Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky

Bakalářská práce

Detekce anomálií založená na strojovém učení v reálném síťovém provozu

Martin Petráček

Vedoucí práce: Ing. Martin Kopp

11. května 2015

Poděkování

Děkuji svému vedoucímu, panu Ing. Martinu Koppovi, za trpělivost, cenné rady a připomínky při psaní této práce. Dále děkuji své rodině a blízkým za podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 11. května 2015 .....

Odkaz na tuto práci

Petráček, Martin. *Detekce anomálií založená na strojovém učení v reálném síťovém provozu*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce se zabývá využítím metod strojového učení v oblasti síťové bezpečnosti. Jejím cílem je implementace vybraných algoritmů z oblasti strojového učení pro detekci anomálií v síťových tocích. Součástí práce je rovněž popis řešení pro zachycování dat o tocích v síti z reálné počítačové sítě. Tato data o síťových tocích jsou použita pro srovnání úspěšnosti implementovaných algoritmů.

Klíčová slova síťová bezpečnost, machine learning, detekce anomálií, NetFlow

Abstract

This work deals with the using of machine learning for network security. Aim of this work is implementation of chosen machine learning algorithms for anomaly detection in network flows. This work also contains description of method for capturing data about network flows from real computer network. Captured data are then used for comparison of implemented algorithms.

Keywords network security, machine learning, anomaly detection, NetFlow

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Pro moderní společnost jsou počítačové sítě velmi důležité a řada každodenních činností je přímo závislá na jejich správném fungování. Vyhledávání informací bez Internetu si dnes téměř nedovedeme představit. Přes Internet ovládáme bankovní účty a obchodujeme. Prostřednictvím počítačových sítí dnes můžeme ovládat řadu domácích spotřebičů, jako plynové kotle, pračky nebo ledničky. Masivní rozšíření Internetu vytvořilo mnoho nových příležitostí a díky němu vznikly celé nové obory lidské činnosti.

S rozšiřováním a rostoucím významem Internetu se objevují stále sofistikovanější pokusy o jeho zneužití. Počet kybernetických útoků stále narůstá, podle studie firmy pwc[1] z roku 2014 55% velkých společností zažilo v uplynulém roce kybernetický útok a 24% z nich zaznamenalo průnik útočníků do jejich sítí. Pro firmy může mít narušení bezpečnosti fatální následky, jak ukazuje příklad certifikační autority DigiNotar, kterou zničil úspěšný útok na její infrastrukturu[2].

Se zvyšujícím se počtem kybernetických útoků se stále zdokonalují metody útočníků a je proti nim potřeba hledat nové metody obrany. Klasický přístup k ochraně sítě stojí na statických, člověkem připravených pravidlech popisujících útoky. Tento přístup ale neumí pružně reagovat na nové hrozby, pouze zabraňuje již známým a popsaným hrozbám. Problém spolehlivé detekce útoků metodami založenými na hledání známých vzorků se projevuje zejména u útoků, které vypadají jako běžný provoz. Typickými příklady jsou DoS (Denial of Service) útoky, které zaplaví cíl velkým množstvím zdánlivě legitimních požadavků a snaží se ho tím přetížit. Zvlášť jeho distribuované varianty jsou velmi obtížně detekovatelné klasickými metodami, protože pro rozpoznání co je ještě běžný provoz a co už je útok je třeba znát informace z minulosti. Provoz je také závislý na denní době, dnu v týdnu a dalších faktorech. Statickými metodami detekce útoků je tento problém obtížně řešitelný.

Řešením těchto problémů statické detekce by mohly být metody strojového učení, které se snaží pochopit, jak vypadá normální provoz na síti a detekovat náhlé změny jeho náhlé změny. Pokud v běžném chování sítě nastane výrazná

změna, může to ukazovat na probíhající útok. Ve svojí práci jsem tyto metody testoval a srovnával jejich vlastnosti.

Cíle práce

- Zachytit síťový provoz pomocí softwarové sondy fprobe, včetně uměle vytvořených útoků
- Implementovat algoritmy pro detekci anomálií založené na strojovém učení (kNN, LOF a Isolation Forest)
- Algoritmy otestovat na zachycených datech, porovnat výsledky z reálné sítě s teoretickými předpoklady z literatury

Struktura práce

První kapitola mojí práce je věnována způsobům získávání informací o dění na síti. Podrobněji se zabývá síťovými toky a popisu protokolu NetFlow. Ve druhé části popisuji použité nástroje pro monitorování toků v síti a způsob, který jsem zvolil pro zachytávání informací o nich. Kapitolu uzavírá popis několika druhů útoků a nástojů, kterými jsem je uměle vyráběl. Ve třetí části představuji implementované algoritmy pro detekci anomálií. Čtvrtá část se věnuje vlastní implementaci a následně jejímu testování na zachycených datech.

Kapitola 1

Síťové toky

Nejsnažším způsobem monitorování sítě je ukládání všech paketů, které sítí projdou. K tomuto účelu slouží např. nástroj tcpdump[3]. Pomocí něj získáme všechny informace o dění na síti, proto se používá například k diagnostice sítě. Problémem tohoto přístupu k monitorování sítí je právě to, že ukládá veškerá data, která prochází sítí. Nároky na kapacitu úložiště na páteřních sítích by brzy přesáhly jakoukoliv rozumnou mez. I na malém segmentu sítě lokálního ISP, kterou jsem používal pro sledování, denně prošlo sítí přes 100 GB dat. Přes páteřní linku do Internetu tohoto (poměrně malého) ISP běžně projde přes 6 TB dat za den. Je těžko představitelné toto množství dat ukládat a dále zpracovávat.

Pro analýzu provozu na síti se můžeme omezit na metadata z hlaviček. Hlavičky jsou menší než tělo paketu, ukládáním pouze hlaviček paketů ušetříme mnoho z kapacity úložiště. Dále, po síti bývá často přenášeno mnoho paketů, které mají shodnou část údajů z hlavičkek - zdroj a cíl. Proto je výhodné jednotlivé pakety agregovat do tzv. síťových toků. Síťový tok je posloupnost paketů, které sdílí zdroj a cíl. Existuje několik řešení pro zachytávání informací o síťových tocích, nejrozšířenějším z nich je dnes zřejmě protokol NetFlow, kterým se budu dále zabývat ve zbytku této kapitoly.

1.1 NetFlow

NetFlow[4][5] je otevřený protokol pro logování síťových toků vyvinutý společností Cisco. NetFlow definuje pravidla, na základě kterých se agregují pakety do toků, a protokol komunikace mezi sondami a kolektorem. Tím zároveň určuje, jaké informace máme k dispozici o každém toku.

1.1.1 Síťový tok podle NetFlow

Síťový tok je podle standardu NetFlow verze 5 definován jako posloupnost paketů, které sdílejí následující charakteristiky:

- Vstupní síťové rozhraní
- Zdrojová IP adresa
- Cílová IP adresa
- IP protokol
- Zdrojový port pro UDP nebo TCP, 0 pro ostatní protokoly
- Cílový port pro UDP nebo TCP, typ a kód pro ICMP, 0 pro ostatní protokoly
- IP typ služby (Type of Service)

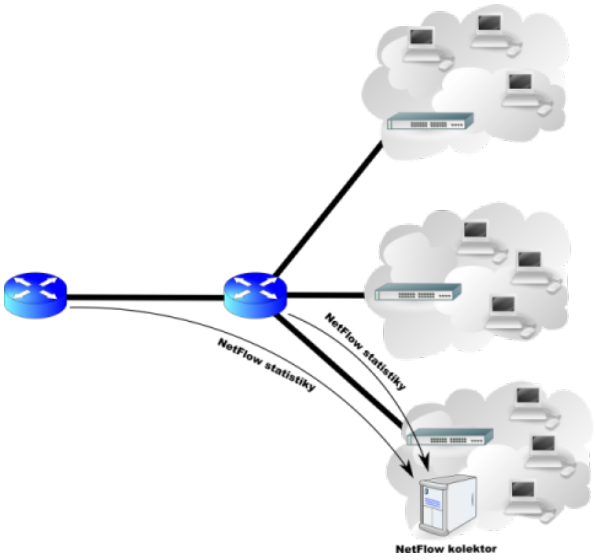
Síťový tok začíná přijetím paketu, který nespadá do žádného již aktivního toku (tzn. sonda dosud nezná výše uvedenou sedmici parametrů). Síťový tok končí pokud ve stanoveném intervalu nebyl přijat žádný paket, který by patřil k toku nebo je ukončen, pokud překročí stanovený maximální časový limit. Po ukončení toku je informace o něm ihned odeslána na kolektor. Maximální časový limit je zaveden proto, aby se informace o dlouho trvajících tocích dostala na kolektor v krátkém čase a ne až po skutečném skončení toku. Takto rozdělené dlouhé toky se na kolektoru opět spojí do jednoho. Z definice je patrné, že síťový tok je pouze jednosměrný. Síťové spojení typicky probíhá obousměrně, vzniknou tedy dva síťové toky.

1.1.2 Architektura sběru dat

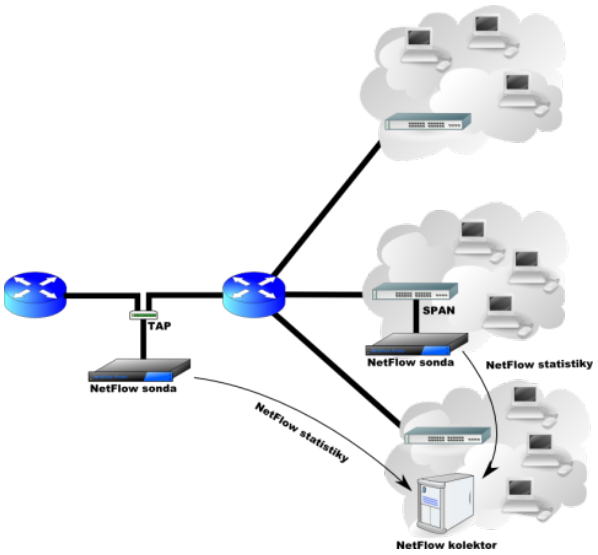
Standard NetFlow popisuje i způsob sběru těchto dat. Sběr dat na síti mají provádět sondy (nazývané exportéry), které potom výsledky posílají na kolektor. Exportérů je v síti typicky více, kolektor jeden.

V době uvedení NetFlow se předpokládalo, že v roli exportérů budou směrovače, které mají pro sběr informací o tocích i výhodné místo na síti. Tento způsob se označuje jako tradiční architektura NetFlow a je znázorněna na obrázku 1.1 Později se ukázalo, že při velkém počtu toků softwarové exportéry v routerech značně vyčerpávají výpočetní výkon určený pro směrování. Existují 2 přístupy k řešení tohoto problému.

