Jednoduchý TCP/UDP scanner v C++

Vložil/a czokl [1], 21 Březen, 2011 - 13:36

- Networks & Protocols [2]
- Programming [3]

Asi před dvěma lety jsem měl ve škole projekt na vytvoření jednoduchého TCP/UDP scanneru. Rozhodl jsem se s vámi o svojí implementaci rozdělit. Mám pro to dva důvody, první je, že možná vás zajímá jak takový scanner může fungovat a určitě je lehčí analyzovat kód triviality než třeba Nmapu. Druhý důvod je ten, že najít funkční příklady odeslání TCP/UDP raw socketů je docela umění. Problém je ten, že někdo kdysi vytvořil nějaký kód pro UDP a všichni ostatní ho od té doby kopírují, přitom u UDP nepotřebujete počítat checksum, čímš se situace dost ulehčuje.

Následující text je výcuc dokumentace k projektu, měli byste tam najít všechno důežité, na konci článku naleznete odkaz na kód.

Zadání znělo asi takto: Vytvořte jednoduchý síťový TCP, UDP skener. Program oskenuje zadanou IP adresu a porty. Na standardní výstup vypíše, v jakém stavu se porty nacházejí (otevřený, filtrovaný, uzavřený). Jako techniku skenování použijte u TCP protokolu SYN scan. Program musí být přeložitelný a spustitelný na operačním systému FreeBSD.

TCP skenování: Posílá pouze SYN pakety. Neprovádí tedy kompletní 3-way-handshake. Pokud přijde odpověď RST - port je označen jako uzavřený. Pokud po daný časový interval nepřijde ze skenovaného portu odpověď, je nutno ověřit dalším paketem a teprve potom port označit jako filtrovaný. Pokud na daném portu běží nějáká služba, je port označen jako otevřený. Více viz RFC 793.

UDP skenování: U UDP skenování můžete uvažovat, že daný počítač při zavřeném portu odpoví ICMP zprávou typu 3, kódu 3 (port unreachable). Ostatní porty považujte za otevřené.

Pakety musí být odeslané pomocí BSD sockets. Odchytávat odpovědi můžete např. pomocí knihovny libpcap.
Volání programu:
scan -pu <port-ranges> -pt <port-ranges> -i interface [<domain-name> | <IP-address>]
kde:
-pt, pu port-ranges - skenované tcp/udp porty, povolený zápis např. -pt 22 nebo -pu 1-65535 nebo -pt 22,23,24
-i interface – zařízení, přes které bude probíhat skenování
domain-name | ip address - doménové jméno, nebo IP adresa skenovaného stroje

Port skener

Pří komunikaci přes TCP/IP se využívá k adresaci jednotlivých stran komunikace dvou základních komponent, adresy a portu. Adresa specifikuje stanici jako celek, port určuje kanál, přes který bude probíhat komunikace. Každý port má svoje identifikační číslo (od 1 do 65536). Některým portům (v kombinaci s protokolem transportní vrstvy - TCP/UDP) jsou přiřazeny typické služby/protokoly, například port 80/tcp je pro HTTP, port 25/tcp pro SMTP apod (více lze nalézt na [6]). Z výše uvedeného plyne důsledek, že vzniká potřeba ověřování dostupných portů pro potencionální komunikaci. K tomuto účelu právě slouží programy nazvané port skenery. Ty ověřují dostupnost portů po portu a svých zjištěních podávají příslušné zprávy.

TCP SYN Scanning

Tato metoda je již ve zkratce popsaná v zadání. Skenovací program vyšle pouze paket s nastavenou značkou SYN (která znamená, začátek komunikace, viz [1]). Stanice se může zachovat třemi způsoby:
1. Odešle zpátky paket s nastavenými značkami SYN a ACK, které znamenají, že stanice chce pokračovat v TCP handshake-u, port lze označit za otevřený.
2. Odešle zpátky paket s nastavenými značkami ACK a RST, které znamenají, že stanice nechce pokračovat v TCP handshake-u, port lze označit za zavřený.
3. Stanice neodpoví nijak, problém může být v dostupnosti stanice jako celku a nebo v tom, že daný port filtrovaný (příchozí pakety jsou zahazovány).

