Masarykova univerzita
Fakulta informatiky

Rozpoznanie užívateľa na základe informácií o HTTP komunikácií

Diplomová práca

Matej Majdiš

Brno, jar 2017

Prehlásenie

Prehlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Matej Majdiš

Vedúci práce: doc. RNDr. Vlastislav Dohnal Ph.D.

Zhrnutie

TODO

Kľúčové slová

keyword1, keyword2, …

Obsah

Zoznam obrázkov

1 Úvod

S pokračujúcim vývojom nových technológií sa Internet a Web stávajú čoraz väčšou súčasťou našich životov. Web ako taký už dávno nie je limitovaný prehliadaním na počítačoch. Musí sa prispôsobovať rôznym technológiám, ako sú napríklad mobilné, či iné multimediálne zariadenia.

Problematika jednoznačnej identifikácie používateľa a jeho zariadenia v rámci Internetu je dnes veľmi dôležitou a riešenou témou. Dôvodmi sú najmä prevencia voči čoraz väčšiemu počtu rôznych typov útokov, ale aj lokalizácia a prispôsobenie obsahu konkrétnemu typu používateľa.

Z toho vyplýva potreba rozoznania a identifikácie používateľov, ktorí s danou aplikáciou interagujú. Existuje niekoľko rôznych prístupov k identifikácii, od mapovania IP adries sieťovej vrstvy až po aplikačnú správu užívateľských účtov. Každý z nich má však svoje výhody aj nevýhody. Podrobne sa nimi zaoberá kapitola 4.

Cieľom tejto práce je vytvoriť unikátny identifikátor na základe informácií dostupných najmä z *HTTP*, prípadne *TCP* protokolu. Pred zostavením samotného algoritmu je však dôležité popísať niektoré kľúčové oblasti a postupy. Nasledujúce odseky úvodu práce sa preto budú stručne zaoberať fungovaním aplikácií typu klient-server a architektúrou domény práce.

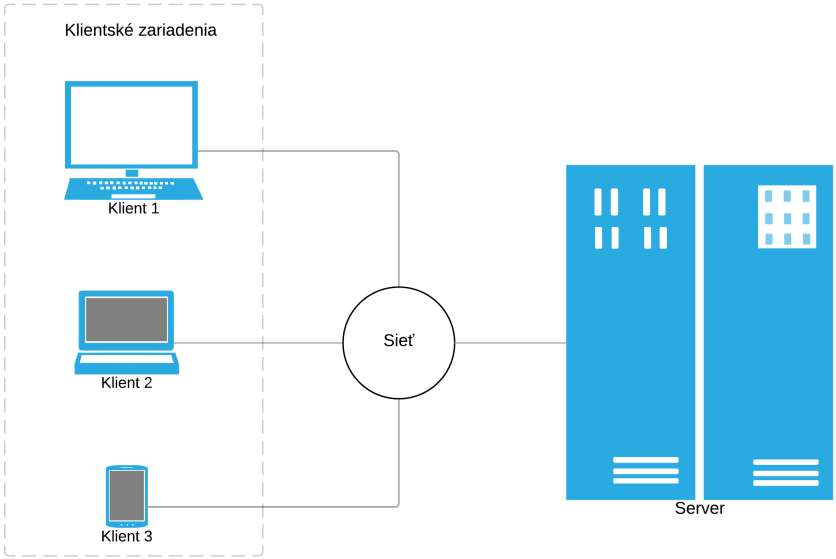
1.1 Aplikácie typu Klient-Server

Najčastejšie používaným modelom sieťovej komunikácie pre architektúru aplikácií je takzvaný klient-server. Základnou myšlienkou tohto modelu je zaslanie požiadavky (*requestu*) klientom na server, ktorý vystupuje ako poskytovateľ služby a vracia odpoveď späť klientovi.

1.1.1 Klient-Server architektúra

Pretože Klient-Server model je používaný rôznymi typmi aplikácií, bolo nutné použiť štandardizované protokoly, na základe ktorých bude možné komunikovať. Niektorými z najpoužívanejších protokolov sú: *File Transfer Protocol (FTP)*, *Simple Mail Transfer Protocol (SMTP)*

a *Hypertext Transfer Protocol (HTTP)*. Bližšie sieťové vrstvy a jednotlivé protokoly popisuje kapitola 2.



Obr. 1.1: Schéma znázorňuje základnú architektúru modelu Klient-Server, zdroj: vlastné spracovanie

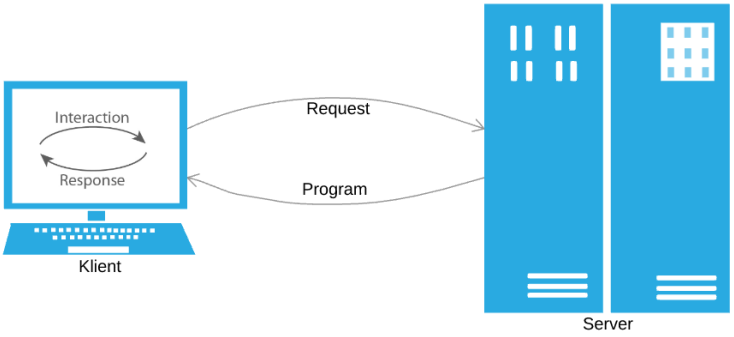
1.1.2 Architektúra

Architektúra klient-server modelu sa vo všeobecnosti typicky skladá z troch častí:

- Aplikačný server
- Databázový server
- Zariadenie klienta

Zároveň existujú dva základné typy architektúr podľa miery zapojenia zariadenia klienta:

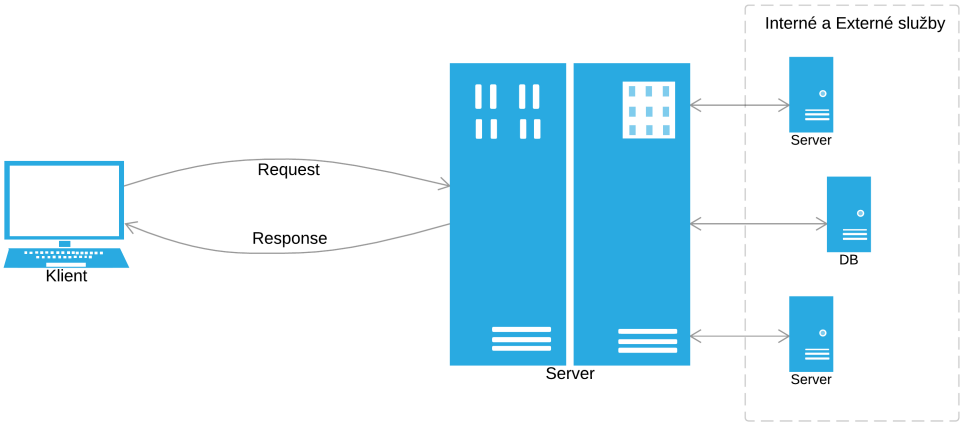
- 2-stupňová *(2-tier)*
- 3-stupňová *(3-tier)*



Obr. 1.2: Grafické znázornenie a popis priebehu komunikácie dvojstupňovej architektúry, zdroj: vlastné spracovanie

Dvojstupňová architektúra zahŕňa len zariadenie klienta a databázový server. U tohoto typu architektúry je aplikácia spustená na zariadení klienta, ktoré sa následne pripája priamo na server. Zariadenie tak obsluhuje zobrazovanie a zároveň *business* logiku aplikácie. Tento typ architektúry nazývame tučný klient (*thick client*).

Trojstupňová architektúra, ktorej aplikácie sú predmetom tejto práce, sa od dvojstupňovej líši najmä tým, že okrem zariadenia klienta



Obr. 1.3: Grafické znázornenie a popis priebehu komunikácie 3-tier architektúry, zdroj: vlastné spracovanie

a databázového serveru, zahŕňa aj aplikačný server. Tento je následne používaný na obsluhu *business* logiky aplikácie a komunikáciu s databázou, pričom zariadenie klienta slúži len na zobrazovanie. Iný názov pre takýto typ architektúry je tenký klient (*thin client*). [Olu14]

1.2 Štruktúra práce

Práca je ďalej rozdelená na dve časti, pozostávajúce z piatich kapitol, z ktorých každá sa zaoberá iným aspektom jej charakteristiky:

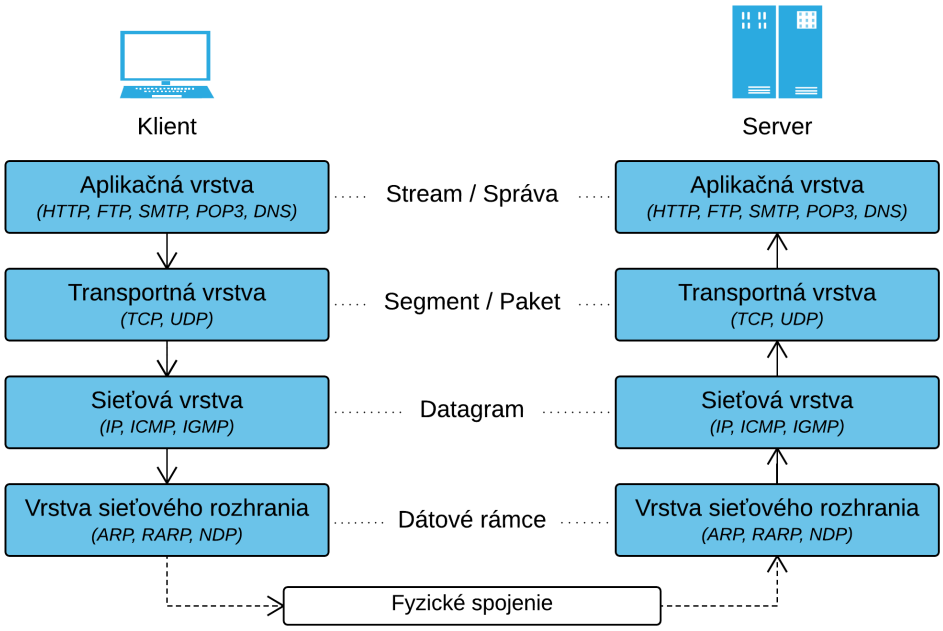
- Prvá časť práce sa zameriava na definíciu dôležitých pojmov, popis použitých technológií a porovnanie existujúcich prístupov k identifikácii.
- Druhá časť sa venuje samotnému návrhu algoritmu identifikátora, jeho implementácií a testovaniu jeho funkcionality.

Najskôr teda kapitola 2 popisuje jednotlivé sieťové vrstvy a protokoly, ktorých informácie bude možné následne použiť k tvorbe identifikátora. Nasleduje kapitola 3 popisujúca hlavnú z motivácií celej práce, ktorou sú útoky typu DOS a prevencia voči nim. Posledná kapitola prvej časti práce je zhrnutím existujúcich prístupov k rozpoznaniu užívateľov v kapitole 4.

Najdôležitejšou časťou sú však samozrejme kapitoly 5 a 6, ktoré popisujú návrh samotného algoritmu identifikátora, jeho implementáciu a následné testovanie s reálnymi dátami.

2 Sieťové vrstvy

Pre lepšie uchopenie problematiky identifikácie používateľa je nutné najskôr popísať jednotlivé sieťové vrstvy TCP/IP modelu a ich protokoly. Nasledujúci text sa preto zameriava najmä na štruktúry, ktorých pochopenie je dôležité pre ďalšie moduly teoretickej a praktickej časti práce. Ide teda o konkrétne informácie jednotlivých protokolov, z ktorých niektoré boli následne použité v implementácií identifikačného algoritmu.



Obr. 2.1: Vizualizácia vrstiev TCP/IP modelu, jeho najpoužívanejších protokolov a prenosových štruktúr, zdroj: vlastné spracovanie

2.1 Aplikačná vrstva

Na vrchole hierarchie je Aplikačná vrstva abstrakciou, ktorá špecifikuje konkrétne protokoly a metódy používané hostiteľskými uzlami v sieti. Táto vrstva je definovaná jednak v TCP/IP, ale aj OSI *(Open Systems Interconnection)* modeli sieťovej komunikácie.

Táto práca narába s modelom TCP/IP, v ktorom aplikačná vrstva definuje práve protokoly a metódy rozhraní pre komunikáciu medzi jednotlivými procesmi spájaných strán v sieti. Samotná aplikačná vrstva však len štandardizuje formu komunikácie, pričom ustálenie uceleného dátového spojenia a správu prenosu dát u aplikácií typu klient-server a *peer-to-peer* ponecháva na protokoloch nasledujúcej - transportnej vrstvy. Keďže aplikačná vrstva nijakým spôsobom nepopisuje a nešpecifikuje konkrétne pravidlá pre formu prenášaných dát, aplikácie samotné musia obsahovať logiku, ktorá zabezpečí, že obe komunikujúce strany budú v tomto ohľade navzájom kompatibilné.

Aplikačná vrstva definuje veľké množstvo používaných protokolov, ako napríklad:

- HTTP/HTTPS
- TLS/SSL poskytujúci zabezpečenú vzdialenú komunikáciu po sieti
- FTP využívaný na prenos súborov
- SMTP určený pre emailovú komunikáciu
- DNS realizujúci systém hierarchie doménových mien, a iné...

