VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Síťové aplikace a správa sítí

Monitoring SSL spojení

2020/2021

Tomáš Ďuriš
(xduris05)

Banská Bystrica
12.10.2020

Obsah

TODO

1. Úvod

Cieľom tejto dokumentácie je popis projektu do predmetu *Síťové aplikace a správa sítí*. Zvolil som si zadanie *Monitoring SSL spojení,* a mojou úlohou bolo vytvoriť jednoduchý nástroj v jazyku C/C++, ktorý spracuje pcap súbor a zobrazí informácie o SSL spojení zo zadaného súboru alebo zo *"živého" odchytu na zadanom rozhraní*. Rozhodol som sa o riešenie v jazyku C obohatné o prvky jazyka C++ *(napr. std::string)*.

Dokumentácia popisuje spôsob implementácie, použité zdroje, použité knižnice a vlastné spôsoby riešenia jednotlivých problematík. Na záver som sa venoval krátkemu zhodnoteniu a prínosom projektu.

2. Uvedenie do problematiky

Ako som už spomínal projekt sa zaoberá spracovaním informácii o danom SSL spojení a ich výpisu na štandardný výstup. K úspešnej realizácii tohto projektu bolo potrebné si naštudovať, ako SSL spojenie funguje a akým štýlom je implementované. K tomuto mi pomohli RFC dokumenty (https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml) na pochopenie jednotlivých častí SSL spojenia a veľkosti jednotlivých položiek v hlavičkách daných štandardom. K samotnému naprogramovaniu aplikácie mi pomohla kniha Síťové aplikace a jejich architektura (Petr Matoušek), kde boli vysvetlené základy TCP spojenia alebo aj ako funguje knižnica pcap a základy jej použitia k odchytávaniu paketov či už na sieti alebo v súbore. Samozrejme tieto informácie bolo nutné kombinovať s oficálnou dokumentáciou knižnice libpcap.

Pri programovaní samotnej aplikácie bolo nutné porozumieť, že TLS/SSL spojenie je šifrované a je súčasťou TCP spojenia. TLS spojenie začína procesom nazvaným "handshake", pri ktorom si obe strany overia, či sú naozaj tým kým tvrdia. V praxi teda za začiatok spojenia môžme teda považovať výmenu paketov *Client Hello, Sever Hello, Change Cipher Spec,* v ktorých dôjde k výmene informácii o tom aká verzia TLS sa použije, aká metóda šifrovania sa použije a overí sa identita oboch koncou komunikácie.

Jednotlivé SSL pakety sa skladajú z hlavičky a dát *(*pokiaľ ide o paket typu *Application Data (23))*. Pre prístup k dátam by sme ich museli najprv dešifrovať, avšak toto v našom projekte nebolo potrebné, keďže nám stačili informácie z hlavičiek, ktoré šifrované nie sú. Konkrétne sa jednalo o údajoch o čase zachytenia paketu, IP adrese a porte klienta, IP adrese servera, SNI (meno požadovaného serveru) a počet bajtov v pakete. Zvyšné informácie ako čas trvania a počet paketov išli, jednoducho odvodiť z predošlých informácií.

3. Implementácia

Aplikácia *ssl-sniffing* je napísaná v jazyku C++, za využitia knižnice Libpcap. Aplikácia postupne vypisuje informácie o SSL spojeniach na štandardný výstup. Informácia o SSL spojení sa vypíše vždy ak bolo SSL spojenie zahájené a aj ukončené. Ako som spomínal za zahájené SSL spojenie považujem, pokiaľ dôjde k výmene prvotných informácii medzi klientom a serverom (*Handshake*) a za ukončené spojenie považujem ak v danom spojení nedošlo k prenosu paketu po dobu 5 sekúnd. Aplikácie umožňuje čítať pakety zo súboru *.pcap/.pcapng* ale aj odpočúvať pakety v reálnom čase na otvorenom rozhraní (k využitiu tejto funkcie je nutné spustiť program s oprávením správcu – príkaz *sudo*). V prípade neúspechu je aplikácia ukončená s návratovým kódom 1 (makro **EXIT_FAILURE**) a vypísaním chybovej hlášky.

3.1 Preklad a spustenie

K preloženiu aplikácie je potrebné v príkazovom riadku zadať požiadavok na preloženie (príkaz **make** prípadne **make build**), ktorý program preloží prostredníctvom pribaleného súboru *Makefile*. Výsledny súbor je určený a spustiteľný pre operačný systém Linux.

Po preložení projektu je možné aplikáciu spustiť prostredníctvom príkazového riadku po zadaní príkazu

```
./sslsniff [-r <file>] [-i interface]
```

Kde parameter [**-r** <file>] označuje, že pôjde o čítanie zo súboru a **<file>** označuje názov súboru (prípadne cestu k súboru). Parameter [**-i** interface] označuje, že pôjde o odposlúcanie na rozhraní **<interface>**. Spracovanie argumentov je bližšie popísané v sekcii 3.3 alebo v priloženom *README.md*.

