VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

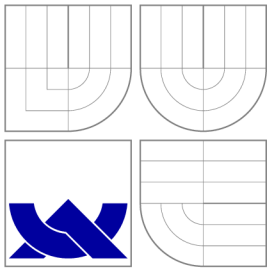
METRIKY PRO DETEKCI BUFFER-OVERFLOW ÚTOKŮ NA UDP SÍŤOVÉ SLUŽBY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

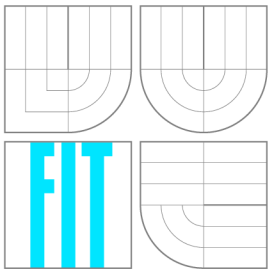
AUTOR PRÁCE
AUTHOR

LADISLAV ŠULÁK

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

METRIKY PRO DETEKCI BUFFER-OVERFLOW ÚTOKŮ NA UDP SÍŤOVÉ SLUŽBY

METRICS FOR BUFFER OVERFLOW ATTACKS DETECTION OF UDP NETWORK SERVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LADISLAV ŠULÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. IVAN HOMOLIAK
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce se zabývá problematikou síťových útoků a jich detekcí v síťovém provozu. Cílem práce je navrhnout takovou sadu metrik, která popisuje síťový provoz na základě jeho chování a pomocí které jsou schopny odhalit i Zero-Day útoky. Další částí práce je popis navrženého nástroje pro jejich extrakci.

Abstract

This bachelor thesis deals with problematic of network attacks and their detection in network traffic. The aim is to propose such collection of metric, that will describe network traffic according to its behaviour, and will be capable of detection of Zero-Day attacks as well. Following part of this thesis is to implement a tool for metric extraction.

Klíčová slova

Buffer overflow útoky, zero day útoky, síťové útoky, IDS, UDP, Honeypot, škodlivý software, síťová bezpečnost

Keywords

Buffer overflow attacks, zero day attacks, network attacks, IDS, UDP, Honeypot, malware, network security

Citace

Ladislav Šulák: Metriky pro detekci buffer-overflow útoků na UDP síťové služby, bakalářská práce, Brno, FIT VUT v Brně, 2015

Metriky pro detekci buffer-overflow útoků na UDP síťové služby

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Ivana Homoliaka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ladislav Šulák
20. mája 2015

Poděkování

Veľmi rád by som sa poďakoval vedúcemu bakalárskej práce Ing. Ivanovi Homoliakovi za vedenie tejto práce ako aj za odbornú pomoc, usmernenie, ochotu a čas strávený pri konzultáciách k tejto práci. Ďalej by som sa rád poďakoval všetkým, ktorí ma v nej podporovali.

Obsah

Kapitola 1

Úvod

V dnešnej dobe, keď je väčšina informácií uložených na počítačoch, a sú prostredníctvom nich ovládané mnohé kritické systémy, je otázka bezpečnosti veľmi častou a dôležitou témou. Dôsledky útokov môžu byť rôzne, od výpadku menej významného systému, až po škody v hodnote niekoľko miliónov eur[1]. V tejto práci sa zameriame na problematiku sieťových útokov s dôrazom na útoky typu *buffer overflow* a na zlepšenie ich detekcie. Výskyt *buffer overflow* útokov je ako v súčasnosti, tak aj v celej histórií [20] sieťových útokov takmer najvyšší. Ich počet sa z roka na rok mení, štatistiky[1] ukazujú, že aj keď niektoré roky majú slabšie percentuálne zastúpenie spomedzi všetkých útokov, pribúdajú čoraz viac a je nutné im venovať zvýšenú pozornosť. Spôsobujú, že anonymný užívateľ získa čiastočnú alebo úplnú kontrolu nad cieľovým zariadením.

Existujú rôzne prístupy pre detekciu sieťových útokov. Väčšina z nich analyzuje prenesené dáta a na základe nich sa detekčné nástroje podľa určitých známych znakov snažia odhaliť či sa jedná o útok. Pri použití takýchto techník klesá šanca na odhalenie nových útokov, pretože útočníci využívajú rôzne metódy pre obfuskáciu[2] kódu ako napríklad použitie pseudo-náhodných čísel pri tvorbe polymorfného kódu, logických operácii a iných substitúcií alebo pridaním prebytočného kódu. Obfuskácia má za následok zmenu útočníkovho kódu a detekčné nástroje nemusia byť pri svojej činnosti úspešné. Dáta, ktoré sú prenášané cez sieť naviac môžu byť zašifrované, a takéto systémy nie sú schopné ich dešifrovať. Táto práca sa zameriava na analýzu sieťovej komunikácie bez využívania známych signatúr, teda bez analýzy prenášaných užitočných dát. V práci boli navrhnuté metriky, ktoré sa zameriavajú na *buffer overflow* útoky nad UDP[3] sieťovými službami. Pod pojmom metrika sa rozumie nejaký štandard alebo funkcia, ktorá meria alebo vyhodnocuje dáta s využitím čísel, poprípade štatistiky[4]. Navrhnutá kolekcia metrík popisuje sieťovú prevádzku na základe jej správania. Vychádza z existujúcich metód a bola navrhnutá pre možné zachytenie aj *Zero-day* útokov. Tieto útoky sú definované ako útoky, ktoré využívajú ešte nezabezpečené zraniteľné miesta v software, teda k nim ešte neexistuje oprava.

Ako ďalším krokom bolo implementovať nástroj, ktorý z prijatých paketov extrahuje tieto metriky. Experimentovaním bude zisťovaná úspešnosť a vlastnosti navrhnutej kolekcie metrík s využitím klasifikátorov v nástroji RapidMiner[5].

[1] Útoky, ktoré mali za následok najväčšie finančné škody boli spomenuté v http://www.businesspundit.com/10-most-costly-cyber-attacks-in-history/.

[2] Obfuscation. http://www.techopedia.com/definition/16375/obfuscation

[3] User Datagram Protocol. https://tools.ietf.org/html/rfc768

[4] Metric. http://dictionary.reference.com/browse/metric

[5] RapidMiner Studio je voľne dostupný analytický a dolovací nástroj [4].

1.1 Štruktúra práce

Nasledujúca kapitola 2 bude venovaná princípom útokov typu *buffer overflow*. Budú diskutované problémy, ktoré útočník musí riešiť pri ich tvorbe a úspešnom nasadení. V ďalšej časti tejto kapitoly budú vysvetlené princípy pre detekciu sieťových útokov. Súčasťou tejto sekcie bude vysvetlenie detekcie pomocou *honeypotov*.

Kapitola 3 sa bude venovať analýze navrhnutej sade metrík s detailným popisom. Návrh nástroja pre extrakciu týchto metrík bude popísaný v kapitole 4. V nasledujúcej kapitole 5 bude popísaný princíp implementácie tohto nástroja. Testovanie a popis experimentov budú vysvetlené v kapitole 6. Posledná kapitola 7 bude vyjadrovať zhodnotenie celého projektu, ako i možnosti ďalšieho zdokonalenia.

Kapitola 2

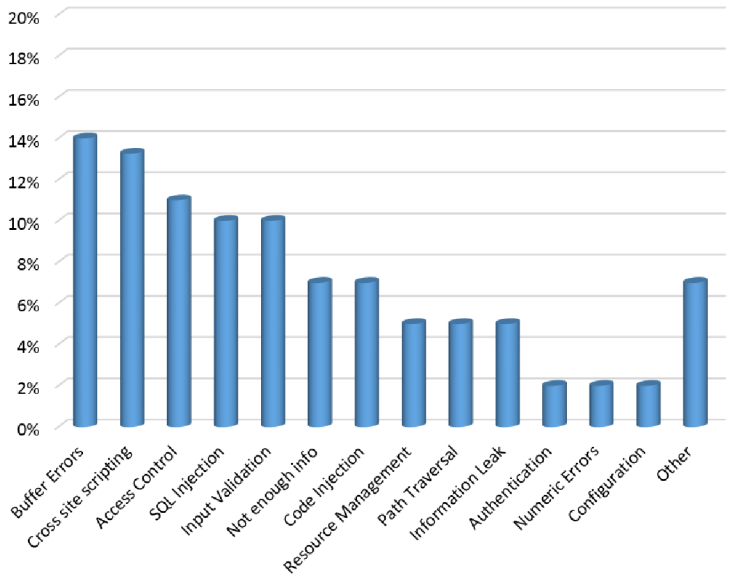
Princípy realizácie sieťových útokov a ich detekcia

Existuje celá rada sieťových útokov. Táto práca sa zameriava na tie, ktoré majú najväčšie zastúpenie, a zároveň aj veľmi závažný dopad na cieľové zariadenie. Jedná sa o útoky typu *buffer overflow*, ktoré spôsobujú že užívateľ bez privilegovaného prístupu môže získať kontrolu nad daným systémom. Tieto útoky začali byť populárne najmä od roku 1996 po vydaní článku *Smashing The Stack For Fun And Profit* [7] zverejnenom na portáli *Phrack*. Štatistika zraniteľností v programoch, ktoré vedú k útokom, sú zachytené na obrázku 2.1. Jednotlivé stĺpce, v grafe znázornené modrou farbou, vyjadrujú percentuálne zastúpenie daného typu zraniteľnosti.
Na druhom mieste majú zastúpenie útoky *Cross site scripting* nasledované útokom typu *SQL injection*.
Cross site scripting (XSS) je technika, ktorá využíva kritické miesta v skriptoch webových stránok, predovšetkým neošetrené vstupy zadané užívateľom. Vďaka tomu je možné napríklad pozmeniť webovú stránku alebo získať údaje o užívateľoch, ktorí ju navštevujú.
SQL injection tiež využíva nedostatočne ošetrené vstupy. Útočník sa snaží bez legitímneho prístupu manipulovať s dátami v databáze.

Štatisticky bolo zistené [20], že zo všetkých produktov má Linuxové jadro najviac potenciálne možných zraniteľných miest, no produkty spoločnosti Microsoft majú najväčšie zastúpenie v kategorizácii podľa predajcu. Čo sa týka mobilnej technológií najmenej zabezpečený systém je iPhone, v porovnaní so systémom Android bolo nahlásených až takmer 9-krát viac zraniteľných miest.

Útočník sa vo všeobecnosti snaží vykonať nasledujúce 4 kroky vedúce k útoku:

- Zbieranie informácií o zraniteľnostiach v čo najväčšom počte. Tieto informácie sa týkajú cieľového systému a siete, napríklad čísla portov zariadení, služieb, operačných systémov a ďalších. Tieto typy nástrojov budú popísané v sekcií 2.1.
- Zneužitie danej zraniteľnosti. Útočník sa snaží kompromitovať nejaký uzol v sieti ako predzvesť útoku.
- Spustenie útoku na cieľovom zariadení s využitím kompromitovaného uzlu z predchádzajúceho bodu. Pri tom sa môžu využívať nástroje popísané v sekcií 2.1.
- Odstránenie stôp. V poslednom kroku sa útočník snaží eliminovať históriu svojej aktivity vyčistením registrov alebo logovacích súborov z každého napadnutého zariadenia.



Obr. 2.1: Súhrnné štatistiky za posledných 25 rokov, rozdelenie podľa najfrekventovanejších typov zraniteľností [20].

Prvá časť tejto kapitoly sa venuje nástrojom, ktoré využívajú útočníci pri generovaní útokov. Ako hlavným zdrojom informácií o tejto problematike bude [15]. V tejto kapitole budú ako ďalšie vysvetlené princípy tvorenia útokov typu *buffer overflow*. Informácie o nich budú čerpané z [7], [13], [12] a [11]. Kapitola bude ďalej pokračovať popísaním techník, ktoré a využívajú pri detekcii sieťových útokov. Informácie o tejto problematike budú čerpané z [16], [10], [9] a o *honeypotoch* z knihy [18].

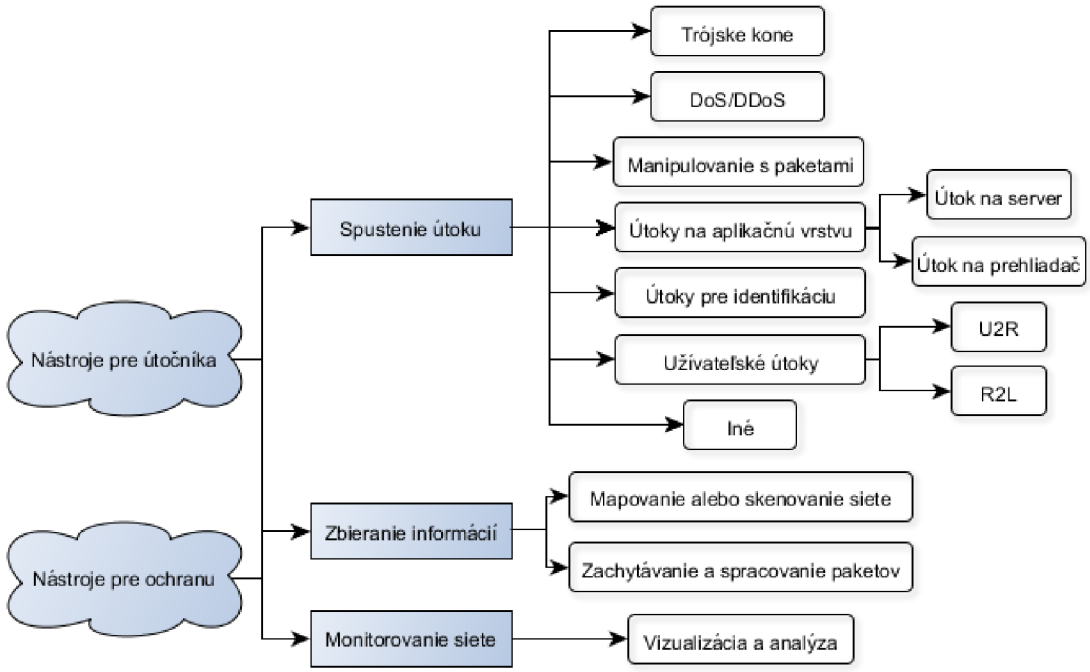
2.1 Sieťové nástroje využívané útočníkmi

Tieto nástroje boli kategorizované [15] do 2 skupín: pre ľudí, ktorí útoky vytvárajú, a ktorí sa im snažia zabrániť. V tejto podkapitole bude vysvetlený princíp fungovania takýchto nástrojov aj s príkladmi. Obrázok 2.2 popisuje kategorizáciu týchto nástrojov.

- **Nástroje slúžiace pre zbieranie informácií**. Podľa [15] sa delia nasledovne:

  Nástroje pre zachytávanie paketov, angl. *Sniffing tools*, no poskytujú aj iné služby. Ich cieľom je zachytiť, preveriť, analyzovať a vizualizovať pakety alebo rámce prechádzajúce cez sieť. Medzi najpoužívanejšie nástroje patria:

  - **Tcpdump**. Jeden z najznámejších a najpoužívanejších nástrojov, ktorý je schopný prezerať, uložiť a odchytiť pakety. Podobný nástroj je Ethereal alebo Snoop, ktorý ale pracuje s odlišným formátom súboru.
  - **Net2pcap**. Oveľa jednoduchší nástroj než predchádzajúci, slúži na prezeranie paketov z nejakého sieťového zariadenia a ukladanie do pcap súboru.
  - **Ngrep** bol vytvorený pre filtrovanie užitočných prenesených dát v pakete.
  - **Ettercap** funguje takmer na všetkých platformách. Dokáže zozbierať heslá pre niekoľko aplikácií, prerušiť spojenia a vložiť pakety alebo príkazy do aktívneho spojenia. Okrem toho má ešte mnoho ďalších vymožeností a doplnkov.



Obr. 2.2: Kategorizácia nástrojov pre oblasť sieťovej bezpečnosti.

- **Nfsen** je používaný pre vizualizáciu NetFlow dát. Dokáže znázorniť zachytené dáta ako tok paketov alebo bytov s využitím grafického rozhrania.
- Niektoré nástroje tejto kategórie sú určené pre získavanie hesiel ako napríklad **Cain and Able** fungujúcim pod operačným systémom Windows.

Nástroje pre skenovanie siete, angl. *Scan tools*. Ich cieľom je identifikovať aktívneho hosta v sieti buď za účelom zaútočiť alebo zistiť informácie o zraniteľnostiach v sieti alebo cieľovom zariadení. Väčšinou sú založené na Linuxovom jadre.

- **Nmap** dokáže skenovať pomerne rýchlo dokonca aj rozsiahle siete. Je efektívny pre identifikovanie rôznych parametrov ako napríklad hosta, poskytovaných služieb, operačného systému a protokolov alebo nasadenie rôznych paketových filtrov, či *firewallov*. Je možné ho spustiť v tichom móde, v ktorom zasiela pakety s niekoľko minútovými prestávkami, čím je takmer nemožné ho odhaliť.
- **Amap** detekuje aplikačný protokol zaslaním určitých paketov.
- **Nessus** [3] je jeden z najpoužívanejších nástrojov, ktorý slúži pre skenovanie zraniteľností na zvolenom cieľovom zariadení. Spolu s nástrojom Hydra dokáže generovať tzv. slovníkový útok, angl. *Dictionary attack*, ktorý sa snaží odhaliť heslo spomedzi nejakej množiny pravdepodobných hesiel z pripraveného slovníku.

- **Nástroje pre vykonávanie útoku**. Mnoho z nich je voľne dostupných, nevyžadujú veľmi podrobné vedomosti o danej problematike, no ich použitie vo verejnej sieti môže byť trestné. Existujú univerzálne nástroje, ktoré dokážu generovať širokú škálu útokov, a špecializované nástroje, ktoré sa zameriavajú na daný typ útokov.

Univerzálne nástroje generujú viacero druhov útokov, niektoré naviac poskytujú informácie o cieľovom zariadení, zraniteľnostiach a podobne. Medzi univerzálny nástroj, určený hlavne pre penetračné testovanie, rozširovanie a zdokonalovanie databázy signatúr pre *IDS* (viď sekcia 2.3), ktorý generuje útoky je *Open Source* nástroj Metasploit Framework [2]. Je voľne dostupný a spadá pod Metasploit project, ktorý ponúka viacero produktov. Obsahuje množstvo prídavných modulov, dokáže generovať viacero druhov útokov, ktoré sú rozdelené podľa typu *payloadu*.

Špecializované nástroje vznikli kvôli existencii mnohých typov útokov smerujúcich na sieťový uzol alebo cieľové zariadenie. Nástroje, ktoré ich dokážu generovať, sa zvyknú na dané typy špecializovať podľa útokov:

- **Trójske kone**
  Jedná sa o spustiteľné súbory, ktoré boli vytvorené za účelom prelomiť zabezpečenie systémov počítača alebo siete. Tento typ súborov sa nachádza v systéme ako obyčajný spustiteľný súbor, no ku škodlivej aktivite dôjde až keď bude spustený. Obete útoku ho väčšinou získajú nevedome sťahovaním z internetu, FTP archívu, pomocou P2P technológie alebo pomocou technológie Internet Messaging.
- **DoS/DDoS útoky**
  Denial of Service. Tieto útoky sa snažia obmedziť alebo zamedziť prístup legitímnym užívateľom k nejakej službe alebo systému.
  A Distrubuted Denial of Service. Býva spustený nepriamo z velkého počtu kompromitovaných zariadení, teda je komplikované zistiť počiatočný zdroj útoku. Smerujú na jeden alebo aj na celú skupinu systémov. Detekcia alebo prevencia voči tomuto druhu útoku nie je triviálna. Štatisticky bolo zistené [15], že počet takýchto útokov narastá.

  Nástrojov, ktoré dokážu generovať tieto druhy útokov, je mnoho, napríklad:

  * **LOIC** využíva IRC, dokáže operovať v 3 módoch podľa protokolov TCP, UDP a HTTP. Existuje v binárnej podobe alebo vo webovom prostredí.
  * **HOIC** generuje *DDoS* útoky na báze protokolu HTTP. Zameriava sa na zahltenie HTTP prevádzky, dokáže zahltiť naraz až 256 web stránok.
  * **UDPFlood** dokáže zahltiť špecifický port na špecifickej IP adrese s UDP paketami. Môže byť použitý aj pre testovanie odolnosti serveru. Podobný nástroj je Panther.
  * **Slowloris** vytvára veľký počet pripojení na cieľový webový server zasielaním čiastočných požiadaviek, ktoré sa snažia byť otvorené čo najdlhšie za účelom zablokovať pripojenia od legitímnych užívateľov.
- **Manipulovanie s paketami**
  Takéto nástroje sa používajú ak chce útočník generovať sieťovú prevádzku a manipulovať s IP adresami.

  * **Nemesis** je veľmi rozšírený nástroj, ktorý funguje pod operačným systémom Windows aj pod operačnými systémami založenými na báse Unixu. Používa sa pre generovanie vlastných paketov a podporuje mnoho protokolov ako ARP, DNS, ICMP, IGMP, IP, RIP, TCP, UDP.

    - **Packeth** je nástroj s grafickým užívateľským rozhraním, ktorý pracuje na báze Linuxu. Zasiela pakety alebo postupnosť paketov využívaním socketov na ethernetovej vrstve. Poskytuje mnoho možností ako napríklad zaslať chybnú hlavičku paketov alebo chybný kontrólny súčet.

  - **Nástroje so zámerom zaútočiť na aplikačnú vrstvu**
    Útočník využíva legitímnu aplikačnú vrstvu HTTP požiadavkov z legitímne pripojených sieťových zariadení so zámerom poškodiť alebo zahltiť webový server. Kvôli tomu, že používa legitímne protokoly a pripojenia, je ťažké ho detekovať. Sú delené do 4 kategórií: HTTP, SMTP, FTP a SNMP útoky, poprípade ich derivácie.