Prvním z nich je sampled NetFlow, kde se neanalyzují všechny pakety, ale jen jeden z n paketů. Tento postup sice snižuje nároky na zpracování toků, ale pro použití v oblasti bezpečnosti se nehodí. Druhý přístup spočívá v tom, že roli exportéru převezme místo routeru specializovaná HW sonda. Sonda se nestará o nic jiného než o sbírání dat. V Česku vyrábí tyto sondy pod názvem FlowMon firma Invea[6]. Existuje celá škála HW sond pro rychlosti sítě od 10Mbps až do 100Gbps. Sondy se do sítě připojují zcela pasivně pomocí mirroringu portu na switchi nebo pasivním rozbočením kabelu (TAP), takže do síťového provozu nijak nezasahují[7]. Tento způsob je v současnosti často



Obrázek 1.1: Tradiční NetFlow architektura. zdroj wikipedia.org



Obrázek 1.2: Moderní NetFlow architektura. zdroj wikipedia.org

používaný, označuje se jako moderní NetFlow architektura a je znázorněn na obrázku 1.2

1.1.3 Struktura toků

Protokol NetFlow verze 5 určuje, že každý exportovaný síťový tok obsahuje informaci o:

- Zdrojové a cílové IP adrese
- IP adrese dalšího skoku (next-hop)
- Vstupním a výstupním rozhraní
- Počet paketů v toku
- Celkový počet bajtů přenesených během toku
- Čas začátku a konce toku
- TCP/UDP zdrojové a cílové porty (nebo jejich ekvivalent)
- TCP flagy, které se objevily během spojení (kumulativní OR)
- IP protokol
- IP typ služby (Type of Service)
- Číslo zdrojového a cílového autonomního systému

Existuje několik dalších verzí NetFlow protokolu, například verze 7 přináší informace ze switchů, verze 9 má formát informace o toku daný šablonou, což umožňuje zpracovávat různé druhy informací z různých sond. Z NetFlow verze 9 vzniknul IETF standard IPFIX, někdy označovaný jako NetFlow verze 10.

Kapitola 2

Zachytávání síťového provozu

2.1 Použité nástroje

V této sekci představím nástroje, které jsem použil pro zachytávání síťových toků.

2.1.1 fprobe

Fprobe[8] je open-source implementace softwarové sondy (NetFlow exportér). Fprobe je založena na knihovně libpcap, kterou využívají i populární nástroje tcpdump[3] a Wireshark[9]. Použití sondy fprobe je jednoduché, fprobe spustíme příkazem

```
fprobe [options] remote:port
```

kde remote a port je adresa a port NetFlow kolektoru a options jsou nepovinné parametry Ty důležité shrnuje tabulka 2.1. Fprobe má řadu dalších parametrů, které slouží k ladění výkonu (velikost vyrovávací paměti, realtime priorita a další, jejich přesný popis nalezneme v manuálu.

Pokud tedy například chceme montitorovat rozhraní eth0 a toky odesílat protokolem NetFlow verze 5 na kolektor na adrese 192.168.1.1:9995, provedeme to příkazem:

Tabulka 2.1: Parametry fprobe

Parametr	Popis
-i <intf>	Síťové rozhraní, které chceme monitorovat
-f <expr>	Filtr (používá libpcap syntaxi filtrů[10])
-d <seconds>	Timeout neaktivních toků
-e <seconds>	Timeout aktivních toků
-n <version>	Verze NetFlow protokolu (1, 5 nebo 7)

```
fprobe -i eth0 -n5 192.168.1.1:9995
```

Pokud chceme navíc vyfiltrovat jen toky na TCP port 80, uděláme to příkazem:

```
fprobe -i eth0 -n5 -f'tcp port 80' 192.168.1.1:9995
```

2.1.2 NFdump

NFdump[11] je open-source balík nástrojů pro práci s NetFlow daty. Balík obsahuje kolektor NetFlow dat a programy pro práci s přijatými síťovými toky.

2.1.2.1 nfcapd

Nástroj nfcapd, přijímá a ukládá data z exportérů. Přijaté toky jsou ukládány do souborů v určené složce. Ve výchozím nastavení se toky přijaté během 5 minut ukládají do jednoho souboru. Po 5 minutách se vytvoří nový soubor a nové toky se ukládají do něj. Výchozí interval 5 minut je možné změnit parametrem programu.

Činnost programu nfcapd je opět možné ovlivnit řadou parametrů, z nich jsou pro nás důležité zejména parametry -l, -p a -t. Povinný parametr -l určuje složku, do které budou ukládány přijaté síťové toky. Parametr -p nastavuje port, na kterém bude nfcapd poslouchat, ve výchozím nastavení je to port 9995. Parametrem -t můžeme změnit interval rotace souborů z výchozích pěti minut na jinou hodnotu.

Program spustíme např. příkazem

```
nfcapd -l flows -p 9995
```

Tento příkaz spustí nfcapd na standardním portu 9995. Data jsou ukládána do složky flows.

2.1.2.2 nfdump

Dalším důležitým nástrojem balíku je program nfdump, který vypisuje sítové toky uložené programem nfcapd.

Možnosti nfdumpu jsou široké, uložené toky dokáže filtrovat, agregovat a počítat z nich různé statistiky.

Datové soubory pro nfdump vybereme pomocí parametru -R. Tento parametr má 3 možnosti zápisu. Buď můžeme za přepínač -R napsat přímo cestu ke složce, v tom případě nfpdump vypíše toky ze všech souborů v dané složce. Nebo je možné specifikovat jen názvu souboru, nfdump v tomto případě vypíše data ze všech souborů počínaje uvedeným souborem. Poslední možností

je předat programu nfdump rozsah souborů které má vypsat oddělených dvojtečkou. Nfdump v tomto případě vypíše všechny soubory lexikograficky ležící v tomto rozsahu.

Toky je možné na výstupu seřadit, pro určení parametru podle kterého se má řadit slouží parametr -O. Seřazení podle času začátku toku docílíme volbou -O tstart.

Užitečnou volbou je také agregace do obousměrných toků. Tu zapneme parametrem -b. V tom případě se 2 jednosměrné toky spojí do jednoho. Agregovaný tok zachová společné atributy obou toků (protokol, flagy) a přidá rozdílné atributy (počty bajtů, paketů). Agregace dat tímto způsobem je pro bezpečnostní použití výhodná, protože je přínosné vědět, kdo tok inicioval, což bychom z jednosměrných toků jednoduše nepoznali.

V určitých případech dochází k tomu, že sondy exportují toky v nesprávném pořadí (tok od cíle předchází tok od zdroje). V tomto případě může pomoci parametr -B. Při použití tohoto parametru se Nfdump pokusí odhadnout směr toků na základě zdrojových a cílových portů. Tato volba bude ještě vysvětlena a diskutována později, protože při zachytávání dat pomocí použité softwarové sondy jsem se s tímto problémem setkal.

Parametr -o slouží k formátování výstupu programu nfdump. Jednou možností je formát csv, který vypíše každý tok na jedné řádce, jednotlivé atributy oddělené čárkou. Určitou nevýhodou je, že formát csv má fixní formát a vypíše řadu nevyužitých a irelevantních atributů. Nejširší možnosti má formát -o "fmt:. . . ", za kterým následuje seznam atributů, které chceme vypsat. Seznam formátovacích řetězců lze nalézt v manuálu.

Uvedený příklad spustí nfdump na všech datech ze složky flows/, seřadí je podle názvu, zagreguje do obousměrných toků a vypíše ve formátu csv.

```
nfdump -R flows/ -O tstart -b -o csv
```

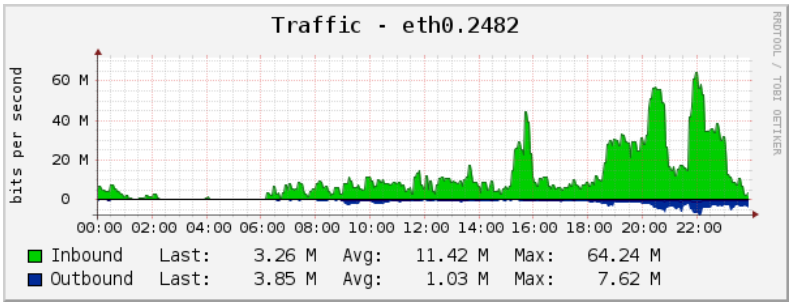
2.1.2.3 Ostatní nástroje

Balík nfdump zahrnuje ještě další nástroje, o kterých se jen stručně zmíním. Program nfanon slouží k anonymizování IP adres v tocích. To může být žádoucí, pokud chceme odstranit z dat citlivé informace. Program nfexpire je určen k čištění starých záznamů o tocích. Programem nfreplay můžeme předat již uložená dat o tocích na jiný kolektor. Data jsou exportována standardním NetFlow protokolem.

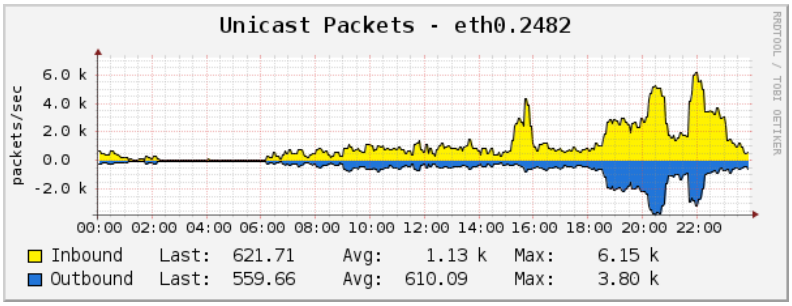
2.2 Zachytávání dat

Data jsem zachytával pomocí výše představených nástrojů fprobe a nfcapd.

Softwarová sonda fprobe byla spuštěna na PC s linuxem sloužícím jako router místního ISP. Tento router směruje pakety zhruba stovky počítačů.



Obrázek 2.1: Vývoj objemu provozu na sledovaném routeru v průběhu dne



Obrázek 2.2: Vývoj počtu paketů na sledovaném routeru v průběhu dne

Průměr provozu za celý den bývá přibližně 1.5 MB/s a ve špičkách dosahuje 9 MB/s. Při maximální velikosti paketu 1500B by to znamenalo průměrně 1000 paketů za sekundu a téměř 6000 paketů ve špičkách, pokud bychom předpokládali, že všechny pakety jsou maximálně velké. Protože tento předpoklad v praxi neplatí, jde jen o odhad spodní hranice, skutečný počet paketů za sekundu bývá vyšší. Grafy 2.1 a 2.2 ukazují vývoj objemu provozu a počtu paketů za den na sledovaném routeru, získané z monitorovacího systému Cacti. Jak je vidět, ve špičkách byl počet paketů za sekundu téměř deset tisíc.

