Masarykova univerzita
Fakulta informatiky

Rozpoznanie užívateľa na základe informácií o HTTP komunikácií

Diplomová práca

Matej Majdiš

Brno, jar 2017

Namiesto tejto stránky vložte kópiu oficiálneho podpísaného zadania práce a prehlásenie autora školského diela.

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. title

Poďakovanie

TODO

Zhrnutie

TODO

Kľúčové slová

keyword1, keyword2, …

Obsah

Zoznam tabuliek

Zoznam obrázkov

1 Úvod

Problematika jednoznačnej identifikácie používateľa je dnes veľmi dôležitou a riešenou témou. Jedným z hlavných dôvodov je fakt, že väčšina dnešných existujúcich, prípadne novo vznikajúcich systémov a aplikácií je nejakým spôsobom zapojená do Internetu. Zároveň zaznamenávame nárast aplikácií, ktoré poskytujú užívateľom webové rozhranie a ústup takzvaných desktopových aplikácií.

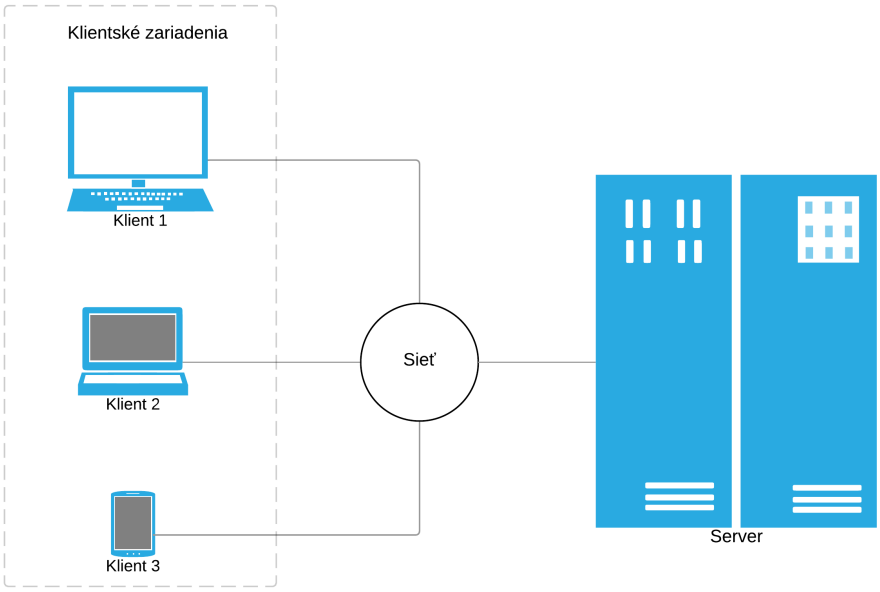
Obr. 1.1: Vizualizácia pomeru počtu užívateľov webových a desktopových aplikácií v čase, zdroj: vlastné spracovanie

Z tohto vyplýva potreba rozoznania a identifikácie používateľov, ktorý s danou aplikáciou interagujú. Existuje niekoľko rôznych prístupov k identifikácií, od mapovania IP adries sieťovej vrstvy až po aplikačnú správu užívateľských účtov. Podrobne sa nimi zaoberá kapitola 4. Cieľom tejto práce je vytvoriť unikátny identifikátor na základe informácií dostupných z *HTTP* protokolu. Pred zostavením samotného algoritmu je preto dôležité popísať niektoré kľúčové oblasti a postupy.

Nasledujúce kapitoly sa preto budú stručne zaoberať fungovaním aplikácií typu klient-server, modelom sieťových vrstiev či útokmi typu *Denial of Service*. Ďalej v práci popíšem spomínané existujúce prístupy a vlastný návrh algoritmu identifikácie užívateľa.

1.1 Aplikácie typu Klient-Server

S pokračujúcim vývojom nových technológií sa Web stáva stále väčšou súčasťou našich životov. Web taktiež už nie je limitovaný prehliadaním na počítačoch. Musí sa prispôsobovať rôznym novým technológiám, ako sú napríklad mobilné, či iné zriadenia. Najčastejšie používaným modelom komunikácie pre architektúru webových aplikácií je tzv. Klient-Server model. Základnou myšlienkou tohto modelu je zaslanie požiadavku (*requestu*) klientom na server, ktorý vystupuje ako poskytovateľ služby.



