VYSOKÉ UČENÍ TECHNICKÉ V BRŇE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IPK - Projekt č. 2

Varianta OMEGA: Scanner sieťových služieb

20. apríla 2019 Natália Holková (xholko02)

Obsah

1 Úvod

Úlohou v projekte bolo vytvoriť jednoduchý sieťový TCP, UDP port skener. Na štandartný výstup vypíše v akom stave sa nachádzajú porty – otvorené, filtrované alebo zatvorené. Výsledné riešenie bolo implementované v jazyku C.

2 Spracovanie argumentov a inicializácia libpcap

Spracovanie argumentov prebieha ručne pomocou cyklu `while`, čo sa ukázalo ako najrýchlejšie riešenie. Zoznam portov, ktorý je spočiatku uložený ako reťazec, je potrebné previesť na pole čísel. Toto rieši `check_port_range`, funkcia, ktorá kontroluje správnosť zadaného formátu. Pokiaľ nebola zadaná presná IP adresa cieľa, ale iba doména, je potrebné vykonať prevod. Základom celého prevodu je funkcia `getaddrinfo()` (viz. [1]). Preferovaná je IPv4 adresa pred IPv6.

Ďalej je potrebné zistiť vlastnú IP adresu. Verzia, ktorá sa bude získavať sa stanoví podľa verzie cieľovej adresy. Základom je funkcia `getifaddrs()` (viz. [2]). Vráti sa adresa, ktorá má správnu verziu a názov rozhrania.

V tejto časti sa inicializuje knižnica *libpcap*. Pomocou `pcap_open_live()` sa vytvorí „handler“, ktorý je uložený ako globálna premenná. Je to z dôvodou, že callback funkcii v `signal()` sa nedá predávať argument a toto predstavuje najjednoduchejšie riešenie. Detailný návod používania tejto knižnice je možné nájsť v [6].

Podľa zadaných portov sa najskôr zavolá UDP a potom TCP skenovanie pre vhodnú verziu IP adresy.

3 TCP skenovanie

Zvyčajne sa používa tzv. *„three-way handshake“* na synchronizáciu prepojenia medzi dvoma hostami. Prvý host vyšle packet s nastaveným flagom SYN, čo značí, že chce nadviazať spojenie. Druhý host na tento packet prijme a odpovie mu zaslaním packetu s flagmi SYN a ACK v pripade, že súhlasí s nadviazaním spojenia. Po prijatí tohto packetu prvý host ešte odpovie poslaním packetu s ACK a spojenie je vytvorené. Môže sa stať, že druhý host nechce nadviazať spojenie, vtedy sa odošle packet s RST a ACK a tým sa končí.

Jedna z metód TCP skenovania portov je SYN skenovanie, známe ako *„half-open SYN scan“*. Postupuje sa ako pri normálnom nadväzovaní spojenia, ale v prípade prijatia packetu s SYN a ACK sa ihneď odošle odpoveď s nastaveným RST, čím sa resetuje spojenie. Pokiaľ je port filtrovaný, neprípde žiadna odpoveď. Podrobnejšie je táto metóda vysvetlená v [4].

3.1 Implementácia pre IPv4

Pre jednoduchosť bude implementácia popisovaná na prípade, že cieľová IP adresa je typu IPv4. Neskôr bude porovnané riešenie pre IPv6. Implementácia TCP SYN skenera portov po tom, čo sú spracované argumenty programu sa dá rozdeliť do niekoľkých častí:

1. Deklarácie používaných štruktúr a vytvorenie socketu
2. Vyplnenie IP hlavičky a výpočet jej checksum
3. Vyplnenie pseudo hlavičky potrebnej pre výpočet TCP checksum
4. Vyplnenie TCP hlavičky a výpočet jej checksum
5. Priprava filtra na packety
6. Nastavenie a zaslanie packetu na cieľových port
7. Zachytávanie prichádzajúcich packetov a ich analýza

Na začiatku je potrebné deklarovať potrebné štruktúry, predovšetkým `struct iphdr` a `struct tcphdr`, ktoré predstavujú IP a TCP hlavičky.[1] Súčasne sa vytvorí socket pomocou funkcie `socket()` so správnymi parametrami, v tomto prípade `PF_INET`, `SOCK_RAW` a `IPPROTO_TCP`, čo znamená že budeme používať IPv4, sami si nastavíme hlavičky a použijeme TCP.

