VYSOKÉ UČENÍ TECHNICKÉ V BRNE
FAKULTA INFORMAČNÝCH TECHNOLOGIÍ

Počítačové komunikace a síťe, 2020/2021

Varinata OMEGA: Scanner síťových služeb

Dokumentácia k projektu č. 2

Marek Miček (xmicek08) 20.4.2021

Obsah

1. Úvod

Predmetom projektu bolo v jazyku C/C++/C# implementovať jednoduchý TCP,UDP port skener, ktorý na zadanom rozhraní a IP adrese oskenuje porty a na štandardný výstup vypíše výsledky tohto skenovania (open/closed pre UDP skenovanie a open/closed/filtered pre TCP skenovanie). Projekt sme sa rozhodli implementovať v jazyku C++ a v nasledujúcich častiach tejto dokumentácie rozoberieme postup pri implementácii tohto programu.

2. Popis princípu skenovania

2.1. TCP skenovanie

V tomto projekte sme mali uvažovať variantu TCP-SYN skenovania, teda nedochádza ku kompletnému 3-way-handshake. Inými slovami, nedôjde k úplnému nadviazaniu TCP spojenia. Skener generuje SYN paket a ak je cieľový port otvorený tak odpovie skeneru paketom typu SYN-ACK, pričom skener odpovie RST paketom, uzatvárajúc spojenie pred tým ako by bolo plne naviazané. V prípade, že cieľový port odpovie RST paketom, prehlásime cieľový port za uzavretý a v prípade, že cieľ neodpovedá, pošleme SYN paket opätovne, a pokiaľ neodpovedal ani po druhý raz, prehlásime ho za filtrovaný. Pri menej spoľahlivých sieťach sa typicky pri neobdržanej odpovedi SYN paket posiela viackrát (viz [1]). Avšak pri našom projekte budeme SYN paket pre každý port posielať maximálne dva razy.

2.2. UDP skenovanie

UDP skenovanie vyžaduje na rozdiel od TCP skenovania menšiu réžiu, pretože vôbec nenadväzuje spojenie. Celý proces funguje tak, že pošleme na cieľový port UDP datagram a čakáme na odpoveď. V prípade, že port je uzavretý obdržíme ICMP správu s kódom 3 (port unreachable error). V prípade, že neobdržíme žiadnu odpoveď prehlásime port za otvorený/filtrovaný. Problémom pri UDP skenovaní je, že port môžeme prehlásiť za otvorený, keďže sme neobdržali žiadnu odpoveď, no v skutočnosti ho môže blokovať firewall a je filtrovaný. Toto čiastočne odstraňuje prístup, kedy posielame UDP pakety so špecifikáciou pre aplikačnú vrstvu, dúfajúc, že práve aplikačná vrstva odpovie na náš dotaz (viz [2][3]). No v tomto projekte budeme považovať každý port, ktorý neodpovie ICMP správou s kódom 3 za otvorený.

3. Implementácia

Pakety odosielané na cieľové porty špecifikované užívateľom sú posielané pomocou takzvaných raw soketov, ktoré nám umožňujú vyplniť hlavičku každého paketu už na aplikačnej/programovej úrovni (viz [4]). Prichádzajúce pakety odchytávame pomocou knihovne `libpcap`, ktorá obsahuje funkcie na nastavenie filtra/odchytávača prichádzajúcich paketov, vďaka čomu môžeme o danom porte rozhodnúť v akom stave sa nachádza (viz [5]).

3.1 Spracovanie argumentov

Na začiatku sa samozrejme skontrolujú parametre príkazovej riadky. Vo funkcii `Check_Target` zistíme IP adresu a rodinu zadaného cieľového stroja a na základe tejto IP rodiny dohľadáme vo funkcii `Check_Interface` zariadenie, ktoré zodpovedá zadanému menu od užívateľa a IP rodine cieľového stroja. Pokiaľ užívateľ nezadal žiadne zariadenie, tak tento postup sa nevykoná, pričom sa vypíšu všetky aktívne zariadenia a ukonči sa program. Následne sa do príslušných vektorov uložia číselné hodnoty portov zadané užívateľom. Vo funkcii `Explode_Ports` dôjde k odstráneniu delimitera (čiarka alebo pomlčka v závislosti na formáte) a naplneniu týchto vektorov. Taktiež si uložíme timeout pre skenovanie jedného portu, ak ho užívateľ zadal (ak nie počítame s 5000 ms). Pokiaľ užívateľ nezadal očakávaný argument, prípadne zadal nesprávny počet argumentov, vypíše sa krátka nápoveda ako daný program používať. Týmto máme spracované argumenty a môžeme prejsť k samotnému skenovaniu.

