VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

CHARAKTERIZACE SÍŤOVÉHO PROVOZU POČÍTAČŮ A JEJICH SKUPIN

CHARACTERISATION OF A COMPUTER DATA NETFLOW AND THEIR GROUPS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ROSTISLAV KUČERA**
AUTHOR

VEDOUCÍ PRÁCE **Mgr. Ing. PAVEL OČENÁŠEK, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Kučera Rostislav**
Program: Informační technologie
Název: **Charakterizace síťového provozu počítačů a jejich skupin**
Characterization of Network Operation of Computers and Their Groups
Kategorie: Web

Zadání:

1. Seznamte se s principy analýzy anomálií v prostředí systémů počítačových sítí.
2. Analyzujte požadavky na systém umožňující analýzu surového síťového provozu v čase na počítačích a jejich skupinách pomocí statistiky, frekvenční analýzy, případně vybraných metod umělé inteligence.
3. Navrhněte systém pro detekci anomálií dle předchozího bodu se zaměřením na detekci provozních a bezpečnostních anomálií.
4. Navržený systém implementujte dle instrukcí vedoucího práce.
5. Implementovaný systém ověřte na vhodně zvolených reálných datech.
6. Diskutujte získané výsledky a možnosti dalšího rozšíření.

Literatura:

- Kurose, J. F. Computer networking: A top-down approach. Pearson, Essex, 2017, ISBN 978-1-292-15359-9.
- Stallings, W. Network security essentials: Applications and standards. Hoboken, 2016, ISBN 978-0-13-452733-8.
- Bishop, M. Computer security: Art & Science. Addison-Wesley, Boston, 2003, ISBN 0-201-44099-7.
- Buczak, A., Guven, E.. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. IEEE Communications surveys and tutorials. IEEE, 2016, 18(2), s. 1153-1176.

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

Vedoucí práce: **Očenášek Pavel, Mgr. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 11. října 2021

Abstrakt

Cílem této práce je implementace modulu pro detekci DDoS útoků. Modul zpracovává síťový provoz, který dále zpracovává, ukládá jeho profil, ze kterého jsou následně vypočítány statistické údaje sloužící pro samotnou detekci. Práce se také věnuje samotné implementaci modulu pro velmi rozšířený systém detekce průniků Suricata.

Abstract

The aim of this work is to implement a module for detecting DDoS attacks. The module processes network traffic, processes it, stores its profile, from which statistical data used for the detection itself are subsequently calculated. The work also deals with the implementation of the module for intrusion detection system Suricata.

Klíčová slova

IDS, DDoS, NetFlow, statistická analýza, detekce anomálie, Suricata

Keywords

IDS, DDoS, NetFlow, statistical approach, anomaly detection, Suricata

Citace

KUČERA, Rostislav. *Charakterizace síťového provozu počítačů a jejich skupin.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Mgr. Ing. Pavel Očenášek, Ph.D.

Charakterizace síťového provozu počítačů a jejich skupin

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Mgr. Ing. Pavla Očenáška, Ph.D. Další informace mi poskytl pan Ing. Petr Chmelař. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Rostislav Kučera
8. května 2022

Poděkování

Chtěl bych tímto velmi poděkovat panu Mgr. Ing. Pavlu Očenáškovi za odborné vedení mé práce. Dále pak i panu Ing. Petru Chmelařovi za jeho velmi cenné rady a konzultace týkající se mé práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Kapitola 1

Úvod

V současné době dochází ke stále většímu využívání internetu jako komunikační linky mezi dvěma, či více uživateli. Obzvláště ve chvílích tzv. koronavirové krize se stále více mezilidských kontaktů ubírá touhle cestou. Dochází ke zvětšování objemu přenášených dat, ve většině případů se jedná také o data velmi citlivá. Tato data se pokouší řada útočníků získat ku svému prospěchu. Ať už se jedná o informace bankovních účtů a karet, až po velmi citlivá osobní data. Lze také touto cestou velmi jednoduše narušit infrastrukturu státních zřízení. Často se snaží o napadení státních institucí, zdravotnických zařízení nebo mediálních služeb, či prostředků.

Vzhledem k faktům, která jsem se pokusil shrnout v předešlém odstavci, se nyní spousta společností a jednotlivců snaží o implementaci softwaru, který by dokázal takovýto pokus napadení včas odhalit a pokusit se jej zablokovat.

Jedná se o tzv. IDS (Intrusion Detection System) a IPS (Intrusion Prevension System) zařízení, které mají za úkol detekovat škodlivou komunikaci v síti. V případě IDS pak dojde pouze k upozornění, že by se mohlo jednat o škodlivou komunikace, zatímco IPS je schopno tomuto útoku předejít a zablokovat jej.

Jedním z dalších způsobů je použití zařízení zvaného Firewall. Jedná se o nástroj, ať už hardwarově či softwarově implementovanou bránu, která má za úkol blokovat nevhodnou komunikaci mezi sítěmi. Tato se však řídí předem určenými pravidly a není tak příliš chytrým řešením v rámci blokování nevyžádané a škodlivé síťové komunikace, jelikož pravidla se týkají pouze filtrování v rámci IP adres a portů.

Daleko lepším řešením je již výše zmíněný systém IDS. Ten již aktivně sleduje průchozí komunikace a je schopen rozpoznat ty škodlivé. Oproti IPS však není toto zařízení schopno této komunikaci zamezit, dokáže pouze upozornit uživatele či jiného administrátora.

V této práci se pokusím objasnit, jak takové systémy IDS/IPS fungují, a jaké techniky rozpoznávání anomálií jsou použity k tomu, aby byly co nejlépe odhaleny. Zároveň se také pokusím vytvořit nástroj, který takové anomálie v síťovém provozu odhaluje.

Nejdříve si však vysvětlíme základní principy monitorování provozu sítě v kapitole 2, druhy útoků, proti kterým má IDS bránit v následující kapitole 3, principy fungování těchto systémů v kapitole 4, nejznámější IDS produkty 5. Hlavním bodem je pak popis návrhu a implementace vlastního detekčního modulu v kapitole 6, jehož detekční schopnosti také v závěru otestujeme a zhodnotíme 7.

Kapitola 2

Monitorování komunikace

Jak jsem již v úvodu zmínil, vzhledem ke stále narůstajícímu množství přenášených dat v síti, je velmi žádoucí takovýto provoz monitorovat. Díky tomuto monitorování tak můžeme snadněji předcházet bezpečnostním problémům, které mohou v síti nastat. V praxi pak takový provoz můžeme detailněji sledovat, sestavovat statistiky a dále jej analyzovat. Díky této analýze jsme pak částečně schopni detekovat nestandardní chování síťové komunikace.

V praxi takto monitorujeme tok dat, u kterého jsou ukládána metadata, obsahující informace o zdrojových či cílových IP adresách, portech a například velikostí toků. Samotný tok je definován jako posloupnost paketů, mající společnou vlastnost (zdrojové a cílové IP adresy, porty a protokol) a procházející bodem pozorování za určitý čas.

Pomocí shromažďování těchto informací o tocích jsme schopni lépe odhalovat nestandardní provoz v síti, tzv. anomální. To však není jediná úloha sledování toků. Dokážeme pomocí těchto technologií např. odhalit místa v síti, kde dochází k velké kumulaci informací a provoz zde tak může tvořit fronty. Můžeme tak lépe navrhnout další rozšíření sítě tak, aby k těmto situacím nedocházelo.

Nás však více zajímá využití bezpečnostní, kterému bych se chtěl více v této kapitole věnovat. Nejdříve si však představíme některé z důležitých protokolů pro zaznamenávání síťových toků.

2.1 NetFlow

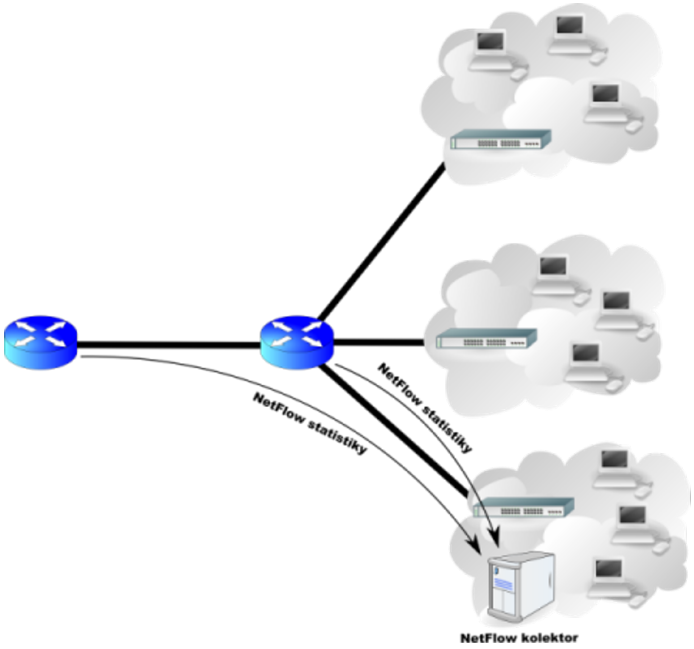
Protokol NetFlow vyvíjený společností Cisco patří k nejpoužívanějším protokolům pro sledování provozu. NetFlow poskytuje informace o jednotlivých tocích, které mohou sloužit pro odhalovánání provozních anomálií v síti, součástí kterých můžou být i bezpečnostní hrozby. Přenášenými informacemi o toku jsou pak např. zdrojová s cílová IP adresa, síťové porty, počet bajtů, časové značky a další. Více se však obsahu budu věnovat v dalších podsekcích [8].

2.1.1 Architektura NetFlow

Typická architektura NetFlow se skládá ze zařízení (případně softwaru), které odesílá informace o tocích – tzv. exportér. Zpravidla se jedná o aplikaci běžící na směrovači, či firewallu. Na pomyslné druhé straně se pak nachází kolektor, zařízení, které naopak sbírá údaje vyslané exportérem a následně je pak ukládá. Komunikace mezi těmito zařízeními pak probíhá právě skrze protokol NetFlow.

NetFlow exportér monitoruje provoz, zpracovává procházející pakety a ukládá data o tocích do doby, než tok považuje za ukončený. Ukončením toku je považováno přijetí konce toku např. příznaky RST či FIN u protokolu TCP, neaktivita toku, přílišná délka toku nebo zaplnění paměti pro ukládání NetFlow záznamů. Jakmile je tedy tok považován za ukončený, je vyřazen z paměti a statistika toku přeposlána je kolektoru.

Kolektorem naopak bývá server, který dále zpracovává přijaté statistiky z několika exportérů, které dále analyzuje a zpracovává. Na takovém serveru pak může běžet aplikace, která tyto statistické údaje zobrazuje ve formě grafů.



Obrázek 2.1: Ukázka klasické konfigurace architektury NetFlow [13].

2.1.2 Protokol NetFlow

Pro přenos informací mezi exportérem a kolektorem slouží protokol NetFlow. Ten obsahuje všechny důležité údaje popisující daný síťový tok bez toho, aby obsahoval samotná data toku. Informace, které takový protokol obsahuje se liší podle jeho verze.

NetFlow protokol je pak nejčastěji zapouzdřen v protokolu UDP. To však s sebou nese jistá rizika. V případě, že se paket cestou ztratí, UDP nezná cestu, jakou dát exportéru zpět vědět, že paket nedorazil. Není tedy možnost, jak jej poslat znovu a paket nesoucí informace exportéru je tak ztracen. Proto se u novějších verzí protokolu NetFlow využívá přenosového protokolu SCTP (Stream Control Transmisson Protocol), který tento problém odstraňuje.

Typickým volně dostupným NetFlow kolektorem je `Nfdump`[1].

[1] http://nfdump.sourceforge.net

<table>
<tr><td colspan="4">Zdrojová IP adresa</td></tr>
<tr><td colspan="4">Cílová IP adresa</td></tr>
<tr><td colspan="4">IP adresa dalšího hopu</td></tr>
<tr><td colspan="2">SNMP index vstupního rozhraní</td><td colspan="2">SNMP index výstupního rozhraní</td></tr>
<tr><td colspan="4">Počet paketů</td></tr>
<tr><td colspan="4">Počet bajtů</td></tr>
<tr><td colspan="4">Čas začátku toku</td></tr>
<tr><td colspan="4">Čas konce toku</td></tr>
<tr><td colspan="2">Zdrojový port</td><td colspan="2">Cílový port</td></tr>
<tr><td>Odsazení</td><td>Příznaky</td><td>Protokol</td><td>Typ služby</td></tr>
<tr><td colspan="2">Číslo autonomního systému vstupu</td><td colspan="2">Číslo autonomního systému výstupu</td></tr>
<tr><td>Prefix zdroj. IP</td><td>Prefix cíl. IP adresy</td><td colspan="2">Odsazení</td></tr>
</table>

Obrázek 2.2: NetFlow protokol ve verzi 5.

