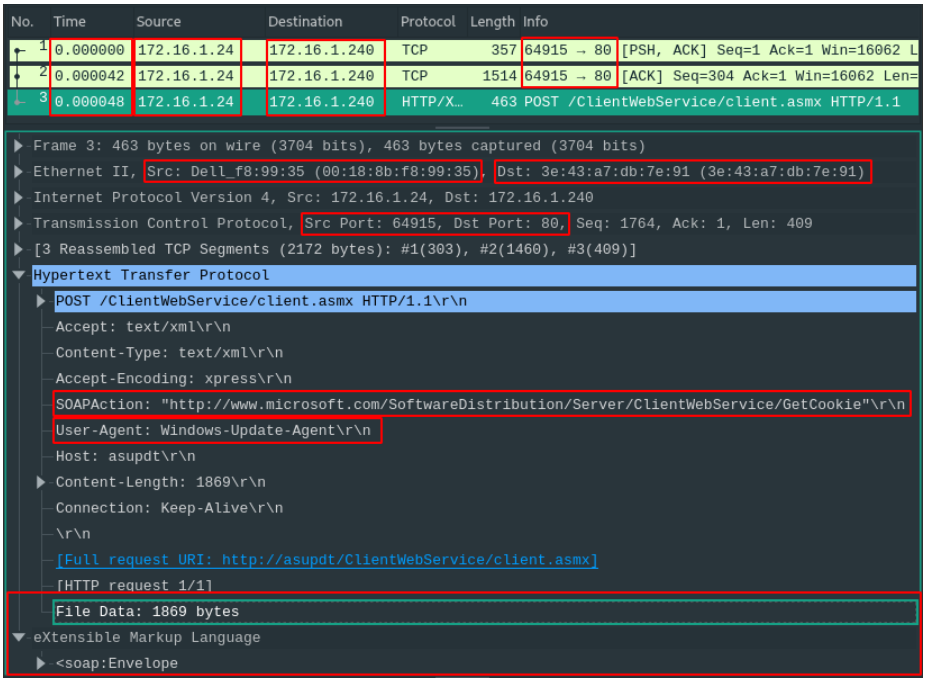
Výsledek anonymizace aplikované na všechny vrstvy modelu TCP/IP lze nalézt v sekci 6.1, která představuje anonymizaci na protokolu HTTP. Kromě samotných vrstev je v sekci ukázána i anonymizace časových razítek příchodů paketů, které nejsou součástí samotných paketů.

Demonstraci strategie `clever` zpracování porušené TCP komunikace demonstruje sekce 6.2 včetně dopadů, které strategie způsobí při následném zpracování anonymizovaného paketu. Sekce 6.3 se zabývá dvěma různými metodami anonymizace IP adres a jejich dopady na adresní prostor. Otestována byla také anonymizace portů transportních protokolů a zabývá se jí sekce 6.4. Časové a pamětové nároky aplikace řeší sekce 6.5.

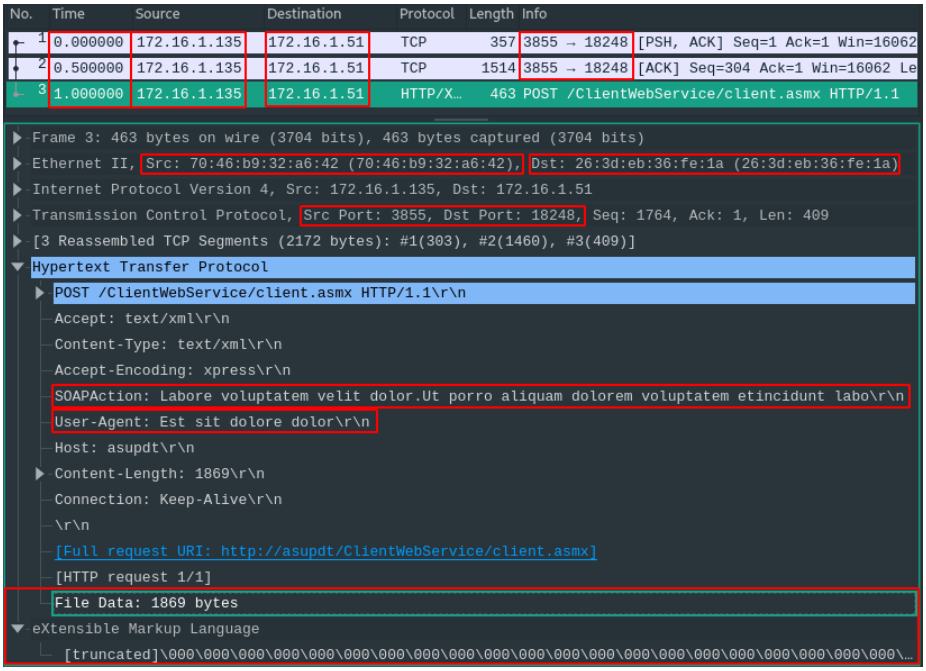
6.1 Demonstrace anonymizace

Demonstraci anonymizace různých atributů napříč všemi vrstvami TCP/IP modelu zobrazují obrázky č. 6.1 a 6.2. Anonymizován byl čas příchodu jednotlivých paketů, který byl pro první paket vynulován a příchod následujících paketů je vždy o půl sekundy opožděný oproti předchozímu paketu. Na obrázcích je tuto situaci možné vidět ve sloupci `Time`, jenž zobrazuje rozdíl v časech jednotlivých paketů. Fyzické adresy zařízení byly anonymizovány metodou `random`, které zachovala příznaky daných adres (multicast apod). IP adresy byly anonymizovány metodou `prefix preservation (24)`, čili prvních 24 bitů bylo zachováno a posledních 8 bitů náhodně doplněno. Modifikovány byly také zdrojové a cílové porty protokolu TCP, a to na náhodné hodnoty metodou `random`. Je možné si všimnout, že díky zachování mapování jedna ku jedné nedošlo k rozbití TCP toku.