3.1 Hierarchia súborov

Projekt sa skladá z následujúcich súborov:

- **sslsniff.cpp –** deklarácia štruktúry s SSL spojeniami, zkompilovania filtra pre odposluch, a odposluch na danom rozhraní alebo prípadné čítanie zo súbora.
- **arg_parser.cpp –** kontrola a spracovanie argumentov, uloženie si potrebných informácií do zdieľaných premenných, výpis nápovedy
- **arg_parser.h –** hlavičkový súbor, definícia zdieľaných premenných medzi súbormi a definícia funkcií.
- **process_packets.cpp –** jednotlivé spracovanie paketov, získanie potrebných informácií z hlavičiek, výpis informácii o jednotlivých SSL spojeniach

- **process_packets.h –** hlavičkový súbor, definícia funkcií, definícia makier pre konštantné hodnoty z hlavičiek, definícia štruktúry pre ip adresy a informácie o ssl spojení

3.3 Kontrola Argumentov

Kontrola uživateľom zadaných argumentov na vstupe prebieha ručne (t.j bez využitia napríklad knižnice *getopt*). V tele programu (súbor **sslsniff.cpp**) sa zavolá funkcia ***parse_args()***, ktorej sú prostredníctvom parametrov predané argumenty programu a ich počet. Všetky užívateľom zadané argumenty sú skontrolované cyklom for a aktuálny argument sa vždy spracuje. Pokiaľ sa jedná o platný argument (**-i** alebo **-r**) je uložená informácia, že tento argument sa vyskytol a ako ďalší sa očakáva buď názov súboru alebo názov rozhrania. Pri viacnásobnom zadaní rovnakých argumentov (obmedzené na maximálny počet argumentov <= 5) sa berie vždy posledne zadaná hodnota názvu súbora alebo rozhrania. Informácie o tom aký parameter bol zadaný je predaná naspäť telu programu prostredníctvom zdieľaných premenných typu *bool* **have_interface, have_file**. Informácia o mene súboru alebo rozhrania je predaná telu programu prostredníctvom zdieľaných premenných typu *string* **file, interface**.

Pokiaľ došlo k zadaniu nesprávnych parametrov, alebo k zadaniu ich nesprávneho počtu sa vypíše chybová hláška a stručná nápoveda. Pri zadaní aj parametru pre súbor, kde meno súboru je platné a parametru pre rozhranie, kde meno rozhrania je platné sa uprednostní zadané rozhranie a odposlúcha sa na ňom. Pri nezadaní žiadneho argumentu sa vypíše nápoveda programu.

Pokiaľ prebehne kontrola argumentov bez problému následuje otvorenie súboru pre čítanie alebo otvorenie rozhrania na odposluch. Po úspešom otvorení súboru alebo rozhrania je vykonaná kompilácia filtra pre filtrovanie paketov (Jedná sa iba o filter *tcp*, keďže protokol SSL je súčasťou protokolu TCP a teda môžme ignorovať pakety, ktoré tento protokol neobsahujú) a jeho následná aplikácia.

3.4 Spracovanie paketu

Hlavu programu tvorí funkcia knižinice libpcap – ***pcap_loop()***, ktorej predáme otvorený súbor alebo rozhranie, počet paketov, ktoré sa majú spracovať (v našom prípade pôjde o číslo **-1**, ktoré značí spracovanie po koniec súboru, prípadne nekonečné spracovávanie paketov na rozhraní), funkciu ***callback()***, ktorá sa stará o spracovanie jednotlivých paketov a ukazateľ na mapu všetkých ssl spojení **ssl_sessions**, kde som ako kľúč zvolil IP adresu klienta a port klienta (**client_ID**), keďže v jeden okamih nemôže byť nadviazaných viacero TCP spojení z jednej IP adresy a portu u klienta (u serveru toto samozrejme neplatí).

Vo funkcii **callback()** (súbor *process_packets.cpp*), sa pomocou ukazateľa na začiatok paketu a makra pre veľkosť ethernetovej hlavičky uloží ukazateľ na začiatok IP hlavičky, k získaniu ostatných informácii z IP hlavičky sa využije štruktúra **ip** z *<netinet/ip.h>*. Následne sa zistí či ide o IPv4 alebo IPv6 verziu, čomu sa aj prispôsobí

veľkosť hlavičky podľa noriem. Z IP hlavičky si pre potreby projektu uložíme údaje o cieľovej a zdrojovej IP adrese. Veľkosť IP hlavičky sa podobne ako pri ethernet hlavičky využije pre získanie ukazateľa na začiatok TCP hlavičky. K uloženiu potrebných informácií z TCP hlavičky využijeme štruktúru **tcphdr** z <netinet/tcp.h>, z týchto údajov si samostatne pre potreby projektu uložíme údaje o cieľovom a zdrojovom čísle portu.