Data ze softwarové sondy byly zasílána na kolektor nfcapd, který jsem měl spuštěný na svém počítači. Počet zachycených síťových toků se během dne pohyboval mezi 2000 a 25000 toků za 5 minut. Data jsem programem nfdump zagregoval do obousměrných toků a uložil je ve formátu CSV. Při prohlížení dat jsem zjistil problém, že směry toků často neodpovídají. Počítače v sledované síti používají vnitřní adresy a při přístupu na Internet jsou adresy přeloženy na veřejné IP adresy mechanismem NAT (Network Adress Translation). Jedním z důsledků tohoto mechanismu a jeho konkrétního nasazení ve sledované síti je, že stroje v Internetu (mimo vnitřní síť) nemohou iniciovat spojení se strojem ve vnitřní síti. Přesto mnoho exportovaných toků mělo za zdroj adresu ve vnějším Internetu a cíl adresu ve vnitřní síti, což reálně není možné.

Zjistil jsem, že problém vzniká už na softwarové sondě fprobe, která v některých případech špatně určí přesný čas začátku toku (přesný čas přijetí prvního paketu). Pokud odpověď od cíle přijde rychle, může se stát že paket od cíle vypadá, jako by přišel dřív než paket od skutečného zdroje toku.

Tento problém se mi se sondou fprobe nepodařilo vyřešit. Vyzkoušel jsem i jiné softwarové sondy - fprobe-ulog, což je fork sondy fprobe, která používá jiný způsob zachytávání paketů (mechanismus netfliteru ULOG) i zcela jinou sondu softflowd (která opět používá knihovnu libpcap). Popsaný problém se objevil na všech těchto sondách. To ukazuje na výše zmíněné problémy tradiční NetFlow architektury. Dnes se v praxi používají spíše hardwarové NetFlow sondy, které jsem ale neměl k dispozici, proto nevím, zda se popsaný problém vyskytuje i u nich.

Existence tohoto problému si jsou vědomi i autoři balíku nfdump, protože program nfdump sloužící k exportu zachycených toků kromě přepínače -b sloužícímu k agregaci obousměrných toků obsahuje i přepínač -B, který zagreguje data do obousměrných toků a navíc se pokusí odhadnout směr toku podle zdrojových a cílových portů. Odhadování směru stojí na předpokladu, že klient volí vysoké číslo portu (>1024) a připojuje se ke službách s nízkými čísly portu(<1024). Tento výběr portů doporučuje RFC 6056 a běžně se používá pro většinu běžných protokolů jako je HTTP(S), FTP, SSH, DNS a další. Nfdump spuštěný s přepínačem -B prohodí zdroj a cíl toku, pokud zaznamená tok se zdrojovým portem <1024 a cílovým portem >1024.

Toto způsob odhadu směru ale zcela selhává u méně obvyklých aktivit na síti, jako jsou například skeny portů. Skeny portů běžně testují i porty vyšší než 1024 a nfdump v tom případě nedokáže správně určit směr toku. Praktické testy ukázaly že k prohození zdroje a cíle u síťových skenů dochází relativně často. Protože síťové skeny byly jedním z typů umělých útoků, které jsem zkoušel detekovat a směr toku byl zásadní pro některé další výpočty, znamenaly tyto chyby v datech poměrně nepříjemný problém.

Problém jsem vyřešil pomocí znalosti struktury sítě a jejích adresních rozsahů. Protože je v této síti použit NAT, pouze počítače ve vnitřní síti mohou iniciovat spojení do Interetu. To znamená, že adresa z vnitřní sítě je vždy zdrojem toku k adresám mimo vnitřní síť a pokud jsou adresy v záznamu o síťovém toku opačně, je téměř jisté, že jsou zdroj a cíl toku je v opačném pořadí. Vytvořil jsem si tedy jednoduchý program, který tímto způsobem vyměňoval adresu zdroje a cíle u síťových toků. Toto řešení není zcela univerzální a musí se přizpůsobovat specifikům konkrétní sítě, ale vyřešilo problém, který jsem se špatnými směry toků měl.

2.3 Generování umělých útoků

Při zachytávání síťového provozu jsem uměle vyrobil několik druhů známých síťových útoků, které jsem pak využíval pro hodnocení výsledků detekce ano-

málií. V dalším textu stručně popíši jednotlivé typy vytvořených útoků a techniky, kterými jsem je vygeneroval.

2.3.1 Síťové skeny

Síťový sken není přímo metodou útoku, ale útoku téměř vždy předchází. Slouží k zmapování sítě a případně služeb, které zařízení v síti poskytují. Síťové skeny se obvykle rozdělují na horizontální a vertikální. Jako horizontální sken bývá označujováno skenování více strojů a jen několika málo portů. Tento typ skenu se používá ke zjištění zařízení na síti. Jako vertikální sken se označuje detailní sken portů jednoho konkrétního zařízení. Skenování vytvoří velké množství síťových toků z jednoho zdroje k mnoha cílům.

K vytvoření síťových skenů jsem použil open-source nástroj nmap[12]. Tento nástroj umí skenovat porty, dostupnost strojů a mnoho dalších věcí. Silnou stránkou nmapu je rozšíření o uživatelské skripty, které významně rozšiřují jeho možnosti. Skripty umožňují detekovat například konkrétní verze softwaru nebo jejich známé zranitelnosti[13].

2.3.2 DoS útoky

DoS (Denial of Service) útoky se snaží o přetížení cílového stroje tak, aby neměl kapacity pro obsluhování ostatních klientů. Existuje mnoho variant tohoto útoku, zaměřených na konkrétní službu nebo na celý stroj. Protože převažujícím typem provozu ve sledované síti byla spojení na webové servery, zvolil jsem varianty útoku DoS s podobnou charakteristikou provozu jako má běžný provoz na síti. Prvním vybraným útokem je klasický HTTP DoS útok usilující o zahlcení webového serveru množstvím dat. Druhou testovanou variantou je tzv. pomalý HTTP DoS útok, který se snaží udržet co nejvíce navázaných spojení s cílem a znemožnit mu tak navazovat spojení s ostatními klienty.

Klasický HTTP DoS útok jsem vytvářel programem LOIC (Low Orbit Ion Cannon)[14]. Tento program po spuštění útoku začne posílat na cíl co největší množství dat. Množstvím dat se snaží zahltit cíl. Útok je velice primitivní, ale obtížně se odfiltrovává od legitimního provozu a pokud mnoho zdrojů útočí na jediný cíl, může být velmi účinný. Účinnost prokázaly i medializované útoky hnutí Anonymous z roku 2012, při které tento program využívaly[15]. Aby bylo vůbec reálné útok na této síti detekovat, musel jsem v nastavení programu výrazně snížit rychlost útoku (snížením počtu threadů). V případě ponechání výchozího nastavení toky vygenerované programem rychle vytvořily většinu veškerého provozu na síti. Pokud toky útoků tvoří většinu síťového provozu, nelze je považovat za anomálie.

K vytvoření pomalého DoS útoku jsem využil skript Slowloris[16]. Tento software vytvoří a udržuje mnoho otevřených spojení na cílový webový server. V každém spojení pošle nekompletní HTTP požadavek. Po čase zasílá další části hlaviček, ale požadavek nikdy nedokončí, pouze tím udržuje spo-

jení aktivní. Server nemůže požadavek vyřídit, dokud není kompletní a musí čekat na jeho dokončení. V závislosti na způsobu zpracování požadavků může tento útok rychle vyčerpat prostředky serveru. Tímto útokem je zranitelný například webový server Apache. Pomalý DoS útok vygeneruje menší počet síťových toků, které dlouho trvají a je jich otevřeno mnoho zároveň.

Kapitola 3

Představení implementovaných algoritmů

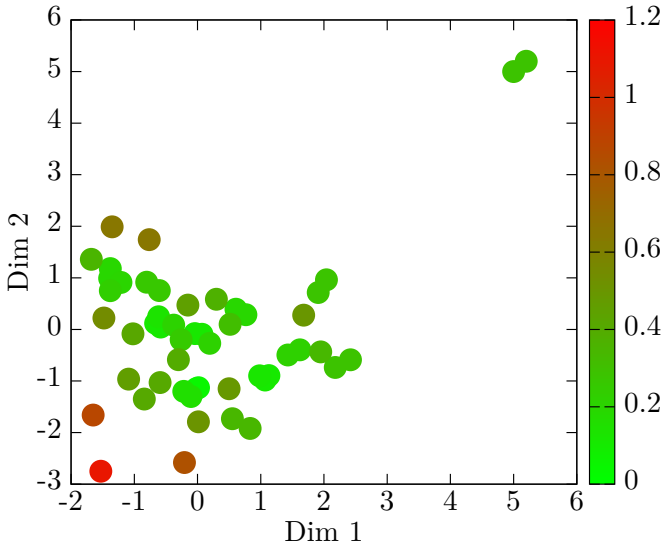
Tato kapitola popisuje implementované algoritmy strojového učení použité pro detekci anomálií. Všechny metody očekávají jako vstup vektory z $\mathbb{R}^n$, které reprezentují souřadnice bodu v prostoru atributů. Atributy v tomto kontextu rozumíme vlastnosti toku jako jsou délka spojení, počet přenesených bajtů a další. Výstupem detekčních algoritmů je reálné číslo vyjadřující míru odlehlosti každého vzorku v rámci předložených dat.

3.1 kNN

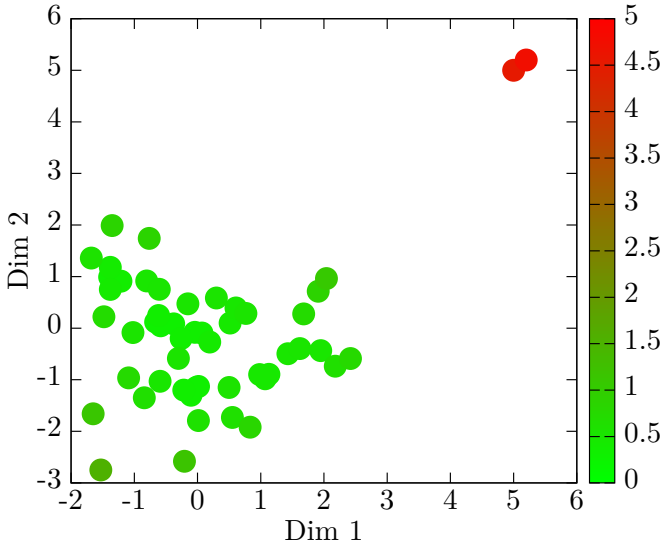
Algoritmus k-nejbližších sousedů (kNN) je asi nejjednodušší algoritmus používaný ve strojovém učení. Jeho využití pro detekci anomálií je založeno na předpokladu, že anomálií je v datech málo a od normálních dat se velice liší, proto i vzdálenost k jejich nejbližším bodům bude v průměru výrazně větší. Použitá varianta algoritmu kNN nalezne ke každému bodu k nejbližších bodů a spočítá vzájemné vzdálenosti. Výstupem algoritmu jsou průměry vzdáleností ke k nejbližším bodům, které určují stupeň odlehlosti.

