Počítačové komunikace a sítě

Sniffer paketů

3. května 2020

Alexander Polok (xpolok03)

Obsah

1 Úvod

Jak již název napovídá, našim úkolem bylo navrhnout a implementovat jednoduchý síťový analyzátor, dále jako **sniffer**, v jazyce C/C++/C#, který bude schopný na určitém síťovém rozhraním zachytávat a filtrovat pakety. Projekt jsem se rozhodl řešit v jazyce C++, z důvodu možnosti využití již implementovaných datových typů jako string, map, apod. a také možnosti snadnějšího odchytávání výjimek.

2 Co je to sniffer?

Analyzátor paketů, také známý jako síťový analyzátor, analyzátor protokolu nebo paketový sniffer, je počítačový program nebo kus počítačového hardwaru, pomocí kterého je možné zachytávat a zaznamenávat komunikaci v počítačové síti nebo v její části. V okamžiku, kdy datové proudy tečou přes síť, tak analyzátor paketů zachycuje každý paket. Ten je poté v případě potřeby dekódován na surová data paketu, která ukazují hodnoty různých polí v paketu. Dále je pak analyzován jeho obsah podle příslušné RFC nebo jiných specifikací.[1]

3 Implementace

Při návrhu snifferu jsem se inspiroval tutoriálem dostupným zde. Aplikaci jsem tedy rozdělil do funkce, kde dochází k načtení všech argumentů programu, následně funkce pro inicializaci všech potřebných struktur, "callback"funkce pro zpracování odchyceného paketu a funkce pro tisk paketu. Pakety jsou tisknuty dokud nedojde k výpisu požadovaného množství paketů.

3.1 Zpracování argumentů

Samotnou implementaci **snifferu** jsem začal zpracováním argumentů. Argumenty jsou zpracovávány ve funkci `getSettings`. Nejprve dochází k inicializaci struktury **programArguments** implicitními hodnotami. Poté jsou ve smyčce načteny všechny argumenty a případně jsou zpracovány jejich hodnoty. Aplikace podporuje jak krátké, tak dlouhé možnosti parametrů. Parametry popsané v zadání jsem rozšířil o vlastní parametry:

- `-ipv4` pro odposlouchávání pouze Ipv4 paketů
- `-ipv6` pro odposlouchávání pouze Ipv6 paketů
- `-multiple_ports ports_comma` pro možnost filtrování vícero portů

Kde `ports_comma` je řetězec síťových portů oddělených ','. Jak v rozšíření, tak v základní verzi jako síťový port uvažuji číslo (0 až 65535).

V případě nezadání nebo zadaní obou možností `-t -u` nebo jejich delších variant dochází k odposlouchaní jak **tcp** tak **udp** paketů.

V případě nezadání rozhraní je vypsán seznam aktivních rozhraní a program je ukončen s návratovým kódem **0**, v případě neexistujícího rozhraní je program ukončen s návratovým kódem **2**.

V případě zadání nevalidního čísla portu nebo počtu odchytávaných paketů je program ukončen s návratovým kódem **1**.

V případě použití neznámých argumentů je program ukončen s návratovým kódem **0**.

3.2 Struktury a jejich inicializace

Pro definici všech potřebných struktur jsem si vytvořil hlavičkový soubor `packetStructures.h`, kde jsou definované struktury pro Ipv4, Ipv6, Udp a Tcp. Pro tvorbu těchto hlaviček jsem se inspiroval příslušnými RFC. Byl zde ale problém, jak reprezentovat některé hodnoty, které nejsou velikosti běžné používaných datových typů v jazyce C++. Rozhodl jsem se některé hodnoty spojit do jedné položky a následně je vyfiltrovat

pomocí bitového součinu a příslušného posunu. Struktura pro **Ethernet** hlavičku zde nebyla nutná, jelikož data přenášená v této hlavičce nebylo nikde nutné použít, a proto jsem ji vynechal a provedl pouze její přeskočení. Její velikost je statická a je taktéž uvedena v hlavičkovém souboru `packetStructures.h`. Pro případné zastavení cyklu a uchování FQDN záznamů jsem si vytvořil strukturu `Configuration`, obsahující počet paketů, které mají být vypsány na standardní výstup, ukazatel na vytvořený "pcap_handle"a kontejner `dnsCache` pro uchovávání příslušných záznamů o Ip adrese a její doménovém jménu.

3.3 Otevření rozhraní pro síťovou analýzu

Po načtení všech argumentů a inicializaci všech struktur dochází k otevření rozhraní pro zachycení paketů. Nejdříve dochází k kontrole, zda se jedná o validní rozhraní a zda je možné ho vůbec otevřít pro zachycování paketů. Následně je inicializován filtr parametry načtenými z argumentů. V případě, že v jednom z předchozích kroků dojde k chybě, je program ukončen s návratovým kódem **2**. Následně dochází již ke spuštění opravdové analýzy.