3.2 Príprava pred skenovaním

Pred samotným skenovaním si vytvoríme v závislosti na IP rodine zadaného cieľa IP hlavičku (IPv4/IPv6 pomocou funkcie `Create_IP_Header/Create_IP6_Header`) (viz [6][7]) . Následne vytvoríme raw soket a pomocou funkcie `setsockopt` a argumentu `IP_HDRINCL` dáme kernelu na vedomie, že nemusí vyplniť IP hlavičky lebo budú vyplnené na programátorskej úrovni (viz [8]). V závislosti, či užívateľ zadal porty pre daný typ skenovania, prejdeme k samotnému skenovaniu. No musel zadať porty aspoň pre jeden typ skenovania, inak hlásime chybu.

3.3 Implementácia UDP skenovania

Na začiatku si vytvoríme odchytávač pre ICMP správy s kódom 3 pomocou funkcie `pcap_open_live`(viz [5]) voláme nami definovanú funkciu `Scan_UDP`. Tu postupne pre každý port, ktorý užívateľ zadal, naplníme UDP hlavičku pomocou funkcie `Create_UDP_Header`. Pokiaľ je cieľová adresa typu IPv4, tak kontrolný súčet pre UDP hlavičku počítať nemusíme, lebo je vyplnená operačným systémom. No pre IPv6 musíme vypočítať kontrolný súčet, pričom k tomu využívame pseudo hlavičku popísanú v štruktúre `UDP_v6_PSEUDO_HEADER` a funkciu `Get_Checksum` (viz [9][10]). Následne zostavíme výraz pre náš filter, ktorý bude, tak ako sme spomenuli vyššie, odchytávať pakety s ICMP správou s kódom 3 a pošleme na cieľovú adresu prvý paket. V tomto momente si vždy, keď skenujeme nejaký port, vytvoríme nové odchytávacie vlákno `UDP_Loop_Sniffer`, kde pomocou funkcie `pcap_loop`(viz [5]) budeme odchytávať nami hľadané pakety, pričom táto funkcia sa ukončí v momente, keď v hlavnom vlákne oskenujeme aktuálny port a zavoláme funkciu `pcap_break_loop`(viz [5]). V hlavnom vlákne po vytvorení odchytávacieho vlákna simulujeme čakanie na odpoveď portu pomocou funkcie `sleep`(viz [11]), ktorú voláme s parametrom `--wait` zadaným úžívateľom (prípadne implicitne 5000 ms). Po uplynutí tohto času sa pozrieme do globálnej premennej `Port_Closed` a v závislosti na jej hodnote určíme, či je daný port otvorený alebo zatvorený. No pre prístup k tejto premennej používame `mutexy`, aby sme zabránili súbežnému prístupu k tejto premennej, keďže aj odchytávacie vlákno má k nej prístup, pomocou funkcie `Callback_UDP` , ktorá je volaná v rámci funkcie `pcap_loop` zakaždým, keď náš filter zachytí správu ICMP s kódom 3, pričom sa modifikuje hodnota premennej `Port_Closed`.

3.4 Implementácia TCP skenovania

V prípade TCP skenovania postupujeme podobne, akurát si vytvoríme až dva odchytávače, jeden pre TCP RST odpovede a jeden pre TCP SYN-ACK odpovede a voláme funkciu `Scan_TCP`. Následne pre každý port, ktorý užívateľ zadal, vyplníme TCP hlavičku, pričom kontrolný súčet počítame pre oba typy cieľových adries, akurát používame iný typ pseudo hlavičky. Pre IPv4 pseudo hlavičku popísanú v štruktúre `TCP_PSEUDO_HEADER` a pre IPv6 v štruktúre `TCP_v6_PSEUDO_HEADER`. (viz [10][12][13]) Na výpočet kontrolného súčtu používame funkciu `Get_Checksum` rovnako ako aj pri UDP hlavičke. Ďalším krokom je zostavenie výrazov pre naše dva filtre. Tieto výrazy sa líšia v závislosti na typu cieľovej adresy (IPv4/IPv6). Jeden filter bude odchytávať pakety s odpoveďou RST a druhý s odpoveďou SYN-ACK. Následne odošleme na cieľovú adresu prvý paket, pričom si vždy, keď skenujeme nejaký port, vytvoríme dve nové odchytávacie vlákna (`RST_Thread, SYN_Thread`), pre každý filter jedno. V oboch vláknach beží v nekonečnej smyčke podobne ako aj u UDP skenovania funkcia `pcap_loop`, ktorá je prerušená v hlavnom vlákne po oskenovaní aktuálneho portu volaním funkcie `pcap_break_loop`. V hlavnom vlákne simulujeme čakanie na odpoveď portu funkciou `sleep` rovnako ako pri UDP skenovaní a následne sa pozrieme na obsah premennej `TCP_Port_State`. Keďže k tejto premennej majú prístup aj obe odchytávacie vlákna, ktoré ju modifikujú volaním funkcií `Callback_TCP_RST/Callback_TCP_SYN` v prípade, že filtre zachytia pakety

s požadovanými odpoveďami, tak znovu použijeme `mutexy`, aby sme zabránili súbežnému prístupu k tejto premennej. V hlavnom vlákne, po uplynutí času vyhradenom na čakanie odpovede skenovaného portu, rozhodneme o ďalšom postupe. Pokiaľ premenná `TCP_Port_State` obsahuje hodnoty definované v makrách `TCP_PORT_OPEN` alebo `TCP_PORT_CLOSED`, tak môžeme daný port prehlásiť za otvorený/uzavretý. Ak však obsahuje hodnotu v makre `TCP_PORT_FILTERED`, znamená to, že ani jeden filter nezachytil požadovanú odpoveď a celý postup opakujeme. Teda znovu pošleme paket na cieľovú adresu a znova po uplynutí určitého času nazrieme do premennej `TCP_Port_State`. Ak však znovu ani jeden filter nezachytí požadovanú odpoveď, tak prehlásime daný port definitívne za filtrovaný.

