Dokumentace k projektu do předmětu ISA

Programování síťové služby

Analyzátor paketů

Autor: Jiří Peška
Login: xpeska05
Datum: 18. listopadu 2017

Obsah

1 Úvod

Cílem projektu je vytvořit off-line analyzátor paketů, který dokáže vyčíst data z hlaviček podporovaných protokolů a následně je vypsat nebo je dále zpracovat podle argumentů programu.

2 Spuštění programu

Program byl po celou dobu vývoje překládán pomocí příkazu:

g++ -std=c++14 –Wall –Wextra –Weffc++ isashark.cpp –o isashark –lpcap

který byl umístěn uvnitř Makefile.

Program byl testován na serveru merlin.fit.vutbr.cz a na systému kali-linux.

Usage:

isashark [-h] [-a aggr-key] [-s sort-key] [-l limit] [-f filter-expression] file ...

-h Vypíše nápovědu a ukončí program.

-a <aggr-key> Zapnutí agregace dle klíče <aggr-key>, což může být srcmac, dstmac, srcip, dstip, srcport, dstport.

-s <sort-key> Zapnutí řazení podle klíče <sort-key>, což může být packets (počet paketů) nebo bytes (počet bytů). Řadit lze agregované tak i neagregované položky. Ve druhém případě je klíč packets bez efektu, protože všechny položky obsahují pouze jeden paket. Řadí se vždy sestupně.

-l <limit> Nezáporné celé číslo v desítkové soustavě udávající limit počtu vypsaných dat.

-f <filter-expression> Program zpracuje pouze pakety, které vyhovují filtru danému řetězcem filter-expression.

file Cesta k souboru ve formátu pcap. Možné je zadat jeden a více souborů.

3 Implementace

Program se skládá ze dvou souborů – isashark.cpp, isashark.h a překládán pomocí Makefile.

3.1 Argumenty

3.1.1 Limit

Limit udává maximální počet vypsaných řádků.

3.1.2 Sort

Pakety nejsou hned po zpracování vypsány, ale jsou uloženy do vektoru, který po zpracování posledního paketu ze vstupních souborů podle zadaného řadicího klíče seřadím. Pokud není použita agregace, tak v případě klíče „packets“ se pakety vypíšou v tom pořadí, ve kterém byly zpracovávány. O řazení se stará funkce std::sort. Pakety jsou seřazeny sestupně.

3.1.3 Agregace dat

Pro účely agregace je vytvořena mapa (std::map), kam podle zadaného agregačního klíče vkládám nový klíč jakožto agregační klíč, který vyčtu z paketu, a jako hodnotu pro daný klíč počítám sumu a počet paketů. Po zpracování a započítání posledního paketu vypíšu mapu jako trojici klíč-počet-velikost.

Pokud je zároveň zadán přepínač sort, tak tato mapa bude ještě před vypsáním patřičně seřazena podle klíče, který byl přepínači zadán a vypsána hned poté.

3.1.4 Filtr

Využívám filtr poskytnutý knihovnou libpcap. Program zpracovává pouze ty pakety, které projdou přes filtr.

3.2 Analýza

Při spuštění programu se dějí následující úkony. Zkontrolují se argumenty programu a jejich formát. Poté načtu vstupní soubory do vektoru, ze kterého je následně postupně beru a načítám z nich v cyklu pakety. Aktuální paket je podroben analýze. Všechna vypreparovaná data ukládám do třídy sloužící jako „container" pro výsledný paket. Při získávání dat musím používat funkce *inet_ntoa()*, *inet_ntohs()*, kvůli rozdílným architekturám typu big-endian a little-endian. Postupně se analyzuje vrstva L2, kde se ještě kontroluje příslušnost k IEEE, dále se čte IP hlavička, která přísluší k L3 vrstvě, kde můžou nastat 2 případy. V prvním je to IPv4, kde je třeba zkontrolovat, jestli aktuálně zpracovávaný paket není fragment a ve druhém případě IPv6, kde musíme přeskočit rozšiřující hlavičky. Pak už lze vyčíst protokol L4 vrstvy – TCP, UDP nebo ICMP. Tam se dostanu celkovým offsetem přičteným k paketu *(paket + SIZE_ETHERNET + IEEE_offset + size_ip).*

V případě, že narazím na protokol, který nemám zpracovávat, vypíšu na *std::cerr* chybu s číslem protokolu a pokračuju se zpracováním dalšího paketu. Číslovány jsou ty pakety, které jdou na výstup, takže i když se nějaký paket zahodí, sekvence čísel bude neporušena. Pokud je paket zahozen, nebude zaindexován do agregačního záznamu ani v případě, že by to data umožňovala. Jakmile je zahozen, nezapočítává se nikam.

Po zpracování L4 protokolu, container s vyčtenými daty buď vypíšu - v případě spuštění programu bez agregace nebo sortu, nebo ho uložím do vektoru pro pozdější zpracování.