Pri tvorbe identifikátora sa však budeme venovať najmä informáciám z protokolov rodiny HTTP, konkrétne teda hlavičkám a atribútom HTTP a HTTPS.

2.1.1 HTTP

HTTP *(Hypertext Transfer Protocol)* je distribuovaný protokol určený na spoluprácu a prenos informácií medzi jednotlivými systémami pre prácu s hypermédiami. Tento protokol je bezstavový a vo všeobecnosti použiteľný nielen pre prenos hypertextu, ale aj správu DNS serverov, prípadne zložitejších objektov a systémov, v ktorých uplatní rozsiahlu škálu svojich hlavičiek a chybových hlášok. Jedným z jeho charakteristických znakov je aj možnosť voľby reprezentácie dát, čo dáva systémom veľkú nezávislosť na prenášanom formáte.

HTTP funguje na princípe požiadavky a odpovede *(request - response)*, založenom na fungovaní samotného klient - server modelu.

Klientom môže byť napríklad webový prehliadač, serverom zasa aplikácia spustená na zariadení hostiteľského uzlu. V tomto prípade teda webový prehliadač zašle správu vo forme HTTP požiadavky na server. V ideálnom prípade server požiadavku *(request)* spracuje a vygeneruje odpoveď (napríklad HTML stránku), ktorú vráti klientovi ako správu vo formáte HTTP odpovede *(response)*. Odpoveď pre klienta vždy obsahuje takzvaný *(status)*, ktorý informuje o dokončení požiadavky a prípadné telo so správou odpovede.

Celá relácia HTTP spojenia medzi dvoma uzlami je teda sekvenciou niekoľkých takýchto *(request - respone)* transakcií. Samotný prenos dát však nie je realizovaný na úrovni HTTP protokolu, ale prostredníctvom protokolu TCP na úrovni transportnej vrstvy. HTTP klient iniciuje ustálenie TCP spojenia pre špecifický port serveru (typickými portami sú 80, 443 alebo 8080). HTTP server počúva na danom porte a čaká na správu požiadavky klienta. Po prijatí správy ju už server štandardne spracuje a vráti odpoveď.

Klient a server komunikujú prostredníctvom zasielania textových správ. Požiadavka pozostáva z nasledujúcich informácií:

- Definícia požiadavky
- HTTP hlavičky (napríklad: *Accept-Language: en*)
- Prázdny riadok
- Prípadné telo správy

Takisto správa odpovede má definovaný formát podobný formátu požiadavky, ktorý sa skladá z nasledujúcich položiek:

- Status spracovania požiadavky a dôvod
- HTTP hlavičky odpovede (napríklad: *Content-Type: text/html*)
- Prázdny riadok
- Prípadné telo správy odpovede

Prvý riadok definície a každá z hlavičiek musia byť zakončené znakmi <*CR*><*LF*> (znak pre *carriage return* nasledovaný znakom *line feed*). Prázdny riadok musí zároveň pozostávať výlučne zo znakov

<*CR*><*LF*>. Z pohľadu identifikácie budú neskôr veľmi dôležité práve HTTP hlavičky. U HTTP verzie 1.1 je jedinou povinnou hlavičkou *Host* definujúci server, na ktorý bola požiadavka odoslaná.

Obr. 2.2: Príklad štruktúry HTTP požiadavky s telom, zdroj: vlastné spracovanie

2.2 Transportná vrstva

V kontexte TCP/IP modelu sieťovej komunikácie sa transportná vrstva nachádza ako druhá v poradí, medzi aplikačnou a sieťovou vrstvou. Ide o súbor protokolov a metód, ktoré poskytujú takzvanú *host-to-host* komunikáciu jednotlivých sieťových uzlov protokolom vyššie popísanej aplikačnej vrstvy. Konkrétne implementuje funkcionalitu samotného prenosu dát v dátovom toku, zabezpečuje jeho spoľahlivosť, riadenie a multiplexovanie.

Medzi najznámejšie protokoly tejto vrstvy patrí napríklad TCP *(Transmission Control Protocol)*, ktorého názov nesie aj samotný sieťový model. Tento protokol je orientovaný na prenos zložitejších spojení, oproti čomu druhý z protokolov - UDP *(User Datagram Protocol)*, je vhodný a používaný skôr pre jednoduchšie správy a spojenia. Protokol TCP je komplexnejší najmä kvôli svojmu návrhu, ktorý uchováva stav jednotlivých transakcií a zabezpečuje spoľahlivosť doručenia všetkých dátových správ.

Ďalšími významnými protokolmi transportnej vrstvy sú DCCP *(Datagram Congestion Control Protocol)* využívaný napríklad pri prenose multimédií, alebo SCTP *(Stream Control Transmission Protocol)*, ktorý sa zaoberá prenosom telefónnej signalizácie.

2.2.1 Protokol TCP

TCP je jeden z najdôležitejších a najpoužívanejších protokolov celého sieťového modelu. Pracuje s ním množstvo ďalších protokolov aplikačnej vrstvy, ako napríklad: HTTP, FTP alebo POP3. Jeho najdôležitejšou charakteristikou je pravdepodobne garancia kompletného doručenia všetkých paketov v správnom poradí. K samotným dátam aplikačnej vrstvy preto TCP pripája ku každému paketu aj takzvanú TCP hlavičku (*TCP Header*), ktorá ho dopĺňa o informácie potrebné pre dosiahnutie tejto funkcionality.

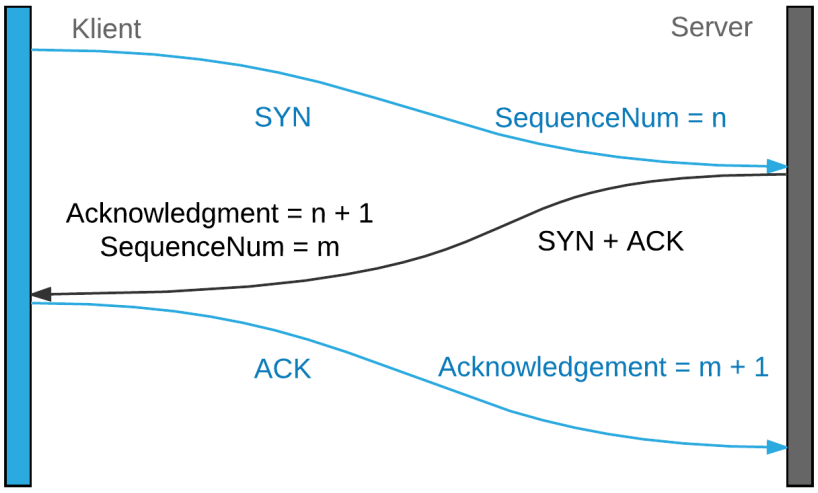
Bajty	0	1	2	3
0	zdrojový port		cieľový port	
4	poradie odoslaného bajtu *(sequence number)*			
8	poradie prijatého bajtu *(acknowledgment number)*			
12	dĺžka záhlavia	rezerva	U A P R S F R C S S Y I G K H T N N	veľkosť okna
16	kontrolný súčet		ukazovateľ naliehavých dát *(urgent pointer)*	
20	voliteľné položky			
24	voliteľné položky			výplň
28	dáta			

Obr. 2.3: Schéma popisujúca štruktúru TCP hlavičky a informácií, ktoré sú v nej uložené, zdroj: vlastné spracovanie

Niektoré informácie z týchto hlavičiek sú pomerne často využívané u existujúcich nástrojov na identifikáciu užívateľov, ktorým sa podrobne venuje kapitola 4. Ich využitie v algoritme identifikátora tejto práce následne popisuje kapitola 5.

Aby mohlo dôjsť k samotnému odoslaniu požadovaných dát, musí byť medzi klientom a hostiteľom ustálené TCP spojenie. Ustálenie aj

ukončenie tohto spojenia má presne definovanú formu komunikácie nazývanú *three-way handshake*, ktorú popisuje diagram 2.4.



Obr. 2.4: *Three-way handshake* v TCP, zdroj: vlastné spracovanie

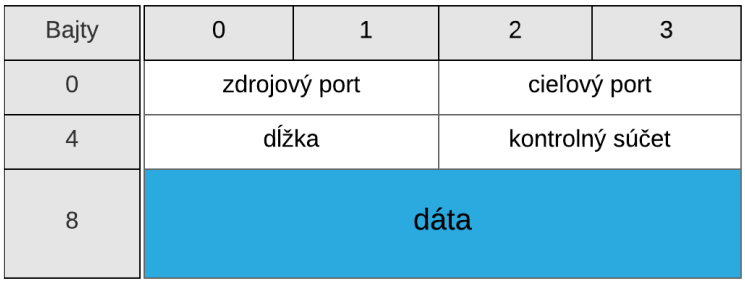
2.2.2 Protokol UDP

UDP je na rozdiel od TCP protokolom, ktorý poskytuje takzvaný nespoľahlivý prenos dát medzi uzlami. Táto nespoľahlivosť spočíva v stratovosti jednotlivých paketov počas prenosu dát, bez možnosti automatického preposielania informácií. UDP zároveň negarantuje žiadne poradie ich doručenia.

Vďaka týmto vlastnostiam je tento protokol veľmi rýchly a jednoduchý, pričom doručuje pakety nezávisle a v pomerne krátkom čase. Ďalšou z jeho typických vlastností je bezstavovosť, vďaka ktorej ho využívajú najmä systémy, ktoré odosielajú veľké množstvo malých paketov viacerým príjemcom. Ide napríklad o: DNS servery, IPTV médiá, online hry a podobne.

Hlavička UDP protokolu je veľmi jednoduchá a pozostáva len z nevyhnutných informácií, ako číslo portu odosielateľa, dĺžka paketu a špecifikácia portu príjemcu. Preto je aj samotný UDP paket viditeľne menší ako paket TCP, čo taktiež napomáha rýchlosti jeho prenosu a spracovania. Keďže však UDP poskytuje len minimálne množstvo informácií, jeho využitie pri identifikácii je minimálne.

Napriek tomu, že tieto dva protokoly sú pomerne odlišné a slúžia na rôzne účely, ich spoločnou funkcionalitou je použitie portov pre identifikáciu aplikácií komunikujúcich strán. Porty pre TCP a UDP sú na sebe navzájom nezávislé, čo prináša možnosť komunikácie jedného zariadenie zároveň v TCP aj UDP kontexte.



Obr. 2.5: Schéma popisujúca štruktúru UDP hlavičky a informácií

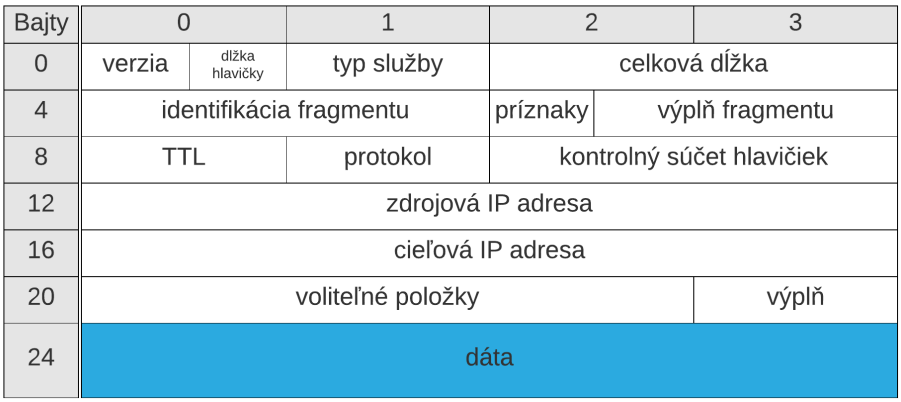
2.3 Sieťová vrstva

Sieťová vrstva je skupina internetových definícií, protokolov a špecifikácií, ktorá v sieťovom modeli TCP/IP slúži (transportnej vrstve) na prenos datagramov (paketov) od odosielateľa, cez jednotlivé sieťové rozhrania, až k príjemcovi. Každý z uzlov v sieti je na sieťovej vrstve identifikovaný takzvanou IP *(Internte Protocol)* adresou. Táto vrstva teda odvodzuje svoj názov od svojej hlavnej funkcionality, ktorou je formovanie a tvorba pripojenia k internetovej sieti prostredníctvom navzájom prepojených uzlov.

Protokoly sieťovej vrstvy pracujú s paketmi založenými na IP adresách jednotlivých zariadení. Jednoznačne najznámejším protokolom tejto vrstvy je už spomínaný IP *(Internte Protocol)*. Sieťová vrstva však definuje aj iné dôležité protokoly ako napríklad: ICMP *(Internet Control Message Protocol)*, ktorý slúži na odosielanie chybových správ operačnými systémami, alebo IGMP *(Internet Group Management Protocol)* pre podporu multicastu smerovačov. Neobsahuje však žiadne protokoly, ktoré definujú komunikáciu na úrovni lokálnej podsiete a fyzických spojení.

2.3.1 Protokol IP

IP protokol sieťovej vrstvy je zodpovedný za adresáciu hostiteľského uzla a prenos datagramu (paketu) od odosielateľa k príjemcovi cez jednu, alebo viacero IP sietí. Pre tento účel definuje IP vlastný formát hlavičky paketov, čím poskytuje logiku jednoznačnej identifikácie v rámci siete. Štruktúru hlavičky IP paketu znázorňuje nasledujúci diagram.



