Programování síťové služby

Analyzátor paketů

Ondrej Svoreň - xsvore01

2017/2018

Obsah:

1 Všeobecná špecifikácia

Analyzátor paketov je konzolová aplikácia slúžiaca na offline analýzu sieťovej komunikácie. Aplikácia podporuje nasledovné protokoly používáné v modely TCP/IP (viď Obrázok 1):

L2 - vrstva sieťového rozhrania:

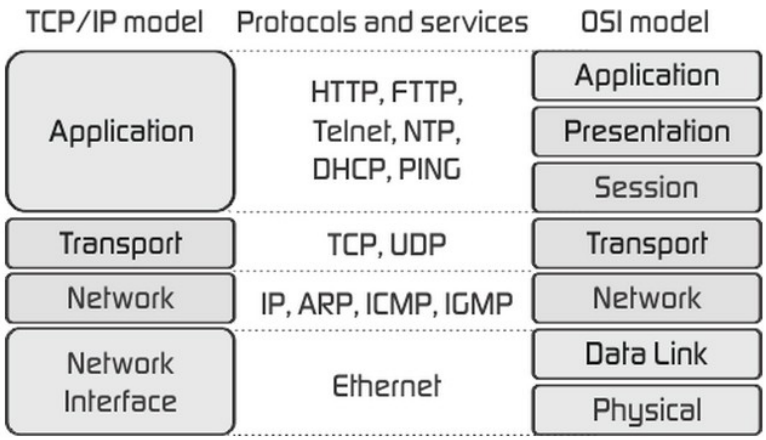
- Ethernet
- IEEE 802.1Q, IEEE 802.1ad

L3 - sieťová vrstva:

- IPv4
- IPv6
- ICMPv4
- ICMPv6

L4 - transportná vrstva:

- TCP
- UDP



Obrázok 1: Porovnanie modelov TCP/IP a ISO/OSI[1]

Aplikácie s podobným využitím určené predovšetkým na ešte podrobnejšiu analýzu prenosu po sieti[2] (pre ďalšie využitie viď "1.1 Využitie programu") sú napríklad desktopová aplikácia Wireshark alebo terminálová utilita *tcpdump*.

1.1 Využitie programu

- analýza problémov vzniknutých pri sieťovej komunukácií
- detekcia pokusov o preniknutie do siete resp. narušenie sieťového prenosu
- detekcia zneužitia siete treťou stranou
- získavanie informácí pre efektívny prenos po sieti
- monitoring siete a kontrola bezpečnosti
- overenie zmien a pohybov sieťového prenosu

Okrem vyššie uvedených protokolov aplikácia ďalej analyzuje a rieši problém IPv4 fragmentácie paketu a rozširujúce hlavičky IPv6, ktorých implementácia je podrobnejšie popísaná v časti "2 Popis implementácie".

2 Popis implementácie

Implementácia je založená na znalostiach získaných z voľne dostupných informačných zdrojov (viď Zdroje). Použité sú knižnice jazyka C a C++ a zdrojový kód je rozdelený do dvoch zdrojových súborov, súboru isashark.cpp a hlavičkového súboru isashark.h.

2.1 Podrobnosti k implementácií

Vo funkcií *main()* nachádzajúcej sa v zdrojovom súbore isashark.cpp nastáva spracovanie argumentov príkazového riadku (viď "2.2 Implementácia funkcionality udávanej vstupnými argumentami programu"). Po rozparsovaní jednotlivých argumentov dochádza k otvoreniu vstupného súboru resp. vstupných súborov na čítanie a k riadeniu analýzy jednotlivých paketov v súbore. To sa deje pomocou funkcí z hlavičkového súboru *pcap.h*, napríklad funkcia *pcap_next()* vracia ukazateľ typu *u_char* na dáta nesúce informácie o aktuálne analyzovanom pakete.

Každý paket ďalej prechádza analýzou podporovaných protokolov na každej vrstve, pričom dochádza k identifikácií požadovaných údajov, ktoré majú byť použité vo výstupe programu. Pri tejto činnosti sú použité jednotlivé datové štruktúry z hlavičkových súborov *netinet/** podľa konkrétneho protokolu (štruktúry *ether_header, ip, ip6_hdr, tcphdr, udphdr a iné*). Napríklad typ protokolu na vrstve sieťového rozhrania (*L2)* udáva údaj *ethertype*[3] v ethernetovom rámci.