Obr. 1.2: Schéma znázorňuje základnú architektúru modelu Klient-Server, zdroj: vlastné spracovanie

1.1.1 Klient-Server model

Pretože Klient-Server model je používaný rôznymi typmi aplikácií bolo nutné použiť štandardizované protokoly, na základe ktorý ch bude možné komunikovať. Základné používané protokoly sú: *FTP*

(File Transfer Protocol), *Simple Mail Transfer Protocol (SMTP)* a *Hypertext Transfer Protocol (HTTP)*. Bližšie sieťové vrstvy a jednotlivé protokoly popisuje kapitola 2.

1.1.2 Architektúra

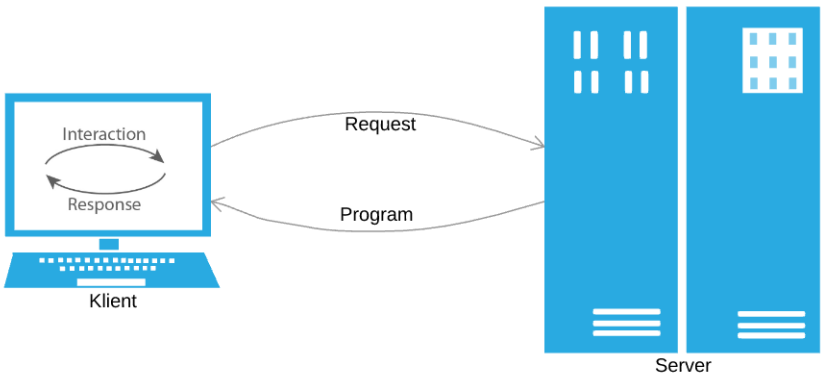
Architektúra modelu Klient-Server sa vo všeobecnosti typicky skladá z troch častí:

- Aplikačný server
- Databázový server
- Zariadenie klienta

Zároveň Existujú dva základné typy architektúr:

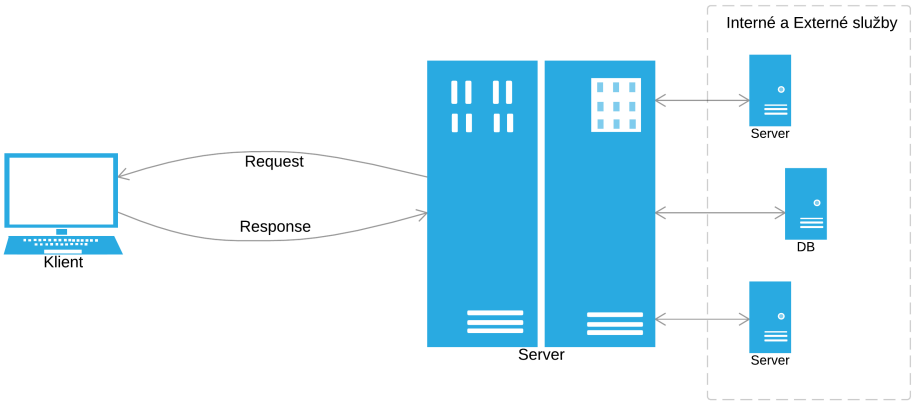
- 2-stupňová *(2-tier)*
- 3-stupňová *(3-tier)*

2-tier architektúra zahrna 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 zároveň *business* logiku aj zobrazovanie aplikácie. Inak tento typ architektúry nazývame aj tučný klient(*thick client*).



Obr. 1.3: Grafické znázornenie a popis priebehu komunikácie 2-tier architektúry, zdroj: vlastné spracovanie

3-tier architektúra, ktorou sa budem zaoberať v tejto práci sa od *2-tier* líši najmätým, že okrem zariadenia klienta a databázového servera zahŕňa aj aplikačný server. Tento je následne používaný na obsluhu *business* logiky aplikácie a komunikáciu s databázou, pričom zariadenie klienta slúž len na zobrazovanie. Iný názov pre takýto typ architektúry je tenký klient (*thincient*).



