VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

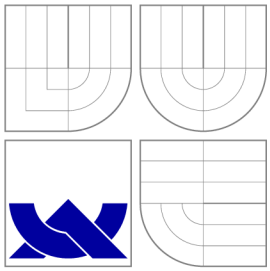
DETEKCE SKENOVÁNÍ PORTŮ NA VYSOKORYCHLOSTNÍCH SÍTÍCH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

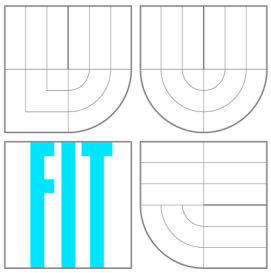
AUTOR PRÁCE
AUTHOR

DANIEL KAPIČÁK

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DETEKCE SKENOVÁNÍ PORTŮ NA VYSOKORYCHLOSTNÍCH SÍTÍCH

PORTSCAN DETECTION IN VERY HIGH-SPEED LINKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DANIEL KAPIČÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÁCLAV BARTOŠ

BRNO 2014

Abstrakt

V této práci budu prezentovat efektivní metodu detekce TCP skenování portů ve vysokorychlostních sítích. Hlavní myšlenka této metody je zahození co největšího množství paketů tak aby to nemělo vliv na přesnost. Ukážu, že pomocí dvojice Bloom filtrů lze zahodit v průměru až 80% ze všech paketů podílejících se na navázání TCP komunikace se zanedbatelným vlivem na přesnost. Toto výrazně redukuje požadavky na paměť a procesor. Dále budu prezentovat mnou navržený rozšíření algoritmu, které výrazně redukuje počet falešných hlášení způsobených absencí komunikace od serveru ke klientovi. Na závěr vyhodnotím algoritmus na zachycených vzorcích dat a online na sondě v síti CESNET. Výsledky ukáží, že tato metoda potřebuje méně než 2 MB paměti aby s velkou přesností vyhodnocovala provoz na vysokorychlostní síti. Díky malým paměťovým nárokům si tato metoda bez problémů vystačí s rychlou vyrovnávací pamětí většiny dnešních procesorů.

Abstract

In this thesis, I present the method to efficiently detect TCP port scans in very high-speed links. The main idea of this method is to discard most of the handshake packets without loss in accuracy. With two Bloom filters that track active destinations and TCP handshakes, the algorithm can easily discard about 80% of all handshake packets with negligible loss in accuracy. This significantly reduces both the memory requirements and CPU cost. Next, I present my own extension of this algorithm, which significantly reduces the number of false positives caused by the lack of communication from the server to the client. Finally, I evaluated this algorithm using packet traces and live traffic from CESNET. The result showed that this method requires less than 2 MB to accurately monitor very high-speed links, which perfectly fits in the cache memory of today's processors.

Klíčová slova

detekce anomálií, detekce skenování portů, detekce TCP skenování portů, detekce skenování portů na vysokorychlostních sítích, detekce TCP skenování portů na vysokorychlostních sítích

Keywords

anomaly detection, portscan detection, TCP portscan detection, portscan detection in very high-speed links, TCP portscan detection in very high-speed links

Citace

Daniel Kapičák: Detekce skenování portů na vysokorychlostních sítích, bakalářská práce, Brno, FIT VUT v Brně, 2014

Detekce skenování portů na vysokorychlostních sítích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vláclava Bartoše a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Daniel Kapičák
19. května 2014

Poděkování

Děkuji Ing. Václavu Bartošovi za odbornou pomoc a velkou ochotu.

Obsah

1 Úvod **3**

2 Typy útokov v počítačových sieťach **4**
2.1 Denial of Service . . . 4
2.1.1 SYN flood útok . . . 4
2.1.2 Internet Control Message Protocol (ICMP) flood . . . 5
2.1.3 Teardrop útok . . . 5
2.1.4 Peer to peer útok . . . 5
2.1.5 Distributed DoS . . . 5
2.2 Brute force útoky . . . 6
2.3 Útoky na aplikačnej úrovni . . . 6
2.3.1 SQL injection . . . 6
2.3.2 Buffer overflow . . . 7
2.4 Malware . . . 7
2.4.1 Počítačový červ . . . 7
2.5 Príprava na útok . . . 7

3 Skenovanie portov **8**
3.1 Popis . . . 8
3.2 Nadviazanie TCP spojenia . . . 8
3.3 Typy skenovania portov . . . 9
3.3.1 TCP connect scan . . . 9
3.3.2 SYN scanning . . . 9
3.3.3 UDP scanning . . . 10
3.3.4 ACK scanning . . . 10
3.3.5 Window scanning . . . 10
3.3.6 Ďalšie typy skenovania portov . . . 10

4 Popis použitej metódy **11**
4.1 Detekčný algoritmus . . . 11
4.1.1 Detekcia zlyhaných spojení . . . 12
4.1.2 Identifikácia skenerov . . . 13
4.2 Rozšírenie detekčného algoritmu . . . 15

5 Implementácia **17**
5.1 Trieda sniffer . . . 17
5.2 Trieda bloom . . . 17
5.3 Trieda span_dec . . . 18

5.3.1 Použité dátové štruktúry 18
5.3.2 Inkrementácia a dekrementácia počítadla elementu 19
5.4 Výstup programu 19

6 Výsledky **20**
6.1 Dáta pre vyhodnotenie 20
6.2 Vyhodnotenie 20
6.2.1 Príklad nastavenia vhodných parametrov algoritmu 20
6.2.2 Presnosť 21
6.2.3 Vplyv konfiguračných parametrov na presnosť 22
6.2.4 Využitie filtrov 24
6.2.5 Rýchlosť 26

7 Záver **27**

A Obsah CD **30**

B Manual **31**
B.1 Inštalácia 31
B.2 Parametre 31

Kapitola 1

Úvod

Každým dňom ľudia, organizácie a celý priemysel čoraz viac závisí na spolahlivosti a bezpečnosti internetového pripojenia. Avšak aj dnes takmer všetky priemyselné odvetvia alebo dokonca celé krajiny môžu byť cieľom internetového útoku. Väčšine internetových útokov predchádza fáza, v ktorej útočník skúma cieľový systém alebo systémy. Skenovanie portov je pravdepodobne najrozšírenejšia technika využívaná počítačovými červami ako aj ľudskými útočníkmi na odhalenie zranitelností cieľového systému alebo systémov. Problémom ako efektívne a spoľahlivo detekovať skenovanie portov sa zaoberá niekoľko prác[11, 14]. Bežné riešenia vyžadujú sledovanie jednotlivých sieťových spojení[6, 13, 14]. Avšak takéto riešenia nie sú vhodné pre vysokorýchlostné linky, kde sa môže vyskytovať veľké množstvo nezávislých spojení. V tomto prípade môžu mať naivné riešenia veľké požiadavky na DRAM a môžu vyžadovať niekoľko prístupov do pamäti na paket. Napriek tomu prístupové doby dnešných DRAM technológii sú omnoho väčšie než stredná doba medzi príchodmi paketov vo vysokorýchlostných sieťach (32 ns na OC-192 alebo 8 ns na OC-768 linkách). Vzorkovanie sieťovej prevádzky je považované za štandardné riešenie tohto problému. Nanešťastie, nedávne štúdiá[5, 7] ukázali, že vplyv vzorkovania sieťovej prevádzky na detekciu skenovania portov je veľmi veľky. Ďalšou alternatívou je použitie pravdepodobnostných, priestorovo efektívnych štruktúr, ktoré redukujú pamäťové nároky detekčných algoritmov[11, 15]. Tieto dátové štruktúry vďaka svojej malej pamäťovej náročnosti môžu byť uložené v rýchlych SRAM, ktoré majú prístupové doby menšie ako 10 ns.

Kapitola 2

Typy útokov v počítačových sieťach

V počítačových sieťach je za útok považovaný akýkoľvek pokus zničiť, odhaliť, zmeniť, zakázať, ukradnúť alebo získať neautorizovaný prístup k zdrojom na internete. Útoky sú zvyčajne páchané niekým so zlými úmyslami. Do tejto kategórie spadajú tzv. Black Hatted útoky[22]. Útoky môžu byť klasifikované podľa ich pôvodu, t.j. či sú vykonávané z jedného alebo z viacerých zdrojov. V druhom prípade sa útok nazýva distribuovaný. K vykonaniu distribuovaného útoku sa využíva botnet. Ďalšie rozdelenie je podľa použitého postupu pri útoku alebo podľa typu zneužitých slabých miest. Útoky môžu byť vedené na sieť mechanizmov alebo hostitelských funkcii.

Niektoré útoky sú fyzické, t.j. ukradnutie alebo poškodenie počítačov a iného príslušenstva. Ostatné sú pokusy o zmenu v logike používanej počítačmi alebo protokolmi používanými v počítačových sieťach, ktoré spôsobia nepredvídané výsledky. Avšak tieto výsledky sú užitočné pre útočníkov. Software určený pre logické útoky na počítačové systémy sa volá malware.

V nasledujúcich podkapitolách popíšem najviac rozšírené typy útokov vyskytujúcich sa v počítačových sieťach.

2.1 Denial of Service

Denial of Service (DoS) alebo Distributed Denial of Service (DDoS)[17] je technika útoku na sieťové zdroje, ktorá ma za cieľ tieto zdroje spraviť nedostupnými pre ostatných užívateľov. Hoci motívy a ciele DoS útokov môžu byť rôzne, väčšinou sa snažia dočasne alebo na neurčitú dobu pozastaviť alebo prerušiť služby bežiace na hostiteľskej stanici pripojenej k počítačovej sieti. Pre rozlíšenie, DDoS útoky sú vykonávaný dvoma alebo viacerými osobami alebo robotmi (botnet). DoS útoky sú vykonávané jednou osobou alebo jedným systémom. Jedna z najčastejších metód útoku umožňuje zahltenie cieľového zariadenia falošnými požiadavkami, a to natoľko, že zariadenie nie je schopné odpovedať na legitímne požiadavky alebo reaguje tak pomaly, že sa javí ako nedostupné. V nasledujúcich podkapitolách popíšem najčastejšie typy DoS a DDoS útokov.

2.1.1 SYN flood útok

SYN flood[19] nastane keď útočník zašle veľké množstvo TCP/SYN paketov, často s falošnou adresou odosielateľa.

2.1.2 Internet Control Message Protocol (ICMP) flood

Smurf attack

Smurf attack[18] je jednou z variánt záplavového DoS útoku. Tento útok spočíva na chybnej konfigurácii systému, ktorý umožní rozoslanie paketov všetkým počítačom v lokálnej sieti prostredníctvom Broadcast.

Ping flood

Ping flood[21] je založený na zahltení cieľového systému paketami typu ping. Tento typ útoku je najjednoduchší na vykonanie.

2.1.3 Teardrop útok

Teardrop útok zahŕňa zasielanie IP fragmentov s prekrývajúcim sa veľkým objemom dát. Chyba v TCP/IP pri preusporiadaní takéhoto paketu môže viesť u starších operačných systémov k ich pádu.

2.1.4 Peer to peer útok

Peer to peer útok využíva masívneho množstva ľudí na P2P sieťach. Útočníci našli cestu ako vzužiť chyby na P2P serveroch k zahájeniu DDoS útoku. Peer to peer útoky sú odlišné od útokov založených na botnet. Útočník využíva chybu v P2P klientovi k odpojeniu od aktuálneho serveru a pokus k pripojeniu k obeti. Pri dobre zorganizovanom útoku môže byť cieľový server vystavený naraz až 750 000 požiadavkám na pripojenie.

2.1.5 Distributed DoS

Distribuovaný DoS útok nastane keď sa viacero systémov snaží zahltiť prenosové pásmo alebo zdroje na cieľovom systéme, zvyčajne jeden alebo viac web serverov. Tento typ útoku je zvyčajne vedený bez vedomia majiteľov útočiacich systémov. Jeden zo spôsobov ako docieliť takéto správanie je infikovanie počítačov prostredníctvom malware, ktorý v sebe obsahuje IP adresu cieľa a dátum a čas zahájenia útoku. Systém môže byť tiež napadnutý programom, ktorý potom beží ako rezidentný proces a čaká na príkazy útočníka. Jedná sa o tzv. zombie, ktorý spolu s ostatnými počítačmi napadnutými rovnakým programom tvorí botnet.