3.3 Fragmentace IPv4

Při zpracování L3 (IP) vrstvy se podívám, jestli je u paketu povolena fragmentace, což zjistím díky flagu *DF == 0*. Pokud je povolana, tak zjistím informace o fragmentu a zařadím ho do mapy podle fragment_id, protokolu vyšší vrstvy, zdrojové a cílové ip adresy, podle kterých se určí, ke kterému paketu fragment přísluší.

Data z L2 a L3 vrstvy jsou pro fragmenty daného paketu stejné, takže z jednoho z nich si je ponechám, a čekám, dokud mi nedorazí poslední fragment paketu. Až se tak stane, tak fragmenty seřadím podle sekvenčního čísla, vložím do bufferu a předám funkcím, které zpracují vrstvu L4. Konečně je paket předán podle argumentů buď výstupu, nebo vložen do vektoru či zahrnut do agregace.

Řazení podle sekvenčního čísla:
std::sort(fragPack->fragments.begin(); fragPack->fragments.end(); [](auto&left, auto &right) {return left.first < right.first;});

3.4 Rozšiřující hlavičky IPv6

„IPv6 extended headers“ obsahují délku hlavičky a číslo vyššího protokolu na stejném místě, takže v cyklu se ptám, jestli je na daném místě číslo následujícího protokolu, nebo číslo další hlavičky.

Z důvodu příslušnosti IPv6 Extended header ESP k ipsec, moje implementace nepočítá s výskytem této rozšiřující hlavičky.

Minimální délka hlavičky je 8B, protože je dáno dle RFC, že má mít délku v násobcích osmi, takže offset vypočítám jako 8*(len + 1).

3.5 IEEE – tag

Když v ethernetové hlavičce v místě len indikuji IEEE místo čísla protokolu vyšší vrstvy, tak k offsetu přičtu 4B, protože IEEE tag je dlouhý právě 4B, a znovu se podívám, jestli je na novém místě číslo protokolu nebo další IEEE příslušnost. Tak cyklím, dokud nenajdu číslo následujícího protokolu. K vyčtení dat z IEEE používám svoji strukturu, do které zarovnám data z paketu s patřičným offsetem.

Celkový offset, nutný k přeskočení IEEE tagů vypočítám jako *pocetIEEE*4*.

4 Omezení a rozšíření

Defragmentace ipv4 – zvolil jsem vlastní implementaci zpracování fragmentace, která se ukázala jako nevyhovující v případě překrývání fragmentů a u dvou posledních hodnot u TCP. Hlavička UDP je v pořádku.

Žádná další omezení nejsou a vše ostatní funguje korektně.

5 Příklady použití a popis

./isashark –h
Vypíše nápovědu a ukončí program.

./isashark -a dstip inputfile.pcap
Agregace paketů podle zadaného klíče – v tomto případě dstip.

./isashark -l 20 inputfile.pcap
Vypíše maximálně 20 paketů.

./isashark -f "src host 2001:db8::1" inputfile.pcap
Zpracovává pouze pakety, které vyhovují filtru.

./isashark –a dstmac –s packets input1.pcap input2.pcap
Dva vstupní soubory k analýze a zapnuta agregace s následným seřazením podle počtu paketů u jednotlivých agregovaných záznamů.

6 Závěr

Navrhli jsme a implementovali analyzátor paketů pro systém LINUX, který se dá popsat jako „ultra-light“ verze programu Wireshark. Isashark dokáže analyzovat jednotlivé hlavičky vybraných paketů, vyčíst z nich data a s použitím argumentů při spuštění nabízí rozšířenou funkcionalitu jako třídění paketů podle velikosti nebo jejich agregaci podle zadaných klíčů apod.

K implementaci bylo potřeba nastudovat si hlavičky vybraných protokolů a jejich chování, RFC a seznámit se s knihovnou libpcap, která byla při implementaci velmi užitečná.

K bližšímu pochopení posloužil velice dobře program Wireshark, kde se lze podívat na jednotlivé bity a byty paketů a jejich tvar.

7 Literatura

[1] http://yuba.stanford.edu/~casado/pcap/section2.html
[2] https://linux.die.net/man/7/pcap-filter
[3] https://www.freebsd.org/cgi/man.cgi?query=ip6&sektion=4&manpath=FreeBSD+9.1-RELEASE
[4] https://en.wikipedia.org/wiki/IPv4
[5] https://www.ietf.org/rfc/rfc2292.txt
[6] http://beej.us/guide/bgnet/output/html/multipage/inet_ntopman.html
[7] https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
[8] https://tools.ietf.org/html/rfc4884
[9] https://tools.ietf.org/html/rfc792
[10] http://help.fortinet.com/fos50hlp/54/Content/FortiOS/fortigate-firewall-52/Concepts/ICMPv6%20Types%20and%20Codes.htm