  - **Útoky pre identifikáciu**
    Sú používané k identifikácií špecifických prvkov sieťových protokolov, napríklad verzie protokolu, informácie o cieľovom zariadení ako je operačný systém a iné. Nmap je jeden z najpoužívanejších nástrojov, ktorý funguje multiplatformovo. Poskytuje detailné informácie o sieti alebo zariadení so zameraním na kritické informácie, ktoré neskôr môžu byť využité pri samotnom útoku.

  - **Užívateľské útoky**
    Táto kategória nástrojov slúži pre získanie práv správcu, angl. *roota* alebo *superusera*, alebo pre získanie prístupu do lokálneho zariadenia s využitím nejakých zraniteľností a bez legitímneho prístupu. Podľa [15] sa delia na *U2R* a *R2L* útoky.

    - *U2R - User to Root* majú za úlohu získať prístup k zariadeniu ako legitimovaný užívateľ. Po získaní prístupu útočník môže vytvoriť zadné vrátka (*backdoor*) pre budúce útoky, môže manipulovať so systémovými súbormi alebo napríklad zbierať rôzne kritické informácie. Hlavnými predstaviteľmi týchto nástrojov sú Yaga a SQL attack.
    - *R2L - Remote to User*. Útočník, vystupujúci ako vzdialený užívateľ bez účtu na lokálnom zariadení, sa snaží získať prístup prostredníctvom siete pomocou zasielania paketov. Príkladom je nástroj Netcat, ktorý využíva útoky typu Trójsky kôň pre inštalovanie a spustenie Netcatu na cieľovom zariadení, s využitím portu 53 (DNS, funguje pod TCP aj UDP). Funguje teda ako zadné vrátka bez akýchkoľvek prihlasovacích údajov.

  - **Iné nástroje**
    Existuje rada nástrojov, ktoré priamo alebo nepriamo súvisia so sieťovými útokmi. Niektoré, ako Ping testujú sieťové pripojenie alebo dostupnosť. Iné, ako napríklad Traceroute ukazujú cestu medzi 2 systémami v sieti. Tieto nástroje teda negenerujú žiadne útoky, nie sú schopné ich detekovať a ani poskytnúť proti nim akúkoľvek obranu.

- **Nástroje určené pre monitorovanie siete**. Slúžia pre pozorovanie, analýzu a identifikovanie anomálií vyskytujúcich sa na sieti. Nástroj, ktorý efektívne dokáže vizualizovať sieťovú prevádzku je užitočným pomocníkom pre administrátorov a ľudí, ktorí sa snažia o to, aby na sieti nedošlo k úspešnému útoku.
  Na operačnom systéme Windows sa môžeme stretnúť s nástrojom NetViewer, ktorý je pre vizualizáciu v reálnom čase pre detekciu vniknutia najlepšou možnosťou. Zároveň je užitočný pri rôznych defenzívnych mechanizmoch. Pod Unixom sa používa nástroj EtherApe, ktorý tiež dokáže zachytiť a monitorovať pakety v reálnom čase.

2.2 Buffer Overflow útoky

Vznikajú kvôli chybám alebo určitým zraniteľnostiam v programoch, ktorých sa dopúšťajú programátori pri implementácii softwaru. Môžu vznikať kvôli nedostatočnému ošetreniu vstupných reťazcov, poškodením zásobníku, hromady alebo iným chybám, napríklad synchronizácie, angl. *race conditions*, alebo napríklad pretečeniu celočíselného typu premennej ktorá slúži na indexovanie do pamäte. Kompilátory niektorých jazykov nekontrolujú operácie zápisu mimo medze poľa kvôli optimalizáciám, a kvôli tomu je možné zapísať na takéto miesto v pamäti (*buffer*) viac dát než je jeho veľkosť (*overflow*). Predovšetkým sa jedná o jazyk C alebo jeho deriváty a podľa štatistík patria k najpopulárnejším. Využívajú sa pri implementácií mnohých kritických systémov ako sú databázy, mailové alebo webové servery, alebo dokonca aj samotné operačné systémy, a sú často cieľmi útokov. Útočníci samozrejme nemusia využívať pri tvorbe škodlivého kódu iba túto rodinu jazykov, často používajú napríklad skriptovacie jazyky ako Perl k vytvoreniu dlhého reťazca obsahujúceho postupnosť inštrukcií (viď sekcia 2.2.4).

Vo všeobecnosti sa *buffer overflow* delia na 2 hlavné kategórie. Pri prvej nastáva pretečenie na zásobníku, teda *stack overflow* (viď 2.2.2). Druhá kategória využíva pretečenie na hromade a je známa pod pojmom *heap overflow* (viď 2.2.3).

2.2.1 Ochrana proti Buffer overflow útokom

Prevencia, detekcia a ochrana pred týmito útokmi prešla dlhým rozvojom, ktorý stále trvá. Existujú metódy, ktoré sú pre praktické nasadenie neprijateľné a nedokážu detekovať *Zero-Day* alebo polymorfné útoky. Niektoré sú koncipované iba pre ochranu proti *stack overflow*. Nasledujúce členenie bude vysvetľovať reprezentatívne prístupy pre ochranu proti *buffer overflow* útokom:

- **Defenzívne programovanie**
  Už pri vyvíjaní softwaru existujú možnosti, ktoré znižujú šancu, že program bude vystavený týmto útokom. Jedná sa o bezpečné používanie pamäte, testovanie pomocou nástrojov pre statickú alebo dynamickú analýzu zdrojového súboru, a čo najlepšie zabezpečenie programu voči manipulácií, hlavne ošetrenia vstupov od užívateľa. Výhodou tohto prístupu je, okrem toho že výsledný program má nižšiu šancu na kompromitovanie, aj to, že programátori explicitne nemusia vytvárať sadu testov, o to sa väčšinou starajú automatizované nástroje. Nevýhodou je dlhší proces vývoja softwaru, s tým spojené zvýšenie nákladov, avšak tým nie je zaručené to, že software bude zabezpečený dostatočne.

- **Ochrana alebo detekcia za behu programu**, angl. *Runtime instrumentation*
  Medzi hlavné výhody patrí často takmer úplné znemožnenie niektorých druhov útokov, najmä *stack overflow*, no často za cenu vyššej réžie a nie príliš praktického nasadenia. Ďalším nedostatkom je to, že sa väčšinou špecializujú na určité kritické miesta a metódy, ktoré útočníci využívajú, teda nezabraňujú útokom z globálnejšieho hľadiska.

  - **Použitie zásobníkov zabraňujúcich spustenie dát**, angl. *Non-executable stacks*
    Procesor odmietne vykonať inštrukcie ak ukazateľ na najbližšiu inštrukciu, resp. register *EIP* ukazuje na vopred označené miesto v pamäti ako nespustiteľné. Môže to byť realizované hardwarovo alebo softwarovo. V operačnom systéme

Windows je možné stretnúť sa s pojmom *Data Execution Prevention (DEP)*.
Softwarovo sú realizované hlavne ako rozšírenia kompilátorov, teda je nutné mať k dispozícií zdrojový kód zraniteľného programu aby mohol byť re-kompilovaný. Zdrojový kód nemusí byť vždy k dispozícií, hlavne pokiaľ sa jedná o programy typu *closed-source*, no existujú aj iné variácie tejto kategórie, ktoré ho nevyžadujú. Je ho možné získať extrahovaním z binárneho súboru. Príklad sofistikovanejšieho nástroja využívajúceho rôzne metódy, hlavne kontrolu návratových hodnôt, bol uvedený v žurnáli *Procedia Computer Science* [8]. Hlavná myšlienka tohto nástroja spočíva v tom, že sa vytvoria viaceré kópie návratových adries na zásobníku, ktoré potom budú umiestnené do pamäti na náhodné miesta. Počet týchto kópií bude vždy náhodné číslo. K úspešnému útoku je nutné poznať počet kópií, ich umiestnenie v pamäti a modifikovanie všetkých súčasne. Tieto návratové hodnoty sa budú kontrolovať, a ak niektorá z nich bude pozmenená, môže to znamenať, že došlo k útoku. Nedostatkom tohto prístupu je to, že nie vždy sa jedná o efektívne metódy, a programy môžu byť aj naďalej kompromitované útokmi typu *heap overflow* alebo inými druhmi, ktoré nevyužívajú pretečenie na zásobníku. Naopak, útoky typu *stack overflow* sú výrazne znemožnené.
Čo sa týka hardwarovej realizácie, súčasné riešenia vyžadujú špecifickú konfiguráciu, hardware alebo oboje. Takéto riešenie nie je veľmi praktické, hlavne pre koncových užívateľov.

– **Kontrola hraníc poľa**
Na rozdiel od ostatných metód, použitie tejto vedie takmer k úplnému zastaveniu *buffer overflow* útokov. Nemôže dôjsť k pretečeniu poľa a útočník týmto spôsobom nemôže zmeniť tok programu. Pri implementácii takého mechanizmu sa je možné stretnúť s viacerými prístupmi:

  * Jones & Kelly využili tohto princípu a vytvorili *patch* pre kompilátor *gcc*, ktorý vykonáva kontroly, či nedošlo k zápisu za koniec poľa. Skompilované programy sú plne kompatibilné s ostatnými modulmi *gcc*, pretože nedošlo k zmene reprezentácie ukazateľov. Je to ale za cenu efektivity výsledného programu, pretože práve kvôli tomu pôvodný kompilátor tieto kontroly nevykonáva.
  * Kontrola integrity ukazateľa. Cieľom tohto prístupu je detekovať, že došlo k prepísaniu ukazateľa ešte pred tým, než dôjde k pristúpeniu na hodnotu, na ktorú ukazoval. Ak útočník úspešne poškodí ukazateľ, nikdy nebude použitý. Tento prístup má nevýhodu v tom, že existujú aj iné metódy ako vykonať útoky typu *buffer overflow*, na ktoré tento prístup nemá vplyv. Známym príkladom je StackGuard, alebo PointGuard, ktorý pracuje na podobnom princípe.
    StackGuard: kompilátorom generované kontroly integrity aktivačného záznamu. Jedná sa o rozšírenie *gcc* kompilátora, ktorý funguje pomerne efektívne a je jeden z najznámejších pre tento druh ochrany. Princíp je založený na tom, že sa na zásobník umiestni hodnota, nazývaná *Canary Value*, ktorá je uložená medzi vyrovnávacou pamäťou a registrami na zásobníku (napr. *EIP*, *EBP*). Počas útoku typu *stack overflow* bude táto hodnota prepísaná ešte predtým než hodnota v registri *EIP*. Pri návrate z funkcie sa táto hodnota kontroluje a ak bola prepísaná, program typicky skončí s chybou. Môže sa jednať o náhodnú 32-bitovú premennú, ktorá sa vytvára pri štarte programu,

angl. *Random Canary*, alebo sa použije hodnota pre ukončenie riadku, reťazca, alebo súboru. V tomto prípade hovoríme o *Terminator Canary*. Útočníkovi zamedzuje použitie bežných knižníc jazyka C pre prácu s reťazcami alebo poľami, pretože keď na tieto symboly narazia, kopírovanie pomocou týchto funkcií bude ukončené. StackGuard bol prvý krát použitý v operačnom systéme Red Hat Linux 5.1.

- **Adress space layout randomization** *(ALSR)* [19]
  Sem patria techniky, ktoré umiestňujú strojový kód programu, dáta a knižnice v operačnej pamäti od náhodne zvolenej adresy. Táto technika sa používa pre posilnenie obrany voči *buffer overflow* útokom. Útočník musí buď vytvoriť špecifický program pre každú inštanciu cieľového programu, alebo pomocou metódy *brute-force* odhadnúť rozmiestnenie adresového priestoru. Tieto metódy nemusia byť úspešné, pretože rozmiestnenie sa mení pri každom reštarte programu. Ak sa útok nepodarí a cieľový program zlyhá, adresový priestor bude opäť náhodne usporiadaný. Tento prístup má značnú výhodu v úspešnosti proti útokom. Bola použitá ako na Linuxe, pomocou *PaX ALSR*, OpenBSD, Windows Vista a ďalších.

  Sofistikovanejšie metódy útokov voči *ALSR* sú napríklad derandomizované útoky, ktoré boli predstavené v [19]. V tejto práci autori *buffer overflow* kód transformovali tak, aby sa útok vykonal aj napriek náhodného rozmiestnenia pamäťového priestoru. Bolo tiež nutné použiť metódu *brute-force*, no iba pre získanie hodnoty tzv. *delta_mmap (mapped data offset)*, ktorá ukazuje na presnú adresu v štandardnej knižnici *libc*. Často sa pri tom využíva funkcia mmap() a ďalšie zo štandardnej knižnice *libc*. Výsledný kód mal rovnaký efekt, no bol pomalší. Experimentovaním [19] bolo zistené, že na 32-bitovej architektúre je problematické navrhnúť systémy, ktoré odolajú *brute-force* útokom. Naopak, na 64-bitovej architektúre sa riziko výrazne znižuje. Hlavné nevýhody *ALSR* sú tie, že kvôli náhodnému rozmiestneniu adresového priestoru je problematickejšie ladenie a analýza programov pri ich zrútení. Kvôli tomu a ďalším nevýhodám je možné *ALSR* vypnúť, no je nutné brať do úvahy, že sa zabezpečenie systému zníži.

- **Izolácia, použitie mechanizmu** *sandbox*[6]
  V iných programovacích jazykoch než je rodina jazykov C sa využívajú celkom odlišné bezpečnostné prístupy. Jeden zo súčasných prístupov, akým sú niektoré programy chránené voči manipulácii a útokom *buffer overflow*, je prítomnosť bezpečnostého mechanizmu *sandbox*, ktorý oddeľuje spustené procesy. Používa sa v programovacích jazykoch ako je Java. Spoľahlivosť bezpečnostného modelu a bezpečnosť programov napísaných v Jave závisí od 3 častí:

  – *The byte code verifier*. Pri skompilovaní zdrojového kódu Javy sa vytvára platformovo nezávislý byte kód. Ten je následne verifikovaný a až potom dochádza k jeho spusteniu. Počas procesu verifikácie sú vykonávané rôzne testy, napríklad kontrola správnosti formátu fragmentov byte kódu.

  – *Java Class loader*. Všetky objekty prislúchajú nejakým triedam. Zisťuje sa kedy a ako môže aplikácia pridať triedy do bežiaceho prostredia Javy. Dôležitou častou

[6] Klúčové prvky podieľajúce sa na bezpečnosti Javy. http://www.javaworld.com/article/2076945/java-security/understanding-the-keys-to-java-security----the-sandbox-and-authentication.html

tohto prístupu je sledovanie, či dôležité časti prostredia Javy nie sú nahradené kódom, ktorý sa aplikácia snaží nainštalovať.

– *Java Security Manager*. Je to v podstate jednoduchý modul, ktorý vykonáva kontroly za behu aplikácie. Tieto kontroly sa týkajú kritických a potencionálne nebezpečných metód. Tento modul môže úplne zakázať operácie vyhodením bezpečnostnej výnimky, angl. *Security exception*. Berie do úvahy aj triedy, ktoré sú načítavané *Java Class loaderom*. Vstavaným triedam sa dáva väčšie oprávnenie než triedam načítaných cez internet.

Spoločne sa nazývajú *sandbox*. Ako bolo spomínané, vykonávajú rôzne kontroly pre načítanie, behové kontroly alebo kontroly súborového systému a sieťového prístupu. Myšlienkou je stanoviť čo aplikácia môže vykonávať, resp. jej obmedziť isté vlastnosti, a sledovať, či sú dodržiavané.

- **Statická alebo dynamická analýza**
  Využíva sa najmä pri defenzívnom programovaní a penetračných testoch. Tieto metódy sa snažia odhaliť zraniteľné miesta v programoch, ktoré útočník môže využiť. Často býva, najmä na túto prevenciu, vynaložené veľké úsilie, pričom nemusia byť odhalené všetky zraniteľnosti.

  – Audit zdrojových súborov

    * Statická lexikálna analýza kódu
      Takéto nástroje sú pomerne jednoduché, ale ich nevýhodou je to, že nie sú vždy presné a tiež nekontrolujú veľkosť poľa. Typicky uchovávajú sadu nejakých konštrukcií alebo znakov, ktoré vyhľadávajú v zdrojovom kóde, napríklad prítomnosť a použitie známych kritických funkcií pre prácu s poľami. Jednoduchšie fungujú na báze *Grepu*, čo je Linuxový nástroj pre vyhľadávanie reťazcov alebo častí textu. Sofistikovanejšie hľadajú známe zraniteľné miesta, napríklad RATS, ITS4 alebo Flawfinder.
    * Statická sémantická analýza kódu
      Tento druh nástrojov vykonáva podobné kontroly ako vyššie uvedená kategória, no berú do úvahy aj kontext. Príkladom je nástroj SPLINT.
    * Použitie algoritmov umelej inteligencie alebo metódy učenia pre statickú analýzu zdrojového súboru
      Zraniteľnosti v kóde sa detekujú a identifikujú pomocou kombinácie lexikálnej, sémantickej analýzy a cez expertné, samoučiace sa systémy.
    * Dynamická analýza
      Do tejto metodiky patria rôzne ladiace nástroje, ktoré detekujú úniky pamäte alebo iné možnosti napomáhajúce pre detekciu možného výskytu *buffer overflow*. Napríklad nástroj Valgrind alebo Rational Purify[7].

  – Audit binárnych súborov

    * Black box testing
      Typicky sa používajú rôzne sady skriptov navrhnutých pre poskytovanie rôznych variácií vstupov do testovaného programu.

[7]Nástroj pre dynamickú analýzu zdrojového textu. Poskytuje detailné diagnostické informácie hlavne o chybách pri práci s pamäťou. http://www-03.ibm.com/software/products/en/rational-purify-family

    * Reverzné inžinierstvo
      Jedná sa o proces dekompilovania binárneho súboru do jazyka symbolických inštrukcií, teda Asembleru, alebo ešte vyššieho jazyka. Táto technika býva často veľmi komplikovaná, hlavne v prípadoch, keď je program chránený obfuskáciou, šifrovaním alebo inými metódami.
    * Vyhľadávanie špecifických chýb
      Proces vyhľadávania *buffer overflow* v kompilovanom programe. Analógia k lexikálnej a sémantickej analýze, ale na úrovni jazyka Asembler.

- **Sieťová detekcia útokov**
  Pri detekcií sieťových útokov sa využívajú systémy, ktoré sú v princípe rozdelené na 2 kategórie, a to detekcia na základe signatúr a detekcia na základe anomálií. V poslednej dobe sa na detekciu útokov často využíva nasadenie monitorovacieho informačného systému *honeypot* alebo celej skupiny *honeypotov*, spoločne nazývaných *honeynet*.
  Uvedené metódy, ich prednosti a nedostatky, budú detailne vysvetlené v kapitole 2.3.

2.2.2 Stack Overflows

Pri tejto kategórií útokov je nutné zmeniť tok behu programu tak, aby bolo možné spustiť útočníkov kód (*shellcode*, detailnejšie vysvetlené v sekcii 2.2.4). Je to realizované pretečením typicky poľa s cieľom využiť register ukazateľa na ďalšiu inštrukciu (*EIP*) tak, aby ukazoval na útočníkov kód, teda na postupnosť inštrukcií procesoru ktorá sa bude najbližšie vykonávať.
V štandardnej knižnici jazyka C sa pre prácu s poľami nachádzajú potencionálne nebezpečné funkcie, ktoré nekontrolujú veľkosť zdrojového reťazca, napr. *strcpy()*, *strcat()*, *sscanf()* a ďalšie. Takmer ku každej existuje ekvivalentná funkcia, ktorá túto kontrolu vykonáva. Je to realizované pomocou parametra, a ak programátor neurobí chybu pri jeho výpočte, k pretečeniu zásobníka pri nich nedôjde. Útočníci ale niekedy využívajú aj iné techniky, napríklad *integer overflow*, ku ktorému dôjde pri pretypovaní premennej, ktorá sa využíva pri uložení dĺžky reťazca, typicky prekročením rozsahu zo znamienkového celočíselného typu na neznamienkové, čo má väčšinou za následok nedefinované chovanie, poškodenie dát alebo dokonca spustenie kódu.

Existujú metódy, napríklad *BSS overflow*[8], pri ktorých sa *EIP* neprepisuje a útočník nemusí použiť žiadny *shellcode*. Namiesto toho využívajú iné metódy ako *Off-by-One overflow*, ku ktorému dochádza ak sa do poľa skopíruje o 1 byte viac tak, aby útočník prinútil program vykonať iný príkaz než programátor pôvodne chcel.

Shellcode je ale v prevažnej väčšine útokov využívaný a jeho spustenie je realizované rôznymi metódami, k najčastejším z nich patria:

- Prepísanie návratovej hodnoty funkcie
  Pri tejto a nasledujúcej technike sa mení adresa ukazateľa na miesto v pamäti, kde začína útočníkov kód. Zápis mimo hranice v oblastiach alokovaných na zásobníku môže mať za následok prepísanie návratových adries funkcií. Najpoužívanejšia technika je prepísanie uloženej hodnoty v bázovom registri *EBP*, ktorý sa používa pri uložení zásobníkového rámca *(stack frame)* pri prologu funkcie. Pri epilogu sa kopíruje táto hodnota do registra *ESP* obsahujúceho hodnotu na vrchole zásobníka, ktorá bude

[8] .bss je segment pamäte, kde sa nachádzajú statické dáta

použitá pri návrate do nadradenej funkcie. Tento prístup má výhodu vo svojej jednoduchosti, no ako hlavným nedostatkom je to, že je pomerne ľahko detekovateľný, napríklad pomocou rôznych techník, ktoré boli spomenuté v sekcií 2.2.1.

