VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MONITOROVÁNÍ SÍTĚ
NETWORK MONITORING

PROJEKTOVÁ DOKUMENTACE
PROJECT DOCUMENTATION

AUTOR PRÁCE Mezigalaktický president MICHAL MRNUŠTÍK
AUTHOR

BRNO 2017

Obsah

Kapitola 1

Úvod

Cílem tohoto projektu je vytvořit program umožňující skenování aktivních hostů v počítačové síti a následně u aktivních hostů provést skenování otevřených TCP nebo UDP portů.

V rámci řešení projektu musíme vyřešit problém zjištění aktivních hostů na přímo a nepřímo připojené síti, následně musíme vyřešit jakým způsobem detekovat, zda jsou na zařízení otevřené TCP nebo UDP porty. Inspirací pro řešení tohoto problémů je pro nás program skenovací program nmap[1].

Kapitola 2

Teoretický úvod

V rámci řešení projektu musíme vyřešit dva základní problémy a to detekci aktivních hostů a následnou detekci otevřených portů.

V této kapitole se podíváme na to jakým způsobem jsme vyřešili detekci aktivních hostů na přímo nebo nepřímo připojené síti. Zároveň se podíváme na způsob detekce otevřených portů TCP a UDP portů na zařízeních.

2.1 Detekce aktivních hostů

V rámci našeho programu máme detekovat aktivní hosty na síti. Dále máme specifikováno v zadání[7], že na přímo připojené síti máme detekovat všechna připojená zařízení a na nepřímo připojené síti detekovat zařízení respektovat RFC 1122[2].

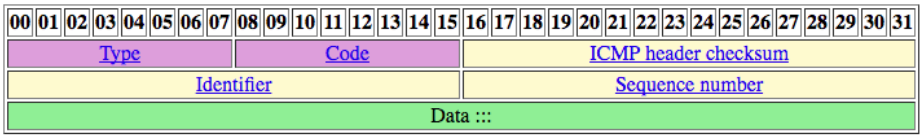
V RFC 1122[2] v sekci 3.2.2.6 je definováno, že každý host na síti musí implementovat ICMP server, který, pokud mu dojde ICMP echo request zpráva, odpoví zprávou typu ICMP Echo reply. Tím pádem všichni hosté respektující RFC 1122 musí odpovídat na tyto zprávy. Pro skenování nepřímo připojené síti proto zvolíme použití ICMP echo zpráv.

V přímo připojené síti je potřeba detekovat všechna zařízení. ICMP protokol je pro toto řešení nevhodný, protože zařízení nemusí na ICMP zprávy odpovídat (viz [4]). Z tohoto důvodu budeme pro detekci v přímo připojené síti používat Address Resolution Protocol, jehož blokování by znemožnilo zařízení síťovou komunikaci.

2.1.1 Internet Control Message Protocol (ICMP)

Internet Control Message Protocol je L3 protokol definovaný v RFC 792[9]. Tento protokol byl navržen pro detekci chyb, řešení problémů na síti a detekci nedostupnosti zařízení. Seznam všech typů naleznete v již zmiňovaném RFC[9].

Nás konkrétně zajímají zprávy typu Echo Request a Echo Reply. Jakmile je na hosta zaslána zpráva zpráva typu Echo Request. Tento host (za předpokladu, že dodržuje RFC



Obrázek 2.1: Tvar hlavičky pro ICMP Echo request/reply

1122[2]) na tuto zprávu odpoví zprávou typu Echo Reply se stejnými daty jako byly v původní zprávě (tato data mohou být použita odesilatelem requestu pro kontrolu správnosti odpovědi).

2.1.2 Address Resolution Protocol (ARP)

Address Resolution Protocol je L3 protokol definovaný v RFC 826[6]. Tento protokol je navržený pro překlad L2 adres (v našem případě MAC adresa) na L3 adresy (v našem případě IPv4 adresa).