Nasleduje vyplnenie IP hlavičky. Tá sa počas celého skenovania nemení, preto ju stačí vyplniť iba raz na začiatku. Dôležité je správne nastaviť verziu, zdrojovú a cieľovú IP adresu cieľa. Presná forma IP hlavičky je definovaná v [8]. Zároveň sa vypočíta checksum.

Po naplnení pseudo hlavičky zdrojovou, cieľovou adresou a príslušným názvom protokolu vstupujeme do cyklu, keď postupne prechádzame všetky porty určené na testovanie. Zvyšné úlohy, na rozdiel od predchádzajúcich, je potrebné vykonať znovu pre každý port.

Pri vypĺňaní TCP hlavičky je potrebné meniť cieľový port (pre vyplnenie viz [9]) a počítať nový checksum. Pretože sa mení port, je treba meniť aj výraz na filtrovanie packetov. Výraz je v tvare:

```
tcp src port {1} and dst port {2}
```

kde *1* je konštantná hodnota - číslo portu, z ktorého odosielame všetky packety a *2* je aktuálne kontrolovaný port na cieľovej IP addrese. Tu sa využívajú funkcie z knižnice libpcap, kontrétne `pcap_compile()` a `pcap_setfilter`. Nastaví sa socket pomocou `setsockopt()`, aby automaticky nepridával hlavičky a odošle sa pomocou `sendto` na správnu adresu a port.

Nasleduje zachytávanie odpovedi pomocou `pcap_loop()`, ktorá zavolá nadefinovanú callback funkciu iba v prípade, že prišiel packet, ktorý prejde cez nastaveý filter. V tom prípade sa zoberie obsah packetu a namapuje sa časť, ktorá má reprezentovať TCP hlavičku na premennú, aby sa dalo pristupovať k flagom. Pokiaľ je nastavený SYN a ACK flag, port sa vyhlási za otvorený, pokiaľ je naopak nastavenýRST a ACK, port je uzavretý.

Môže nastať situácia, že port je filtrovaný alebo z nejakého dôvodu neprišiel packet do svojej destinácie. V tom prípade `pcap_loop()` nezachytí žiadny packet a zacyklí sa. Toto je ošetrené pomocou signálov, kedy je pred samotným odchytávaním nastavený alarm funkciou `alarm()` na určitý čas – *TIMEOUT* – a po jeho uplynutí sa vykoná `pcap_breakloop()`, ktorá zastaví odchytávanie packetov. Podľa návratovej hodnoty `pcap_loop()` sa dá rozpoznať, že bolo použité `pcap_breakloop()` a pokiaľ ani na druhý pokus nepríde odpoveď, odznačí sa port za filtrovaný.

3.2 Potrebné zmeny pre IPv6

Implementácie pre IPv6 je v podstate obdobná tej pre IPv4, ale je potrebné vykonať určité zmeny. Líši sa presný postup, ako odosielať TCP packety. Pre presný popis viď [3]. Pre IP hlavičku sa používa štrukturá `struct ip6_hdr` a pre addresu štruktúra `struct sockaddr_i6`. Výpočet checksum pre IP sa preto tiež líši.

4 UDP skenovanie

UPD je protokol, ktorý nenadväzuje spojenie. Ak je port otvorený, cieľ vôbec neodpovie. V prípade, že je port, zavretý, odošle správu *ICMP destination port unreachable* (typ 3, kód 3). Na rozdiel od TCP sa nedá rozlíšiť medzi otvoreným a filtrovaným portom. Presnejšie je to opísané v [5].

Pre každý skenovaný port sa vyplní UDP hlavička podľa a vypočíta checsum podľa [7]. Odošla sa packet pomocou `sendto()` a čaká sa na odpoveď.

Odchytávanie odpovede je znovu riešené funkciou `pcap_loop()`. Pokiaľ sa zachytí packet cez nastavený filter, znamená to, že port je uzavretý. Callback funkcia má za úlohu iba toto vypísať. Možnosť, že nepríde žiadny packet je znovu riešená cez signály. Nastaví sa alarm na vopred určitý čas a po ňom sa zavolá `pcap_breaklookp()`. V prípade, že nastane timeout sa pokladá port za otvorený.

4.1 Implementácia pre IPv4

Implementácia UDP skeneru je jednoduchšia ako TCP skeneru, najmä preto, že stačí kontrolovať, či neprišla odpoveď *ICMP typu 3, kód 3*. Celé riešenie je opäť možné rozdeliť na niekoľko častí:

[1] Možné ich nájsť v hlavičkových súboroch netinet/ip.h a netinet/tcp.h

1. Deklarácie používaných štruktúr a vytvorenie socketu
2. Vyplnenie IP hlavičky a výpočet checksum
3. Vyplnenie pseudo hlavičky potrebnej pre výpočet UDP checksum
4. Príprava filtra na packety
5. Vyplnenie UDP hlavičky a výpočet checksum
6. Odoslanie packetu
7. Zachytenie odpovede *destination port unreachable* alebo prehlásenie portu za otvorený

Pre IP hlavičku sa opäť používa `struct iphdr` a pre UDP hlavičku `struct udphdr`.[2]. Vyplňovanie IP hlavičky je presne zhodné s postupom pri TCP skenovaní, rovnako aj vyplnenie pseudo hlavičky, iba sa mení názov protokolu a veľkosť štruktúry.

Pretože sa iba kontroluje, či neprišla odpoveď *destination port unreachable*, stačí nastaviť filter raz na začiatku funkcie. Opäť sa využijú funkcie z knižnice *libpcap* a filtrovaný výraz je tvaru `"icmp[0] = 3"`.

4.2 Potrebné zmeny pre IPv6

Princíp riešenia je rovnaký ako pre IPv4. Pre IP hlavičku je použitá `struct ip6_hdr`, líši sa aj výpočet checksum.

5 Testovanie

Program bol testovaný na distribúcii Ubuntu a na poskytnutej referenčnej virtuálke. Pri programovaní posielania packetov bol používaný nástroj *Wireshark*[3], ktorý poskytuje prehľad všetkých prichádzajúcich a odchádzajúcich packetov, zobraziť obsah jednotlivých hlavičiek a aj kontrolu správnosti checsumu.

Pri testovaní bol tiež využitý nástroj *nmap*, ktorý tiež poskytuje skenner portov s možnosťou TCP a UDP skenu. Referečné výsledky nástroja nmap boli porovnávané s výsledkami mojej implementácie.Pri kontrolu TCP skenu som používala `nmap -sS -p <port-range> <IP>` a pre UDP sken `nmap -sU -p <port-range> <IP>`.

Väčšina testovania prebiehala na *localhost* alebo testovaní portov referenčnej virtuálky. Pre kontrolu funkčnosti pri IPv6 adrese sa použila adresa `https://ipv6.google.com`.

[2]Túto štruktúru je možné nájsť v súbore netinet/udp.h

[3]K dispozícii na `https://www.wireshark.org/`.

6 Referencie

[1] *GETADDRINFO(3) Linux Programmer's Manual* [online]. Marec 2019. http://man7.org/linux/man-pages/man3/getaddrinfo.3.html.

[2] *GETIFADDRS(3) Linux Programmer's Manual* [online]. Marec 2019. http://man7.org/linux/man-pages/man3/getifaddrs.3.html.

[3] CASTRO, E. M. Porting applications to IPv6 HowTo. [online]. http://long.ccaba.upc.edu/long/045Guidelines/eva/ipv6.html.

[4] ETUTORIALS.ORG. 4.2 TCP Port Scanning. [online]. http://etutorials.org/Networking/network+security+assessment/ /Chapter+4.+IP+Network+Scanning/4.2+TCP+Port+Scanning/.

[5] ETUTORIALS.ORG. 4.3 UDP Port Scanning. [online]. http://etutorials.org/Networking/network+security+assessment/ /Chapter+4.+IP+Network+Scanning/4.3+UDP+Port+Scanning/.

[6] GARCIA, L. M. Programming with Libpcap - Sniffing the Network From Out Own Application. *Hacking*. Feb 2008, roč. 3, č. 2. S. 38–46.

[7] POSTEL, J. *User Datagram Protocol* [Internet Requests for Comments]. August 1980. RFC, 793. http://www.rfc-editor.org/rfc/rfc768.txt.

[8] POSTEL, J. *Internet Protocol* [Internet Requests for Comments]. September 1981. RFC, 791. http://www.rfc-editor.org/rfc/rfc791.txt.

[9] POSTEL, J. *Transmission Control Protocol* [Internet Requests for Comments]. September 1981. RFC, 793. http://www.rfc-editor.org/rfc/rfc793.txt.