IPK 2. Projekt - Scanner síťovýh služeb

Jaromír Wysoglad - xwysog00

20. Dubna 2019

Obsah

1 Úvod

Cílem projektu je vytvoření jednoduchého nástroje pro skenování portů zadaného cíle. Skenování je v případě TCP uskutečněno pomocí odeslání SYN packetu a čekáním na odpověď v podobě ACK packetu pro otevřený port, nebo RST packetu pro uzavřený port. V přípádě, že žádná odpověď nepřijde ani po druhém odeslaném packetu, považuje se daný port za filtrovaný. V případě UDP skenování probíhá odesláním datagramu na daný port a čekáním na odpověď v podobě ICMP zprávy "destination unreachable," v tomto případě se port považuje za uzavřený, pokud zpráva nepřijde, je port otevřený, nebo filtrovaný.

2 Nastudované informace

2.1 TCP

TCP, neboli Transmission control protocol je spolehlivý protokol na transportní vrstvě. TCP hlavička obsahuje několik řídících bitů, využívaných pro zajištění spolehlivosti.

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

Figure 1: Struktura TCP hlavičky

Zdroj: [3]

Pro účely skenování je zajímavý tzv. "Three-way handshake," ke keterému dochází při navazování každého nového spojení.

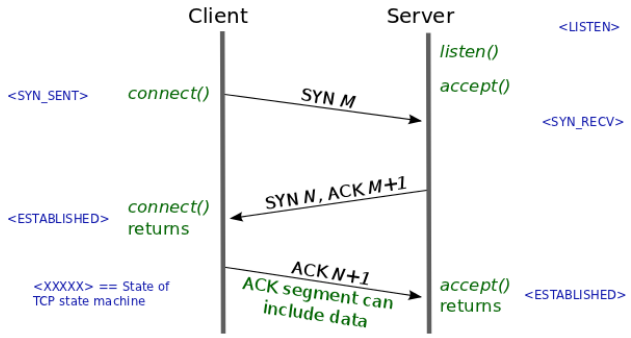

Figure 2: Three-way handshake

Zdroj: https://static.lwn.net/images/2012/tfo/3whs.png

Při handshake klient nejprve posílá packet s nastaveným SYN bitem. Server následně potvrdí přijetí odesláním packetu s nastaveným ACK bitem a pokud je port otevřen, nastaví také SYN bit, pokud je port uzavřen, nastaví RST bit. Klient poté potvrdí přijetí packetu odesláním packetu s nastaveným ACK bitem. Tohoto chování lze využít při skenování.

2.2 UDP

UDP, neboli User datagram protocol je nespolehlivý protokol na transportní vrstvě. Narozdíl od TCP není přesně definováno chování při úspěšném navázání spojení, proto nelze rozlišit otevřený a filtrovaný port. Při uzavřeném portu však cíl skenu odpovídá ICMP zprávou destination unreachable, což je zpráva typu 3, kódu 3 pro IPv4 a typu 1, kódu 4 pro IPv6. Čehož lze využít pro skenování.

UDP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Figure 3: Struktura UDP hlavičky

Zdroj: [5]

2.3 Checksum

Důležitou součástí hlavičky téměř všech protokolů je checksum, jehož výpočet je pro všechny protokoly stejný a to: "The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero." [4] Implementaci výpočtu checksum jsem použil z: https://www.linuxquestions.org/questions/programming-9/raw-sockets-checksum-function-56901/ [1] Pro výpočet checksum TCP a UDP je však nutné před hlavičku protokolu přidat některá data z hlavičky IP protokolu. Hlavička pro UDP, ze které se checksum počítá tedy v závislosti na verzi IP vypadá jako jeden z následujících obrázků (obdobně také pro TCP)

IPv6 Pseudo Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv6 Address																															
4	32																																
8	64																																
12	96																																
16	128	Destination IPv6 Address																															
20	160																																
24	192																																
28	224																																
32	256	UDP Length																															
36	288	Zeroes																								Next Header							
40	320	Source Port																Destination Port															
44	352	Length																Checksum															
48	384+	Data																															