Kromě průměrování vzdáleností k nejbližším sousedům jsem vyzkoušel ještě jiné možnosti, jako průměr kvadrátů nebo maximum, ale ukázalo se, že to nemá podstatný vliv na výsledky. Pouze se změnily absolutní vzdálenosti, ale po seřazení zůstalo pořadí bodů téměř stejné.

Počet hledaných nejbližších sousedů (k) volí uživatel a je konstantní pro všechny body. Tento parametr určuje maximální velikost shluku, jehož body mohou být považovány za anomální. Obrázek 3.1 demonstruje výsledky algoritmu 1-NN pro 50 náhodných bodů z normálního rozdělení a 2 uměle přidané body v pravém horním rohu. Barevná škála znázorňuje vzdálenost k nejbližšímu sousedovi. Je vidět, že přidané body by při volbě $k = 1$ byly považovány za zcela normální. Obrázek 3.2 ukazuje výsledky algoritmu 5-NN na stejných



Obrázek 3.1: Ukázka výsledků 1-NN



Obrázek 3.2: Ukázka výsledků 5-NN

datech. Zde jsou přidané body již zřetelně rozpoznány jako anomální.

Základní varianta k-nejbližších sousedů má kvadratatickou výpočetní složitost vzhledem k počtu bodů. Pro každý bod algoritmus počítá vzdálenosti ke všem ostatním. S počtem bodů rychle roste čas potřebný k výpočtu. Navíc výpočet každé vzdálenosti má lineární složitost vzhledem k počtu dimenzí. Protože záznamy o síťových tocích mají mnoho atributů (mnoho dimenzí vek-

toru) a bývá jich velké množství, výpočet by trval velmi dlouho. Proto je třeba algoritmus optimalizovat.

3.1.1 Optimalizace

Základní variantu algoritmu kNN jsem optimalizoval metodou pomocí shlukování blízkých bodů[17]. Tato metoda umožňuje výrazně zredukovat počet bodů, ke kterým je nutné počítat vzdálenost, což významně šetří výpočetní čas. Samotná optimalizace má 2 fáze. V první z nich se vytváří shluky (clustery) o fixním poloměru. V druhé fázi pro každý bod najdeme nejbližší clustery a pouze v nich hledáme nejbližší sousedy.

V první fázi se z bodů vytváří shluky o fixním poloměru a body se do nich přiřazují. Přiřazení probíhá velice jednoduše, postupně se prochází body a spočítají se vzdálenosti k již existujícím shlukům. Pokud neexistuje shluk, jehož střed by byl blíže než určený poloměr, vytvoří se nový shluk a bod se prohlásí za jeho střed. Pokud takový shluk existuje, bod se přidá do tohoto shluku. Může se stát, že existuje víc takových shluků (shluky se mohou překrývat), bod se přidá vždy pouze do jednoho z nich. Není podstatné do kterého, ale při přidávání bodů do shluků se snažíme, aby byly rovnoměrně rozděleny.

V druhé fázi se již hledá k nejbližších sousedů postupně pro všechny body. Algoritmus si udržuje množinu možných nejbližších sousedů bodu, na začátku prázdnou. Nejdříve se spočítají vzdálenosti ke všem shlukům a seznam shluků se seřadí podle těchto vzdáleností.

Poté se postupně prochází seznam shluků od nejbližších. V každém kroku se přidají body z procházeného shluku do množiny možných nejbližších sousedů, tato operace se nazývá otevření shluku. Po otevření shluku se určí minimální vzdálenost bodů, které v množině ještě nejsou. Tuto vzdálenost označme $dist_{min}$, dále označme poloměr $diameter$, $dist_i$ vzdálenost ke středu aktuálního shluku a $dist_{i+1}$ vzdálenost ke středu dalšího nejbližšímu shluku. Pokud platí $dist_{i+1} > dist_i + diameter$, potom se následující shluk nepřekrývá s aktuálním shlukem a nastavíme $dist_{min} = dist_i + diameter$, v opačném případě se překrývat může a nastavíme $dist_{min} = dist_{i+1} - diameter$.

U všech bodů z množiny, které jsou blíže než $dist_{min}$ máme zaručeno, že žádný bod mimo množinu nemůže ležet blíže. S otvíráním dalších shluků tato vzdálenost roste. Stačí, pokud se alespoň k bodů z množiny nachází blíže než $dist_{min}$. V tu chvíli můžeme procházení shluků ukončit. Nejbližšími sousedy bodu jsou body z množiny bližší než $dist_{min}$. Pokud je bodů v množině víc než k, stačí použít k nejbližších.

Tato optimalizace umožňuje výrazně snížit počet vzdáleností mezi body, které je třeba počítat. Nepočítají se již vzdálenosti ke všem bodům, ale pouze vzdálenosti ke shlukům (kterých má být výrazně méně než všech bodů) a poté pouze k bodům v několika nejbližších shlucích (v závislosti na rozložení bodů). Pro algoritmus je třeba zvolit poloměr shluků, který ale ovlivňuje pouze rychlost algoritmu a nemá vliv na přesnost.

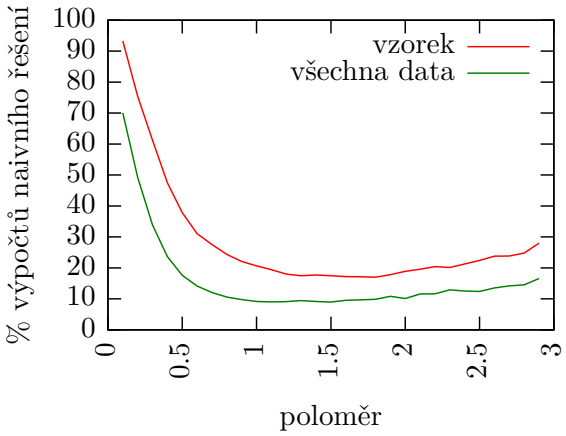
Volba ideálního poloměru se ukázala jako nesnadná. Intuitivně, při nulovém poloměru algoritmus degeneruje na naivní variantu algoritmu kNN, každý shluk obsahuje jediný bod a shluků bude stejný počet jako bodů. Dokonce počet spočítaných vzdáleností proti naivní variantě ještě naroste, protože se budou počítat jak při vytváření shluků, tak při jejich hledání v druhé fázi. Ideální poloměr určuje rovnováha mezi několika protichůdnými požadavky. Protože pro každý bod se musí vždy počítat vzdálenosti ke všem shlukům, je žádoucí co nejmenší počet shluků. Jejich počet se zmenšuje se zvětšováním poloměru (stále více bodů je blíže než poloměr). Se snižujícím se počtem shluků ale roste počet bodů v nich zahrnutých a to zvyšuje počet počítaných vzdáleností k těmto bodům. V extrémním případě, pokud poloměr bude příliš velký, optimalizovaný algoritmus opět degeneruje na naivní variantu algoritmu kNN, vytvoří se pouze jediný shluk, který zahrne všechny body a bude opět třeba počítat vzdálenosti ke všem z nich. Optimální poloměr závisí na počtu shluku a rozložení bodů v nich. To se mi nepodařilo analyticky nijak dopředu odhadnout.

Optimální poloměr zaručuje minimální počet vzdáleností mezi body, které bude potřeba spočítat. Zjistil jsem, že dobrý odhad poloměru je možné získat na malém vzorku dat. Optimální poloměr vzorku je často blízký optimálnímu poloměru všech dat. Na malém vzorku dat můžeme rychle vyzkoušet několik různých poloměrů. Nejlepší z nich potom použijeme jako poloměr pro algoritmus spuštěný na všech datech.

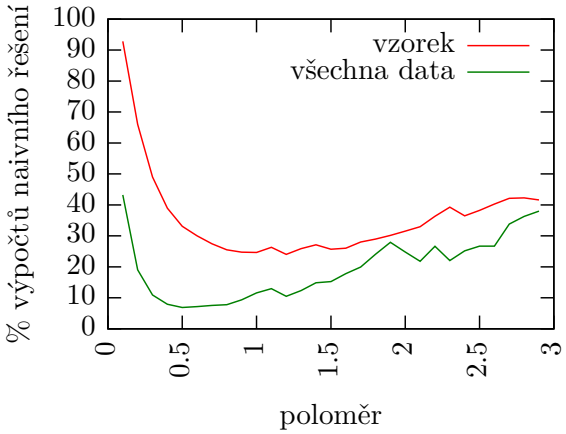
Grafy 3.3 a 3.4 zobrazují vliv volby poloměru na počty počítaných vzdáleností u vzorku 5000 bodů a u všech dat. Poloměr vybraný tímto způsobem byl často optimální (nebo velmi blízko optimálnímu) pro malá data. Při větším počtu bodů se stávalo, že odhad optimální nebyl. Obecně se ukázalo, že při zvyšujícím se počtu bodů se optimální poloměr posouval do nižších hodnot. Důvodem je, že na velkých datech se výrazněji projevoval vliv rostoucího počtu bodu ve shlucích než klesajícího počtu shluků. Tento vliv se zvětšováním poloměru narůstal a rychleji narůstaly i počty spočítaných vzdáleností. Na testovacím vzorku se tento jev tak výrazně neprojevoval a proto odhad poloměru byl při velkém počtu bodů často větší než optimální, jak je možné pozorovat i na grafu 3.4. Přesto, odhad získaný testováním na vzorku byl dostatečně dobrý a ve všech testovaných případech dokázal výrazně zrychlit běh algoritmu kNN oproti naivnímu řešení. Graf 3.5 porovnává počty spočítaných vzdáleností v naivní a optimalizované verzi algoritmu (s poloměrem automaticky voleným na vzorku 5000 bodů).