- Prepísanie ukazateľa na funkciu
  Táto varianta sa využíva keď je ukazateľ na funkciu definovaný ako lokálna premenná. Jedna z výhod je tá, že pri napadnutí systému obsahujúceho kontroly či došlo k modifikácií uloženej návratovej hodnoty, ako napríklad StackGuard, nebude útok detekovaný.

- VPTR smashing
  Kompilátory C++ používajú virtuálnu tabuľku funkcií *VTBL* asociovanú s každou triedou. *VTBL* obsahuje pole ukazateľov na funkcie *VPTR* (*virtual pointers*). Prepísanie virtuálneho ukazateľa môže mať za následok volanie funkcie útočníka. To nastane v prípade volania virtuálnej funkcii. Výhoda tohto prístupu spočíva v tom, že sa môže používať pri poškodzovaní zásobníku aj hromady, no je komplikovanejšia a nie je univerzálna. Je nutné, aby bol cieľový program napísaný v jazyku C++ resp. aby bol použitý objektovo orientovaný prístup.

Okrem nich existujú ešte iné postupy, ktoré majú za úlohu spustiť externý kód. Napríklad prepísanie oblastí, ktoré vznikajú pri zavádzaní programu do pamäte. Toto môže byť často využité pri *stack overflow* aj *heap overflow*. Je nutné mať informácie o týchto oblastiach, čo nie je vždy možné. Spustiteľný formát obsahuje mnoho sekcií, útočníci sa zameriavajú na poškodenie najmä nasledovných:

- *.data* - statické dáta
- *.dtors* - ukazatele na funkcie, ktoré sa majú vykonať po ukončení hlavnej funkcie programu
- *.got* - ukazatele na adresy funkcií používaných z dynamicky linkovaných knižníc. V prípade statického linkovania obsahuje ukazatele na adresy všetkých funkcií.

2.2.3 Heap Overflows

Táto kategória útokov má štatisticky o niečo menšie zastúpenie v porovnaní s predchádzajúcou, no v princípe je scenár veľmi podobný - poškodenie určitého miesta pamäti pretečením, v tomto prípade hromady, za účelom zmeniť tok programu, dát alebo vložiť a spustiť kód. Hromada je časť pamäte počítača dynamicky alokovaná za behu programu a obsahuje hlavne dynamicky alokované premenné. Skladá sa z objektov nazývaných *chunks*, ktoré obsahujú hlavičkovú štruktúru a voľné miesto, v ktorom sa pri alokovaní ukladajú dáta. Hlavička obsahuje informácie o veľkosti aktuálneho a predchádzajúceho objektu, a tiež je v nej uložené či je predchádzajúci objekt uvoľnený, alebo nie. To je umiestnené v poslednom významovom bite v časti pre aktuálnu veľkosť objektu. Knižnice v jazyku C poskytujú funkcie pre manipuláciu s týmto typom pamäte, ako napríklad *malloc()* a *free()*.
Útočník často využíva 2 objekty susediace na hromade vedľa seba. Snaží sa prepísať virtuálny ukazateľ na funkciu susediaceho objektu. Pri úspechu sa vloží hodnota ukazujúca na nejaké miesto v pamäti. Tam je vytvorená nová tabuľka virtuálnych funkcií za účelom spustenia útočníkovho *shellcodu*, a to ak dôjde k spusteniu jednej z triednych funkcií. Často sa využíva deštruktor, pretože je spustený vždy keď dôjde k vymazaniu objektu z pamäte,

alebo sekcie vytvorenej pri spustiteľnom formáte (viď koniec 2.2.2). Rôzne iné metódy [16], ako využiť poškodenie hromady pre zmenu toku programu, môžu byť:

- *Off-by-One* a *Off-by-Five* útoky využívajúce prepísanie posledného významového bitu časti hlavičky, ktorá určuje aktuálnu alebo predchádzajúcu veľkosť objektu.
- *Double-free* útoky nemusia využívať pretečenie. Snažia sa prepísať ukazateľ na predchádzajúci alebo nasledovný voľný objekt.

Útoky typu *heap overflow* vyžadujú v cieľovom programe používanie objektov alebo abstraktných dátových štruktúr alokovaných na hromade, no často je pre úspešný útok nutné použiť iné metódy.

2.2.4 Shellcode

Shellcode je byte kód, ktorý obsahuje postupnosť inštrukcií. K jeho spusteniu dochádza po úspešnom zneužití nejakej zraniteľnosti a následného vloženia do zraniteľného programu. Pre pochopenie a vytváranie *shellcodu* je nutná znalosť jazyka *Asembler*, cieľovej architektúry a operačného systému. Typicky sa vytvára nasledovne:

- Implementácia programu, ktorá využíva nejaké kritické miesto alebo zraniteľnosť cieľového zariadenia, s využitím systémových volaní. Tie je možné spustiť na úrovni jazyka Asembler pomocou inštrukcie pre softwarové prerušenie. Niektoré operačné systémy predpokladajú umiestnenie jeho argumentov do registrov (Linux), iné na zásobník (FreeBSD).
- Následné prepísanie vzniknutého programu do jazyka Asembler. Pomocou operačných kódov inštrukcií sa prepíše do hexadecimálnej postupnosti s dôrazom na minimálnu veľkosť a maximálnu prenositeľnosť. Táto postupnosť je typicky realizovaná ako reťazec. Ak nebude dostatočne spoľahlivý, program sa môže zrútiť alebo vytvoriť zápis do logovacieho súboru. Útok nebude úspešný a šanca na jeho detekciu, alebo dokonca na odhalenie útočníkovej aktivity, sa zväčšuje. To môže viesť k ošetreniu danej zraniteľnosti alebo dokonca zamedzeniu útočníkovho prístupu.
- Vzniknutý *shellcode* je možné uložiť buď do nejakého súboru ako premennú prostredia definovanú užívateľom, alebo zaslať ako požiadavok prostredníctvom siete. Pre úspešné prevzatie kontroly nad procesom je nutné *shellcode* umiestniť do pamäti a neskôr spustiť.

Pri jeho vytváraní a úspešnom nasadení je nutné prekonať tieto problémy:

- Problém nulového bytu, angl. *Null byte problem*
  Funkcie pre prácu s reťazcami predpokladajú ukončenie reťazca znakom *NULL*. To má za následok to, že od miesta kde sa nachádza tento znak program nebude interpretovať ďalšie byty. Existuje viacero techník a dobrých návykov, pomocou ktorých sa tomu dá vyhnúť, napríklad použitie menšieho registra (namiesto *EAX* iba *AL* ak to samozrejme má zmysel) alebo použitie inštrukcií pre kopírovanie a mazanie registrov, ktoré v registri nezanechajú nulové byty.
- Problém adresovania, angl. *Adressing problem*
  Nutné staticky zistiť adresu pamäte, kde sa *shellcode* bude nachádzať. Metódy, ktoré sa využívajú pre jeho spustenie, budú vysvetlené v nasledujúcich odstavcoch.

Toto členenie reprezentuje najpoužívanejšie metódy, ktoré sa snažia zmeniť tok programu tak, aby došlo k spusteniu útočníkovho kódu:

- **Priamy skok**
  Jednoduchá a často využívaná technika. Je realizovaná priamym skokom na miesto v pamäti, kde sa nachádza *payload*. *Payload* je kód, ktorý vykonáva akciu útoku. To zabezpečuje tzv. *injection vector*. Je to presne špecifikovaný vstupný reťazec, ktorého účelom je vložiť do programu *payload* a spustiť ho. Ak sa na adrese na ktorú sa má skočiť nachádza *NULL* byte, celý *payload* musí byť umiestnený pred týmto vektorom. V tomto prípade bude ale veľkosť *payloadu* limitovaná. S tým prichádza ďalší problém. Adresa, kde sa nachádza nemusí byť stále na tom istom mieste v pamäti. *Shellcode* sa zvykne špecializovať na konkrétny systém tým, že sa experimentovaním zistí bázová adresa a offset. Týmto spôsobom je výsledný *shellcode* neprenosný, no je presne určené kde sa má skočiť. Táto metóda nie je spoľahlivá, dokonca niekedy nefunguje ani na rovnakom systéme. Ak je útočník schopný určiť adresu len približne, využíva sa tzv. *NOP sled*. *NOP* je inštrukcia ktorá nič nevykonáva a vznikla pôvodne pre účely ladenia alebo kvôli situáciám, ktoré vyžadujú určité oneskorenie. Postupnosť týchto inštrukcií sa umiesňuje pred *shellcode*. Ak sa skočí na miesto v pamäti kde sa *NOP sled* nachádza, postupne dôjde k vykonávaniu samotného *shellcodu*. Výhodou tejto metódy je zvýšenie šance na úspešné spustenie kódu, nevýhodou je výsledná veľkosť.

- **Blind return**
  Register *ESP* obsahuje hodnotu na vrchole zásobníka. Inštrukcia, ktorá vracia riadenie nadradenej funkcii (*RET*) spôsobí, že sa do registru, ktorý slúži ako ukazateľ na ďalšiu inštrukciu (*EIP*), vloží hodnota z *ESP*. Ak útočník dokáže zneužiť tento princíp vo svoj prospech, môže dôjsť k spusteniu *shellcodu*.

- **Pop Return**
  Útočník postupne vyberá hodnoty zo zásobníka až pokiaľ nie je dosiahnutá tá správna, teda adresa *shellcodu*. To je možné dosiahnuť iba vtedy, ak sa požadovaná adresa nachádza blízko vrcholu zásobníka. Pozmenená hodnota v registri *EIP* ukazuje na postupnosť inštrukcií *POP*, slúžiacich pre odobranie hodnoty z vrcholu zásobníku. Táto postupnosť je nasledovaná inštrukciou *RET*.

  Okrem nich sa pri vytváraní a spúšťaní využívajú aj iné metódy, pri *buffer overflow* útokoch na operačný systém Windows napríklad *Call register*. Tento systém obsahuje mnoho príkazov na fixných adresách, napríklad v knižnici *Kernel32.dll*. To môže útočník pri tvorbe *shellcodu* využiť. Nie vždy je tento kód prenositeľný, dokonca ani medzi rôznymi verziami tohto systému najmä kvôli možným odlišnostiam v adresách a knižniciach.

Existuje niekoľko druhov *shellcodu*. Mnohé využívajú podobné metódy, no vo všeobecnosti sa delia na tieto kategórie:

- **Generický shellcode**
  Najbežnejší druh, ktorý typicky spúšťa *shell*, pre prístup k cieľovému zariadeniu. Jeden z prvých článkov[7], ktorý ho vysvetľoval, bol uvedený v magazíne *Phrack*. Jeho autorom bol *Aleph One*, podľa ktorého sa tento druh niekedy zvykne aj označovať. Využíva funkciu *execve()*, ktorá sa nachádza v jadre Linuxu. Útočníci zvyknú pre svoje potreby spustiť práva správcu. Využívajú pri tom čísla *UID* a *EUID* a systémového volania *setuid()*.

- **Prenositeľný shellcode**
  Do tejto kategórie patria tie, ktoré sú nezávislé na operačnom systéme a procesorovej architektúre. To je realizované pomocou úvodného reťazca, ktorý obsahuje postupnosť takých bajtov, že pre niektoré operačné systémy alebo architektúry majú význam skoku na požadované miesto v pamäti, pre niektoré sú to len nepodstatné inštrukcie. Nie vždy je nutné použiť takýto reťazec, napríklad pri identifikácii cieľového operačného systému stačí zistiť obsah segmentových registrov *FS* a *GS*, prípadne iných poznávacích znakov. Čím je *shellcode* prenositeľnejší, tým je väčší a nespoľahlivejší, naviac detekcia takéhoto druhu je jednoduchá, napríklad pomocou filtrov. Jeho výhodou je hlavne univerzálnosť.

- **Špeciálne druhy shellcodu**
  Vznikli kvôli potrebe obísť detekciu a zvýšiť pravdepodobnosť úspechu útokov. Pri ich tvorbe sa používajú inštrukcie, ktoré majú ako hexadecimálnu hodnotu operačného kódu alfanumerické znaky, a teda majú tvar bežného textu. Detekcia založená na princípe filtrovania je preto často nedostatočná.
  Polymorfný shellcode patrí do kategórie najťažšie detekovateľných napríklad pomocou systémov pre detekciu vniknutia (*IDS* na základe signatúr, ktorý bude vysvetlený v kapitole 2.2.3). Samotný polymorfizmus je realizovaný zakódovaním kódu, ktorý vykonáva požadovanú činnosť. Dekodér je umiestnený typicky (no nie vždy) pred tento kód a po skončení vykonávania dôjde ku skoku na *shellcode*. Možné spôsoby zakódovania bývajú pomocou pseudo-náhodných čísel, alebo vložením zbytočných operácií.

  Ako nedostatok tohto druhu je ten, že má niekedy príliš veľkú veľkosť, čo má za následok zvýšenie pravdepodobnosti jeho detekovania alebo zlyhania, ale naopak ak bude vytvorený efektívne, situácia sa mení. Existuje možnosť rozdeliť dekóder na čo najviac častí, ktoré budú neskôr prepojené do funkčného celku.

  Ďalšie sieťové útoky využívajú napríklad *Bindshell*, ktorý vloží kód pomocou aktívneho sieťového pripojenia. Tieto typy útokov nemusia byť vždy úspešné už len kvôli tomu, že na napadnutom systéme môže byť *firewall* s politikou implicitného odmietnutia a pripojenie na *shell* nebude možné. Napadnutie systému iným, už kompromitovaným systémom, môže tejto ochrane predísť. *Shellcode* generuje spojenie z napadnutého systému.

  Posledná kategória sa zameriava na opätovné využitie už definovaných premenných v programe. Takýto *shellcode* má minimálnu veľkosť. Využíva sa hlavne v programoch typu *open-source*, pri ktorých sú zdrojové súbory dostupné a je možné zistiť, kde sa daná premenná používa. V *closed-source* programoch je možnosť využiť nástroje reverzného inžinierstva a zo spustiteľného formátu zistiť informácie o použitých segmentoch (hlavne *.data* a *.rodata*, ktoré obsahujú premenné programu). Z nich je možné vypočítať miesto v pamäti, kde sa nachádza potencionálne využiteľná premenná.

Pre pochopenie existujúcich *shellcodov* je možné využiť nástroje ktoré disasemblerujú spustiteľný formát, napríklad nasm (ndisasm), s cieľom získať inštrukcie, z ktorých vznikol. Následne sa zistia aké systémové volania a s akými parametrami boli použité. Túto problematiku popisuje projekt *LSD*[9] s mnohými ukážkami. K celosvetovo najväčším verejným databázam patrí napríklad skupina Offensive Security [5], ktorá v súčasnosti obsahuje vyše 33 000 exploitov, teda programov využívajúcich nejakých zraniteľností. Pre zaujímavosť, za posledné 4 mesiace ich pribudlo približne 2000, no nie všetky obsahujú spomínaný *shellcode*. Ďalším zdrojom je napríklad komunita SecurityFocus [6], ktorá obsahuje databázu

zraniteľností, alebo rôzne webové stránky[10] ako aj spomínaná komunita Phrack.

2.3 Detekcia sieťových útokov

Úloha systémov na detekciu vniknutia (*IDS*) je pokúsiť sa zachytiť prítomnosť útoku na kompromitovanej sieti. Naviac si zachytené útoky zapamätať a pokúsiť sa im predísť v budúcnosti. Tieto systémy sa podľa [9] v princípe delia na 2 kategórie: detekcia na základe anomálií a detekcia na základe signatúr.

Okrem týchto systémov existujú ešte systémy pre prevenciu proti vniknutiu (*IPS*). Tieto systémy sú veľmi sofistikované. Majú schopnosť odhaliť prítomnosť útočníkov, ich akcií a naviac im zabrániť aby k útoku došlo. Často kombinujú správanie *firewallu* a *IDS*. Kvôli tomu je u nich efektivita nižšia než u jednotlivých špecializovanejších systémov ako sú spomínané *IDS*, a kedže je rýchlosť väčšinou jedna z najdôležitejších požiadaviek, ich použitie je často nevhodné.

Pri týchto systémoch nastávajú 4 scenáre ako môže detekcia skončiť:

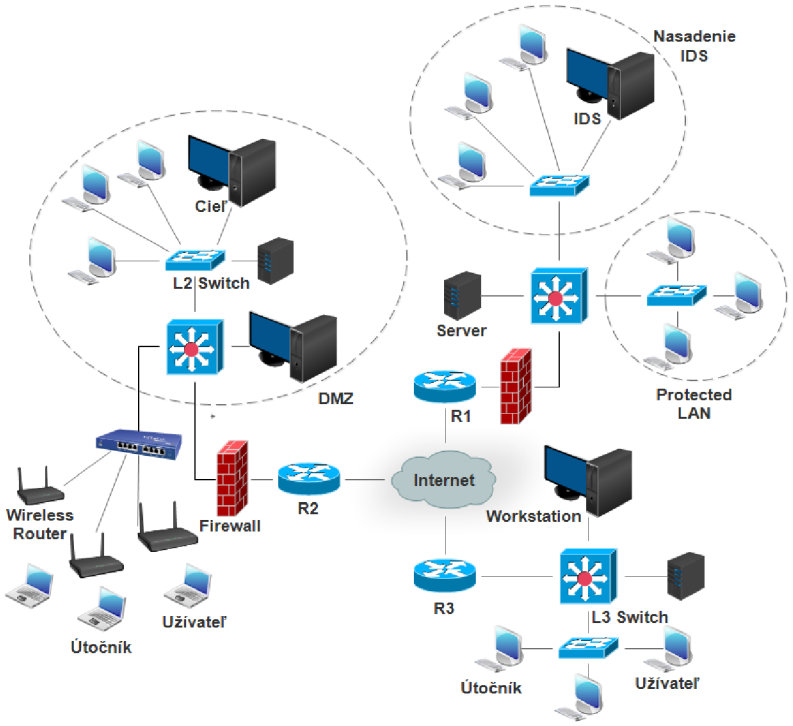
- *True positive* - Ak dôjde k útoku, a *IDS* spustí patričný alarm. Často je ďalej vyžadovaná akcia administrátora. Ak nie je, znamená to že je nastavené nejaké správanie, ktoré sa bude ďalej vykonávať.
- *True negatives* - Normálne prebiehajúca aktivita podľa očakávaní. Pri sieťovej prevádzke bez útoku systém nespustí alarm.
- *False positives* - Typicky k tomu dochádza ak *IDS* chybne rozpozná, že sa jednalo o útok, no k žiadnemu nedošlo.
- *False negative* - Ak k útoku došlo, ale systém ho nezachytil. Pokiaľ k tomuto scenáru dochádza často, systém nie je dostatočne kvalitný pri detekcií.

Odozva, aká sa uplatňuje keď tieto systémy spustia alarm, môže byť rôzna. Závisí od typu útoku a typu alarmu. Väčšinou sa jedná o zablokovanie útočníkovej IP adresy, zrušenie spojenia alebo získanie dodatočných informácií, napríklad pre analyzovanie útoku. Typická architektúra siete, v ktorej je prítomný *IDS*, zabezpečená sieť *LAN* a demilitarizovaná zóna (*DMZ*) je zobrazená na obrázku 2.3. Demilitarizovaná zóna je fyzická alebo logická podsieť, ktorá obsahuje služby nejakej väčšej nedôveryhodnej sieti, najčastejšie internetu. Jej účelom je pridanie ďalšej bezpečnostnej vrstvy v *LAN*. Útočník získa prístup iba k zariadeniu nachádzajúcom sa v *DMZ*, ale zvyšok lokálnej siete ostáva zabezpečený. Ako je zobrazené na tomto obrázku, útočník môže spustiť útok zo zariadení, ktoré sa pripájajú na sieť rôznymi prístupmi. Príkladom systému *IDS* je Snort. Jedná sa o známy nástroj, ktorý funguje multiplatformovo a dokáže detekovať širokú škálu anomálií v sieťovej prevádzke na TCP/IP spojeniach. Pre detekovanie sieťových anomálií používa pravidlá a algoritmy pre porovnávanie. Dokáže skúmať užitočné dáta paketov a sleduje špecifické znaky sieťovej prevádzky na aplikačnej vrstve. Informácie, ktoré o tejto sieťovej prevádzke zozbiera, poskytuje užívateľovi v podobe výstupu.

Nasledujúce podsekcie budú popisovať princípy systémov pre detekciu vniknutia. V časti 2.3.3 bude vysvetlený pojem *honeypot*.

[9] Last stage of Delirium. http://lsd-pl.net

[10] Webová stránka obsahujúca kvalitné *shellcody*. Patrí pod skupinu Security Site Preservation Group (SSPG). http://safemode.org



Obr. 2.3: Typická architektúra siete so zabezpečenou LAN, DMZ a nasadením IDS.

2.3.1 Detekcia na základe anomálií

Pri týchto systémoch sa uplatňuje politika, že všetko čo je abnormálne je podozrivé. Konštrukcia takýchto detektorov si vyžaduje stanoviť aké je normálne správanie sledovaného objektu a aké je abnormálne, čo je často problém. Tento prístup si vyžaduje natrénovať klasifikátor, mať k dispozícii dáta, a je netriviálne určiť, ktoré sieťové spojenia sú abnormálne, pokiaľ sa dáta zozbierali z reálnej prevádzky. Manuálne označovanie toho, čo je abnormálne a čo nie, je pri veľ kých objemoch dát takmer nemožné. Automatické označovanie nemusí byť vždy dokonalé a prináša určité riziká výskytu chyby. S tým sa zvyšuje aj miera *false positive*, čo je jedna z najväčších nevýhod pri ich nasadení. Tieto nevýhody čiastočne prekonáva použitie verejne dostupných dátových kolekcií, ako napríklad kolekcia KDD Cup, ktorá bola zozbieraná na skutočnej sieťovej prevádzke. Výhoda pri ich nasadení je tá, že dokážu byť veľmi užitočné pri vytváraní a modifikovaní signatúr útokov.