Obr. 1.4: Grafické znázornenie a popis priebehu komunikácie 3-tier architektúry, zdroj: vlastné spracovanie

Nasledujúce kapitoly tejto práce sa budú zaoberať jedným z najdôležitejších problémov Webových aplikácií, ktorým je identifikácia užívateľa. Najskôr kapitola 2 popisuje jednotlivé sieťové vrstvy a protokoly, ktorých informácie je možné použiť ka následnej identifikácií. Ďalšou časťou je zhrnutie existujúcich prístupov k rozoznaniu užívateľov v kapitole 4 a popis útokov typu DOS v kapitole 3. Najdôležitejšou časťou je však samozrejme kapitola 5, ktorá popisuje návrh samotného algoritmu identifikátoru.

2 Sieťové vrstvy

Nasledujúca kapitola stručne popisuje jednotlivé sieťové vrstvy TCP/IP modelu a ich protokoly. Zameriava sa na štruktúry a informácie, ktorých pochopenie je dôležité pre ďalšie zložky teoretickej a praktickej časti práce. Ide najmä o konkrétne informácie jednotlivých protokolov skutočne 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)* modele 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, pri čom 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 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 doménovým mien, a iné...

Pri tvorbe identifikátoru 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 pre spoluprácu a prenos informácií medzi jednotlivými systémami určenými na prácu s hyper-médiami. Tento protokol je bez-stavový 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. Charakteristickým znakom tohto protokolu je aj možnosť voľby reprezentácie dát, čo dáva systémom veľkú nezávislosť na prenášanom formáte.

HTTP funguje na princípe požiadavku a odpovede *(request - respone)* založenom na fungovaní samotného klient-server modelu. Klientom môže byť napríklad webový prehliadač, serverom zasa aplikácia spustená na počítači hostiteľského uzla. V tomto prípade teda webový prehliadač zašle správu vo forme HTTP požiadavku na server. V ideálnom prípade server tento požiadavok *(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 požiadavku a prípadne samotné 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 portmi sú 80, 443 alebo 8080). HTTP server počúva na danom porte a čaká na správu požiadavku klienta. Po prijatí správy ju už server štandardne spracuje a vráti odpoveď.

Klient a server komunikujú prostredníctvom zasielanie plain-textových správ. Požiadavok pozostáva z nasledujúcich informácií:

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

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

- Status spracovania požiadavku 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 bol požiadavok odoslaný.

Obr. 2.2: Príklad štruktúry HTTP požiadavku a odpovede

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, naproti 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 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.

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

Niektoré informácie týchto hlavičiek sú využívané jednak u existujúcich nástrojov na identifikáciu užívateľov, ktorým sa podrobne venuje kapitola 4. Ich vyžitie v algoritme identifikátoru 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 nasledujúci diagram.

Obr. 2.4: *Three-way handshake* v TCP

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 stratených 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 znateľne menší ako paket TCP, čo taktiež napomáha rýchlosti jeho prenosu a spracovania. Keďže však UDP poskytuje len minimálne množstvo informácií jeho využitie pri identifikácií 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 metód, protokolov a špecifikácií, ktorá v sieťovom modely 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 dôležite 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. Sieťová vrstva však neobsahuje 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 uzlu 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

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

V skutočnosti existujú dve verzie IP protokolu: IP verzie 4 a IP verzie 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.

2.4 Vrstva sieťového rozhrania

Vrstva sieťového rozhrania funguje 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.