Už tedy z prvního odeslaného/přijatého paketu lze zjistit otevřenost portu. To je také hlavní výhodou této metody, neposílá se pří ověřování tolik paketů a metoda je rychlejší oproti standratnímu TCP skenování (navazování kompletního handshake-u). Nevýhodou je obtížnější implementace a fakt, že některé operační systémy (FreeBSD, nové verze Windows, ...) standartně přes schránky nedovolují přijímat tzv.raw pakety (pakety obsahují i hlavičky protokolů) protokolu TCP, musí se pak vyuvat dalších knihoven (například pcap).

UDP ICMP port unreachable scanning

Tato metoda byla také ve zkratce popsána v zadání. UDP protokol nenavazuje žádné spojení, proto nelze ověřovat dostupnost portů podobným způsobem jako u TCP. Lze ale využít současně protokolu ICMP. Tento protokol je určen pro signalizaci dostupnosti sítě,stanic apod. Pří zaslání UDP paketu na nedostupný port operační systém stanice generuje ICMP paket typu 3 destination unreachable s kódem 3 port unreachable. Pokud tedy při zaslaní paketu na daný port porgram neobdrží jako odpověď paket ICMP popsaný výše, lze daný port považovat za otevřený. Není zde bohužel žádný mechanismus na zjištění, že daný port je filtrovaný. Problémem je, že ICMP ani UDP protokol nemají zabezpečeno doručování paketů cílové stanici, proto je třeba ověřit otevřenost portu vícekrát.

TCP skenování - implementace

Nejdříve je třeba vyplnit IP hlavičku. Ta je po celou dobu skenování naplněna stejnými údaji (ani jeden údaj není potřeba měnit). Dále se vytvoří soket(schránka) a všechny potřebné struktury pro odesílání paketů a inicializuje se knihovna libpcap pro odchytávání paketů na určitém síťovém zařízení. TCP hlavička se vyplní standartními hodnotami. Některé části TCP hlavičky je třeba měnit pro každý skenovaný port. Konrétně jsou to položky: zdrojový a cílový port, sekvenční číslo a kontrolní součet. Zdrojový port se nastavuje náhodně z rozsahu 32768-6100 (tento rozsah byl zvolen podle operačního systému Linux). Cílový port se nastavuje podle skenovaného portu, sekvenční číslo se vybírá náhodně (pomocí standartního pseudonáhodného generátoru čísel). Kontrolní součet se počítá podle RFC 1071 (viz [4]).

Dále je třeba vyřešit posílání a příjímání paketů. Paket se posílá pomocí standartních BSD soketů. Jak už bylo řečeno, BSD sokety v operačním systému FreeBSD nepříjímají raw sokety. Tento problém řeším pomocí knihovny libpcap (funkce **pcap_loop()**), která odchytává všechny pakety procházející přes určité zařízení. S výhodou se zde použije ještě filtr této knihovny (viz man pcap-filter). S pomocí něho program zachytne pouze TCP pakety, které mají odpovídající zdrojový i cílový port i IP adresu.

Po odeslání paketu tedy program začne zachytávat vhodné příchozí pakety. Pokud dorazí jeden z očekávaných (viz. popis skenovací metody), program vypíše stav portu(otevřený/zavřený). V případě, že dojde paket s příznaky ACK a SYN (druhá strana chce pokračovat v komunikaci – port je otevřený) dojde k automatickému odeslání druhé stanici paket s příznakem RTS. Tento paket je odesílán standartně jádrem operačního systému (jádro neví o tom, že byl odeslán nějaký paket z příznakem SYN), tímto způsobem je vyřešeno i ukončení komunikace s druhou stranou (není třeba aby RST paket posílal program). Pokud za vhodný časový okamžik žádný paket nepříjde(timeout), je odchytávání přerušeno, paket je poslán znovu a znovu se čeká na příchod paketu. Pokud nepříjde ani na podruhé port je označen za filtrovaný. Mechanismus timout-u je implementován pomocí signálů,

konkrétně signálu SIGALARM – funkce pcap_loop() vytvoří nekonečnou smyčku a ta je přerušena (funkcí pcap_breakloop()) při příchodu signálu SIGALARAM.

