VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Síťové aplikace a správa sítí

Monitoring SSL spojení

2020/2021

Tomáš Ďuriš
(xduris05)

Banská Bystrica
15.11.2020

Obsah

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 pcapng 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++.*

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

2. Uvedenie do problematiky

Ako som už v úvode 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 pochopeniu jednotlivých častí SSL spojenia a veľkosti jednotlivých položiek v hlavičkách daných štandardom mi pomohli RFC dokumenty (https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml). K samotnému naprogramovaniu aplikácie mi pomohla kniha *Síťové aplikace a jejich architektura* (Petr Matoušek, 2014), v ktorej 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 oficiálnou dokumentáciou knižnice libpcap.

Pri programovaní samotnej aplikácie bolo nutné porozumieť, že TLS 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 aktívne ssl spojenia môžeme považovať spojenie, kde dôjde k výmene 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 koncoch komunikácie.
Za koniec spojenia považujem TCP paket obsahujúci príznak FIN a odpoveď v podobne TCP paketu s príznakmi FIN a ACK. Taktiež za koniec spojenia považujem TCP paket s príznakom RST či už z jednej alebo druhej strany. Pakety o ukončení spojenia s príznakom FIN musí odoslať aj klient aj server, aby sa dané spojenie považovalo za ukončené.

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 prišiel paket s príznakom FIN či už od klienta alebo od serveru a následne prišla odpoveď v podobe TCP paketu s príznakmi FIN a ACK z opačnej strany, prípadne prišiel paket s príznakom RST od jednej zo strán komunikácie. 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 oprávnením 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 požiadavku na preloženie (príkaz make), ktorý program preloží prostredníctvom pribaleného súboru Makefile. Výsledný súbor je určený a spustiteľný pre operačný systém Linux.

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

```
./sslsniff <args>
```

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 odposluch na rozhraní <interface>. Spracovanie argumentov je bližšie popísané v sekcii 3.3 alebo v priloženom readme - **sslsniff.1**.

3.2. Hierarchia súborov

Projekt sa skladá z nasledujúcich súborov:

- **sslsniff.cpp** – deklarácia štruktúry s SSL spojeniami, kontrola linkovej vrstvy, skompilovania filtra pre odposluch, a odposluch na danom rozhraní alebo prípadné čítanie zo súboru.
- **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 užívateľom 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 súboru alebo rozhrania. Informácie o tom aký parameter bol zadaný je predaná naspäť telu programu prostredníctvom zdieľaných premenných typu bool *have_interface* a *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 vypíše sa 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 odpočúva sa na ňom. Pri nezadaní žiadneho argumentu sa vypíše nápoveda programu.

Pokiaľ prebehne kontrola argumentov bez problému nasleduje otvorenie súboru pre čítanie alebo otvorenie rozhrania na odposluch. Po úspešnom 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 môžeme ignorovať pakety, ktoré tento protokol neobsahujú) a jeho následná aplikácia.

3.4. Spracovanie jednotlivých paketov

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, port klienta, ip adresu serveru a port serveru (**client_ID**), keďže táto kombinácie jednoznačne určuje jednu TCP komunikáciu. Do štruktúry s informáciami o spojení si taktiež uložíme aj kombináciu

kde je najprv ip adresa a port serveru a až potom ip adresa a port klienta pre overenie keď paket v komunikácii smeruje z opačnej strany.

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 začiatku TCP dát až po koniec, pričom sa hľadá začiatok SSL hlavičky a po jej nájdení preskočíme jej dáta a hľadáme novú hlavičku. 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. bajt musí byť rovný **0x03** a 3. bajt musí byť rovný **0x01, 0x02, 0x03** alebo **0x04**, ktoré značia o akú verziu TLS ide (Rozhodol som sa podporovať verzie TLS 1.0, TLS 1.1, TLS 1.2 a TLS 1.3" staršie verzie neobsahujú SNI a nie sú využívané).

O kontrolu či daný paket splňuje vyššie uvedené požiadavky sa stará funkcia ***filter_ssl_packets()***, ktorej predáme aktuálny paket a ukazateľ na začiatok údajnej TLS hlavičky a v prípade, že sa jedná o TLS paket je vrátený ukazateľ na tento paket, inak je vrátená hodnota NULL. V prípade, že sa jedná o TLS paketom, uloží sa jednoznačný identifikátor spojenia v podobne zdrojovej IP adresy, portu, cieľovej IP adresy a portu, z ktorého odstránime znak "." (**client_ID**) pre porovnávanie paketov v opačnom smere si uložíme aj identifikátor, ktorý má vymenené údaje o IP adrese a porte klienta s údajmi o IP adrese a porte serveru (**server_ID**).