ARP paket obsahuje informace o:

- typu zařízení
- typu L3 protokolu jehož adresu potřebujeme (v našem případě IPv4)
- velikosti hardwareové (L2) adresy
- velikosti adresy používané požadovaným L4 protokolem
- L2 a L3 adresa odesilatele
- L2 (broadcast) a L3 adresa, kterou hledáme

Odesilatel nejprve pošle na L2 broadcast dotaz, kdo má danou L3 adresu. Pokud je na síti připojen příjemce, tak na zprávu odpoví tentokrát už na unicast L2 adresy odesilatele, že on je ten hledaný a má tuto L3 adresu.

2.2 Zjišťování otevřených portů

V případě zjišťování, které porty jsou otevřené, musíme rozlišovat transportní protokoly, který služba běžící na daném využívá pro přenos dat. Výrazně se liší, jakým způsobem funguje zjišťování otevřených portů služeb používajících TCP, které vytváří spojení, a služeb používajících UDP, které je bezstavové.

2.2.1 Transmission Control Protocol (TCP)

Transmission Control Protocol je L4 protokol definovaný v RFC 793[10]. Tento protokol je používaný pro spolehlivý a bezchybný přenos dat přes síť.

Při navázání spojení TCP dochází pomocí techniky zvané threeway handshake. Navázání spojení vypadá takto:

1. Klient odešle na server TCP paket s nastaveným příznakem SYN, náhodným číslem sekvence a potvrzovacím číslem 0.
2. Server paket přijme a odpoví a odešle klientovi paket s nastavenými příznaky SYN a ACK, zvýšeným potvrzovacím číslem rovným klientovu číslu sekvence + 1 a vlastním náhodným číslem sekvence.
3. Klient odešle na server paket s příznakem ACK a potvrzovacím číslem rovným serverovému číslu sekvence 1 větší a vlastním číslem sekvence zvětšeným rovněž 0 1.

Při skenování portů můžeme právě tohoto využít. Pokud pošleme na všechny porty paket s nastaveným příznakem SYN, tak by nám ze všech otevřených portů měla přijít odpověď s nastavenými SYN a ACK příznaky. V případě, že bude port zavřený měla by nám přijít zpráva s nastavenými příznaky RST a ACK.

Důležité je v případě, že jsme se dozvěděli, že je port otevřený zaslat klientovi požadavek s příznakem RST a ukončit tak připojení, aby nezůstávalo na druhé straně napůl otevřené připojení.

2.2.2 User Datagram Protocol (UDP)

User Datagram Protocol je L4 protokol definovaný v RFC 768[8]. UDP je nespojovaný transportní protokol s minimálním množstvím režie. Pakety posílané pomocí UDP nemají žádnou jistotu jejich pořadí, doručení nebo duplikace.

Vzhledem k tomu, že je protokol nespojovaný, nelze u něj provádět detekci otevřených portů jako u TCP (více v kapitole 2.2.1). UDP díky svému nespojovanému chování neposkytuje žádný způsob detekce zda paket došel z klienta na server. V tomto nám pomáhá ICMP, které jsme tu zmínili v předchozí kapitole. V případě, že pošleme UDP paket na server, nám server odpoví ICMP zprávou typu Destination Unreachable, díky které jsme schopni říct, že port je uzavřen. V případě, že nám ICMP zpráva nepřijde je port považován za otevřený.

Kapitola 3

Návrh

V rámci kapitoly teoretický úvod (viz. 2) jsme si popsali, teoretická řešení jednotlivých typů skenování, která má náš program podporovat. V této kapitole se budeme věnovat tomu, jakým způsobem se na náš program podíváme z hlediska návrhu jednotlivých částí programu. Vzhledem k tomu, že je umožněno projekt psát v C++, rozhodl jsem se pro použití objektově orientovaného přístupu k návrhu kódu.

3.1 Reprezentace IPv4 sítě

V rámci projektu potřebujeme vyřešit, jakým způsobem reprezentovat IPv4 sítě a jejich adresy, abychom byli jednoduše schopní zkontrolovat, zda je síť validní, a zároveň, aby bylo jednoduché iterovat nad všemi adresami dané sítě.