DDoS má oproti DoS niekoľko výhod. Väčšie množstvo útočníkov môže generovať väčšie množstvo sieťovej prevádzky a tým spôsobí väčšiu záťaž cieľového systému. Útok viacerých útočníkov je ťažsie odhalitelný. Viac jednotlivých útočiacich systémov môže byť menej agresívnych na rozdiel od situácie keď útok vedie iba jediný systém. Výhodou je škálovatelnosť útoku kedy útočník môže meniť počet útočiacich systémov podľa potreby.

Reflected/Spoofed (DRDoS) útok

Distributed Reflected Denial of Service útok spočíva v rozoslaní podvrhnutých požiadaviek na veľké množstvo počítačov, ktoré na tieto požiadavky odpovedajú. Podvrhnuté požiadavky majú ako zdrojovú adresu uvedenú adresu obeti, ktorá je potom zahltená odpoveďami na tieto požiadavky. K vykonaniu tzv. odrazeného útoku útočníci môžu využiť veľa sieťových služieb. Jedna z variánt využíva DNS servery.

2.2 Brute force útoky

Útok hrubou silou je vo väčšine prípadov pokus o rozlúštenie šifry bez znalosti jej kľúča k dešifrovaniu. Jedná sa o systematické skúšanie všetkých kombinácii alebo ich podmnožiny. Útok hrubou silou sa často používa pre uhádnutie dvojíc užívateľ a heslo. Je možné používať náhodne prihlasovacie mená a heslá ale častejšie sa používajú rôzne obmedzenia pre kombinácie. Napríklad ak útočník získa zoznam užívateľských mien a skúša pomocou pripraveného slovníka rôzne heslá. Vďaka tomu, že si užívatelia väčšinou volia slabé heslá, je tento jednoduchý a automatizovatelný spôsob pomerne úspešný a rozšírený. V prípade, že útočník získa jednosmerne zašiforvané heslá, môže si pomocou slovníka rôzne heslá zašifrovať a porovnávať ich so zašifrovaným tvarom, ktorý získal. Ak nájde zhodu, získa prístup k danej službe. Z toho dôvodu je vhodné heslá ukladať na zabezpečené miesta.

2.3 Útoky na aplikačnej úrovni

Útoky na aplikačnej vrstve sa zameriavajú na aplikačné servery, na ktorých útočníci úmyselne spôsobujú chyby operačného systému alebo aplikácii. Chyby operačného systému alebo aplikácii umožňujú útočníkom obísť systém kontroly prístupu a tak získať kontrolu nad aplikáciami užívateľov, systémom a sieťou. To znamená že môže čítať, pridávať, mazať alebo modifikovať dáta užívateľov a taktiež môže vykonávať čokoľvek z nasledujúceho zoznamu:

- Čítať, pridávať, alebo modifikovať užívateľské dáta.
- Zaviesť vírus, ktorý využíva systém a jeho aplikácie ku kopírovaniu vírusov prostredníctvom siete, v ktorej je daný systém pripojený.
- Zaviesť sledovací program pre analýzu siete, do ktorej je postihnutý systém pripojený. Tatko získané informácie môžu byť použité pre zhodenie systémov v sieti alebo siete samotnej.
- catSCAN - Vypnúť aplikácie alebo operačné systémy.
- Vypnúť niektoré zabezpečenia pre umožnenie ďalších útokov v budúcnosti.

2.3.1 SQL injection

SQL injection je technika napadnutia databázovej vrstvy vsunutím SQL príkazu cez neošetrený vstup a vykonanie vlastnej SQL požiadavky. Toto nechcené chovanie nastáva pri prepojení aplikačnej a databázovej vrstvy. Takémuto útoku sa zabraňuje pomerne jednoducho, escapovaním potencionálne nebezpečných znakov.

SQL injection na internetových stránkach

Na internetových stránkach je SQL injection vykonávaný cez neošetrený formulár, manipuláciou URL, alebo podstrčením upravených cookies. Aj napriek pomerne jednoduchému ošetreniu je na internete stále veľké množstvo internetových stránok spravovaných neskúsenými programátormi, ktorí o tomto type útoku jednoducho nevedia a túto zranitelnosť nijak neošetrujú.

Ukážka útoku

Internetová stránka odosiela nasledujúcu požiadavku do databázy:

```
statement := "SELECT * FROM users WHERE login = '" + name + "';"
```

Ak útočník zadá:

```
x' or '1'='1
```

Internetová stránka doplní požiadavku a odošle ju vo forme:

```
statement := "SELECT * FROM users WHERE login = 'x' or '1'='1';"
```

Ak sa tento úsek SQL kódu použije pri autentizačnej procedúre, užívateľské meno bude vždy nájdené pretože podmienka $1 = 1$ je vždy pravidvá.

2.3.2 Buffer overflow

Bufer overflow alebo pretečenie zásobniká je stav kedy program zapisuje dáta do zásobníka, následkom čoho príde k jeho pretečeniu a prepíše sa susedná pamäť. Pretečenie zásobníka môže byť vyvolané vstupmi, ktoré majú vlastnosť spustitelného kódu. To môže spôsobiť nestále správanie programu. Napríklad chyby prístupu do pamäti, nesprávne výsledky, alebo narušenie bezpečnosti systému. Tieto chyby sú základom mnohých softvérových zraniteľností, preto sú často využívané k nezákonným zásahom do systémov.

2.4 Malware

Malware je počítačový program určený k vniknutiu a prípadne poškodeniu počítačového systému. Pod súhrnné označenie malware patria počítašové vírusy, trojské kone, spyware a adware. Najznámejšie typy malware, vírusy a červy sú známe najmä pre ich spôsob šírenia. Termínom počítačový vírus je označovaný program, ktorý infikuje spustitelný software a keď sa spustí, rozšíri sa do iných spustitelných aplikácii. Počítačový červ je program, ktorý aktívne prenáša sám seba cez sieť aby infikoval ostatné počítače.

2.4.1 Počítačový červ

Počítačový červ je počítačový program, ktorý automaticky rozosiela svoje kópie prostredníctvom siete na iné počítače. Potom, čo infikuje cieľový systém, prevezme kontrolu nad prostriedkami sieťovej komunikácie a využíva ich k svojmu vlastnému šíreniu.

2.5 Príprava na útok

Pre naplánovanie väčšiny typov útokov je najskôr potrebné analyzovať sieť a nájsť zranitelné počítače. Napríklad pre Brute force útok na SSH je najskôr potrebné zistiť, na ktorých počítačoch je otvorený port 22. K tomuto účelu sa využíva práve skenovanie portov. Významným zdrojom skenovania portov sú počítačoví červy pretože práve skenovaním portov sa snažia zistiť kam ďalej sa majú šíriť. Skenovanie portov popíšem v nasledujúcej kapitole.

Kapitola 3

Skenovanie portov

Skenovanie portov[20] je v informatike metóda zistovania otvorených sieťových portov na vzdialenom systéme v počítačovej sieti. Je tak možné zistiť aké služby sú na vzdialenom systéme spustené. Skenovanie portov je považované za nežiadúcu techniku pretože je často zneužívané útočníkmi na zistovanie slabých miest systémov. Táto technika môže mať však aj legitímne využitie ako nástroj pre zlepšovanie počítačovej bezpečnosti.

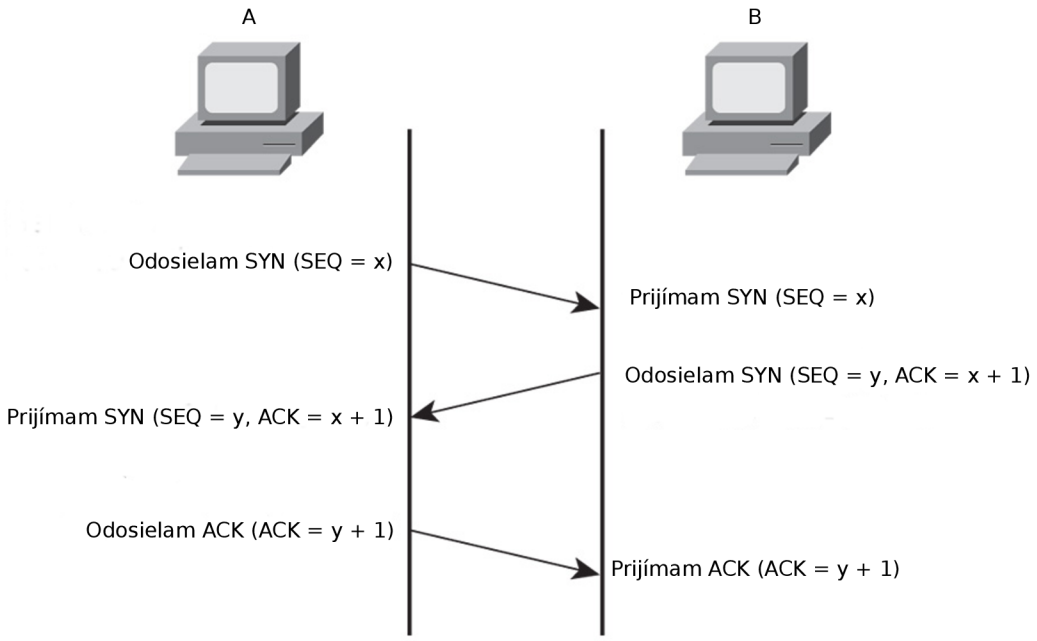
3.1 Popis

Sieťové služby fungujú typicky na princípe klient-server, kde serverová časť funguje v podobe démona. Aby sa k serverovej časti mohli pripájať klienti, musí serverová časť počúvať na sieťovom porte a reagovať na pokusy klienta. Komunikácia medzi klientom a serverom prebieha typicky pomocou protokolov TCP a UDP. V obidvoch prípadoch je možné simulovať klienta, ktorý má o službu záujem a vyžiadať si tak reakciu serverovej časti. Pretože implementácia rodiny protokolov TCP/IP je veľmi komplikovaná a RFC ju typicky nedefinuje v okrajových podmienkach, je možné pomocou vhodných techník rozoznať od seba rôzne operačné systémy a dokonca aj rôzne verzie týchto systémov. Skenovanie portov komplikuje firewall, ktorý môže pokusy o spojenia blokovať alebo rozoznať skenovanie portov hneď v počiatku a prevádzku blokovať.

3.2 Nadviazanie TCP spojenia

TCP je spojovo orientovaný protokol. Z toho vyplýva, že prenosu dát predchádza fáza nadviazania spojenia. Tento proces sa nazýva tzv. 3 way handsake pretože pozostáva z troch fáz. Schéma na obrázku 3.1 podrobne popisuje proces nadviazania TCP spojenia. V prvej fáze klient (A) pošle SYN (synchronizačnú) správu serveru (B) s náhodným sekvenčným číslom x. V druhej fáze ak server prijme túto komunikáciu, odpovie SYN+ACK (synchronizácia potvrdená) správou s hodnotou x + 1. To znamená, že ďalšie číslo, ktoré server bude očakávať je x+1. Server si tiež vygeneruje náhodné sekvenčné číslo y, ktoré odošle spolu s číslom x+1 klientovi. Následne v tretej a záverečnej fáze klient pošle serveru ACK (potvrdzujúcu) správu s číslom y+1. Po tejto fáze je TCP spojenie nadviazané.

Obrázek 3.1: Diagram detekčného algoritmu



3.3 Typy skenovania portov

Vďaka zložitosti rodiny protokolov TCP/IP je možné použiť viaceré techniky skenovania portov [20]. Tieto techniky popíšem v nasledujúcich podkapitolách.