V aplikačním protokolu HTTP byly anonymizovány atributy `SOAPAction` a `User-Agent` metodou `random`, která zachovala prefixy daných atributů (např. `SOAPAction:`) a suffixů nového řádku, aby nedošlo k porušení formátu protokolu HTTP. Je možné vidět, že protokol HTTP obsahoval data SOAP protokolu, která byla odstraněna metodou `clear`.



Obrázek 6.1: Zobrazení síťové komunikace nástrojem Wireshark před anonymizací. Jednotlivé červnené rámečky zobrazují atributy, které byly vybrány k anonymizaci.

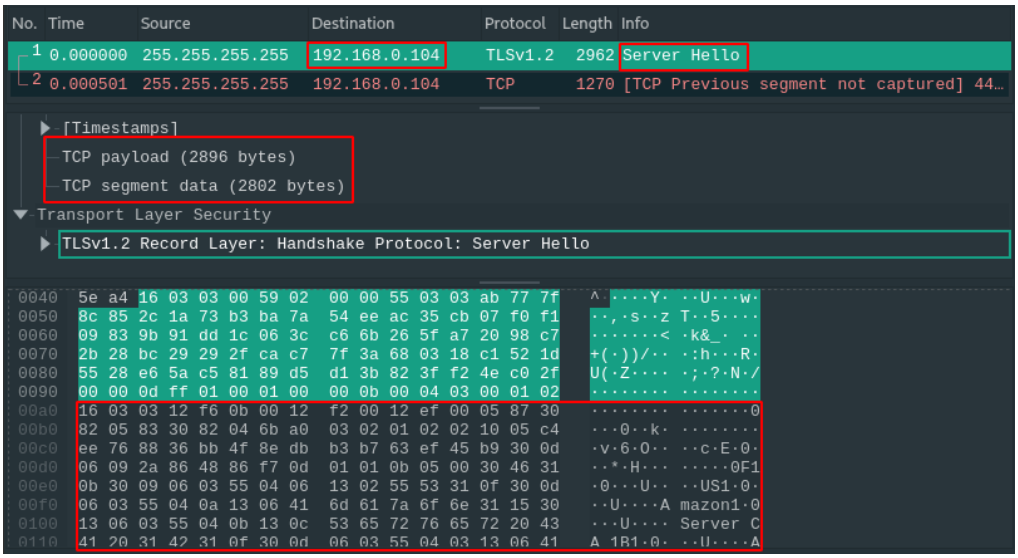


Obrázek 6.2: Zobrazení síťové komunikace nástrojem Wireshark po anonymizaci. Jednotlivé červnené rámečky zobrazují atributy, které byly vybrány k anonymizaci. Lze např. vidět odstranění části dat přenášených protokolem HTTP (`File_data`) či nahrazení atributu `User-Agent` náhodným textem.

6.2 Anonymizace porušeného TCP toku

Aplikace umožňuje tři strategie řešení porušené (ztráta paketu) TCP komunikace. Tato sekce demonstruje výsledek anonymizace paketu při užití strategie `clever`, jež odstraní pouze části TCP zprávy, které nebylo možné nástrojem TShark zpracovat.

Na obrázku č. 6.3 je možné vidět záznam TCP komunikace. První paket obsahuje dvě aplikační zprávy protokolu TLS, avšak pouze první zpráva byla zpracována. Druhá zpráva nebyla zpracovaná, jelikož její data nese následující paket, který nebyl zachycen. Tuto situaci lze vyvodit z paketu číslo dva a také z informace, že první paket obsahuje TCP segment o délce 2802B (část zobrazena červeným rámečkem ve spodní části obrázku), což představuje nezpracovaná data.