Pri spracovaní každého paketu sa pozrieme či už nemáme vytvorenú štruktúru s informáciami o tomto spojení. Ak nie (jedná sa o prvý SYN paket v TCP spojení) vytvoríme štruktúru a uložíme informácie:

1. ID pre identifikáciu paketu smerujúceho z opačnej strany komunikácie
2. Inkrementujeme počet paketov
3. Ukazateľ na štruktúru typu **tm**, ktorý získame po zavolaní funkcie **localtime()** s parametrom obsahujúcim adresu času z hlavičky paketu.
4. Time stamp obsahujúci sekundy a milisekundy času prvého paketu

Po uložení počiatočných informácií, vložíme štruktúru do mapy štruktúr všetkých SSL spojení, kde využijeme k jej identifikácií ID. Následne si uložíme ukazateľ na túto štruktúru a ďalej pracujeme s ním. Po nájdení paketu s SSL hlavičkou kontrolujeme o aký typ ide. Ak sa jedná o paket typu *Handshake* a Handshake protokol je *Client Hello*, vieme že sa jedná o paket, ktorý zahajuje TLS spojenie a musí sa k nemu pristúpiť inak ako k ostatným. Z paketu Client Hello si uložíme:

1. SNI – k získaniu mena požadovaného serveru sa využije funkciu ***get_SNI()***, ktorej sa predá ukazateľ na začiatok ssl spojenia + fixné posunutie cez 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. Počet bajtov z TLS hlavičky. Na ich získanie a prevedenie do celočíselnej podoby použijeme funkciu ***process_packet()***, ktorej predáme ukazateľ na začiatok SSL hlavičky posunutie k prvému bajtu tohto čísla

Po uložení všetkých potrebných informácii do štruktúry **ssl_session** inkrementujeme počítadlo bajtov v pakete o veľkosť TLS časti.

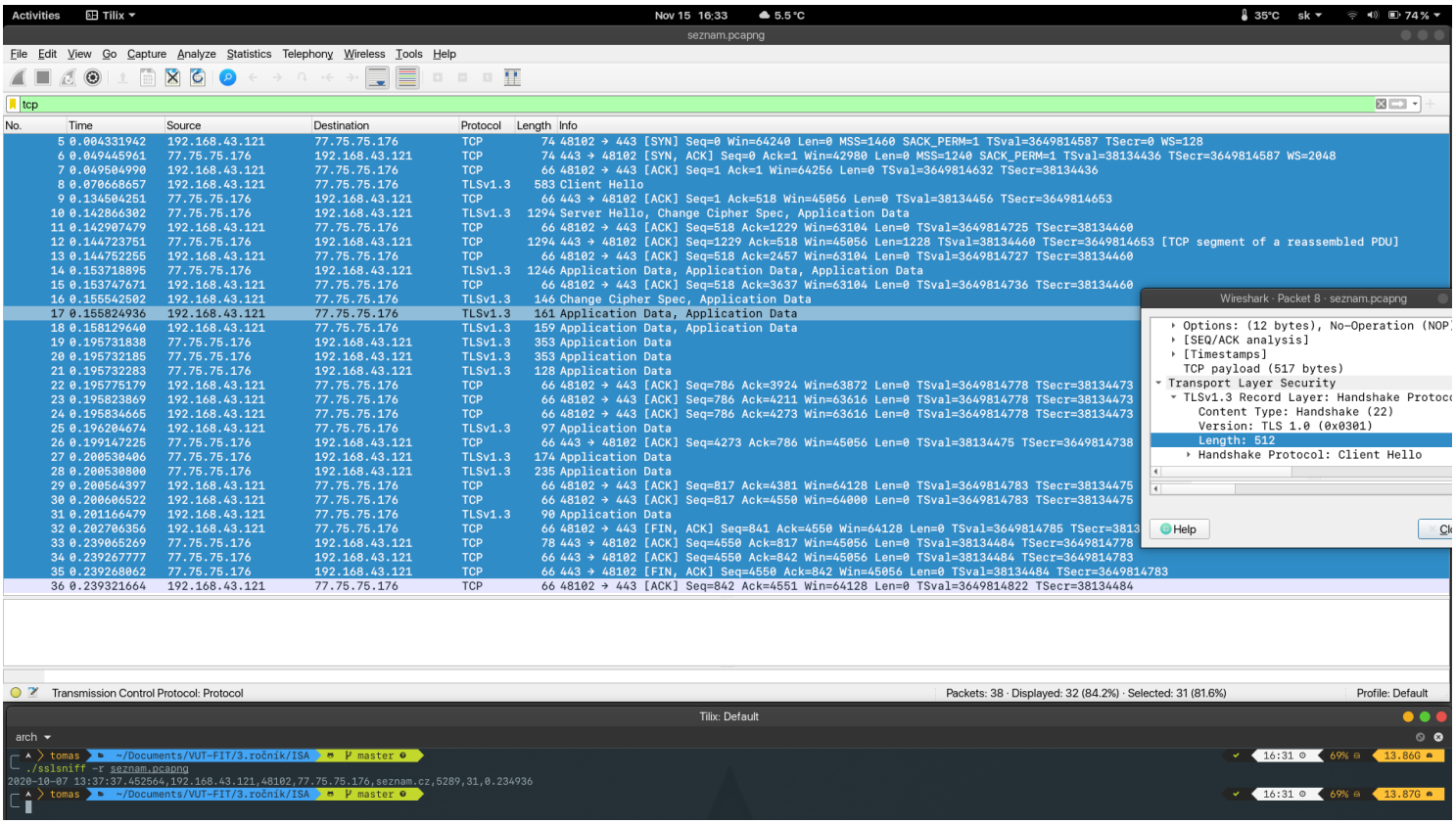
Zvyšné pakety typu *Handshake* alebo pakety typu *Alert, Change Cipher Spec, Application Data* sa spracujú rovnakým štýlom, a to zavolaním funkcie ***process_packet()***, ktorej predáme ukazateľ na štruktúru s konkrétnym ssl spojením a ukazateľ na začiatok ssl dát v aktuálnom pakete. Funkcia uloží do štruktúry s aktuálnym spojením informáciu o počte bajtov, ktorý daná TLS hlavička udáva a vráti posun - počet bajtov, ktorý môžeme v pakete preskočiť lebo sa jedná o telo TLS časti, o ktorý inkrementujeme počítadlo bajtov v pakete.

