VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

Fakulta informačních technologií
Faculty of Information Technology

IPK

PROJEKT 2

Brno, 2019

Lukáš Havlíček (xhavli46)

Obsah

Obsah 1
1 Popis programu 2
1.1 TCP skenování 2
1.2 UDP skenování 2
2 Implementace 3
2.1 Argumenty 3
2.2 TCP hlavička 3
2.3 UDP hlavička 4
2.4 IP hlavička 5
2.5 Posílání packetů 5
2.6 Chytání packetů 6
3 Testování 7
Literatura 8
Seznam obrázků 9

1 Popis programu

Program slouží pro oskenování portů na zadané IP adrese či doménovém jméně pomocí TCP/UDP a byl napsán v jazyce C++ pomocí RAW socketů a knihovny libcap.

1.1 TCP skenování

Při TCP skenování se využívá nastavení flagu SYN, který značí „inicializaci“ komunikace a na základě odpovědi je daný port označen za otevřený, filtrovaný či uzavřený.
Přesněji podle následujících příznaků přijaté odpovědi:

-Příchozí packet má nastaven RST flag = port je uzavřený

-Příchozí packet nemá nastavený RST flag = port je otevřený

-Nepřišla odpověď = port je filtrovaný

1.2 UDP skenování

Při UDP skenování je možnost prohlásit daný port pouze za otevřený nebo uzavřený. Protože u UDP nejsou žádné flagy, pouze port odesílatele, příjemce, délka a „checksum“, který zde nastavuji pouze při skenování ivp6 adresy, protože u ipv4 je toto nepovinná hodnota a je možno ji nahradit nulou.
Stav portu je určen dle následujícího:

-Přišla odpověď Port unreachable = port je uzavřený

-Nepřišla odpověď = port je otevřený

2 Implementace

Jak jsem již psal v popis programu, program využívá pro skenování RAW socketů pro odesílání a pro chycení knihovnu libcap. Díky tomu je potřeba program spouštět se sudo oprávněním jinak nastane „socket() error“ a program bude ukončen s návratovou hodnotou -1.

2.1 Argumenty

K parsování argumentů byla doporučena funkce getopt(), která ale bere pouze krátké argumenty tj. například „-a“ a getopt_long bere pouze dlouhé argumenty jako například „--aa“, díky tomu jsem se rozhodl si tuto část udělat manuálně za pomocí for cyklu přes argumenty, aby byla zachována možnost zadat argumenty v jakémkoliv pořadí. Kde si jak v případě portů zadaných jako rozsah, tak oddělených čárkou vytvořím vektor čísel pro lehčí opakování skenování na další porty.

Možné argumenty:

-pt ports – porty na oskenování pomocí TCP

-pu ports – porty na oskenování pomocí UDP

-i interface – volitelný argument specifikující interface

IP | Domain Name – argument specifikující cíl pro skenování

Upřesnění:

Ports = možnost zadat jako čísla oddělena čárkou, jako rozsah pomocí pomlčky

Interface = v případě absence je zvolen první neloopbackový aktivní interface

2.2 TCP hlavička

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.) ...																															

Obrázek 1 – TCP header

Jako reprezentaci hlavičky v programu používám strukturu *struct tcphdr* která se nachází v hlavičkovém souboru *netinet/tcp.h*. Kde mi při plnění daty pomohla stránka [1], která byla v zadání zmíněna jako doporučená literatura. Největším problém při implementaci byl pravděpodobně

Checksum, aby bylo zaručenu že packet přišel nepoškozen, ke kterému byla potřeba vytvořit další struktura pro tzv. pseudo hlavičku, která obsahuje určité informace z IP hlavičky.

TCP pseudo-header for checksum computation (IPv4)

Bit offset	0–3	4–7	8–15	16–31
0	Source address			
32	Destination address			
64	Zeros		Protocol	TCP length

Obrázek 2 – Pseudo header for ipv4

Tato Pseudo hlavička viz. Obrázek č. 2, byla poté přidána před reálnou TCP hlavičku a byl nad nimi spočítám checksum. Normálně se počítá i nad daty za hlavičkami, ale zde se žádná data neodesílají. U protokolu ipv6 se tato pseudo hlavička mírně liší, jak lze vidět na obrázku č. 3.

TCP pseudo-header for checksum computation (IPv6)

Bit offset	0–7	8–15	16–23	24–31
0	Source address			
32				
64				
96				
128	Destination address			
160				
192				
224				
256	TCP length			
288	Zeros			Next header

Obrázek 3 – Pseudo header for ipv6

2.3 UDP hlavička

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														

Obrázek 4 – UDP header

Jako reprezentaci UPD hlavičky jsem se rozhodl použít strukturu ze stránky [1], nejdříve jsem zkoušel stejně jako u TCP hlavičky použít strukturu z hlavičkového souboru *netinet/udp.h*, ale při úpravách programu mi UDP přestalo správně fungovat a rozhodl jsem se použít jinačí strukturu. Jak jsem již psal v popisu UDP skenování, tak lze i na obrázku č. 4 vidět že UDP hlavička nemá žádné flagy k nastavení. Tím pádem nastavuji pouze port příjemce, odesílatele, délku a checksum a ten pouze v případě že skenuji ipv6 adresu, u ipv4 je tato hodnota nepovinná, takže nastavuji hodnotu 0.