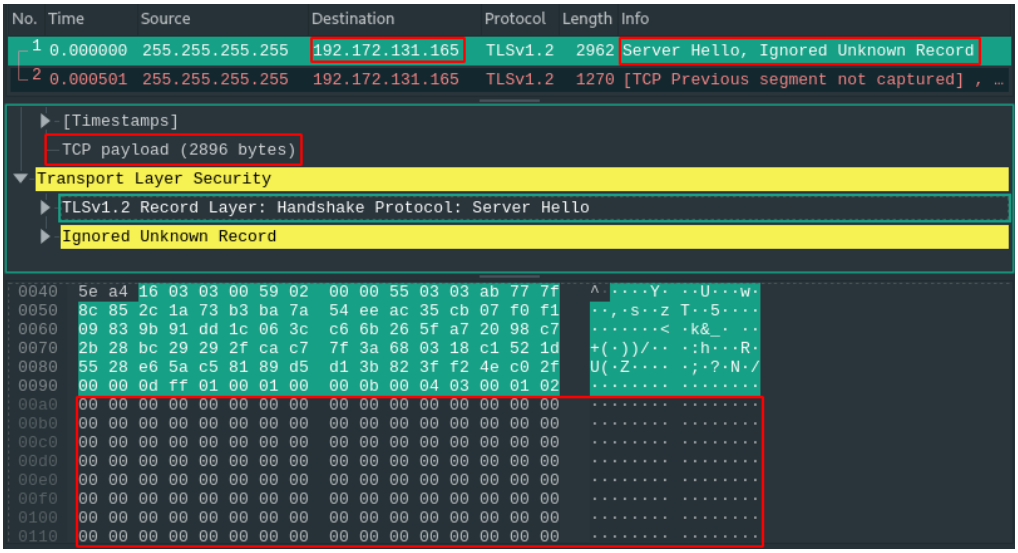


Obrázek 6.3: Zobrazení síťové komunikace nástrojem Wireshark před anonymizací. Zobrazeny jsou dva pakety představující TCP komunikaci, kde schází prostřední paket.

Síťový záznam byl anonymizován anonymizační politikou obsahující pravidlo týkající se cílové IP adresy. Pro anonymizaci adresy byla použita metoda `prefix-preserving`. Pro zpracování TCP komunikace byla užita strategie `clever`. Výsledek anonymizace ilustruje obrázek č. 6.4.

Obrázek č. 6.4 zobrazuje síťovou komunikaci po anonymizaci. Cílová IP adresa byla úspěšně anonymizovaná metodou `prefix-preserving` a nezpracovatelná data TCP komunikace byla odstraněna. Na obrázku je možné vidět, že nástroj Wireshark již nezobrazuje informace o TCP segmentu, ale pouze o TCP payloadu. Odstranění dat je možné vidět ve spodní části obrázku, kde jsou zobrazena červeným rámečkem. Z paketu fyzicky odstraněna nebyla, ale jejich hodnota je vynulována. Jelikož Wireshark ví, že se jedná o protokol TLS, označuje tato data za neznámý ignorovaný záznam.

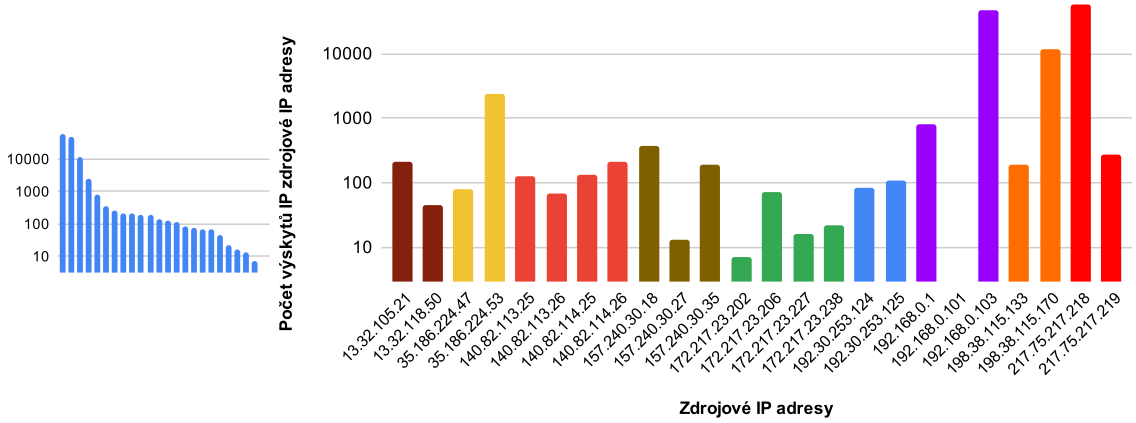
K modifikaci dochází i v paketu číslo dvě, který není na obrázcích demonstrovaný. Dochází k odstranění celého TCP segmentu, který představoval chybějící data předchozího paketu.



Obrázek 6.4: Zobrazení síťové komunikace nástrojem Wireshark po anonymizaci. Ve spodní části obrázku je možné vidět anonymizovaná data nezpracovatelného TCP segmentu paketu.

6.3 Anonymizace IP adres

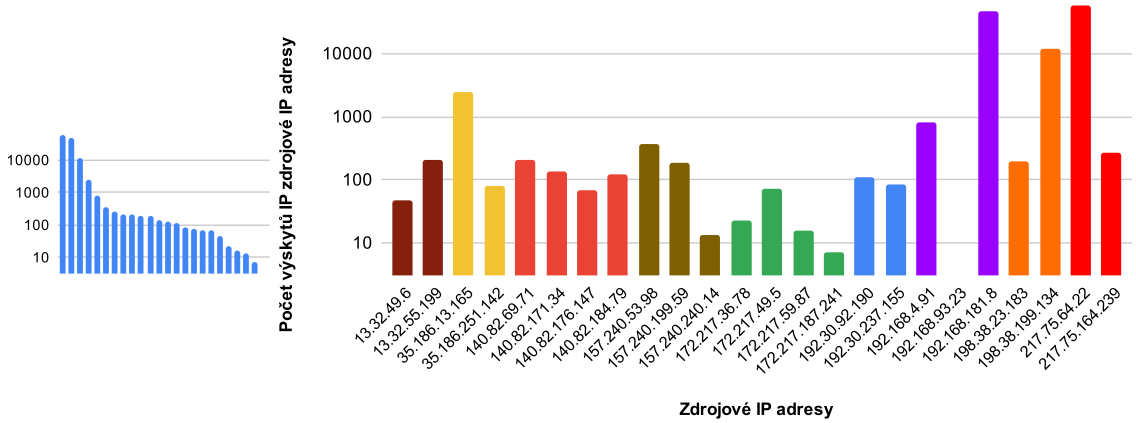
Zvolená anonymizační metoda značně ovlivňuje, jak si bude vstupní soubor podobný s výstupním – anonymizovaným – souborem. Je-li například použita metoda `permutation`, případně metoda, která kromě samotné permutace zachovává i strukturu vstupních dat, např. `prefix-preservation`, lze očekávat, že počet hodnot před anonymizací a po anonymizaci zůstane zachovaný. Naopak rozložení hodnot atributů na uzavřeném intervalu, kterým jsou anonymizované hodnoty ohraničeny bude rozdílné, jelikož původní hodnoty jsou zaměněny za jiné – anonymizované – hodnoty. Adresy, které k sobě patří, podle stejného prefixu, budou spolu i po anonymizaci.