3.3.1 TCP connect scan

TCP connect scan je základnou metódou skenovania portov. Využíva sieťové funkcie operačného systému. Ak je port otvorený, funkcia connect() prebehne úspešne a operačný systém dokončí 3 way handshake. Potom sa okamžite zavrie spojenie aby sa zabránilo realizácii určitej varianty DoS útoku. V prípade, že je port zatvorený, funkcia connect() vráti chybový kód. Tento typ skenovania je výhodný v tom, že nepotrebuje žiadne zvláštne privilégiá v operačnom systéme. Na druhej strane používaním sieťových funkcii operačného systému nemá táto metóda kontrolu nad nízkoúrovnovými prvkami systému a preto nie je príliš využívaná. Taktiež je IP adresa skenera portov ľahko odhalitelná.

3.3.2 SYN scanning

SYN scanning je určitou formou TCP connect scan. Táto metóda nevyužíva sieťové funkcie operačného systému ale generuje RAW pakety a vytvára monitory pre odpovede. Táto metóda je tiež známa ako tzv. half-open scanning pretože nikdy neotvára kompletné TCP spojenie. Skener portov generuje SYN paket. Ak je cieľový port otvorený, server odpovie poslaním SYN+ACK paketu. Klient odpovie poslaním RST paketu.

Vyuzžitie RAW má niekoľko výhod. Skenery portov majú plnú kontrolu nad odosielanými paketmi, hodnotami časovačov a pod.

3.3.3 UDP scanning

UDP scanning posiela paket na port, ktorý keď nie je otvorený, cieľový systém odpovie správou ICMP typu port unreachable. Ak cieľový systém neodpovie, znamená to, že port je otvorený. Avšak ak je port blokovaný firewallom, bude táto metóda považovať port za otvorený. Táto metóda je veľmi pomalá.

3.3.4 ACK scanning

ACK scanning je unikátnym typom skenovania portov. Pri tejto metóde nemožno presne určiť, ktorý port je otvorený alebo uzavrený. Pomocou ACK skenovania zistíme či je port filtrovaný alebo nie. Táto vlastnosť je obzvlášť výhodná pre zistovanie nastavení firewallu.

3.3.5 Window scanning

Window scanning súvisí s veľkosťou okna TCP. Je pomerne nespoľahlivý pri zistovaní otvorených alebo uzatvorených portov. Táto metóda je princípom podobná ACK skenovaniu.

3.3.6 Ďalšie typy skenovania portov

- FIN scanning
- X-mas a NULL scanning
- Proxy scanning
- catSCAN - kontroluje porty
- ICMP scanning - či počítač reaguje na výzvy pomocou protokolu ICMP

Kapitola 4

Popis použitej metódy

Táto práca sa zaoberá implementáciou metódy skenovania portov popísanej v [10]. Táto metóda patrí medzi metódy, ktoré analyzujú sieťovú prevádzku na úrovni paketov. Je navrhnutá tak aby spracovávala čo najväčšie množstvo paketov a mohla tak pracovať na vysokorýchlostných sieťach (10 Gb/s).

Kľúčovou myšlienkou použitej metódy je zahodenie väčšiny TCP paketov, ktoré sa podieľajú na nadviazaní komunikácie pri zachovaní schopnosti úspešného odhalenia skenovania portov.

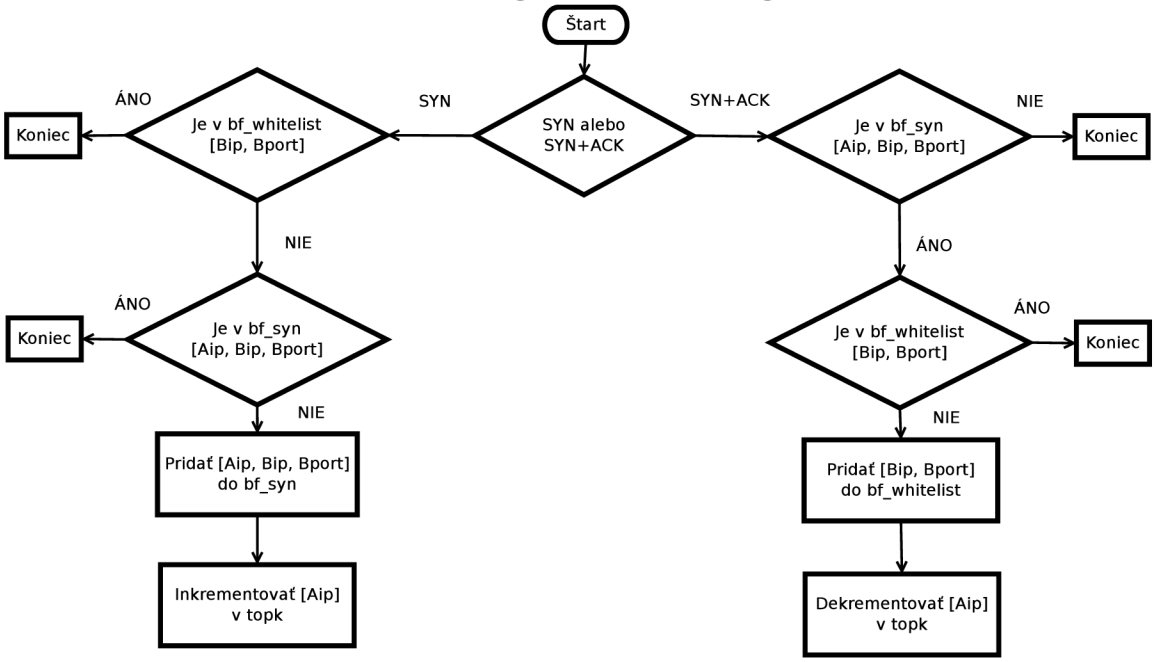
V prvom kroku sa ignorujú všetky legitímne pokusy o nadviazanie komunikácie pomocou zoznamu aktívnych serverov, ktorý uchováva IP adresu servera a port, na ktorom beží konkrétna služba. Ďalej sa zahodia také neúspešné spojenia, ktoré nie sú typické pre skenery portov. Sú to opakované TCP prenosy, pakety z útokov v iných sieťach alebo konfiguračné chyby. Pre tento účel sa použijú dva Bloom filtre [4]. V nasledujúcich kapitolách ukážem, že toto jednoduché riešenie dokáže redukovať počet TCP paketov podieľajúcich sa na nadviazaní spojenia až o 80% so zanedbateľným vplyvom na presnosť. Redukcia počtu paketov významne zníži počet prístupov do pamäti, pamäťové a výkonnostné požiadavky.

Po odfiltrovaní väčšiny sieťovej prevádzky je stále potrebné sledovať množstvo neúspešných pokusov o nadviazanie TCP spojenia pre zostávajúce zdroje. Hoci je v počítačovej sieti veľmi veľa aktívnych zdrojov, väčšina z nich neuspeje len v niekoľkých pokusoch o TCP spojenie, zatiaľ čo skenery portov neuspejú vo veľmi veľa pokusoch o TCP spojenie. Teda, problém detekcie skenerov portov môžeme previesť na dobre známy problém nájdenia najviac vyskytujúcich sa elementov v dátovom prúde [8]. Vzhľadom na efektívnosť detekcie skenovania portov som použil top-k dátovú štruktúru založenú na top-k štruktúre Stream-Summary popísanej v [9], ktorá má konštantnú pamäťovú náročnosť.

4.1 Detekčný algoritmus

Skenery portov su charakterizované jednoduchým správaním. Pokúšajú sa pripojiť k veľmi veľa cieľom, ale iba od niekoľko málo cieľov dostanú odpoveď. Tento nepomer medzi pokusmi a odpoveďami je základom pre mnoho deteknčných algoritmov. Algoritmus detekcie skenovania portov môže byť rozdelený na dva odlišné problémy: (1) detekcia neúspešných TCP spojení, (2) sledovanie zdrojov zodpovedných za neúspešné pokusy o TCP spojenia. Obidva problémy sú náročné vo vysokorýchlostných počítačových sieťach. Majú veľké požiadavky na pamäť a výpočtový výkon. Naivné riešenia založené na hashovacích tabuľkách sú v tomto prípade nepraktické, hoci môžu byť použité v malých počítačových sieťach.

Obrázek 4.1: Diagram detekčného algoritmu



V nasledujúcich podkapitolách budem prezentovať praktické riešenie, ktoré rieši tieto problémy redukovaním množstva spracovanej sieťovej prevádzky a redukovaním pamäťových nárokov detekčného algoritmu. V kapitole 4.1.1 popíšem jednoduchú metódu filtrovania nepotrebnej sieťovej prevádzky pomocou Bloom filtrov, ktoré výrazne zjednodušujú problém (1). V kapitole 4.1.2 sa sústredím na popis dátovej štruktúry, ktorá rieši problém (2).

Pre prehľadnosť označím klientskú stanicu, ktorá iniciuje spojenie ako A, s IP adresou A_{ip} a server, ktorý prijíma požiadavku na TCP spojenie ako B, s IP adresou B_{ip} a portom B_{port}.

4.1.1 Detekcia zlyhaných spojení

Neúspešné TCP spojenie môžeme definovať ako spojenie, pri ktorom klient nedostane odpoveď vo forme SYN+ACK paketu od servera hneď potom ako odošle korešpondujúci SYN paket. Preto pri detekcii neúspešných pokusov o TCP spojenie môžeme ignorovať ostatnú sieťovú prevádzku a sústrediť sa len na SYN a SYN+ACK pakety. V mnou použitých odchytených dátach pre vyhodnotenie algoritmu tvoria kontrolné pakety len 1,5% z celej TCP prevádzky.

Ďalej môžeme ignorovať legitímne pokusy o TCP spojenia. Aby sa mohli efektívne zahadzovať TCP spojenia smerujúce k fungujúcim službám v počítačovej sieti využijeme Bloom filter, ktorý uchováva zoznam bežiacich služieb v tvare dvojíc IP servera a číslo portu (bf_whitelist). Pre každú odpoveď SYN+ACK pridáme dvojicu $[B_{\text{ip}}, B_{\text{port}}]$ do tohoto Bloom filtra.

Vďaka tomu, že sa zaujímame iba o také zdroje, ktoré sa pokúšajú pripojiť k veľa unikátnym IP adresám a portom, môžeme zahodiť opakované pokusy o TCP spojenia k rovnakým destináciám. Opakované pokusy o TCP spojenie k rovnakej destinácii môžu zapríčiniť napríklad P2P body, zle nakonfigurované proxy, poštové servery alebo VPN aplikácie. Toto

správanie vyplýva zo štandardu popisujúceho TCP protokol. V kapitole 6.2 ukážem, že opakované pokusy o TCP spojenia k rovnakej destinácii sú v počítačových sieťach bežné. Pre efektívne zahadzovanie duplicitných SYN paketov do tej istej destinácie adresovanej IP adresou a číslom portu som použil Bloom filter (bf_syn). Pre každý detekovaný SYN paket uložíme trojicu $[A_{\text{ip}}, B_{\text{ip}}, B_{\text{port}}]$ do Bloom filtra. Ako ukážem neskôr, použitie tohto Bloom filtra má ešte ďalšie využitie. Zabraňuje saturácii bf_whitelist filtra veľkým množstvom SYN+ACK paketov (SYN+ACK pakety sú ignorované ak nie sú odpoveďou na prechádzajúci SYN paket).

Hoci Bloom filtre môžu produkovať falošné výskyty, majú zanedbateľný vplyv na presnosť použitej metódy, čo ukážem v kapitole 6.2. V prípade, že sú jeden alebo obidva filtre saturované (napríklad z dôvodu zlej dimenzácie), algoritmus môže označiť niektoré zdroje, ktoré sú skenermi portov za legitímne namiesto toho aby niektoré legitímne zdroje označil za skenery portov, čo je dôležitá vlastnosť systémov, ktoré automaticky blokujú skenery portov [15].