IPv4 Pseudo Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source IPv4 Address																															
4	32	Destination IPv4 Address																															
8	64	Zeroes								Protocol								UDP Length															
12	96	Source Port																Destination Port															
16	128	Length																Checksum															
20	160+	Data																															

Figure 4: Struktura UDP pseudohlavičky pro výpočet checksum

Zdroj: [5]

3 Implementace

Program je rozdělen do 4 modulů. Scan obsahuje main funkci, provádí parsování argumentů programu, rozhoduje kterou verzi IP použít a poté zavolá skenovací funkci pro příslušnou verzi IP. Common obsahuje funkce pro získání jednotlivých portů. Ipv4 obsahuje funkce potřebné pro provedení IPv4 skenu. Ipv6 podobně jako ipv4 obsahuje funkce pro provedení IPv6 skenu.

3.1 Parsování argumentů

3.1.1 Argumenty programu

Pro parsování argumentů programu je použita funkce *getopt*. Ta sice slouží pro parsování krátkých jednopísmenných argumentů, ale argumenty -pu a -pt lze chápat jako dva po sobě jdoucí argumenty -p -t nebo -p -u, díky čemuž lze funkci *getopt* bez problému použít. Nejprve jsou tedy pomocí *getopt* zpracovány argumenty začínající pomlčkou a zbývající argument je poté považován za cíl skenu. Pokud je přítomen neznámý argument, nebo je argumentů špatný počet, vytiskne se chybová hláška, příklad správného použití podle zadání a program se ukončí s chybou. Parsování hodnot skenování portů probíhá až při samotném skenu, kdy se podle toho, jestli argument obsahuje pouze číslo, čísla s čárkami, nebo čísla a pomlčku rozhodne, zda se jedná pouze o jediný port, nebo o seznam portů a nebo a rozmezí portů. Podle toho se volá jedna z funkcí z modulu common, která si do statické proměnné ukládá poslední navrácený port a vrací číslo dalšího portu v pořadí.

```
//determine in witch format the ports were given
if(ports_tcp.find(',') != string::npos)
    get_next_port = next_in_list;

else if(ports_tcp.find('-') != string::npos)
    get_next_port = next_in_range;

else
    get_next_port = next_in_solo;

//the scan
while((port = get_next_port(ports_tcp, false)) >= 0)
```

3.1.2 Rozhodnutí o použité verzi IP

Po úspěšném rozparsování argumentů programu se pomocí funkce *getifaddrs* získá IPv4 a IPv6 adresa zdrojového síťovného rozhraní a pomocí *getaddrinfo* se provede případný překlad cílové adresy na obě verze IP. Následně se hledá verze IP, pro kterou máme cílovou i zdrojovou adresu a volá se funkce pro skenování z příslušného modulu s tím, že přednost má IPv4.

3.2 IPv4

3.2.1 Sken

Jako první se provede inicializace hlaviček protokolů, inicializace socketu a kontrola dostupnosti cílové sítě ze zdrojového síťového rozhraní. Inicializaci hlaviček jsem provedl podle https://www.tenouk.com/Module43a.html [2]. A poté se v závislosti na argumentech programu provede sken v jednotlivých transportních protokolech. Pokud byl zadán argument -pt, provede se TCP sken postupným odesíláním SYN packetů v cyklu na jednotlivé porty a následným čekáním na odpověď. Pokud byl zadán argument -pu, provede se UDP sken.

3.2.2 Zachytávání odpovědí

Zachytávání odpovědí probíhá pomocí knihovny pcap. Jako první je inicializován handle, poté se díky funkce *pcap_activate* spustí zachytávání packetů a pak se nastaví filtr podle toho, co chceme zachytit. Nyní stačí pouze v cyklu pomocí *pcap_next* projít všechny zachycené rámce, zkontrolovat jejich obsah a podle toho vypsat výsledek skenu pro daný port.