Pre každú vrstvu je implementovaná funkcia (nazvaná podľa vrstvy napr. *l3_protocol()*), ktorá podľa určitých pravidiel (obvykle hodnota uložená v hlavičke protokolu na predchádzajúcej vrstve[4]) rozlišuje použitý protokol na aktuálnej vrstve. Z hlavičky daného protokolu, ktorá je reprezentovaná niektorou z vyššie uvedených dátových štruktúr, sú zisťované všetky informácie potrebné pre výstup programu (napr. MAC adresa, IP adresa, TCP a UDP číslo portu, ICMP typ a kód). Ako príklad získavania informácí z hlavičiek uvediem uloženie adries typu MAC a IP, pri ktorom používam typ kontajnera *std::string*, čo mi uľahčuje prácu ako pri vypisovaní, tak aj pri kontrole formátu, porovnávaní a pod. Pri získavaní údajov z týchto dátových štruktúr je vždy potrebné k momentálnemu indexu v poli *u_char* pripočítať offset v podobe veľkosti hlavičky aktuálne analyzovanej vrstvy, na ktorom sa má nachádzať hlavička protokolu z nasledujúcej vrstvy.

Každý paket je modelovaný ako objekt triedy *Packet.* Atribúty tejto triedy reprezentujú jednotlivé informácie o pakete resp. protokoloch a zároveň sú to zdroje výstupu celej analýzy, napríklad zdrojové a cieľové IP a MAC adresy, čísla portov atď. Metódy tejto triedy slúžia na priradenie získaných informácí o pakete do objektu, ktorý daný paket reprezentuje. Sú pomenované podľa vrstvy, ktorej informácie zoskupujú (napr. *set_l3_layer()).*

Po získaní všetkých potrebných informácií o pakete je paket ako objekt triedy *Packet* uložený do kontajnera *std::vector* s názvom *packets*, čo sa mi zdá ako dobré riešenie pre uloženie a kumulovanie všetkých paketov či už z jedného alebo viacerých súborov. Výstup je realizovaný iterovaním v tomto vektore a vypisovaním konkrétnych informácií o každom pakete pomocou metódy *output()* triedy *Packet*, ktorá formátuje jednotlivé atribúty paketu do požadovaného výstupného formátu a zároveň ich vypisuje na štandardný výstup.

2.2 Implementácia funkcionality udávanej vstupnými argumentami programu

Nasledujúca časť rozoberá spracovanie každého vstupného argumentu príkazového riadku, ktoré je implementované za pomoci POSIX funkcie *getopt()*. V závislosti na použitých vstupných argumentoch sa vyhodnocuje, do akej miery resp. akým spôsobom má byť vstupný súbor analyzovaný.

Limit *[-l limit]*:

Po spracovaní všetkých paketov zo vstupných súborov sa výstup riadi hodnotou argumentu *limit* tak, že na výstupe sa zobrazí len požadovaný počet paketov definovaný hodnotou tohto argumentu. Obmedzenie nastáva v počte opakovaní cyklu, pričom v každom cykle sa vypíšu informácie o práve jednom pakete.

Agregácia *[-a aggr-key]*:

Na základe agregačného kľúča program zoskupuje pakety s rovnakou hodnotou tohoto kľúča, ktorý predstavuje niektorý z atribútov objektu paketu (viď “4.1 Význam vstupných argumentov”). Tieto pakety potom ukladá ako objekty triedy *AggregatedPackets* do vektora *aggr_packets*, nakoľko použitie tohto typu kontajnera uľahčuje vyhľadávanie atribútov s rovnakou hodnotou. Výpis sa realizuje iterovaním vo vektore *aggr_packets* a volaním metódy *print_aggr()* triedy *AggregatedPackets*, ktorá okrem výpisu na *stdout* realizuje aj formátovanie výstupu. V prípade, že hodnota agregačného kľúča paketu je nedefinovaná (napríklad srcport a dstport pri ICMP), nasleduje výpis upozornenia (viď Výnimky a chybové hlášky).

Radenie *[-s sort-key]*:

V prípade radenia podľa veľkosti sa použitím C++ funkcie *sort()* a mnou implementovanej funkcie *sortByBytes()*, ktorá je použitá ako jej parameter zoradia pakety uložené vo vektore *packets* podľa atribútu objektu určujúceho veľkosť daného paketu na základe porovnania týchto atribútov vo funkcí *sortByBytes()*. Na rovnakom princípe fungujú aj nasledujúce prípady. V prípade radenia podľa počtu agregovaných paketov musí byť pre efekt použitý aj vstupný argument príkazového riadku symbolizujúci agregáciu *[-a aggr-key]*. V tomto prípade sa zoraďujú agregované pakety uložené vo vektore *aggr_packets* použitím funkcie *sort()* v kombinácí s funkciou *sortByPackets()*. Agregovaný výstup sa radí podľa veľkosti kombináciou funkcí *sort()* a *sortByBytes_a()*.