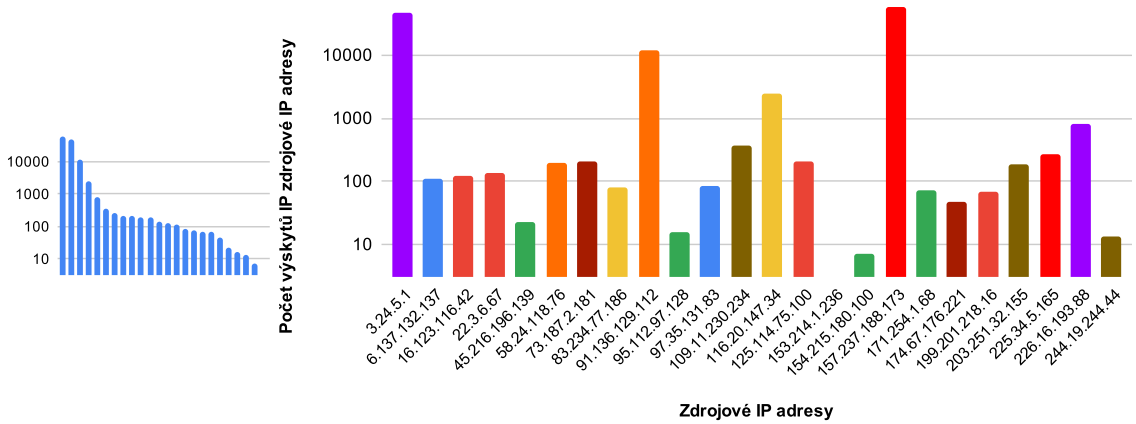
Obrázek 6.5: Histogram výskytu zdrojových IP adres v záznamu síťové komunikace před procesem anonymizace. Levá část obrázku obsahuje histogram IP adres seřazených podle jejich četností. Pravá část zobrazuje seřazené IP adresy podle jejich hodnoty. Adresy sdílející stejný 24bitový prefix jsou barevně odlišeny.

Obrázek č. 6.5 zobrazuje histogramy zdrojových IP adres anonymizovaného souboru. IP adresy, které sdílí 24bitový prefix, jsou barevně odlišeny. Adresy byly anonymizovány dvěma metodami. První metodou je `prefix-preservation (24)`. Užití této metody poskytuje větší prostor pro analýzu, jelikož zachovává vztahy jednotlivých adres, na úkor nižší míry anonymity. Druhou metodou je `permutation`, která naopak poskytuje vyšší míru anonymity, na úkor menšího prostoru k analýze, jelikož zahazuje vztahy mezi adresami.

Histogram anonymizovaných IP adres metodou `prefix-preservation (24)` zobrazuje obrázek č. 6.6. Z histogramů je patrné, že počet jednotlivých IP adres i četnost jejich výskytů zůstala zachována (levá část). Lze také vidět, že adresy, které sdílely stejný prefix před anonymizací, jej sdílejí i po anonymizaci.



Obrázek 6.6: Histogram výskytu zdrojových IP adres v záznamu síťové komunikace po procesu anonymizace metodou prefix preservation (24). Levá část obrázku obsahuje histogram IP adres seřazených podle jejich četností. Pravá část zobrazuje seřazené IP adresy podle jejich hodnoty. Vztah neanonymizovaných IP adres je barevně zachován.



Obrázek 6.7: Histogram výskytu zdrojových IP adres v záznamu síťové komunikace po procesu anonymizace metodou permutation. Levá část obrázku obsahuje histogram IP adres seřazených podle jejich četností. Pravá část zobrazuje seřazené IP adresy podle jejich hodnoty. Vztah neanonymizovaných IP adres je barevně zachován.

Případ, kdy nejsou vztahy mezi jednotlivými IP adresami zachovány, demonstruje obrázek č. 6.7. Je možné vidět, že počet IP adres včetně jejich četností zůstal zachován. Ovšem jednotlivé adresy, které k sobě patřily podle sdíleného prefixu, k sobě v anonymizované komunikaci nepatří. Barevné rozlišení jednotlivých IP adres odpovídá jejich původním – neanonymizovaným – hodnotám.

6.4 Anonymizace portů

Zdrojové a cílové porty transportních protokolů mohou napomoci ke správnému určení aplikačního protokolu, který je přenášen jako datová zpráva. Jsou-li tyto porty změněny, je možné, že aplikační zprávu nebude možné zpracovat nebo bude označena za jiný aplikační protokol.