Obr. 2.6: Schéma popisujúca štruktúru hlavičky IP protokolu

V skutočnosti existujú dve verzie IP protokolu: IP verzia 4 a IP verzia 6. Každá z týchto verzií definuje štruktúru IP adries rozdielne. IP adresy v IPv4 sú 32 bitové, čo obmedzuje ich počet na približne 4,2 miliardy adries. Z tohto počtu sú niektoré konkrétne adresy rezervované na špeciálne účely, ako napríklad privátne siete (18 miliónov adries), alebo adresy určené pre multicast (270 miliónov). Rapídny pokles počtu voľných IPv4 adries preto vyústil do rozšírenia protokolu o IPv6. IPv6 definuje 128 bitové adresy, z čoho vyplýva, že poskytne 3.4x1038 adries. Tento počet by mal byť do budúcna viac ako dostačujúci.

Zrejme najzaujímavejšou informáciu z hľadiska identifikácie bude samotná IP adresa, ktorá okrem identifikácie zariadenia poskytuje aj informáciu o približnej polohe uzla, ktorému bola priradená. Podrobne sa problematike identifikácie pomocou IP adries venuje kapitola 4.

2.4 Vrstva sieťového rozhrania

Vrstva sieťového rozhrania pracuje v TCP/IP modeli na najnižšej úrovni. Táto vrstva popisuje samotnú sieťovú architektúru a jej komunikáciu na úrovni fyzického pripojenia. Ide o skupinu metód a komunikačných protokolov, ktoré teda operujú na fyzickom spojení dvoch uzlov.

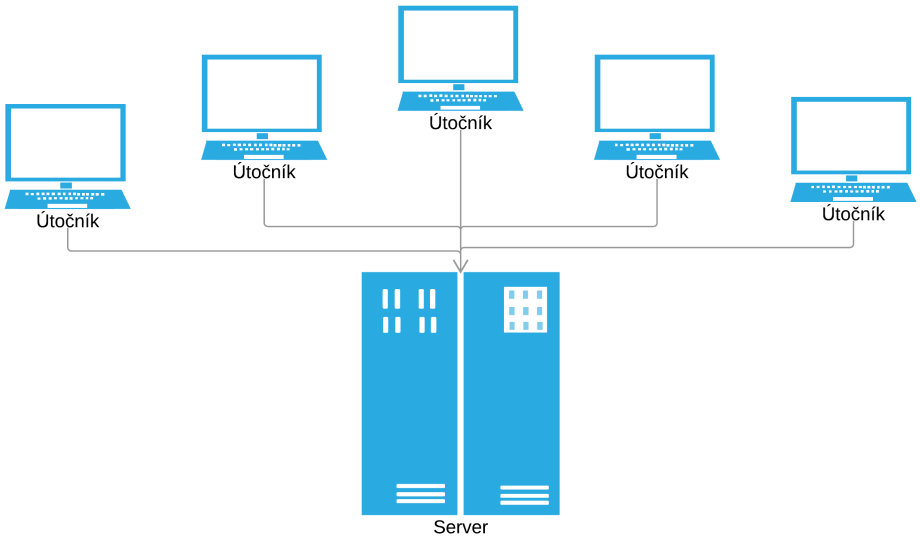
Niektorými z jej dôležitých protokolov sú: ARP *(Address Resolution Protocol)*, RARP *(Reverse Address Resolution Protocol)*, NDP *(Neighbor Discovery Protocol)*, OSPF *(Open Shortest Path First)* a iné.

Informácie týchto protokolov však operujú na príliš nízkej úrovni sieťovej infraštruktúry, preto ich nebude možné využiť pri identifikácii. V prehľade je teda vrstva sieťového rozhrania uvedená len pre úplnosť.

3 Útoky typu *Denial of Service*

Jedným z hlavných dôvodov a motivácií identifikácie užívateľov je prevencia proti útokom. Medzi najznámejšie z útokov, proti ktorým je možné brániť sa práve týmto spôsobom, patrí takzvaný útok typu *Denial of Service* (ďalej len *DoS*).

Vo všeobecnosti je za *DoS* útok považovaná snaha útočníka zabrániť oprávneným užívateľom v prístupe k informáciám, prípadne službám poskytovateľa. Snahou útočníka je znefunkčniť pripojenie neustálym narúšaním služby serveru, prípadne sieťovej infraštruktúry, v dôsledku čoho môže dôjsť k čiastočnej, či úplnej strate pripojenia hostiteľa.



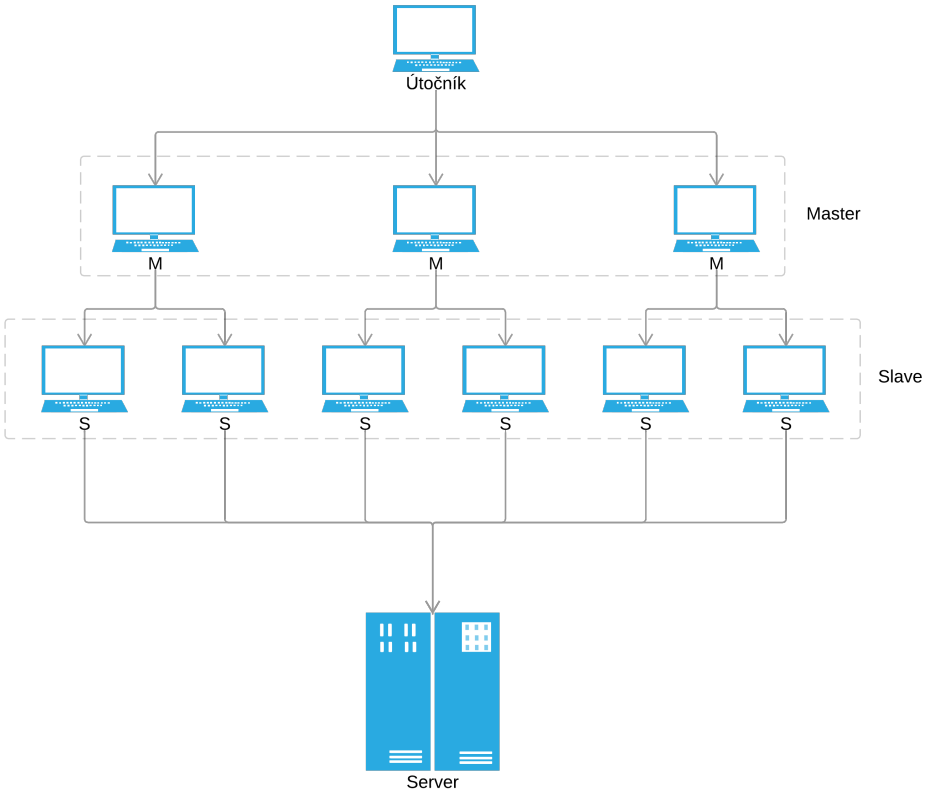
Obr. 3.1: Schéma DoS útoku, zdroj: vlastné spracovanie

3.1 Základne typy a techniky

Existuje mnoho rôznych typov a techník vykonávania *DoS* útokov. Nasledujúce odseky preto popisujú najpoužívanejšie z nich a identifikujú prostriedky, pomocou ktorých je možné sa im brániť.

3.1.1 Distribuované DoS útoky

O distribuovaných *DoS* útokoch hovoríme v prípade, že viaceré zariadenia zaplavia celú šírku pásma, prípadne zdrojov cieľového systému, ktorým je zvyčajne jeden, alebo viacero serverov. Takýto útok je často dôsledkom použitia rôznych systémov a zariadení (napríklad takzvaného *botnetu*), ktoré sa snažia vyťažiť cieľový systém. *Botnet* je rozsiahla virtuálna sieť umelých (*zombie*) počítačov, ktorých cieľom je prijímať príkazy bez vedomia majiteľa. Keď cieľový systém spotrebuje všetky voľné spojenia, ďalšie už nie je možné nadviazať.



Obr. 3.2: Schéma rozloženia DDoS útoku, zdroj: vlastné spracovanie

Hlavné výhody útočníka pri využití Distribuovaného *DoS* útoku spočívajú v skutočnostiach, že viaceré zariadenia dokážu generovať väčšiu záťaž ako jedno, pričom použitie množstva systémov zabez-

pečuje omnoho ťažšiu detekciu jeho identity. Správanie sa každého z týchto zariadení je zároveň horšie pozorovateľné, čo sťažuje obranu voči samotnému útoku.

Tieto výhody taktiež spôsobujú vývoj obranných mechanizmov. Na strane cieľového serveru už nebude stačiť jednoduché zvýšenie šírky pásma nad hranicu momentálnej veľkosti útoku, pretože útočník môže napríklad zvýšiť počet zapojených zariadení, čím by taktiež spôsobil zaťaženie a výpadok systému.

3.1.2 Sémantické DoS útoky

Sémantické útoky využívajú špecifickú funkcionalitu alebo implementačnú chybu aplikácie, prípadne protokolu zariadenia obete na zneužitie určitého množstva jeho zdrojov. Napríklad v prípade *TCP SYN* útoku je touto zneužitou funkcionalitou alokácia značného množstva priestoru čakajúcich pripojení ihneď po potvrdení *TCP SYN requestu*. Útočník otvorí viaceré spojenia, ktoré nikdy neuzavrie, čím zahlcuje server.

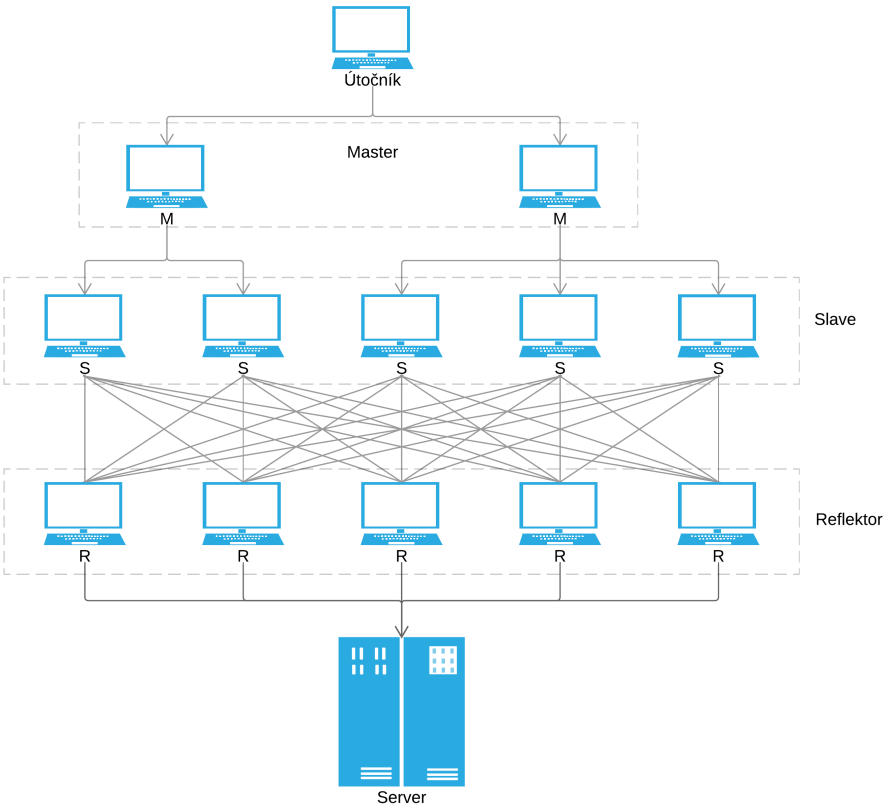
Pri *CGI* útoku je zasa cieľom útočníka takýmto spôsobom zahltiť procesor viacerými *CGI requestami*.

Jedným z obzvlášť nebezpečných útokov je *NAPTHA* útok, ktorý sa zameriava na *TCP* protokol. Inicializuje mnoho *TCP* spojení, ktoré zaplnia zdroje serveru.

3.1.3 Reflexia a amplifikácia

U reflexie (*DRDoS*) je za reflektujúce zariadenie považovaná akákoľvek hostiteľská IP adresa, ktorá v prípade odoslania paketu, tento paket spätne aj vráti. Skupiny útočníkov cielene organizujú nimi kontrolované hostiteľské servery tak, aby sa zasielali falošné dáta, simulujúce napadnutý uzol, až ku reflektujúcemu serveru. Vo výsledku sú teda obete napadnuté z výrazne väčšieho počtu zdrojových bodov, obsiahlo rozptýlených v sieti, čím útočníci účelne odrezávajú ich spojenie zo zvyškom Internetu.

Amplifikačný útok je typ reflektujúceho útoku, kde reflektujúci server odosiela mnohonásobne väčší objem odpovedí, než prijíma. Týmto spôsobom teda znásobuje aj vyťaženie spojenia medzi obeťou a zdrojovým uzlom.