2.2 Nfdump

Nfdump je soubor nástrojů pro práci s protokolem NetFlow. Celý projekt se skládá z nástrojů [14]:

- `Nfcapd` je nástroj pro zpracování příchozí NetFlow komunikace, kterou následně uloží do binárních Nfdump souborů. Jeho volitelné rozšíření Nfpcapd také umí zpracovat vstupní pcap komunikaci a vygenerovat z ní komunikaci ve formátu protokolu NetFlow. Takové soubory ukládá v určitých časových intervalech (zpravidla v intervalu 5 minut), pro lepší práci s uloženými daty.
- `Nfdump` slouží pro zobrazení uložených dat pomocí nástroje Nfcapd. Ze souborů nfdump umí vytvořit statistiky, nebo je jen prostě zobrazit na výstup. Zobrazení pak umí provádět v několika formátech: raw, line, long a extended. Takový formát může uživatel zvolit pomocí přepínače -o a zadání názvu příslušného formátu (např. -o long). Nástroj nfdump také umožňuje vyfiltrovat určitou informaci obsahu vstupního NFDUMP soubory a následně tyto vyfiltrované informace uložit do nového NFDUMP souboru.
- `Nfanon` anonymizuje IP adresy v NetFlow záznamech pomocí metody CryptoPAn.
- `Nfreplay` čte uložená NetFlow data a přeposílá je na další zařízení v síti.
- `Nfsen` je pak nástavba sloužící pro grafické zobrazení statistik z Netflow záznamů.

Date first seen	Duration	Proto	Src IP Addr:Port	Dst IP Addr:Port	Packets	Bytes	Flows
2018-08-11 07:15:08.833	0.000	UDP	192.168.1.95:53526 ->	192.168.1.6:53	1	36	1
2018-08-11 07:15:08.834	0.000	UDP	192.168.1.6:53 ->	192.168.1.95:53526	1	111	1
2018-08-11 07:15:09.082	0.000	UDP	192.168.1.95:61895 ->	192.168.1.6:53	1	55	1
2018-08-11 07:15:09.083	0.000	UDP	192.168.1.6:53 ->	192.168.1.95:61895	1	123	1
2018-08-11 07:15:09.084	0.000	UDP	192.168.1.95:53976 ->	192.168.1.6:53	1	50	1

Tabulka 2.1: Výpis záznamu NetFlow vygenerovaného programem Nfdump.

Kapitola 3

Vybrané typy útoků odepření služby v počítačových sítích

Nejprve je potřeba se zaměřit na to, jak vůbec takové útoky v počítačových sítích vypadají a jakým způsobem napadání probíhá. V mé práci se věnuji primárně detekci anomálií pomocí metod statistické analýzy. Provoz v síti je tak neustále sledován a v případě určitých výkyvů je vyvoláno upozornění, týkající se možného probíhajícího útoku, na který je třeba se zaměřit.

Jak takové útoky vypadají a čím se liší se pokusím popsat v této kapitole, konkrétně v sekci 3.2. Nejdříve si pak pro lepší pochopení vysvětlíme funkci ISO/OSI modelu v sekci 3.1 [4].

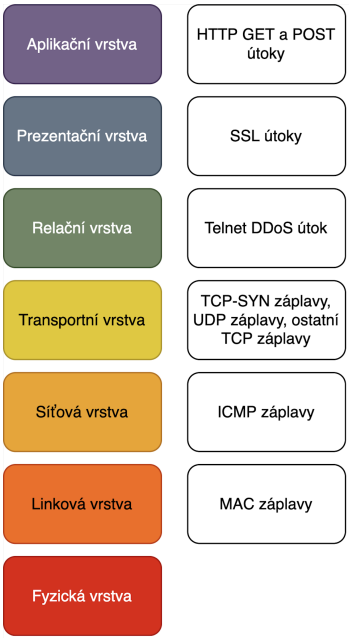
3.1 ISO/OSI model

Referenční model vytvořený společností ISO tzv. ISO/OSI model má za úkol normovaně popsat vzájemnou komunikaci počítačů v síti. Definuje 7 abstraktních vrstev, které mají za úkol vytvořit kompletní komunikaci mezi počítači. Každá taková vrstva má jinou funkci. Model je zobrazen na obrázku 3.1 včetně příkladů DDoS útoků příslušných dané vrstvě.

Fyzická vrstva je nejnižší vrstvou celého modelu. Má za úkol definovat přenos na úrovni konektorů a jejich pinů. Základní jednotkou přenosu této vrstvy je **bit**.

Linková vrstva navazuje na fyzickou vrstvu a má za úkol přenos dat skrze přenosovou linku. Přenos je prováděn vytvořením tzv. rámců. Linková vrstva má význam hlavně na lokální úrovni, kdy na její úrovni dochází k adresování zařízení v sítí. Typickým zařízením, které funguje na této vrstvě, je přepínač, angl. switch. Což je zařízení, které dokáže směrovat na lokální úrovni síťový tok ke správnému zařízení pomocí adresy MAC. Protokolem pracujícím na této vrstvě je Ethernet.

Síťová vrstva zabezpečuje správně směrování přenosu napříč počítačovými sítěmi. Data zde tvoří pakety. Pro správné směrování je však nutno znát cílovou adresu počítače - IP adresu. Zařízení pracující na této vrstvě je směrovač, angl. router. Ten se stará o správně směrování paketů, to je přeposílání z jedné sítě do druhé. Protokoly pracující na této vrstvě jsou např. IP nebo ICMP.



Obrázek 3.1: ISO/OSI model s příklady DDoS útoků na jednotlivých vrstvách [4].

Transportní vrstva je v pořadí čtvrtou vrstvou modelu ISO/OSI. Tato vrstva umožňuje přímé adresovaní aplikaci pomocí tzv. aplikačních portů. Na aplikační vrstvě tak pracují základní protokoly jako TCP a UDP.

Relační vrstva má za úkol řízení dialogu mezi dvěma koncovými účastníky - aplikacemi.

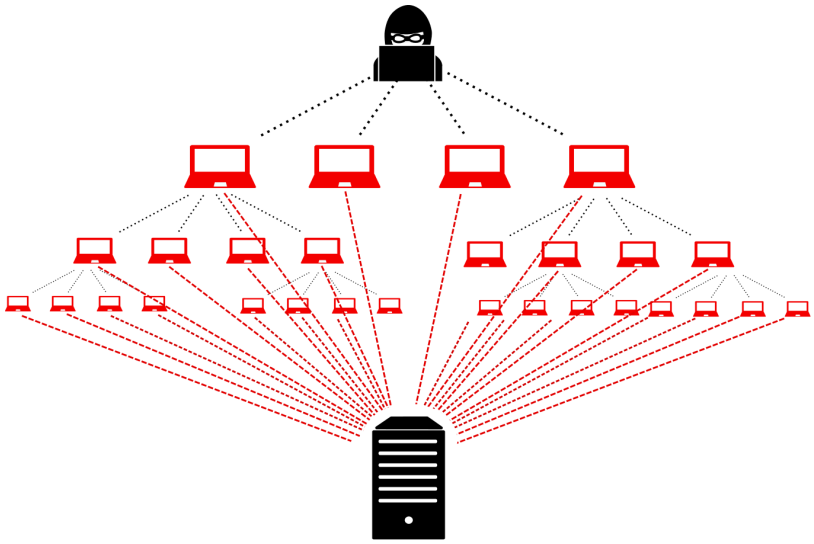
Prezenční vrstva má za úkol zpracování jednotného formátu dat posílaných z nejvyšší aplikační vrstvy tak, aby byla dobře srozumitelná.

Aplikační vrstva je nejvyšší vrstvou modelu ISO/OSI. Jejím úkolem je poskytování přístupu aplikacím ke komunikačnímu systému.

3.2 Vybrané DDoS útoky

Jedním z nejznámějších a také nejčastějších prováděných útoků je **Odepření služby**, angl. **Denial of Service** (dále již **DoS**). V případě, že tato škodlivá komunikace přichází z několika zdrojů, označujeme tento útok dodatkem „Distributed“, tedy jako **Distributed Denial of Service** (dále **DDoS**). Každé zařízení má jen omezené množství výpočetního výkonu CPU, velikost paměti nebo rychlost síťového připojení. Základním kamenem DoS typu útoku je předpoklad, že pokud zařízení oběti zahrneme velkým množstvím dotazů, z nichž některé vůbec nemusí dávat smysl, vyčerpáme alespoň jeden z těchto zdrojů, a tím vyřadíme z provozu na něm probíhající službu. Takovým dotazem může být vysílání velkého množství UDP paketů, TCP paketů s příznaky, které samy o sobě nemusejí mít význam, případně také dotazování ICMP zprávami. Podle toho, skrze jaký protokol je útok vykonáván, rozdělujeme DoS na další typy. DDoS útoky lze také rozdělit podle vrstvy ISO/OSI modelu, na které je útok provozován [23].

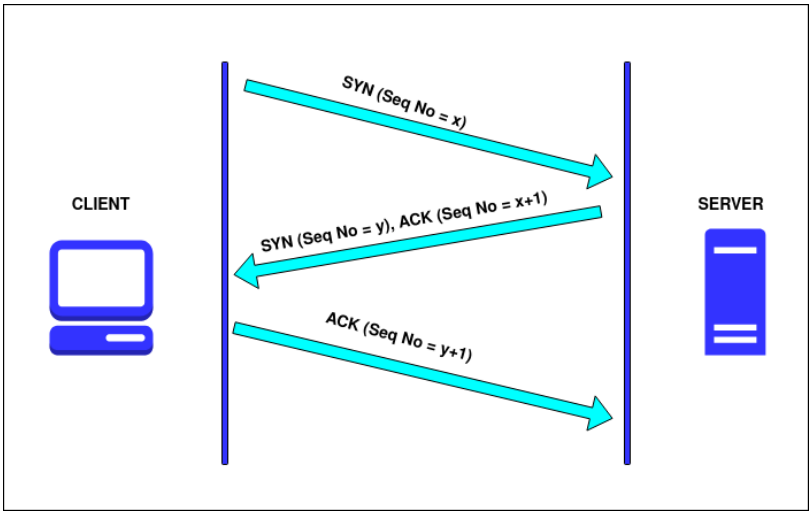
Útočník často využívá tzv. botnetů. Botnetem se označuje síť počítačů, které jsou napadeny malwarem, tedy softwarem, díky kterému může být počítač ovládán na dálku útočníkem. Tyto sítě počítačů tak generují síťový provoz útočící následně na jeden, nebo více cílových zařízení.



Obrázek 3.2: Ukázka DDoS útoku skrze botnety [22].

3.2.1 TCP SYN záplavy

Před vysvětlením tohoto útoku bychom si nejdřív měli něco říct o metodě trojcestného navázání TCP spojení. Každé připojení klient - server je nejdříve potřeba navázat právě skrze posloupnost paketů, z nichž každý má určitý příznak. Nejprve klient vyšle TCP paket s příznakem SYN. Od serveru pak očekává paket s příznaky SYN a ACK. Jakmile tento paket obdrží, vyšle potvrzovací paket s příznakem ACK a tím je spojení považováno za založené.



Obrázek 3.3: Trojcestné navázání TCP spojení [12].

Všechny útoky TCP probíhají na transportní vrstvě modelu ISO/OSI. V případě útoků **TCP SYN záplav** je pak na zařízení vysíláno nepřeberné množství TCP paketů s příznakem **SYN**. Zařízení, které tento paket přijme samozřejmě odešle paket s příznaky SYN-ACK, avšak odpovědi ACK se již nedočká. Jelikož však ACK pro server znamená, že připojení by mělo být navázáno, vyřadí si se systémových zdrojů určitou část právě pro toto budoucí připojení. Zatím je spojení tak otevřeno pouze částečně, úplné by bylo v případě obdržení paketu s příznakem ACK. Vzhledem k faktu, že těchto TCP SYN paketů je opravdu velké množství, vznikne zároveň i mnoho polovičně otevřených spojení a může tak dojít k vyčerpání prostředků zařízení oběti.

3.2.2 TCP RST záplavy

Příznak **RST** slouží v běžném provozu pro případ, že by z nějakého důvodu došlo k přerušení spojení mezi dvěma zařízeními, z nichž zařízení B o tomto přerušení neví a dál posílá A data. Jakmile však dojde k obnovení připojení A, začne opětovně přijímat pakety od B, o které už momentálně nemá zájem, vyšle zařízení B paket s příznakem RST. Tím poté ukončení spojení mezi oběma zařízeními.

V případě útoku však útočník zasílá pakety s příznakem RST, které se nijak nevztahují k předchozí TCP komunikaci. Zařízení oběti pak hledá v paměti, ke které TCP komunikaci by se mohl vztahovat a tím zabere část svého výpočetního výkonu. Opět však dochází zasílání velkého množství těchto RST zpráv, dochází tak zpravidla k vyčerpání zdrojů oběti.