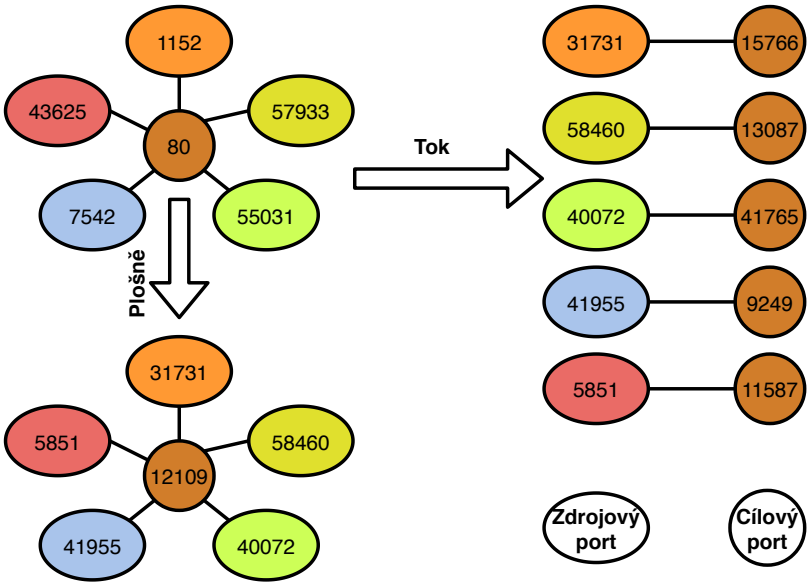
Tabulka č. 6.1 demonstruje anonymizaci portů transportního protokolu TCP. Pro demonstraci různého přístupu k anonymizaci byl vybrán aplikační protokol HTTP, jehož rezervovaný cílový port serverové strany je 80. Z anonymizovaného souboru bylo vybráno pět TCP toků, které zajišťovaly HTTP komunikaci. Sloupec **anonymizovaný plošně** představuje anonymizované hodnoty cílového nebo zdrojového portu TCP protokolu metodou `permutation`. Z dat sloupce cílového portu je možné vidět, že všechny porty s hodnotou 80 byly přemapovány na hodnotu 12 109.

Sloupec **anonymizovaný tok** představuje anonymizované hodnoty cílového nebo zdrojového portu, které byly anonymizovány v rámci TCP toku, do kterého spadají. Cílový port 80 má tak více anonymizovaných hodnot v anonymizovaném souboru, přičemž v rámci jednoho TCP toku jsou všechny hodnoty 80 mapovány na právě jednu hodnotu. Prolomení anonymizace cílového portu jednoho TCP toku tak nutně nemusí znamenat prozrazení anonymizace ostatních toků, ke kterému by v případě plošné anonymizace nastalo. Dopad rozdílného způsoby anonymizace zobrazuje obrázek č. 6.8.

Anonymizaci v rámci toku je možné aplikovat i na pakety, které jsou přenášeny transportním protokolem UDP. Anonymizovaný atribut také není nijak omezen, avšak pro některé atributy postrádá smysl – např. anonymizace IP adresy, jelikož by došlo ke ztrátě informací o komunikaci adresy v ostatních protokolech.

	Cílový port			**Zdrojový port**		
		Anonymizovaný			**Anonymizovaný**	
Index	**Original**	**Plošně**	**Tok**	**Original**	**Plošně**	**Tok**
1	80	12 109	15 766	1 152	31 731	31 731
2	80	12 109	11 587	43 625	5 851	5 851
3	80	12 109	13 087	57 933	58 460	58 460
4	80	12 109	9 249	7 542	41 955	41 955
5	80	12 109	41 765	55 031	40 072	40 072

Tabulka 6.1: Demonstrace rozdílného způsobu anonymizace portů transportního protokolu TCP. Řádky tabulky představují jednotlivé TCP toky přenášející HTTP komunikaci.



Obrázek 6.8: Dopad plošné a tokové anonymizace na zdrojové a cílové porty protokolu TCP. Kruhy představují cílové porty, elipsy zdrojové porty.

6.5 Výkonnost nástroje

Tato sekce ze zabývá výkonností implementované aplikace. Testování výkonnosti probíhalo na osobním zařízení s SSD diskem, celkovou pamětí RAM 16 GB a procesorem Intel Core i5 6 300 HQ. Testována byla časová i paměťová náročnost aplikace.