Ještě je třeba vyřešit jeden problém, na systémech BSD má hlavička linkové vrstvy pouze 4 byty, standartní etherentový rámec má 14 bytů. Tato situace je již vyřešená v knihovně lippcap, stačí testovat typ linkové vrstvy pomocí funkce **pcap_datalink()** a podle typu program určí velikost hlavičky linkové vrstvy. Tuto informaci potřebuje program znát proto, aby byl schopen analyzovat přijatý paket, knihovna pcpap totiž pracuje na linkové vrstvě (součástí paketu je tedy i hlavička linkové vrstvy).

UDP skenování - implementace

Mechanismus je velice podobný jako u TCP skenování, vytvoří se IP hlavička, struktura soket pro odesílání a všechny ostatní potřebné struktury pro odesílání paketů. Ovšem, ICMP pakety lze přijímat za pomocí BSD schránek, proto můžeme použít pro příjem ICMP paketů BSD schránky a pro implementaci timeout-u například funkci **select()**. Při implementaci byla tato možnost prověřena a otestována – je zcela funkční. Ale i zde jsem se rozhodl použít pro zpracování odpovědi knihovnu libpcap. Použití libpcap má dvě základní výhody, za prvé má program kontrolu nad všemi přichozími pakety a nespoléhá tedy na to, které pakety dostane od jádra operačního sytému. A za druhé lze opět s výhodou použít libpcap filtr. Jelikož je v tomto případě analýza paketu jednoduchá, v programu se vytvoří takový filtr, který omezí pakety přijímané aplikací pouze na takové, které odešle druhá stanice v případě nedostupnosti daného portu (viz popis metody UDP skenování). Potom již není potřeba provádět analýzu, ale při obdržení paketu se daný port prohlásí za nedostupný.

Problémem u UDP i ICMP je, že ani jeden protokol nemá implementován žádný mechanismus na spolehlivé doručování paketů. Může se tedy stát, že paket se po cestě sítí ztratí (je zahozen). Dalším problémem je, že jádra operačních systému většinou omezují odesílání stavových ICMP paketů (z důvodu nadměrného provozu na síti). Oba tyto problémy lze částečně řešit opakováním testů. Tedy v případě, že nepříjde paket indikující nedostupnost portu, ještě několikrát se odešle UDP paket na daný port, než se port prohlásí za otevřený. Timeut i počet opakování testů jsou nastaveny na konstantní hodnoty.

Timeout lze implementovat také pomocí knihovny libpcap bez použití signálů. Klíčové je použití kombinací funkcí **pcap_open_live()**, kde lze nastavit timeout a funkce pcap_dispatch(), která, pokud za daný časový interval neobdrží program žádný paket, skončí a vrátí počet přijatých paketů. Tento mechanismus se tedy u UDP skenování použije. (tento princip lze samozřejmě použít u TCP skenování, ale v rámci projektu jsem si chtěl vyzkoušet různé formy zpracování, proto je u každé metody použitý jiný princip).

Několik poznámek k implementaci

- Kdyby to někdo nepoznal - je to napsané v C++
- Program by měl fungovat na Linux a BSD strojích
- hlavičky jsou pouze little endian architekturu
- Pro kompilaci použijte příslušný makefile

Competition

Na závěr mám pro vás malou competion - v implementaci je jedna principiální chyba - program fugnuje tak jak má, ale jen díky jisté benevolenci síťových prvků. Zkuste přijít na to, kde je problém - svoje poznání můžete psát do komentů. Pro ty co se mnou studovali v ročníku a vědí v čem je háček, neprozrazujte, nechte ostatní potrápit. Kdo na to příjde první, tomu koupím při nejbližší příležitosti pivo.

Použitá literatura

[1] RFC 793 - Transmission Control Protocol
[2] RFC 791 - Internet Protocol
[3] RFC 768 - User Datagram Protocol
[4] RFC 1071 – Computing the internet checksum
[5] RFC 792 – Internet control message protocol
[6] http://www.iana.org/assignments/port-numbers [4]

URL článku: https://www.security-portal.cz/clanky/jednoduch%C3%BD-tcpudp-scanner-v-c

Odkazy:
[1] https://www.security-portal.cz/users/czokl
[2] https://www.security-portal.cz/category/tagy/networks-protocols
[3] https://www.security-portal.cz/category/tagy/programming
[4] http://www.iana.org/assignments/port-numbers