3.3 IPv6

Implementace IPv6 skenu je prakticky totožná s IPv4, rozdíly jsou pouze v použitých funkcích a jejich parametrech, které bylo nutno změnit pro práci s IPv6 a v naplňování IP hlavičky, která má jinou strukturu než IPv4 hlavička. Veškerý kód tohoto modulu je můj vlastní, nebo ze stejných zdrojů jako IPv4, ale upravený pro IPv6.

4 Testování

Testování probíhalo skenováním virtuálního stroje s nakonfigurovaným apache serverem a iptables tak, aby na něm šly otestovat všechny výsledky skenů. V prvotních fázích projektu jsem správnou práci programu kontroloval pomocí nástroje tcpdump, kterým jsem zachytával příchozí a odchozí packety a kontroloval jejich správnost. V pozdějších fázích projektu jsem výsledky skenu projektu srovnával se skenem stejných portů pomocí nástroje nmap.

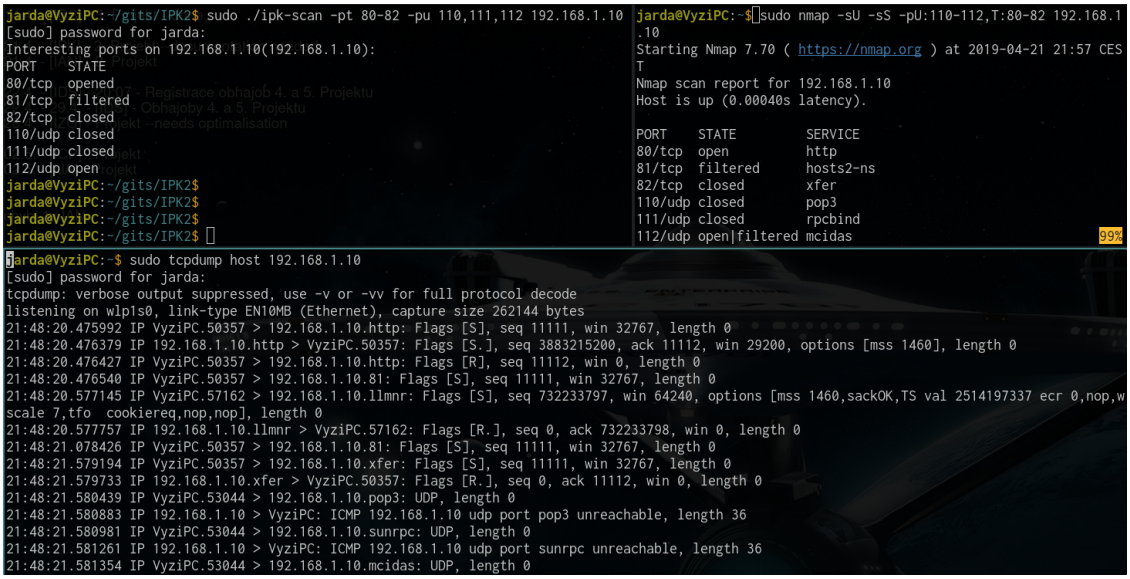

Figure 5: Ukázka testování projektu

References

[1] Encrypted. *Raw Sockets, Checksum Function*. Jan. 2013. URL: https://www.linuxquestions.org/questions/programming-9/raw-sockets-checksum-function-56901/.

[2] *LINUX SOCKET PART 17 Advanted TCP/IP - THE RAW SOCKET PROGRAM EXAMPLES*. URL: https://www.tenouk.com/Module43a.html.

[3] *Transmission Control Protocol*. Apr. 9, 2019. URL: https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

[4] Information Sciences Institute University of Southern California. *RFC: 791*. Sept. 1981. URL: https://tools.ietf.org/html/rfc791.

[5] *User Datagram Protocol*. Apr. 20, 2019. URL: https://en.wikipedia.org/wiki/User_Datagram_Protocol.