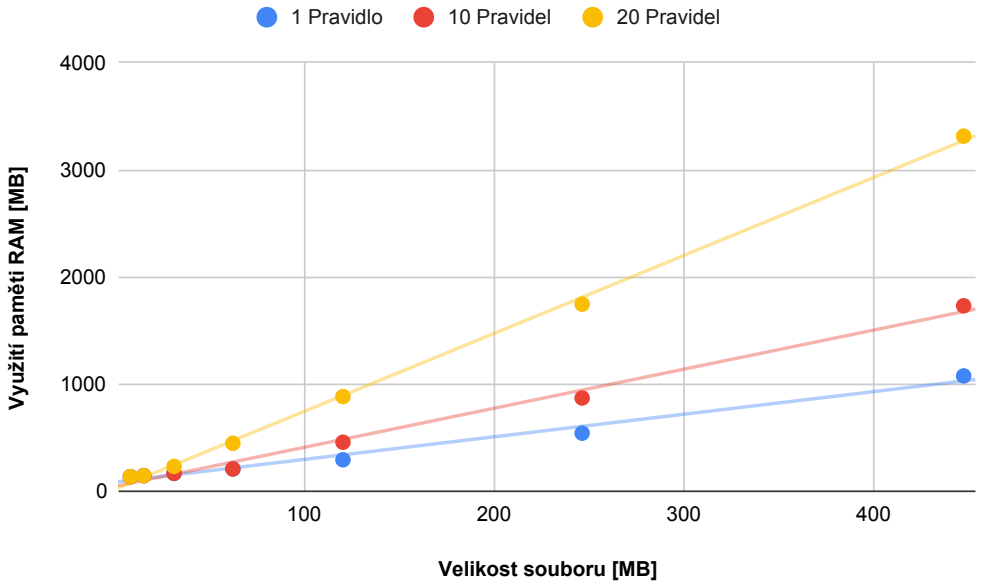
Testování probíhalo na sedmi vstupních souborech velikosti 8 MB až 447 MB. Jednotlivé soubory vznikly půlením intervalu největšího ze souborů, aby při testování obsahovaly jednotlivé soubory co nejvíce stejných dat. Zvolena byla inkrementální anonymizační politika, která pro každé kolo testování nástroje navýšila počet anonymizačních pravidel. Pro všechna kola anonymizace byla užita `clever` strategie validace TCP komunikace.

Výsledky jednotlivých kol testování zobrazuje tabulka č. 6.2. Je možné vidět, že pro menší vstupní soubory nemá počet pravidel velký vliv na dobu anonymizace. U větších souborů dochází k prodlužování dobry zpracování z důvodu možné existence většího počtu modifikací, které je nutné řadit, kopírovat či odstraňovat pro potřeby TCP komunikace. Důvodem také může být znovupoužití již anonymizovaných hodnot, které jsou ukládány do poolů a je třeba je vyhledávat.

Velikost použité paměti RAM pro anonymizaci jednotlivých souborů je počtem pravidel značně ovlivněna. Lineární závislost využití paměti RAM na počtu pravidel zobrazuje obrázek grafu č. 6.9. Nárůst je způsobený nutností existence jednotlivých instancí třídy `FieldModification` pro každou změnu vyvolanou pravidlem. Každé pravidlo ovlivňuje využití paměti RAM podle svého rozsahu – pokud pravidlo anonymizuje často se vyskytující atribut (např. IP adresa, časové razítko), může nastat situace, kdy pro každý paket existuje jedna instance modifikace daného pravidla. Nastat může i opačný případ, kdy se atribut anonymizačního pravidla v souboru nevyskytuje ani jednou.

Velikost vstupu [MB]	Počet paketů [tis.]	Počet pravidel	Dobra zpracování [s]	RAM [MB]
8	7	1	8	135
		10	8	135
		20	9	135
15	15	1	19	145
		10	16	145
		20	18	146
31	30	1	34	166
		10	36	166
		20	41	231
62	60	1	68	208
		10	77	208
		20	85	447
120	121	1	145	293
		10	159	457
		20	178	883
246	243	1	302	542
		10	329	870
		20	371	1 746
447	486	1	581	1 076
		10	662	1 730
		20	718	3 315

Tabulka 6.2: Vývoj časové a paměťové náročnosti anonymizační aplikace pro zvyšující se velikost vstupních souborů a počet pravidel.



Obrázek 6.9: Závislost počtu anonymizačních pravidel na využití operační paměti RAM.

Kapitola 7

Závěr

Cílem diplomové práce bylo navrhnout a implementovat snadno rozšířitelnou aplikaci pro anonymizaci síťového provozu ve formátu PCAP. Důvodem k vytvoření aplikace byla potřeba firmy Flowmon networks anonymizovat PCAP soubory obsahující citlivá data pro jejich následné sdílení mezi výzkumnými skupinami. Existující nástroje umožňují anonymizovat pouze malou množinu atributů a neposkytují dostatečnou míru konfigurace a rozšiřovatelnosti. Aplikace neměla být omezená pouze na některé vrstvy modelu ISO/OSI, ale měla umožňovat anonymizaci napříč všemi vrstvami. Omezení se nemělo týkat ani jednotlivých síťových protokolů či jejich atributů.

Před vlastním návrhem a implementací aplikace došlo ke zkoumání atributů vhodných k anonymizaci. Důraz byl kladen na atributy, které umožňují identifikovat uživatele či organizaci na základě informací obsažených v zaznamenané komunikaci. Pro jejich anonymizaci byly představeny některé z existujících metod, jež některé z nich aplikace implementuje.

Implementovaná aplikace staví na konzolovém nástroji TShark. Nástroj je vybraný ze dvou důvodů. Prvním důvodem je možnost zpracování vstupních dat síťové komunikace ve formátu PCAP, čímž odpadá nutnost zpracovávat jednotlivé pakety — ve smyslu transformace binárních dat do podoby protokolů — přímo implementovanou aplikací. Tato skutečnost částečně souvisí s druhým důvodem, kterým je velká podpora síťových protokolů. Lze tak říci, že všechny protokoly podporované nástrojem TShark je možné anonymizovat pomocí implementované aplikace. Jelikož je nástroj TShark úzce spjatý s nástrojem Wireshark, používají oba nástroje stejný zápis pro označení jednotlivých atributů protokolů. Jméno atributu, které je možné pouze zkopírovat z nástroje Wireshark, je tak platným názvem atributu i v implementované aplikaci, což značně usnadňuje tvorbu anonymizačních politik.