Dochází k zavolání funkce `pcap_loop` s počtem paketů -1 (pro případ, že by pakety mohly být vadné a bylo by nutné odchytit více paketů), která v případě zachycení paketu, zavolá funkci `packetHandler`, kde dochází k zpracování příslušného paketu a inkrementování počtu zpracovaných paketů. V případě zpracování požadovaného počtu paketů je nekonečná smyčka ukončena funkcí `pcap_breakloop`.

3.4 Zpracování paketu

Za zpracování příslušných paketů zodpovídá funkce `packetHandler`. Nejprve zde dochází k inicializací předem definovaných struktur, je zde taktéž využitá statická proměnná pro počítání úspěšně zobrazených paketů. Následně dochází k naparsování příchozího paketu do Ipv4 struktury a je ověřeno, zda se opravdu jedná o Ipv4 paket, v opačném případě je ověřeno, zda se jedná o Ipv6 paket a data jsou přeparsované do příslušné struktury. V případě, že ani jedna z výše uvedených podmínek nebyla splněna, je parsování paketu ukončeno, paket není vypsán a počítadlo není inkrementováno.

Následně pomocí funkce `ipToDomainName` je přeložena Ip adresa na doménové jméno (v případě neúspěchu je vrácena Ip adresa) a z hlavičky je vyparsovaný "timestamp"pomocí funkce `unixTimeStampConverter`. FQDN záznam je uložený do kontejneru `dnsCache` a v případě dotazu na stejnou Ip adresu již nedochází k dotazování na FQDN a je zabráněno zbytečnému cyklení v případě "loopbacku".

Dalším krokem je vyparsování **UDP** nebo **TCP** hlavičky z paketu. Zde dochází k pomocnému výpočtu pro určení hranice mezi hlavičkou a daty. Poté již dochází k samotnému tisku paketů pomocí funkce `parsePacket`.

3.5 Tisk paketu

Funkci `parsePacket` jsou předány již všechny potřebné informace a nejprve dochází k výpisu nadpisu (čas, Ip adresy / doménové jména, porty). Poté již dochází k výpisu dat. Rozhodl jsem se pro výpis hlaviček a dat odděleně (pomocí volného řádku mezi těmito položkami) a automatické zarovnání výpisu do velikosti 16 hodnot na řádek. O samostatný výpis se stará procedura `printData`, kde nejdříve dochází k výpočtu nutných řádku pro výpis.

Následně jsou vypsány postupně řádky a to tak, že nejdříve je vypsán "offset"spolu se znakem ':', mezera, výpis 8 hexadecimálních hodnot oddělených mezerou, mezera, dalších 8 hexadecimálních hodnot, mezera, 8 příslušných tisknutelných znaků (v případě netisknutelných znaků je vypsán znak: '.'), mezera a zbylých 8 znaků.

Příslušné řádky jsou odděleny mezerou a v případě chybějících hodnot doplněny mezerami.

4 Testování

Funkcionalitu aplikace jsem ověřoval jak na vlastním operačním systému **Ubuntu 19.10**, tak na přiloženém virtuálním stroji. Zpracování paketů jsem porovnával s výstupem programu **Wireshark**, který se dá považovat za referenční.

Nejprve jsem ověřoval funkcionalitu správného filtrování a tisku Udp a Tcp paketů. Zde jsem postupně odchytával příslušné množství paketů tvořených běžícími aplikacemi na počítači a porovnával výstupy vlastní aplikace a Wiresharku pro příslušné vstupní filtry. Následně jsem se pokoušel o odchytávaní paketů vyvolaných explicitně v příkazové řádce. Zde bylo velmi výhodné použití `netcatu` pro komunikaci na rozhraní **lo**.

Nejprve jsem tedy vytvořil spojení pomocí příkazů `nc -l 2399` a `nc -l 2399`. Zvolil jsem tento port pro snadnější odlišení paketů. V tomto případě byly pakety odesílaný pomocí Tcp, jak se dalo očekávat. Výsledky byly opět totožné s výstupem aplikace Wireshark. Následně jsem se pokusil o komunikaci pomocí Udp a to pomocí příkazů `nc -u -l 2399` a `nc -u localhost 2399`, výsledky byly opět totožné.

Následně jsem ještě ověřil, že dochází ke správnému filtrování Ipv4 a Ipv6 paketů. Toto chování jsem ověřoval pomocí příkazu `curl -g -6 "http://[::1]:80/"` pro explicitní vytvoření Ipv6 paketů, jelikož jejich aktivita na mém zařízení byla velmi nízká.

Ještě jsem ověřil, zda aplikace nezkolabuje na velmi vysokém počtu paketů, avšak k tomu nedošlo. Nakonec jsem ověřil zda vstupní argumenty jsou správně zpracovány a zda nedochází k únikům alokované paměti, které by mohly ohrozit běh programu pomocí nástroje **valgrind**.

5 Rozšíření

Jako rozšíření nad rámec zadání bych považoval rozšířené možnosti filtrování paketů popsané výše, jakožto i uchovávání FQDN záznamu, poslouchání na více portech zároveň a podporu Ipv6 paketů, u které jsem si ze zadání nebyl zcela jistý, zda bylo nutné ji implementovat.

Reference

[1] Wikipedie. Analyzátor paketů — wikipedie: Otevřená encyklopedie, 2018.