3.2 LOF

LOF[18][19] je zkratkou Local Outlier Factor, což se dá do češtiny přeložit jako míra místní odlehlosti. Algoritmus LOF odhaduje hustotu v okolí bodu a porovnává ji s hustotou v okolí nejbližších sousedů. Předpokládá, že u normál-



Obrázek 3.3: Odhad poloměru shluku - 30000 bodů

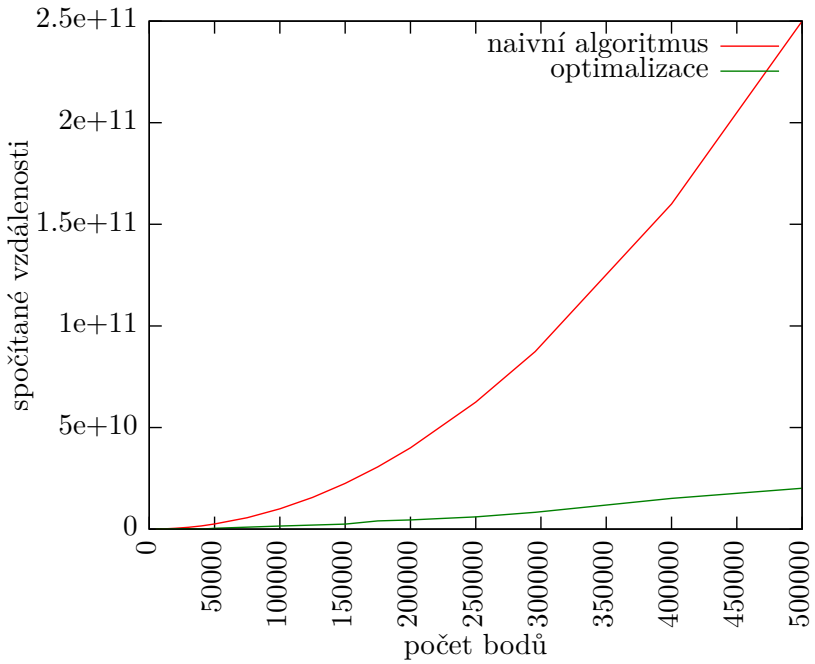


Obrázek 3.4: Odhad poloměru shluku - 500000 bodů

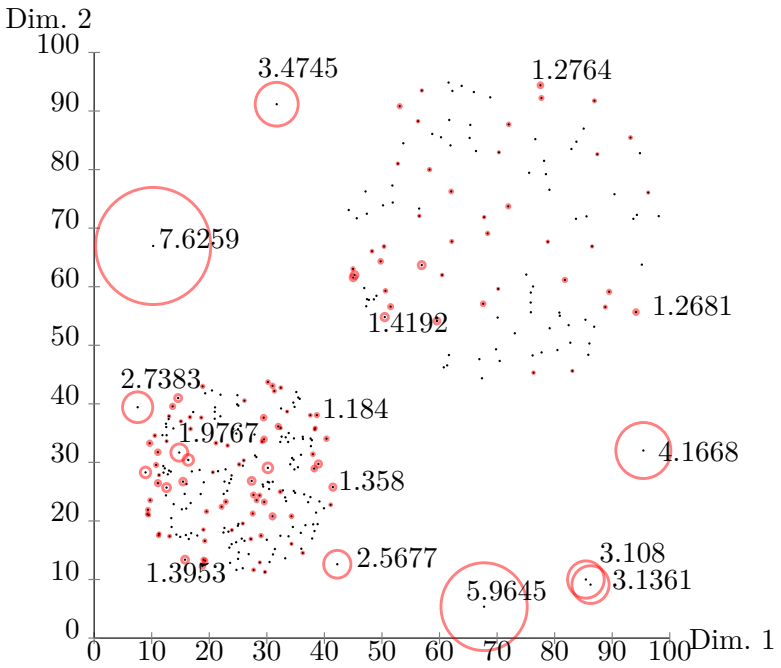
ního bodu jsou hustoty velmi podobné a naopak u anomálního bodu výrazně odlišné.

Výhodou algoritmu LOF oproti kNN je, že bere v úvahu hustoty shluků v datech. Názorně to lze demonstrovat na dvourozměrných datech, na obrázku 3.6. Anomální body vlevo dole algoritmus správně rozpoznal, přestože byly ke svým sousedům blíže než jsou body v pravém horním shluku. Pokud bychom na stejná data použili kNN (nebo jakoukoliv jinou metodu pro detekci anomálií založenou pouze na porovnávání vzdáleností), anomalitu těchto bodů v levém dolním rohu by vůbec nerozpoznala.

Algoritmus LOF odhaduje hustotu v okolí bodu pomocí vzdáleností k nejbližším sousedům bodu. Pro zvětšení stability výsledků používá místo normální vzdálenosti speciální vzdálenost (označovanou jako reachability distance).



Obrázek 3.5: Porovnání naivní a optimalizované verze algoritmu kNN



Obrázek 3.6: Lof skóre. zdroj wikipedia.org

Ta je definována takto:

$$reach\_dist(A, B) := max(k\_distance(B), distance(A, B)), \tag{3.1}$$

kde $k\_distance(B)$ je vzdálenost ke k-tému nejbližšímu sousedovi bodu B. To znamená, že reachability distance je skutečná vzdálenost mezi body A a B, ale minimálně vzdálenost k-tého nejbližšího souseda bodu B.

Hustotu v okolí (označovanou jako LRD - local reachability density) algoritmus odhaduje zprůměrováním reachability distance ke všem k nejbližším sousedům a inverzí této hodnoty. Pokud jsou v datech duplicitní body (body jejichž vzdálenost je 0), může být průměrná vzdálenost k bodům nulová. Ve floating point aritmetice je výsledkem dělení nulou $+\infty$, proto hustota i výsledné LOF může vyjít $+\infty$.

$$lrd(A) := 1/\frac{\sum_{B \in NN(A)} reach\_dist(A, B)}{|NN(A)|} \tag{3.2}$$

Tato hodnota vyjde vysoká pro body ležící uvnitř shluku (s velkou hustotou bodů v jeho okolí) a naopak nízká pro osamocené body mimo shluky.

Konečně, výsledné LOF bodu je určeno podle vzorce

$$lof(A) := \frac{\sum_{B \in NN(A)} lrd(B)}{|NN(A)|} / lrd(A). \tag{3.3}$$

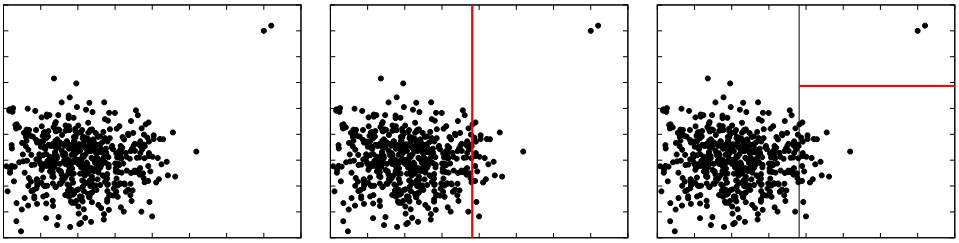
Vypočítá se jako průměrná hustota v okolí nejbližších sousedů bodu dělená hustotou v okolí bodu. Pokud se bod nachází uvnitř shluku, budou přibližně stejné hustoty v okolí jeho sousedů a v okolí bodu samotného. V tom případě vyjde hodnota LOF blízká jedné. Pokud je ale hustota v okolí bodu výrazně nižší než u jeho sousedů, hodnota LOF vychází výrazně vyšší než jedna.

Na rozdíl od algoritmu kNN lze výsledky algoritmu LOF přímo interpretovat. Hodnoty blízké 1 jsou zcela normální data. Vyšší výsledné LOF, znamená vyšší pravděpodobnost, že bod je anomálie.

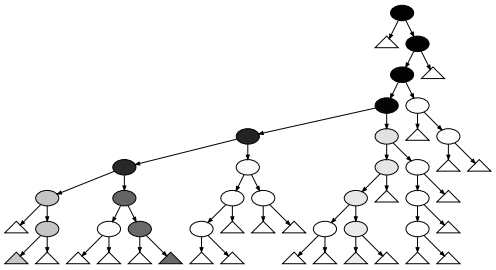
Výsledky algoritmu LOF silně závisí na volbě počtu nejbližších sousedů. Původní článek[18] popisuje i způsob výběru vhodného parametru. Výsledky algoritmu LOF mají výrazné výkyvy pro velmi malé počty sousedů, proto je doporučeno minimálně 10 nejbližších sousedů.

3.3 Isolation Forest

Algoritmus Isolation Forest[20] je naprosto odlišný od předchozích dvou uvedených algoritmů, na rozdíl od nich vůbec nepočítá vzdálenosti mezi body ani neodhaduje hustotu. Isolation Forest má lineární výpočetní složitost vzhledem k počtu bodů a nízké paměťové nároky. Jeho výpočetní náročnost prakticky neroste se zvyšující se dimenzí dat. Algoritmus je založen na předpokladu, že anomálií je v datech výrazně méně než normálních bodů a hodnoty jednotlivých atributů se u nich velmi liší od normálních bodů.



Obrázek 3.7: Ukázka separace dat



Obrázek 3.8: Ukázka izolačního stromu

Základní myšlenkou algoritmu je, že zatímco normální data se od sebe obtížné oddělují, anomální data od normálních lze izolovat snadno. Anomální data mají vyšší pravděpodobnost, že budou při náhodném rozdělování prostoru brzy oddělena od normálních dat. Obrázek 3.7 ilustruje myšlenku algoritmu na 2D datech, zobrazuje shluk bodů vygenerovaných náhodně z normálního rozdělení a 2 navíc přidané body v pravém horním rohu. Při vytváření izolačního stromu se náhodně rozděluje prostor. Už po dvou náhodných rozděleních prostoru jsou přidané body izolovány od ostatních bodů. Normální data od sebe naopak oddělena příliš nejsou.

Metoda funguje ve dvou fázích. V první se z náhodně vybraných vzorků vytvoří předem daný počet stromových struktur podobných binárním vyhledávacím stromům. Ve druhé fázi se pro každý vzorek dat porovnávají určité vlastnosti s již vytvořenými stromy. Počet stromů je určen předem, [20] doporučuje 100 stromů.

Stromy vytořené v první fázi algoritmu Isolation Forest jsou plné binární stromy, které v každém vnitřím uzlu obsahují klíč podobně jako binární vyhledávací stromy a dále také i index atributu v datech, kterého se tento klíč týká. Listy stromu obsahují číslo udávající zbývající počet prvků. Na obrázku 3.8 je ukázka izolačního stromu, čím tmavší barva uzlu, tím větší je počet vzorků v podstromu s tímto uzlem jako kořenem.