Schéma na obrázku 4.1 detailne prezentuje použitý algoritmus [10]. Po príchode paketu určím, či sa jedná o SYN alebo SYN+ACK paket. Inak je paket zahodený. V prípade, že sa jedná o SYN paket, skontrolujem, či dvojica $[B_{\text{ip}}, B_{\text{port}}]$ korešponduje s niektorou zo známych destinácii v Bloom filtri bf_whitelist. Ak áno, paket je zahodený. Ak nie, skontrolujem či ide o opakovaný pokus o TCP spojenie v Bloom filtri bf_syn. V tomto prípade je paket tiež zahodený. V opačnom prípade je trojica $[A_{\text{ip}}, B_{\text{ip}}, B_{\text{port}}]$ vložená do Bloom filtra bf_syn a adresa zdroja A_{ip} je inkrementovaná v dátovej štruktúre popísanej v kapitole 4.1.2. Pre SYN+ACK paket najskôr skontrolujem či v Bloom filtri bf_syn existuje záznam o korešpondujúcom SYN pakete. Inak je paket zahodený. Ďalej skontrolujem či dvojica $[B_{\text{ip}}, B_{\text{port}}]$ je v Bloom filtri bf_whitelist. Ak nie, dvojica $[B_{\text{ip}}, B_{\text{port}}]$ je uložená v Bloom filtri bf_whitelist a adresa zdroja A_{ip} je dekrementovaná. Bloom filter bf_whitelist plní dva účely. Slúži na sledovanie aktívnych destinácii a kontroluje či má byť zdroj dekrementovaný po nadviazaní úspešného TCP spojenia.

4.1.2 Identifikácia skenerov

Algoritmus popísaný v kapitole 4.1.1 produkuje sériu inkrementácii a dekrementácii pre nové TCP spojenia, respektíve pre úspešné TCP spojenia. Z tejto sekvencie potrebujeme identifikovať najviac aktívnych producentov zlyhaných TCP spojení, ktoré s vysokou pravdepodobnosťou korešpondujú k skenerom portov. Tento problém je dobre známym problémom identifikácie najviac vyskytujúcich sa elementov v dátovom prúde.

Pre tento účel potrebujeme dátovú štruktúru s malou pamäťovou náročnosťou a podporou inkrementácie ako aj dekrementácie. Použil som štruktúru nazvanú Span-Dec od autorov použitej detekčnej metódy [10]. Je to modifikácia dátovej štruktúry Stream-Summary [9]. Originálna Stream-Summary štruktúra nepodporuje dekrementáciu. Rozšírenie Span-Dec podporuje limitované množnstvo dekrementácii. V kontexte detekcie skenovania portov, má dátová štruktúra vlastnosti takmer ako hash tabuľka s výrazne nižšími pamäťovými nárokmi. V nasledujúcich podkapitolách v krátkosti popíšem obidve spomínané štruktúry.

Stream-Summary

Táto štruktúra hľadá najfrekventovanejšie elementy v dátovom prúde. Je schopná pokryť $elem_{\text{max}}$ nezávislých elementov naraz. Každý element e_{i} má priradené počítadlo cnt_{i}. Všetky počítadla s rovnakou hodnotou sú spojené do toho istého zhluku. Zhluky sú spojené do jedného celku a môžu byť dynamicky vytvárané a rušené. Keď je element e_{i} inkrementovaný,

je vyňatý z jeho zhluku a vložený do susedného zhluku s novou hodnotou. Ak je dosiahnutý maximálny počet sledovaných elementov ($elem_{\max}$), nový element vylúči element s najmenším počítadlom. Každý element má maximálnu odchýľku ε_{i}, ktoré závisí na hodnote vylúčeného elementu. Frekvencia elementu je vypočítaná ako $freq(e_{\mathrm{i}}) = cnt_{\mathrm{i}} - \varepsilon_{\mathrm{i}}$.

Span-Dec

V tejto podkapitole popíšem rozdiely medzi originálnou dátovou štruktúrou Stream-Summary a jej Span-Dec modifikáciou. Originálna Stream-Summary štruktúra [9] pozostáva zo zhlukov usporiadaných do zoznamu. Každý zhluk pozostáva z počítadiel s rovnakou hodnotou. Element e_{i} má priradené počítadlo cnt_{i} a chybu ε_{i}. Ak na vstup príde už obsiahnutý element, cnt_{i} je vyňaté z jeho zhluku a presunuté do susedného zhluku s hodnotou zvýšenou o 1. Prázdne zhluky sú uvolnené z pamäti a nové zhluky sú vytvárané na požiadanie. Ak počet preddefinovaných nezávislých elementov dosiahne $elem_{\max}$, nové elementy vyradia elementy v najnižšom zhluku(ak e_{k} je vyradený element a e_{i} je nový element, $cnt(e_{\mathrm{i}}) = cnt(e_{\mathrm{k}}) + 1$ a chyba $\varepsilon_{\mathrm{i}} = cnt(e_{\mathrm{k}})$).

Predstavme si situáciu kde legitímny zdroj pošle 3 SYN pakety do odlišných destinácii a neskôr dostane 3 SYN+ACK pakety. Prosté dekrementovanie počítadla by mohlo viesť k situácii kde $cnt(e_{\mathrm{i}}) < \varepsilon_{\mathrm{i}}$ a frekvencia výskytu by mohla byť menšia ako 0. Preto v Span-Dec modifikácii sú použité dve počítadla pre jeden element. Nižšie počítadlo cnt_{L} a vyššie počítadlo cnt_{H}. V Span-Dec modifikácii je definované $span(e_{\mathrm{i}}) = cnt_{\mathrm{H}} - cnt_{\mathrm{L}}$. Keď je element e_{i} inkrementovaný, $cnt_{\mathrm{H}}(e_{\mathrm{i}})$ je inkrementovaný ako v originálnej Stream-Summary štruktúre. Keď je element e_{i} dekrementovaný, $cnt_{\mathrm{H}}(e_{\mathrm{i}})$ je dekrementovaný, ale nikdy nie pod hranicu $cnt_{\mathrm{L}}(e_{\mathrm{i}})$. Ak je výsledok inkrementácie $span(e_{\mathrm{i}}) > span_{\max}$ (kde $span_{\max}$ je preddefinovaný parameter), inkrementuje sa aj $cnt_{\mathrm{L}}(e_{\mathrm{i}})$. Hodnota elementu e_{i} sa rovná $freq(e_{\mathrm{i}}) = cnt_{\mathrm{H}}(e_{\mathrm{i}}) - \varepsilon(e_{\mathrm{i}})$.

V prípade, že počet nezávislých elementov v štruktúre dosiahne $elem_{\max}$, nový element vyradí element z najnižšieho zhluku. Vyradený element označíme ako e_{k}. Nový element bude mať $cnt_{\mathrm{L}}(e_{\mathrm{i}}) = cnt_{\mathrm{H}}(e_{\mathrm{i}}) = cnt_{\mathrm{L}}(e_{\mathrm{k}})$. Touto cestou nikdy nedosiahneme situáciu kedy $cnt_{\mathrm{L}}(e_{\mathrm{i}}) < \varepsilon_{\mathrm{i}}$ pretože $cnt_{\mathrm{L}}(e_{\mathrm{i}})$ je spodný limit, ktorý môže $cnt_{\mathrm{H}}(e_{\mathrm{i}})$ dosiahnuť.

Príliš malá hodnota $span_{\max}$ redukuje schopnosť algoritmu dekrementovať (Ak je $span_{\max} = 3$ a príde 6 SYN paketov a neskôr príde 6 SYN+ACK paketov, $cnt_{\mathrm{H}}(e_{\mathrm{i}})$ bude inkrementovaný 6 krát a dekrementovaný bude len 3 krát, pretože po príchode štvrtého SYN paketu musí byť inkrementovaný $cnt_{\mathrm{L}}(e_{\mathrm{i}})$ pre zachovanie $span(e_{\mathrm{i}}) < span_{\max}$)). Na druhej strane, ak bude $span_{\max}$ príliš veľké, ε bude rásť rýchlo: cnt_{H} bude rásť rýchlo a cnt_{L} bude malé. To má za následok to, že elementy môžu byť ľahko vyradené a nové elementy môžu mať vysoké ε pretože cnt_{H} je vysoké. Ak $span_{\max} = 0$, Span-Dec a Stream-Summary majú rovnaké vlastnosti.

Algoritmus 1 Inkrementácia elementu e

---

ak e je v S **potom**
  $counter_{\mathrm{H}}(e)++$
  ak $counter_{\mathrm{H}}(e) - counter_{\mathrm{L}}(e) > span_{\max}$ **potom**
    $counter_{\mathrm{L}}(e)++$
  koniec
inak e nie je v S
  ak je miesto pre nové elementy **potom**

vlož nový element s $counter_{\mathrm{L}} = counter_{\mathrm{H}} = 1$ a $\varepsilon = 0$
inak nie je voľné miesto
$\varepsilon_{\mathrm{new}} = counter_{\mathrm{H}}(e_{\mathrm{min}})$
vymaž e_{min}
vlož nový e s $counter_{\mathrm{L}} = counter_{\mathrm{H}} = \varepsilon_{\mathrm{new}} + 1 a \varepsilon = \varepsilon_{\mathrm{new}}$
koniec
koniec

Algoritmus 2 Inkrementácia (dekrementácia) počítadla $counter_{\mathrm{i}}$

---

vyber $counter_{\mathrm{i}}$ z $bucket_{\mathrm{i}}$
ak existuje $bucket_{\mathrm{i+1}}$ ($bucket_{\mathrm{i-1}}$) **potom**
vlož $counter_{\mathrm{i}}$ do $bucket_{\mathrm{i+1}}$ ($bucket_{\mathrm{i-1}}$)
inak
vytvor nový bucket a vlož ho za (pred) $bucket_{\mathrm{i}}$
vlož $counter_{\mathrm{i}}$ do vytvoreného bucketu
koniec
ak $bucket_{\mathrm{i}}$ je prázdny **potom**
vymaž $bucket_{\mathrm{i}}$
koniec

Algoritmus 3 Dekrementácia elementu e

---

ak e je v S **potom**
ak $counter_{\mathrm{H}}(e) > counter_{\mathrm{L}}(e)$ **potom**
$counter_{\mathrm{H}}(e)$–
inak
nerob nič
koniec
inak
nerob nič
koniec

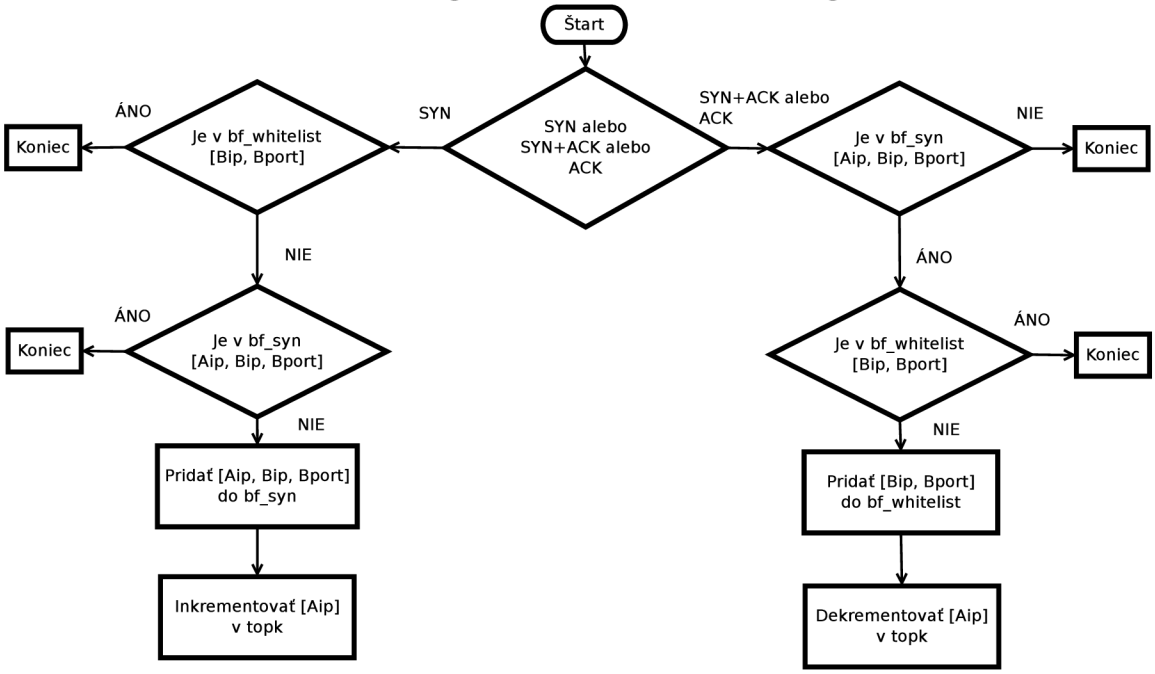
Algoritmus 1,2 a 3 ukazujú operácie nad počítadlami a zhlukmi. Použité symboly: $bucket_{\mathrm{i}}$ je zhluk počítadiel cnt_{i}, $bucket_{\mathrm{i+1}}$ je sused zhluku $bucket_{\mathrm{i}}$ s hodnotu o 1 vyššou ($bucket_{\mathrm{i-1}}$ analogicky), S je množina všetkých elementov vo všetkých zhlukoch, e_{min} je element s najmenšou hodnotou cnt_{L} v celom S.