- **Samoučiace sa systémy**
  Zdokonalujú sa dlhodobým pozorovaním sieťovej prevádzky a postupne budujú jej model. Môžu byť:

  – Časovo neopakovateľné
  Kolektívny pojem pre detektory, ktoré modelujú normálne správanie využitím stochastického modelu. Môžu fungovať tak, že systém sám sleduje prevádzku a formuluje pravidlá, ktoré popisujú normálne správanie systému, resp. komunikácie. V detekčnom móde systém aplikuje pravidlá a sleduje, či sa dodržiavajú. Porovnávanie funguje na princípe váhovania, a ak nastane slabá zhoda tak systém spustí alarm.

Iné systémy v tejto kategórií zbierajú jednoduché štatistiky sieťovej prevádzky do profilu. Následne vytvoria vektor vzdialeností pre pozorovanú sieťovú prevádzku a profil. Ak je vzdialenosť dostatočne veľká, systém spustí alarm.

  - Časovo opakovateľné
    Tento druh modelov má komplexnejšiu povahu. Využívajú pri svojej činnosti Markovské rozhodovacie procesy alebo neurónové siete.

- **Naprogramované systémy**
  Pri účení systému detekovať určité abnormálne udalosti je potrebný programátor, ktorý formuluje čo je považované za dostatočne abnormálne na to, aby systém signalizoval porušenie bezpečnosti. Tieto systémy sa delia na:

  - Systémy zbierajúce deskriptívne štatistiky o rôznych parametroch, ako je napríklad počet neúspešných prihlásení, počet sieťových pripojení, alebo počet príkazov, ktoré vrátili chybnú návratovú hodnotu.
  - Systémy s politikou implicitného odmietnutia. Myšlienka týchto systémov je explicitne stanoviť podmienky, za ktorých pozorovaný systém má pracovať normálne a označiť všetky odchýlky ako útoky. Tento prístup korešponduje s implicitne zakazovacou bezpečnostnou politikou, ktorá definuje čo všetko je prípustné. Všetko ostatné potom označí ako zakázané.

2.3.2 Detekcia na základe signatúr

Tento druh detekcie funguje na báze modelu škodlivého procesu a stôp, ktoré môžu byť zanechané na pozorovacom systéme. Môže byť definované čo tvorí legálne a ilegálne správanie a priamo porovnať pozorované správanie. Tieto detektory sa snažia detekovať prítomnosť škodlivej aktivity bez ohľadu na normálnu komunikáciu prebiehajúcu v pozadí. Tieto systémy nemajú žiadne samoučiace sa časti. Mali by byť schopné fungovať bez ohľadu na to, čo tvorí normálne správanie systému, namiesto toho vyhľadávajú určité známe signatúry, ktoré sú napríklad uložené v databáze. Nedostatkom tohto prístupu je to, že nemusí byť vždy efektívny, pretože prehľadávanie celej databázy nie je optimálny proces. Naviac nie je možné odhaliť *Zero-Day*, alebo rôzne polymorfné alebo obfuskované útoky. Ich prednosťou je to, že pre známe útoky dochádza k vysokej miere úspešnosti detekcie. Delia sa na 4 kategórie:

- **Stavovo modelované systémy**
  Útok je reprezentovaný ako množina rozdielnych stavov, pričom každý z nich musí byť aktivovaný pre úspešné detekovanie útoku. Svojou povahou sa jedná o modely časových sérií. Existujú 2 podtriedy:

  - Stavovo prechodové, pričom spomínané stavy tvoria jednoduchú postupnosť, ktorá musí byť prejdená od začiatku po koniec.
  - Založené na Petriho sieťach. Môže mať obecnejšiu stromovú štruktúru s niekoľkými prípravnými stavmi, ktoré môžu byť splnené v akomkoľvek poradí.

- **Expertné systémy**
  Používajú stanovené pravidlá popisujúce správanie útokov. Často fungujú na princípe za sebou idúcich pravidiel, ktoré musia byť splnené pre úspešné detekovanie útoku. Zvyčajne sú celkom úspešné a flexibilné, užívateľom sú k dispozícií mechanizmy ako

unifikácia. To je často za cenu rýchlosti, preto sa tieto systémy môžu vyskytovať v zjednodušenej forme.

- **Systémy porovnávajúce zhodu reťazcov**
  Principiálne jednoduché, pracujú zvyčajne case-sensitive a často v texte porovnávajú podreťazce. Text je prenášaný medzi systémami. Táto metóda nie je flexibilná, ale je jednoduchá na pochopenie. Mnoho efektívnych algoritmov funguje na tomto princípe.

- **Systémy založené na jednoduchých pravidlách**
  Tieto systémy sú podobné expertným systémom, no sú zjednodušené kvôli rýchlejšiemu vyhodnoteniu.

2.3.3 Detekcia s využitím honeypotov

Rozdielny a modernejší prístup pre detekciu vniknutia a odozvy. Tieto systémy fungujú ako návnada, pričom monitorujú sieťovú prevádzku a snažia sa čo najskôr indikovať, že došlo k útoku. Primárne vznikli pre výskumné alebo produkčné účely. Pre výskumné účely *honeypoty* zhromažďujú informácie o nových a nebezpečných hrozbách a o nových trendoch v útokoch. Kategória produkčných *honeypotov* slúži na prevenciu, detekciu a odozvu k útokom. *Honeypoty* majú vo všeobecnosti mnoho výhod, predovšetkým v tom, že sú koncipované tak aby dokázali zachytiť nové, neznáme útoky, polymorfné alebo zašifrované dáta, a naviac dokázali zachytiť informácie o správaní, úmysloch a identite útočníka. Detailnejšie výhody a nedostatky budú diskutované pri konkrétnych typoch týchto systémov. Podľa [18] sa delia na:

- **Fyzické honeypoty**, ktoré bežia na skutočnom, fyzickom zariadení. Zároveň sa väčšinou jedná o systémy s vysokou interakciou. Medzi nevýhody patrí to, že môžu byť kompletne kompromitované, typicky sú náročné na inštaláciu a údržbu. Ich rozmiestnenie nad väčším adresným priestorom je veľmi nepraktické až nemožné, pretože by ich muselo byť mnoho, pre veľký počet IP adries.

- **Virtuálne honeypoty**, ktorých hlavná výhoda spočíva v použití mnohých systémov na jednom zariadení a nenáročnej údržbe. Obsahujú viac systémov, systémových alebo iných služieb, a dokážu pokryť väčšiu škálu útokov. Sú simulované zariadením, ktoré reaguje na sieťovú prevádzku tak, že ju zasiela virtuálnemu *honeypotu*. Príkladom často používaného virtuálneho *honeypotu* je Argos[11]. Útoky, ktoré zachytí, sa ukladajú a neskôr ich je možné dôkladne analyzovať.

Podľa ďalšej kategorizácií sa delia na nízko interaktívne a vysoko interaktívne, pričom pod interakciou sa v tomto prípade rozumie úroveň aktivity útočníka s *honeypotom*. Podrobnejšie sa nimi budú zaoberať nasledujúce sekcie.

Nízko interaktívne honeypoty

Tento druh systémov poskytuje emuláciu operačného systému a systémových služieb. Aktivity útočníka sú limitované nízkou úrovňou emulácie, ktorú *honeypot* poskytuje. Automaticky zbierajú dáta o sieťovej prevádzke. Tieto dáta môžu byť použité pre rôzne štatistiky alebo získanie informácií o charaktere útokov.

[11] Monitorovací systém navrhnutý za účelom identifikovať škodlivú aktivitu prichádzajúcu z internetu. http://www.techopedia.com/definition/10278/honeypot

Výhody, ktoré prináša použitie týchto systémov, sú hlavne tie, že sa riziko ich kompromitovania, vďaka nízkej úrovni emulácie, znižuje. Sú ťažšie na detekciu, pretože simulujú limitované množstvo služieb. Útočník si teda myslí, že je pripojený na skutočný systém. Ďalšia výhoda je v ich jednoduchosti a nenáročnej údržbe. Medzi ich hlavným nedostatkom je to, že nie sú vhodné pre detekciu *Zero-day* útokov.

Vysoko interaktívne honeypoty

Tieto systémy sú komplexnejšie, môžu zachytiť mnoho informácií o útočníkovi a jeho správaní. Bežia na skutočnom operačnom systéme, a preto je viac prístupný ku kompromitovaniu a následnému použitiu k útokom na iné zariadenia v sieti. Nevyužívajú žiadnu emuláciu, útočník dokáže interagovať a napadnúť skutočný systém a jeho služby.
Vysoko interaktívne honeypoty nemajú žiadnych aktívnych užívateľov, nemali by mať žiadne nezvyčajné procesy ani generovať sieťovú prevádzku okrem pravidelných *daemonov* alebo bežiacich systémových služieb. Každá interakcia prostredníctvom siete s týmito systémami je podozrivá a môže viesť k útoku. Kvôli tomu je sieťová prevádzka monitorovaná a ukladaná do logovacích súborov. Systémová aktivita sa zaznamenáva tiež, kvôli neskoršej analýze.

Ich prednosti spočívajú v získavaní dát o rôznych druhoch útokoch naraz. Vďaka nim je možné zistiť podrobné údaje o útočníkovi, aké techniky použil pre vyhľadanie zraniteľností o cieľovom systéme alebo napríklad spôsob akým zaútočil, aké nástroje použil, jeho komunikácia s inými ľuďmi, alebo aké kroky podnikol pre odstránenie svojich stôp. V porovnaní so systémami *IDS* majú absenciu miery *false positive* a to je jedna z ich najväčších výhod. Tento druh *honeypotov*, ako fyzické tak aj virtuálne, obsahuje aj radu nevýhod. Útočník je schopný získať úplný prístup k systému, dokáže zistiť či napadol virtuálny alebo fyzický *honeypot*, čo môže mať za následok zmenu stratégie útoku. Následne môže zaútočiť tak, aby jeho aktivita nebola detekovaná. V konečnom dôsledku o ňom *honeypot* neposkytne žiadne podrobné informácie. Tiež je nutné dodať, že sú náročnejšie na vývoj a údržbu než nízko interaktívne *honeypoty*.

Kapitola 3

Návrh metrík pre detekciu útokov nad UDP

Navrhnuté metriky budú reprezentovať určité vlastnosti sieťovej komunikácie. Získavajú sa zo sieťovej prevádzky, predovšetkým z hlavičiek prijatých alebo odosielaných paketov. Sada navrhnutých metrík sa viaže k UDP komunikácií so zameraním na útoky typu *buffer overflow*. Čo sa týka TCP komunikácie, metriky pre klasifikáciu sieťových útokov boli navrhnuté v práci [14] a [17]. Niektoré metriky, ktoré boli navrhnuté v týchto prácach, budú do tejto kolekcie zahrnuté. Pred zahájením TCP komunikácie dochádza k jeho nadviazaniu pomocou tzv. *3-way handshake*, ktoré iniciuje spojenie. Po úspešnom nadviazaní komunikácie sa začne samotný prenos dát, ktorý prebieha na rovnakých portoch a IP adresách.

Pri UDP komunikácií nedochádza k počiatočnej ani záverečnej fáze spojenia a jednotlivé pakety nie sú potvrdzované. Pakety môžu dôjsť v inom poradí než boli odoslané alebo sa dokonca nemusia doručiť. Hlavička UDP paketov je tiež jednoduchšia než pri spojení TCP. Počas odosielania UDP paketov môže dôjsť k zmene portu alebo IP adresy a môže nastať problém zatriediť jednotlivé pakety do prislúchajúcej komunikácie. Pri návrhu sady metrík ako aj samotného nástroja treba tieto problémy brať do úvahy.

3.1 Kritické UDP služby

Táto kapitola popisuje služby využívajúce porty UDP[12], ktoré sú predmetom záujmu z oblasti sieťovej bezpečnosti. Pri ich popise boli využívané štandardy RFC. Návrh kolekcie metrík bude vychádzať aj z hlavných charakteristík týchto služieb.
Služby, ktoré sú často predmetom útokov:

- DNS, port 53. Jeho hlavnou úlohou je preklad doménových mien na IP adresy uzlov siete a naopak. Pri útokoch na túto službu sa využívajú zraniteľnosti[13] nachádzajúce sa v resolveroch. Resolver je program, ktorý zaisťuje samotný preklad.

- DHCP, porty 67 a 68 pre klienta a pre server. Tento protokol nahradzuje starší protokol BOOTP. Používa sa pre automatickú konfiguráciu počítačov pripojených k sieti zasielaním konfiguračných parametrov.

[12] Čísla portov, na ktorých bežia kritické služby boli vyberané z www.iana.org/assignments/port-numbers.

[13] *BO* zraniteľnosti v DNS. http://www.cert.org/historical/advisories/ca-2002-19.cfm?

- TFTP, port 69. Nezabezpečený protokol, používa sa pre prenos súborov. Typicky sa používa tam, kde nehrozí nebezpečenstvo poškodenia dát alebo iného útoku. Útoky typu *buffer overflow* využívajú napríklad zasielanie požiadavku s veľkým názvom súboru[14] ktoré môže mať za následok reštart alebo zlyhanie cieľového zariadenia.

- RPC, port 111. Technológia umožňujúca vykonávanie procedúr vzdialene, prostredníctvom siete. Serializuje dáta pomocou formátu *XDR*. Pri útokoch na túto službu sa využíva zaslanie veľkého množstva *XDR* elementov s cieľom vykonať nejaký kód s právami *roota* na cieľovom zariadení[15].

- NTP, port 123. Je to protokol pre synchronizáciu vnútorných hodín počítača po paketovej sieti s vnútorným oneskorením. Útoky na túto službu môžu mať za následok napríklad zlyhanie NTP *daemona* alebo spustenie útočníkovho kódu.

- Microsoft EPMAP (End Point Mapper), port 135. Funguje ako RPC, je známy pre vzdialené ovládanie služieb DHCP server alebo DNS server.

- NetBIOS, porty 137 a 138. Poskytuje služby týkajúce sa relačnej vrstvy a dovoľuje aplikáciám bežiacich na vzdialených počítačoch navzájom medzi sebou komunikovať a sprístupniť dáta.

- SNMP, port 161 a 162. Pomocou neho je možné zbierať rôzne dáta a vyhodnocovať ich v podobe štatistík, ktoré sa používajú pri prostriedkoch a nástrojoch pre správu a monitorovanie siete. Úspešný útok môže viesť ku kompletnému kompromitovaniu cieľového zariadenia.

- ESRO, port 259. Tento protokol je navrhnutý špeciálne pre použitie v bezdrôtových sieťach. Poskytuje efektívny mechanizmus pre realizáciu vzdialeného volania procedúr. Jeho model je principiálne podobný ako model služby RPC.

- SYSLOG, port 514. Jedná sa o štandard pre záznam programových správ. Umožňuje oddeliť systém, ktorý generuje správy od systému, ktorý ich ukladá a systému, ktorý ich analyzuje a poskytuje hlásenia. Môže byť použitý napríklad pre bezpečnostný audit ako zdroj analýzy. Knižnice, ktoré využíva, používajú internú rýchlu vyrovnávaciu pamäť pre vytváranie správ, ktoré sú zasielané *daemon* (SYSLOGD). Je možné zaslať taký reťazec, ktorý bude mať za následok pretečenie tejto pamäte a kompromitovanie cieľového zariadenia.

3.2 Kolekcia navrhnutých metrík

Samotné metriky vychádzajú z hodnôt nachádzajúcich sa v hlavičkách jednotlivých vrstiev, ďalej z veľkostí a časov príchodov paketov. Pakety budú agregované do množiny spojení a nad každým bude prebiehať extrakcia metrík. Metriky označené suffixom *Op* označujú opačný smer k danému spojeniu. Metriky pracujúce s nadväzujúcim spojením majú suffix *Conseq*. Pod nadväzujúcim spojením sa myslí také, ktoré prebieha na rovnakom kontexte, teda má rovnaké MAC a IP adresy, a má istú časovú súvislosť s daným spojením.

[14] *BO* v TFTP http://www.iss.net/security_center/reference/vuln/cisco-tftp-filename-bo.htm.
[15] *BO* v RPC http://www.iss.net/security_center/reference/vuln/sunrpc-xdr-array-bo.htm.

- **Metriky získané extrahovaním dát z hlavičiek paketov**

  Dáta sú získavané z hlavičiek ethernetovej, sieťovej a transportnej vrstvy. MAC a IP adresy neboli v laboratórnych podmienkach využívané kvôli tomu, že sa príliš nemenili, viď 6.2. Taktiež zdrojový port nie je pri týchto podmienkach relevantný. Metriky sú podrobnejšie popísané v prílohách k tejto práci, viď tabuľky v časti A.1.

- **Metriky získané extrahovaním dát z počtov a veľkostí paketov**

  Zoznam metrík reprezentujúcich túto kategóriu slúži pre bližšie informácie o počte prenesených paketov v rámci daného spojenia a o veľkostiach prenášaných dát (*packet payload*) v týchto paketoch. Všetky metriky navrhnuté v tejto kategórií sú popísané v A.2.

- **Metriky získané extrahovaním dát z fragmentácií paketov**

  Pri fragmentovaných paketoch nie je vždy jednoznačné určiť do ktorého spojenia patrili. Fragmentácia prebieha na úrovni sieťovej vrstvy a pri tomto procese nemusí byť prítomná hlavička transportnej vrstvy, ktorá obsahuje napríklad porty, na ktorých prebiehalo spojenie. Tieto metriky dodávajú základné informácie o fragmentácií a fragmentovaných paketoch, ale môžu prinášať určité znepresnenie, preto je ich navrhnutých iba niekoľko, ako je možné vidieť v A.3.

- **Metriky získané extrahovaním dát z časových priebehov prenosov**

  Časový priebeh *buffer overflow* útokov je veľmi dôležitým subjektom analýzy kvôli tomu, že komunikácia s útočníkom nemusí prebiehať rovnomerne. Môže mať zachovanú určitú podobnosť a kvôli tomu boli navrhnuté metriky popísané v A.4.

- **Metriky získané extrahovaním dát z časových úsekov spojenia**

  Navrhnutá skupina metrík popísaná v A.5 reprezentuje extrakciu dát nad 4 časovými úsekmi daného spojenia. V tejto práci budú označované ako kvartily (s prefixom q1–q4), ktorých účelom je poskytnúť presnejšie informácie o priebehu toku dát v rámci daného spojenia v čase.

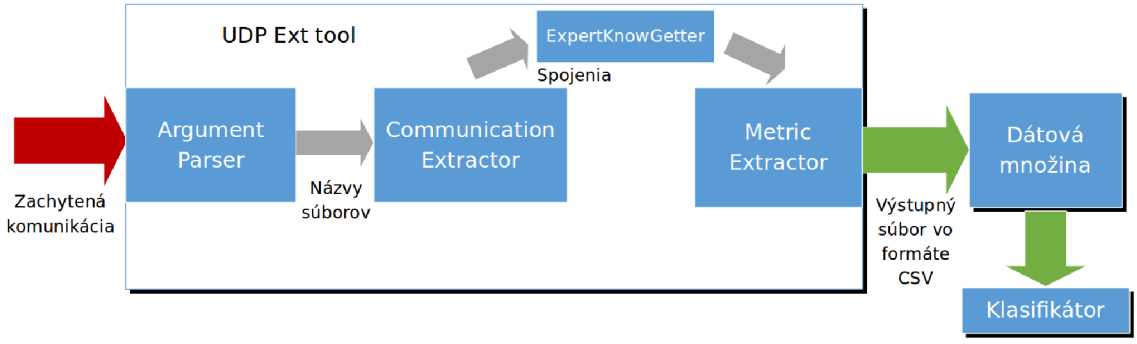
- **Metriky získané extrahovaním dát z nadväzujúcich spojení**

  Posledná kategória metrík slúži pre bližšiu predstavu toho, čo sa deje mimo časového intervalu daného spojenia. Skúma sa počet spojení, ktoré prebiehali pred začatím alebo po ukončení daného spojenia, ale vždy iba na danom kontexte. Pod tým sa myslia spojenia prebiehajúce na rovnakých IP, MAC adresách a určitých časových súvislostiach.
  Pre útoky typu *buffer overflow* je časté to, že si útočník otvorí zadné vrátka, angl. *backdoor*, pre ďalšiu komunikáciu na inom porte. Pri extrakcií metrík uvedených v tejto podsekcii sa tento fakt berie do úvahy skúmaním nasledujúceho príbuzného spojenia s určitým maximálnym časovým odstupom, ktorý bude uvedený v prislúchajúcich metrikách a bližšie popísaný v kapitole 6.2. V prípade prvých dvoch metrík sa uvažujú TCP a ICMP pakety. Celá kategória týchto metrík je popísaná v tabuľkách v časti A.6.

Kapitola 4

Návrh nástroja na extrakciu metrík

V tejto kapitole bude popísaný návrh nástroja, ktorý bude pre každé UDP spojenie extrahovať kolekciu metrík popísanú v kapitole 3. Nástroj prijíma zachytenú komunikáciu zo súborov získaných pomocou nástrojov ako TCPdump alebo Wireshark, pričom sa skúmajú iba spojenia typu UDP. Hlavný dôraz je kladený na útoky typu *buffer overflow*, ktoré boli vysvetľované v druhej časti kapitoly 2. Na obrázku 4.1 je znázornená architektúra celého systému.



Obr. 4.1: Schéma celého systému zobrazujúca moduly najvyššej úrovne.