Pro reprezentaci sítě jsem navrhl třídu `IpNetwork`. Pro zajištění, že může vzniknout pouze objekt reprezentující validní síť, jsem použil návrhový vzor tovární metoda[11]. V rámci třídy IpNetwork je statická metoda `IpNetwork::FromCidr(std::string network)`, která umožňuje vytvoření instance objektu IpNetwork pouze z řetězce reprezentujícího validní síť ve formátu CIDR[3]. V případě, že je předaný řetězec nevalidního

Pro zajištění možnosti jednoduché interace nad všemi adresami v síti jsem navrhl třídu `IpNetworkEnumerator`. Jedná se implementaci návrhového vzoru iterátor[12]. Tento návrhový vzor je pro umožnění iterování nad kolekcí objektů bez nutnosti znalosti kolik objektů v kolekci je. Třída `IpNetworkEnumerator` definuje metodu `MoveNext()`, která vrací bool a indikuje, zda můžeme v iteraci ještě pokračovat (v našem případě, zda následující adresa není broadcast). Dále je definována metoda `Current()`, která vrací aktuální adresu reprezentovanou objektem `IpAddress`. Enumerator je možné získat zavoláním metody `GetEnumerator()` nad objektem `IpNetwork`.

3.2 Skenovací nástroje

Za účelem skenování jsem se rozhodl pro vytvoření jedné abstraktní třídy `ScannerBase`, která bude použita jako bázová pro všechny třídy, které provádět skenování. Tato třída bude svým podtřídám nabízet sadu základních metod, které umožní sdílet kód pro síťovou komunikaci, a oddělí tak platformně specifické části kódu (tvorba/uzavírání soketů, nastavování socketů,...). Zároveň se tak budeme moci v podtřídách `ScannerBase` soustředit na skenování jako takové a neřešit problémy při tvorbě a nastavování socketů.

Třídy pro vyhledávání aktivních hostů `ArpScanner` a `IcmpScanner` obsahují metodu `ScanNetwork` vracející `std::vector<IpAddress>`, jehož obsahem budou objekty `IpAddress` reprezentující aktivní hosty ve skenované síti.

Třídy pro skenování otevřených portů `TcpScanner` a `UdpScanner` obsahují metodu `Scan`, přijímající jako parametr objekt typu `IpAddress` reprezentující adresu hosta, jenž má být skenován. Zároveň tyto třídy obsahují metody, které jsou specifické pro daný typ skenování (vytvoření TCP hlavičky, odeslání TCP RST zprávy,...).

Kapitola 4

Implementace

V minulé kapitole jsme si popsali, jakým způsobem jsem navrhl námi vybrané řešení. V této kapitole si popíšeme, jaké prostředky operačního systému a knihovny libpcap[5] jsme použili pro implementaci našeho řešení.

4.1 Detekce lokální sítě

Pro výběr mezi skenováním aktivních hostů pomocí protokolu ARP nebo ICMP je potřeba vědět, jestli je zadaná síť přímo připojená (tzn. nějaká síťová karta zařízení je k ní připojená) nebo nepřímo připojená. Pro řešení tohoto problému je nejprve potřebné zjistit, jaká sítová zařízení jsou ke stanici připojena. V tom jsem využil možností knihovny libpcap[5]. V rámci této knihovny je přítomna funkce `pcap_findalldevs()`, která umožňuje přístup ke všem síťovým zařízením, která jsou na zařízení přítomná a jejich adresám. Poté už stačilo iterovat nad zařízeními a porovnávat adresy sítě k nim připojeným s adresou, kterou nám zdal uživatel.

4.2 Síťová komunikace

Pro síťovou komunikaci jsem vybral řešení za pomocí BSD schránek, které jsou nám poskytované operačním systémem. Ve všech případech odesílání používáme neblokující sockety (ukázka nastavení socketu na neblokující Listing 4.1).

```
int flags = fcntl(socketFd, F_GETFL, 0);
fcntl(socketFd, F_SETFL, flags | O_NONBLOCK);
```

Listing 4.1: Nastavení socketu na neblokující.