4.2 Rozšírenie detekčného algoritmu

V tejto kapitole popíšem mnou navrhnuté rozšírenie algoritmu detekcie skenovania portov, ktoré pre sledovanie nadviazania úspešnej TCP komunikácie používa aj ACK pakety.

Pri testovaní presnosti v kapitole 6.2.2 som zistil, že pri analýze zachytených dát zo siete CESNET algoritmus generuje pomerne veľa falošných hlásení. Je to spôsobené tým, že u niektorých zdrojov bol zachytený iba jeden smer komunikácie. V prípadoch, ktoré

Obrázek 4.2: Diagram rozšírenia detekčného algoritmu



spôsobili falošné hlásenia to bol smer od A do B, teda SYN pakety. SYN+ACK pakety, ktoré reprezentujú opačný smer komunikácie boli pravdepodobne smerované inou trasou. To znamená, že základný algoritmus predpokladá, že bude mať k dispozícii obidva smery komunikácie a ak nie, produkuje veľa falošných hlásení. Schéma na obrázku 4.2 detailne prezentuje rozšírenie algoritmu detekcie skenovania portov vo vysokorýchlostných sieťach. Rozšírenie oproti základnému algoritmu sleduje ešte pakety smerujúce od A do B s nastaveným ACK príznakom. Je dôležité poznamenať, že SYN a ACK pakety, ktoré spracováva algoritmus sú odosielané v smere z A do B na rozdiel od SYN+ACK paketov, ktoré sú odosielané v smere z B do A. To znamená, že zdrojová a cieľová IP adresa a port sú na A a B mapované na základe príznakov SYN a ACK. Ak príde na vstup ACK paket, znamená to, že spojenie bolo pravdepodobne úspešne nadviazané a prebieha komunikácia. Cieľový port je teda otvorený a algoritmus sa môže zachovať ako pri zachytení SYN+ACK paketu.

Po príchode ACK paketu na vstup najskôr skontrolujem či v Bloom filtri bf_syn existuje záznam o korešpondujúcom SYN pakete. Inak je paket zahodený. Ďalej skontrolujem či dvojica [B_{ip}, B_{port}] je v Bloom filtri bf_whitelist. Ak nie, dvojica [B_{ip}, B_{port}] je uložená v Bloom filtri bf_whitelist a adresa zdroja A_{ip} je dekrementovaná. Bloom filter bf_whitelist tak isto ako aj v základnej verzii algoritmu slúži na sledovanie aktívnych destinácii a kontroluje či má byť zdroj dekrementovaný po nadviazaní úspešného TCP spojenia. Jediným rozdielom oproti základnej verzii je, že algoritmus na sledovanie aktívnych destinácii využíva aj ACK pakety v smere od A do B. Vo vyhodnotení ukážem, že po tejto meodifikácii algoritmus už viac nie je závislý na komunikácii v smere od B do A. No je to za cenu toho, že algoritmus musí spracovávať omnoho väčšie množstvo paketov.

Kapitola 5

Implementácia

V tejto kapitole popíšem samotnú implementáciu algoritmu popísanom v [10]. Program je tvorený niekoľkými triedami. Triedou sniffer implementovanou v súbore sniffer.cpp, triedou bloom implementovanou v súbore bloom.cpp a triedou span_dec implementovanou v súbore span_dec.cpp. Samotný algoritmus implementujem v samostatnej funkcii process() v súbore main.cpp. V nasledujúcich podkapitolách popíšem jednotlivé triedy.

5.1 Trieda sniffer

Trieda sniffer zabezpečuje čítanie a analýzu paketov zo sieťového rozhrania alebo zo súboru podľa nastaveného filta. V tomto prípade je filter nastavený tak aby prepúšťal SYN, SYN+ACK prípadne ACK pakety. Na čítanie paketov využíva funkcie z knižnice libpcap.

Pri implementácii analýzy paketov som kládol dôraz na efektivitu a rýchlosť, preto som navrhol dátové štruktúry pre ethernet hlavičku, ip hlavičku a tcp hlavičku, ktoré sa dajú namapovať na jednotlivé úseky paketu uloženého v pamäti počítača. Mapovaním jednotlivých dátových štruktúr na úseky paketu uloženého v pamäti som sa vyhol náročným operáciam alokácie nových premenných a kopírovania úsekov pamäti. Takto pracujem iba s úsekom pamäti, ktorú alokovala funkcia z knižnice libpcap. Paket uložený funkciou z kižnice libpcap predstavuje retazec znakov typu unsigned char.

5.2 Trieda bloom

Trieda bloom implementuje funkcionalitu Bloom filtra. Ide o veľmi jednoduchú implementáciu, ktorá na vyhľadanie prvku vo filtri používa metódu look_up() a pre pridanie prvku do filtra metódu add_data(). Najdôležitejšou časťou implementácie triedy bloom je výpočet niekoľkých hashovacích funkcii, ktorých hodnoty sú používané metódami look_up() a add_data() ako indexy do bitového vektoru. V mojej implementácii počítam 7 rôznych hodnôt. V skutočnosti na výpočet týchto 7 hodnôt nepoužívam 7 hashovacích funkcii. Z dôvodu rýchlosti bloom filtra som zvolil iba dve hashovacie funkcie, konkrétne Super Fast Hash a Murmur Hash 2. Pre prehľadnosť označím Super Fast Hash ako h1 a Murmur Hash ako h2. Ostatných 5 hodnôt dopočítavam v cykle s počítadlom opakovaní i = 2 kde vyhodnocujem výraz:

```
hashes[i] = h1 + i*h2;
```

Takýmto spôsobom je teoreticky možné dopočítať ľubovoľný počet hodnôt, ktoré nahradia vypočítané hodnoty z rôznych hashovacích funkcii. Ďalším spôsobom ako zjednodušiť a teda aj zrýchliť výpočet hodnôt je využitie iba jednej hashovacej funkcie a funkcie random() na dopočítanie ostatných hodnôt a to tak, že vždy pre jeden záznam vypočítame iba jednu hashovaciu funkciu a vypočítanú hodnotu použijeme ako parameter funkcie seed(). Tým, že funkcia random() generuje iba pseudonáhodné čísla, pri tej istej hodnote parametru funkcie seed() generuje vždy tú istú postupnosť čísel. Hoci je tento spôsob efektívny, produkuje viac kolízii ako spôsob, ktorý som použil v mojej implementácii.

5.3 Trieda span_dec

Trieda span_dec reprezentuje top-k štruktúru popísanú v kapitole 4.1.2 . Okrem metód, ktoré nastavujú parametre štruktúry (maximálny počet elementov, $span_{\max}$, prah detekcie a pod.), trieda span_dec implementuje metódy pre inkrementáciu, dekrementáciu a vyhľadanie prvku v štruktúre. Tým, že táto štruktúra podporuje aj dekrementácie sa proces inkrementácie a dekrementácie elementu komplikuje. Pseudokód inkrementácie a dekrementácie elementu v štruktúre som popísal v kapitole 4.1.2.

5.3.1 Použité dátové štruktúry

Každý element v top-k štruktúre musí obsahovať svoju identifikáciu a počítadlo. V prípade použitej span-dec štruktúry musí každý element naviac obsahovať ďalšie počítadlo a údaj o veľkosti chyby počítadla. Pre element som navrhol nasledujúcu dátovú štruktúru:

```
struct element {
        struct in_addr Aip; /* IP adresa zdroja */
        int cnt_L;          /* nižšie počítadlo */
        int cnt_H;          /* vyššie počítadlo */
        int overestimation; /* chyba */
};
```

Elementy sa združujú do zhlukov. Elementy s rovnakou hodnotou počítadla tvoria jeden zhluk. Zhluk je reprezentovaný nasledovnou dátovou štruktúrou:

```
struct bucket {
        std::vector<struct element> elements;
        int elem_id;
};
```

Dátová štruktúra je tvorená vektorom elementov a premennou elem_id, ktorá identifikuje zhluk hodnotou, ktorú majú počítadlá združených elementov. Hoci sa táto hodnota dá dodatočne zistit z hodnoty počítadla elementu, premennú elem_id som zaviedol z implementačných dôvodov.

Top-k štruktúra je tvorená vyššie popísanými zhlukmi, ktoré sú organizované do zoznamu. Zoznam zhlukov som implementoval pomocou triedy list zo štandardnej knižnice C++. Táto trieda reprezentuje obojsmerný zoznam a operácie nad ním. Uloženie v obojsmernom zozname je výhodné pre operácie vkladania nových zhlukov a rušenia prázdnych zhlukov.

5.3.2 Inkrementácia a dekrementácia počítadla elementu

Algoritmus inkrementácie a dekrementácie elementu je popísaný pomocou pseudokódu v kapitole 4.1.2. Algoritmus začína vyhľadaním elementu v top-k štruktúre. Ak je element nájdený, uloží sa pozícia zhluku v rámci top-k štruktúry a pozícia elementu v rámci zhluku prostredníctvom iterátorov. S informáciou o presnej pozícii zhluku a elementu v top-k štruktúre sú operácie presunu elementu do nasledujúceho alebo predchádzajúceho zhluku, vytvorenie alebo zrušenie zhluku pomerne jednoduché.

5.4 Výstup programu

Zdroje označené ako skenery portov sú odosielané cez špeciálne rozhranie poskytované knižnicou TRAP do systému pre sieťovú analýzu Nemea [3]. Dáta sa odosielajú vo formáte "SRC_IP,EVENT_SCALE,TIMESLOT". SRC_IP reprezetuje IP adresu zdroja, ktorý bol označený ako skener portov, EVENT_SCALE je počítadlo neúspešných TCP spojení, ktoré sa pokúsil nadviazať daný zdroj. TIMESLOT je údaj o časovom intervale, v ktorom bol zdroj detekovaný.

Kapitola 6

Výsledky

V tejto kapitole vyhodnotím rýchlostné a pamäťové požiadavky použitého algoritmu pre detekciu skenovania portov použitím zachytenej sieťovej prevádzky. (možno aj live)

6.1 Dáta pre vyhodnotenie

Vo vyhodnotení použijem 6 vzoriek dát. Vzorka A0 je upravenou verziou vzorky, ktorá bola zachytená z 1GigE prístupovej cesty UPC, ktorá pripája približne 50000 užívateľov. A0 je vzorka, ktorú autori algoritmu použili k testovaniu a zverejnili ju. Vzorka A0 bude podrobne popísaná neskôr. Vzorka B je prevzatá od MAWI Working Group Traffic Archive [16]. Vzorka C bola zachytená v sieti CESNET a obsahuje iba tie pakety, ktoré majú aktívny SYN príznak. Ako ukážem v kapitole 6.2.2, v tejto sieti pre detekciu skenovania portov pakety so SYN príznakom nepostačujú. Trasy D až F sú tiež zachytené zo siete CESNET ale obsahujú všetky TCP pakety.