Samotná aplikace je zodpovědná za aplikaci anonymizační politiky na záznam síťové komunikace, přičemž politiku lze libovolně upravovat. Tato skutečnost aplikaci odlišuje od existujících nástrojů, které buď definovaly fixní anonymizační politiky či pouze malou možnost jejich modifikace. Implementovaná aplikace nabízí možnost definice vlastní anonymizační politiky tak, aby pokryla co nejvíce požadavky uživatelů. Tvorba vlastní anonymizační politiky přináší možnost ovlivnit dopad anonymizace na anonymizovaný soubor. Tvůrce politiky má tak možnost vybrat vhodné atributy k anonymizaci včetně anonymizačních metod tak, aby dopad na síťová data byl co nejmenší. Nevhodně zvolená anonymizační politika může síťový záznam komunikace zcela zneplatnit pro plánovaný způsob dalšího zpracování. Pro specifikaci anonymizačních politik byl zvolen formát YAML.

Anonymizační politika je tvořena anonymizačními pravidly. Každé anonymizační pravidlo obsahuje modifikátor, který představuje anonymizační metodu. Aplikace disponuje

několika modifikátory, ale je možné je do aplikace i přidávat. Lze tak pro anonymizaci užít i vlastních anonymizačních metod. Definice anonymizačního pravidla také umožňuje specifikovat, které hodnoty anonymizovaných atributů mají být anonymizovány. Lze tak definovat více anonymizačních pravidel pro stejný atribut protokolu, které se chovají jinak, např. používají jinou anonymizační metodu. Definovat lze také sdílení anonymizovaných hodnot mezi atributy, lze tak zajistit stejnou anonymizaci jednoho atributu ve více protokolech. Jako příklad je možné uvést IP adresu v protokolu IP a v protokolu DHCP.

Pomocí anonymizační politiky je také možné definovat způsob, kterým aplikace reaguje na porušené TCP toky. Jako porušený tok je myšlen tok, ve kterém došlo k výpadku alespoň jednoho paketu. Dále je sem zařazena část toku, kterou není možné zpracovat nástrojem TShark z důvodu chybějícího začátku či konce toku. Data přenášených toků je možné v anonymizovaném souboru ponechat, zcela odstranit či odstranit pouze ty aplikační segmenty, kterým nástroj TShark nerozumí a nedokáže je zpracovat.

Výsledná aplikace byla otestována z pohledu několika anonymizačních metod na vybrané atributy a jejich dopad na síťovou komunikaci. Zkoumány byly také paměťové a časové nároky aplikace při anonymizaci různě velkých souborů a různého množství anonymizačních pravidel. Doba zpracování souboru o velikosti cca 500 MB, což představuje přibližně 500 tisíc paketů, zabrala na osobním počítači kolem dvanácti minut a využila přes 3 GB operační paměti RAM, přičemž síťový záznam byl anonymizovaný dvaceti anonymizačními pravidly.

Budoucí vývoj aplikace by bylo vhodné zaměřit na definici anonymizačních pravidel. Současná aplikace umožňuje definovat jaké atributy mají být anonymizovány, přičemž je možné upřesnit, které hodnoty těchto atributů mají být anonymizovány. Ovšem chybí možnost definovat anonymizaci atributu na základě hodnoty jiného atributu. Jako příklad lze uvést odstranění dat přenášených protokolem UDP pouze pro ty pakety UDP, jichž zdrojové porty nabývají definované hodnoty.

Vhodným rozšířením aplikace by byla možnost prozkoumání vstupního souboru, kde by výsledkem byla vygenerované kostra anonymizační politiky. Kostra by byla vygenerována pro protokoly obsažené v komunikaci podle předem definované strategie protokolů. Důvodem je možnost velkého počtu použitých protokolů ve vstupním souboru, na které je snadné při tvorbě anonymizační politiky zapomenout.

Literatura

[1] Biryukov, A. Known Plaintext Attack. In: Tilborg, H. C. A. van a Jajodia, S., ed. *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2011, s. 704–705. ISBN 978-1-4419-5906-5. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_588.

[2] Boschi, E. a Trammell, B. *IP Flow Anonymization Support* [Internet Requests for Comments]. RFC 6235. RFC Editor, May 2011. Dostupné z: https://tools.ietf.org/html/rfc6235.

[3] Brekne, T. a Årnes, A. Circumventing IP-address pseudonymization. In: *Communications and Computer Networks*. Leden 2005, s. 43–48.

[4] Burkhart, M., Schatzmann, D., Trammell, B. et al. The Role of Network Trace Anonymization under Attack. *SIGCOMM Comput. Commun. Rev.* New York, NY, USA: Association for Computing Machinery. Leden 2010, sv. 40, č. 1, s. 5–11. ISSN 0146-4833. Dostupné z: https://doi.org/10.1145/1672308.1672310.

[5] Combs, G. et al. *Tshark - Dump and analyze network traffic* [online]. Wireshark. Dostupné z: https://www.wireshark.org/docs/man-pages/tshark.html.

[6] Farah, T. a Trajković, L. Anonym: A tool for anonymization of the Internet traffic. In: *2013 IEEE International Conference on Cybernetics (CYBCO)*. červen 2013, s. 261–266. ISBN 978-1-4673-6469-0. Dostupné z: https://ieeexplore.ieee.org/document/6617434.

[7] Farinacci, D., Li, T., Hanks, S., Meyer, D. a Traina, P. *Generic Routing Encapsulation (GRE)* [Internet Requests for Comments]. 2784. RFC Editor, březen 2000. Dostupné z: https://tools.ietf.org/html/rfc2784.