4.1 Kompozícia systému

Táto sekcia sa zaoberá stručným popisom jednotlivých modulov na najvyššej úrovni. Celý systém sa skladá z troch hlavných častí a modulu pre získanie expertnej znalosti:

- *ArgParser* - spracúva požiadavky od užívateľa v podobe argumentov príkazového riadku. Jeho činnosť zahŕňa predovšetkým získanie názvov vstupných (môže byť 1 alebo aj viac), poprípade výstupných súborov. Užívateľ má možnosť vytlačiť na štandardný výstup nápovedu alebo všetky spojenia v zadaných vstupných súboroch.

- *CommunicationExtractor* - jeho úlohou je agregovať pakety do jednotlivých UDP spojení. Skladá sa z menších modulov, ktoré budú popísané v kapitole 5. Tieto moduly sa volajú postupne tak, aby z množiny vstupných súborov spracovávali dáta z paketov, ktoré obsahujú. Pakety neskôr začlenili do tokov a toky do spojení, pričom tok je množina všetkých paketov, ktoré majú rovnaké IP, MAC adresy a porty. Spojenie vzniká rozdelením toku do časových úsekov, no čas príchodov medzi paketami nesmie prekročiť 30 sekúnd, inak sa jedná o ďalšie spojenie. V prípade, že dané spojenie prebiehalo obojsmerne, táto doba sa zvyšuje na 180 sekúnd.

- *ExpertKnowGetter* - prijíma spojenia získané modulom *CommunicationExtractor* a vykoná nad nimi vyhodnotenie expertnej znalosti. Takéto spojenia sú potom predané ďalej modulu pre extrakciu metrík.

- *MetricExtractor* - vykonáva nad každým UDP spojením získaným z prvého modulu extrakciu navrhnutej sady metrík, ktorá bola popísaná v kapitole 3. Výsledok, ktorý vygeneruje, ukladá do výstupného súboru vo formáte DSV, čo je skratka pre *Delimiter Separated Values*. Je to všeobecný pojem pre súbory obsahujúce dáta, ktoré sú medzi sebou oddelené nejakým špeciálnym znakom, nazývaným ako delimiter. V ďalších kapitolách je možné sa stretnúť s označením CSV, čo je konkrétna špecifikácia DSV súboru, a značí, že ako delimiter bol použitý znak čiarka.

Vygenerovaný výstup sa ďalej využije ako trénovacia a validačná množina dát pre klasifikátor, napríklad RapidMiner [4], za účelom naučiť systém detekovať spomínaný druh útokov čo najpresnejšie.

.

[16] Delimiter separated values. http://c2.com/cgi/wiki?DelimiterSeparatedValues

Kapitola 5

Popis implementácie nástroja

V tejto kapitole bude popísaný princíp implementácie celého nástroja. Kapitola začína tehnológiami, knižnicami a vývojovým prostredím, ktoré boli použité. Ďalej pokračuje popisom implementovaných modulov a ich funkcionalít. Pre ich lepšie znázornenie bol použitý diagram tried.

5.1 Vývojové prostredie a použité technológie

Pri implementácií nástroja bol použitý jazyk Python s verziou interpretu 2.7. Tento interpretovaný jazyk je v súčasnosti veľmi populárny a má mnoho výhod, napríklad:

- podpora pre mnohé operačné systémy, ako pre Linux, Mac OS tak i pre Windows,
- množstvo verejne dostupných, odladených knižníc tretích strán, ako i open-source projektov,
- vývoj aplikácií pri jeho použití je zvyčajne pomerne rýchly, čo platí aj pre prácu s dátovými štruktúrami,
- moduly kritické na rýchlosť alebo prostriedky často implementované v iných jazykoch, ktoré je možné ľahko prepojiť s Pythonom.

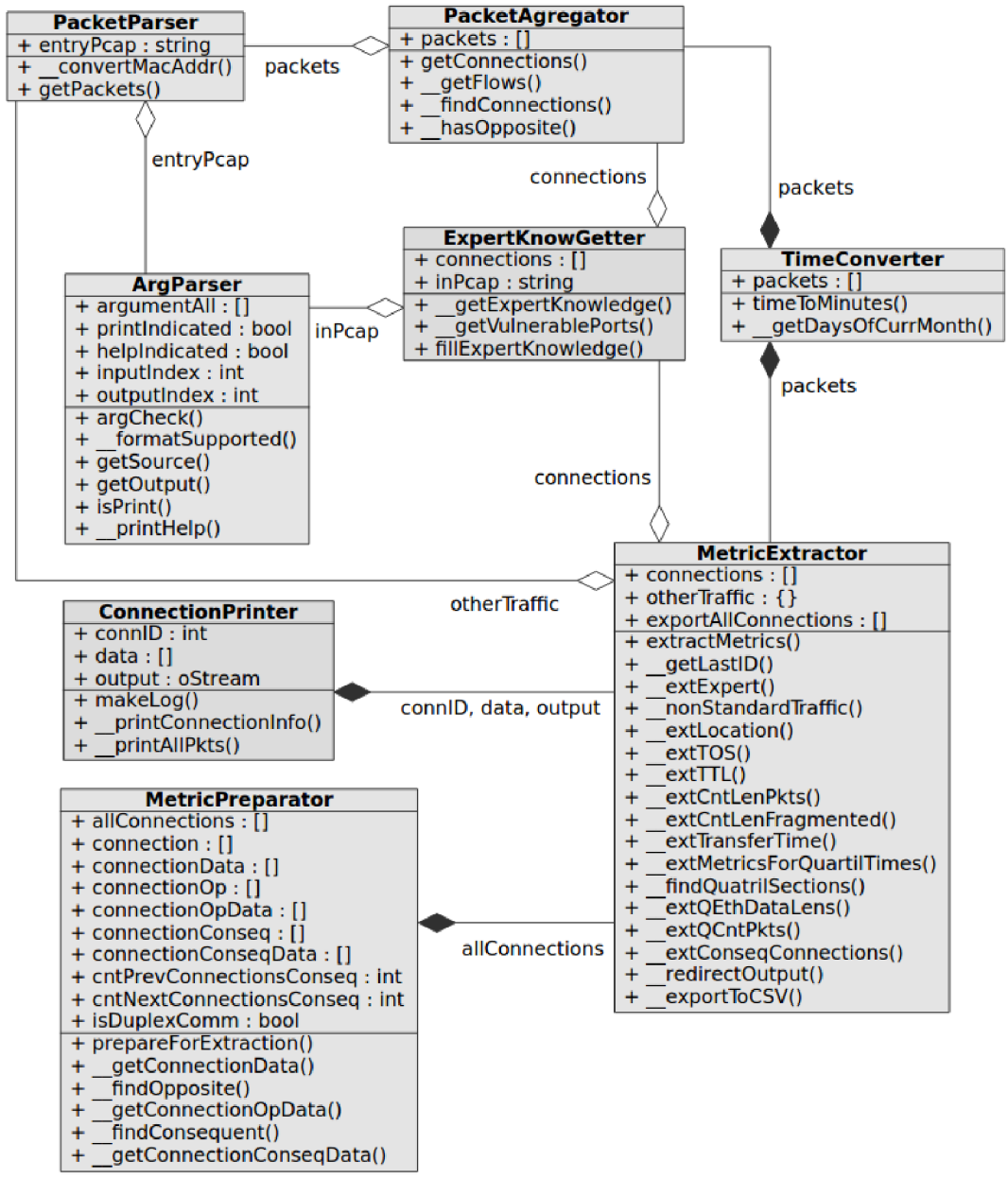
Medzi požiadavkami pre úspešné spustenie a vykonávanie programu je nutné mať nainštalovanú verziu Pythonu minimálne 2.5 a knižnicu *dpkt*[17]. Táto knižnica bola použitá pre spracúvanie paketov zo súborov obsahujúcich zachytenú sieťovú prevádzku. Experimentovaním bolo zistené, že pre dané potreby patrí *dpkt* medzi najrýchlejšie knižnice. V rámci testovania na rýchlosť vykonávania bola porovnávaná s knižnicami scapy, pypcap a pcapy. Počas vývoja bol pre účely zálohovania použitý verzovací systém Git[18]. Testovanie a vývoj prebiehali pod 64-bitovými operačnými systémami Linux Mint 17.1 a Ubuntu 14.10.

5.2 Implementácia jednotlivých modulov

Celá činnosť implementovaného nástroja je vykonávaná spustením skriptu *UDPExt-tool.py*, teda hlavnej časti systému, v ktorom sa, ako bolo znázornené v kapitole 4, postupne volajú tri hlavné moduly: *ArgParser*, *ConnectionExtractor* a *MetricExtractor*. Pre získanie

[17] Knižnica pre vytváranie alebo spracovávanie paketov https://code.google.com/p/dpkt/.
Príklad výsledkov záťažových testov http://libtins.github.io/benchmark/#test3-graphic.
[18] Voľne dostupný distribuovaný systém pre správu revízií http://git-scm.com/.

expertnej znalosti bol implementovaný modul *ExpertKnowGetter*, ktorého inštancia bude vytvorená po úspešnom ukončení procesu vyhľadávania spojení, teda ešte pred samotnou extrakciou metrík.
Diagram tried znázorňujúci implementačnú časť práce je možné vidieť na obrázku 5.1. Popis modulov a ich tried bude detailne vysvetlený v nasledujúcich podkapitolách.



Obr. 5.1: Diagram tried znázorňujúci štruktúru systému.

5.2.1 Modul ArgParser

Obsahuje triedu rovnakého názvu. Táto trieda obsahuje metódy pre spracovanie argumentov príkazového riadku, pre kontrolu vstupných súborov obsahujúcich zachytenú komunikáciu, alebo pre kontrolu formátu výstupného súboru. Tieto činnosti vykonávajú nasledovné metódy:

- *argCheck* - skontroluje správnosť poradia argumentov a výskyt duplicity. Metóda ukladá informácie aj o tom, ktoré argumenty užívateľ zadal, v prípade zadaní výstupného súboru si ukladá index, na ktorom bol jeho názov špecifikovaný.
- *getSource* - ukladá názov vstupného súboru, ktorý obsahuje odchytené pakety. Pri špecifikovaní celého priečinku ako vstupu táto metóda nájde súbory, ktoré sa v ňom nachádzajú. Kontrola vstupných súborov prebieha postupne pomocou metódy *formatSupported*.
- *formatSupported* - obsahuje kontrolu formátu zadaného vstupného súboru. Ak je definovaný vstup ako priečinok, vyhľadá ho a zistí jeho obsah. Potom skontroluje formát všetkých týchto súborov. Povolené formáty sú pcap, cap, pcapng, dmp, alebo ich varianty s číslom na konci.
- *getOutput* - kontroluje formát výstupného súboru a ukladá jeho názov pre budúce použitie pri získaných dátach z procesu extrahovania metrík. Metóda nastavuje cestu, kde sa bude tento súbor nachádzať, konkrétne vždy do zložky *outputs* o jednu úroveň vyššie v hierarchii priečinkov. Ak nebol zadaný parameter pre špecifikovanie názvu výstupného súboru, nastaví názov na *udp_extracted.csv*.
- *isPrint* a *printHelp* - slúžia pre vypísanie nápovedy užívateľovi, ak bol zadaný argument indikujúci, že sa má táto činnosť vykonať.

5.2.2 Modul CommunicationExtractor

Tento modul sa skladá z tried *PacketParser*, *PacketAgregator* a *TimeConverter*. Slúžia pre získanie a uloženie jednotlivých spojení zo vstupnej množiny súborov.

- **PacketParser** ukladá dáta z paketov do slovníka obsahujúceho zoznamy reprezentujúce TCP, UDP a ICMP pakety. UDP pakety budú neskôr použité pre zatriedenie do spojení. Táto trieda obsahuje nasledujúce metódy:
  - *convertMacAddr* - konvertuje reťazec, ktorý obsahuje MAC adresu a bol získaný zo vstupného súboru na čitateľnejšiu formu xx:xx:xx:xx:xx:xx.
  - *getPackets* - spracúva vstupný súbor ukladaním TCP, ICMP a UDP paketov. Z nich si ukladá len relevantné dáta, ktoré vloží do viac rozmerných zoznamov. Tieto dáta, ktoré sa využívajú pri extrahovaní kolekcie metrík, sú získané najmä z hlavičiek jednotlivých vrstiev paketov. Pri výskyte UDP paketu si ukladá väčšie množstvo dát, napríklad ich veľkosti alebo čas príchodov.
- **PacketAgregator** prijíma v konštruktore časť slovníku obsahujúceho zoznam UDP paketov s dátami, ktorý vrátila metóda *getPackets* triedy *PacketParser*. Úlohou tejto triedy je rozdeliť pakety do tokov a tie podľa určitých časových súvislostí ďalej rozdeliť na spojenia. Metódy podieľajúce sa na tejto činnosti:

- *getConnections* - hlavná metóda tejto triedy. Volá ostatné metódy postupne vykonávajúce popísanú činnosť a vracia viac rozmerný zoznam spojení. Každé spojenie obsahuje k nemu prislúchajúce pakety. V tejto metóde sa vytvára inštancia triedy *TimeConverter* za účelom konvertovať čas na číslo, s ktorým sa ďalej bude dať pracovať a vykonávať rôzne výpočty.
- *getFlows* - agreguje pakety do tokov, a to na základe zhody zdrojových aj cieľových MAC adries, IP adries a portov. Berie do úvahy aj fragmentované spojenia. Pri fragmentácii na linkovej úrovni nie je vždy jednoznačne isté do ktorého toku paket patrí, pretože hlavička sieťovej vrstvy, obsahujúca porty, chýba. Ak nastane zhoda u MAC a IP adries, fragmentovaný paket sa takto môže prideliť aj do viacerých tokov naraz. Pre zlepšenie presnosti sa uvažujú časové následnosti, čo rieši nasledujúca metóda.
- *findConnections* - parametrom prijíma zoznam získaných tokov zo vstupného súboru. Táto metóda vytvára spojenia tým, že skúma rozdiel príchodov 2 po sebe idúcich paketov v rámci toku. Paket patrí do daného spojenia pokiaľ prišiel maximálne do 30 sekúnd od príchodu predchádzajúceho spojenia. Ak je komunikácia obojsmerná, toto časové obmedzenie sa zvyšuje na 180 sekúnd. Toto zlepšuje zároveň presnosť zatriedenia fragmentovaných paketov do spojení. Keďže komunikácia typu UDP nie je iniciovaná *3-way handshakom* a ani iným mechanizmom, ktorým by bolo možné určiť presný počiatok a koniec spojenia, pri rozdeľovaní tokov do spojení sa používal iba tento princíp.

- **TimeConverter** prijíma v konštruktore zoznam paketov obsahujúcich čas v podobe reťazca, ktorý uložil program pre odchytávanie sieťovej prevádzky. Tento reťazec bude konvertovaný na reálne číslo, čo vykonávajú metódy:
  - *timeToMinutes* - pre konvertovanie reťazca reprezentujúceho čas príchodu paketu na minúty. Tento reťazec obsahuje dni, hodiny, minúty, sekundy a milisekundy. V prípade UDP paketov konvertuje čas tak, že si ukladá výsledky 2 výpočtov, prvý ako časový rozdiel príchodov 2 po sebe idúcich paketov v rámci daného toku, a druhý ako globálnejší čas v rámci celého vstupného súboru. Prvý čas je použitý najmä pre vyhľadávanie spojení a druhý hlavne pri hľadaní väzieb medzi spojeniami v opačnom smere alebo v naväzujúcich spojeniach.
  - *getDaysOfCurrMonth* - volá sa v prípade, že došlo k zmene mesiaca alebo roka. Metóda vracia maximálny počet dní v aktuálnom mesiaci, no je nutné poznamenať, že reálne čísla spôsobujú určité nepresnosti, ktoré sa môžu časom prejaviť. Toto riziko sa zvyšuje pri vstupnej komunikácii obsahujúcej extrémne dlhý časový priebeh.

5.2.3 Modul ExpertKnowGetter

Modul obsahuje jednu triedu rovnakého názvu, ktorá v konštruktore prijíma názov vstupného súboru spolu s jeho cestou a všetky spojenia, ktoré vrátila metóda *getConnections* triedy *PacketAgregator*. Úlohou tohto modulu je do pripravených položiek v zoznamoch spojení doplniť expertnú znalosť, a predať tieto spojenia ďalej modulu *MetricExtractor*, ktorý v ten moment už bude mať k dispozícii všetko potrebné na to, aby mohol začať pracovať s uloženými dátami.

Základná činnosť, teda uloženie expertnej znalosti do adresárovej štruktúry, bola inšpirovaná skriptom *recurs-walker.sh*, ktorý rekurzívne nad zložkami volal skript *tcpdump_fill.py*.

Tieto zdrojové kódy boli vypracované v rámci projektu [14]. Samotné skripty neboli použité, hlavne kvôli zníženiu optimálnosti programu. Pôvodná myšlienka sa ale z časti zachovala a samotná realizácia bola prispôsobená aktuálnej implementácií nástroja pre extrakciu metrík a integrovaná do hlavného skriptu *UDPExt-tool.py*. Celý priebeh získania expertnej znalosti vykonávajú metódy:

- *getExpertKnowledge* - získava expertnú znalosť z nasledovných zdrojov:
  - adresárová štruktúra. Vstupný súbor, obsahujúci zachytenú komunikáciu, je uložený buď v zložke *Attacks*, vtedy obsahuje aj spojenia reprezentujúce útoky, alebo v zložke *Legit*, ktorá vždy obsahuje iba legitímne komunikácie. Pokiaľ je uložený v inej zložke, expertná znalosť nie je plne vyhodnotená, a dané spojenia sú vyhodnotené ako legitímne.
  - názov vstupného súboru. V rámci experimentov k tejto práci obsahoval vždy minimálne názov primárnej služby na ktorú sa zameriavalo. V prípade že súbor obsahoval veľmi veľa rôznych služieb a nebola primárna žiadna, nachádza sa na tomto mieste názov samotnej dátovej množiny. Ak súbor obsahoval útoky, tak ďalšia položka v názve súboru je aj číslo *CVE-id*, ak bolo známe. *CVE-id* označuje jednoznačnú identifikáciu zraniteľnosti, angl. *Common Vulnerability and Exposures identification*.
  - zoznam zraniteľných cieľových portov. Tie nájde a vráti metóda *getVulnerablePorts*. Následne sa pre každé spojenie porovnávajú. Ak zhoda nenastane, spojenie sa považuje za legitímne. Môže sa jednať napríklad o sieťové procesy, ktoré boli prenášané nezávisle na útokoch. Pravdepodobne sem budú patriť aj *backdoor* spojenia, kedže často prebiehajú na náhodne zvolených portoch. Ak dôjde k ich výskytu, nebudú síce označené ako útoky, no v metódach *findConsequent* a *extConseqConnections* z modulu *MetricExtractor* sa budú hľadať pre ich následnú analýzu.
- *getVulnerablePorts* - ukladá obsah súboru *VulnPorts.csv* obsahujúceho cieľové porty, ktoré boli predmetmi útokov. Získané porty vracia ako zoznam, neskôr použitý pre porovnávanie jednotlivých spojení.
- *fillExpertKnowledge* - pre každé spojenie doplní expertnú znalosť získanú z metódy *getExpertKnowledge*. Súčasťou expertnej znalosti je názov sieťovej služby, typ komunikácie (útok alebo nie) a v prípade že je známa, tak aj *CVE-id*.

5.2.4 Modul MetricExtractor

Tento modul obsahuje triedy *MetricPreparator*, *MetricExtractor* a *ConnectionPrinter*. Ako sám názov napovedá, slúži na extrakciu navrhnutej kolekcie metrík popísaných v kapitole 3. Tieto metriky sa zapíšu do CSV súboru, ktorý sa môže využiť ako vstupná množina dát pre dolovacie nástroje typu RapidMiner. Malá časť toho, ako vyzerá výsledný CSV súbor je možné vidieť v prílohe B, kde je znázornená expertná znalosť a hodnoty prvých metrík. Zoznam po stĺpcoch ďalej pokračuje extrahovanými metrikami. Po riadkoch je zoznam spojení, pričom každý má jedinečné *id*. Súčasťou expertnej znalosti je jednak trieda komunikácie útok je *True* a legitímna komunikácia *False*. *Label* označuje triedu komunikácie, ako prefix je názov služby, sieťovej prevádzky alebo dátovej množiny nasledované číslom označujúce triedu experimentov. V tomto prípade 0 reprezentuje legitímnu komunikáciu a 1 útok.