Pre vyhodnotenie je potrebná trasa, na ktorej sa overí či sú detekované skenery portov skutočné skenery portov alebo legitýmne zdroje. Pre tento účel autori algoritmu upravili trasu A odstránením všetkých reálnych skenerov portov. Trasu skenovali pomocou Bro [12] s jeho štandardným algoritmom (s prahom 25) a s TRW algoritmom (východzie nastavenia). Napriek veľkému počtu detekovaných skenerov boli odstránené všetky toky z nahlásených IP adries aj za cenu odstránenia niektorých legitímnych zdrojov. Týmto postupom sa dosiahlo úplne odstránenie prevádzky vytvorenej skenermi portov. Ďalej, podľa metodológie navrhnutej v [11] do trasy vložili umelé skeny: 1000 skenerov s úspešnosťou na nadviazanie spojenia 0.2 a 1000 bežných zdrojov s úspešnosťou 0.8. Interval medzi syn a syn+ack paketmi bol navrhnutý pomcou rovnomerného rozloženia s rozsahom 0.450 ms. Odstránením reálnych a pridaním umelo vytvorených skenerov portov vznikla vzorka A0.

6.2 Vyhodnotenie

Táto kapitola pokrýva vyhodnotenie použitého algoritmu. Najskôr budem prezentovať príklad nastavenia vhodných rozmerov a parametrov. Ďalej vyhodnotím presnosť a vplyv parametrov algoritmu na presnosť a v závere vyhodnotím rýchlosť algoritmu.

6.2.1 Príklad nastavenia vhodných parametrov algoritmu

Ako je popísane v kapitole 4.1, algoritmus závisí na niekoľkých parametroch. V nasledujúcom texte navrhneme rozmery obidvoch Bloom filtrov: bf_syn a bf_whitelist a topk štruk-

túry. Algoritmus pracuje v intervaloch. Dĺžku jedného intervalu som nastavil na hodnotu 120 sekúnd. Nasledujúci príklad je vytvorený pre trasu A0.

bf_whitelist

Ako je popísané v kapitole 4.1.1, tento filter uchováva informácie o aktívnych destináciách. Jeho veľkosť záleží od počtu dvojíc $[B_{\text{ip}}, B_{\text{port}}]$. Pre trasu A0 je priemerný počet destinácii na interval $n_{\text{avg}} = 6592$.

Pre pokrytie veľkého počtu dvojíc $[B_{\text{ip}}, B_{\text{port}}]$ a pre pokrytie výkyvov v sieťovej prevádzke zvolíme rozmer s dostatočnou rezervou, $n = 3 * n_{\text{avg}} = 15153$. S použitím 7 hash funkcii dostávame optimálnu veľkosť Bloom filtra $m = 138432$. Z implementačných dôvodov som rozmer upravil na mocnicnu so základom 2, teda $m = 2^{18} = 262144b = 32kB$.

bf_syn

Tento filter je zodpovedný za sledovanie spojení. Použil som tú istú procedúru ako na filter bf_whitelist vzhľadom na trojice $[A_{\text{ip}}, B_{\text{ip}}, B_{\text{port}}]$. Pre trasu A0 je je priemerný počet trojíc $n = 15761$. Pre každú trasu je pomer medzi počtom trojíc $[A_{\text{ip}}, B_{\text{ip}}, B_{\text{port}}]$ a dvojíc $[B_{\text{ip}}, B_{\text{port}}]$ 1.53 až 3.12. Takže pre jednoduchosť bude bf_syn dvakrát väčší než bf_whitelist.

topk

Počíta najviac vyskytujúce sa elementy s konštantnými pamäťovými nárokmi. Pre použitú top-k štruktúru som zvolil počet elementov a nastavil hĺbku dekrementácie ($span_{\text{max}}$). Vybral som náhodný ale dostatočne veľký počet elementov $elem_{\text{max}} = 10000$ (V mojej implementácii v najhoršom prípade zaberá približne 1500 kB). Hodnota $span_{\text{max}}$ závisí od počtu syn paketov odoslaných zdrojom medzi syn_{i} a odpoveďou $syn + ack_{\text{i}}$. Pre trasu A0 sa táto hodnota pohybuje na úrovni 5 paketov, preto je hodnota $span_{\text{max}} = 5$.

Ten istý postup je aplikovaný aj na ostatné trasy. Zhrnutie parametrov algoritmu pre jednotlivé trasy ukazuje tabuľka 6.2.1.

Tabuľka 2: Parametre pre vyhodnotené trasy.

	trasa A0	trasa B	trasa C	trasa D	trasa E	trasa F
bf_whitelist veľkosť	32 kB	64 kB	128 kB	64 kB	64 kB	128 kB
bf_syn veľkosť	64 kB	128 kB	256 kB	128 kB	128 kB	256 kB
span_max	5	6	7	7	7	7

6.2.2 Presnosť

Pre vyhodnotenie presnosti som použil niekoľko trás. Najskôr trasu A0 s umelo vloženými skenermi portov a legitímnymi zdrojmi. Výsledky mnou implementovaného programu som porovnal s dostupným zoznamom skenerov a legitímnych zdrojov. Ak som použil parametre algoritmu tak ako som ich vypočítal v predchádzajúcej sekcie, algoritmus pre prah detekcie vyšší ako 20 detekoval všetky skenery a nevykazoval žiadne falošné hlásenia.

Trasu B som spracoval programom HostStats [2] a pomocou skriptu na detekciu skenovania portov z frameworku NADEX [1] a výsledky som porovnal s výsledkami z mnou

implenetovaného programu. Pri tomto teste bol prah detekcie nastavený tak isto ako pri vyhodnocovaní ostatnými dvoma nástrojmi na hodnotu 50 a parametre algoritmu tak ako v tabuľke 6.2.1. Parameter $elem_{\max}$ okrem testov, pri ktorých bol skúmaný vplyv tohto parametru na presnosť bol nastavený na hodnotu 10000. Celkovo mnou implementovaný program detekoval všetko to čo dva ostatné programy a okrem toho ďalších 5 prípadov skenovania portov. Manuálnou analýzou sporných prípadov sa ukázalo, že 3 prípady boli pravými skenermi portov a ostatné dva boli falošné hlásenia.

Trasu C som tak isto ako trasu B spracoval pomocou HostStats a NADEX. Parametre algoritmu a prah detekcie bol nastavený rovnakým spôsobom ako pri trase B. Mnou implementované riešenie v tomto prípade označilo za skenery portov výrazne viac zdrojov oproti ostatným dvom programom. Skúmaním prevádzky produkovanej náhodne vybranými zdrojmi z množiny zdrojov, ktoré ako skenery portov označil len môj program som zistil, že sa jedná o falošné hlásenia. Tieto falošné hlásenia mali jednu spoločnú vlastnosť. V dátach na trase C pre tieto zdroje chýbala komunikácia v smere z B do A. V tomto smere sú odosielané aj SYN+ACK pakety, ktoré su pre mnou implementovaný algortimus veľmi dôležité. Na základe tejto skutočnosti som navrhol rozšírenie, ktoré je založené na sledovaní aktívnych destinácii aj pomocou paketov s nastaveným ACK príznakom. Toto rozšírenie je bližšie popísané v kapitole 4.2.

Kedže trasa C obsahuje len pakety s nastaveným SYN príznakom, nemožno na nej otestovať rozšíenie algoritmu popísané v kapitole 4.2. Pre overenie funkčnosti rozšírenia algoritmu som použil trasu B, v ktorej som odstránil SYN+ACK pakety patriace do komunikácie niektorých najviac aktívnych legitímnych zdrojov, ktoré pôvodne neboli označené ako skenery portov. Pri vypnutom rozšírení algoritmus za skenery portov označil tie isté zdroje ako pri pôvodnej trase B a navyšše za skenery portov označil niektoré zo zdrojov, z ktorých komunikácie boli odobrané SYN+ACK pakety. Pri zapnutom rozšírení algoritmus označil za skenery portov len tie zdroje, ktoré boli označené aj pri pôvodnej neupravenej trase B teda žiadne zdroje, z ktorých komunikácie boli odobrané SYN+ACK pakety. V ďalšom testovaní boli odstránené z trasy všetky SYN+ACK pakety. S vypnutým rozšírením algoritmus produkoval veľké množstvo falošných hlásení, naopak pri zapnutom rozšírení boli výsledky totožné s výsledkami z pôvodnej trasy B. Tieto výsledky potvrdzujú to, že rozšírenie funguje presne podľa očakávaní.

Trasy D, E a F boli odchytené za účelom testovania rozšírenia algoritmu. Tieto trasy obsahujú všetku TCP komunikáciu, teda aj pakety s nastaveným ACK príznakom, ktoré sú potrebné pre správnu funkčnosť rozšírenia. Tak isto ako pri upravenej trase B, algoritmus s vypnutým rozšírením generoval veľké množstvo falošných hlásení. So zapnutým rozšírením algoritmus na všetkých troch trasách produkoval približne o 80% menej falošných hlasení ako algoritmus s vypnutým rozšírením.

6.2.3 Vplyv konfiguračných parametrov na presnosť

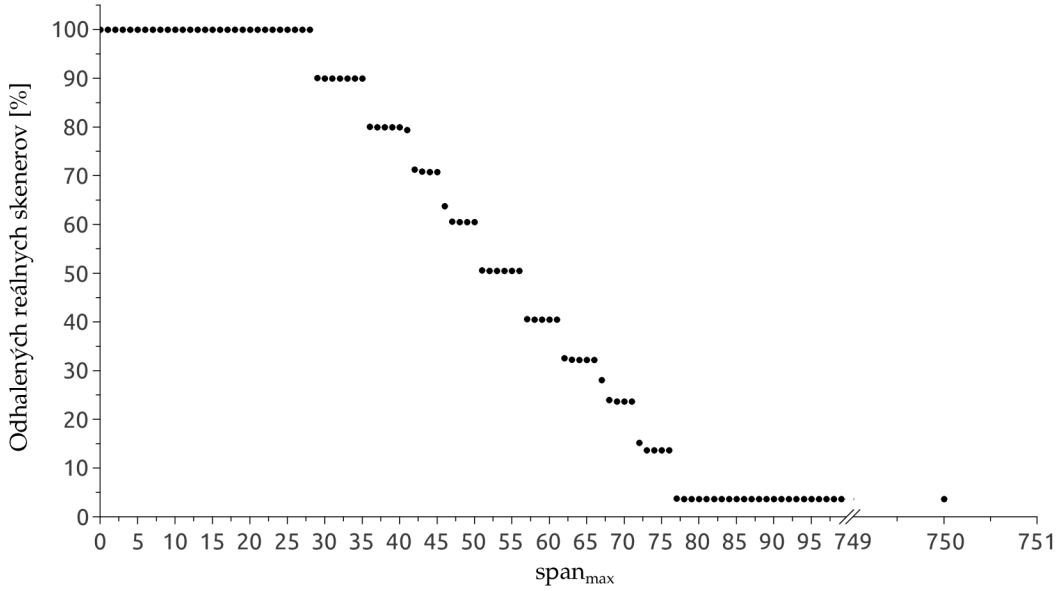
Na nasledujúcich grafoch ukážem vplyv konfiguračných parametrov algoritmu na presnosť. Vyhodnocovaná trasa A0 bola spracovaná v rámci jedného časového intervalu. Hodnota $elem_{\max}$ bola nastavená na 6000. Bloom filtre bf_syn a bf_whitelist boli nastavené na veľmi veľké hodnoty tak aby neovplyvňovali výsledok.