U protokolu ipv6 je potřeba daný checksum vypočítat a stejně jako u protokolu TCP se k tomu využívá pseudo hlavička, přesněji je i stejná jako u TCP ipv6 viz. Obrázek č. 3, jen teda s tím rozdílem že zde není TCP length, ale UDP length.

2.4 IP hlavička

IPv4 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	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

Obrázek 5 – Ipv4 header

Pro ipv4 používám *struct ip* z hlavičkového souboru *netinet/ip* a stejně jako u ostatních hlaviček jsem se při plnění daty inspiroval stránkou [1].

Fixed 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	*Version*				*Traffic Class*								*Flow Label*																			
4	32	*Payload Length*																*Next Header*								*Hop Limit*							
8	64	*Source Address*																															
12	96																																
16	128																																
20	160																																
24	192	*Destination Address*																															
28	224																																
32	256																																
36	288																																

Obrázek 6 – Ipv6 header

Pro ipv6 jsem se rozhodl napsat si vlastní strukturu kterou budu používat přesněji v programu tedy *struct my_ip6*, protože mi struktura z hlavičkového souboru *netinet/ip6* přišla trochu zmateně napsaná. Na rozdíl od ipv4 se zde již nepočítá checksum, co jsem se dočetl je to z důvodu, že v dnešní době je síť více „stabilní“ a packety se nepoškozují tak často, aby byl potřeba checksum viz. [2].

2.5 Posílání packetů

K posílání packetů využívám funkce sendto() které je potřeba předat buffer ve které je již vyplněná ip hlavička a dále pak TCP/UDP hlavička. V programu mám přesněji tyto buffery 2. První slouží, pro již zmíněný účel a zároveň pro výpočet checksumu do ip hlavičky u ipv4 protokolu a druhý pro výpočet

checksumu up TCP/UDP hlavičky (u UDP hlavičky pouze při ipv6). Přičemž tedy druhý buffer, co používám obsahuje pseudo hlavičku a dále pak hlavičku pro kterou se checksum vypočítává.

2.6 Chytání packetů

Chytání packetů je pomocí knihovny libcap, která byla zmíněna jako doporučená/vyžadovaná v zadání. Kde ji nejprve nastavuji pomocí pcap_open_live(), přičemž tato funkce bere jako jeden z argumentů timeout. Bohužel při testování a následném studiu na internetu jsem zjistil, že linux tento timeout nepodporuje a je potřeba jej vyřešit jinak, k tomu jsem musel použít alarm(), který jsem nastavil jak u TCP tak u UDP na 3s, což mi při testování přišlo jako dostatečná hodnota aby program zachytával příchozí packety. Ještě ale před nastavením alarmu je potřeba nastavit filtr, aby knihovna zachytávala pouze správné packety, které má zpracovat. Filtr u TCP jsem zvolil na základě portů a typu zprávy tcp u UDP jsem pak zvolil že typ zprávy je ICMP port unreachable. Tím pádem ve funkci starající se o zpracování packetu UDP stačí pouze vypsat, že je daný port uzavřený. Protože pokud je otevřený tak žádná odpověď nepřijde. V případě tcp je potřeba se podívat na flagy přijatého packetu a rozhodnout se na základě RST flagu. Pokud při TCP nepřijde odpověď tak zkusím packet zaslat znovu a pokud i na podruhé nepřijde odpověď tak je daný port označen za filtrovaný. Tím pádem může nastat maximální zdržení při skenování 1 portu TCP až 6s a u UDP 3s.

3 Testování

K testování jsem převážně využíval programu Wireshark, který mi velice pomohl jak při různých opravách naplnění dat do hlaviček, tak kontrola checksumu, ale i také kontrola, zda mi můj program zachytává správné packety a správně vypisuje stav portu dle této odpovědi. Většina testů probíhala vůči localhostu a následně vůči určitým serverům, jedním z nich byl například nemeckay.net, sever jednoho ze spolužáků, od kterého jsme dostali souhlas ke skenování. Nejhůře se testovala ipv6, vzhledem k tomu že doma nemáme ipv6 adresu, tak jsem testoval přes VPN, ale nedocházeli mi odpovědi, takže si zde bohužel nejsem jist, do jaké míry program zpracuje příchozí odpovědi při ipv6, jelikož jsem otestoval pouze zachytávání odpovědí při ipv6 vůči localhostu.

Literatura

[1] Tenouk: THE RAW SOCKET PROGRAM EXAMPLES [online]. Dostupné na URL: < https://www.tenouk.com/Module43a.html >

[2] Pim van Pelt: why there is no checksum in IPv6 header? [online]. Nov 20, 2002 Dostupné na URL: < https://mailman.isi.edu/pipermail/6bone/2002-November/006839.html >

Seznam obrázků

Obrázek č. 1-3: https://en.wikipedia.org/wiki/Transmission_Control_Protocol
Obrázek č. 4: https://en.wikipedia.org/wiki/User_Datagram_Protocol
Obrázek č. 5: https://en.wikipedia.org/wiki/IPv4#Header
Obrázek č. 6: https://en.wikipedia.org/wiki/IPv6_packet#Fixed_header