Samotný paket je prejdený od úplného začiatku po koniec, pričom sa hľadá začiatok SSL hlavičky. Za SSL hlavičku považujem postupnosť bajtov v hexadecimálnej reprezentácii v tvare:

1. 1. bajt údajnej hlavičky musí byť rovný **0x14, 0x15, 0x16** alebo **0x17** na základe RFC štandardu, kde typ obsahu je 0x14 – *Change Cipher Spec,* 0x15 – *Alert,* 0x16 – *Handshake,* 0x17 – *Application Data*
2. 2. a 3. bajt po prevedení na reťazec musí byť rovný **“0300”,”0301”,”0302”,”0303”** alebo **“0304”**, ktoré značia o akú verziu SSL/TLS ide (Rozhodol som sa podporovať, všetky aktuálne podporované verzie SSL/TLS a to SSLv3, TLS 1.0, TLS 1.1, TLS 1.2 a TLS 1.3”, pričom verzia nijako neovplyvňuje implementáciu, akurát sa v hlavičke musí nachádzať označenie jeden z týchto verzií na spomínanom 2. a 3. bajte).

O kontrolu či daný paket splňuje vyššie uvedené požiadavky sa stará funkcia ***filter_ssl_packets()***, ktorej predáme aktuálny paket a ukazateľ na začiatok údajnej SSL/TLS hlavičky a v prípade, že sa jedná o SSL/TLS paket je vrátený ukazateľ na tento paket, inak je vrátená hodnota NULL. V prípade, že sa jedná o SSL/TLS paketom, uloží sa jednoznačný identifikátor spojenia v podobne zdrojovej IP adresy a portu, z ktorého odstránime znak “.” (**client_ID**) pre jednoduchšie pracovanie, uložíme si taktiež aj cieľovú IP adresu a číslo portu, pre porovnávanie paketov v opačnom smere (**server_ID**).

Nasleduje spracovanie paketu, na základe toho ako aký typ obsahu ide. Ak sa jedná o paket typu *Handshake* a Handshake protokol je *Client Hello*, vieme že sa jedná o paket, ktorý zahajuje SSL/TLS spojenie a musí sa k nemu pristúpiť inak ako k ostatným. Vytvorí sa štruktúra **ssl_session** a uloží sa do nej postupne:

1. SNI – k získaniu mena požadovaného serveru sa využije funkciu ***get_SNI()***, ktorej sa predá ukazateľ na začiatok ssl spojenia + fixné posunutie cez SSL/TLS hlavičku na prvý údaj s hodnotou, ktorá udáva premenlivú dĺžku. Následne sa vždy ukazateľ posunie a vypočíta sa nové posunutie až k SNI.
2. Cieľovú a zdrojovú IP adresu
3. Nastavíme počítadlo paketov na 1
4. Počet bajtov z SSL/TLS hlavičky. Na ich získanie a prevedenie do celočíselnej podoby použijeme funkciu ***get_int_from_two_bytes()***, ktorej predáme ukazateľ na začiatok SSL hlavičky posunutie k prvému bajtu tohto čísla
5. Ukazateľ na štruktúru typu **tm**, ktorý získame po zavolaní funkcie **localtime()** s parametrom obsahujúcim adresu času z hlavičky paketu.
6. Time stamp obsahujúci sekundy a milisekundy času prvého paketu

Po uložení všetkých potrebných informácii do štruktúry **ssl_session** inkrementujeme počítadlo bajtov v pakete o veľkosť SSL/TLS časti a štruktúra ***ssl_session*** sa vloží do mapy všetkých SSL/TLS spojení pod kľúčom identifikujúcim spojenie (**client_ID**).

Zvyšné pakety typu *Handshake* alebo pakety typu *Alert, Change Cipher Spec, Application Data* sa spracujú rovnakým štýlom, a to zavolaním funkcie ***process_packet()***, ktorej predáme kľúč, identifikujúci spojenie, v premennej **client_ID** alebo **server_ID** závisejúcej od smeru prenosu paketu. Funkcia nájde príslušné spojenie v mape spojení a uloží do neho informáciu o počte bajtov, ktorý daná SSL/TLS hlavička udáva, vypočíta časový rozdieľ medzi aktuálnym paketom a prvým paketom spojenia a uloží do štruktúry **ssl_session**. Pokiaľ spracovávaný paket ešte nebol započítaný do počítadla paketov v danom ssl spojení započíta sa. Funkcia vráti posun - počet bajtov, ktorý môžme v pakete preskočiť lebo sa jedná o telo SSL/TLS časti, o ktorý inkrementujeme počítadlo bajtov v pakete.

TODO: VYPIS PAKETOV, UKONCENIE SPOJENIA

4. Testovanie

TODO

5. Záver

TODO

6. Referencie

TODO