- **MetricPreparator** predspracováva dáta a metadáta potrebné k extrakcii. Táto trieda vznikla kvôli rýchlejšiemu behu programu, aby sa počas procesu extrahovania nemuseli dáta z paketov a spojení už vyhľadávať a iterácia prebehla iba raz.
  - *prepareForExtraction* - hlavná metóda, ktorá riadi predspracovanie dát volaním ostatných metód.
  - *getConnectionData*, *getConnectionOpData* a *getConnectionConseqData* - metódy, ktoré z paketov daného spojenia získavajú dáta potrebné k predspracovaniu. Tie ukladajú do viac rozmerných zoznamov. Každá metóda má jemne odlišnú implementáciu, pretože pre dané, pre opačné, alebo pre nadväzujúce spojenie boli definované iné metriky a v rámci zrýchlenia behu programu sú ukladané iba tie dáta, ktoré neskôr nutne budú potrebné.
  - *findOpposite* - hľadá spojenie v opačnom smere k danej komunikácii. Porovnáva MAC, IP adresy a porty. Čas začiatku komunikácie na danom a opačnom spojení sa musí líšiť maximálne o 30 alebo 180 sekúnd, čo o koncovom čase neplatí. Ten môže byť ľubovoľne veľký, kvôli presnejším a úplným informáciám o spojení v opačnom smere.
  - *findConsequent* - vyhľadáva spojenia odohrávajúce sa na rovnakých MAC, IP adresách a časových súvislostiach. Porty môžu byť rozdielne, čo vychádza z možného správania *buffer overflow* útokov. Metóda ukladá počty takýchto predchádzajúcich a nasledovných spojení a ukladá si prvé nasledujúce spojenie kvôli detailnejšiemu skúmaniu.
- **MetricExtractor** postupne extrahuje sadu metrík z pripravených zoznamov obsahujúcich dáta potrebné k extrakcii a aj samotné spojenia.
  - *extractMetrics* - riadi celý proces. Volá jednotlivé metódy, ktoré ukladajú do interného zoznamu *metricList* postupne hodnoty každej metriky. Tento zoznam sa na konci extrakcie každého spojenia pridá do zoznamu *exportAllConnections*. Po úspešnom spracovaní všetkých spojení zapíše tieto hodnoty do výstupného súboru a prípadne pokračuje ďalším vstupným súborom. Ak užívateľ zadá argument pre vytvorenie výstupu zobrazujúceho detaily o každom spojení, inštancia triedy *ConnectionPrinter* bude v tejto metóde vytvorená.
  - *getLastID* - vyhľadá výstupný súbor. Ak existuje a obsahuje nejaké výsledky extrakcie metrík, metóda vráti poradové číslo posledného spojenia. V opačnom prípade sa začína indexovať od 0.
  - *extExpert* - vyberie uloženú expertnú znalosť, a zapíše či spojenie predstavuje útok alebo nie. Okrem toho zapíše do kolekcie metrík ešte aj označenie služby a kategórie daného typu. Aktuálne sa používajú dve kategórie, no pre budúce použitie je ľahko možné doplniť v rámci experimentov viac.
  - *nonStandardTraffic* - extrahuje metriky informujúce o tom, či sa na danom kontexte nachádza alebo nenachádza aspoň jeden TCP alebo ICMP paket. Metóda je volaná dva krát, pretože vyhľadáva oba druhy sieťovej prevádzky zvlášť. Vyhľadávanie je uskutočňované porovnávaním MAC a IP adries a príbuzností časov. Maximálny rozdiel časov príchodu takéhoto paketu a posledného, v rámci daného UDP spojenia, bol stanovený na 30 sekúnd.

- *extLocation* - z daného spojenia extrahuje zdrojové aj cieľové IP, MAC adresy a porty. Metóda nie je použitá, pretože v laboratórnych podmienkach sa adresy nemenili. Pri nasadení do reálnej sieťovej prevádzky avšak má jej použitie zmysel.
- *extTOS* - extrahuje štatistické údaje o ToS (*Type of Service*) z hlavičky IP paketu z oboch smerov.
- *extTTL* - vygeneruje štatistické hodnoty TTL (*Time to Live*) získané z IP hlavičky z oboch smerov.
- *extCntLenPkts* - extrahuje počty paketov v oboch smeroch a ich pomery. Metóda ďalej počíta rôzne štatistické hodnoty z veľkostí užitočných dát paketov.
- *extCntLenFragmented* - metóda pre extrahovanie metrík pracujúcich s fragmentovanými paketami. Zisťuje ich počty a dĺžky prenesených dát.
- *extTransferTime* - pracuje s prenosovými časmi, časmi príchodov paketov a na základe nich počíta rôzne štatistické hodnoty pre oba smery. Naviac počíta prenosovú rýchlosť a celkový čas spojenia.
- *extMetricsForQuartilTimes* - obsluhuje vykonávanie procesov extrakcie metrík definovaných nad kvartilmi. Tie boli zavedené kvôli spresneniu výsledkov, pretože jasnejšie vyjadrujú priebeh spojenia počas 4 časových úsekoch. Získavanie dát a ich ukladanie do interného zoznamu *metricList* vykonávajú ďalšie metódy.
- *findQuatrilSections* - rozdelí spojenie na 4 rovnako dlhé úseky podľa času. Metóda vracia viac rozmerný zoznam obsahujúci kvartily a v nich pakety k nim prislúchajúce.
- *extQEthDataLens* - slúži pre extrahovanie štatistických hodnôt pre veľkosti ethernetových dát. Metóda je volaná postupne nad každým kvartilom.
- *extQCntPkts* - má podobnú úlohu ako predchádzajúca metóda, teda extrahovanie metrík postupne nad každým kvartilom. Jej činnosť je jednoduchá, pracuje s počtami paketov v rámci daného kvartilu.
- *extConseqConnections* - bola implementovaná kvôli spresneniu výsledkov. Metóda vyhľadáva spojenia v celom súbore, ktoré majú časové náväznosti na dané spojenie či už v čase pred zahájením alebo po ukončení spojenia. Adresy MAC a IP musia byť zhodné, no porty sa môžu meniť. Metriky, ktoré sa tu počítajú, pracujú okrem toho hlavne s prvým nasledujúcim nadväzujúcim spojením, z neho sa skúmajú rôzne údaje o počte, veľkostí dát, alebo aj celých paketov.
- *redirectOutput* - bola implementovaná pre účely experimentovania s dátovou sadou *pcap190flows*. Na základe IP adries alebo služby prebiehajúcej na danom spojení presmeruje výstup zobrazujúci detaily o spojeniach do viacerých súborov. Hlavnou motiváciou je lepšia prehľadnosť s cieľom analyzovať väčšiu množinu dát. Výstup, ktorý metóda vracia, bude potom použitý ako parameter pre konštruktor triedy *ConnectionPrinter*, ktorá už iba spojenia vypíše.
- *exportToCSV* - zapisuje do CSV súboru sadu metrík pre všetky spojenia v danom vstupnom súbore. Ak je súbor prázdny alebo neexistuje, vygeneruje prvý riadok obsahujúci názvy metrík a následne ich hodnoty.

- **ConnectionPrinter** zapúzdruje proces výpisu detailov o danom spojení. Môže byť ľahko modifikovaná pre experimentálne účely. V konštruktore získa poradové číslo aktuálneho spojenia, jeho dáta, a výstup do ktorého má zapisovať. Výstupom môže byť už otvorený súbor alebo štandardný výstup do konzoly. Má nasledujúce metódy:

– *makeLog* - prebieha v ňom celý proces vytvárania výstupného záznamu alebo logu.
– *printConnectionInfo* - vypíše informácie o danom spojení, teda adresy, porty, expertnú znalosť a názov danej sieťovej služby.
– *printAllPkts* - postupne vypíše všetky pakety v danom spojení. Pre jednoduchšiu analýzu bude zobrazená iba časť dát, ktoré obsahujú, ako napríklad časy príchodov, alebo veľkosti prenášaných užitočných dát.

Kapitola 6

Testovanie a experimenty

Testovanie bolo realizované počas vývoja samotného i po jeho dokončení. Zameriavalo sa na funkčnosť, rýchlosť vykonávania a extrémne prípady, ktoré môžu nastať. Pod tým sa myslí hlavne chyba spôsobená operáciami s desatinnými číslami používaných v časoch príchodov paketov. Keďže sú na tom založené kľúčové časti systému, pri pretečení číselného typu alebo pri kumulovanej chybe spôsobenej sieťovou prevádzkou nad veľkým časovým intervalom môže dôjsť k chybným výsledkom, ktoré nástroj vygeneruje. Uvedené problémy boli v istej miere otestované a ošetrené, no z popisu problému riziko v istej miere stále pretrváva.

6.1 Rozšírenie trénovacej množiny

Pôvodne malo byť experimentovanie vykonávané nad dátovou sadou *cdx2009*[19], no záznam nástroja Snort ukázal, že neobsahuje žiadne útoky, na aké sa zameriava táto práca. Zároveň bola k dispozícií dátová sada s názvom *pcap190flows* obsahujúca UDP komunikáciu na službách DNS, Multicast DNS, Multicast SSDP, NetBIOS, SIP a TFTP. Nebolo ale možné úplne presne určiť, ktoré pakety a spojenia boli útoky, pretože záznam komunikácie obsahoval expertnú znalosť iba pre TCP spojenia. Vedelo sa avšak, ktoré zraniteľné aplikácie boli cieľmi útokov. Podľa nich sa vyhľadali v databáze [5] dostupné exploity a určilo sa, ktoré z nich predstavujú útoky relevantné k tejto práci. Bolo zistené, že táto dátová množina obsahuje 5 útokov, z toho 1 na službu SIP a 4 na TFTP. Je ale neznáme, či zvyšná sieťová komunikácia pri ktorej neboli poskytnuté žiadne vedomosti náhodou neobsahuje nejaké útoky. Kvôli riziku, že by sa takto znepresnila trénovacia množina, boli tieto zvyšné spojenia stanovené ako legitímne komunikácie.

Pre vierohodnejšie výsledky pri klasifikácií je potrebné čo najviac dát. Kvôli tomu bolo rozhodnuté, že sa trénovacia množina rozšíri. Ďalšia časť sa skladala z 13 útokov na službu TFTP. Rozloženie komunikácie na jednotlivých službách a jej zadelenie do triedy komunikácie je možné vidieť v tabuľke 6.1. V nej sú zobrazené údaje o celej dátovej sade, ktorá bola použitá. Niektoré útoky obsahovali aj viac spojení, kvôli tomu je v stĺpci *Útoky* väčšie číslo než súčet všetkých útokov. Exploitov bolo použitých 10, no niektoré útoky boli simulované viacerými spôsobmi. Ukázalo sa, že počet známych zraniteľností a útokov na UDP služby nie je zďaleka tak početný ako na TCP. Spočiatku bol problém útoky nájsť a použiť. Kvôli tomu bol napísaný skript prijímajúci jednoznačné čísla zraniteľností, ozn. *CVE-id*, na základe ktorých sa vyhľadávali a automaticky sťahovali útoky a zraniteľné aplikácie z verejne

[19] Dátova sada *cdx2009* sprostredkovaná národnou agentúrou The National Security Agency je prístupná na http://www.westpoint.edu/crc/SitePages/DataSets.aspx.

dostupnej databázy [5] pomocou HTTP dotazov. Spomínané *CVE-id* boli výsledkami PostgreSQL databázových dotazov nad databázou výskumnej skupiny Brno University Security Laboratory. Dáta uložené v tejto databáze pochádzali z National Vulnerability Database. Zraniteľné aplikácie boli nainštalované vo virtuálnom operačnom systéme Windows XP SP3, a zdrojové kódy mierne upravené pre účely experimentov. Okrem tohto bolo treba nájsť legitímnu sieťovú prevádzku nad uvedenými službami TFTP a SIP, k tomu boli použité ukážky zachytenej komunikácie zo stránky pre nástroj Wireshark a portál obsahujúci mnoho podobných príkladov, viď http://pcapr.net.

Trieda / Služba	Útoky	Legitímne spojenia
SIP	1	18
TFTP	22	6
Iné	-	4746

Tabuľka 6.1: Počet spojení u jednotlivých služieb v použitej dátovej sade.

6.2 Rozvoj kolekcie metrík

Po úspešnom implementovaní nástroja boli vykonávané experimenty zameriavajúce sa na analýzu a vyhodnotenie výsledkov. Korektné zadelenie paketov do jednotlivých spojení je nutná požiadavka na správnosť výsledných dát z extrakcie metrík. Ukázalo sa, že je veľmi podstatné správne, resp. pokiaľ možno čo najoptimálnejšie určiť časový limit používaný pri tomto vyhľadávaní. Z počiatku bol definovaný časový rozdiel príjmov dvoch po sebe idúcich paketov na 2 minúty. Neskôr sa ukázalo, že tento čas v mnohých prípadoch nesprávne triedi pakety do spojení. Podľa mnohých Cisco zariadení, niektorých MikroTik, alebo aj v jadrách niektorých operačných systémov sa používajú hodnoty dve, a to 30 a 180 sekúnd. S hodnotou 180 sekúnd sa je možné stretnúť aj v protokoloch niektorých služieb, aj keď nie vždy pre rovnaké účely. Tieto hodnoty boli použité aj v implementačnej časti práce, pričom 30 sekúnd bolo stanovených ak medzi komunikujúcimi stranami prebieha výmena dát len jedným smerom. Pokiaľ príde z opačného smeru aspoň jeden paket, spojenie je tzv. zaistené a tento čas sa zmení na 180 sekúnd. Tieto čísla sa používajú aj pri vyhľadávaní opačných, poprípade nadväzujúcich spojení. Len pre predstavu závažnosti problému, pri experimentoch so stanoveným jedným nemenným číslom 30 sekúnd sa našlo v dátovej sade *pcap190flows* vyše 900 spojení, a pri 180 sekundách okolo 500.

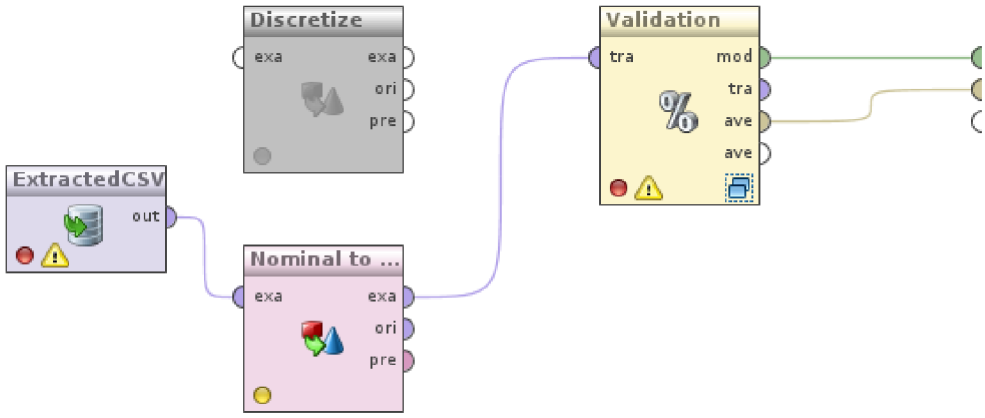
Ďalšou zmenou bolo vylúčenie metrík, ktoré v laboratórnych podmienkach nemajú žiadny zmysel. Jedná sa o zdrojové a cieľové IP a MAC adresy, a zdrojový port. Adresy sa príliš nemenili, pretože útoky boli umelo vygenerované a zdrojový port bol často náhodné, vysoké číslo.

Posledným prírastkom sa stali 2 metriky, ktoré indikujú prítomnosť príchodu TCP a ICMP paketu. Pri tom sa využíva časový limit 30 sekúnd od konca aktuálneho spojenia. Okrem toho sa porovnávajú IP a MAC adresy. Porty nemá zmysel porovnávať, pretože sa jedná o iné služby. Vďaka tomu, že v tejto práci bola generovaná vlastná komunikácia obsahujúca útoky sa zistilo, že niekedy pri *buffer overflow* útokoch po ukončení UDP spo-

jenia prichádzajú nejaké pakety z TCP alebo ICMP. Prvý prípad môže byť nápomocný pri indikácií *backdoor* spojenia a druhý pri násilnom ukončení služby, teda *Denial of Service*. Práca sa síce zameriava iba na útoky *buffer overflow*, no tie často majú aj takéto následky. Navrhnutá kolekcia metrík by sa možno dala prispôsobiť buď pre generickú detekciu sieťových útokov, alebo pre ďalšiu špecifickú kategóriu.

6.3 Experimenty s klasifikačným modelom

Pre účely klasifikácie bol použitý nástroj RapidMiner Studio s verziou 6.4. Spôsob experimentovania s jednotlivými modelmi bol inšpirovaný prácou [14]. Bloková schéma zapojenia jednotlivých modulov v nástroji RapidMiner je znázornená na obrázku 6.1. Modul *Discretize* zafarbený na šedo nebol v tej chvíli zapojený. Schéma sa skladá z modulov:



Obr. 6.1: Bloková schéma hlavného procesu pre analýzu vyhodnotených metrík.

- *ExtractedCSV* pre načítanie dát, ktoré vytvoril nástroj implementovaný v tejto práci. Načítané dáta sú považované za atribúty, ku ktorým sa pridelia dátové typy a role,
- *Discretize*, ktorý nie je povinný, no v niektorých prípadoch spresňuje klasifikáciu. Jeho úlohou je konverzia numerických atribútov na nominálne tým, že diskretizuje numerické hodnoty do tzv. košov, angl. *bins*, ktorých počet sa dá definovať. S týmto počtom sa experimentovalo tiež, pričom sa pohybovalo v rozmedzí od 2 do 6,
- *Nominal to Numerical*, v prípade niektorých validačných modeloch nevyhnutný pre správne načítanie dát. Nenumerické atribúty a ich hodnoty zmení na numerické,
- *Validation* využívajúci princíp krížovej validácie. Tá vyhodnocuje efektivitu a výkonnosť celého modelu. Na obrázku 6.2 sú znázornené 2 subprocesy. Každý obsahuje nejaké komponenty, pričom v subprocese pre trénovanie dát je aktívny iba jeden, ostatné sú znázornené šedou farbou. V rámci experimentov boli pre porovnanie výsledkov tieto komponenty obmieňané. Trénovací subproces trénuje model a jeho výsledky sa potom aplikujú na testovací subproces. Ak sa nastaví počet validácií väčší než 1, tak sa tento proces opakuje. V tejto práci bolo experimentované s rôznym počtom týchto opakovaní. Testovací subproces aplikuje dáta vyhodnotené trénovaním na operátor, ktorý vyhodnocuje kvalitu klasifikačného procesu nad testovacími dátami. Výkon modelu

následne vyhodnocuje modul *Validation*. Pri experimentoch sa najviac uplatnilo stratifikované vzorkovanie, ktoré vytvára vlastné náhodné podmnožiny a pomocou nich zaisťuje rovnomerné zastúpenia tried.

Najviac sa experimentovalo s komponentami krížovej validácie v procese trénovania dát. Detailnejší popis obsahujú nasledujúce podsekcie. Niektoré zaujímavé rozloženia hustoty metrík sú znázornené v grafoch v prílohe C. Zaujímavé je hlavne skúmanie najbližšieho nadväzujúceho spojenia, prvý a posledný kvartil a TTL. Experimenty s metódou *ID3* prípadne ďalšie popísané nebudú, pretože výsledky boli výrazne neuspokojivé. V prípade *ID3* bolo okolo 13% legitímnej komunikácie zle klasifikovaných ako útoky, pričom nasledujúce experimenty nanajvýš 5%. Klasifikácia útokov bola tiež v mnohých prípadoch niekedy až výrazne nepresnejšia.

6.3.1 Rozhodovací strom

Typ kritéria rozhodovacieho stromu sa používal *gain ratio* a počet validácií bol 10. Tento klasifikačný model bol prepojený s komponentou *Nominal to Numerical*. Ako ukazuje výsledná matica zámen 6.2, detekovaných útokov bolo 95.65% a presnosť klasifikácie legitímnych komunikácií 100%. Celková úspešnosť klasifikačného modelu je teda 99.98% (±0.06%).

99.98% (±0.06%)	**Skutočné True**	**Skutočné False**	**Presnosť tried**
Predikované True	22	0	100%
Predikované False	1	4770	99.98%
Odozva modelu	95.65%	100%	

Tabuľka 6.2: Výsledky z rozhodovacieho stromu zobrazené ako matica zámen.

6.3.2 SVM

Metóda *Support Vector Machine* bola použitá tiež s komponentou *Nominal to Numerical* a počet validácií bol tiež 10. Výsledky zaznamenáva matica zámen 6.3. Útokov bolo správne detekovaných 91.30%. Chybná klasifikácia legitímnych spojení ako útokov nastala v 0.1% prípadoch. Celková úspešnosť klasifikačného modelu je teda 99.85% (±0.13%).

99.85% (±0.13%)	**Skutočné True**	**Skutočné False**	**Presnosť tried**
Predikované True	21	5	80.77%
Predikované False	2	4765	99.96%
Odozva modelu	91.30%	99.90%	

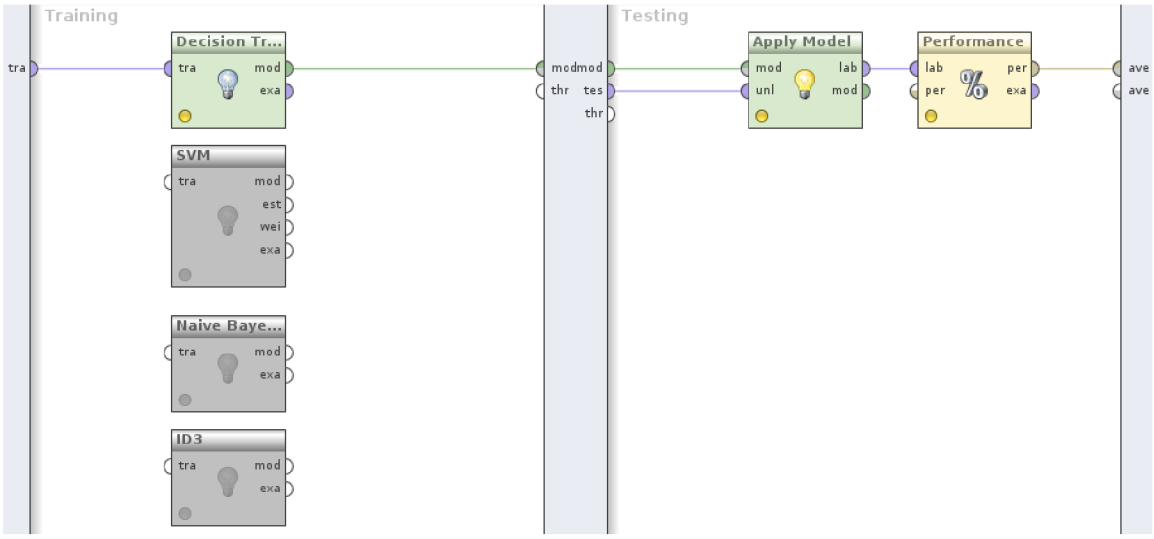
Tabuľka 6.3: Výsledky z metódy SVM zobrazené ako matica zámen.