4.2.1 ICMP komunikace

Pro komunikaci pomocí protokolu ICMP (více o protokolu v kapitole 2.1.1) je potřeba nastavit při vytváření nastavit socket na tzv. RAW socket a zároveň nastavit hodnotu protokolu na `IPPROTO_ICMP`, pro obě tyto nastavení je potřeba, aby program byl spuštěný s právy root.

Poté je potřeba odeslat vyplněnou ICMP hlavičku. V této věci nám pomáhá operační systém, který poskytuje makra a struktury pro ICMP hlavičku v hlavičkovém souboru `netinet/ip_icmp.h`.

Po odeslání hlavičky na všechny zařízení stačí akorát naslouchat a kontrolovat příchozí zda příchozí packety jsou ICMP pakety typu ECHO Reply a mají stejné id jako námi odeslané pakety.

4.2.2 ARP komunikace

Pro zaslání a získávání ARP požadavků pomocí BSD socketů je potřeba nejprve nastavit do domény socketu hodnotu `AF_PACKET`. Tato hodnota nastaví socket, aby komunukoval na L2 úrovni.

V případě ARPu operační systém bohužel nenabízí ani makrat ani struktury pro ARP hlavičku, což znamená, že si je musíme poskytnout sami (námi použitá struktura Listing 4.2). Následně musíme hlavičku vyplnit potřebnými konstantami a odeslat na ethernetový broadcast (FF:FF:FF:FF:FF:FF).

```
typedef struct {
    struct {
        uint8_t destinationMac[6];
        uint8_t sourceMac[6];
        uint8_t type[2];
    } eth;
    struct {
        uint16_t hardwareType;
        uint16_t protocolType;
        uint8_t hardwareLength;
        uint8_t protocolLength;
        uint16_t operationCode;
        uint8_t senderMac[6];
        uint8_t senderIp[4];
        uint8_t targetMac[6];
        uint8_t targetIp[4];
    } arp;
    uint8_t padding[36];
} arpEthPacket;
```

Listing 4.2: Struktura reprezentující ethernet a ARP hlavičku.

Následně se přijímají pakety a pokud se jedná o packety typu ARP Reply, uloží se adresa ze které informace přišla.

4.2.3 UDP komunikace

Pro skenování UDP portů je potřeba otevřít dva BSD sockety. Jeden pro klasickou UDP komunikaci (s natavením `SOCK_DGRAM`) a druhý pro ICMP (stejně jako v subsection 4.2.1).

Nejprve je potřeba odeslat na všechny porty cokoliv pomocí UDP. Toto můžeme udělat, protože na testovacích zařízeních bude v rámci sysctl nastavený `net.ipv4.icmp_ratelimit` na 0.

Poté stačí pouze poslouchat na ICMP socketu a čekat jestli nám příjdou zprávy typu ICMP Unreachable. V rámci této zprávy také nachází část původní zprávy, ze které jsme schopni vyčíst o jaký port se jednalo a označit ho za uzavřený.

4.2.4 TCP komunikace

Pro skenování TCP portů postačí jeden RAW socket s nastaveným protokolem na `IPPROTO_TCP`.

Do tohoto socketu musíme vyslat vyplněnou IP hlavičku společně s TCP hlavičkou. Tyto můžeme opět získat z operačního systému a to z hlavičkových souborů `netinet/ip.h` a `netinet/tcp.h`. Po vyplnění potřebných bitů do obou hlaviček můžeme je obě v rámci jednoho bufferu odeslat.

Poté opět naslouchámе na odpovědi ve formě TCP packetů s nastaveným příznakem SYN nebo RST. Po přijetí těchto paketů odešleme zpět na hosta TCP paket s příznakem RST, aby spojení mezi stanicemi nezůstalo napůl otevřené.