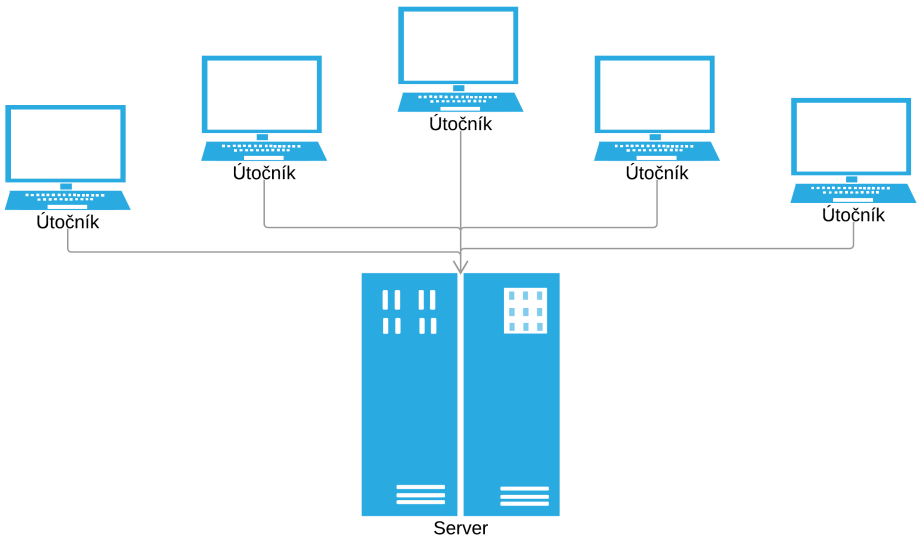
Niektorými z jej dôležitých protokolov sú: ARP *(Address Resolution Protocol)*, RARP *(Reverse Address Resolution Protocol)*, NDP *(Neighbor Discovery Protocol)*, OSPF *(Open Shortest Path First)* a iné.

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

3 Útoky typu *Denial of Service*

Jedným z hlavných dôvodov identifikácie užívateľov je prevencia proti útokom. Medzi najznámejšie z útokov patrí tzv. *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 internetového pripojenia hostiteľa.



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

U distribuovaného *DoS* útoku môže útočník použiť k útoku na server počítače klientov. Nad týmito zariadeniami je možné prevziať kontrolu využitím bezpečnostných chýb alebo nedostatkov. Takto je následne možné donútiť počítač posielať obrovské množstvo dát na webové servery, prípadne odosielanie nevyžiadanej pošty na konkrétne e-mailové adresy. Útok sa nazýva "distribuovaný", pretože útočník používa viac zariadení na začatie útoku *denial-of-service*.

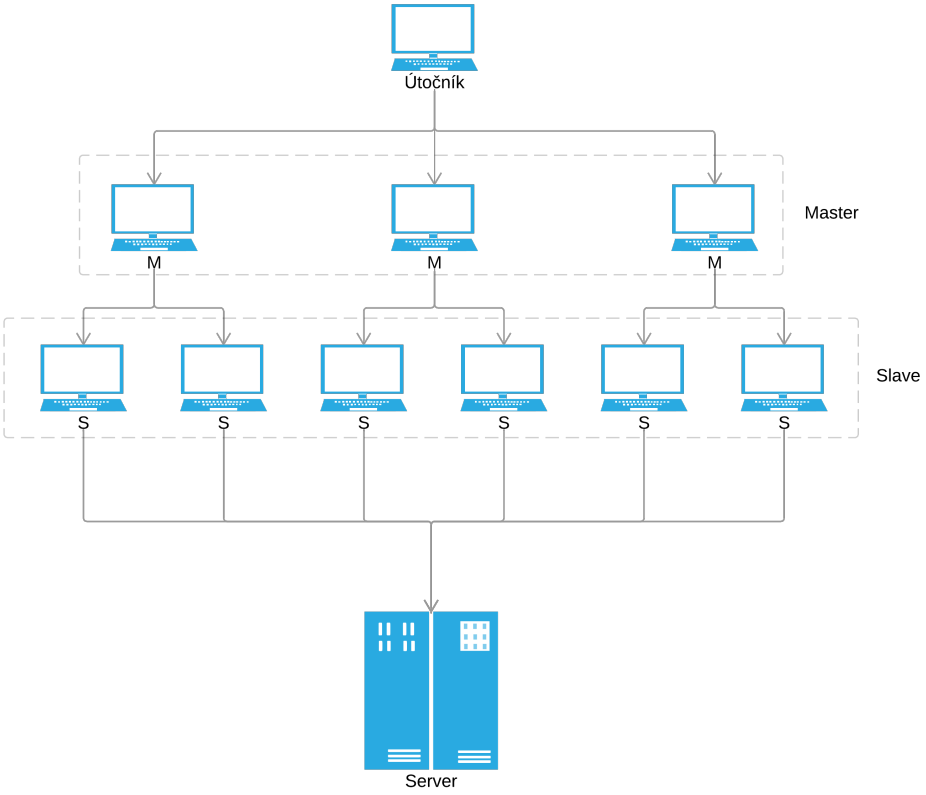
DoS útok je podobný veľkej skupine ľudí, ktorá sa zhromažďuje pri vstupe do obchodu a bráni vo vstupe skutočným zákazníkom, ktorých záujmom sú reálne služby. Útočníci vykonávajúci tieto útoky sa často zameriavajú na webové služby a servery, ktoré sú poskytované vysoko profitujúcimi inštitúciami, ako sú napríklad banky, prípadne platobné brány.