Konstrukce stromu je jednoduchá. Nejdříve se náhodně vybere předem daný počet vzorků pro tvorbu stromu. (označovaný jako sampling size, v [20]

je doporučeno 256). Strom se vytváří rekurzivním algoritmem, který v každém kroku náhodně vybere jeden atribut a náhodně zvolí hodnotu v rozsahu tohoto atributu ve vzorku. Stejně jako v binárním vyhledávacím stromu, v levém podstromu jsou data s hodnotou menší (nebo rovnou) než klíč a v pravém s hodnotou větší než klíč. Každý vnitřní uzel stromu tak vlastně rozdělí prostor na 2 podprostory. List je vytořen, pokud počet vzorků v předaných datech je menší nebo roven 1, případně pokud je dosaženo maximální povolené hloubky stromu $log_2(sampling\_size)$.

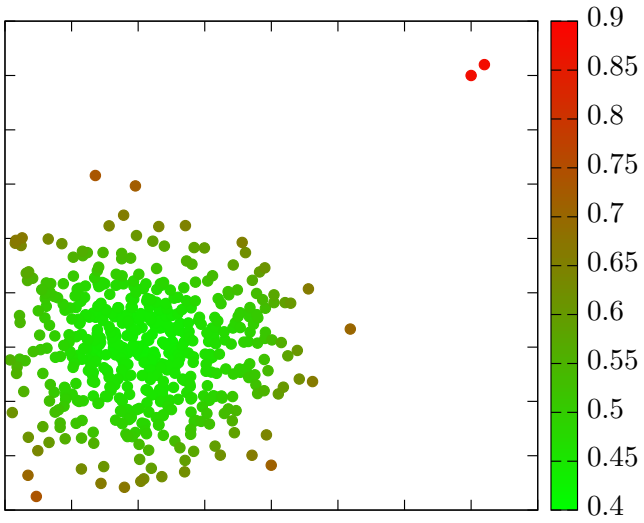
V druhé fázi se každý vzorek nechá projít všemi vytvořenými stromy, jako by to byly binární vyhledávací stromy. Vzorek prochází stromem, dokud nedorazí k listu. Při pohybu stromem se zároveň počítá délka průchodu vzorku stromem, každá hrana přidává 1 do délky cesty. Pokud vzorek skončí v listu, kde je více než jeden prvek, odhadne se zbývající délka cesty pomocí vzorce 3.4. Tento vzorec se používaná pro odhad délky neúspěšného vyhledávání u BST s daným počtem uzlů. $H(n)$ v tomto vzorci je i-té harmonické číslo, které můžeme odhadnout jako $ln(n) + e$. Tento odhad představuje délku cesty, kterou bychom museli projít, kdyby byl strom plně rozvinutý.

$$c(n) = 2H(n-1) - 2(n-1)/n \tag{3.4}$$

Spočítané délky cest pro všechny stromy se potom zprůměrují a normalizují podle vzorce 3.5 na rozsah <0,1>. Pokud vzorek bude mít průměrné délky cest ve stromech, bude mít hodnotu blízkou 0.5. Čím delší bude průměrná délka cest ve stromech, tím bude výsledné skóre blíže 0, naopak pro kratší průměrné délky cest bude výsledné skóre blíže 1.

$$s(A) = 2^{-\frac{E(h(A))}{c(n)}} \tag{3.5}$$

Předpokládá se, že anomálie mají v jednotlivých atributech výrazně odlišné hodnoty od ostatních bodů, proto mají vyšší také pravděpodobnost, že při náhodné tvorbě stromu budou brzy odděleny od zbytku dat. S každým dalším stromem tato pravděpodobnost narůstá. Pokud vzorek často prochází stromy jen do malé hloubky (jeho výsledná hodnota bude vyšší než 0.5), je pravděpodobné, že vzorek je anomální. Míru jeho anomálie určuje srovnání s průměrnou hloubkou stromů. Obrázek 3.9 demonstruje výsledky algoritmu na datech tvořených 550 náhodnými body z normálního rozdělení a 2 uměle přidanými body v pravém horním rohu. Jako nejvíce anomální byly podle očekávání označeny přidané body.



Obrázek 3.9: Ukázka výsledků algoritmu Isolation Forest

Kapitola 4

Implementace a testování

4.1 Implementace

4.1.1 Volba technologií

K implementaci jsem zvolil jazyk C++ ve verzi C++11. Hlavním důvodem pro toto rozhodnutí byly mé předchozí zkušenosti s ním. Programy byly vyvíjeny a testovány pod systémem Linux. Některé časově náročné části kódu je možné jednoduše zrychlit pomocí paralelizace, k tomu jsem využil API OpenMP[21].

Implementované řešení sestává z několika programů - jeden pro každý algoritmus a program pro dopočítání dalších atributů (viz. část 4.2.2). Programy jsou čistě výpočetní a nemají žádné GUI. Všechny programy očekávají 2 parametry, prvním z nich je název vstupního souboru a druhým název výstupního souboru.

Vstup programy očekávají ve formátu CSV (Comma Separated Values). Tento formát je velice jednoduchý, každý záznam je na jedné řádce a jednotlivé atributy záznamů jsou od sebe odděleny čárkou. První řádek souboru je chápán jako hlavička. Data o síťových tocích lze ve formátu CSV přímo exportovat z programu nfdump. Program pro dopočítání dalších atributů očekává, že data budou mít jména atributů v hlavičce tak, jak je exportuje nfdump. Pro tento program je důležitý význam jednotlivých atributů. Programy implementující algoritmy se o jejich význam nijak nezajímají.

Všechny programy využívají společný základ pro načítání a výpis dat reprezentovaný třídou `Flows`. V ní se po celou dobu běhu programu uchovávají záznamy o síťových tocích. Metody této třídy umožňují načíst data ze souboru, pracovat s nimi a potom je opět vypisovat.

4.1.2 Načtení dat

Třída `Flows` obsahuje funkci `parseData`, která slouží k načítání dat z CSV souboru. Jednotlivé atributy se interně ukládají jako reálná čísla. Hodnoty atributů síťových toků exportovaných z nfdumpu ale nejsou vždy pouze čí-

selná. Data obsahují např. datum a čas, IP adresy, flagy nebo protokol. Pro jejich zpracování popsanými algoritmy bylo třeba tyto údaje reprezentovat jako reálná čísla.

Zjevným řešením je přiřadit všem unikátním hodnotám nečíselných atributů čísla a místo skutečných hodnot ukládat pouze tato čísla. Ta ale potom nereflektují žádné souvislosti, které mohly být mezi originálními hodnotami. Navíc, pokud chceme původní hodnoty později vypisovat, musíme si udržovat v paměti tabulku mapování mezi číselnými a textovými hodnotami. Proto jsem se snažil v implementaci provést převod způsobem, který více zachovává jejich charakter.

Jedním z nečíselných atributů jsou IP adresy. Obvykle se IPv4 adresa zapisuje jako čtveřice čísel (v rozsahu 0-255) oddělená tečkami. To je ale jen konvence zápisu, IPv4 adresa ve skutečnosti není nic jiného než 32bitové číslo. Vhodnějším způsobem reprezentace IPv4 adresy je pouze její převedení textové reprezentace na číselnou. Tento způsob jednak zachovává souvislosti (např. sousední adresy v rozsahu) a za druhé není potřeba žádná tabulka mapování.

Nečíselným atributem je také datum a čas. Často používaným způsobem reprezentace datumu a času v počítačích je tzv. unix timestamp, který reprezentuje časový okamžik jako počet sekund od 1. 1. 1970. Ukládání datumu a času v tomto formátu je také vhodnější než mapování řetězců na čísla.

Metoda `parseData` při načítání testuje, zda hodnoty atributů nepředstavují IP nebo datum a pokud ano, použije pro ně specifický formát. Mapování netextových hodnot na čísla použije pouze tehdy, pokud je nedokáže využít lépe. V základních datech zůstaly pouze 2 atributy, které jsem nedokázal využít lépe - protokol a flagy.

Po načtení data zůstávají ve třídě Flows a je možné s nimi přímo pracovat. Třída Flows přetěžuje operátor indexace, který vrací konkrétní tok reprezentovaný jako pole hodnot typu double.

4.1.3 Normalizace dat

Algoritmy, které počítají rozdíly mezi atributy vyžadují normalizaci jejich hodnot. Normalizace eliminuje rozdílné rozsahy hodnot. Například zjevně rozdílný rozsah hodnot je mezi počtem bajtů a trváním toku v sekundách, počty bajtů mají o několik řádů vyšší hodnoty. Bez normalizace by atributy s velkými rozsahy hodnot zcela převážily ostatní.

Třída `Flows` nabízí funkci `normalize_data`, která provede na jednotlivých atributech tzv. z-index normalizaci, definovanou vztahem:

$$x_{new} = \frac{x - \mu}{\sigma},$$

kde μ je střední hodnota a σ směrodatná odchylka. Střední hodnotu určíme

jako průměr hodnot atributu a směrodatnou odchylku spočítáme podle vzorce

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2 - \mu^2}.$$

Po normalizaci je průměr hodnot 0 a většina hodnot velmi blízká nule. Normalizace pouze zmenšuje rozsah hodnot, ale zachovává pořadí i extrémní hodnoty. Ty, které jsou výrazně odlišné od průměru v původních datech, budou i po normalizaci výrazněji odlišné od 0.

4.1.4 Výpis dat

Všechny programy implementující algoritmy jsou navrženy tak, že data po zpracování vypíší do souboru. Pro snadnější interpretaci výsledků se do výstupního souboru vypíší všechny originální atributy dat a přidá se k nim nový atribut, který obsahuje vypočítaný stupeň odlehlosti. Data se vypíší seřazená podle stupně odhlehlosti, což usnadňuje orientaci ve výsledcích.