3.2.3 Ostatní TCP záplavy

V případě ostatních TCP záplav už většinou nedochází k vyčerpání zdrojů zařízení, ale primárně slouží pouze k zahlcení síťového připojení. Předpokladem pro takový útok však je, že síťové připojení útočníka je rychlejší než oběti. Zasílány jsou v takových případech pakety s příznaky ACK, FIN atd.

3.2.4 UDP záplavy

UDP záplavový útok stejně jako předchozí TCP probíhá na transportní vrstvě. Na rozdíl od protokolu TCP, UDP spojení se nezakládá skrze určitou sekvenci paketů, ale je navázáno automaticky ihned. Díky tomu jsou útoky UDP o něco jednodušší. Zpravidla útok probíhá tak, že útočník generuje velký objem UDP paketů, které mají náhodný cílový port. Jakmile tento paket dojde na cílové zařízení, toto hledá jemu příslušnou aplikaci na daném portu. Jakmile zkontroluje, že na tomuto portu nepřísluší žádná aplikace, odpoví na něj pomocí zprávy ICMP Destination Unreachable, tedy že cílová destinace není dostupná. Zahlceno je tak nejen síťové připojení, ale také výpočetní zdroje zařízení.

3.2.5 ICMP záplavy

Protokol ICMP slouží k zasílání informací o tom, že některé zařízení není v síti dostupné. Tento útok probíhá na síťové vrstvě ISO/OSI modelu. Nejznámějším použitím je skrze nástroj `ping`, který posílá zprávu ICMP echo-request, skrze niž se dotazuje, zda je cílové zařízení dostupné. Očekává odpověď ICMP echo. V případě útoku ICMP Flooding na hostitelské zařízení vysíláno obrovské množství paketů se zprávou **ICMP echo-request**. Díky tomuto velkému množství tak dojde k zahlcení uzlu a zároveň také k možnému vyčerpání zdrojů, protože se zařízení oběti snaží odpovědět zprávou ICMP echo.

Kapitola 4

Výběr vhodné metody analýzy sítového provozu pro systém detekce anomálií

Následující kapitola se zabývá různými typy a metodami, kterými dochází k odhalování provozních anomálií v síťovém provozu. Každá z těchto metod má nejen své klady, ale také zápory, které je třeba při výběru metody dobře zvážit.

Nejdříve si vyjmenujeme v sekci 4.1 dva druhy IDS podle jejich umístění v síti. Dále se v sekci 4.2 se zabývám jednou z nejrozšířenějších metod, která se zabývá detekcí škodlivého provozu na principu porovnávání vzorů. Dalším celým souborem jsou metody založené na detekci anomálie. Těm se podrobně věnuji v sekci 4.3. Informace v této kapitole byly čerpány především z prací [3], [9] a [10].

4.1 Podle umístění

Existují 2 typy systému IDS, Host-based IDS (HIDS) a Network IDS (NIDS). První zmíněná varianta se stará o příchozí a odchozí síťový provoz probíhající nad jednou počítačovou stanicí, např. nad klientskou stanicí nebo serverem. Pokud by takový systém měl být nasazený pro složitější síť, bylo by velmi složité jej udržovat vzhledem k pravděpodobně většímu množství různých zařízení používajících různé operační systémy a s nimi tak různé modely chování. Ovšem díky HIDS lze rozpoznat i útok provedený přes šifrovaný kanál, který bohužel NIDS rozpoznat díky šifrování nemůže (dokáže to např. analýzou logů). Ovšem mezi největší jeho zápory patří možná napadnutelnost v rámci hostitelského zařízení nebo vyřazení možných DoS útokem [9].

Naproti tomu síťově orientovaná varianta (NIDS) dokáže vhodně monitorovat celý probíhající provoz v síti. Tento systém zpravidla běží na síťovém zařízení (dedikovaný server), které je vhodně umístěno v síti tak, aby skrze linku, na kterou je napojeno, procházela veškerá komunikace, kterou je takto třeba v síti vyšetřovat. Informace je pak schopen tento systém získávat ze síťových rozhraní v promiskuitním režimu. Nejvíce je však možno správnou funkčnost tohoto systému ovlivnit právě již zmíněným vhodným umístěním zařízení. Mezi jeho další klady tak potom patří také to, že tento systém prakticky neovlivňuje provoz sítě. Ovšem stinnou stránkou systému zůstává nemožnost zachytit šifrovaný provoz, složitá obtížnosti zpracování požadavků v sítích s velkým provozem a také ne zcela jednoznačná možnost určení, zda byl pokus o útok již dokončen, nebo pouze započat.

Existují také hybridní systémy IDS kombinující obě varianty a z nich vycházející klady i zápory. Tyto systémy jsou ovšem mnohem složitější na nasazení a jejich administraci.

4.2 Metoda porovnávání vzoru

Tato metoda je založena na databázi pravidel, podle nichž se IDS řídí. Toto řešení je poměrně jednoduché, ovšem ne příliš samospásné – je závislé na vložených pravidlech, podle kterých se systém chová. Například může existovat pravidlo, že jakmile se zdrojová i cílová adresa paketu sobě rovná, jedná se o útok. Zásadním problémem však je, že této metodě pak musíte pravidelně dodávat aktualizovanou databázi takovýchto pravidel, která někdo vytvořil podle již známých a objevených útoků – tato metoda neumí rozpoznat tzv. zero-day attack, tedy útok, který doposud ještě není známý a není tak zachycen v této databázi. Výhodou této metody, kromě její jednoduché implementace, je také spolehlivost rozpoznávání již dříve známých útoků. Příkladem typického komerčního IDS je pak Snort, který v současné době spravuje společnost Cisco[1] [9].

Toto řešení však není v našem případě až tolik vhodné. Více bych se zaměřil na zpracovávání průchozí komunikace pomocí detekce anomálie.

Tuto metodu lze dále rozdělit na 3 její podtypy. A to konkrétně na metodu založenou na statistické analýze, metodu strojového učení a metody založené na dolování dat.

V této práci bych se rád zaměřil především na metodu statistické analýzy, ovšem nejprve vysvětlím i variantu strojového učení a dolování dat.

4.3 Metoda založená na detekci anomálie

V případě IDS, který je založen na metodách detekce anomálie, je jeho funkčnost postavena na zcela jiných principech. Zatímco v předchozím případě jsme vůbec nemuseli mít informace o tom, jak síťový provoz vypadá, nemuseli jsme mít ani vypracovaný jeho normovaný model, v aktuálním případě jsou však tyhle informace nezbytné. Metody detekcí anomálie jsou založeny na principu, kdy porovnáváme jakýsi „normální“ model provozu s provozem aktuálním. Každý paket je pak detekován a následně je výpočtem určeno, zda-li tento paket ještě stále odpovídá našemu „normálnímu“ modelu, nebo se již jedná o anomálii.

Ideální systém detekce anomálie funguje tak, že každou anomálií je opravdu škodlivý paket, či rovnou sekvence. Množina anomálií se tedy rovná množině vniknutí. Ve skutečnosti tomu tak ovšem není. Pravidlem je, že existuje množina vniknutí, která je podmnožinou množiny anomálií. Tedy ne každá anomálie je vniknutí a systém tak generuje řadu planých poplachů.

Metod založených na detekci anomálie existuje několik. Věnovat se budu metodám strojového učení 4.3.1, neuronovým sítím 4.3.2, Bayesovským sítím 4.3.3, metodám shlukování 4.3.4, rozhodovacím stromům 4.3.6, Fuzzy logice 4.3.7 a nakonec statistické analýze provozu 4.3.8.

4.3.1 Metody strojového učení/dolování dat

Strojové učení je oblast umělé inteligence, která funguje na principu matematických modelů dat, kdy je systém schopen se zlepšovat v předpovídání podle svých předchozích zkušeností. Systém hledá v datech vzory, podle kterých pak dokáže definovat předpovědi. Čím vetší je

[1] https://www.snort.org

tedy množství dat, tím je systém přesnější. Systém je poměrně schopný přizpůsobovat se měnícím se podmínkám a stále podávat přesvědčivé výsledky [3].

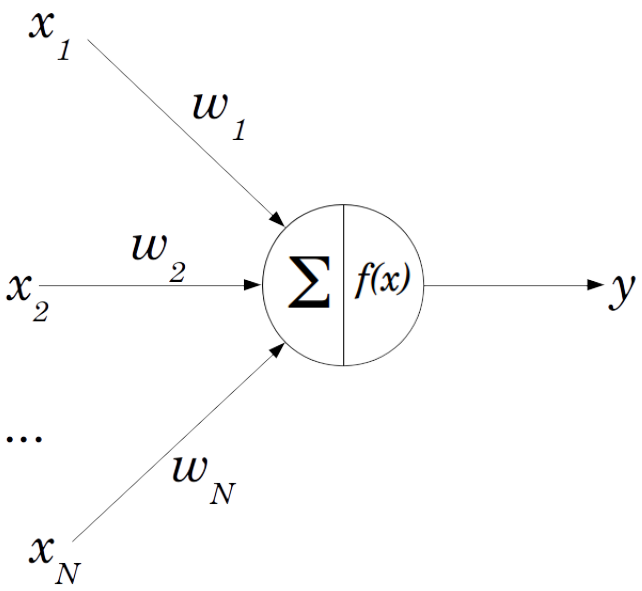
Podle toho, jaká vstupní data takovému systému jsme schopni dodat, liší se metody strojového učení na učení s učitelem a bez učitele.

Učení s učitelem dostane na začátku data, která jsou označkovaná. Systém tedy z dat pozná, který síťový tok je v pořádku, a který naopak je útokem. Podle toho se pak snaží vypracovat tak, aby další data rozpoznával podle stejných pravidel a výsledky byly tedy pokud možno totožné.

Naproti tomu učení bez učitele je princip, při kterém se systém pokouší v neoznačených datech najít správné struktury, třídy a vzory, podle kterých by se mohl rozhodovat, zda jsou data provozu anomálií, či nikoliv.

4.3.2 Neuronové sítě

Neuronové sítě vycházejí z biologické stavby člověka, konkrétně jeho mozku. Neuron je nervová buňka, která dokáže generovat signály na výstupu na základě vstupu. Umělé neurony fungují tak, že neurony v první vrstvě generují signály, které jsou poté zachytávány neurony ve vrstvě druhé [3]. Nejznámějším takovým umělým neuronem je perceptron [1].



Obrázek 4.1: Model perceptronu přijímajícího vstupy x s váhami w a výstupem y [11].

Perceptron funguje tak, že na svých vstupech přijímá podněty. Tyto podněty jsou dále ohodnoceny určitou váhou. Pokud suma všech těchto podnětů vynásobených danou váhou přesáhne určitou hranici/práh, pak je vygenerována předem daná hodnota, v opačném případě hodnota jiná. Tento práh je vyjádřen aktivační funkcí.

Perceptron je tedy definován dvěma základními vztahy. Tzv. lineární bázovou funkcí $\boldsymbol{u}$ 4.1, což je ona suma všech vstupů x s váhami w a dále skokovou aktivační funkcí $\boldsymbol{f(x)}$. 4.2 je příkladem předpisu aktivační funkce.

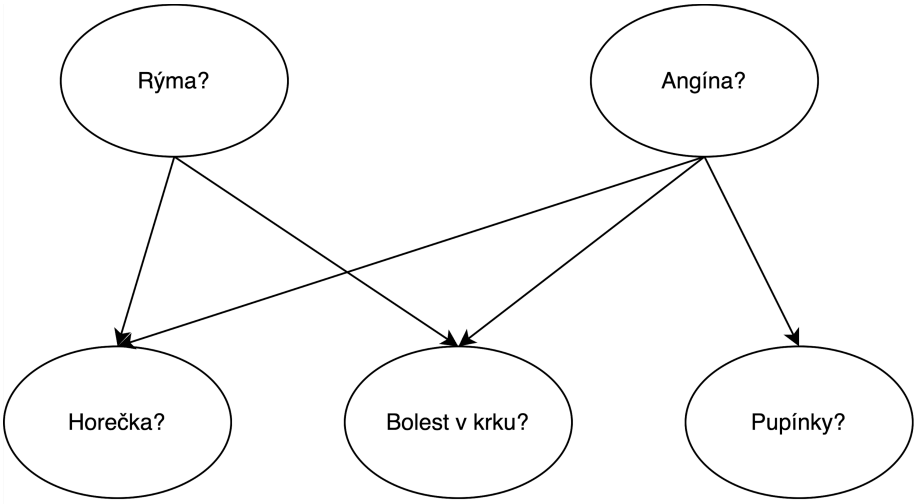
$$u = \sum_{i=1}^{n} w_i x_i \tag{4.1}$$

$$f(x) = \begin{cases} 1 & u \geq 0 \\ -1 & u < 0 \end{cases} \tag{4.2}$$

Neuronové sítě jsou však založené hlavně na své trénovací fázi. Tato bývá zpravidla dlouhá a náročná na výkon, což je jednou z nevýhod neuronových sítí. Kladem však je vysoká tolerance vůči nepřesným nebo nejasným datům a schopnost vytvářet řešení bez znalosti zákonitostí ve vstupních datech.