6.3.3 Metóda Naive Bayes

Najoptimálnejšie výsledky sa ukázali keď nebola použitá komponenta *Discretize* ani *Nominal to Numerical*. Počet validácií bol až 12, pričom mód výpočtu komponenty *Naive Bayes (Kernel)* bol stanovený ako *greedy* a počet jadier 4. Výsledky sú zaznamenané v tabuľke ako matica zámen 6.4. Útokov bolo správne detekovaných 95.45%. Legitímne spojenia boli v 1.51% prípadoch označené ako útoky. Celková úspešnosť klasifikačného modelu je teda 98.47% (±0.76%). Pri ďalšom experimentovaní s touto metódou a pri použití 2 jadier boli všetky útoky identifikované správne, no legitímna komunikácia iba v 94.96% prípadoch.

98.47% (±0.76%)	**Skutočné True**	**Skutočné False**	**Presnosť tried**
Predikované True	21	72	22.58%
Predikované False	1	4688	99.98%
Odozva modelu	95.45%	98.49%	

Tabuľka 6.4: Výsledky z metódy Naive Bayes zobrazené ako matica zámen.



Obr. 6.2: Bloková schéma validačného bloku obsahujúceho 2 subprocesy.

Kapitola 7

Záver

Hlavnou náplňou tejto práce bolo vytvoriť takú kolekciu metrík, ktorou bude možné úspešne detekovať sieťové útoky typu *buffer overflow* nad UDP sieťovými spojeniami. Metodológia realizovaná v tejto práci postupuje tak, aby bolo možné detekovať aj *Zero-day* útoky, pretože sa zameriava na správanie a dôsledky útoku, nie na jeho konkrétnu realizáciu. Preto bol navrhnutý a implementovaný nástroj, ktorý na základe vytvorenej sady metrík vyhodnotil vstupné súbory obsahujúce zachytenú komunikáciu. Výsledky tejto extrakcie boli analyzované pomocou klasifikačného nástroja RapidMiner.

Úvodná časť tejto práce zahrňovala štúdium a analýzu *buffer overflow* útokov a princípy uplatňujúce sa pri ich tvorbe a realizácií. S ohľadom na tieto vedomosti následne štúdium existujúcich spôsobov ich detekcie so zameraním na IDPS systémy, ako aj štúdium nástrojov používaných pri sieťových útokoch vo všeobecnosti.

Ďalším krokom bolo nájsť kritické UDP služby, resp. také, ktoré obsahujú najviac zraniteľností a sú k nim k dispozícií útoky. Potom nasledovala analytická činnosť, ktorej úlohou bolo vytvoriť metriky opisujúce správanie *buffer overflow* útokov po sieti s požiadavkou na UDP sieťovú prevádzku. Táto požiadavka vychádzala z toho, že táto oblasť nie je plne preskúmaná, ako tomu nasvedčovali aj príbuzné práce, ktoré sa zameriavali skôr na TCP komunikáciu. Metrík bolo vytvorených 117, pričom 5 nebolo v laboratórnych podmienkach využívaných vôbec.

Potom nasledovala implementácia pozostávajúca z dvoch hlavných častí. Úlohou prvej bolo zatriediť pakety do jednotlivých spojení a druhej ich následné vyhodnotenie pomocou navrhnutých metrík. Okrem toho bolo nutné ešte získať expertnú znalosť pre klasifikátor, teda zistiť, či vstupná komunikácia v úlohe trénovacej množiny dát predstavuje útok alebo legitímnu prevádzku. K tomu sa využili metadáta a adresárová štruktúra, v ktorej sa súbory nachádzali.

Po úspešnej implementácií nasledovala analýza poskytnutej UDP komunikácie a zber ďalšej komunikácie, hlavne takej, ktorá obsahuje sieťové útoky relevantné k tejto práci. Pomocou skriptu implementovaného v tejto práci pre automatické sťahovanie exploitov bolo možné rozšíriť dátovú množinu o 13 útokov na službu TFTP. Celkovo táto práca obsahovala 18 útokov, avšak iba na dve rozdielne služby, a to SIP a spomínané TFTP.

Posledným krokom bolo experimentovanie v nástroji RapidMiner. Najlepšie výsledky vyhodnotil model *Rozhodovací strom*. Pre zadanú dátovú sadu vyhodnotil úspešnosť detekcie útokov v 95.65% prípadoch a legitímnu komunikáciu vyhodnotil správne vo všetkých prípadoch. Celková úspešnosť bola teda 99.98% ($\pm$0.06%), no je nutné poznamenať, že v dátovej sade bolo nájdených okolo 4800 spojení, čo je veľmi malé číslo s porovnaní s 24 hodinovou sieťovou prevádzkou v rozsiahlejšej sieti. Na druhom mieste bola dosiahnutá úspešnosť

99.85% (±0.13%) pomocou metódy *SVM*. Správna detekcia útokov bola v 91.30% prípadoch a nesprávna detekcia legitímnej komunikácie ako útokov v 0.10% prípadoch. Tretí najlepší výsledok vyhodnotil model *Naive Bayes* s presnosťou 98.47% (±0.76%), pričom mal väčší problém správne klasifikovať legitímnu komunikáciu, konkrétne v 1.51% prípadoch. Zaujímavé výsledky prinieslo rozdelenie spojenia na 4 rovnaké časové úseky. Tento prístup sa väčšinou v príbuzných prácach nevyskytuje. Výsledky ukazujú, že najčastejšie práve posledný, poprípade prvý kvartil je najdôležitejší, pretože pakety obsahujúce *shellcode* sú prenášané počas tohto intervalu. Okrem toho sa zdá, že UDP útoky majú rozdielne hodnoty TTL a ich hustotu rozloženia, než legitímne komunikácie.

7.1 Možnosti ďalšieho rozvoja

V laboratórnych podmienkach nebol plne využitý potenciál niektorých metrík a teda nebolo možné posúdiť detekciu *Zero-day* útokov. Je vysoko pravdepodobné, že by nad väčšou množinou dát klasifikačný model vyhodnotil odlišné výsledky. V prípade, že by sa v práci ďalej pokračovalo, by bolo veľmi prínosné získať veľké objemy dát. Nasadenie do reálnej sieťovej prevádzky v súčasnosti nie je možné, pretože spracovávanie dát by sa muselo výrazne zrýchliť.

Kedže charakter práce nekladie žiadny dôraz na rýchlosť vykonávania programu, nebol použitý rýchlejší jazyk ani rozsiahle optimalizácie. Je nutné poznamenať, že napriek optimalizáciám interpretovaný jazyk Python nemusí vyhovovať. Použitie hardwarového riešenia a výraznej paralelizácie by sa mohlo ukázať ako dobrým riešením tohto problému.

Ďalším zlepšením pripadá do úvahy experimentovanie s rozličnými detekčnými modelmi, najmä takými, ktoré neobsahuje nástroj RapidMiner. Skvalitnenie detekcie je pravdepodobne možné modifikáciou navrhnutej kolekcie metrík za účelom čo najviac zvýšiť mieru správneho detekovania útoku a čo najviac znížiť mieru chybného vyhodnocovania legitímnej sieťovej prevádzky ako útoku. Napríklad sofistikovanejšie analyzovanie posledného a predposledného kvartilu a zrušenie alebo zmena niektorých metrík môže priniesť lepšie výsledky.

Implementovaný skript pre automatické sťahovanie zraniteľných aplikácií a exploitov je možné použiť aj v budúcnosti, napríklad pri tvorbe ďalšej dátovej množiny pre sofistikovanejšie experimenty. Žiaľ niekedy je problematickejšie tieto útoky nájsť a úspešne spustiť, pretože informácia o tom, či útok funguje nad TCP alebo UDP sa nachádza v samotných zdrojových kódoch. Ďalším faktom je, že zraniteľné aplikácie často nie sú dostupné.

Literatúra

[1] Databáza National Vulnerability Database (NVD).
https://web.nvd.nist.gov/view/vuln/statistics.

[2] Metasploit. Verejne dostupný nástroj pre generovanie útokov.
http://www.rapid7.com/products/metasploit/.

[3] Nessus. Verejne dostupný nástroj pre skenovanie zraniteľností.
http://www.tenable.com/blog.

[4] RapidMiner Studio.
https://rapidminer.com/products/studio/.

[5] Verejne dostupná databáza exploitov.
http://www.exploit-db.com.

[6] Verejne dostupná databáza zraniteľností v programoch.
http://www.securityfocus.com.

[7] AlephOne: Smashing The Stack For Fun And Profit. *Phrack Vol. 7, Issue 49*, 1996.

[8] Alouneh S., Kharbutli M., AlQurem R.: Stack Memory Buffer Overflow Protection Based on Duplication and Randomization. *Procedia Computer Science 21*, 2013: s. 250 – 256.

[9] Axelsson, S.: Intrusion Detection Systems: A Survey and Taxonomy. Technická zpráva, Chalmers University of Technology, 2000.

[10] Cole, E.: *Network Security Bible.* Wiley Publishing, Inc, druhé vydání, 2009, ISBN 978-0-470-50249-5.

[11] Cowan, C.: Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. *DISCEX'00. Proceedings. Vol. 2. IEEE*, 2000.

[12] Dobšíček M., Ballner R.: *Linux Bezpečnost a exploity.* Kopp, 2004, ISBN 80-7232-243-5.

[13] Foster, J.: *Buffer Overflow Attacks: Detect, Exploit, Prevent.* Syngress Publishing, 2005, ISBN 1-932266-67-4.

[14] Homoliak, I.: *Metriky pro detekci útoků v síťovém provozu.* diplomová práca, Vysoké učení technické v Brně, 2012.

[15] Hoque N., Bhuyan M., Baishya R.C., Bhattacharyya D.K., Kalita J.K.: Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, 2013.

[16] McNab, C.: *Network Security Assessment.* O'Reilly, 2004, ISBN 0-596-00611-X.

[17] Moore A., Zuev D., Crogan M.: Discriminators for use in flow-based classification. Technická zpráva, University of London, 2005.

[18] Provos N., Holz T.: *Virtual Honeypots: From Botnet Tracking to Intrusion Detection.* Addison-Wesley, 2008, ISBN 978-0-321-33632-3.

[19] Shacham H., Matthew P., Pfaff B., Goh E., Modadugu N., Boneh D.: On the Effectiveness of Address-Space Randomization. Technická zpráva, Stanford University, 2004.

[20] Younan, Y.: 25 Years of Vulnerabilities: 1988-2012 [online]. https://courses.cs.washington.edu/courses/cse484/14au/reading/25-years-vulnerabilities.pdf.

Dodatok A

Navrhnuté metriky

A.1 Dáta z hlavičiek paketov

Číslo	Názov metriky	Popis
1	srcMAC	Zdrojová MAC adresa.
2	dstMAC	Cieľová MAC adresa.
3	srcHost	Zdrojová IP adresa.
4	dstHost	Cieľová IP adresa.
5	srcPort	Zdrojový port.
6	dstPort	Cieľový port.
7	modusTOS	Modus hodnôt ToS (*Type of Service*) z IP hlavičiek paketov v rámci daného spojenia.
8	modusTOSop	Modus hodnôt ToS z IP hlavičiek paketov v spojení v opačnom smere.
9	medianTOS	Medián hodnôt ToS z IP hlavičiek paketov v rámci daného spojenia.
10	medianTOSop	Medián hodnôt ToS z IP hlavičiek paketov v spojení v opačnom smere.
11	aMeanTOS	Aritmetický priemer hodnôt ToS z IP hlavičiek paketov v rámci daného spojenia.
12	aMeanTOSop	Aritmetický priemer hodnôt ToS z IP hlavičiek paketov v spojení v opačnom smere.

Tabuľka A.1: Navrhnuté metriky získané extrahovaním dát z hlavičiek paketov z UDP spojení (časť 1).

Číslo	Názov metriky	Popis
13	minTTL	Minimálna hodnota TTL (*Time to Live*) z IP hlavičiek paketov v rámci daného spojenia.
14	minTTLop	Minimálna hodnota TTL z IP hlavičiek paketov v spojení v opačnom smere.
15	maxTTL	Maximálna hodnota TTL z IP hlavičiek paketov v rámci daného spojenia.
16	maxTTLop	Maximálna hodnota TTL z IP hlavičiek paketov v spojení v opačnom smere.
17	sumTTL	Súčet hodnôt TTL z IP hlavičiek paketov v rámci daného spojenia.
18	sumTTLop	Súčet hodnôt TTL z IP hlavičiek paketov v spojení v opačnom smere.
19	medianTTL	Medián hodnôt TTL z IP hlavičiek paketov v rámci daného spojenia.
20	medianTTLop	Medián hodnôt TTL z IP hlavičiek paketov v spojení v opačnom smere.
21	modusTTL	Modus hodnôt TTL z IP hlavičiek paketov v rámci daného spojenia.
22	modusTTLop	Modus hodnôt TTL z IP hlavičiek paketov v spojení v opačnom smere.
23	aMeanTTL	Aritmetický priemer hodnôt TTL z IP hlavičiek paketov v rámci daného spojenia.
24	aMeanTTLop	Aritmetický priemer hodnôt TTL z IP hlavičiek paketov v spojení v opačnom smere.
25	stdDevTTL	Smerodajná odchýlka hodnôt TTL z IP hlavičiek paketov v rámci daného spojenia.
26	stdDevTTLop	Smerodajná odchýlka hodnôt TTL z IP hlavičiek paketov v spojení v opačnom smere.

Tabuľka A.2: Navrhnuté metriky získané extrahovaním dát z hlavičiek paketov z UDP spojení (časť 2).

A.2 Dáta získané z počtov a veľkostí paketov

Číslo	Názov metriky	Popis
27	cntPktsInConnection	Počet všetkých paketov v rámci daného spojenia.
28	cntPktsInConnectionOp	Počet všetkých paketov v spojení v opačnom smere.
29	ratioCntPktsInOut	Pomer počtu paketov na danom spojení a v opačnom smere.
30	cntDataPkts	Počet paketov, ktoré majú nenulovú veľkosť dát v rámci daného spojenia.
31	cntNoDataPkts	Počet paketov, ktoré majú nulovú veľkosť dát v rámci daného spojenia.
32	ratioCntNodataDataPkts	Pomer počtu paketov, ktoré majú nulovú veľkosť dát a paketov, ktoré majú nenulovú veľkosť dát v rámci daného spojenia.
33	lenAllDataPktsPayload	Suma veľkostí dátových polí v UDP segmentoch v rámci daného spojenia.
34	modusLenAllDataPktsPayload	Medián veľkostí dátových polí v UDP segmentoch v rámci daného spojenia.
35	medianLenAllDataPktsPayload	Medián veľkostí dátových polí v UDP segmentoch v rámci daného spojenia.
36	aMeanLenAllDataPktsPayload	Aritmetický priemer veľkostí dátových polí v UDP segmentoch v rámci daného spojenia.
37	stdDevLenAllDataPktsPayload	Smerodajná odchýlka veľkostí aritmetických priemerov dátových polí v UDP segmentoch v rámci daného spojenia.

Tabuľka A.3: Navrhnuté metriky získané extrahovaním dát z počtov a veľkostí paketov z UDP spojení.

A.3 Skúmanie fragmentácie v spojeniach

Číslo	Názov metriky	Popis
38	cntFragPkts	Počet fragmentovaných paketov v rámci daného spojenia.
39	cntNonFragPkts	Počet nefragmentovaných paketov v rámci daného spojenia.
40	ratioFragNonfragPkts	Pomer počtu paketov, ktoré boli fragmentované k počtu nefragmentovaných v rámci daného spojenia.
41	lenAllFragPktsPayload	Suma veľkostí dátových polí v UDP segmentoch fragmentovaných paketoch v rámci daného spojenia.
42	ratioLenFragNonfragPktsPayload	Pomer sumy veľkostí dátových polí v UDP segmentoch fragmentovaných paketov a nefragmentovaných paketov v rámci daného spojenia.

Tabuľka A.4: Navrhnuté metriky získané extrahovaním dát o fragmentácií paketov z UDP spojení.

A.4 Skúmanie celého časového priebehu spojenia

Číslo	Názov metriky	Popis
43	durationTimeMinutes	Celkový čas spojenia.
44	bytesPerSec	Veľkosť prenesených bajtov za sekundu. Reprezentácia rýchlosti v rámci daného spojenia.
45	sumNoTransferTimeAllMinutes	Suma časov, počas ktorých neprebiehajú žiadne dátové prenosy v rámci daného spojenia.
46	sumNoTransferTime1750Minutes	Suma časov počas ktorých neprebiehajú žiadne dátové prenosy v rámci daného spojenia a ktoré sú väčšie než 1750 milisekúnd.

Tabuľka A.5: Navrhnuté metriky získané extrahovaním dát o časovom priebehu prenosov paketov z UDP spojení (časť 1).

Číslo	Názov metriky	Popis
47	ratioNoTransferTime1750Duration	Pomer časov počas ktorých neprebiehajú žiadne dátové prenosy v rámci daného spojenia a ktoré sú väčšie než 1750 milisekúnd a dĺžky celého spojenia.
48	pktsReceivTimeMin	Minimálna doba medzi príchodmi paketov v rámci daného spojenia.
49	pktsReceivTimeMinOp	Minimálna doba medzi príchodmi paketov v spojení v opačnom smere.
50	pktsReceivTimeMax	Maximálna doba medzi príchodmi paketov v rámci daného spojenia.
51	pktsReceivTimeMaxOp	Maximálna doba medzi príchodmi paketov v spojení v opačnom smere.
52	pktsReceivTimeMedian	Medián dôb príchodov paketov v rámci daného spojenia.
53	pktsReceivTimeMedianOp	Medián dôb príchodov paketov v spojení v opačnom smere.
54	pktsReceivTimeAMean	Aritmetický priemer dôb príchodov paketov v rámci daného spojenia.
55	pktsReceivTimeAMeanOp	Aritmetický priemer dôb príchodov paketov v spojení v opačnom smere.
56	pktsReceivTimeStdDev	Smerodajná odchýlka dôb príchodov paketov v rámci daného spojenia.
57	pktsReceivTimeStdDevOp	Smerodajná odchýlka dôb príchodov paketov v spojení v opačnom smere.

Tabuľka A.6: Navrhnuté metriky získané extrahovaním dát o časovom priebehu prenosov paketov z UDP spojení (časť 2).

A.5 Skúmanie jednotlivých časových úsekov spojenia

Číslo	Názov metriky	Popis
58	q1minLenDataEth	Minimálna hodnota z veľkostí dátových polí ethernetových rámcoch v 1. štvrtine časového úseku z daného spojenia.
59	q1maxLenDataEth	Maximálna hodnota z veľkostí dátových polí ethernetových rámcoch v 1. štvrtine časového úseku z daného spojenia.
60	q1medianLenDataEth	Medián z veľkostí dátových polí ethernetových rámcoch v 1. štvrtine časového úseku z daného spojenia.
61	q1modusLenDataEth	Modus z veľkostí dátových polí ethernetových rámcoch v 1. štvrtine časového úseku z daného spojenia.
62	q1aMeanLenDataEth	Aritmetický priemer z veľkostí dátových polí ethernetových rámcoch v 1. štvrtine časového úseku z daného spojenia.
63	q1stdDevLenDataEth	Smerodajná odchýlka z veľkostí dátových polí ethernetových rámcoch v 1. štvrtine časového úseku z daného spojenia.
64	q1ratioAMeanThisQAllQDataEth	Pomer veľkostí aritmetických priemerov dátových polí ethernetových rámcoch v 1. štvrtine spojenia a v celom spojení.
65	q1cntPkts	Počet paketov v 1. štvrtine časového úseku z daného spojenia.
66	q1cntDataPkts	Počet paketov, ktoré majú nenulovú veľkosť dát v 1. štvrtine časového úseku z daného spojenia.
67	q1cntPktsOp	Počet paketov v 1. štvrtine časového úseku v spojení v opačnom smere.
68	q1cntDataPktsOp	Počet paketov, ktoré majú nenulovú veľkosť dát v 1. štvrtine časového úseku v spojení v opačnom smere.
69	q1ratioCntPktsInOut	Pomer počtu paketov v 1. štvrtine časového úseku z daného spojenia a v opačnom smere.

Tabuľka A.7: Navrhnuté metriky získané extrahovaním dát z časových úsekov UDP spojenia (časť 1).

Číslo	Názov metriky	Popis
70	q2minLenDataEth	Minimálna hodnota z veľkostí dátových polí ethernetových rámcoch v 2. štvrtine časového úseku z daného spojenia.
71	q2maxLenDataEth	Maximálna hodnota z veľkostí dátových polí ethernetových rámcoch v 2. štvrtine časového úseku z daného spojenia.
72	q2medianLenDataEth	Medián z veľkostí dátových polí ethernetových rámcoch v 2. štvrtine časového úseku z daného spojenia.
73	q2modusLenDataEth	Modus z veľkostí dátových polí ethernetových rámcoch v 2. štvrtine časového úseku z daného spojenia.
74	q2aMeanLenDataEth	Aritmetický priemer z veľkostí dátových polí ethernetových rámcoch v 2. štvrtine časového úseku z daného spojenia.
75	q2stdDevLenDataEth	Smerodajná odchýlka z veľkostí dátových polí ethernetových rámcoch v 2. štvrtine časového úseku z daného spojenia.
76	q2ratioAMeanThisQAllQDataEth	Pomer veľkostí aritmetických priemerov dátových polí ethernetových rámcoch v 2. štvrtine spojenia a v celom spojení.
77	q2cntPkts	Počet paketov v 2. štvrtine časového úseku z daného spojenia.
78	q2cntDataPkts	Počet paketov, ktoré majú nenulovú veľkosť dát v 2. štvrtine časového úseku z daného spojenia.
79	q2cntPktsOp	Počet paketov v 2. štvrtine časového úseku v spojení v opačnom smere.
80	q2cntDataPktsOp	Počet paketov, ktoré majú nenulovú veľkosť dát v 2. štvrtine časového úseku v spojení v opačnom smere.
81	q2ratioCntPktsInOut	Pomer počtu paketov v 2. štvrtine časového úseku z daného spojenia a v opačnom smere.