4. Testovanie

Výsledky skenovania portov pomocou nášho programu sme porovnávali s programom `NMAP`, ktorý je typu open-source a poskytuje funkcionalitu či už pre UDP alebo TCP-SYN skenovanie(viz [14]). Stav portov sme skenovali predovšetkým na `localhost`, kde sme testovali aj IPv4 aj IPv6 adresy, pričom výsledky boli uspokojivé. Mohlo by sa zdať, že by bolo vhodné porovnať aj rýchlosť akou sú dané porty oskenované, no pri našom projekte to je relatívne zbytočné, keďže pre každý skenovaný port máme parametrom --wait vyhradený čas, na čakanie odpovede od skenovaného portu. Ale asi každému je jasné, že program `NMAP` je omnoho výkonnejší a dokáže oskenovať všetkých 65536 portov vo veľmi krátkom čase, hoci napríklad náš program podporuje multivláknové programovanie, čo náš program môže relatívne urýchliť. No práca s vláknami nebola zvolená kvôli rýchlosti, ale skôr z implementačných dôvodov.

5. Záver

Na záver môžeme konštatovať, že nami implementovaný skener sieťových služieb do veľkej miery spĺňa požiadavky zo zadania projektu, pričom pre malý rozsah portov dokáže relatívne spoľahlivo zistiť ich dostupnosť. Projekt bol prínosný a istotne má potenciál nás podrobne oboznámiť s problematikou programovania na sieťovej vrstve. No žiaľ na našej škole máme aj množstvo iných projektov, čo nás dosť obmedzilo pri hľadaní rôznych optimalizácii výsledného programu, ktoré by náš výsledný program mohli spraviť efektívnejším.

Referencie

[1] TCP SYN (Stealth) Scan (-sS) | Nmap Network Scanning. [online]. [cit. 20.4.2021]. Dostupné z: https://nmap.org/book/synscan.html

[2] Lyon, G.: Nmap: The Art of Port Scanning. [online]. [cit. 20.4.2021]. Dostupné z: https://en.wikipedia.org/wiki/Port_scanner

[3] Port scanner – Wikipedia. [online]. [cit. 20.4.2021]. Dostupné z: https://nmap.org/nmap_doc.html#port_unreach

[4] Saxena, S.: A Guide to Using Raw Sockets - open source for you. [online]. 2015 [cit. 20.4.2021]. Dostupné z: https://www.opensourceforu.com/2015/03/a-guide-to-using-raw-sockets/

[5] pcap(3): Packet Capture library - Linux man page. [online]. [cit. 20.4.2021]. Dostupné z: https://linux.die.net/man/3/pcap

[6] IPv4 – Wikipédia. [online]. [cit. 20.4.2021]. Dostupné z: https://sk.wikipedia.org/wiki/IPv4

[7] IPv6 Packet Format - S12700 V200R011C10 Configuration Guide - IP Service – Huawei. [online]. [cit. 20.4.2021]. Dostupné z: https://support.huawei.com/enterprise/en/doc/EDOC1000178109/490ce67d/ipv6-packet-format

[8] raw(7) - Linux manual page. [online]. Marec 2021 [cit. 20.4.2021]. Dostupné z: https://man7.org/linux/man-pages/man7/raw.7.html

[9] Fairhurst, G.; Westerlund, M.: IPv6 UDP Checksum Considerations. [online]. Október 2010 [cit. 20.4.2021]. Dostupné z: https://tools.ietf.org/id/draft-ietf-6man-udpzero-02.html

[10] The TCP/IP Guide - TCP Checksum Calculation and the TCP "Pseudo Header". [online]. [cit. 20.4.2021]. Dostupné z: http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm

[11] sleep(3) - Linux manual page. [online]. Marec 2021 [cit. 20.4.2021]. Dostupné z: https://man7.org/linux/man-pages/man3/sleep.3.html

[12] Understanding the IPv6 Header. [online]. [cit. 20.4.2021]. Dostupné z: https://www.microsoftpressstore.com/articles/article.aspx?p=2225063&seqNum=6

[13] Transmission Control Protocol – Wikipedia. [online]. [cit. 20.4.2021]. Dostupné z: https://en.wikipedia.org/wiki/Transmission_Control_Protocol

[14] Nmap: the Network Mapper - Free Security Scanner. [online]. [cit. 20.4.2021]. Dostupné z: https://nmap.org/