4.3.3 Bayesovské sítě

Bayesovská síť představuje grafický model pravděpodobnostních vztahu mezi jevy. Jedná se o acyklický orientovaný graf. Každý jeho uzel představuje náhodnou veličinu, pro každý je také vytvořena podmíněná pravděpodobnostní funkce. Graf obsahuje tzv. uzly rodiče a dítěte. Každý uzel dítěte je pak závislý na svém uzlu rodiče. V následujícím příkladu na obrázku 4.2 takové uzly rodiče vyjadřují *angína* a *rýma*, uzly dítěte *horečka*, *bolest v krku* a *pupínky*.



Obrázek 4.2: Příklad jednoduchého modelu Bayesovské sítě. Horečka může být způsobena rýmou nebo angínou, bolest v krku taktéž a pupínky způsobuje pouze angína [17].

Každý takový uzel pak obsahuje také tabulku podmíněných pravděpodobností, která obsahuje informace o pravděpodobnosti, že se uzel bude nacházet v určitém stavu [15].

Bayesův vztah má podobu vzorce

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{4.3}$$

kde P(A|B) značí podmíněnou pravděpodobnost jevu A na základě vzniku jevu B, P(B|A) podmíněnou pravděpodobnost jevu B, jestliže nastal jev A, P(A) je pravděpodobnost jevu A a P(B) pravděpodobnost jevu B.

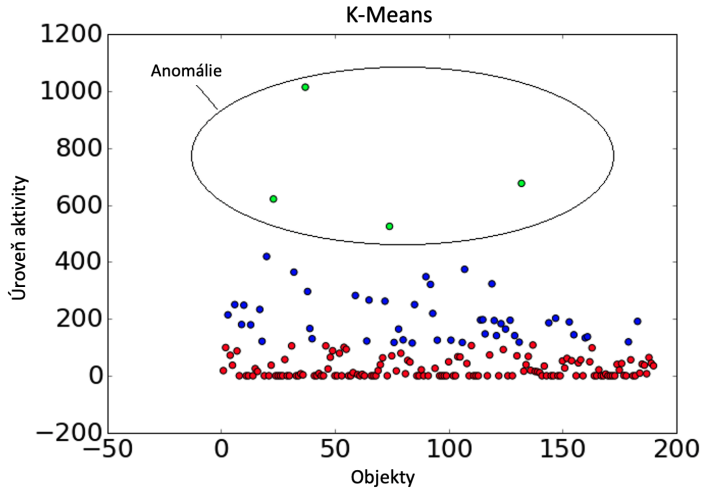
4.3.4 Metody shlukování

Metody shlukování jsou založeny na hledání vzorů v neoznačkovaných datech. Jedná se tedy o metodu učení bez učitele. Základem je seskupování dat na základě jejich podobností. Existují dva přístupy k tomuto řešení. Zmínil jsem, že se zpravidla jedná o metodu učení bez učitele. Jsou však zde i varianty tzv. posilovaného učení. Systém v takovém případě má k dispozici množinu dat označených jako „normální“, ze kterých si vytvoří „normální“ model provozu v síti [9].

Obecně se toto řešení zakládá na pravidle, že normální provoz v síti tvoří vetší množina dat. Větší shluk tedy musí představovat množinu „normálních“ dat, menší shluky poté množinu anomálií, či rovnou útoků – tyto shluky leží mimo shluk „normální“.

4.3.5 K-Means

Algoritmus iterativně přepočítává těžiště shluků. V každém kroku jsou shlukované objekty přidány právě do toho sluku, k jehož těžišti je vzdálenost nejmenší. Algoritmus tyto kroky opakuje až do stavu, kdy jsou všechny shluky ustáleny. Velké shluky jsou považovány za normální provoz, zbytek prvků, které leží mimo ně, je považován za anomálii, viz obrázek 4.3 [9].

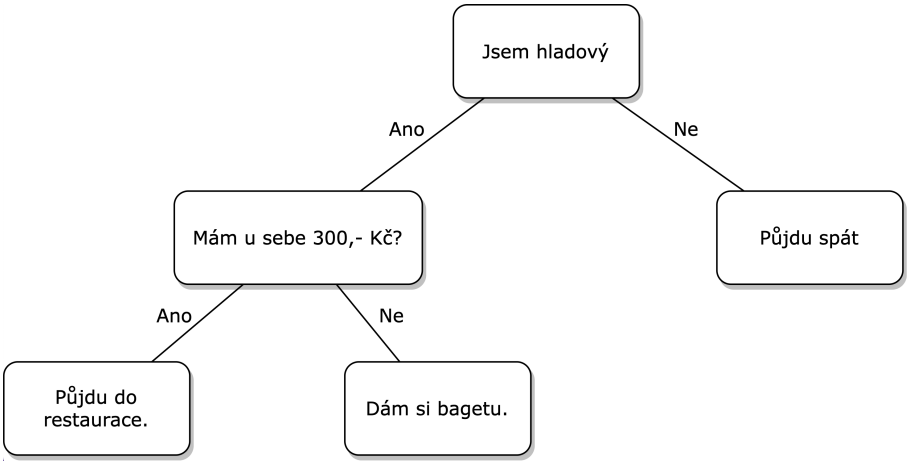


Obrázek 4.3: Příklad grafu znázorňujícího normální shluk a mimo ležící anomálie. Zpracováno algoritmem K-Means [6].

4.3.6 Rozhodovací stromy

Rozhodovací stromy, jako jedna z metod dolování dat, slouží k popisování objektů na základě hodnot jejich vlastností. Algoritmus funguje na základě pokládání otázek, na které odpovídáme dvěma způsoby. Podle toho se pak takovýto uzel štěpí na dva různé podstromy.

Tato metoda má řadu výhod i nevýhod. Metoda je poměrně časově úsporná, neboť dokáže zpracovat velké množství dat za poměrně malou jednotku času. Ovšem některé problémy je velmi těžké pochopit, a proto je i zpracovat do podoby rozhodovacího stromu. Jedná se také o metodu učení s učitelem, takže je zapotřebí trénovací množina dat.

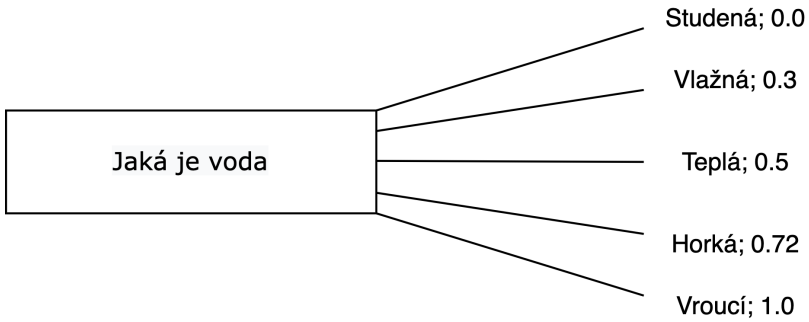


Obrázek 4.4: Příklad rozhodovacího stromu, kde se rozhoduje, zda jít na oběd [18].

4.3.7 Fuzzy logika

Metoda vychází z předpokladu, že každou neostrou hodnotu (která nenabývá pouze hodnot 1 a 0), můžeme vyjádřit mírou příslušnosti. Může tedy na rozdíl od výrokové logiky, která využívá pouze hodnot 0,1, využívat jakoukoliv hodnotu z intervalu <0,1> [3].

Uveďme si příklad na vnímání teploty vody. Podle výrokové logiky by existovala jen voda studená a teplá. Představovaná hodnotou 0 a 1. Z principů fuzzy logiky však můžeme zahrnout daleko více možností. Můžeme například uvažovat možnosti, že je voda studená, vlažná, teplá, horká a vroucí – každou z nich poté ohodnotit určitou hodnotou právě z intervalu <0,1>.



Obrázek 4.5: Příklad grafu znázorňujícího vnímání teploty vody metodami fuzzy logiky [7].

4.3.8 Metoda statistické analýzy

Základem této metody je dlouhodobé sledovaní provozu v síti. Tím si systém vytvoří jakýsi popis normálního chování zařízení v síti a podle porovnávaní s aktuální situací dokáže rozpoznat anomálii.

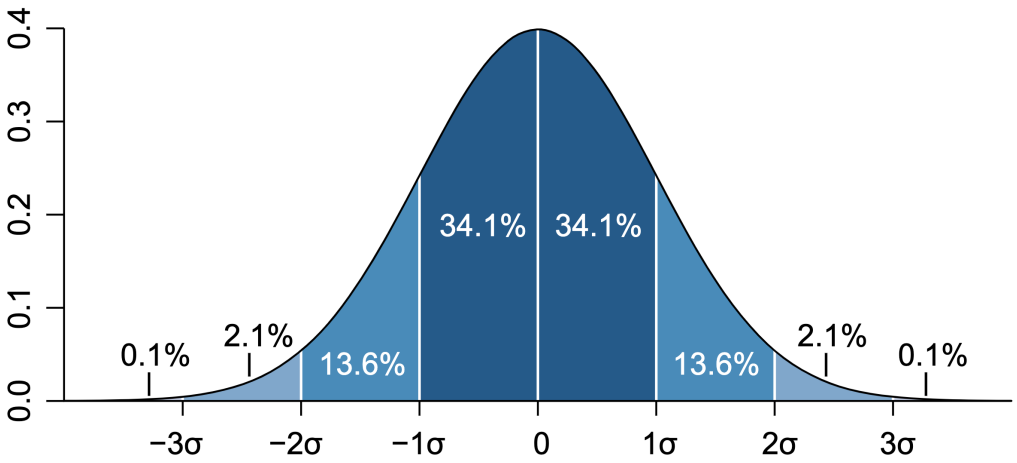
Systém má k dispozici data dlouhodobého chování systému a jeho statistické údaje. Zároveň je hlídána i jakási aktuální situace, aktuální chování. Dochází tak k porovnání aktuální a „normální“ situace a jakmile dojde k překročení určité hranice (např. bude

pozorován dvojnásobný nárůstek DNS dotazů), dojde vyvolání upozornění o anomálii. Je vypočítáváno skóre anomality. Obrovskou výhodou tohoto systému je fakt, že nepotřebuje znát žádné údaje o průběhu útoku, tak jako je tomu u metody signatur. Naopak velkou nevýhodou je pak fakt, že se zde může vyskytnout značné množství falešných zpráv. Pokud také útočník zná alespoň základní údaje o síti, může si ten tzv. normální stav svým způsobem vycvičit, postupně bude po malých krocích upravovat tok v síti tak, až nakonec jeho útočná fáze nebude k rozpoznání od jeho „normální“.

Zásadní je také správné určení hranice anomality, neboť v případě, že je hranice příliš nízko, bude docházet k častému zasílání falešných zpráv. Naopak v případě, že bude nastaveno příliš vysoko, může dojít k nerozpoznání již probíhajícího útoku.

Statistické metody se dále dělí na modely univarietní, multivarietní a na časové řady [9].

Z počátku se nejčastěji používaly modely univarietní, založené na Gaussově normálním rozložení, znázorněném na obrázku 4.6, které určovalo rozsah přijatelných hodnot jednotlivých proměnných v provozu. Toto rozložení se řídí dvěma parametry – střední hodnotu a rozptylem. Později se však přešlo na multivarietní, které berou v potaz více dimenzí.



Obrázek 4.6: Typický graf normálního rozložení se znázorněnými násobky standardní odchylky σ [21].

V současné praxi se však k výpočtům nejčastěji používají časové řady, neboť potřebujeme data zaznamenávat ve předem určených časových intervalech. V daném intervalu je pak podle normálního modelu vypočtena pravděpodobnost výskytu dané proměnné. Pokud je tato pravděpodobnost příliš nízká, proměnná je označena za anomálii.

Nejčastěji sledovanými proměnnými síťového provozu jsou pak například počty paketů daného protokolu v čase, délky trvání spojení či počet IP adres.

Jedním z příkladů statistické analýzy založené na modelu normálního rozložení je řešení zakládající se na pravidle 6σ, které vychází z práce [10]. Ze zaznamenaného provozu určíme střední hodnotu proměnné, kterou budeme potřebovat pro další výpočet. Dále také vypočteme směrodatnou odchylku podle vzorce

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{4.4}$$

kde N značí celkový počet měření, x naměřené hodnoty a μ aritmetický průměr.

Vzorec UCL 4.5 (Upper Control Limit - horní limit propustnosti) nám určí hranici (hodnotu), nad kterou se již jedná o anomálii a je tak vyvolána výjimka. Vozrec LCL 4.6 (Lower Control Limit - spodní limit propustnosti) naopak nejnižší hranici, pod kterou se již jedná o anomálii. Metoda 6σ pak propustí až 99.9999998027% [24] běžné komunikace, přičemž zbylý provoz označí za anomálii. Pokud bychom použili na stejném principu metodu 3σ, pak by propustnost této metody byla 99.73%.

$$UCL = \mu + \frac{12\sigma}{\sqrt{N}} \tag{4.5}$$

$$LCL = \mu - \frac{12\sigma}{\sqrt{N}} \tag{4.6}$$

kde μ značí aritmetický průměr, naměřené hodnoty a σ směrodatnou odchylku a N počet měření.