Parameter $\mathbf{span_{max}}$

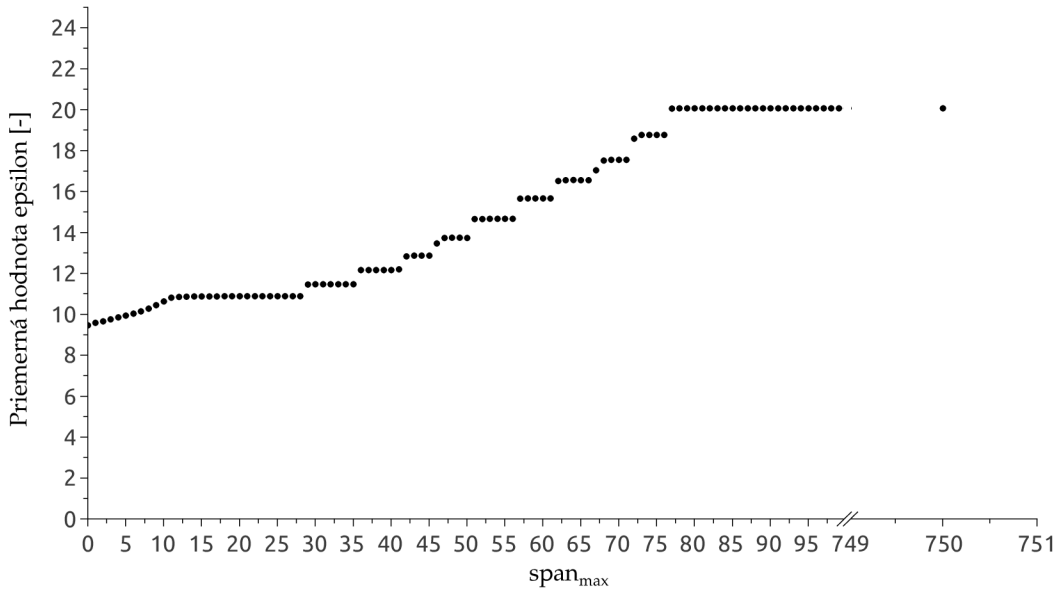
Graf na obrázku 6.1 ukazuje vplyv parametru $span_{\max}$ na presnosť. Ak je hodnota $span_{\max}$ veľmi veľká, presnosť klesá. To sa deje z dôvodu narastajúcej hodnoty ε_{i} (Graf na obrázku

6.2).

Obrázek 6.1: Vplyv parametru $span_{\max}$ na úspešnosť odhalenia reálnych skenerov v dátach z trasy A0.



Obrázek 6.2: Vplyv parametru $span_{\max}$ na hodnotu ε (trasa A0).

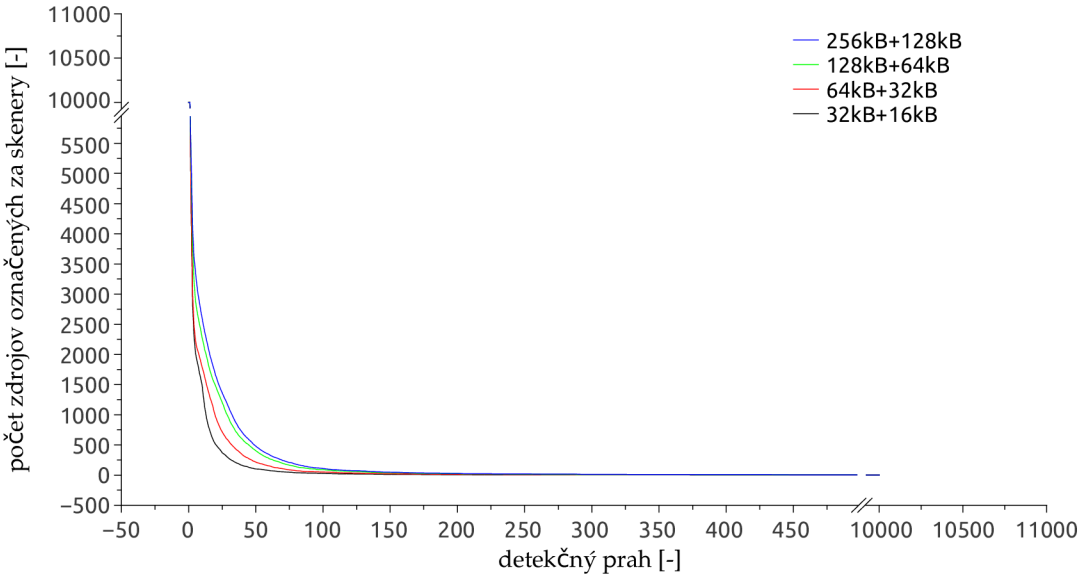


Veľkosť pamäti

V tejto sekcii vyhodnotím vplyv veľkosti pamäti na presnosť detekčného algoritmu s využitím trasy C. Najskôr ukážem vplyv veľkosti Bloom filtrov s 10000 položkami v topk štruktúre. Výsledky sú zobrazené v grafe na obrázku 6.3. Detekčný algoritmus s Bloom filtrami s veľkosťou pod 192 kB (128 kB + 64 kB) vykazuje neodhalené hlásenia z dôvodu kolízii popísaných v kapitole 4.1.1. Detekčný algoritmus s Bloom filtrami s veľkosťou nad 192 kB a detekčným prahom nad 50 vykazuje výsledky veľmi blízke výsledkom s použitím veľkostí Bloom filtrov vypočítaných v kapitole 6.2.1. Z grafu je vidieť, že počet zdrojov označených za skenery sa od veľkosti Bloom filtrov 192 kB už takmer nemení.

V grafe na obrázku 6.4 je zobrazený vplyv maximálneho počtu elementov ($elem_{\max}$) v topk štruktúre. Veľkosti Bloom filtrov boli nastavené na hodnoty vypočítané v kapitole 6.2.1. Detekčný algoritmus s detekčným prahom nad 100 a maximálnym počtom elementov v topk štruktúre 2500 dosahuje výsledky veľmi blízke výsledkom algoritmu s nastaveným oveľa väčším maximálnym počtom elementov v topk štruktúre. V grafe je vidieť, že počet zdrojov označených ako skenery sa od maximálneho počtu elementov 2500 už takmer nemení.

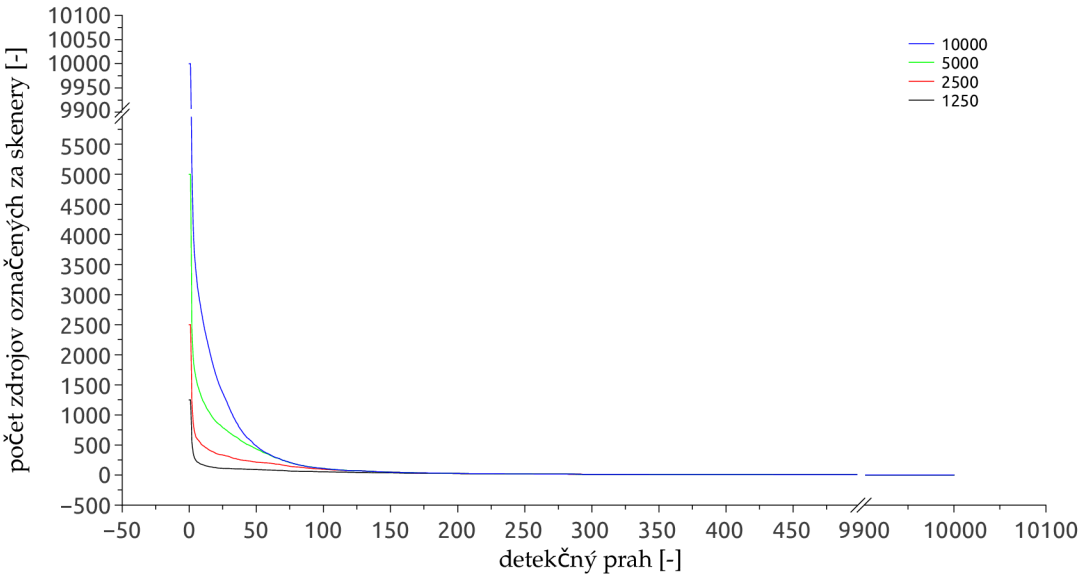
Obrázek 6.3: Vplyv veľkosti pamäti na presnosť (filtre bf_syn + bf_whitelist). (trasa C).



6.2.4 Využitie filtrov

Tabuľka 6.2.4 prezentuje využitie Bloom filtrov v základnej verzii algoritmu. Riadky využitie: bf_syn a využitie: bf_whitelist ukazujú maximálne využitie Bloom filtrov. Riadky vylúčenia: bf_syn a vylúčenia: bf_whitelist predstavujú percentuálny podiel paketov zahodených Bloom filtrami k počtu všetkých paketov, ktoré boli spracované algoritmom. Riadok vylúčenia spolu ukazuje celkový percentuálny podiel zahodených paketov oboma filtrami. Oba filtre zahodili približne 75 až 85% zo všetkých SYN a SYN+ACK paketov. Len 15 až 25% SYN a SYN+ACK paketov viedlo k inkrementácii alebo dekrementácii v topk štruktúre. To znamená, že chyba počítadla v topk štruktúre priamo závisí na počte paketov prepustených oboma Bloom filtrami. Z tabuľky možno vyčítať, že pri vyhodnotení trás C,

Obrázek 6.4: Vplyv veľkosti pamäti na presnosť (počet elementov v top-k štruktúre). (trasa C).



Tabulka 6.1: Využitie filtrov (základný algoritmus)

	trasa A0	trasa B	trasa C	trasa D	trasa E	trasa F
dátum	18.5.2010	1.4.2009	4.3.2014	16.4.2014	16.4.2014	17.4.2014
využitie: bf_syn	16,71%	11,61%	35,18%	44,58%	44,84%	17,20%
využitie: bf_whitelist	13,34%	0,52%	0%	0,05%	0,05%	0%
vylúčenia: bf_syn	38,56%	54,28%	75,85%	86,21%	86,94%	81,74%
vylúčenia: bf_whitelist	41,47%	20,05%	0%	0,02%	0,02%	0%
vylúčenia spolu	80,3%	74,33%	75,85%	86,23%	86,96%	81,74%

D, E a F nebol vôbec využívaný Bloom filter bf_whitelist. Z toho vyplýva, že zachytené trasy neobsahovali bud žiadne alebo zanedbatelné množstvo SYN+ACK paketov. Vďaka absencii SYN+ACK paketov nedochádza k dekrementáciám počítadiel v topk štruktúre po úspešnom nadviazaní komunikácie a tak dochádza k produkovaniu veľkého množstva falošných hlásení. Ako falošné skenery portov sú v tomto prípade najčastejšie označované zdroje, ktoré nadväzujú veľké množstvo úspešných TCP spojení.

Tabuľka 6.2.4 prezentuje využitie Bloom filtrov v rozšírenej verzii algoritmu. V tomto prípade obidva filtre zahodili podstatne viac paketov ako v základnej verzii algoritmu. Približne 99% všetkých SYN, SYN+ACK alebo ACK paketov. Tento nárast je spôsobený tým, že pre sledovanie úspešného nadviazania TCP spojení sa používajú aj ACK pakety z A do B. Najviac je tento rozdiel vidiet na trasách D,E a F, ktoré neobsahujú takmer žiaden SYN+ACK pakety a teda v základnej verzii algoritmu nie je Bloom filter bf_whitelist takmer vôbec využívaný. V rozšírenej verzii algoritmu kde sa na sledovanie aktívnych destinácii využívajú aj ACK pakety je Bloom filter bf_whitelist už využívaný. Hoci je nárast počtu spracúvaných paketov v rozšírenej verzii algortimu takmer 50%, čas potrebný na spracovanie trasy je len o 20% väčší. Je to spôsobené tym, že výrazná väčšina ACK paketov je zahodená

Tabulka 6.2: Využitie filtrov (rozšírený algoritmus)

	trasa A0	trasa B	trasa C	trasa D	trasa E	trasa F
dátum	18.5.2010	1.4.2009	4.3.2014	16.4.2014	16.4.2014	17.4.2014
využitie: bf_syn	-	11,13%	-	16,09%	16,27%	9,52%
využitie: bf_whitelist	-	0,93%	-	10,11%	10,38%	5,81%
vylúčenia: bf_syn	-	92,49%	-	89,87%	90,62%	89,36%
vylúčenia: bf_whitelist	-	6,47%	-	9,98%	9,23%	10,48%
vylúčenia spolu	-	98,96%	-	98,85%	99,85%	99,84%

Bloom filtrami a teda sa ďalej nespracovávajú. V rámci každého TCP spojenia sa spracuje vždy maximálne iba jeden ACK paket.