Obr. 3.3: Schéma rozloženia DRDoS útoku, zdroj: vlastné spracovanie

3.2 DoS nástroje a DoS as a Service

Typickou metódou prenosu mechanizmov *DoS* a *DDoS* útokov je malvér. Jedným z príkladov bol takzvaný *MyDoom*. Ide o *DoS* mechanizmus, ktorý sa spúšťal vo vopred naplánovanom čase. Tento útok zahŕňal nastavenie hodnoty *IP* adresy cieľového systému pre nasadenie malvéru, pričom pre spustenie útoku nebola potrebná žiadna interakcia s používateľom.

Ďalším spôsobom zneužitia systému pre *DDoS* útok je použitie skrytej časti softvéru tretej strany, ktorý umožní útočníkovi stiahnutie *zombie* agenta, prípadne ho už softvér sám obsahuje. Útočník môže preniknúť do systému taktiež pomocou automatizovaných nástrojov, ktoré zneužívajú chyby v programoch počúvajúcich vzdialené pripo-

jenia. Tento scenár zasahuje primárne systémy, ktoré sa správajú ako webové servery. Typickým príkladom *DDoS* nástroja z tejto oblasti je takzvaný *Stacheldraht*. *Stacheldraht* využíva vrstevnatú štruktúru, v ktorej útočník používa program klienta na pripojenie sa k *handlerom*, ktoré sú zneužívané na prenos príkazov k *zombie* agentom. Agenti následne vykonávajú samotný *DDoS* útok. Agenti sú cez handleri zneužívaní útočníkom za pomoci použitia automatizovaných algoritmov na vyhľadávanie zraniteľností v programoch, ktoré prijímajú vzdialené pripojenia. Každý *handler* je schopný kontrolovať až tisíc agentov.

DDoS nástroje ako *Stacheldraht* stále používajú klasické *DoS* metódy zamerané na zosilnenie a podvrhovanie IP adries, ako napríklad útok vyťaženia šírky pásma. Ďalšou z možností je zahltenie zdrojov - *SYN Flood* útok. Novšie nástroje používajú na *DoS* útoky taktiež *DNS* servery.

Nástroje ako *MyDoom* môžu byť použité voči ľubovoľnej IP adrese. Menej skúsení útočníci ich používajú k znemožneniu dostupnosti populárnych a známych webových serverov. Naopak sofistikovanejší útočníci používajú tieto nástroje na vydieranie, napríklad voči svojim obchodným protivníkom.

V niektorých prípadoch sa však môže zariadenie stať časťou *DDoS* útoku zámerne - so súhlasom majiteľa. Príkladom je distribuovaný útok *Operation Payback* organizovaný skupinou *Anonymous*.

3.3 DoS - Zhrnutie

Ako je možné vidieť z predchádzajúcich odsekov, existuje naozaj veľké množstvo variácií *DoS* a *DDoS* útokov. Zároveň ich počet postupom času narastá a preto je nutné vyvíjať nové nástroje na obranu voči nim. Najťažšou časťou je však takýto útok, najmä jeho pôvodcu, odhaliť a identifikovať. Jedným z využití algoritmu popísaného v kapitole 5 a implementačnej časti je preto práve táto funkcionalita.

4 Existujúce prístupy k identifikácií

Dôležitou súčasťou tejto práce je popis existujúcich prístupov, ktoré sú aktuálne využívané na účely identifikácie užívateľa. Práve ich porovnaniu a hodnoteniu sa venuje nasledujúca kapitola. Ide najmä o techniky využívajúce protokoly sieťovej vrstvy (najmä IP adresy), monitorovanie TCP komunikácie a vlastné aplikačné identifikátory.

4.1 Využitie sieťovej vrstvy

Jedným z najtypickejších spôsobov identifikácie užívateľa a jeho zariadenia je použitie informácií sieťovej vrstvy, predovšetkým jej IP protokolu. Táto technika je veľmi rozšírená napríklad pre účely blokovania prístupu na server, či u systémových *firewallov*.

4.1.1 Možnosti IP protokolu

Ako popisuje kapitola 2, *Internet Protocol* slúži na prenos paketov medzi jednotlivými sieťovými uzlami - zariadeniami, ktoré sú identifikované IP adresami. V ideálnom svete by bolo každé zariadenie v sieti identifikované jednou statickou IP adresou. Bolo by teda možné užívateľa rozpoznať len na základe množiny jeho adries. Bohužiaľ adresový priestor protokolu IPv4 je obmedzený na 3.7 miliardy verejne dostupných adries [1]. Z toho vyplýva potreba metód ich dynamického prideľovania, proxy serverov a prekladu, čo identifikáciu značne komplikuje.

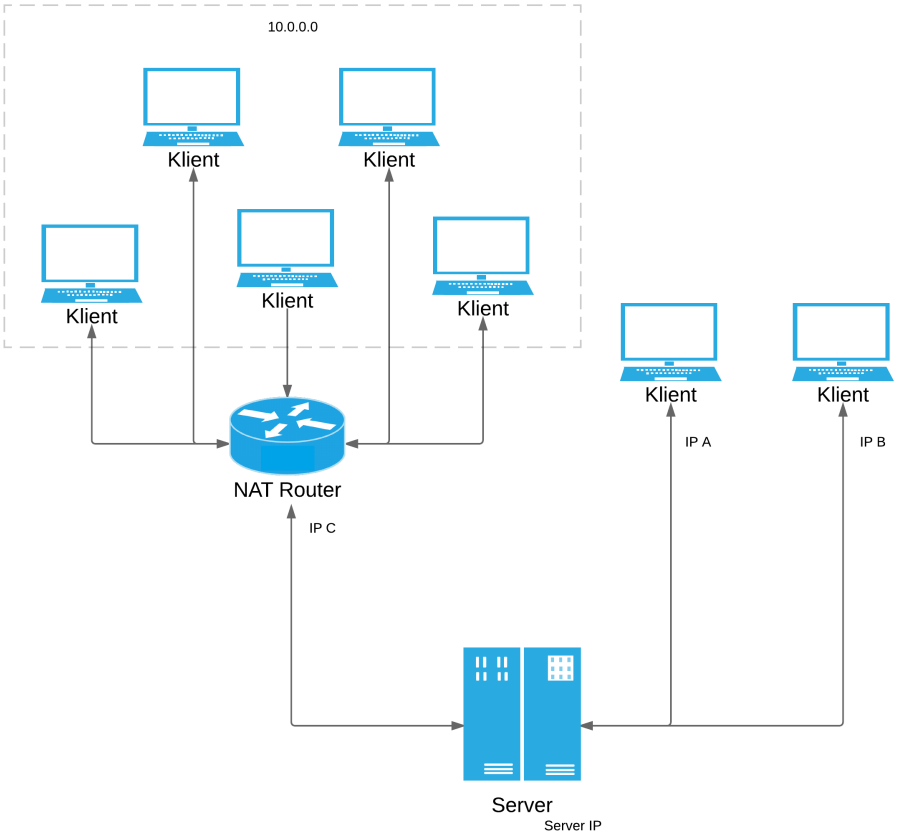
U dynamického prideľovania adries ide o jednoduché mapovanie pre zariadenia na úrovni smerovača. Identifikácia je teda stále možná, len sťažená o vyšší počet možných adries pre jedno zariadenie.

Preklad (*Network Address Translation*) IP adries však identifikáciu prakticky znemožňuje. Vo všeobecnosti ide o techniku prekladu jednej podmnožiny IP adries na inú pomocou zmeny informácie o sieťovej adrese v datagrame IP protokolu. Pôvodne sa táto technika používala pre zjednodušenie presmerovania komunikácie bez nutnosti opätovnej adresácie každého uzla.

1. Uvedené množstvo zahŕňa všetky možné dostupné IPv4 adresy - približne 4.2 miliardy po odpočítaní rezervovaných adries - 588 miliónov.

V pokročilých implementáciách však dnes *NAT* predstavuje veľmi populárnu možnosť takzvaného IP maskovania. Maskovanie IP adries je technika zdieľania jednej verejnej IP adresy celou privátnou podsieťou. Adresný priestor privátnej podsiete, ktorá má byť skrytá je teda vždy mapovaný na verejnú IP adresu smerovača. Táto adresa samotná je taktiež zvyčajne súčasťou väčšej podmnožiny adresného priestoru. Pojem prekladu (*NAT*) IP adries sa dnes už stal synonymom ich maskovania.

Paket odoslaný zo zariadenia, ktoré je prekryté *NAT* smerovačom bude teda ako zdrojovú adresu niesť namiesto IP adresy zariadenia, z ktorého pochádza, hodnotu verejnej IP adresy smerovača. Identifikácia na základe IP adresy preto v tomto prípade stráca význam.



Obr. 4.1: Vizualizácia rozloženia IP adresového priestoru pri použití techniky prekladu *NAT*, zdroj: vlastné spracovanie

4.1.2 Výhody a nevýhody IP

Hlavnými výhodami použitia IP adries sú jednoduchosť, rýchlosť a flexibilita. Napríklad pre účely znemožnenia prístupu stačí filtrovať len povolené IP adresy, prípadne neumožniť prístup tým zakázaným. Oba spôsoby sú technicky veľmi ľahko implementovateľné.

Ich nevýhodou je najmä rastúci trend použitia proxy serverov a *NAT* prekladu, čo má napríklad pri banovaní za následok znemožnenie prístupu aj užívateľom, u ktorých to nie je potrebné, ale stoja za rovnakou IP adresou.

Vo všeobecnosti je teda identifikácia zariadení používateľa na základe IP adresy možná. V prípade, že sa však zariadenie skrýva za proxy serverom, prípadne sú adresy prekryté *NATom* bude táto technika viesť takisto k nechcenému prekrytiu jednotlivých užívateľov vrámci podsiete.

4.2 Monitoring TCP

Ako popisuje kapitola 2, TCP *(Transmission Control Protocol)* je jedným z protokolov transportnej vrstvy sieťovej hierarchie, ktorého najtypickejším znakom je na rozdiel od UDP garancia spoľahlivého doručenia paketov v presnom poradí. Prijatie každého z paketov musí byť potvrdené príjemcom, inak je paket odoslaný znovu. Poradie jednotlivých paketov je zaručené ich sekvenčnými číslami. Tieto vlastnosti robia z TCP jasnú voľbu pre aplikácie, ktoré vyžadujú nulovú stratovosť dát.

Aplikácie typicky zasielajú na TCP vrstvu *stream* dát pomocou takzvaných *stream socketov*, čím sú dáta *streamu* rozdelené na primerane veľké segmenty. K segmentom je zároveň spočítaný kontrolný súčet-*TCP checksum*, pomocou ktorého je možné určiť, či dáta neobsahujú poškodené pakety. Kontrolné súčty však nie sú v žiadnom ohľade kryptograficky zabezpečené.[2]

Samotný TCP paket neobsahuje žiadne dáta, ktoré by mohli byť využité na priamu identifikáciu používateľa (pokiaľ takéto informácie nie sú obsiahnuté v nešifrovanej dátovej časti paketu), čo však neplatí pre IP protokol nižšej úrovne. Zdrojové a cieľové porty TCP v spolu-

2. Podrobnému popisu štruktúry TCP paketu sa taktiež venuje kapitola 2.

práci s IP adresou odosielateľa a príjemcu môžu pomôcť identifikovať obe zúčastnené strany.

TCP je však dobrým príkladom doplnenia identifikácie na základe niekedy neúplných informácií monitorovanej komunikácie. Táto neúplnosť spočíva napríklad v spôsobe inicializácie položky sekvenčného čísla. Jej výpočet je definovaný, počiatočná hodnota však zostáva neznáma. Možnosť výberu zostáva na vývojárovi, pričom najčastejšie ide o náhodne zvolené číslo. To isté platí pre veľkosť TCP okna (*TCP Window*) používaného na ovládanie preťaženia. Nakoľko riadenie preťaženia má kľúčový vplyv na celkový výkon TCP, veľa pokusov o optimalizáciu je vykonávaných už výrobcom operačného systému.

Všetky podobné implementačne závislé informácie bývajú používané na takzvané zaznamenávanie *TCP/IP zásobníka*. Cieľom tejto techniky je odhadnúť, aký *zásobník* a operačný systém beží na vzdialenom počítači.

Zaznamenávanie TCP/IP (ako aj mnoho jeho ďalších spôsobov) môže byť vykonané aktívne alebo pasívne, pozorovaním alebo upravovaním. Pasívne pozorovanie je založené na monitorovaní sieťových odkazov a pokusoch odhadnúť daný TCP/IP zásobník z odpočúvaných dátových paketov. Aktívny pozorovací pokus však dátové pakety aj zasiela. Útočník úmyselne vyberá také pakety, aby získal čo najviac informácií od vzdialeného hostiteľa. Avšak, keďže ide o pozorovací útok, útočník zvyčajne prispôsobí svoje správanie v závislosti od protokolu. Pravidlá protokolu by mohol samozrejme aj porušiť, čo by mu dalo väčšiu šancu uhádnuť TCP/IP zásobník, avšak, týmto by sa jeho útok stal nápadnejším a zreteľnejším.

4.2.1 Výhody a nevýhody TCP

Jednou z pomerne často používaných vlastností TCP je možnosť odhadu informácií o komunikujúcej aplikácii z portov tejto vrstvy bez nutnosti prístupu k samotným dátam. Je to možné vďaka skutočnosti, že na internete sú používané štandardné porty, ako napríklad port 25 pre požiadavky FTP alebo port 80 pre požiadavky HTTP.