Příklad použití statistické metody IDS

V našem případě, kdy je zapotřebí rozpoznávat škodlivé počítače, můžeme tuto metodu velice dobře využít. Můžeme si ukázat na příkladu internetové kavárny.

Tato kavárna má otevírací dobu od 8 do 20 hodin, přičemž předpokládejme, že počítače uživatelé zpravidla využívají k surfování po internet, vyhledávání videí a čtení mailů. Hned z první informace můžeme tušit, že po 20. hodině nebude v síti již žádný provoz. Pokud tedy náš systém zachytí jakoukoliv komunikaci, zařízení, které se této komunikace zúčastnilo můžeme automaticky považovat za škodlivé.

Dále můžeme uvažovat, že by ideálně nemělo docházet k přenosu velkých objemů dat. Pokud se tak stane, předpokládejme, že buď se snaží uživatel stahovat či odesílat nelegální soubory, nebo je takovéto chování zapříčiněno vnějším přístupem škodné.

Dalším takovým případem může být snaha o přihlášení do systému, ve kterém jsou uživatelé kavárny registrováni tak, aby bylo možno počítač používat. Řekněme, že pokud dojde k několikanásobnému neúspěšnému pokusu o přihlášení se do systému, považujeme tuto činnost o pokus útočníka proniknout do systému.

Velmi výhodné využití má pak pro odhalení útoků odepření služby (DoS) a jeho distribuované varianty (DDoS). V takovém případě dochází často k vysílání nesmyslných a obsáhlých dat sítí tak, aby došlo k zahlcení zařízení a jejich následného odstavení. Častou jsou zasílány pakety ICMP, UDP či jiné jim podobné s tak velkými daty, že následně zahltí právě konečná zařízení. A právě včasné odhalení takového provozu dokáže ochránit naše zařízení a zamezit tak případné ztrátě našich dat.

Kapitola 5

Vybrané nástroje IDS

V následující kapitole se budu věnovat popisu vybraných IDS nástrojů a jejich využití pro můj účel hledání anomálií v síťovém provozu.

Nejprve se budu v sekci 5.1 věnovat velmi rozšířenému programu **Snort**[1] společností Cisco. V následující sekci 5.2 pak uživatelsky rozšiřitelnému **Zeeku**[2], který lze programovat díky vlastnímu skriptovacímu jazyku, a nakonec v sekci 5.3 také velmi oblíbenému programu **Suricata**[3] vyvíjenému sdružením OISF (Open Internet Security Foundation), který jsem si vybral pro svou bakalářskou práci, a který jsem se pokusil svým vlastním modulem rozšířit o další modul detekce.

5.1 Snort

Jedním z nejzásadnějších programů na poli systémů detekce průniků je Snort vyvíjený společností **Cisco**. Snort je typických příkladem programu založeném na detekci škodlivého provozu na základě pravidel. Ačkoliv této technologii se ve své práci nevěnuji, Snort lze však využívat i jinými způsoby.

Snort lze v základním stavu spouštět ve 3 módech [19]:

- **Sniffer mód** - základní mód, ve kterém jsou čteny pakety procházející sítí a následně vypisovány do konzole.
- **Packet Logger mód** - ukládá pakety do souboru.
- **Network Intrusion Detection System mód** - v tomhle režimu Snort obstarává funkci systému detekce průniků.

V rámci mé práce také zkoumám možnost těchto nástrojů, kdy program pouze zaznamenává informace o probíhajících komunikacích v rámci síťových toků, nikoliv pouze paketů, jako to umí většina z nich. Standardní verze Snortu tuhle možnost nemá. Existuje však rozšíření, které by mohlo v tomhle směru být více nápomocno. Konfigurace Snortu je možná změnou souboru "snort.lua".

[1] https://snort.org
[2] https://zeek.org
[3] https://suricata.io

5.1.1 OpenAppID

Od roku 2014 je volně dostupné rozšíření základní funkčnosti Snortu pod názvem OpenAppID. Toto rozšíření je schopno monitorovat a zaznamenávat využívání sítě jednotlivými aplikacemi. To ze Snortu pak dělá volně dostupný aplikační firewall. Dokáže tak detekovat škodlivé aplikace, omezit využití sítě aplikací a další. V současné době obsahuje přes 3000 jednotlivých detektorů.

Kromě toho jej však můžeme využít pouze jako program, který nám vytvoří záznam o provozu v síti, který taktéž můžeme použít pro další analýzu.


```
{
    "session_num": "0.31",
    "pkt_time": "2018−08−11 06:15:10.591407",
    "pkt_num": 181,
    "apps": {
            "service": null,
            "client": "DNS",
            "payload": null,
            "misc": null,
            "referred": null
    },
    "proto": "UDP",
    "client_info": {
            "ip": "192.168.1.95", "port": 54513, "version": null
    },
    "service_info": {
            "ip": "192.168.1.6",
            "port": 53,
            "version": null,
            "vendor": null
    },
    "user_info" : {
            "id": 0,
            "username": null,
            "login_status": "n/a"
    },
    "tls_host": null,
    "dns_host": "_ldap._tcp.Sputnikhouse−DC.sputnikhouse.org",
    "netbios_info": {
            "netbios_name": null,
            "netbios_domain": null
    },
    "http": {
            "http2_stream": null,
            "host": null,
            "url": null,
            "user_agent": null,
            "response_code": null,
            "referrer": null
    }
}
```


Výpis 5.1: Příklad síťového provozu vygenerovaného modulem AppID Listener rozšíření OpenAppID Snortu.

Tyto záznamy se však zabývají hlavně síťovým provozem na straně aplikací, což pro mou další práci není příliš vhodné.

5.2 Zeek

Zeek (dříve známy pod názvem Bro) je komplexní systém sloužící analyzování síťového provozu. Nejvíce je využíván pro svou funkci odchytávání škodlivé komunikace. Ačkoliv stejně jako výše zmíněný Snort dokáže odhalovat škodlivý provoz na základně signatur, jeho IDS funkce je založena také na principech umělé inteligence a behaviorální analýzy [20].

V současné době je velmi podstatnou výhodou Zeeku jeho uživatelská rozšiřitelnost. Pro Zeek je totiž možno psát skripty v jeho vlastním skriptovacím jazyce a lze jej tak velmi dobře a poměrně jednoduše rozšířit o další detektory, kterými třeba sám nedisponuje.

Zároveň je opět možné využít funkce zaznamenávání síťového provozu. V případě Zeeku už však je možno zaznamenávat také data podobná síťovým tokům.


```
{
    "ts":1533964509.288025,
    "uid":"CFFTJi1HDxeuce9Q6l",
    "id.orig_h":"192.168.1.95",
    "id.orig_p":49161,
    "id.resp_h":"192.168.1.6",
    "id.resp_p":389,
    "proto":"tcp",
    "duration":0.007348060607910156,
    "orig_bytes":2161,
    "resp_bytes":2582,
    "conn_state":"RSTR",
    "missed_bytes":0,
    "history":"ShADdar",
    "orig_pkts":6,
    "orig_ip_bytes":2413,
    "resp_pkts":5,
    "resp_ip_bytes":2794
}
```


Výpis 5.2: Příklad síťového provozu vygenerovaného Zeekem v souboru "conn.log".

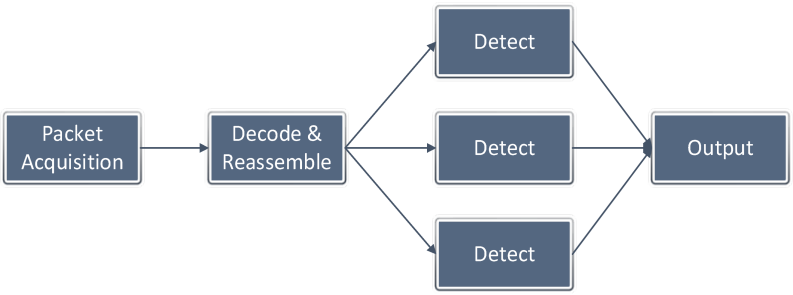
Jednotlivé součásti záznamu pak obsahují časovou značku začátku toku, počty přenesených bajtů a paketů, zdrojové a cílové IP adresy a porty, délku trvání toku, stav spojení (TCP příznaky), historii stavu spojení a další. Je zde tak velmi mnoho užitečných informací, pomocí kterých lze detekovat anomální chování provozu. Zeek zaznamenává toky obousměrné, obsahující přenos jak ze zdroje do cíle, tak i zpětnou odpověď.

5.3 Suricata

Dalším velmi rozšířeným projektem, který se zabývá analýzou síťového provozu, je Suricata vyvíjena organizací Open Internet Security Foundation (OISF) [16].

Suricata je komplexní program s více vláknovou architekturou, který zastává funkci IDS/IPS. Napsán je v jazyce C.

Základní architektura Suricaty se skládá ze 4 modulů. V prvním modulu s názvem `Packet aquisition` dochází ke čtení paketů ze síťového provozu. Ty následně pokračují do dalšího modulu, kterým je `Decode module`. V tomto modulu dochází ke zpracovávání paketů do formy síťových toků, probíhá zde kontrola toho, zda proběhlo vše potřebné k úspěšnému navázání toku. Pokud je vstupní komunikace formou paketů protokolu TCP, dochází k rekonstrukci původního toku dat. Nakonec je zde také kontrolována aplikační vrstva (protokoly HTTP, DCE/RPC). Nyní je síťový provoz předán dalšímu modulu, který patří k těm nejvíce vytíženým v celém programu. Může však běžet paralelně v několika vláknech. Tímto modulem je `Detect`. Ten obstarává samotnou detekci škodlivého provozu na základě databáze pravidel. Nakonec je pak volán modul `Output`, který zpracovává všechny hlášky a upozornění o zpracování daného provozu.

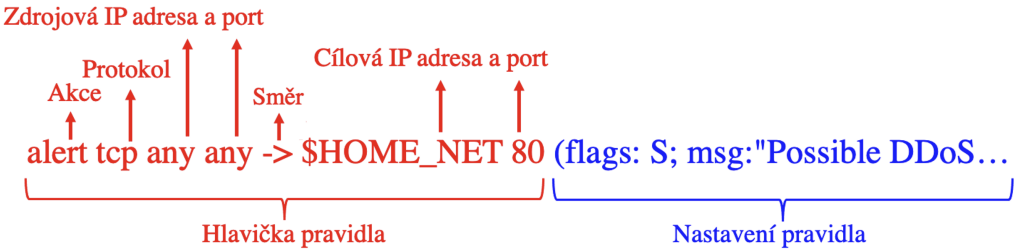


Obrázek 5.1: Architektura programu Suricata [2].

Pravidla, podle kterých je rozpoznáván škodlivý provoz jsou psána způsobem, který lze vidět ve na obrázku 5.2.

Kromě již zmíněné detekce škodlivého provozu, jej však lze také využít jako aplikaci pro analýzu síťového provozu, ze kterého jsou vygenerovány podrobné statistiky, záznamy určitých typů přenosu (HTTP, TLS, atd.) nebo je schopen generovat záznamy o tocích dvou druhů - obousměrný formát *Flow* a také jednosměrný standart *NetFlow*.

Co se týče běhu v módu IDS/IPS, je tento implementován technologií porovnávání signatur, jak bylo již v předchozích odstavcích zmíněno. Obsahuje tedy stejně jako Snort a Zeek aktualizovanou databázi pravidel, podle kterých je škodlivý provoz vyhodnocován.



Obrázek 5.2: Neúplný příklad pravidla Suricaty s grafickým rozdělením podle částí.

Každé pravidlo je rozděleno na část popisující hlavičku pravidla a na část nastavující pravidlo. Prvním polem hlavičky je akce, která se má vykonat. `alert` značí vyvolání upozornění, pokud je současný provoz shodný s pravidlem. Akcí však může být také akce `pass`, která provoz povolí, nebo `drop`, která naopak provoz zahodí. V další části jsou pak defino-

vány IP adresy a porty včetně směru, jakým má provoz proudit. Tato část může obsahovat i předem definované proměnné v konfiguračním souboru (př. $HOME_NET). Nastavení pravidla pak může obsahovat mnoho informací, které se v rámci probíhající komunikace srovnávají. Může obsahovat informaci o příznacích, různé vzory, které jsou v komunikaci hledány, nebo také definici zprávy, která je při nalezení nevhodné komunikace generována.

V našem případě však chceme využít hlavně módu, který zpracovává surový síťový provoz a zpracovává z něj informace o tocích formátu NetFlow. Suricatu je však pro tento mód potřeba nejdříve správně nakonfigurovat.