3.1 Základne typy a techniky

Nasledujúce odseky popisujú najpoužívanejšie typy a techniky vykonávania *DoS* útokov a identifikujú prostriedky, ktorými je im možné zabrániť.

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 *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 (nové) už nie je možné nadviazať. Hlavné výhody útočníka pri využití Distribuovaného *DoS* útoku spočívajú v skutočnostiach, že viaceré zariadenia dokážu generovať väčšiu záťaž ako jedno, pričom použitie množstva systémov zabezpečuje omnoho ťažšiu detekciu útočníka a správanie sa každého z týchto zariadení je menej pozorovateľné čo sťažuje obranu voči útočníkovi. 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.



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

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 v zozname pripojení ihneď po potvrdení *TCP SYN requestu*. Útočník otvorí viaceré spojenia, ktoré nikdy neuzavrie, čím zahlcuje server.

Pri *CGI* útoku je 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.

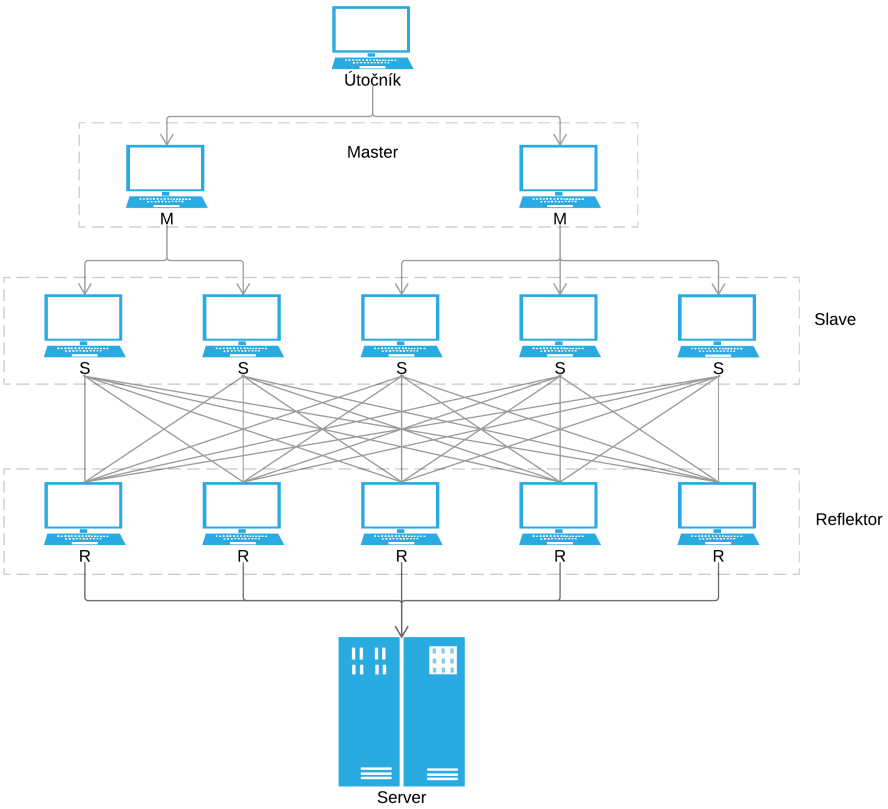
3.1.3 Brute-force útoky

TODO

Source ref: https://www.eecis.udel.edu/ sunshine/publications/ccr.pdf

3.1.4 Reflexia a zosilnenie

TODO



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

3.1.5 HTTP POST DoS útoky

TODO

3.2 DoS nástroje a DoS as a Service

Typickou metódou prenosu mechanizmov *DDoS* útokov je malvér. Jedným z príkladov bol takzvaný *MyDoom*. Ide o *DoS* mechanizmus, ktorý sa spúšťal vo predom 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 pripojenia. 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 zneužívané 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 skupinu *Anonymous*.