Filtrácia *[-f filter-expression]*:

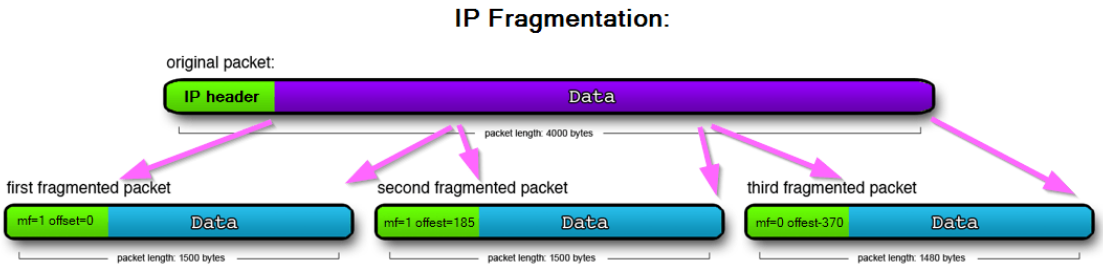
Filtrácia je implementovaná pomocou funkcí *pcap_compile* a *pcap_filter*, pričom reťazec *filter-expression* je použitý ako parameter funkcie *pcap_compile*.

2.3 Fragmentácia a IPv6 extended headers

V tejto časti je podrobnejšie popísaný spôsob implementácie defragmentácie fragmentovaného paketu a implementácia chovania programu v prípade, že paket obsahuje rozširujúce hlavičky protokolu IPv6.

2.3.1 Fragmentácia

Proces defragmentácie je implementovaný vo funkcií *fragmentation_reassembly()*, pričom základná myšlienka je založená na identifikácií toho, že je paket fragmentovaný (viď Obrázok 2), ďalej na získaní všetkých dát z jednotlivých fragmentov a ich usporiadaní v dátovom buffri. Fragmentáciu je možné identifikovať na základe údajov z IP hlavičky, konkrétne príznaku MF a offsetu. V prípade, že je hodnota príznaku MF alebo offsetu rôzna od nuly, volá sa funkcia *fragmentation_reassembly()*, ktorá pracuje s triedou *FragmentedPacket* obsahujúcou ako atribút okrem iného aj spomínaný *data_buffer()* a pomocou metódy tejto triedy s názvom *create_fragmented_packet()* sa inicializuje a vytvára nový objekt tejto triedy. Prichádzajúce fragmenty sú ďalej identifikované pomocou ip adresy zdroja a cieľa, čísla protokolu vyššej vrstvy (*L4*) a identifikátora (údaje z IPv4 hlavičky[5]). V prípade, že fragment obsahuje rovnaké hodnoty vyššie uvedených atríbútov, nasleduje kontrola validity na základe offsetu vo funkcí *hole_filler()*. Táto funkcia pracuje s vektorom *hole_descriptor_list()*, ktorý je jedným z atribútov objektu a ktorý obsahuje objekty triedy *Hole_Descriptor*. Objekty tohto typu nesú informáciu o veľkosti dát fragmentovaného paketu, ktoré chýbajú k tomu, aby bol daný paket defragmentovaný. Jednotlivé fragmenty dát sú na základe offsetu ukladané do dátového buffra pomocou funkcie *save_data()*. Ak sa podarí identifikovať a spojiť všetky fragmentované dáta, nastavuje sa v pakete atribút *is_reassembled* a je možné prejsť na analýzu paketu z pohľadu vyššej vrstvy. Postup ukladania atribútov objektu získaných analýzou na predchádzajúcich vrstvách sa pri fragmentácí nemení. Algoritmus je inšpirovaný algoritmom z RFC 815[6].



Obrázok 2: Znázornenie IP fragmentácie paketu [7]

2.3.2 IPv6 extended headers

V prípade protokolu IPv6[8] uvažujeme tzv. rozširujúce hlavičky. Ak paket obsahuje rozširujúce IPv6 hlavičky[9], vo funkcí *l4_protocol()* nastáva prechod do funkcie *extended_IPv6_header()*, ktorá cyklí v pakete na základe hodnoty premennej reprezentujúcej dĺžku rozširujúcej hlavičky získanú zo štruktúry *ip6_ext*. Podľa určitých pravidiel sa pomocou tejto hodnoty mení offset nasledujúcej hlavičky prirátavaný v každom cykle. Cyklus končí, ak nájde hodnotu reprezentujúcu podporovaný L4 protokol (TCP, UDP, ICMPv6) alebo keď dôjde na koniec paketu.

3 Výnimky a chybové hlášky

Táto časť opisuje chovanie programu v prípade nežiadúcich vstupných argumentov a nesprávnej manipulácie s programom.