Pokiaľ ide o ostatné TCP atribúty, ich využitie v úplnej identifikácii užívateľa je minimálne, avšak v kombinácií s dátami ostatných protokolov budú pomerne nápomocné.

4.3 Aplikačné identifikátory

Na aplikačnej úrovni je identifikácia užívateľa oveľa komplexnejším a do istej miery aj abstraktnejším procesom. Z pohľadu systému sú jeho používateľmi typicky ľudia, prípadne procesy iných služieb, ktoré existujú mimo neho. Identifikátorom (ďalej len ID) konkrétneho užívateľa je teda projekcia aktuálneho jednotlivca, prípadne procesu, do počítačového systému.

Systém používa typicky abstraktný objekt - účet užívateľa, ktorý obsahuje množinu atribútov pre každého jednotlivca, alebo proces. Tento objekt má jednoznačné a unikátne ID, prípadne meno, ktorým je reprezentovaný v rámci systému. Ďalej môže objekt obsahovať dodatočné atribúty, ktoré ho popisujú. Atribúty môžu, ale nemusia zahŕňať osobné údaje jednotlivca. Odhliadnuc od ID objektu, bezpečný systém typicky priradí každému z používateľov jednoznačný identifikátor (číslo, alebo reťazec), ktorým sa odkazuje na abstraktný objekt reprezentujúci aktuálnu entitu.

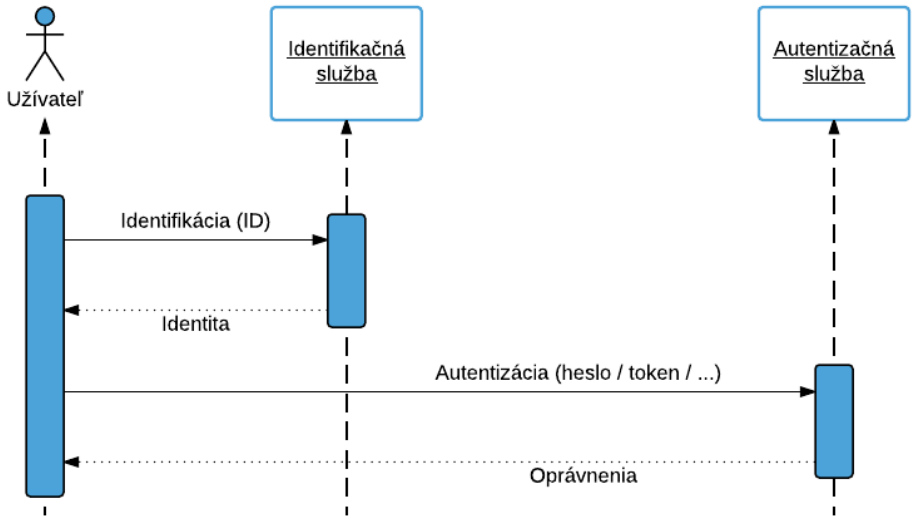
Tvorba unikátneho abstraktného objektu vo forme užívateľského účtu pre každého jednotlivca alebo proces komunikujúci so systémom je na aplikačnej úrovni veľmi dôležitá. Tento objekt je následne používaný na identifikáciu užívateľa v rámci celého systému. Zároveň tento objekt slúži ako odkaz pre jednotlivé akcie systému umožňujúce prístup k údajom komunikujúcej entity. ID používateľa sa tak stáva základom pre kontrolu prístupu. Z toho dôvodu je nevyhnutné uchovávať unikátne ID pre každého používateľa, keďže každý z nich môže mať rozličné požiadavky a individuálne zodpovednosti v rámci akcií tohto systému.

Metóda identifikácie systému poskytuje na aplikačnej úrovni jednoznačnú identitu používateľa. Táto identita je typicky reprezentovaná jeho identifikátorom. Systém preto prehľadáva všetky dostupné abstraktné objekty a vráti objekt vyhovujúci privilégiám a atribútom entity s ktorou aktuálne komunikuje. Po úspešnom dokončení tohto procesu je užívateľ jednoznačne identifikovaný.

Po identifikácii typicky nasleduje krok validácie získanej identity, vo všeobecnosti nazývaný autentizácia používateľa. Fakt, že užívateľ tvrdí, že je reprezentovaný špecifickým abstraktným objektom, nemusí totiž nutne znamenať, že je to pravda. Pre potvrdenie, že aktuálny používateľ môže byť skutočne mapovaný na konkrétny abstraktný

objekt, čim mu budú udelené jeho práva a privilégiá, musí komunikujúci preukázať svoju identitu systému. Autentizácia je teda procesom validácie danej poskytnutej identity. Informácie, ktoré predkladá komunikujúca entita, sa nazývajú poverenia (*credentials*). Tieto poverenia sa môžu v rôznych systémoch líšiť, prípadne môžu niektoré systémy vyžadovať ich väčšie množstvo. Najčastejšími formami týchto údajov sú: meno, heslo, pin, token, prípadne certifikáty, a iné.

Akonáhle prebehne úspešná autentizácia, môže používateľ vykonať akcie, na ktoré ma oprávnenia. Všetky akcie, ktoré vykoná, sú zviazané s jeho identitou a je ich preto možné dodatočne trasovať.



Obr. 4.2: Grafické znázornenie procesu identifikácie užívateľa na aplikačnej úrovni a jeho následnej autentizácie, zdroj: vlastné spracovanie

4.3.1 Výhody a nevýhody

Identifikátory na aplikačnej úrovni sú veľmi jednoznačné a presné z hľadiska priradenia účtu užívateľovi. Je pomocou nich preto následne možné presne sledovať jeho akcie a manipulovať s privilégiami.

Tieto identifikátory sa však v abstrakcii komunikácie nachádzajú príliš vysoko, preto nie je pomocou nich možné ovplyvniť napríklad DoS útoky popísané v kapitole 3.

4.4 Zhrnutie

Na identifikáciu užívateľa, prípadne jeho zariadenia, sú v súčasnosti používané rôzne techniky na rôznych vrstvách sieťovej infraštruktúry, od IP adries sieťovej vrstvy, cez atribúty TCP protokolu, až po komplexné aplikačné identifikátory.

V praktickej časti tejto práce, ktorá sa zaoberá návrhom a implementáciou unikátneho identifikátora, je použitá kombinácia informácií zo všetkých spomínaných vrstiev (IP adresy, informácie z HTTP a TCP protokolu, ...) tak, aby vznikol čo najpresnejší možný odtlačok aktivity jedného užívateľa.

5 Tvorba unikátneho identifikátora

Hlavným cieľom a výstupom tejto práce je algoritmus definujúci jednoznačný identifikátor užívateľa. Nasledujúca kapitola sa zaoberá práve popisom jeho fungovania, štruktúry, použitých technológií a následnou implementáciou. Na jeho tvorbu boli použité informácie HTTP a TCP protokolu popísané v kapitolách 2 a 4. Vďaka nim je algoritmus schopný rozpoznať a priradiť aktivitu každému z používateľov hostiteľského systému.

Princípom fungovania algoritmu je spúšťacia akcia, ktorou je prijatie HTTP requestu hostiteľským serverom.[1] Na základe každého takéhoto requestu je následne spustená jeho analýza a porovnávanie. Výstupom algoritmu je miera podobnosti aktuálneho requestu v podobe jeho vzdialenosti od najbližšieho podobného requestu z databázy. Doplnkovou funkcionalitou je aj možnosť upozornenia hostiteľskej aplikácie pri prekročení veľkého množstva podobných requestov od jedného používateľa.

5.1 Popis Algoritmu

Vo všeobecnosti pozostáva celý algoritmus z piatich hlavných, po sebe nasledujúcich krokov:

1. Zachytenie prebiehajúceho requestu pred samotným spracovaním logikou hostiteľskej aplikácie.
2. Zhromaždenie HTTP a TCP informácií daného spojenia.
    - HTTP dáta sú parsované zo zachyteného requestu.
    - TCP dáta poskytuje samostatný sieťový monitor.
3. Spracovanie a serializácia informácií do podoby objektu unikátneho odtlačku aktuálneho requestu.
4. Analýza a nájdenie najbližšieho podobného identifikátora v databáze pomocou funkcie na hľadanie najmenšej vzdialenosti.

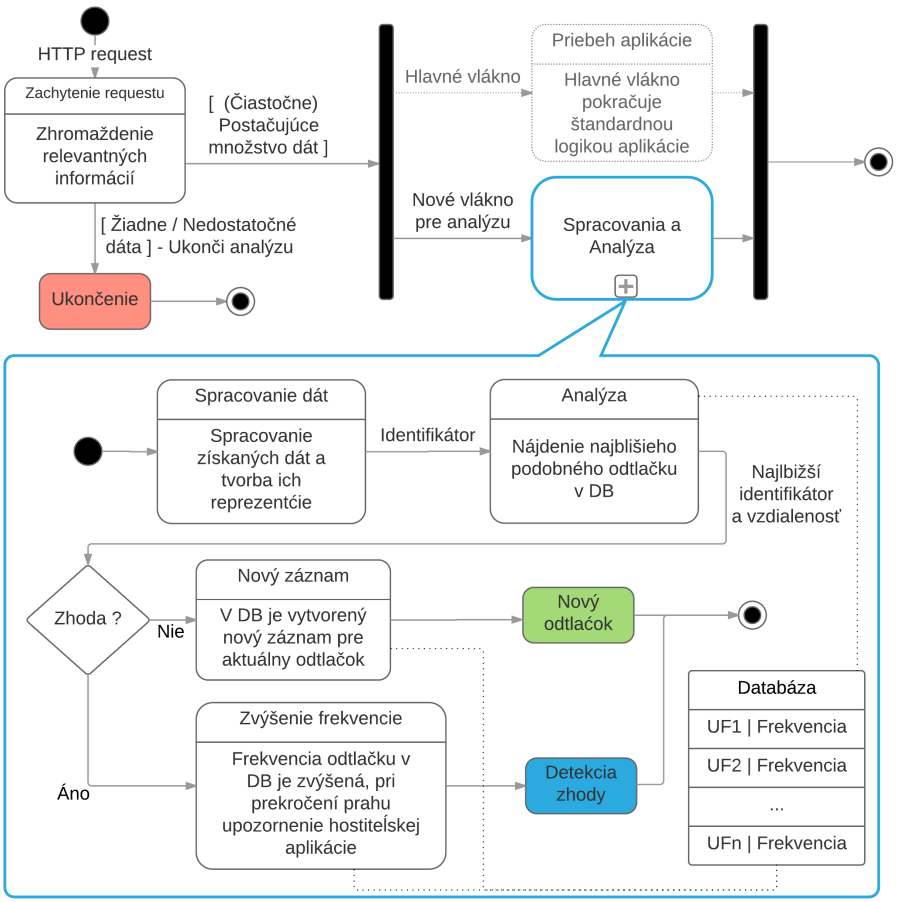
1. Hostiteľským serverom je typicky backendová aplikácia, ktorá obsahuje implementáciu algoritmu ako nezávislý modul.

5. Spracovanie výsledku v podobe vzdialenosti odtlačkov a následná reakcia.
    - Ak ide o novú reláciu, pre identifikátor je vytvorený nový záznam v databáze s frekvenciou 1.
    - V prípade, že miera podobnosti prekročila stanovený prah, nevytvára sa nový záznam. Frekvencia daného odtlačku v databáze je však zvýšená.
    - Ak je frekvencia výskytu konkrétneho odtlačku veľmi vysoká existuje možnosť upozornenia hostiteľskej aplikácie.

Ako je podrobne znázornené na diagrame 5.1, analýzu spúšťa každý nový HTTP request, na ktorý ďalej nadväzujú akcie tvorby a porovnávania identifikátorov. Zároveň je pri štarte aplikácie spustené vlákno vykonávajúce sieťovú analýzu transportnej vrstvy, ktorá uchováva informácie z niekoľkých posledných spojení. Z týchto HTTP a TCP informácií je teda pre každý request zostavený unikátny odtlačok - identifikátor *(Unique Footprint - UFoo)*. Aktuálny identifikátor je následne porovnávaný s každým odtlačkom v databáze. Algoritmus na hľadanie najbližšieho suseda k nemu následne pomocou porovnávacej funkcie nájde najbližší identifikátor. Podľa miery podobnosti ďalej algoritmus určí, či ide o requesty jedného užívateľa, čo spôsobí zvýšenie jeho frekvencie, alebo ide o nový odtlačok, pre ktorý vytvorí nový záznam. Ako bolo popísané vyššie, jednou z dodatočných funkcionalít je aj upozornenie hostiteľskej aplikácie pri prekročení limitu frekvencie requestov od jedného užívateľa, čo by mohlo indikovať napríklad DoS, prípadne DDoS útok.

5.2 Štruktúra identifikátora

Odtlačok aktivity užívateľa je teda kombináciou špecificky vybraných HTTP a TCP informácií. Objekt sa skladá z dvoch častí - statických a relačných dát. Dôležitejšou z nich sú takzvané statické dáta, ktoré slúžia porovnávacej funkcii na samotné určenie vzdialenosti dvoch identifikátorov. Vzdialenosť vypočítaná touto funkciou môže byť ďalej ovplyvňovaná vzťahmi relačných dát, ktoré riešia špecifické aspekty vybraných informácií, ako napríklad určenie menej bezpečných geozón geolokáciou IP adresy klienta.



Obr. 5.1: Diagram znázorňujúci celý proces algoritmu spracovania requestu a hľadania podobnosti, zdroj: vlastné spracovanie

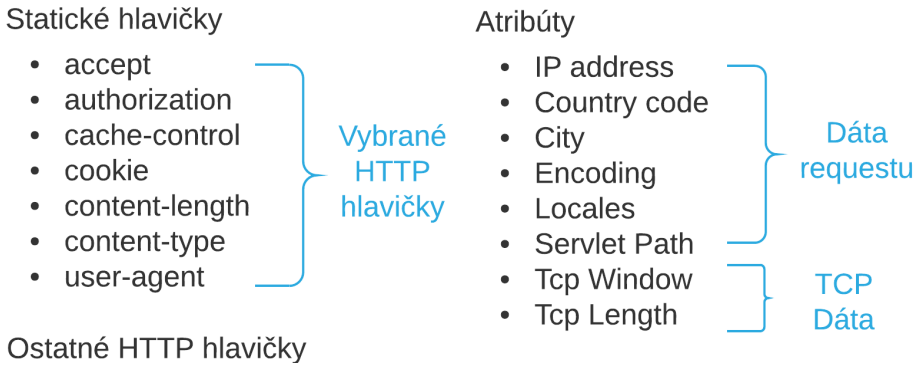
5.2.1 Statické dáta

Statická časť identifikátora obsahuje väčšie množstvo dát, preto je ukladaná a reprezentovaná v podobe reťazca. Skladá sa z nasledujúcich troch podčastí dát pre porovnávanie:

- Statické hlavičky *(Static headers)* - špecifické vopred vybrané a zoradené hodnoty HTTP hlavičiek, ako napríklad: *authorization*, *cookie*, *content-length*, alebo *user-agent*.
- Neznáme hlavičky *(Unknown headers)* - ostatné hlavičky daného HTTP requestu, ktoré neboli použité ako statické.

- Atribúty - konkrétne informácie z HTTP a TCP protokolu, ako napríklad *IP adresa*, *geo-lokácia*, alebo pre *veľkosť TCP okna*.

Súhrnne znázorňuje všetky použité informácie jednotlivých kategórií statických dát schéma 5.2.



Obr. 5.2: Detailný pohľad na štruktúru statických dát identifikátora, zdroj: vlastné spracovanie

Každá z týchto informácii má pri použití v porovnávacej funkcii svoj účel a konkrétnu váhu, ktorá určuje jej dôležitosť. Veľmi podstatný je napríklad obsah statických hlavičiek. Naopak váha hodnôt TCP okna je v porovnaní s IP adresou a geolokáciu menšia, môže však hrať rolu u veľkého množstva requestov. Podrobnejšie sa porovnávacej funkcií a spôsobu výpočtu podobnosti jednotlivých parametrov venuje podkapitola 5.3.2.

5.2.2 Relačné dáta

Relačné dáta neslúžia na porovnávanie a hľadanie podobných hodnôt. Ich účelom je ošetriť špeciálne prípady u informácií, ktoré si to vyžadujú. Takýchto informácií nie je mnoho, preto nie je nutné ich komprimovať a serializovať do podoby reťazca ako dáta statické. Ide teda o objekt obsahujúci nasledujúce položky:

- Timestamp - presný čas nadviazania spojenia aktuálnej relácie.
- Geolokáciu - krajina pôvodu určená na základe IP adresy klienta.

- Relačné hlavičky - HTTP hlavičky, ako napríklad *forwarded*, alebo *x-csrf-token*.

Jednotlivé položky sú vyhodnocované samostatne a ich výsledky sú aplikované na vzdialenosť vrátenú algoritmom pre nájdenie podobnosti. Napríklad pri veľmi malom časovom rozdiele porovnávaných requestov, je ich vzdialenosť dodatočne znížená. U geolokácie môže byť zmena parametra výpočtu dosiahnutá po vyhodnotení requestu klienta z krajiny s najčastejším pôvodom DoS útokov, akými sú napríklad: Čína, Kórea, alebo Rusko.

5.2.3 Variabilita štruktúry dát

Štruktúra statických a relačných dát odtlačku je v aplikácii kalibrovaná pre čo najpresnejšie určenie podobnosti requestov od jedného používateľa. Samozrejme je však možné ju v budúcnosti obmieňať, prípadne pridávať do nej nové informácie.

5.3 Algoritmus hľadania podobnosti

Ako bolo spomínané v predchádzajúcich odsekoch, hlavnou funkcionalitou procesu identifikácie je algoritmus na určenie podobnosti a hľadanie najbližšieho suseda aktuálne spracovávaného identifikátora.

Vstupom pre algoritmus je teda tento odtlačok spolu s množinou obsahujúcou odtlačky doteraz analyzovaných requestov.

Po inicializácii a načítaní vstupných parametrov je na každú dvojicu odtlačkov aplikovaná funkcia pre výpočet ich vzdialenosti, ktorá vráti hodnotu v rozmedzí od nula do jedna. Pre totožné odtlačky je vzdialenosť nulová, naopak u odtlačkov, ktoré sa líšia v mnohých parametroch sa hodnota blíži k jednej. Funkcia postupne porovnáva a zohľadňuje jednotlivé hodnoty statickej časti identifikátora podľa ich váhy a miery podobnosti. Detailným popisom fungovania tejto funkcie sa zaoberá podkapitola 5.3.2.

Po spracovaní výsledkov algoritmus nájde najbližší identifikátor, ktorý spolu s jeho vzdialenosťou vráti ako výstup na ďalšie spracovanie a prípadnú úpravu relačnými dátami.

Nasledujúci pseudo-kód teda formálne popisuje tento algoritmus a jeho prácu s výsledkami funkcie na výpočet vzdialeností:

```
algorithm get-nearest-neigbour is
    input: UFooEntity uFooEntity
           UFooEntity[] allStockItems
    output: Result<int distance, UFooEntity nearestNeigbour>

    minDistance ← MAX_INT;
    nearestneigbour ← null;

    for each stockUFooEntity in allStockItems do

        acctualDistance ← getDistance(uFooEntity.staticData,
                          stockUFooEntity.staticData);

        # Addition analysis - manual checks based on Relation Data
        # Possibly can SLIGHTLY afffect distance

        if acctualDistance < minDistance then
            minDistance ← acctualDistance;
            nearestneigbour ← stockUFooEntity;
        end if

    return Result<minDistance, nearestneigbour>
```

5.3.1 Použité postupy

Pred samotným popisom funkcie pre výpočet vzdialeností odtlačkov je nutné vysvetliť techniky a pojmy, s ktorými tento algoritmus pracuje. Konkrétne ide okrem porovnávania komprimovaných reťazcov pre jednoduché atribúty, najmä o určenie miery podobnosti množín pomocou výpočtu takzvaného *Jaccardovho* indexu.

Jaccardov index

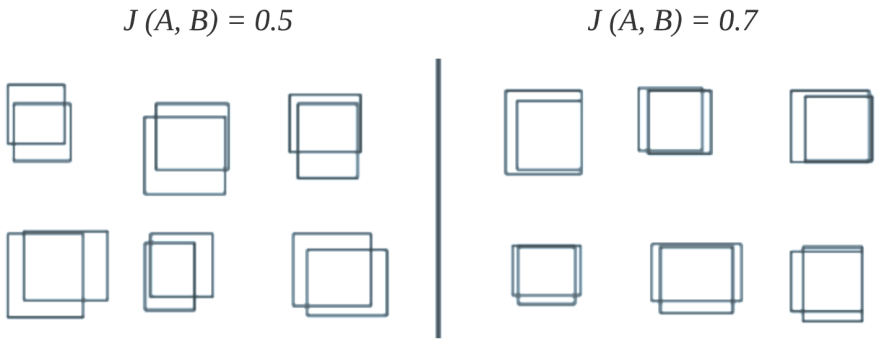
Výpočet Jaccardovho koeficientu, taktiež známy ako podiel prieniku a zjednotenia, je funkcia pôvodne popísaná Paulom Jaccardom, ktorá slúži na porovnanie podobnosti a rozdielov testovaných množín. Jaccardov koeficient teda meria mieru podobnosti medzi dvoma konečnými množinami. Matematicky je definovaný ako podiel veľkostí zjednotenia a prieniku týchto množín:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{5.1}$$

V prípade, že sú obe množiny prázdne *J(A,B) = 1*. Jaccardova vzdialenosť, ktorá naopak určuje rozdielnosť dvoch množín, je doplnkom k Jaccardovmu koeficientu. Počíta sa teda odrátaním jeho hodnoty od jednotky, prípadne ako podiel, ktorého čitateľom je rozdiel zjednotenia s prienikom a menovateľom je zjednotenie týchto množín:

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{5.2}$$

$$0 \leq J(A,B) \leq 1 \tag{5.3}$$



Obr. 5.3: Znázornenie prieniku a zjednotenia množín pre výpočet Jaccardovho koeficientu, zdroj: ??

Porovnávanie atribútov

Pri porovnávaní konkrétnych reťazcov u HTTP hlavičiek, alebo atribútov, ako napríklad *servletPath*, alebo *locales*, sú tieto hodnoty pred porovnaním komprimované. Konkrétne ide o odstránenie nadbytočných bielych znakov a symbolov, ktoré by mohli pri porovnaní spôsobovať nepresnosti. Rovnaké pravidlá sú použité aj pri porovnávaní položiek pri výpočte Jaccardovho indexu.

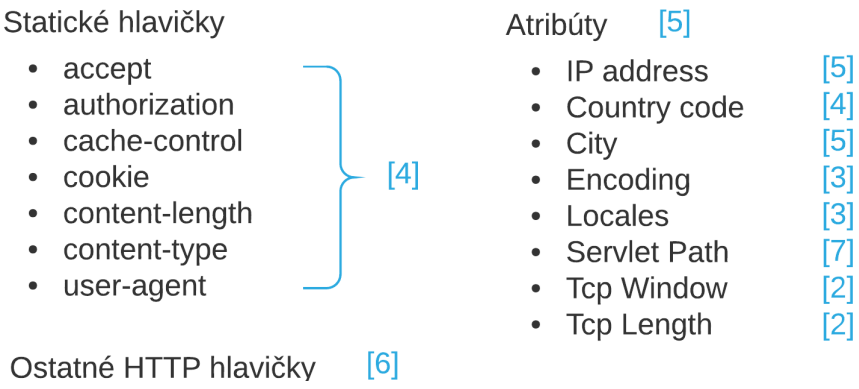
5.3.2 Funkcia na výpočet vzdialenosti

Vstupom pre funkciu na výpočet vzdialenosti sú teda vždy dve reprezentácie identifikátorov, konkrétne hodnoty v ich statickej časti.

Výsledná vzdialenosť je počítaná ako doplnok váženého priemeru podobností troch podčastí týchto dát nasledovne:

- Ako prvý je spočítaný Jaccardov koeficient pre statické hlavičky.
- Následne algoritmus použije Jaccardov index aj pre výpočet podobnosti zvyšných hlavičiek.
- Nakoniec je vypočítaná miera podobnosti jednotlivých atribútov.

Každá z týchto častí je pri následnom sčítaní vážená vlastnou konštantou. Zároveň má vlastnú váhu každý z atribútov tretej časti. Konkrétnu kalibráciu váh jednotlivých položiek znázorňuje schéma 5.4.



Obr. 5.4: Príklad kalibrácie váh jednotlivých parametrov statickej časti odtlačku pre výpočet vzdialenosti, zdroj: vlastné spracovanie

Kľúčovú rolu pri výpočte samozrejme zohráva práve spomínaná kalibrácia váh a parametrov tohto algoritmu. Nasledujúci pseudo-kód formálne popisuje funkciu výpočtu vzdialenosti a jej fungovanie:

```
algorithm get-distance is
    input: String actualData
           String stockEntityData
    output: double distance

    if actualData = stockEntityData
        return 0
    end if
```

```
# Compute weighted Jaccard for static headers
jaccardStaticHeaders
    ← jaccardIndex(actualData.staticHeaders,
                   stockEntityData.staticHeaders);
jaccardStaticHeaders
    ← jaccardStaticHeaders * SH_WEIGHT;

# Compute weighted Jaccard for unknown headers
jaccardUnknownHeaders
    ← jaccardIndex(actualData.unknownHeaders,
                   stockEntityData.unknownHeaders);
jaccardUnknownHeaders
    ← jaccardUnknownHeaders * UH_WEIGHT;

# Compare values of remaining attributes by similarValue
# function. Every of this attributes will have sub-weight
# which will be computed in attributesIndex.
isSameIp
    ← similarValue(actaulData.IP, stockEntityData.IP)
    * IP_SUB_WEIGHT;
isSameCountry
    ← similarValue(actaulData.country,
                   stockEntityData.country)
    * COUNTRY_SUB_WEIGHT;
...
isSameLength
    ← similarValue(actaulData.length,
                   stockEntityData.length)
    * LENGTH_SUB_WEIGHT;

attributesIndex
    ← ∑(isSameIp, isSameCountry, ..., isSameLength)
     / ∑(IP_SUB_WEIGHT, COUNTRY_SUB_WEIGHT,
         ..., LENGTH_SUB_WEIGHT);
attributesIndex
    ← attributesIndex * ATTR_WEIGHT

similarity
    ← ∑(jaccardStaticHeaders, jaccardUnknownHeaders,
        attributesIndex)
     / ∑(SH_WEIGHT, UH_WEIGHT, ATTR_WEIGHT)

return 1 - similarity;
```

Hodnoty podobností jednotlivých častí sú teda násobené pridelenými váhami a sčítané. Podielom súčtov týchto hodnôt a ich možných váh je následne vypočítaný koeficient podobnosti statických dát vstupných identifikátorov. Výstupom funkcie je vzdialenosť, ktorá je doplnkom k tomuto koeficientu.

5.4 Implementácia

Súčasťou tejto práce je okrem návrhu algoritmu aj jeho implementácia v jazyku Java. Ide o presnú implementáciu funkcionality popísanej v predchádzajúcich podkapitolách. Knižnica samotná je čo najuniverzálnejšia a nezávislá na konkrétnom *frameworku*, či technológii. Po vložení jej závislosti do hostiteľského projektu sama detekuje prístupové metódy a zachytáva jednotlivé HTTP requesty pre ďalšie spracovanie. Zároveň pri spustení, ak je to možné, začne monitorovať sieť pre doplnenie informácií z TCP protokolu.

5.4.1 Použité technológie

Pre korektnú implementáciu funkcionality celého algoritmu bolo nutné použitie mnohých špecifických technológií, od knižnice pre monitoring TCP paketov, až po geolokáciu IP adries. Nasledujúce odseky popisujú najdôležitejšie z nich.

Aspektovo-orientované programovanie

Aspektovo-orientované programovanie (ďalej len AOP) je paradigma určená k zvýšeniu modularity kódu bez narúšania prirodzeného behu aplikácie. Ide o pridanie novej funkcionality k už existujúcemu kódu, bez nutnosti modifikácie pôvodného zdroja. Namiesto toho je osobitne definovaný takzvaný prístupový bod, ktorý pridá požadovanú funkcionalitu na vopred určené miesto.

V implementačnej časti je AOP, konkrétne knižnica *AspectJ*, použitá pre injektovanie jednotlivých prístupových metód hostiteľskej aplikácie a následné odchytávanie informácií z ich HTTP requestov. Je tak možné vytvoriť prístupové body pre ľubovoľný z používaných *frameworkov*, či prístupov. Momentálne, na testovacie účely, implementácia obsahuje prístupový bod pre metódy so *Springovou* anotáciou

@RestController. Nie je však problém rozhranie rozšíriť o akýkoľvek existujúci, či vlastný prístupový bod. Ďalšou z výhod tohto prístupu je asynchrónnosť celého výpočtu. Po zachytení informácií z HTTP requestu je proces ich spracovania vykonávaný v novom vlákne. Logika hostiteľskej aplikácie tak nie je ničím zdržovaná a môže pokračovať štandardným spôsobom.

Sieťová analýza

Pri štarte hostiteľskej aplikácie, je štandardne jedným z modulov v samostatnom vlákne spustený monitoring, analýza a parsovanie TCP paketov. Pre prístup k týmto informáciám je využívaná knižnica *libpcap* a program *tcpdump*. Samozrejme tieto možnosti je nutné vopred povoliť na hostiteľskom serveri, preto existuje možnosť aplikáciu používať aj bez monitorovania sieťovej aktivity. Výpočty vzdialeností identifikátorov, ktoré tieto informácie obsahujú sú však presnejšie a produkujú menej falošne pozitívnych detekcií.

IP Geolokácia

Ako bolo popísané v sekcii 5.2, jedným z atribútov, ktoré sú posudzované pri vyhodnotení podobnosti identifikátorov je geolokácia klienta. Konkrétne ide o krajinu, prípadne mesto, určené na základe jeho IP adresy. Zdrojom týchto informácií je databáza poskytnutá službou *MaxMind GeoIP*.

Okrem tejto databázy systém používa aj statický zoznam menej bezpečných zón. Ide o konkrétne krajiny, ktoré sú najčastejším pôvodcom DoS a DDoS útokov. Ich detekcia môže mať za následok napríklad zníženie prahu pre upozornenie hostiteľskej aplikácie pri prekročení väčšieho množstva podobných odtlačkov.

5.4.2 Funkcionalita a štruktúra kódu

Celá akcia spracovania requestu začína v triede *Injector*. Táto trieda obsahuje prístupové body AOP pre jednotlivé metódy hostiteľskej aplikácie. Zároveň slúži na správu závislostí a inicializáciu *singleton* vlákna triedy *PacketStream*. *PacketStream* a ostatné pomocné triedy balíka *endpoint.collector.tcp* zabezpečujú, ak je to možné, parsovanie a

správu prijatých TCP paketov. Tieto sú následne dostupné ako mapa posledných *n* zachytených paketov v podobe množiny objektov triedy *TCPFootprint* a ich IP adries.

Z Injectora dáta pokračujú vždy v samostatnom vlákne na spracovanie triedou *RequestHandler* a ostatnými triedami balíka *endpoint.collector.http*. Ich účelom je extrahovať potrebné HTTP informácie, ktoré sú následne reprezentované triedou *HTTPFootprint*. [2]

Ďalej už s reprezentáciami *HTTPFootprint* a *TCPFootprint* pracuje trieda *UFooProcessor*. Táto trieda obsluhuje hlavnú logiku analýzy podobnosti aktuálne spracovávaného requestu s ostatnými identifikátormi v databáze. *UFooProcessor* pomocou metód triedy *Serializer* spracuje informácie do konečnej podoby objektu *UFooEntity*, ktorý je reprezentáciou samotného unikátneho identifikátora. *UFooEntity* obsahuje všetky extrahované statické a relačné dáta.

Najdôležitejšou časťou celého procesu je však analýza podobnosti vykonávaná triedou *FootprintSimilairtyService*, ktorá implementuje algoritmus hľadania najbližšieho suseda formálne popísaný v sekcii 5.3. Jej metódy *getNearestNeighbour*, *computeDistance* a *jaccardIndex* teda spočítajú konečný výsledok v podobe vzdialenosti a najbližšieho podobného identifikátora z databázy.

Na základe tohto výsledku sa následne *UFooProcessor* rozhodne, či ide o aktivitu neznámeho užívateľa a vytvorí pre aktuálny odtlačok nový záznam v databáze, alebo ide o opakujúcu sa reláciu, čím zvýši jeho frekvenciu, prípadne upozorní hostiteľskú aplikáciu na možné prekročenie jej limitu.

Celý priebeh spracovania a analýzy podobnosti reqestu je zároveň znázornený sekvenčným diagramom v prílohe A.

Spracovanie a analýza dát z jednotlivých requestov prebieha asynchrónne, aplikácia teda nemusí čakať na dokončenie a jej výsledok. V prípade hrozby je samozrejme upozornená.

Pre účely testovania a kalibrácie parametrov algoritmu obsahuje praktická časť práce okrem knižnice samotnej aj príklad hostiteľskej aplikácie, v ktorej je knižnica použitá. Tento server takisto poskytuje možnosť zobrazenia jednoduchých štatistík z analýzy podobnosti.

2. Štruktúra tried *HTTPFootprint* a *TCPootprint* je dostupná v prílohe B.

6 Test s reálnymi dátami

Algoritmus a jeho implementácia popísaná vyššie boli testované a kalibrované na simulácií reálneho D-DoS útoku. [1] Nasledujúce odseky popisujú jednak priebeh tohto útoku, ako aj priebeh a výsledky jeho simulácie.

6.1 Parametre útoku

Útok, na ktorom je následná simulácia založená, bol pomerne rozsiahly a dobre organizovaný. Celkovo trval 56 minút a zúčastnili sa ho klienti zo 136 rôznych IP adries. V priebehu týchto 56 minút server prijal dohromady 1 345 446 requestov, pričom všetky smerovali na rovnakú URL a jednu konkrétnu akciu. Jeho ďalším znakom je pomerne malá diverzita obsahu HTTP hlavičiek. Príklady týchto hlavičiek niektorých z requestov zobrazuje nasledujúca schéma:

```
[
    "Connection": "keep-alive",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; )...",
    "Content-Type": "application/json; charset=UTF-8",
    "Host": "sample-domain.com",
    "Accept": "application/json, text/javascript, q=0.01",
    "Origin": "http://sample-domain.com",
    "Cookie": "__gvf=DN-24; _be=DV1.2; SampleTool...",
    "Accept-Encoding": "gzip, deflate",
    "Referer": "http://sample-domain.com/pages/homepage...",
    "X-Requested-With": "XMLHttpRequest",
    "Accept-Language": "en-US,en;q=0.8"
]
```

Na tento útok bolo teda použité veľké množstvo zariadení, ktoré postupne odosielali požadované dáta. Všetky requesty však obsahovali podobnú skupinu HTTP hlavičiek, s rôznymi obmenami, čo pomohlo k jeho lepšej identifikácií.

---

1. Všetky citlivé dáta, ako napríklad reálne IP adresy boli agregované a anonymizované, aby nemohlo dôjsť k ich následnému zneužitiu.

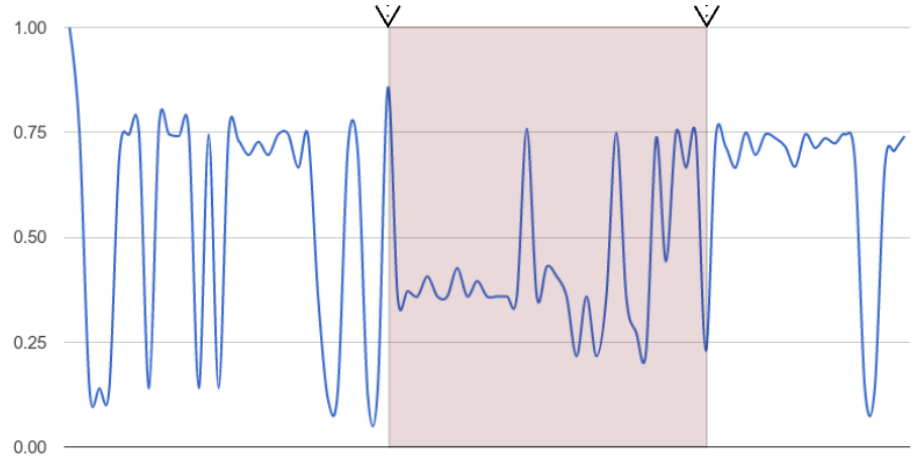
6.2 Parametre simulácie

Simulácia vyššie popísaného útoku sa ho snažila čo najvernejšie napodobniť. Samozrejme však nebolo možné uskutočniť ho v plnom rozsahu. Na testovací server bola preto vytvorená štandardná záťaž v podobe 25 000 requestov. Tieto requesty sa snažili svojim obsahom čo najvernejšie napodobniť reálnych používateľov a ich aktivitu. Paralelne bol na server v priebehu niekoľkých desiatok minút odosielaný obsah simulujúci spomínaný D-DoS útok v rozsahu 13 454 requestov z 13 rôznych IP adries. Ostatné parametre pôvodného útoku a obsah HTTP hlavičiek zostali zachované.

Účelom tejto simulácie bolo overiť správnosť fungovania algoritmu, najmä jeho schopnosť identifikácie a rozlíšenia aktivity používateľov D-DoS útoku od štandardných relácií.

6.3 Interpretácia výsledkov

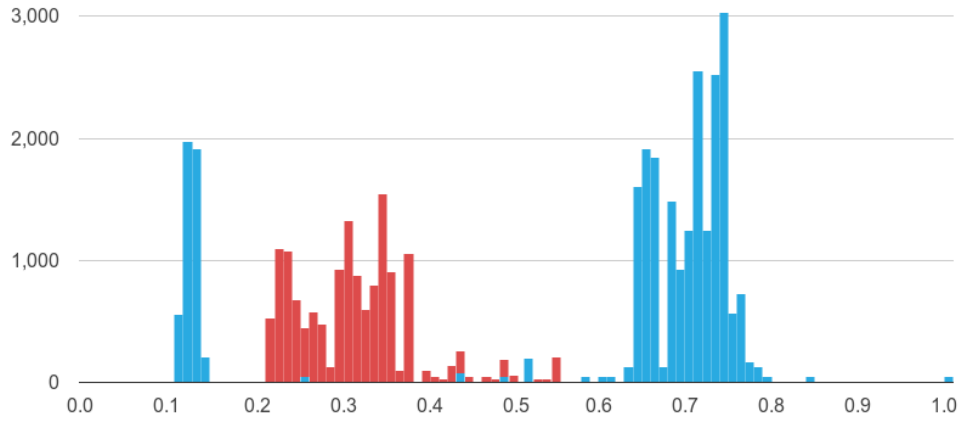
Nasledujúce grafy zobrazujú výsledky simulovaného útoku v porovnaní z aktivitou bežných užívateľov. Všetky výsledky, ako aj ich grafické znázornenie boli produkované aplikáciou, používajúcou implementáciu knižnice pre účely testovania algoritmu, ktorá je súčasťou praktickej časti práce.



Obr. 6.1: Vzdialenosť reqestov simulácie, spolu z ohraničením začiatku a konca D-DoS útoku v čase, zdroj: implementačná časť práce

Graf 6.1 znázorňuje výsledky výpočtu vzdialeností jednotlivých requestov[2] v čase. Vyznačený úsek poukazuje na začiatok a koniec simulovaného D-DoS útoku.

Histogram 6.2 ďalej zobrazuje presné rozloženie počtu vzdialeností celej simulácie. Zároveň farebne rozlišuje vzdialenosti vypočítané pre requesty D-DoS útoku od vzdialeností reálnych užívateľov.

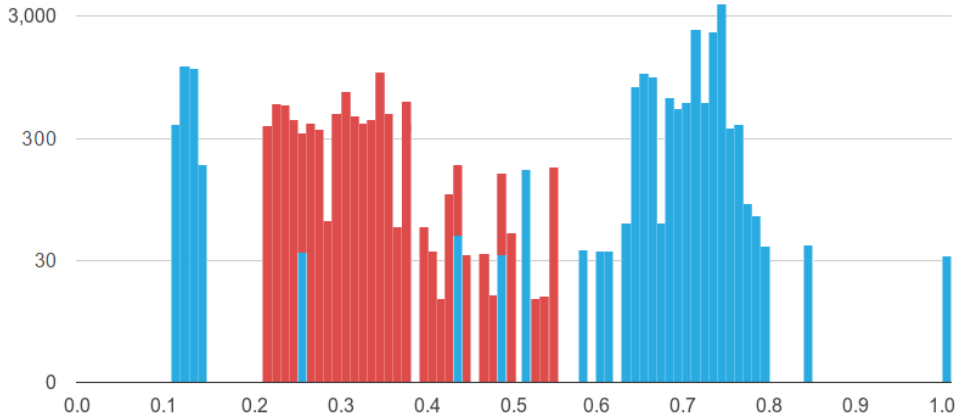


Obr. 6.2: Histogram počtu vzdialeností requestov simulácie. Modrá farba reprezentuje vzdialenosti reálnych užívateľov, červená aktivitu D-DoS útoku, zdroj: implementačná časť práce

6.4 Zhrnutie

Z výsledkov simulácie, najmä z výsledkov zobrazených v histograme 6.2 je možné vidieť, že aktivitu requestov D-DoS útoku je možné odlíšiť od aktivity reálnych používateľov meraním vzdialenosti jednotlivých requestov. Zatiaľ čo vzdialenosti simulácie aktivity reálnych užívateľov sa pohybujú vo väčšine prípadov v intervaloch <*0.1,0.15*> a <*0.6,0.8*>, väčšina vzdialeností simulácie aktivity D-DoS útoku sa pohybuje v intervale <*0.2,0.4*>. Zároveň existuje interval <*0.4,0.6*> v ktorom sa v

2. Pre účely lepšej čitateľnosti je počet vzdialeností, z ktorých bol graf rendrovaný redukovaný. Výsledkom je teda aproximácia jednotlivých vzdialeností simulácie v čase



Obr. 6.3: Histogram počtu vzdialeností requestov simulácie s použitím logaritmickej osi y, zdroj: implementačná časť práce

menšom množstve oba typy prelínajú. Pre lepšie znázornenie stredného intervalu používa histogram 6.3 logaritmickú os y. Podľa týchto údajov je teda možné kalibrovať aj prah pre upozornenie hostiteľskej aplikácie na možnú hrozbu.

7 Záver

Výsledky testov s reálnymi dátami dokázali funkčnosť algoritmu a jeho schopnosť identifikácie užívateľov aj počas trvania simulácie prebiehajúceho DoS útoku.

Tieto výsledky sú však do istej miery aj pomerne prekvapivé. Pôvodná myšlienka odhalenia requestov DoS útoku totiž počítala s veľkým počtom nízkych výsledných vzdialeností u útočníka a vysokých vzdialeností u bežnej záťaže.

Reálne však, ako je možné vidieť na histograme 6.2, simulácia requestov reálnych užívateľov vykazovala maximá u okrajových hodnôt vzdialeností a simulácia requestov DoS útoku u stredných hodnôt vzdialeností.

Dôvodom tohto rozloženia je zrejme práve spôsob, akým útočník na rozdiel od bežných užívateľov so serverom komunikoval. Bežní užívatelia totiž zasielali navzájom pomerne rozdielne requesty, pričom requesty jedného užívateľa sa v jeho relácii na seba podobali. To spôsobilo ich rozloženie do okrajových hodnôt vzdialeností. Útočník ale celý čas používal jeden druh requestu, ktorý však neustále obmieňal. Preto je rozloženie jeho vzdialeností v stredných hodnotách škály.

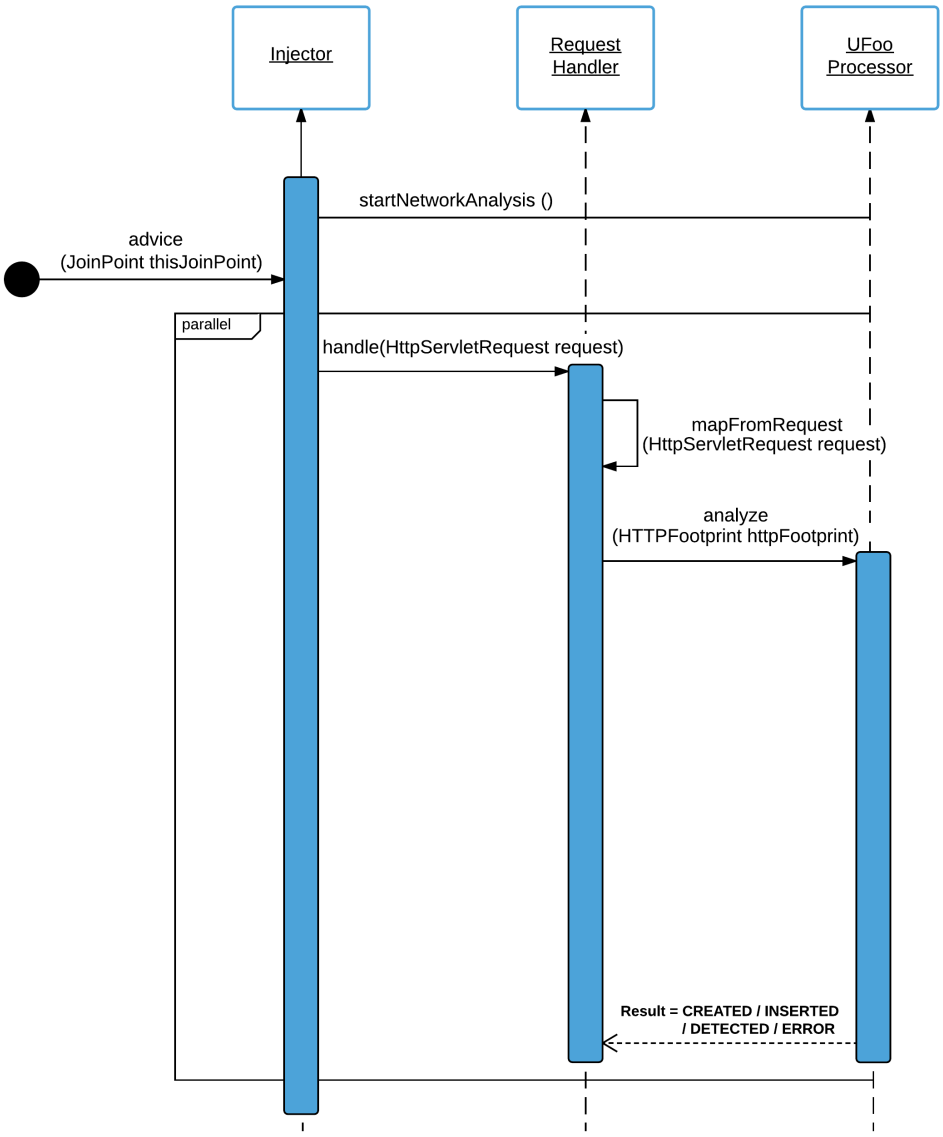
Na základe týchto výsledkov je teda možné nastaviť prahy pre určenie útoku práve v intervale stredných vzdialeností útočníka, podľa výsledkov simulácie.

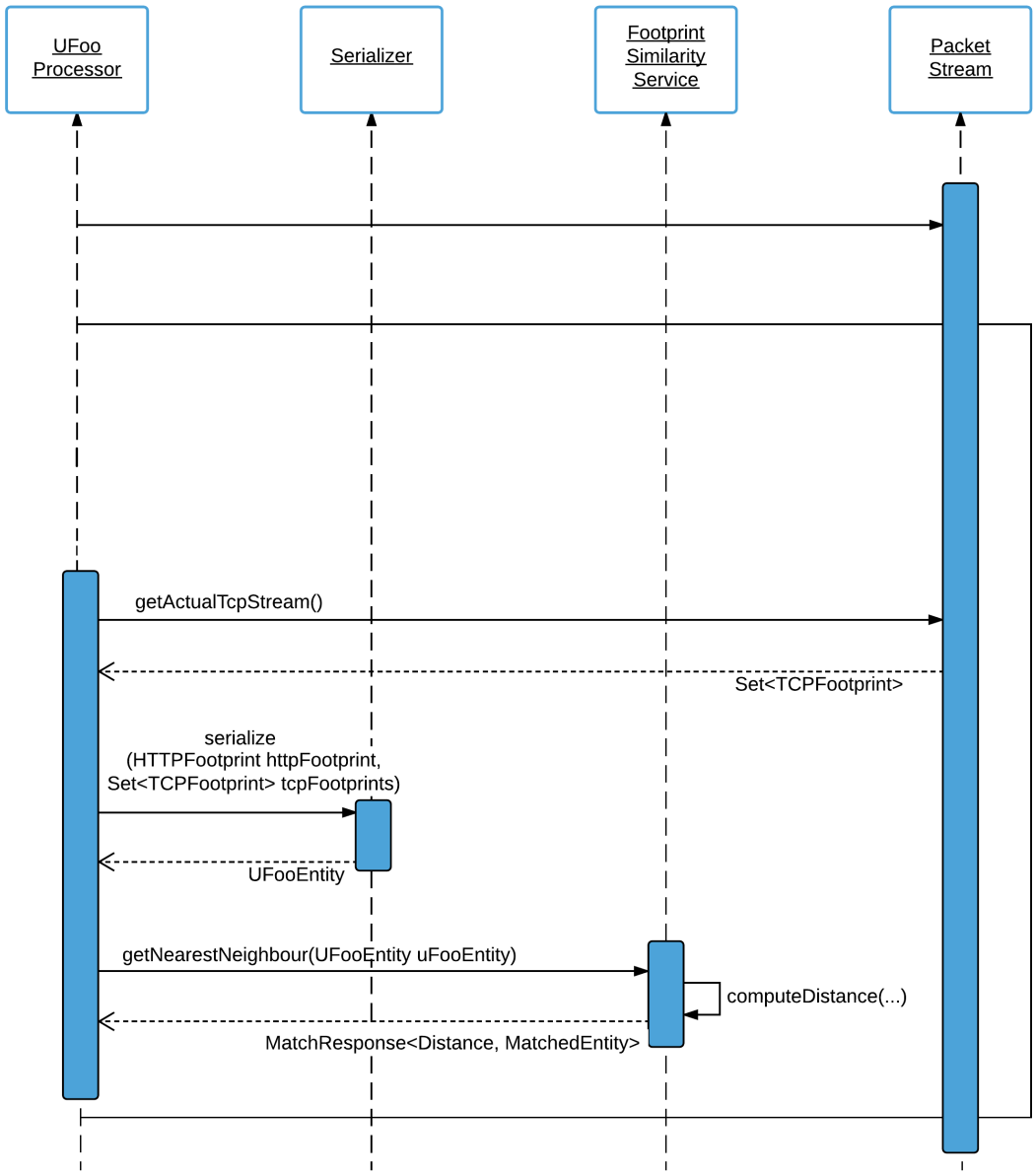
Táto problematika ma veľmi zaujala, preto prácu plánujem ďalej rozvíjať, a dopĺňať o ďalšiu zaujímavú funkcionalitu od možnosti zdieľania databázy odtlačkov, až po zlepšenie sledovania a použitia TCP informácií.

Bibliografia

[Olu14] Haroon Shakirat Oluwatosin. „Client-Server Model“. In: *IOSR Journal of Computer Engineering* 16 (feb. 2014), s. 67–71. URL: http://www.iosrjournals.org/iosr-jce/papers/Vol16-issue1/Version-9/J016195771.pdf.

A Priebeh spracovania reqestu




UFoo Processor
Serializer
Footprint Similarity Service
Packet Stream
getActualTcpStream()
Set<TCPFootprint>
serialize
(HTTPFootprint httpFootprint,
Set<TCPFootprint> tcpFootprints)
UFooEntity
getNearestNeighbour(UFooEntity uFooEntity)
computeDistance(...)
MatchResponse<Distance, MatchedEntity>

B Triedy *HTTPFootprint* a *TCPootprint*