Kapitola 5

Návod

Nástroj isamon je program spustitelný z příkazové řádky používaný ke skenování počítačových IPv4 sítí.

```
isamon [-h] [-i <interface>] [-t] [-u] [-p <port>] [-w <ms>] -n <net_address/mask>
```

Listing 5.1: Možnosti použití nástroje isamon.

5.1 Parametry

Povinné parametry:

- **-n** <adresa_sítě/maska> - adresa a mask sítě, která má být skenována ve formátu CIDR[3].

Nepovinné parametry:

- **-h** - vypíše nápovědu k užívání programu isamon
- **-i** <rozhraní> - specifikuje rozhraní, které má být použito ke komunikaci
- **-t** - určuje, že má dojít ke skenování otevřených TCP portů
- **-u** - určuje, že má dojít ke skenování otevřených UDP portů
- **-p** <port> - zobrazí aktivní klienty, kteří mají otevřený tento port
- **-w** <ms> - maximální povolený RTT v ms

5.2 Ukázky použití

```
$ isamon -n 192.168.1.0/30 -t -u -w 5
192.168.1.1
192.168.1.1 TCP 80
192.168.1.1 TCP 22
192.168.1.1 UDP 53
```

Listing 5.2: Ukázka použití isamon pro skenování TCP a UDP portů sítě 192.168.1.0/30 s maximálním RTT 5 ms.

```
$ isamon -n 192.168.1.0/30
192.168.1.1
192.168.1.2
```

Listing 5.3: Ukázka použití isamon pro skenování aktivních hostů na síti 192.168.1.0/30.

Literatura

[1] Port Scanning Techniques. Online, Navštíveno 11. 11. 2017.
URL https://nmap.org/book/man-port-scanning-techniques.html

[2] Braden, R.: Requirements for Internet Hosts - Communication Layers. STD 3, RFC Editor, October 1989, http://www.rfc-editor.org/rfc/rfc1122.txt.
URL http://www.rfc-editor.org/rfc/rfc1122.txt

[3] Fuller, V.; Li, T.: Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. BCP 122, RFC Editor, August 2006, http://www.rfc-editor.org/rfc/rfc4632.txt.
URL http://www.rfc-editor.org/rfc/rfc4632.txt

[4] Gite, V.: Linux disable or drop / block ping packets all together. Jan 2007, [Online; navštíveno 7.11. 2017].
URL https://www.cyberciti.biz/faq/howto-drop-block-all-ping-packets/

[5] Harris, G.: Programming with pcap. Online, Navštíveno 11. 11. 2017.
URL http://www.tcpdump.org/pcap.html

[6] Plummer, D. C.: Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. STD 37, RFC Editor, November 1982, http://www.rfc-editor.org/rfc/rfc826.txt.
URL http://www.rfc-editor.org/rfc/rfc826.txt

[7] Pluskal, J.: Zadání projektu. [Online; navštíveno 7. 11. 2017].
URL
https://github.com/nesfit/ISA/blob/master/projects/MonitorovaniSite.md

[8] Postel, J.: User Datagram Protocol. STD 6, RFC Editor, August 1980, http://www.rfc-editor.org/rfc/rfc768.txt.
URL http://www.rfc-editor.org/rfc/rfc768.txt

[9] Postel, J.: Internet Control Message Protocol. STD 5, RFC Editor, September 1981, http://www.rfc-editor.org/rfc/rfc792.txt.
URL http://www.rfc-editor.org/rfc/rfc792.txt

[10] Postel, J.: Transmission Control Protocol. STD 7, RFC Editor, September 1981, http://www.rfc-editor.org/rfc/rfc793.txt.
URL http://www.rfc-editor.org/rfc/rfc793.txt

[11] Shvets, A.: Factory Method Design Pattern. Online, Navštíveno 11. 11. 2017.
URL https://sourcemaking.com/design_patterns/factory_method

[12] Shvets, A.: Iterator Design Pattern. Online, Navštíveno 11. 11. 2017.
URL https://sourcemaking.com/design_patterns/iterator