5.3.1 Konfigurace Suricaty

Konfigurování běhu Suricata probíhá prostřednictvím souboru "suricata.yaml". Pokud chceme zapnout funkci zaznamenávání NetFlow toků, je třeba v sekci "eve-log"odstranit znak komentáře ('#') u možnosti "netflow", ostatní možnosti pak tímto znakem komentáře označit. Tím je při následném spuštění programu automaticky spuštěn i modul ukládající Netflow toky, jejichž záznamy jsou následně ukládány do souboru "eve.json"v textovém formátu JSON.


```
{
    "timestamp":"2018−08−01T21:44:32.834262+0200",
    "flow_id":1886985540974944,
    "event_type":"netflow",
    "src_ip":"86.209.164.209",
    "src_port":1421,
    "dest_ip":"172.27.224.250",
    "dest_port":502,
    "proto":"TCP",
    "netflow":{
            "pkts":1,
            "bytes":174,
            "start":"2018−08−01T21:49:49.178528+0200",
            "end":"2018−08−01T21:49:49.178528+0200",
            "age":0,
            "min_ttl":64,
            "max_ttl":64
    },
    "tcp":{
            "tcp_flags":"02",
            "syn":true
    }
}
```


Výpis 5.3: Příklad záznamu NetFlow v souboru "eve.json".

Vidíme, že záznam obsahuje velké množství informací, ze kterých lze DDoS útoky odhalovat. Kromě zdrojových a cílových IP adres a portů, také typ protokolu, počet přenesených paketů, velikost toku v bytech, časovou značku počátku a konce toku a také velmi důležité příznaky TCP protokolu. Záznam je tedy velmi podobný tomu ze Zeeku. Na rozdíl od toku, který prezentuje Zeek, je však NetFlow tok jednosměrný. Zaznamenává tedy komunikaci jedním směrem, odpověď na ni je představena již záznamem dalšího tokem. Obousměrný tok lze v Suricatě taktéž zaznamenat, v souboru "suricata.yaml"je třeba vybrat přepínač "-flow".

Vzhledem k velmi užitečným informacím, které NetFlow záznam obsahuje, jsem se rozhodl využít právě zpracování NetFlow toků Suricaty pro práci na detektoru DDoS útoků.

Kapitola 6

Návrh a implementace

Následující kapitola se zabývá použitými technologiemi 6.1, návrhem 6.2 a implementací modulu pro odhalování DDoS útoků v síťovém toku v sekci 6.3.

6.1 Použité technologie

Finální řešení bylo vytvořeno v jazyce C s využitím knihoven a modulů projektu Suricata ve verzi 6.0.4[1]. Výstup modulu je ve formátu JSON. Pro práci s verzemi programu bylo využito nástroje Git. Program byl testován na operačním systému Ubuntu verze 20.04.4.

6.2 Návrh

Návrh řešení byl vytvořen v jazyce Python.

Pro prvotní návrh řešení jsem využil výstupního souboru Suricaty "eve.json", který jsem pomocí jednoduchého kódu v Pythonu načetl a vytvořil z něj slovník. Tento slovník jsem následně seřadil vzestupně podle času začátku toku a dále jej pak zpracoval. Rozdělil jsem jej podle intervalů na časová okna a v nich počítal výskyt toků jednotlivých typů protokolů. Z celkových statistik jsem pak vypočítal pomocí pravidla X-σ horní limit počtu toků v každém okně. Pokud počet v jednotlivém okně přesáhne tuto hodnotu, je vyvoláno upozornění na možný útok. Tento kód následně sloužil jako jakási šablona pro následné vytvoření již plnohodnotného modulu Suricaty napsaném v jazyce C.

6.3 Implementace modulu pro Suricatu

Vzhledem k faktu, který jsem zmínil v předchozí kapitole, že Suricata dokáže velmi dobře analyzovat síťový provoz, ze kterého vygeneruje záznamy toků NetFlow, vybral jsem si tento projekt jako základ, pro který vytvořím rozšiřující modul detekující možné DDoS útoky v síťovém provozu.

6.3.1 Modul OutputJsonDoS

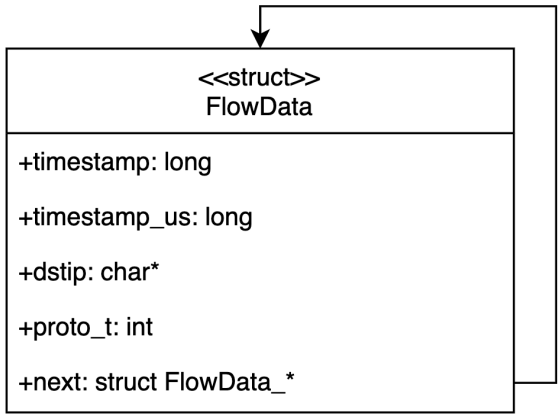
Při tvorbě samotného modulu jsem pak vycházel z výstupního modulu Suricaty, který obstarává zápis informací o NetFlow tocích do souboru "eve.json"ve formátu JSON. Modul je tvořen zdrojovými soubory "output-json-netflow.c"a "output-json-netflow.h".

[1] https://www.openinfosecfoundation.org/downloads/suricata-6.0.4.tar.gz

Vstupními daty modulu jsou již zpracované datové struktury obsahující podrobné informace o přenesených tocích formátu NetFlow.

V tomto modulu jsou postupně procházeny všechny toky vstupního síťového provozu, které jsou následně ukládány ve formátu JSON do výstupního souboru. Zpracování těchto toků probíhá vícevláknově, je proto třeba dát pozor na vzájemné přepisování paměti paralelně běžícími vlákny. V popisovaném modulu je takovým místem výstupní soubor "evelog.json".

Funkci `JsonNetFlowLogger`, ve které dochází k postupnému procházení informací obsažených v datové struktuře `Flow` popisující tok, jsem využil a upravil tak, abych tyto informace uložil do vlastního jednosměrně vázaného seznamu tvořeného datovými strukturami FlowData. Upravenou funkci podle názvu modulu přejmenoval na `JsonDoSLogger`. Ukládání informací o toku dochází ve dvou funkcích, které jsou velmi podobné. `FlowSaverToClient` ukládá data toku, který směřuje z serveru ke klientovi, `FlowSaverToServer` naopak od klienta k server. Tato funkce je ukázána ve výpisu 6.1. Nejprve se však podívejme na strukturu FlowData popisující tok 6.1.



Obrázek 6.1: Datová struktura FlowData obsahující informace o toku.

Popis obsahu struktury je následující:

- `timestamp` je proměnná obsahující časovou značku začátku toku v sekundách.
- `timestamp_us` obsahuje doplněk časové značky začátku toku v milisekundách.
- `dstip` je ukazatel do paměti, kde je uložen textový řetězec popisující cílovou IP adresu toku.
- `proto_t` je proměnnou, která obsahuje informace o typu protokolu, v případě protokolu TCP pak hodnotu TCP příznaků.
- `next` je ukazatelem do paměti, kde je uložen další záznam rámce datové struktury `FlowData`.

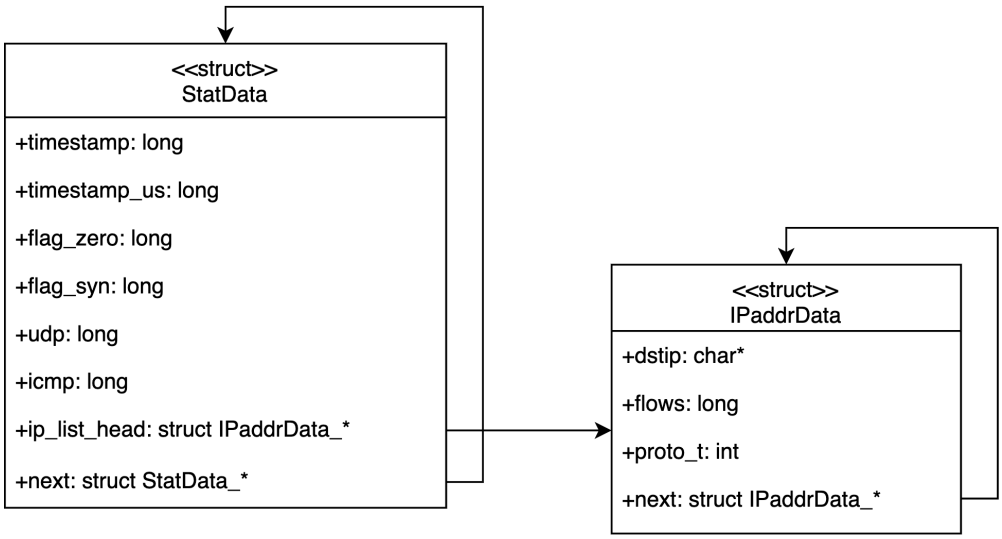
```
59 static void FlowSaverToServer(Flow *f)
60 {
61     FlowDataCurr->timestamp = f->lastts.tv_sec;
62     FlowDataCurr->timestamp_us = f->lastts.tv_usec;
```

```
63
64      FlowDataCurr->dstip = SCMalloc(46);
65      //alloc memory for destination IP address
66
67      if(unlikely(FlowDataCurr->dstip == NULL))
68      {
69          SCLogDebug("Malloc ERROR");
70          exit(1);
71      }
72
73      IPaddrLoader(f, &FlowDataCurr->dstip, 0);
74
75      if (f->proto == IPPROTO_TCP)
76      {
77          TcpSession *ssn = f->protoctx;
78          FlowDataCurr->proto_t = (ssn ? ssn->client.tcp_flags : 0);
79      }
80      else if(f->proto == IPPROTO_ICMP || f->proto == IPPROTO_ICMPV6)
81      {
82          FlowDataCurr->proto_t = ICMP_P;
83      }
84      else if(f->proto == IPPROTO_UDP)
85      {
86          FlowDataCurr->proto_t = UDP_P;
87      }
88
89      FlowDataCurr->next = createFlowData(0);
90      FlowDataCurr = FlowDataCurr->next;
91  }
```

Výpis 6.1: Kód implementované funkce pro výpočet horního limitu propustnosti.

Jakmile už není k dispozici žádný nový tok, je zavolána funkce `DoSDetector`, nyní běžící už pouze jednovláknově, která tyto záznamy o tocích zpracovává. Protože toky, které jsou ukládány, nejsou seřazeny postupně, je třeba je nejdříve seřadit vzestupně dle časové značky začátku toku. K seřazení je využito algoritmu `QuickSort`. Výběr vhodného algoritmu byl však zásadní pro správné fungování programu, proto bych se mu rád věnoval v samostatné podsekci 6.3.2. Jakmile je seznam obsahující informace o tocích seřazen, z toků jsou vytvořeny časové rámce, které obsahují statistické informace o přenosu toků v daném rámci - časové značky začátku a konce toku, počet toků TCP s příznakem 0 (ZERO), počet TCP toků s příznaky SYN a SYN-ACK, počet toků UDP a ICMP. Z počtu toků v každém takovém časovém rámci je následně vypočítána horní hranice propustnosti 4.5, která představuje maximální hodnotu toků v daném rámci. Počítána je funkcí `calcUCL`. Jestliže je toků v tomto časovém rámci více, než je dáno vypočítanou hranicí, dojde k upozornění na anomálii.

Na každý z těchto časových rámců rovněž navazuje seznam cílových IP adres toků. Z tohoto seznamu je pak vybrána ta adresa, která se pro daný typ toku vyskytuje nejvíce. Tím je poukázáno na možnou IP adresu zařízení, na které je útočeno. Tato datová struktura



Obrázek 6.2: Datové struktury StatData a IPaddrData.

pak obsahuje danou cílovou IP adresu, počet toků stejného typu, které mají tuto IP adresu společnou. Obě na sebe navazující struktury jsou pak prezentovány na obrázku 6.2.

Obsahem datové struktury `StatData` jsou:

- `timestamp` je proměnná obsahující časovou značku začátku časového rámce v sekundách.
- `timestamp_us` obsahuje doplněk časové značky začátku rámce v milisekundách.
- `flag_zero` je proměnná obsahující informaci o počtu toků protokolu TCP s příznakem 0 v daném rámci.
- `flag_syn` obsahuje počet TCP toků s příznaky 2 (SYN) nebo 18 (SYN-ACK).
- `udp` obsahuje počet UDP toků.
- `icmp` je proměnnou s informací o počtu ICMP toků.
- `ip_list_head` je ukazatelem do paměti, kde je uložen první záznam seznamu vyskytujících se IP adres, seznam je tvořen datovými strukturami typu `IPaddrData`.
- `next` je ukazatelem do paměti, kde je uložen další záznam rámce datové struktury `StatData`.

Strukturu `IPaddrData` tvoří:

- `dstip` je ukazatelem do paměti - na textový řetězec popisující cílovou IP adresu toku.
- `flow` je počet toků, které danou cílovou adresu obsahují.
- `proto_t` je opět proměnná odlišující toky podle typu protokolu, nebo TCP příznaků. Slouží pro odlišení záznamu počtu toků obsahujících danou IP adresu dle tohoto typu.
- `next` je ukazatelem na další záznam seznamu tvořeného strukturami `IPaddrData`.