[8] Fuller, V., Li, T., Yu, J. J. Y. a Varadhan, K. *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy* [Internet Requests for Comments]. RFC 1519. RFC Editor, září 1993. Dostupné z: https://tools.ietf.org/html/rfc1519.

[9] GuyHarris. Libpcap File Format. *Wireshark*, 23. srpna 2015 [cit. 2020-05-11]. Dostupné z: https://wiki.wireshark.org/Development/LibpcapFileFormat.

[10] Iveson, S. IP Fragmentation in Detail. *Packet Pushers*, 26. listopadu 2019 [cit. 2020-05-11]. Dostupné z: https://packetpushers.net/ip-fragmentation-in-detail/. Path: Home; Blogs; IP Fragmentation in Detail.

[11] Jun Xu, Jinliang Fan, Ammar, M. H. et al. Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme. In: *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* IEEE, Listopad 2002, s. 280–289. ISBN 0-7695-1856-7. Dostupné z: https://ieeexplore.ieee.org/document/1181415.

[12] Koukis, D., Antonatos, S. a Anagnostakis, K. G. On the Privacy Risks of Publishing Anonymized IP Network Traces. In: Leitold, H. a Markatos, E. P., ed. *Communications and Multimedia Security.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, s. 22–32. ISBN 978-3-540-47823-2. Dostupné z: https://link.springer.com/chapter/10.1007/11909033_3.

[13] Matoušek, P. *Síťové aplikace a jejich architektura.* 1. vyd. Brno: VUTIUM, 2014. ISBN 978-80-214-3766-1.

[14] Mivule, K. a Anderson, B. A study of usability-aware network trace anonymization. In: *2015 Science and Information Conference (SAI).* IEEE, červenec 2015, s. 1293–1304. ISBN 978-1-4799-8547-0. Dostupné z: https://ieeexplore.ieee.org/document/7237310.

[15] Pang, R., Allman, M., Paxson, V. et al. The Devil and Packet Trace Anonymization. *SIGCOMM Comput. Commun. Rev.* New York, NY, USA: Association for Computing Machinery. Leden 2006, sv. 36, č. 1, s. 29–38. ISSN 0146-4833. Dostupné z: https://doi.org/10.1145/1111322.1111330.

[16] Peuhkuri, M. A Method to Compress and Anonymize Packet Traces. In: *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement.* New York, NY, USA: Association for Computing Machinery, 2001, s. 257–261. IMW '01. ISBN 1581134355. Dostupné z: https://doi.org/10.1145/505202.505233.

[17] Pfitzmann, A. a Köhntopp, M. Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology. In: Federrath, H., ed. *Designing Privacy Enhancing Technologies.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, s. 1–9. ISBN 978-3-540-44702-3. Dostupné z: https://doi.org/10.1007/3-540-44702-4_1.

[18] Postel, J. *User Datagram Protocol* [Internet Requests for Comments]. 6. RFC Editor, srpen 1980. Dostupné z: https://tools.ietf.org/html/rfc768.

[19] Postel, J. *Transmission Control Protocol* [Internet Requests for Comments]. 7. RFC Editor, září 1981. Dostupné z: https://tools.ietf.org/html/rfc793.

[20] Riboni, D., Villani, A., Vitali, D. et al. Obfuscation of sensitive data in network flows. In: *2012 Proceedings IEEE INFOCOM.* IEEE, Březen 2012, s. 2372–2380. ISBN 978-1-4673-0775-8. Dostupné z: https://ieeexplore.ieee.org/document/6195626.

[21] Slagell, A., Lakkaraju, K. a Luo, K. FLAIM: A Multilevel Anonymization Framework for Computer and Network Logs. In: *LISA '06: 20th Large Installation System Administration Conference.* Leden 2006, s. 63–77. Dostupné z: https://www.usenix.org/legacy/event/lisa06/tech/full_papers/slagell/slagell_html/.

[22] Yurcik, W., Woolam, C., Hellings, G. et al. Privacy/Analysis Tradeoffs in Sharing Anonymized Packet Traces: Single-Field Case. In: *2008 Third International Conference on Availability, Reliability and Security*. Březen 2008, s. 237–244. ISBN 978-1-4673-6469-0. Dostupné z: https://ieeexplore.ieee.org/document/6617434.

Příloha A

Obsah DVD

Přiložené DVD obsahuje následující soubory:

- Text diplomové práce ve formátu PDF:
  - soubor `text_prace_tisk.pdf` obsahuje tištěnou verzi,
  - soubor `text_prace_wis.pdf` obsahuje verzi odevzdanou do systému.
- Zdrojové kódy textové části práce v LaTeX(adresář `zdrojove_kody_textu`).
- Zdrojové kódy implementované aplikace (adresář `zdrojove_kody_aplikace`):
  - Adresář `examples` obsahuje ukázku anonymizačních pravidel, originálních a anonymizovaných souborů včetně vygenerovaných meta souborů.
  - Adresář `modifiers` obsahuje anonymizační modifikátory.
  - Soubor `prepare_environment.sh` obsahuje skript pro inicializaci pracovního prostředí.
  - Soubor `README.md` či `README.pdf` obsahují návod pro instalaci aplikace, její spuštění a ovládání. Popsán je také způsob tvorby anonymizačních pravidel a nových modifikátorů.