Všetky chybové hlásenia sú vypisované na štandardný chybový výstup, pričom návratové hodnoty programu sú v prípade chyby rôzne od nuly. Program detekuje nasledujúce chyby a im príslušné chybové výstupy a návratové hodnoty:

Chyba pri otvorení vstupného súboru:

Ak sa nepodarí otvoriť vstupný súbor zadávaný ako vstupný argument programu na príkazovom riadku (pomocou funkcie *pcap_open_offline())*, program končí s výpisom *"Can't open file [filename] for reading!"* a vracia **hodnotu 3**.

Chyba pri detekcí nepodporovaného protokolu:

V prípade, že analyzovaný paket obsahuje nepodporovaný protokol, je na mieste určenom pre výpis údajov získaných analýzou paketu vytlačené chybové hlásenie "Unsupported protocol", pričom program pokračuje v analýze nasledujúceho paketu.

Chyba pri zadaní nesprávnej hodnoty vstupných argumentov:

Aplikácia končí s chybou a vracia **hodnotu 1** v prípade chybného zadania hodnoty vstupných argumentov (viď "4.1 Význam vstupných argumentov"), pričom na štandardný chybový výstup vypíše reťazec *"Wrong argument value [option]!"*(option je konkrétny typ argumentu).

V prípade zadania nepodporovaného alebo nesprávne zadaného filtra pri použití *[-f filter expression]* vracia program **hodnotu 2** a na stderr vypíše hlásenie *pcap_compile() failed* resp. *pcap_filter() failed.*

Výnimky a ICMP správy:

Pri zadaní *srcport* alebo *dstport* ako hodnoty argumentu pre agregáciu môže dôjsť k výnimke v prípade, že paket neobsahuje žiadnu hodnotu pre tento typ argumentu (protokol ICMP). V tomto prípade sa paket neagreguje s ostatnými paketmi a program vypisuje na stderr upozornenie v znení "*Packet doesn't contain any value of aggregation key.*". Program v tomto prípade vracia hodnotu nula, nakoľko nejde o fatálnu chybu.

Jednotlivé ICMP správy sú vypisované na základe typu a kódu získaného z hlavičiek ICMP protokolov (použité sú dátové štruktúry *icmp* a *icmp6_hdr)* a znením zodpovedajú RFC 792[10] pre ICMPv4 a RFC 4443[11] pre ICMPv6.

4 Použitie

Použitie a formát vstupu:

$./isashark [-h] [-a aggr-key] [-s sort-key] [-l limit] [-f filter-expression] file(1) file(2) … file(n)

4.1 Význam vstupných argumentov

[-h] – v prípade použitia vypíše nápovedu resp. návod na použitie

[-a aggr-key] – v prípade použitia tohto parametra sú pakety zo súboru *file* agregované na základe agregačného kľúča *aggr-key*, ktorý môže nadobúdať hodnôt:

- *srcip* – zdrojová IP adresa
- *dstip* – cieľová IP adresa
- *srcmac* – zdrojová MAC adresa
- *dstmac* – cieľová MAC adresa
- *srcport* – zdrojový port
- *dstport* – cieľový port

[-s sort-key] – v prípade použitia je výstup radený na základe radiaceho kľúča *sort-key*, ktorý môže nadobúdať hodnôt:

- *packets* – podľa počtu agregovaných packetov (pri použití bez agregácie nemá efekt)
- *bytes* – výstup je radený podľa veľkosti (od najväčšieho po najmenší)

[-l limit] – v prípade použitia je výstup obmedzený na počet paketov reprezentovaný hodnotou *limit*-u (nezáporné celé číslo v desiatkovej sústave)

[-f filter-expression] – v prípade použitia je výstup filtrovaný na základe filtra vo forme reťazca *filter-expression*, ktorého formát musí zodpovedať špecifikácí

file(1) file(2) … file(n) – vstupné súbory, ich počet nie je obmedzený

5 Zdroje

[1] http://fiberbit.com.tw/wp-content/uploads/2013/12/TCP-IP-model-vs-OSI-model.png

[2] https://en.wikipedia.org/wiki/Packet_analyzer

[3] https://en.wikipedia.org/wiki/EtherType

[4] https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers

[5] https://tools.ietf.org/html/rfc791

[6] https://tools.ietf.org/html/rfc815

[7] https://cdn.plixer.com/wp-content/uploads/2014/07/frag1.png

[8] https://tools.ietf.org/html/rfc3513

[9] https://www.microsoftpressstore.com/articles/article.aspx?p=2225063&seqNum=4

[10] https://tools.ietf.org/html/rfc792

[11] https://tools.ietf.org/html/rfc4443