4.1.5 Optimalizace hledání nejbližších sousedů

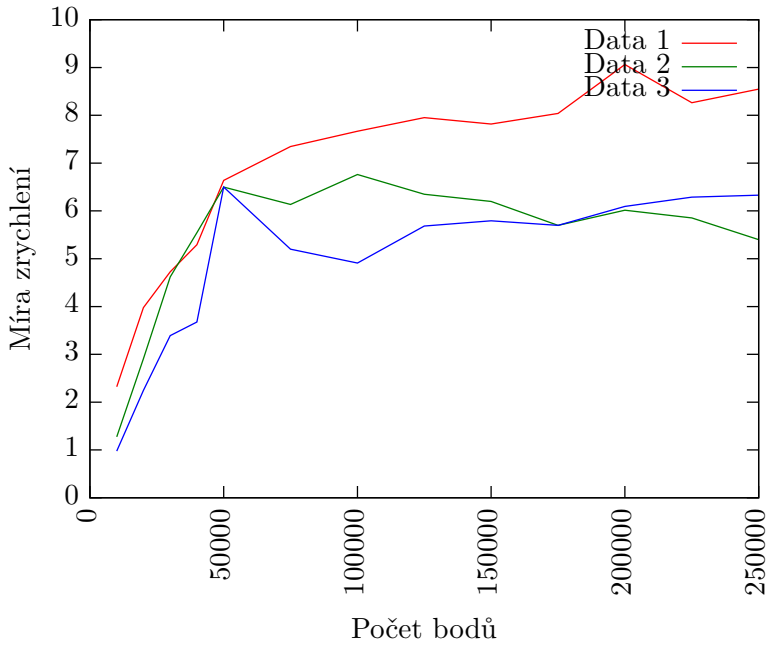
V části 3.1.1 jsem teoreticky popisoval optimalizaci algoritmu kNN pomocí shlukování. Diskutována byla zejména volba vhodného poloměru shluku. Jak jsem zjistil, optimální poloměr pro vzorek a pro všechna data je často velmi podobný.

V implementaci se pro odhad poloměru používá vzorek 5000 záznamů, na kterém se následně algoritmus testuje s různými poloměry. Hodnotu 5000 jsem zvolil proto, aby testování bylo co nejvíce vypovídající a zároveň rychlé. Poloměr se na začátku nastaví na 0,1 (data se pro algoritmus kNN normalizují, proto nezáleží na vzdálenostech originálních dat) a postupně se zvětšuje. Testování skončí ve chvíli, kdy se algoritmus přestane zlepšovat (nebo při dosažení zvoleného maximálního poloměru). Nejlepší nalezený poloměr vzorku se následně použije pro algoritmus spuštěný na všech datech. Hledání nejlepšího poloměru na vzorku trvá na mém počítači s procesorem Intel i3-2330M pouze několik vteřin.

Graf 4.1 ukazuje zrychlení optimalizované verze oproti naivní na třech různých typech dat. Poloměr pro algoritmus byl automaticky zvolen výše popsaným způsobem. Na grafu je vidět, že zrychlení dosažené díky této optimalizaci je velmi závislé na konkrétních datech. Hlavní roli hraje rozložení bodů v prostoru.

4.2 Testování

Testování algoritmu probíhalo na NetFlow datech zachycených ze sítě. Pozoroval jsem, které toky algoritmy označují jako nejvíce anomální a úspěšnost



Obrázek 4.1: Zrychlení optimalizované verze algoritmu kNN

detekce uměle vygenerovaných síťových útoků. Počet toků používaných k testování se podle vytížení sítě pohyboval mezi 30 a 100 tisíci.

4.2.1 Testování se základními daty

Jednotlivé algoritmy jsem testoval na datech z nfdumpu a pozoroval, které toky algoritmy vyhodnocují jako nejvíce anomální. Protože výsledky detekce anomálií jsou velmi závislé na normálních datech, popíšu nejdříve, jak vypadá obvyklý provoz na sledované síti.

Nejčastějšími toky pozorovanými v datech byly spojení k webovým serverům (porty TCP/80 nebo TCP/443). Tyto toky většinou trvaly krátce a přenášely pouze malé množství dat. Většinu z nich lze interpretovat jako stažení jedné webové stránky ze serveru. Dalším velmi často pozorovaným typem provozu byly DNS dotazy (na porty UDP/53 nebo TCP/53). Tyto dotazy často předcházely spojení na webové servery - DNS dotazem byla získána IP adresa a na ní pak vytvářeny další toky. DNS dotaz je pouze příprava pro další spojení, hlavním účelem není dotaz samotný, ale spojení které následuje po něm. Spojení na webové servery a DNS servery dohromady tvořilo více než 80% veškerého zachyceného provozu.

Tabulka 4.1: Zastoupení útoku v datech

Typ útoku	Toky útoku
Vertikální sken	2,07%
Horizontální sken	3,63%
HTTP DoS	11,84%
pomalý HTTP DoS	6,45%

Tabulka 4.2: Detekce útoků na základních datech

Typ útoku	kNN	iForest	LOF
Vertikální sken	2,24%	6,72%	1,70%
Horizontální sken	2,19%	3,67%	3,74%
HTTP DoS	88,16%	85,16%	1,57%
pomalý HTTP DoS	18,58%	39,33%	1,03%

4.2.1.1 Detekce umělých útoků

Při zachytávání síťových toků jsem vyráběl různé druhy síťových útoků (popsaných v kapitole 2.3). Tabulka 4.1 ukazuje poměrné zastoupení toků útoku v datech. Algoritmy jsem spouštěl na zachycených datech a následně sledoval, kolik toků bude mít vyšší stupeň odlehlosti než toky z útoku.

Výsledky ukazuje tabulka 4.2. Tabulka uvádí počet procent toků hodnocených jako anomálnější než nejanomálnější tok útoku. Vidíme, že toků označených jako více anomálních než útok je mnoho a bylo by velmi obtížné útok najít, pokud bychom o něm neměli žádné konkrétní informace.

Výsledky ukazují na relativní úspěšnost všech algoritmů při detekci síťových skenů. Toky vytvořené síťovými skeny jsou ve srovnání s HTTP DoS útoky vzájemně méně podobné, liší se porty a cílovými adresami a je jich menší počet. HTTP DoS útoky vytvoří velké množství prakticky stejných toků, které se liší pouze minimálně, například drobnými rozdíly v době trvání toku nebo času začátku toku. Na výsledcích vidíme velmi malou úspěšnost algoritmů kNN a Isolation Forest při detekci HTTP DoS útoků. Špatné výsledky způsobilo, že všechny simulované útoky byly poměrně masivní a vygenerovaly velké množství velmi podobných toků. Velké množství podobných toků pro tyto algoritmy přestává být anomální. Mírně lepší výsledky při detekci pomalého DoS útoku jsou způsobeny menším počtem toků a většími odlišnostmi mezi nimi.

Výrazně se liší výsledky algoritmu LOF. Jeho úspěšnost lze obtížně hodnotit, vzhledem k jeho velké citlivosti na mírné odchylky v datech. Například u útoku HTTP DoS algoritmus LOF detekoval jako velmi anomální tok, který trval (v rámci útoku) nezvykle delší dobu. Zatímco absolutní většina toků útoku probíhala pouze setiny sekundy, objevilo se mezi nimi několik toků o délce přibližně jedné sekundy. Takové toky byly pak detekovány jako výrazně

Tabulka 4.3: Vliv parametru k na výsledky kNN a LOF - horizontální sken

k	kNN	LOF
20	2,06%	4,20%
30	2,20%	4,40%
40	2,21%	3,72%
50	2,24%	3,58%
100	2,19%	3,74%
200	1,53%	3,82%
500	1,09%	4,18%

Tabulka 4.4: Vliv parametru k na výsledky kNN a LOF - DoS útok

k	kNN	LOF
20	83,72%	1,22%
30	84,27%	1,41%
40	86,42%	1,45%
50	87,43%	1,47%
100	88,16%	1,57%
200	88,16%	1,72%
500	88,16%	2,03%

více anomální než zbytek útoku.

Algoritmus Isolation Forest je randomizovaný a jeho výsledky při testování mírně kolísaly. Výsledky se shodovaly v nejvýše hodnocených anomalitách, ale u méně anomálních toků se pořadí mírně lišilo. Hodnoty uvedené v tabulce výsledků 4.2 jsou průměrem z pěti běhů algoritmu.

Parametrem algoritmů kNN a LOF je počet nejbližších sousedů. Tento parametr určuje maximální velikost shluku, který se ještě může považovat za anomální a teoreticky by mohl řešit problém s velkým počtem podobných bodů. Tabulka 4.3 ukazuje vliv parametru k na výsledky algoritmů kNN a LOF při detekci vertikálního skenu. Vidíme, že se zvětšováním tohoto parametru se výsledky zlepšují. Naproti tomu, u DoS útoků se zvětšováním parametru výsledky nezlepšují, jak ukazuje tabulka 4.4. Důvodem je, že útok je velmi masivní a podobných toků vzniklo obrovské množství. Ideální volba parametru k velmi závisí na charakteru útoku a sama o sobě nestačí k zlepšení detekce. Volba nízkého k vede k méně stabilním výsledkům. Jako kompromisní hodnotu jsem zvolil $k = 100$ a tuto hodnotu použil pro všechna testování.

4.2.2 Separace útoků pomocí dalších atributů

Pokusy s detekcí anomálií na základních NetFlow datech ukázaly, že útoky vytvoří v základních datech mnoho velmi podobných záznamů a algoritmy je potom považují za normální. Pro zmenšení podobnosti toků útoku navrhuje

Tabulka 4.5: Detekce útoků na datech s dopočítanými atributy

Typ útoku	kNN	iForest	LOF
Vertikální sken	0,87%	4,59%	0,82%
Horizontální sken	0,70%	2,70%	0,09%
HTTP DoS	63,16%	14,76%	0,13%
pomalý HTTP DoS	7,01%	9,01%	0,07%

článek [22] dopočítat ke každému toku další umělé atributy pro zachycení chování zdroje a cíle v minulosti.

Článek popisuje projekt MINDS (Minnesota Intrusion Detection System), který zmiňuje dopočítávání několika nových atributů (v terminologii data miningu nazývané features) pro separaci síťového chování projevujícího se vytvářením velkého počtu toků. Tyto nové atributy mají zachytit určité vazby jednotlivých záznamů mezi sebou. V článku jsou uvedeny 4 nově dopočítané typy atributů:

- Počet unikátních cílových adres ze stejné zdrojové adresy
- Počet spojení na stejný cílový port ze stejné zdrojové adresy
- Počet unikátních zdrojových adres na stejnou cílovou adresu
- Počet spojení ze stejného zdrojového portu na stejnou cílovou adresu

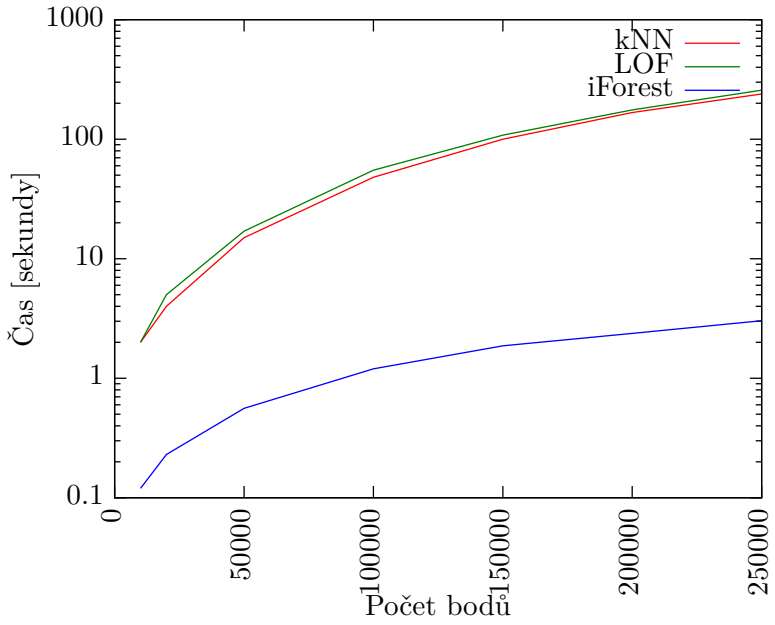
Atributy se dopočítávají buď v časovém okně (posledních X sekund) nebo během posledních X spojení. Časové okno je vhodné pro separaci aktivit, které se projevují rychlým vytváření velkého počtu toků. Okno počtu spojení se podle [22] hodí například pro separaci pomalých skenů.