Tabuľka A.8: Navrhnuté metriky získané extrahovaním dát z časových úsekov UDP spojenia (časť 2).

Číslo	Názov metriky	Popis
82	q3minLenDataEth	Minimálna hodnota z veľkostí dátových polí ethernetových rámcoch v 3. štvrtine časového úseku z daného spojenia.
83	q3maxLenDataEth	Maximálna hodnota z veľkostí dátových polí ethernetových rámcoch v 3. štvrtine časového úseku z daného spojenia.
84	q3medianLenDataEth	Medián z veľkostí dátových polí ethernetových rámcoch v 3. štvrtine časového úseku z daného spojenia.
85	q3modusLenDataEth	Modus z veľkostí dátových polí ethernetových rámcoch v 3. štvrtine časového úseku z daného spojenia.
86	q3aMeanLenDataEth	Aritmetický priemer z veľkostí dátových polí ethernetových rámcoch v 3. štvrtine časového úseku z daného spojenia.
87	q3stdDevLenDataEth	Smerodajná odchýlka z veľkostí dátových polí ethernetových rámcoch v 3. štvrtine časového úseku z daného spojenia.
88	q3ratioAMeanThisQAllQDataEth	Pomer veľkostí aritmetických priemerov dátových polí ethernetových rámcoch v 3. štvrtine spojenia a v celom spojení.
89	q3cntPkts	Počet paketov v 3. štvrtine časového úseku z daného spojenia.
90	q3cntDataPkts	Počet paketov, ktoré majú nenulovú veľkosť dát v 3. štvrtine časového úseku z daného spojenia.
91	q3cntPktsOp	Počet paketov v 3. štvrtine časového úseku v spojení v opačnom smere.
92	q3cntDataPktsOp	Počet paketov, ktoré majú nenulovú veľkosť dát v 3. štvrtine časového úseku v spojení v opačnom smere.
93	q3ratioCntPktsInOut	Pomer počtu paketov v 3. štvrtine časového úseku z daného spojenia a v opačnom smere.

Tabuľka A.9: Navrhnuté metriky získané extrahovaním dát z časových úsekov UDP spojenia (časť 3).

Číslo	Názov metriky	Popis
94	q4minLenDataEth	Minimálna hodnota z veľkostí dátových polí ethernetových rámcoch v 4. štvrtine časového úseku z daného spojenia.
95	q4maxLenDataEth	Maximálna hodnota z veľkostí dátových polí ethernetových rámcoch v 4. štvrtine časového úseku z daného spojenia.
96	q4medianLenDataEth	Medián z veľkostí dátových polí ethernetových rámcoch v 4. štvrtine časového úseku z daného spojenia.
97	q4modusLenDataEth	Modus z veľkostí dátových polí ethernetových rámcoch v 4. štvrtine časového úseku z daného spojenia.
98	q4aMeanLenDataEth	Aritmetický priemer z veľkostí dátových polí ethernetových rámcoch v 4. štvrtine časového úseku z daného spojenia.
99	q4stdDevLenDataEth	Smerodajná odchýlka z veľkostí dátových polí ethernetových rámcoch v 4. štvrtine časového úseku z daného spojenia.
100	q4ratioAMeanThisQAllQDataEth	Pomer veľkostí aritmetických priemerov dátových polí ethernetových rámcoch v 4. štvrtine spojenia a v celom spojení.
101	q4cntPkts	Počet paketov v 4. štvrtine časového úseku z daného spojenia.
102	q4cntDataPkts	Počet paketov, ktoré majú nenulovú veľkosť dát v 4. štvrtine časového úseku z daného spojenia.
103	q4cntPktsOp	Počet paketov v 4. štvrtine časového úseku v spojení v opačnom smere.
104	q4cntDataPktsOp	Počet paketov, ktoré majú nenulovú veľkosť dát v 4. štvrtine časového úseku v spojení v opačnom smere.
105	q4ratioCntPktsInOut	Pomer počtu paketov v 4. štvrtine časového úseku z daného spojenia a v opačnom smere.

Tabuľka A.10: Navrhnuté metriky získané extrahovaním dát z časových úsekov UDP spojenia (časť 4).

A.6 Skúmanie príbuzných spojení

Číslo	Názov metriky	Popis
106	TCPfollowing	Informuje o príchode aspoň jedného TCP paketu. Tento príznak nadobúda hodnotu *True*, pokiaľ paket prišiel najneskôr do 30 sekúnd od príchodu posledného paketu v danom spojení.
107	ICMPfollowing	Informuje o príchode aspoň jedného ICMP paketu. Tento príznak nadobúda hodnotu *True*, pokiaľ paket prišiel najneskôr do 30 sekúnd od príchodu posledného paketu v danom spojení.
108	portChanged	Príznak označujúci zmenu portu u nadväzujúceho spojenia. Maximálny rozdiel času začiatku nasledujúceho spojenia a času ukončenia aktuálneho spojenia bol stanovený na 30/180 sekúnd.
109	cntPrevConnectionsConseq	Počet predchádzajúcich spojení, pri ktorých musí byť dodržaná časová postupnosť s maximálnym rozdielom 30/180 sekúnd do začiatku aktuálneho spojenia.
110	cntNextConnectionsConseq	Počet nasledujúcich spojení, pri ktorých musí byť dodržaná časová postupnosť s maximálnym rozdielom 30/180 sekúnd od konca aktuálneho spojenia.
111	cntPktsInNextConseq	Počet paketov u nasledujúceho nadväzujúceho spojenia.
112	bytesPerSecNextConseq	Veľkosť prenesených bajtov za sekundu u nasledujúceho nadväzujúceho spojenia. Reprezentácia rýchlosti v rámci daného spojenia.
113	aMeanAllPktsPayloadNextConseq	Aritmetický priemer veľkostí dátových polí v UDP segmentoch u nasledujúceho nadväzujúceho spojenia.
114	lenAllPktsPayloadNextConseq	Suma veľkostí dátových polí v UDP segmentoch u nasledujúceho nadväzujúceho spojenia.

Tabuľka A.11: Navrhnuté metriky získané extrahovaním dát z nadväzujúcich UDP spojení (časť 1).

Číslo	Názov metriky	Popis
115	aMeanLenAllPktSizesNextConseq	Aritmetický priemer veľkostí paketov u nasledujúceho nadväzujúceho spojenia.
116	lenAllPktSizesNextConseq	Suma veľkostí paketov u nasledujúceho nadväzujúceho spojenia.
117	stdDevLenAllPktSizesNextConseq	Smerodajná odchýlka veľkostí paketov u nasledujúceho nadväzujúceho spojenia.

Tabuľka A.12: Navrhnuté metriky získané extrahovaním dát z nadväzujúcich UDP spojení (časť 2).

Dodatok B

Ukážka CSV súboru

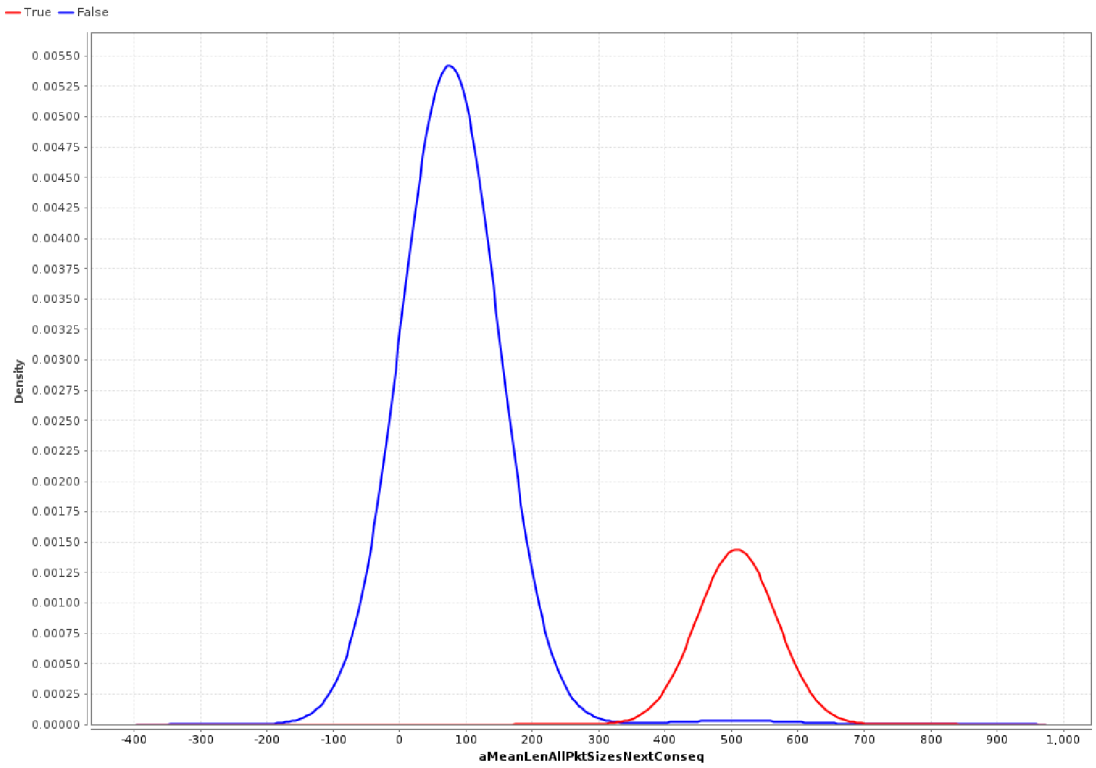
id	isAttack	label	dstPort	modusTOS	modusTOSop	medianTOS	medianTOSop	aMeanTOS
0	True	SIP_1	5060	0	0	0	0	0
1	False	SIP_0	5061	0	0	0	0	0
2	False	SIP_0	5060	0	0	0	0	0
3	False	SIP_0	5001	0	0	0	0	0

Obr. B.1: Časť výstupného CSV súboru od jeho začiatku.

Dodatok C

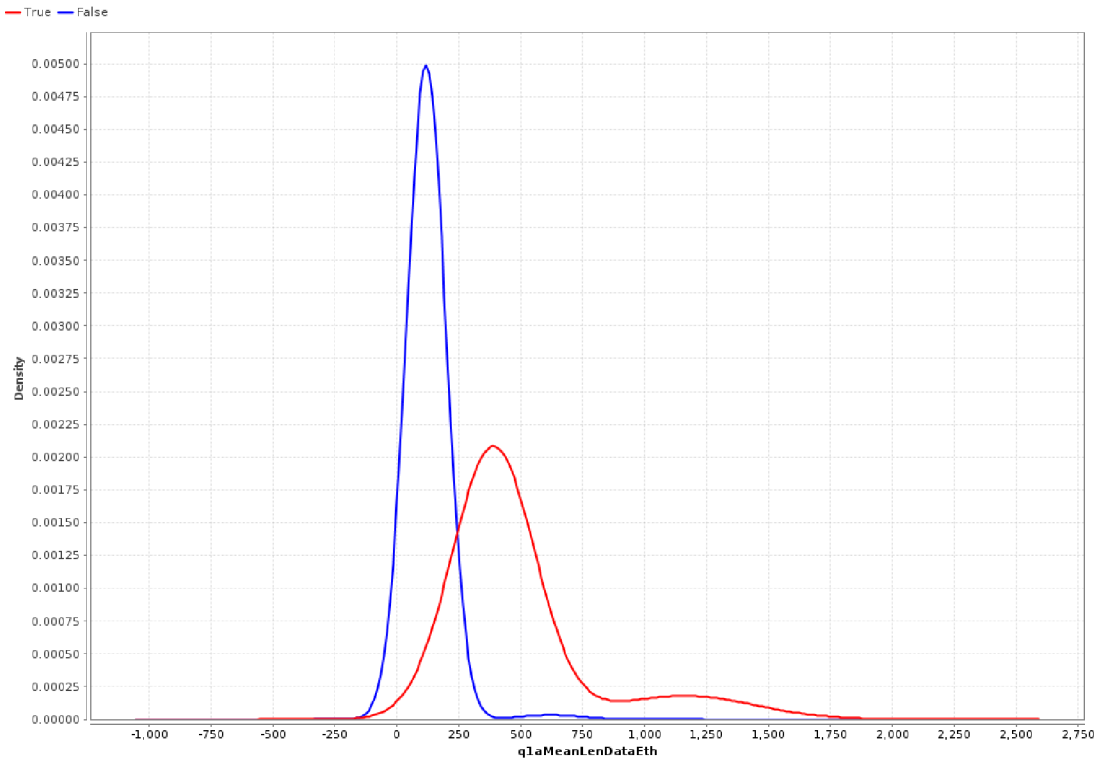
Grafy rozloženia hustoty metrík

C.1 Veľkosť paketov v nasledujúcom nadväzujúcom spojení

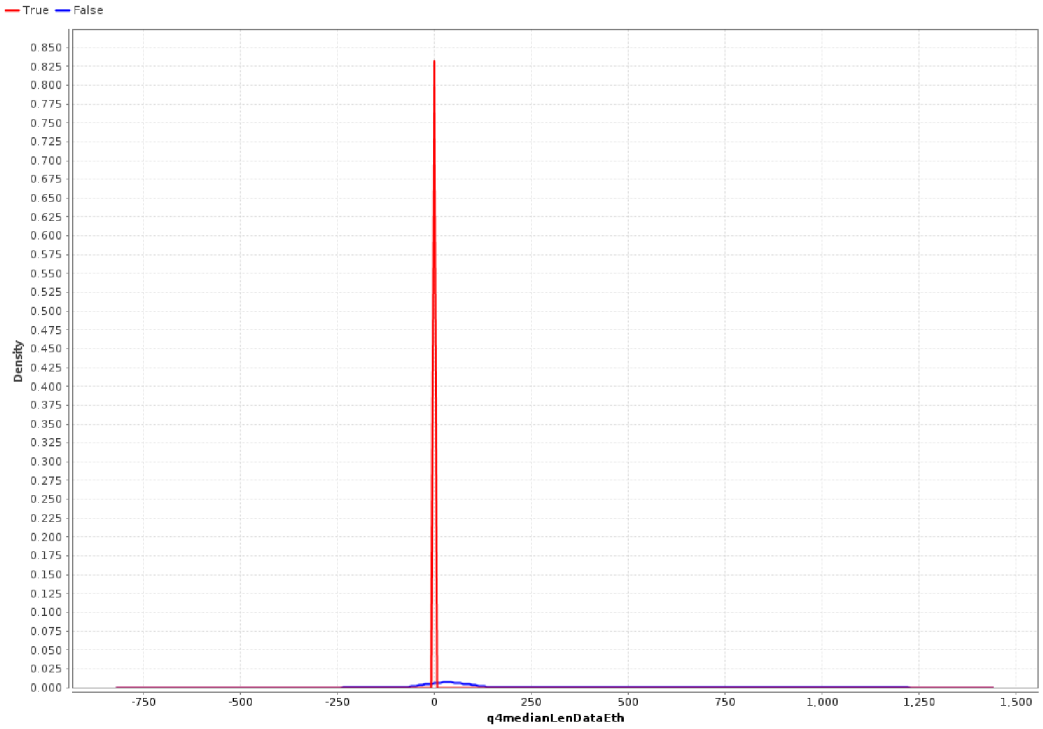


Obr. C.1: Rozloženie hustoty metriky aMeanLenAllPktSizesNextConseq.

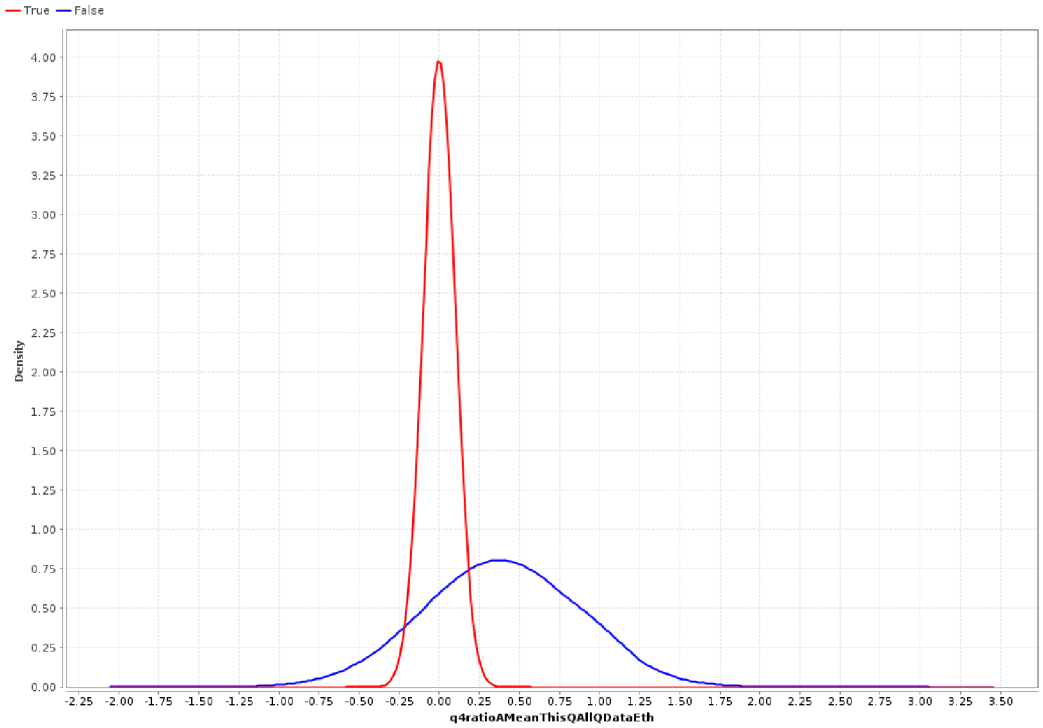
C.2 Prvý a posledný kvartil



Obr. C.2: Rozloženie hustoty metriky q1aMeanLenDataEth.

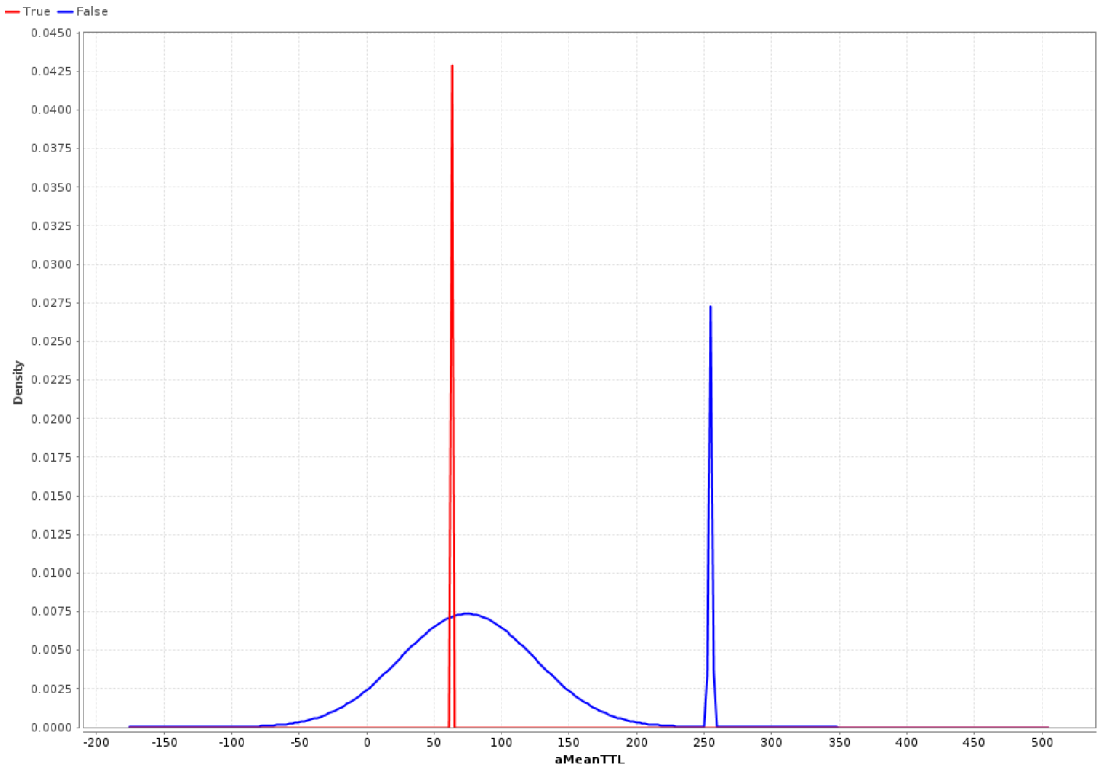


Obr. C.3: Rozloženie hustoty metriky q4medianLenDataEth.



Obr. C.4: Rozloženie hustoty metriky q4ratioAMeanThisQAllQDataEth.

C.3 Time to live



Obr. C.5: Rozloženie hustoty metriky aMeanTTL.

Dodatok D

Obsah CD

- **doc** - zdrojové kódy textovej časti bakalárskej práce pre systém LaTeX,
- **experiments** - dáta potrebné pre experimenty, predovšetkým súbory obsahujúce zachytenú komunikáciu, zdrojové súbory útokov, zraniteľné aplikácie a ďalšie,
- **outputs** - výstupy vygenerované nástrojom na extrakciu kolekcie metrík,
- **src** - všetky zdrojové súbory v jazyku Python,
- **README.txt** - popis nástroja, návod na jeho inštaláciu a používanie.