—

TODO DoSaaS

3.3 DoS - Zhrnutie

TODO

4 Existujúce prístupy k identifikácií

Nasledujúca kapitola sa venuje popisu, porovnaniu a hodnoteniu existujúcich prístupov, ktoré sú momentálne využívané na účely identifikácie používateľa. 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 zriadenia je IP protokol sieťovej vrstvy. Táto technika je veľmi rozšírená napríklad pre účely blokovania prístupu na server u systémových *firewallov* a podobne. TODO - rozšíriť

4.1.1 Internet Protocol (IP)

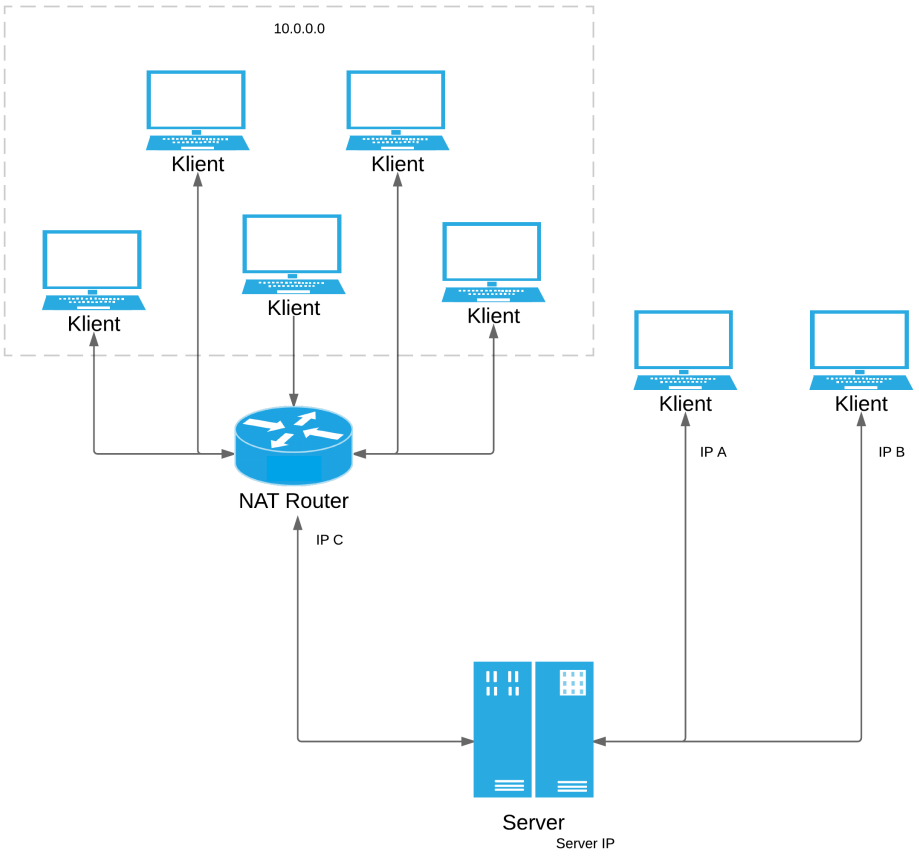
Ako popisuje kapitola 2, Internet Protocol slúži na prenos paketov medzi jednotlivými sieťovými uzlami - zariadeniami, ktoré sú identifikované IP adresami. V ideálnom by bolo každé zariadenie v sieti identifikované jednou statickou IP adresou. Bolo by teda možné užívateľa rozpoznať len na základe množiny 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 uzlu. V pokročilých implementáciách však dnes

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

NAT predstavuje veľmi populárnu možnosť takzvaného IP maskovania. Maskoavanie 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

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 prekrytiu jednotlivých užívateľov v rámci podsiete.