Kód funkce CalcUCL, počítající horní hranici propustnosti, je prezentován výpisem 6.2. Parametry funkce jsou proměnné `flow_sum` představující součet všech toků, `win_sum` jako součet všech časových rámců a `*head` je ukazatel na první záznam v seznamu časových rámců struktury `StatData`. Kód taktéž obsahuje definované makro `SIGMA` představující hodnotu násobící směrodatnou odchylku σ.

```
59 double CalcUCL(long flow_sum, long win_sum, StatData *head){
60     if(win_sum == 0)
61         return -1.0;
62 
63     double avg = all_count/win_sum;
64     double dev = 0; //deviation
65 
66     StatData *current = head;
67     while(current->next)
68     {
69         dev += (current->flows - avg) * (current->flows - avg);
70 
71         current = current->next;
72     }
73 
74     double varc = dev/win_sum; //variance
75     double std_dev = sqrt(varc); //standard deviation
76 
77     return (double)avg + ((SIGMA * std_dev)/sqrt(win_sum));
78 }
```

Výpis 6.2: Kód implementované funkce pro výpočet horního limitu propustnosti.

V původní verzi mého programu jsem horní hranici propusti počítal pro všechny výše zmíněné typy toků (ICMP, UDP, TCP-SYN a TCP-ZERO toky). Následně jsem porovnával aktuální počet toků v rámci s touto hodnotou pro daný typ toku. Toto však generovalo velké množství falešných zpráv. Důvodem bylo, že pokud se např. v našem 6hodinovém PCAP záznamu provozu vyskytnou pouze čtyři toky protokolu ICMP, horní hranice propustnosti bude menší než 1. V případě tedy, že se v našem časovém rámci bude vyskytovat pouze jeden jediný ICMP tok, dojde k vyvolání upozornění na možný DDoS útok, což je ale při této hodnotě naprosto špatně.

Upravil jsem tedy kód programu tak, aby byla horní hranice počítána pouze z počtu všech toků, tedy bez rozpoznávání typu. Až v situaci, kdy modul kontroluje počty aktuálně vyskytujících se toků, je kontrolován počet každého typu toku s touto hodnotou. Ošetřeno je tak tím to, že daný typ protokolu musí tvořit podstatnou většinu všech přenášených toků.

6.3.2 Výběr algoritmu řazení

Vzhledem k faktu, že jsou počty toků, které jsou ze síťového provozu generovány, velké, tvoří tak velký jednosměrně vázaný seznam, který musím seřadit dle začátku toku. Algoritmus, který jsem pro řazení použil musí být jednak dostatečně rychlý, aby výpočet netrval příliš dlouho, nesmí být ovšem také příliš paměťově náročný. Údaje o paměťové a časové náročnosti algoritmů byly čerpány z práce [5].

BubbleSort - prvním algoritmem, který jsem zkoušel použít, byl jednoduchý algoritmus BubbleSort, který jen postupně několikrát prochází seznam a porovnává hodnoty aktuálního a následujícího uzlu seznamu, uzly jsou navzájem vyměněny. Ačkoliv tento algoritmus je paměťově nenáročný, jeho běh však zabírá velmi dlouhou dobu. Typicky je tato časová složitost $O(n^2)$, kde n značí počet uzlů seznamu. Program při použití tohoto algoritmu běžel příliš dlouho. Musel jsem tedy vybrat algoritmus jiný, který je daleko méně časově náročný.

MergeSort - prvním takovým algoritmem, který přišel v úvahu, byl algoritmus MergeSort. Jeho časová náročnost je řádově několikanásobně nižší. Nejhůře $O(n \cdot \log n)$. Problémem je však obrovská paměťová náročnost, která dosahuje až $O(n)$. Při použití tohoto algoritmu pak docházelo k padání programu - vyvolání chyby paměťové ochrany (angl. segmentation fault). Proto jsem musel hledat jinde.

QuickSort - dalším testovaným byl algoritmus QuickSort, který jsem nakonec vybral pro finální řešení. Ačkoliv je papírově jeho časová náročnost podobná jako u MergeSortu (v nějhorším případě může být ale až $O(n^2)$), má o něco lepší paměťovou náročnost $O(\log n)$. Protože oba algoritmy jsou volány rekurzivně, je spotřebovávána paměť zásobníku. Rekurze v případě QuickSortu je však nižší. Při použití tohoto algoritmu tak běh programu nebyl při testování nijak ovlivněn.

6.3.3 Možnosti spuštění

Pro aktivaci modulu slouží soubor "suricata.yaml"(jehož cesta je zpravidla `\etc\suricata\suricata.yaml`), ve kterém je pomocí přepínače možno tento modul zapnout - přidáním řetězce "- dos"v sekci "eve-log".

Součástí mé modifikace Suricaty je též přidání několika přepínačů pro načítání souboru, ukládání souboru a načtení hodnoty určující velikost časového rámce. Přidáním těchto přepínačů musel být modifikován jeden ze stěžejních modulů Suricata - soubory "suricata.c"a "suricata.h". Došlo zde však pouze k přidání možností přepínačů a taktéž proměnných, ve kterých je předán název souboru, nebo zmíněná velikost rámce.

Uvedu zde také ukázku použití těchto přepínačů:

- `-w` - volba šířky časového rámce v sekundách (v základním stavu je tato hodnota nastavena na 300 sekund).

```
suricata -w 120 -r eth2dump-pingFloodDDoS5m-6h_1.pcap
```

Výpis 6.3: Spuštění Suricaty s velikostí časového rámce 120 sekund.

- `-B` - modul uloží model síťového provozu do binárního souboru, jehož název je argumentem spuštění definován.

```
suricata -B model.dat -r eth2dump-pingFloodDDoS5m-6h_1.pcap
```

Výpis 6.4: Spuštění Suricaty s uložením modelu do souboru "model.dat".

- `-L` - modul načte model síťového provozu z binárního souboru, který slouží jako vzor pro výpočet limitů provozu. Jehož název je dán argumentem při spuštění.

```
suricata -L vzor.bin -r eth2dump-pingFloodDDoS5m-6h_1.pcap
```

Výpis 6.5: Spuštění Suricaty s načetním modelu ze souboru "vzor.bin".

Kapitola 7

Experimenty

V následující kapitole se budu věnovat testování a experimentům s přídavným modulem programu Suricata. Nejdříve se budu v sekci 7.1 zabývat popisem datasetů, které jsem pro verifikaci použil. Následně pak v sekci 7.2 samotným experimentům s programem.

7.1 Vstupní datasety s DDoS útoky

Pro experimentování s vytvořeným modulem jsem využil PCAP datasety dostupné na službě GitHub.[1] Soubory obsahují záplavové útoky ICMP, UDP a TCP protokolů s příznakem SYN. Složky také obsahují soubory zachycující čistý nezávadný síťový provoz. Každý soubor je označen typem útoku, přičemž název rovněž obsahuje údaj o délce trvání útoku a také o délce celého síťového záznamu. Všechny soubory jsou ve formátu PCAP verze 2.4.

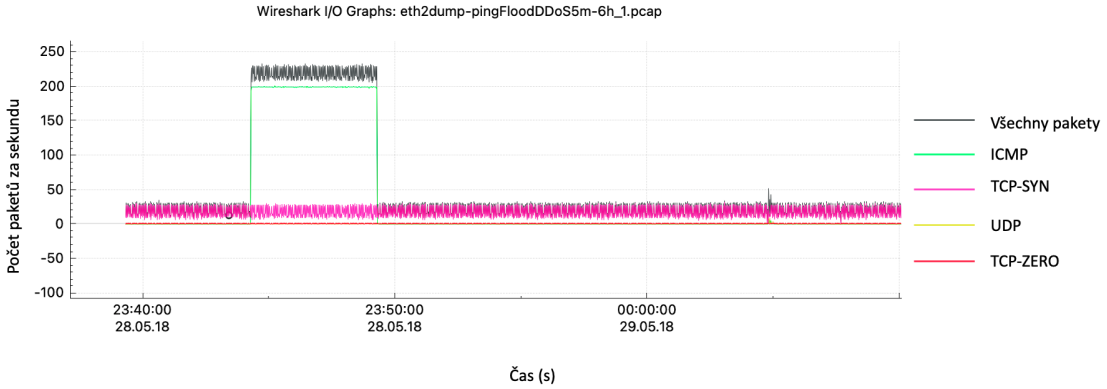
7.2 Testování

Pro verifikaci správné implementace této teorie jsem využil programu Wireshark[2]. Tento program kromě jiného umožňuje zobrazení grafů ("I/O Graphs"lze vybrat z horní nabídky "Statistics"), které obsahují infomrace o množství přenášených paketů, ať už jejich počet nebo velikost, v čase. Tyto pakety lze také filtrovat, v mém případě použitými filtry jsou:

- `icmp or icmpv6` pro pakety protokolu ICMP a ICMPv6 (zelená barva)
- `tcp.flags.syn` pro pakety protokolu TCP s příznaky SYN a SYN-ACK (růžová barva)
- `tcp.analysis.flags` pro pakety protokolu TCP, které vytváří toky s příznakem ZERO (červená barva)
- `udp` pro pakety protokolu UDP (žlutá barva)
- všechny pakety označeny barvou černou

Prvním takovým analyzovaným souborem bude "eth2dump-pingFlooDDoS5m-6h_1.pcap". Již z názvu víme, že by útok pomocí protokolu ICMP měl trvat 5 minut. Tuto informaci si také můžeme ověřit z následujícího grafu.

[1] https://github.com/tjcruz-dei/ICS_PCAPS/releases
[2] https://www.wireshark.org



Obrázek 7.1: Výřez grafu přenosu paketů v čase z programu Wireshark.

V grafu můžeme vidět velký nárůst přenosu paketů protokolu ICMP v čase 23:44 - 23:49. Následně tuto informaci srovnáme s informacemi z výstupního soubory "eve.json", který byl vygenerován Suricatou s mým rozšířením.

```
suricata -w 60 -r eth2dump-pingFloodDDoS5m-6h_1.pcap
```

Výpis 7.1: Spuštění Suricaty s PCAP souborem.

```
{
    "timestamp":"2018−05−28T23:43:27.601241+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:44:28.104970+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:45:29.669731+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:46:30.421013+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:47:31.764875+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:48:32.647772+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
```


Výpis 7.2: Výstup v souboru "eve.json"- 12σ

Program Suricata byl spuštěn s velikostí časového okna 60 sekund, kontrola útoků tak probíhá po 1 minutě. Ve výstupním souboru vidíme, že pravděpodobný útok probíhal v době 23:43 - 23:48, což odpovídá grafu z Wiresharku. Drobný rozdíl v časech je způsoben rozdílným způsobem měření. Wireshark měří přenos na úrovni paketů, Suricata naopak na úrovni toků. Odhalení útoku v tomhle případě tedy můžeme považovat za úspěšné.

7.3 Vliv počtu směrodatných odchylek a velikosti časového rámce na kvalitu detekce

Pro správné fungování programu má v našem případě zásadní vliv nastavení jednak hodnoty, kterou vynásobíme ve statistické analýze provozu směrodatnou odchylku σ - dále ji označujme jako **X** (její výchozí hodnota je nastavena na 12), ale také velikost časového okna , pro které jsou daná statistická ukládána a následně také kontrolována. Nyní však vyzkoušíme, jak budou výsledky ovlivněny, pokud hodnotu budeme měnit hodnotu X, v následujícím testu ji snížíme na polovinu - tedy 6.

```
{
    "timestamp":"2018−05−28T23:43:27.601241+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:44:28.099950+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:45:29.976408+0200",
    "attack_type":"DDoS_UDP",
    "dest_ip_addr":"172.27.224.255"}
{
    "timestamp":"2018−05−28T23:45:29.976408+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:46:30.134757+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:47:31.051542+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
{
    "timestamp":"2018−05−28T23:48:32.456863+0200",
    "attack_type":"DDoS_ICMP",
    "dest_ip_addr":"172.27.224.250"
}
```


Výpis 7.3: Výstup v souboru "eve.json"- 6σ

Můžeme vidět, že je zde navíc oznámení o možném UDP záplavovém útoku na adresu `172.27.224.225`. Pokud porovnáme toto oznámení s grafem programu Wireshark, můžeme jej pravděpodobně považovat za planý poplach. Protože jsme snížili hodnotu X, násobící směrodatnou odchylku, je náš horní limit propusti anomálií nižší, je upozorňováno tedy na větší množství přenosů. Hodnotu X bychom mohli také zvyšovat, v tomhle případě bychom to však dělali až do té doby, než by nebylo vygenerováno upozornění na žádnou anomálii.

Větší vliv na kvalitu detekce má však velikost časového rámce. Jestliže bude rámec malý, bude přesněji určen čas začátku a konce útoku. Detekce bude mnohem citlivější. Může tak docházet k falešným poplachům vyvolaných velmi malým místním výkyvem provozu. Větším časovým rámcem tato upozornění na výkyvy můžeme eliminovat, zároveň však můžeme tímto můžeme i zanedbat upozornění na opravdový útok, který trvá velmi krátkou dobu. Hodnota velikosti časového rámce je v základním nastavení 5 minut, lze ji však jednoduše měnit, zmínka je v sekci 6.3.3.