6.2.5 Rýchlosť

Aby som overil schopnosť mnou implementovaného programu zvládať rýchlostné požiadavky počítačovej siete, v ktorej bude využívaný som s pomocou vedúceho mojej bakalárskej práce program otestoval na sonde v sieti CESNET. Hardvér sondy pozostáva z dvojice procesorov Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz, ktoré majú spolu 12 fyzických jadier. Vďaka technológii hyperthreading disponujú až 24 logickými jadrami. Veľkosť pamäti RAM je 64 GB. Monitorované dáta sú prijímané 10 Gbps sieťovou kartou Intel 82599ES. Z hľadiska algoritmu je asi najdôležitejším parametrom veľkosť vyrovnávacej pamäti, ktorá činí 15360 kB. Všetky dátové štruktúry algoritmu sa teda pri vyššie uvedených parametroch pohodlne vojdú do tejto vyrovnávacej pamäti. Prevádzka na monitorovanej linke počas merania bola približne 700000 paketov za sekundu, čo je takmer maximum, ktoré sa na meranej linke bežne vyskytuje. Aj za týchto okolností program so základnou ako aj rozšírenou verziou algoritmu zvládal spracovávať takmer všetky príchodzie pakety. Počet zahodených paketov bol počas celého testovania zanedbateľný.

Kapitola 7

Záver

Cieľom tejto práce bola implementácia algoritmu pre detekciu skenovania portov vo vysokorýchlostných sieťach. Hlavnou myšlienkou tohto algoritmu je zahodenie čo najväčšieho objemu sieťovej prevádzky aby sa dosiahla redukcia nárokov algoritmu na procesor a pamäť. Použil som dva Bloom filtre, ktoré udržujú zoznam aktívnych destinácii a efektívne sledujú TCP spojenia a na sledovanie zlyhaných spojení som použil efektívnu top-k štruktúru. Obidva Bloom filtre spolu dokážu zahodiť v priemere až 80% zo všetkých paketov podieľajúcich sa na nadivazaní TCP spojenia.

Moje vyhodnotenie pre trasu A0 a B ukázalo, že algoritmus dosahuje perfektnú presnosť s veľmi malými pamäťovými požiadavkami, ktoré bez problémov spĺňajú rýchle výrovnávacie pamäte procesora. Vyhodnotenie trás C, D a F ukázalo, že algoritmus produkuje veľké množstvo falošných hlásení. Podrobnejším skúmaním týchto vzoriek dát som zistil, že v zachytenej komunikácii chýba smer komunikácie z B do A, teda SYN+ACK pakety. Je to spôsobené tým, že v sieti, z ktorej boli vzorky zachytené sa komunikácia smeruje rôznymi cestami. Vďaka tomu boli aj legitímne zdroje, ktoré nadvezovali väčšie množstvo úspešných TCP spojení označené za skenery portov. Na základe týchto zistení som navrhol rozšírenie algoritmu, ktoré na sledovanie aktívnych destinácii používa aj ACK pakety. Po zavedení rozšírenia som dosiahol redukciu počtu falošných hlásení v priemere až o 80%. Hoci po zavedení tohto rozšírenia algoritmus musel spracovať oveľa väčšie množstvo paketov, čas potrebný na spracovanie dát narástol v priemere len o 20% a to vďaka tomu, že po zapnutí rozšírenia Bloom filtre spolu zahodili v priemere takmer 99% zo všetkých spracovávaných paketov. Algoritmus som tiež vyhodnocoval online v sieti CESNET. Program bol spustený v čase špičky, teda sieťová prevádzka dosahovala svoje maximum, ktoré sa za bežných okolností na sieti vyskytuje. Aj so zapnutým rozšírením program bez problémov stíhal vyhodnocovať všetky pakety podieľajúce sa na TCP komunikácii.

Zavedením rozšírenia detekčného algoritmu som poukázal na výrazný vplyv smerovania komunikácie viac než jednou cestou v sieti na presnosť algoritmu, ktorého implementáciou a vyhodnotením sa zaoberá táto práca. Rozšírením algoritmu som podstatne znížil vplyv absencie komunikácie z B do A na presnosť. Týmto sa otvárajú ďalšie možnosti výskumu na čo najväčšiu minimalizáciu závislosti presnosti algoritmu aj na smer komunikácie z B do A. Zároveň sa táto práca zaoberá len detekciou TCP skenovania portov vo vysokorýchlostných sieťach. Preto by som rád detekciu UDP skenovania portov vo vysokorýchlostných sieťach ponechal ako predmet ďalšieho výskumu.

Literatura

[1] Bartoš, V.; Žádník, M.: Framework for comparision of network anomaly detection algorithms. Technická zpráva fit-tr-2012-02, Fakulta Informačních Technologií, VUT v Brně, CZ, 2012.

[2] Bartoš, V.; Žádník, M.; Varga, T.; aj.: HostStats. `http://sourceforge.net/projects/hoststats/`, 2013-08-30 [cit. 2014-04-10].

[3] Bartoš, V.; Čejka, T.; Žádník, M.: Nemea: Framework for stream-wise analysis of network traffic. Technická zpráva 9/2013, CESNET, Praha, CZ, 2013.

[4] Bloom, B.: *Space/time trade-offs in hash coding with allowable errors.* Communications of the ACM, ACM New York, NY, USA, str. 422-426., 1970, iSSN 0001-0782.

[5] Brauckhoff, D.; Tellenbach, B.; Wagner, A.; aj.: *Impact of packet sampling on anomaly detection metrics.* In Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, ACM New York, NY, USA, str. 159-164., 2006, iSBN 1-59593-561-4.

[6] Jung, J.; et al.: *Fast portscan detection using sequential hypothesis testing.* Security and Privacy, Proceedings. 2004 IEEE Symposium on, str. 211-225., 2004, iSBN 0-7695-2136-3.

[7] Mai, J.; Sridharan, A.; Chuah, C.; aj.: Impact of packet sampling on portscan detection. Technická zpráva, Sprint ATL, (accepted by IEEE JSAC Special Issue on Sampling the Internet)., 2006.

[8] Manerikar, N.; Palpanas, T.: *Frequent items in streaming data: An experimental evaluation of the state-of-the-art.* Data & Knowledge Engineering, Elsevier Science Publishers B. V. Amsterdam, The Netherlands, The Netherlands, str. 415-430., 2009, iSSN 0169-023X.

[9] Metwally, A.; Agrawal, D.; Abbadi, A.: *Efficient computation of frequent and top-k elements in data streams.* Proceedings of the 10th international conference on Database Theory, Springer-Verlag Berlin, Heidelberg, str. 398-412., 2005, iSBN 3-540-24288-0.

[10] Mikians, J.; et al.: *A practical approach to portscan detection in very high-speed links.* In Proceedings of the 12th international conference on Passive and active measurement, Springer, Berlin, str. 112-121., 2011, iSBN 978-3-642-19259-3.

[11] Nam, S.; Kim, H.: *Detector SherLOCK: Enhancing TRW with Bloom filters under memory and performance constraints.* Computer Networks: The International Journal of Computer and Telecommunications Networking, Elsevier North-Holland, str. 1545-1566., 2008, iSSN: 1389-1286.

[12] Paxson, V.: *Bro: a system for detecting network intruders in real-time.* Computer Networks: The International Journal of Computer and Telecommunications Networking, Elsevier North-Holland, Inc. New York, NY, USA, str. 2435-2463., 1999, iSSN 1389-1286.

[13] Schechter; et al.: *Fast detection of scanning worm infections.* Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, str. 59-81., 2004, iSBN 978-3-540-23123-3.

[14] Sridharan, A.; Ye, T.; Bhattacharyya, S.: *Connectionless port scan detection on the backbone.* Proc. of IPCCC, 2006.

[15] Weaver, N.; Staniford, S.; Paxson, V.: Very fast containment of scanning worms. In *Proc. of the 13th Conf. on USENIX Security Symposium*, USENIX, 2004.

[16] WWW stránky: MAWI Working Group Traffic Archive. `http://mawi.wide.ad.jp/mawi/`.

[17] WWW stránky: Denial-of-Service attack. `http://en.wikipedia.org/wiki/Denial-of-service_attack`, 2002-02-25 [cit. 2014-01-26].

[18] WWW stránky: Smurf attack. `http://en.wikipedia.org/wiki/Smurf_attack`, 2003-01-29 [cit. 2014-01-26].

[19] WWW stránky: SYN flood. `http://en.wikipedia.org/wiki/SYN_flood`, 2003-05-29 [cit. 2014-01-26].

[20] WWW stránky: Skenování portů. `http://en.wikipedia.org/wiki/Port_scanner`, 2004-04-04 [cit. 2014-01-26].

[21] WWW stránky: Ping flood. `http://en.wikipedia.org/wiki/Ping_flood`, 2005-07-25 [cit. 2014-01-26].

[22] WWW stránky: Hacker (computer security). `http://en.wikipedia.org/wiki/Hacker_(computer_security)`, 2005-08-07 [cit. 2014-01-26].

Příloha A

Obsah CD

Obsahom CD je priečinok ”src”, v ktorom sa nachádzajú zdrojové texty programu, inštalačný balíček a súbor ”README.txt”, ktorý obsahuje návod k inštalácii a manuál programu. Ďalej je obsahom CD priečinok ”doc”, kde sa nachádzajú všetky zdrojové súbory písomnej správy. CD taktiež obsahuje súbor ”pisomna_sprava.pdf”, ktorý obsahuje písomnú správu.

Příloha B

Manual

Táto aplikácis implementuje algoritmus detekcie skenovania portov vo vysokorýchlostných sieťach. Číta priamo dáta buď zo súboru alebo zo sieťového rozhrania. Dokáže čítať naraz z dvoch súborov alebo z dvoch sieťových rozhraní. Zároveň odosiela informácie o zdrojoch, ktoré označil ako skenery portov v UniRec formáte SRC_IP,EVENT_SCALE,TIMESLOT. Aplikácia bola vyvinutá pre operačný systém Linux.

B.1 Inštalácia

K inštalácii možno využiť inštalačný balíček nemea-1.0.0.tar.gz na priloženom CD. Postup inštalácie pozostáva z niekoľkých krokov. Najskôr je potrebné archív rozbaliť. Napríklad v terminále nasledujúcim príkazom.

```
tar -xzvf nemea-1.0.0.tar.gz
```

V ďalšom kroku prejdeme do priečinka nemea-1.0.0 a pomocou nasledovných príkazov nainštalujeme.

```
./configure
make -C libtrap
sudo make -C libtrap install
make -C unirec
```

K aplikácii sa z priečinka nemea-1.0.0 dostaneme pomocou nasledujúceho príkazu.

```
cd modules/portscan_detector
```

Potom už pre preloženie stačí aplikovať príkaz "make"

B.2 Parametre

-o FILE, –offline=FILE
Čítanie zachytenej prevádzky zo súboru. FILE predstavuje cestu k súboru. Aplikácia je schopná čítať maximálne z dvoch súborov zároveň.

-l INTERFACE, –online=INTERFACE
Čítanie sieťovej prevádzky priamo zo sieťového rozhrania. INTERFACE predstavuje názov sieťového rozhrania. Aplikácia je schopná čítať maximálne z dvoch sieťových rozhraní zároveň.

-s NUM, –syn=NUM
Veľkosť Bloom filtra bf_syn v bitoch. NUM predstavuje počet bitov.

-w NUM, –whitelist=NUM
Veľkosť Bloom filtra bf_whitelist v bitoch. NUM predstavuje počet bitov.

-k NUM, –topk=NUM
Maximálny počet elementov v top-k štruktúre. NUM predstavuje počet elementov.

-t NUM, –interval=NUM
Dĺžka intervalu v sekundách. NUM predstavuje počet sekúnd.

-r NUM, –treshold=NUM
Počet zlyhaných spojení, nad ktorý je zdroj považovaný za skener portov.

-p NUM, –spanm=NUM
Maximum medzi nižším a vyšším počítadlom elementu.

-z, –extension
Tento parameter aktivuje rozšírenie detekčného algoritmu.