4.2.3 Experimenty na rozšířených datech

Výsledky detekce anomálií na datech rozšířených o nové atributy shrnuje tabulka 4.5, která udává kolik procent toků je hodnoceno jako více anomální než nejanomálnější tok útoku. Dopočítání dalších atributů výrazně zlepšilo detekci všech testovaných typů útoku proti základním atributům.

Nejlepších výsledků dosahoval algoritmus LOF. Algoritmus LOF detekoval v místech s vysokou hustotou i malé odchylky. Přidané atributy zvýšily vzdálenosti mezi body, což zvýšilo odlehlost některých toků útoku.

Algoritmus kNN měl lepší výsledky při detekci skenů, naopak při detekci DoS útoku byly lepší výsledky algoritmu Isolation Forest. Zvlášť výrazný je rozdíl v případě běžného HTTP DoS útoku, ve kterém algoritmus kNN zcela propadl. Útok HTTP DoS vytvořil mnoho vzájemně velmi podobných toků a ani přidané atributy nedokázaly vzdálenosti mezi nimi výrazněji zvýšit.



Obrázek 4.2: Závislost doby běhu algoritmů na počtu bodů

4.2.4 Doba běhu algoritmů

U algoritmů jsem testoval také dobu jejich běhu. Graf 4.2 zobrazuje závislost doby běhu na počtu toků. Můžeme pozorovat, že zatímco doba běhu algoritmů kNN a LOF s množstvím dat rychle narůstá, doba výpočtu algoritmu Isolation Forest roste velmi pomalu. Test byl proveden na procesoru Intel i3-2330M v jednom vlákně, program jsem zkompiloval s použitím nejvyšší úrovně optimalizace -Ofast.

Závěr

V rámci této práce jsem popsal postupy a nástroje pro zachycení síťového provozu a vytvoření umělých útoků, představil jsem metody detekce anomálií založené na strojovém učení a implementované algoritmy na zachycených datech otestoval.

O výsledcích detekce anomálií lze obecně říci, že algoritmy kNN a Isolation Forest shodně jako nejvíce anomální detekovaly síťové toky s velkým počtem přenesených dat. Ty byly zvláštní ve více ohledech – v počtech přenesených bajtů a paketů (přijatých i odeslaných) i v délce trvání. Většina ostatních toků naproti tomu byla krátká a bylo při nich přeneseno pouze malé množství dat.

Výsledky algoritmu LOF byly výrazně odlišné. Algoritmus LOF neoznačoval jako nejvíce anomální toky s velkými počty přenesených dat, ale spíše neobvyklé kombinace zdrojových a cílových portů (nízké zdrojové porty nebo vysoké cílové porty). Je obtížné zpětně odhadnout, co tyto toky znamenaly. Podle provozu ze zdrojové adresy ve stejném čase odhaduji, že některé vysoce anomální toky jsou spojení P2P protokolů (BitTorent nebo podobné). Tyto protokoly často používají náhodně zvolené porty[23].

Při testování na zachycených datech jsem zjistil, že algoritmy umělé útoky nerozpoznají jako výrazně anomální. Za důvod považuji velké množství velmi podobných síťových toků vygenerovaných útoky. Algoritmy kNN a Isolation Forest nepovažují shluky podobných dat za anomální. Nejlépe útoky detekoval algoritmus LOF, který rozpoznal i malé odchylky ve shlucích podobných dat.

Pro zlepšení detekce jsem vyzkoušel metodu dopočítávání dalších statistických atributů popsanou v [22]. Ukázalo se, že použití těchto atributů umožňuje výrazně zvýšit úspěšnost detekce útoků. Nejlepších výsledků na rozšířených datech dosahoval opět algoritmus LOF, který detekoval všechny testované útoky mezi jedním procentem nejvíce anomálních toků. Výsledky algoritmů kNN a Isolation Forest byly horší. Článek [20] na umělých datech ukazuje, že algoritmus Isolation Forest podává lepší výsledky než LOF, to se při testování na reálných datech nepotvrdilo.

I s použitím dopočítaných atributů bylo ale mnoho toků hodnoceno jako

více anomální než uměle vytvořené útoky. Jejich množství se pohybovalo pod jedním procentem, to ale i na poměrně malé testované síti znamená stovky toků. Při snaze o detekci útoků bychom zaznamenali velké množství falešných poplachů.

Zlepšení pozorované při použití dopočítaných atributů z [22] ukazuje možnou cestu pro další vylepšování detekce. Pomocí nových, více specializovaných atributů by zřejmě bylo možné detekci útoků dále zlepšovat.

Literatura

[1] pwc: 2014 Information Breaches Survey [online]. 2014, [cit. 2015-05-05]. Dostupné z: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/307296/bis-14-767-information-security-breaches-survey-2014-technical-report-revision1.pdf

[2] ENISA: Operation Black Tulip: Certificate Authorities lose authority [online]. 2011, [cit. 2015-05-05]. Dostupné z: https://www.enisa.europa.eu/media/news-items/operation-black-tulip

[3] Group, T. T.: Tcpdump [software]. 2015. Dostupné z: http://www.tcpdump.org/

[4] Cisco: Cisco IOS NetFlow. 2015. Dostupné z: http://www.cisco.com/web/go/netflow

[5] Wikipedia: NetFlow — Wikipedia, The Free Encyclopedia [online]. 2015, [cit. 2015-05-05]. Dostupné z: http://en.wikipedia.org/wiki/NetFlow

[6] Invea: FlowMon. 2015. Dostupné z: https://www.invea.com/cs/produkty-sluzby/flowmon

[7] Invea: FlowMon sonda [online]. 2015, [cit. 2015-04-22]. Dostupné z: https://www.invea.cz/data/flowmon/flowmon_probe_pb_cz.pdf

[8] fprobe: fprobe [software]. 2015. Dostupné z: http://fprobe.sourceforge.net/

[9] Foundation, W.: Wireshark [software]. 2015. Dostupné z: https://www.wireshark.org/

[10] Group, T. T.: PCAP-FILTER [online]. 2015, [cit. 2015-05-03]. Dostupné z: http://www.tcpdump.org/manpages/pcap-filter.7.html

[11] NFdump: NFdump [software]. 2015. Dostupné z: http://nfdump.sourceforge.net/

[12] Nmap: Nmap [software]. 2015. Dostupné z: https://nmap.org/

[13] Nmap: Nmap Scripting Engine (NSE) [online]. 2015, [cit. 2015-05-05]. Dostupné z: http://nmap.org/book/man-nse.html

[14] LOIC: LOIC [software]. 2015. Dostupné z: http://loic.sourceforge.net/

[15] Čepský, P.: Útoky jménem Anonymous: Jak se rodí hackeři? [online]. *Lupa.cz*, Únor 2012, ISSN 12130702, [cit. 2015-04-29]. Dostupné z: http://www.lupa.cz/clanky/utoky-jmenem-anonymous-jak-se-rodi-hackeri/

[16] Hansen, R.: Slowloris [software]. 2009. Dostupné z: http://ha.ckers.org/slowloris/

[17] Prerau, M. J.; Eskin, E.: Unsupervised anomaly detection using an optimized K-nearest neighbors algorithm. *Undergraduate Thesis, Columbia University: December*, 2000. Dostupné z: http://www.music.columbia.edu/~mike/publications/thesis.ps

[18] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; aj.: LOF: Identifying Density-based Local Outliers. *SIGMOD Rec.*, ročník 29, č. 2, Květen 2000: s. 93–104, ISSN 0163-5808, doi:10.1145/335191.335388. Dostupné z: http://webdocs.cs.ualberta.ca/~zaiane/pub/check/breunig.pdf

[19] Wikipedia: Local outlier factor — Wikipedia, The Free Encyclopedia [online]. 2015, [cit. 2015-05-05]. Dostupné z: http://en.wikipedia.org/wiki/Local_outlier_factor

[20] Liu, F.; Ting, K. M.; Zhou, Z.-H.: Isolation Forest. In *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, Dec 2008, ISSN 1550-4786, s. 413–422, doi:10.1109/ICDM.2008.17.

[21] openMP: openMP [software]. 2015. Dostupné z: http://www.openmp.org/

[22] Ertoz, L.; Eilertson, E.; Lazarevic, A.; aj.: Minds-minnesota intrusion detection system. *Next Generation Data Mining*, 2004: s. 199–218. Dostupné z: http://www.cs.umn.edu/~kumar/papers/minds_chapter.pdf

[23] Karagiannis, T.; Broido, A.; Faloutsos, M.; aj.: Transport layer identification of P2P traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, ACM, 2004, s. 121–134. Dostupné z: http://conferences.sigcomm.org/imc/2004/papers/p121-karagiannis.pdf

Příloha **A**

Seznam použitých zkratek

CSV Comma Separated Values

DNS Domain Name System

DoS Denial of Service, typ útoku, který se snaží zahltit cíl

ENISA European Network and Information Security Agency

FTP File Transfer Protocol

GUI Graphical User Interface

HTTP HyperText Transfer Protokol, protokol pro přenos souborů

ICMP Internet Control Message Protocol

IP Internet Protokol

P2P Peer-to-peer, typ síťové komunikace, při které spolu komunikují přímo klienti

RFC Request For Comments, dokumenty popisující internetové protokoly

SSH Secure Shell

TCP Transmission Protocol

UDP User Datagram Protocol

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `data` ..... testovací data
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS