

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií

POČÍTAČOVÉ KOMUNIKACE A SÍŤ
2018/2019

2. Projekt

Scanner Síťových služeb

Obsah

Úvod	2
Implementace	2
Zpracování argumentů.....	2
TCP skenování.....	2
UDP skenování.....	3
Výpis.....	3
Testování.....	4
Závěr.....	6
Použitá literatura a zdroje	6

Úvod

Zadání projektu bylo vytvořit jednoduchý TCP/UDP skener, který oskenuje zadané porty na zadaném zařízení a na standardní výstup vypíše stav těchto portů (tj. otevřený, filtrovaný, uzavřený). Pakety musí být odeslané pomocí BSD soketů a program by měl být co nejvíce multiplatformní na operačních systémech založených na unixu.

Implementace

Program je implementován v programovacím jazyce C++ ve standardu C++11. K odesílání paketů používá RAW sokety a pro jejich zachytávání externí knihovnu libpcap, kterou vyžívá například i známý program Wireshark. Program pracuje na portu 49224 a podporuje pouze IP protokol verze IPv4.

Zpracování argumentů

Jako první krok bylo potřeba zpracovat vstupní parametry programu. Jelikož program podporuje poměrně velké množství parametrů bylo by výhodné použít dostupné funkce pro jejich zpracování, ovšem já se rozhodl jít cestou podmínek, cyklů a regulárních výrazů.

V cyklu procházíme argumenty a pokud narazíme na známý přepínač, tak se rozhodneme o další akci, tj. načtení portů, interfacu či adresy vzdáleného stroje, pokud nepřecházel žádný přepínač. Ta může být zadána přímo adresou nebo jménem domény. Pro převod byla použita funkce `getaddrinfo()`, která je součástí knihovny `netdb.h`. Porty mohou být zadané výčtem nebo rozsahem. O který zápis se jedná je rozhodnuto regulárním výrazem a následně se volá funkce, která tyto argumenty vyčte a uloží do příslušného globálního vektoru. Interface je nepovinný argument, podle kterého volíme naši IP adresu. Pokud nebyl interface zadán, vezmeme první neloopbackovou adresu a informace uložíme do globální struktury reprezentující koncové zařízení komunikace.

TCP skenování

Pokud byl zadán aspoň jeden TCP port zavolá se funkce `TCPscan()`, která provede celé skenování. Jako první se inicializuje paket, do kterého na začátek vložíme IP hlavičku a naplníme ji informacemi (adresu příjemce a odesílatele, protokol atd..) a vypočítáme kontrolní součet. Za IP hlavičku vložíme TCP hlavičku a naplníme ji informacemi (například zdrojový a cílový port atd..) a kontrolní součet se v tomto případě prozatím nastaví na 0. Pro výpočet TCP kontrolního součtu musíme vytvořit pomocný paket, do kterého vložíme tzv. „pseudo TCP hlavičku“ a za ni nakopírujeme tu originální. Pak vypočítáme kontrolní součet nad celým pomocným paketem a výsledek uložíme do originální TCP hlavičky.

Poté co máme nastavený paket inicializujeme a nastavíme RAW soket, přes který paket odešleme. Otevřeme zachytávání paketů přes funkci `pcap_open_live()`, která je součástí knihovny `pcap/pcap.h`. Dále nastavíme filtr, který bude filtrovat všechny příchozí pakety, pouze na ten, který nás zajímá. Nastavíme tedy filtrování na TCP protokoly, IP odesílatele, IP příjemce a cílový port. Filtr zkompilujeme a nastavíme přes funkce

`pcap_compile()` a `pcap_setfilter()`. Pak už jen stačí paket odeslat přes funkci `sendto()` a zachytit odpověď přes `pcap_dispatch()`. Ovšem odpovědi se nemusíme dočkat, pokud server má port filtrovaný nebo se odpověď ztratí, proto čas na přijmutí paketu musíme omezit. To lze přes systémový signál `SIGALRM` a nastavíme čas na 2 sekundy. Po uplynutí tohoto intervalu voláme funkci `pcap_breakloop()`, která ukončí čekání.

Výsledek TCP skenu závisí na odpovědi serveru, ten se může zachovat třemi způsoby:

- 1) Odešle zpátky paket s flagy SYN a ACK, které znamenají, že server chce pokračovat v komunikaci, a proto lze port označit za otevřený.
- 2) Odešle zpátky paket s flagy ACK a RST, které znamenají, že server nechce pokračovat v komunikaci, a proto port lze označit za uzavřený.
- 3) Server neodpoví, a to může znamenat, že náš paket zahodil a je port je filtrovaný nebo se paket po cestě ztratil, a proto odešleme paket pro jistotu ještě jednou, než ho označíme za filtrovaný.

IP hlavička je naplněna pouze jednou. TCP hlavička je naplněna standartními údaji, ale některé části je potřeba měnit pro každý skenovaný port. Konkrétně to je cílový port, sekvenční číslo a kontrolní součet. Kontrolní součet je náhodně zvolené číslo, které se s každým paketem inkrementuje.

UDP skenování

UDP skenování provádí funkce `UDPscan()` a princip je velmi podobný TCP skenování, proto zmíníme jen zajímavé odlišnosti. Jako první je naplnění UDP hlavičky, která obsahuje podstatně méně informací než TCP a nemusíme v ní počítat kontrolní součet (alespoň v případě IPv4). Dále UDP protokol nenavazuje žádné spojení, proto nelze ověřovat dostupnost portů podobným způsobem jako u TCP, ale lze využít protokolu ICMP. Tento protokol je určen pro signalizaci dostupnosti. Při zaslání našeho UDP paketu na nedostupný port, operační systém stanice generuje ICMP paket typu 3 kódem 3 (port unreachable). Proto pokud tuto zprávu neobdržíme, lze považovat port za otevřený, jinak za uzavřený. Bohužel zde není žádný způsob, jak zjistit, že port je filtrovaný. Další problém je, že ICMP zpráva je zapouzdřena přímo v jediném datagramu, a tak jako UDP nezaručuje doručení paketů, proto jak v případě TCP odešleme dotaz ještě jednou.

Výpis

Během celého programu je potřeba zaznamenávat důležité informace od IP adresy, přes seznam portů až po jejich dostupnost. To je realizováno globálním stringem, do kterého postupně tyto informace ukládáme a na konci ho vypíšeme. Další výhodou má, že v případě chyby nejsou na standardním výstupu napůl vypsané informace. Pokud nastane chyba je na standardní chybový výstup vypsaná chybová hláška a program je ukončen návratovou hodnotou 1.

Testování

1. Test – ipk-scan

```
student@student-vm: ~
File Edit View Search Terminal Help
student@student-vm:~$ sudo ./ipk-scan -i lo -pt 21-23 -pu 30,50 localhost
Interesting ports on 127.0.0.1:
PORT      STATE
21/tcp    closed
22/tcp    closed
23/tcp    closed
30/udp    closed
50/udp    closed
student@student-vm:~$
```

1. Test – Wireshark

Capturing from any

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 49224 || udp.port == 49224

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	56	49224 → 21 [SYN] Seq=0 Win=65535 Len=0
2	0.000031492	127.0.0.1	127.0.0.1	TCP	56	21 → 49224 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	1.164746314	127.0.0.1	127.0.0.1	TCP	56	49224 → 22 [SYN] Seq=0 Win=65535 Len=0
4	1.164820155	127.0.0.1	127.0.0.1	TCP	56	22 → 49224 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	2.156442314	127.0.0.1	127.0.0.1	TCP	56	49224 → 23 [SYN] Seq=0 Win=65535 Len=0
6	2.156496092	127.0.0.1	127.0.0.1	TCP	56	23 → 49224 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	3.251976915	127.0.0.1	127.0.0.1	UDP	44	49224 → 30 Len=0
8	3.252023273	127.0.0.1	127.0.0.1	ICMP	72	Destination unreachable (Port unreachable)
9	4.324548525	127.0.0.1	127.0.0.1	UDP	44	49224 → 50 Len=0
10	4.324605723	127.0.0.1	127.0.0.1	ICMP	72	Destination unreachable (Port unreachable)

Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 49224, Dst Port: 21, Seq: 0, Len: 0

any: <live capture in progress> Packets: 38 · Displayed: 10 (26.3%) Profile: Default

2. Test – ipk-scan

```
student@student-vm: ~
File Edit View Search Terminal Help
student@student-vm:~$ sudo ./ipk-scan nemeckay.net -pt 22,80,1000,1001,1002,1003
Interesting ports on 46.28.109.159:
PORT      STATE
22/tcp    open
80/tcp    open
1000/tcp  filtrated
1001/tcp  closed
1002/tcp  closed
1003/tcp  filtrated
student@student-vm:~$
```

2. Test – Wireshark

The Wireshark interface shows a capture on interface 0 with the filter `tcp.port == 49224 || udp.port == 49224`. The packet list displays 16 packets. Packets 3, 4, 6, 7, 9, 10, 11, 13, 14, 15, and 16 are TCP SYN packets from 10.0.2.15 to 46.28.109.159 on port 49224. Packets 5 and 8 are RST responses from 46.28.109.159 to 10.0.2.15. Packets 12 and 14 are RST, ACK responses from 46.28.109.159 to 10.0.2.15. Packets 13 and 15 are SYN, ACK responses from 46.28.109.159 to 10.0.2.15. The packet details pane shows the selected packet (No. 3) as a TCP SYN packet. The packet bytes pane shows the raw data of the selected packet.

No.	Time	Source	Destination	Prot	Length	Info
3	0.043271683	10.0.2.15	46.28.109.159	TCP	56	49224 → 22 [SYN] Seq=0 Win=65535 Len=0
4	0.059768287	46.28.109.159	10.0.2.15	TCP	62	22 → 49224 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
5	0.059848769	10.0.2.15	46.28.109.159	TCP	56	49224 → 22 [RST] Seq=1 Win=0 Len=0
6	1.116538068	10.0.2.15	46.28.109.159	TCP	56	49224 → 80 [SYN] Seq=0 Win=65535 Len=0
7	1.135270411	46.28.109.159	10.0.2.15	TCP	62	80 → 49224 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
8	1.135309258	10.0.2.15	46.28.109.159	TCP	56	49224 → 80 [RST] Seq=1 Win=0 Len=0
9	2.175836346	10.0.2.15	46.28.109.159	TCP	56	49224 → 1000 [SYN] Seq=0 Win=65535 Len=0
10	4.176195838	10.0.2.15	46.28.109.159	TCP	56	[TCP Retransmission] 49224 → 1000 [SYN] Seq=0 Win=65535 Len=0
11	6.235770027	10.0.2.15	46.28.109.159	TCP	56	49224 → 1001 [SYN] Seq=0 Win=65535 Len=0
12	6.251647714	46.28.109.159	10.0.2.15	TCP	62	1001 → 49224 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	7.319308923	10.0.2.15	46.28.109.159	TCP	56	49224 → 1002 [SYN] Seq=0 Win=65535 Len=0
14	7.333833169	46.28.109.159	10.0.2.15	TCP	62	1002 → 49224 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15	8.431550029	10.0.2.15	46.28.109.159	TCP	56	49224 → 1003 [SYN] Seq=0 Win=65535 Len=0
16	10.431914455	10.0.2.15	46.28.109.159	TCP	56	[TCP Retransmission] 49224 → 1003 [SYN] Seq=0 Win=65535 Len=0

3. Test – ipk-scan

The terminal shows the command `sudo ./ipk-scan -pu 20,10 nemeckay.net -pt 100` being executed. The output displays the interesting ports on 46.28.109.159: 100/tcp is filtered, 20/udp is open, and 10/udp is open.

```
student@student-vm:~$ sudo ./ipk-scan -pu 20,10 nemeckay.net -pt 100
Interesting ports on 46.28.109.159:
PORT      STATE
100/tcp   filtrated
20/udp    open
10/udp    open
student@student-vm:~$
```

3. Test – Wireshark

The Wireshark interface shows a capture on interface 0 with the filter `tcp.port == 49224 || udp.port == 49224`. The packet list displays 10 packets. Packets 3, 4, 5, 6, 8, 9, and 10 are UDP packets from 10.0.2.15 to 46.28.109.159 on port 49224. Packets 7 and 10 are TCP SYN packets from 46.28.109.159 to 10.0.2.15. The packet details pane shows the selected packet (No. 3) as a UDP packet. The packet bytes pane shows the raw data of the selected packet.

No.	Time	Source	Destination	Prot	Length	Info
3	0.045236998	10.0.2.15	46.28.109.159	TCP	56	49224 → 100 [SYN] Seq=0 Win=65535 Len=0
4	2.060233786	10.0.2.15	46.28.109.159	TCP	56	[TCP Retransmission] 49224 → 100 [SYN] Seq=0 Win=65535 Len=0
5	4.136925257	10.0.2.15	46.28.109.159	UDP	44	49224 → 20 Len=0
6	6.137464900	10.0.2.15	46.28.109.159	UDP	44	49224 → 20 Len=0
8	8.212833615	10.0.2.15	46.28.109.159	UDP	44	49224 → 10 Len=0
9	8.212833615	10.0.2.15	46.28.109.159	UDP	44	49224 → 10 Len=0
10	10.213284553	10.0.2.15	46.28.109.159	UDP	44	49224 → 10 Len=0

Závěr

Zpočátku mě projekt velmi bavil a líbí se mi jeho praktická použitelnost. Díky němu se moje teoretické i praktické znalosti sítí nemálo obohatily. Jeho náročnost byla vyšší, než jsem původně očekával. Jelikož jsem měl problémy při testování IPv6, tak jsem se rozhodl ji raději vynechat než odevzdat řádně neotestovaný program.

Použitá literatura a zdroje

- Příklady RAW soketů - <https://www.tenouk.com/Module43a.html>
- ICMP - <https://cs.wikipedia.org/wiki/ICMP>
- Převod domény na IP adresu - <https://stackoverflow.com/questions/5760302/when-i-do-getaddrinfo-for-localhost-i-dont-receive-127-0-0-1>
- Kontrolní součet - <https://stackoverflow.com/questions/8845178/c-programming-tcp-checksum>
- Zpracovávání odpovědí - <https://www.tcpdump.org/pcap.html>
- SIGALRM - <https://stackoverflow.com/questions/4583386/listening-using-pcap-with-timeout>