4.2 Monitoring TCP

Ako popisuje predchádzajúca kapitola, TCP *(Transmission Control Protocol)* je jedným z protokolov transportnej vrstvy sieťovej hierarchie. Jeho 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. Podrobný popis štruktúry TCP paketu znázorňuje ==obrazok kapitola 2?==

//TODO dokončiť

4.2.1 Vyhody a nevyhody

TODO

4.3 Aplikačné identifikátory

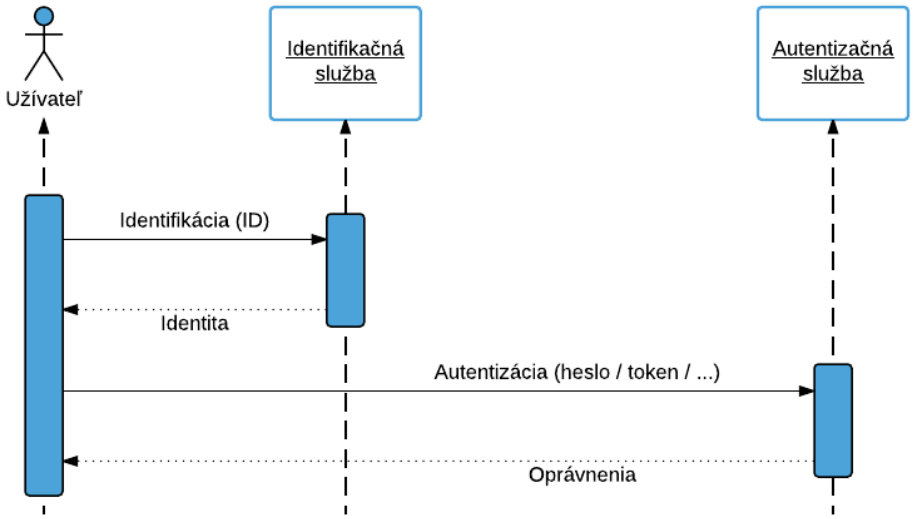
Na aplikačnej úrovni je identifikácia užívateľa oveľa komplexnejším a do istej miery aj abstraktnejším procesom. Z pohľadu systému sú jeho používateľmi typicky ľudia, prípadne procesy iných služieb, ktoré existujú mimo neho. Identifikátorom (ďalej len ID) užívateľa je teda projekcia aktuálneho jednotlivca, prípadne procesu do počítačového systému. Systém používa typicky abstraktný objekt - účet užívateľa, ktorý obsahuje množinu atribútov pre každého jednotlivca, prípadne 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 úspešnej identifikácií 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 abstrakcií 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 tvorbou unikátneho identifikátoru, bude 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 identifikátoru

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 práve 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.

5.1 Popis Algoritmu

Vo všeobecnosti pozostáva celý algoritmus z piatich hlavných krokov:

1. Zachytenie requestu pred samotným spracovaním hostiteľskou aplikáciou
2. Zhromaždenie HTTP a TCP informácií daného spojenia
    - HTTP dáta sú pasované zo zachyteného requestu
    - TCP dáta poskytuje samostatný sieťový monitor
3. Spracovanie informácií a vytvorenie unikátneho odtlačku requestu
4. Nájdenie najbližšieho podobného identifikátoru v databázy
5. Spracovanie výsledku v podobe vzdialenosti odtlačkov a následná reakcia
    - Ak ide o novú reláciu, pre identifikátor je vytvorený nový záznam
    - V prípade, že miera podobnosti prekročila stanovený prah, nevytvára sa nový záznam. Frekvencia daného odtlačku v databázy je však zvýšená.
    - Ak je frekvencia výskytu konkrétneho odtlačku veľmi vysoká existuje možnosť upozornenia hostiteľskej aplikácie.

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

Ako je 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 databázi. Algoritmus na hľadanie najbližšieho suseda k nemu následne pomocou porovnávacej funkcie nájde najviac podobný 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 v DB. Jednou z dodatočných funkcionalít je aj upozornenie hostiteľskej aplikácie pri prekročení limitu frekvencie requestov od jedného užívateľa, čo by mohlo indikovať napríklad DoS útok.

5.2 Štruktúra identifikátoru

Odtlačok je teda kombináciou konkrétne 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 funkcií 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 geo-zón geolokáciou IP adresy klienta.