7.4 Hodnocení kvality detekce

Mnou vytvořený modul si při testování na daném souboru vzorků síťové komunikace obsahující DDoS útoky vedl poměrně slušně. Pro testování kvality detekce bylo vybráno 12 PCAP souboru se záznamy síťového provozu. Vybráno bylo náhodně 6 souborů s útoky TCP-SYN záplav, 6 souborů ICMP záplav a 6 souborů s Modbus záplavami dotazů. Soubory byly vybrány z výše zmíněného datasetu 1 ze složky "capturesv2". K těmto souborů byly také přidány dva s čistou komunikací. Nutno však dodat, že pro daný soubor je třeba vhodně nastavit délku časového rámce z důvodu popsáných výše v sekci 7.3. Srovnání výsledků mého modulu pak bylo porovnáno s informacemi dostupnými s grafu přenosů programu Wireshark. Výsledky testování detekce pak dopadly následovně:

Název souboru	**Čas útoku podle Wiresharku**	**Čas útoku dle detektoru**
eth2dump-pingFloodDDoS1m-6h_1.pcap	19:45:30 - 19:46:30	19:45:16 - 19:46:18
eth2dump-pingFloodDDoS5m-1h_1.pcap	16:34:00 - 16:39:00	16:33:54 - 16:38:33
eth2dump-pingFloodDDoS15m-1h_1.pcap	17:34:20 - 17:49:20	17:34:33 - 17:48:30
eth2dump-pingFloodDDoS1m-1h_1.pcap	15:34:00 - 15:35:00	15:33:48 - 15:34:50
eth2dump-pingFloodDDoS30m-6h_1.pcap	13:47:00 - 14:17:00	13:46:51 - 14:16:49
eth2dump-pingFloodDDoS15m-6h_1.pcap	07:46:30 - 08:01:30	07:46:17 - 08:01:16
eth2dump-tcpSYNFloodDDoS30m-6h_1.pcap	00:26:00 - 00:56:00	00:25:55 - 00:55:54
eth2dump-tcpSYNFloodDDoS5m-6h_1.pcap	12:25:10 - 12:30:10	12:24:58 - 12:30:06
eth2dump-tcpSYNFloodDDoS5m-1h_1.pcap	21:47:00 - 21:52:00	21:46:47 - 21:51:26
eth2dump-tcpSYNFloodDDoS30m-1h_1.pcap	23:48:00 - 00:18:00	23:28:04 - 00:17:27
eth2dump-tcpSYNFloodDDoS1m-0,5h_1.pcap	18:45:20 - 18:46:20	18:45:07 - 18:46:08
eth2dump-tcpSYNFloodDDoS15m-1h_1.pcap	22:47:20 - 23:02:20	22:47:16 - 23:01:58
eth2dump-modbusQueryFlooding5m-6h_1.pcap	01:56:10 - 02:01:10	01:55:57 - 02:01:07
eth2dump-modbusQueryFlooding15m-6h_1.pcap	07:56:50 - 08:11:50	07:56:38 - 08:11:37
eth2dump-modbusQueryFlooding1m-6h_1.pcap	19:55:20 - 19:56:20	19:55:11 - 19:56:12
eth2dump-modbusQueryFlooding1m-1h_1.pcap	02:22:00 - 02:23:00	02:21:48 - 02:22:49
eth2dump-modbusQueryFlooding15m-1h_1.pcap	04:22:50 - 04:37:50	04:23:12 - 04:37:08
eth2dump-modbusQueryFlooding5m-0,5h_1.pcap	01:20:05 - 01:25:05	01:20:27 - 01:24:35
eth2dump-clean-1h_1.pcap	-	čisto
eth2dump-clean-6h_1.pcap	-	čisto

Tabulka 7.1: Srovnání výsledků vlastního DDoS detektoru s daty z programu Wireshark.

Z tabulky 7.1 můžeme vidět, že výsledky testování programu dopadly přesvědčivě. Útoky byly poměrně přesně časově určeny, včetně toho, že nebyl odhalen žádný v případě čistého provozu. U něj je však hlavně potřeba zmínit to, že pokud je nastavena malá délka časového rámce, může dojít k falešnému upozorňování na útoku - malou délkou je myšlena délka do cca 5 minut. V případě malých časových rámců jsou totiž odhaleny místní odchylky průměrného provozu, které jsou však velmi krátké a jejich objem je velmi malý na to, aby se jednalo o útok.

Problém metody, kterou jsem při tvorbě využil, statistické analýzy je však v tom, že lze tzv. normální model provozu sítě útočníkem předem připravit. Statistická analýza funguje na principu odhalování anomálií na základě měření výskytů určitých proměnných (např. počtu toků). Jestliže však útočník bude provoz v síti postupně pomalu zvyšovat tak, aby již útok nevytvořil velký nárůst přenosu, pravděpodobně dojde k prolomení anomální detekce a škodlivý provoz nebude odhalen. Záleží však také na objemu dat, které daný útok tvoří. Jestliže útok bude trvat krátký časový interval a náš časový rámec bude větší, než je tento interval, i přesto může být útok odhalen díky jeho velkému objemu.

Co se týče samotné práce, určitě je zde také místo pro vylepšování do budoucna. Pro lepší a přesnější detekci je možno vytvořit větší množství modelů provozu (např. podle

denní doby) tak, aby bylo více myšleno na aktuální provoz, který se může v různé denní doby svým objemem lišit. Taktéž je možno při detekci zahrnout více proměnných, detekci také provozovat na vrstvě aplikační.

Kapitola 8

Závěr

Cílem mé práce bylo navrhnout rozšíření programu Suricata, fungující na principech detekce anomálií statistickou analýzou, který bude odhalovat DDoS útoky ze surového síťového provozu. Bylo při tom využito definice Gaussova normálního rozložení a jeho vlastnosti, kdy při určitém násobku směrodatné odchylky dojde k oddělení běžného provozu od anomálního. Počtem těchto směrodatných odchylek pak můžeme ovlivnit citlivost našeho detekčního modulu. Výstupy tohoto modulu byly porovnávány s analýzou síťového provozu pomocí programu Wireshark.

Během práce na tomto projektu došlo také ke studiu celé řady detekčních metod, včetně metod strojového učení. Věnoval jsem se taktéž studiu útoků odepření služby a jejich mitigaci. Došlo však také k praktickému testování a výběru nástroje, který bude při práci využit. Nakonec tímto nástrojem byla Suricata, která mě oslovila svou využitelností a architekturou.

Nejdříve došlo k vytvoření návrhu aplikace v jazyce Python, který sloužil jako hrubý náčrt toho, co bylo nakonec v jazyce C zpracováno a čím byla Suricata rozšířena. Díky implementaci v tomto jazyce jsem si také opět procvičil práci s pamětí, při které nedochází k paměťovým únikům. Vzhledem k tomu, že je taktéž Suricata vícevláknovou aplikací, musel jsem vzít v úvahu paralelní přístup k paměti a vyloučit jej. Což se mi nakonec podařilo.

Protože v poslední době dochází ke stále většímu nárůstu DDoS útoků a postupné změně zastoupení jejich typů, je zde určitě místo pro možná vylepšení do budoucna. Je zde možno statistické záznamy rozdělovat podle denní doby pro větší přesnost, výpočty a procházení záznamů zrychlovat a optimalizovat.

Literatura

[1] Bhardwaj, A. *What is a Perceptron? – Basics of Neural Networks* [online]. Duben 2020 [cit. 2022-05-01]. Dostupné z: https://towardsdatascience.com/what-is-a-perceptron-basics-of-neural-networks-c4cfea20c590.

[2] Bu, X. *Suricata Packet Processing Pipeline* [Digitální obrázek]. Prosinec 2017 [cit. 2022-04-25]. Dostupné z: https://xbu.me/images/suricata-thread-model/fig1.png.

[3] Buczak, A. L. a Guven, E. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys Tutorials*. 2016, sv. 18, č. 2, s. 1153–1176. DOI: 10.1109/COMST.2015.2494502.

[4] CISA. *DDoS Quick Guide* [online]. Říjen 2020 [cit. 2022-04-09]. Dostupné z: https://www.cisa.gov/uscert/sites/default/files/publications/DDoS%20Quick%20Guide.pdf.

[5] Fakulta elektrotechnická ČVUT v Praze. *Algoritmy řazení* [online]. [cit. 2022-05-01]. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/x33dsp/dsp-p3.pdf.

[6] Hazrat, A. *Anomaly detection through k-means clustering (first dataset)* [Digitální obrázek]. Červenec 2018 [cit. 2022-04-27]. Dostupné z: https://www.researchgate.net/profile/Hazrat-Ali-3/publication/326619835/figure/fig4/AS:654171346849792@1532978011669/Anomaly-detection-through-k-means-clustering-first-dataset.png.

[7] javaTpoint. *Fuzzy Logic Tutorial* [online]. 2021 [cit. 2022-04-30]. Dostupné z: https://www.javatpoint.com/fuzzy-logic.

[8] Kentik. *Netflow Overview* [online]. Březen 2022 [cit. 2022-03-31]. Dostupné z: https://www.kentik.com/kentipedia/netflow-overview/.

[9] Khraisat, A., Gondal, I., Vamplew, P. a Kamruzzaman, J. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*. Prosinec 2019, sv. 2. DOI: 10.1186/s42400-019-0038-7.

[10] Kumar, S., Kumar, A. a Srinivasan, S. Statistical Based Intrusion Detection Framework using Six Sigma Technique. *International Journal of Computer Science and Network Security*. Květen 2007.

[11] Lucidarme, P. *Simple Perceptron* [Digitální obrázek]. 2021 [cit. 2022-04-27]. Dostupné z: https://lucidar.me/en/neural-networks/files/perceptron.png.

[12] Nagori, A. I. *The Three-Way HandShake Model in TCP/IP* [Digitální obrázek]. Květen 2021 [cit. 2022-04-25]. Dostupné z: https://ruggedtooling.com/wp-content/uploads/2018/09/Botnet_Attack.png.

[13] NetManiac. *Wikimedia Commons: Schéma tradiční NetFlow architektury* [Digitální obrázek]. Březen 2008 [cit. 2022-03-25]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/9/98/NetFlowTradicniArchitektura.png.

[14] Nfdump. *Nfdump Documentation* [online]. Prosinec 2014 [cit. 2022-04-02]. Dostupné z: http://nfdump.sourceforge.net.

[15] Nguyen, L. Overview of Bayesian Network. *Science Journal of Mathematics & Statistics*. Červenec 2013, sv. 2013. ISSN 2276-6324.

[16] OISF. *Suricata User Guide* [online]. 2019 [cit. 2022-05-01]. Dostupné z: https://suricata.readthedocs.io/en/suricata-6.0.4/.

[17] Osadciw, L. A. *A simple example of Bayesian Network* [Digitální obrázek]. Duben 2004 [cit. 2022-04-29]. Dostupné z: https://www.researchgate.net/figure/A-Simple-Bayesian-Network-A-simple-example-of-Bayesian-Network-is-shown-in-Figure-2-The_fig4_228772036.

[18] Sharma, M. *Wikimedia Commons: Real Example Of Decision Tree* [Digitální obrázek]. Leden 2021 [cit. 2022-05-05]. Dostupné z: https://k21academy.com/wp-content/uploads/2020/12/Yes_No.png.

[19] The Snort Project. *SNORT Users Manual 2.9.16* [online]. Duben 2020 [cit. 2022-05-01]. Dostupné z: https://www.snort.org/documents/snort-users-manual.

[20] The Zeek Project. *About Zeek* [online]. 2021 [cit. 2022-05-01]. Dostupné z: https://docs.zeek.org/en/lts/about.html.

[21] Toews, M. W. *Wikimedia Commons: Standard deviation datagram* [Digitální obrázek]. Duben 2007 [cit. 2022-03-29]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/8/8c/Standard_deviation_diagram.svg.

[22] Tooling, R. *DDoS attack creation* [Digitální obrázek]. Duben 2018 [cit. 2022-04-25]. Dostupné z: https://ruggedtooling.com/wp-content/uploads/2018/09/Botnet_Attack.png.

[23] Tripathi, N. a Mehtre, B. *DoS and DDoS Attacks: Impact, Analysis and Countermeasures*. Prosinec 2013.

[24] Wikipedia. *Standard deviation* [online]. 2022. Dostupné z: http://en.wikipedia.org/w/index.php?title=Standard%20deviation&oldid=1083858387.

Příloha A

Obsah přiloženého CD

- **/bp/** - zdrojové soubory pro vytvoření PDF bakalářské práce
- **/suridos/** - všechny instalační soubory projektu
- **/suridos/src** - zdrojové soubory programu Suricata, včetně zdrojových souborů implementovaného DDoS detektoru
- **/suridos/README.md** - popis instalace a spuštění projektu