5.2.1 Statické dáta

Statická časť identifikátora obsahuje väčšie množstvo dát, preto je ukladaná reprezentovaná v podobe reťazca. Skladá sa z troch kategórií dát:

- Statické hlavičky *(Static headers)*- špecifické vopred vybrané HTTP hlavičky ako napríklad: *authorization, cookie*, alebo *user-agent*
- Ostatné 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 informácie použité pre statické dáta obrázok 5.2.

Každá z týchto informácii má pri použití v porovnávacej funkcií svoj účel a konkrétnu váhu. Veľmi podstatný je napríklad obsah statických hlavičiek. Naopak váha hodnôt TCP okna je porovnaní z IP adresou a geolokáciu menšia, môže však hrať rolu u veľkého množstva requestov. Podrobnejšie sa porovnávacej funkcií a kalibrácií jednotlivých parametrov venuje podkapitola ??x??.

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

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

vyžadujú. Takýchto informácií nie je mnoho, preto nie je nutné ich komprimovať a serializovať ako dáta statické. Konkrétne sa jedná o:

- 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 rozdieli porovnávaných requestov, je ich vzdialenosť dodatočne znížená. U geolokácie môže byť toto zníženie dosiahnuté po vyhodnotení requestu klienta z krajiny s najčastejším pôvodom DoS útokov akými sú napríklad: Čína, Kórea, alebo Rusko.

Táto štruktúra statických a relačných dát odtlačku je v aplikácií 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 predchádzajúcej kapitole, hlavnou funkcionalitou práce je algoritmus na určenie podobnosti a hľadanie najbližšieho suseda aktuálne spracovávaného identifikátoru.

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

Obr. 5.3: Pseudo-kód popisujúci algoritmus pre nájdenie odtlačku s najmenšou vzdialenosťou, zdroj: vlastné spracovanie

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 známi 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:

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

$$0 \leq J(A,B) \leq 1 \tag{5.2}$$

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 a prieniku a menovateľom je zjednotenie týchto množín:

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{5.3}$$

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.

Obr. 5.4: Znázornenie prieniku a zjednotenia množín pre výpočet Jaccardovho koeficientu, zdroj: vlastné spracovanie

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 5.6.

Obr. 5.5: Príklad kalibrácie váh jednotlivých parametrov statickej časti odtlačku pre výpočet vzdialenosti, zdroj: vlastné spracovanie

Kľúčovú rolu pri výpočte samozrejme zohráva kalibrácia váh a parametrov tohto algoritmu. Nasledujúci pseud-kód podrobne popisuje funkciu a jej fungovanie:

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

    if actualData = stockEntityData
        return 0
    end if

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

```
# Compute Jaccard and consider weight (strength) for
# unknown headers
jaccardUnknownHeaders
    ← jaccardIndex(actualData.unknownHeaders,
                   stockEntityData.unknownHeaders);
jaccardUnknownHeaders
    ← jaccardUnknownHeaders * UH_WEIGHT;

# Compare values of remaining attributes by simple
# 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 technológií. 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 adreis. 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 vopered 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 a 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 spracovanie 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 servery, preto existuje možnosť aplikáciu používat 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 vyššie, 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 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 konfirmácie, ktoré sú následne reprezentované triedou *HTTPFootprint*. Štruktúra tried *HTTPFootprint* a *TCPootprint* je dostupná v prílohe ??.

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

Najdôležitejšou časťou celého procesu je však analýza triedou *FootprintSimilairtyService*, ktorá implementuje algoritmus hľadania najbližšieho suseda formálne popísaný v predchádzajúcich častiach práce. 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 databázy, alebo ide o opakujúcu sa reláciu, čím zvýši jeho frekvenciu, prípadne upozorní hostiteľskú aplikáciu na možné prekročenie jej limitu.

Obr. 5.6: Diagram znázorňujúci priebeh spracovania a analýzy podobnosti reqestu, zdroj: vlastné spracovanie

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 kalibrácia parametrov algoritmu obsahuje repozitár okrem knižnice samotnej aj príklad hostiteľskej aplikácie, v ktorej je knižnica použitá. Tento server takisto poskytuje možnosť zobrazenia jednoduchých štatistík z analýzy podobnosti.

6 Test s reálnymi dátami

TODO

7 Záver

TODO

A Príloha

Appendices of thesis.