Pokiaľ bol prijatý TCP paket s príznakom RST nastaveným na 1 overíme či je spojenie aktívne (jedná sa o SSL spojenie) a vypíšeme ho, získame rozdiel medzi časmi prvého a aktuálneho (posledného) paketu pomocou funkcie ***timediff()*** a odstránime z mapy všetkých SSL spojení. Pokiaľ bol prijatý TCP paket s príznakom **FIN** nastaveným na 1 zavolá sa funkciu ***process_FIN_packet()***, ktorej sa predá ukazateľ na tcp štruktúru, ukazateľ na konkrétne spojenie, mapu všetkých SSL spojení, ukazateľ na hlavičku paketu, ID klienta a ID štruktúry. Funkcia skontroluje či sa jedná o ukončenie zo strany klienta alebo serveru. Ak sa jedná o prvý paket s príznakom **FIN**, uložíme si informáciu o jeho príchode a pri ďalšieho paketu v rovnakom spojení s príznakom **FIN** a **ACK** skontrolujeme či sa jedná o paket smerujúci z opačnej strany spojenia. Po úspešnom overení získame rozdiel medzi časmi prvého a aktuálneho (posledného) paketu pomocou funkcie ***timediff()*** a vypíšeme všetky požadované informácie na štandardný výstup. Po vypísaní ho odstránime z mapy všetkých SSL spojení.

4. Testovanie a porovnávanie

Ako vzor pre testovanie som sa rozhodol použiť už existujúci open-source program WireShark. Myslím si, že som dosiahol dobrý výsledok, keďže vo väčšine prípadov je môj výpis zhodný s informáciami, ktoré poskytuje WireShark po analýze rovnakého paketu. V niektorých prípadoch ani WireShark nedokáže jednoznačne určiť SSL hlavičky a aké veľkosti obsahujú, jedná sa hlavne o *reassembled* pakety. V týchto prípadoch bolo porovnávanie s WireSharkom náročnejšie, avšak aspoň približne sa dalo odvodiť či je táto informácia správna.

Testovanie bolo realizované na referenčnom virtuálnom stroji (distribúcia Ubuntu), na serveri merlin a aj na mojom operačnom systéme (distribúcia Arch). Vo všetkých prípadoch bol výsledok zhodný s výsledkom v programe WireShark a program bol plne funkčný. Testovaná bola ako aj verzia IPv4 tak aj verzia IPv6. Nižšie prikladám ukážkový výstup môjho programu a programu WireShark pre rovnaký vstup.



Na vyššie priloženom obrázku môžeme vidieť že sa zhoduje počet paketov v programe WireShark a aj v našom programe sslsniff (v tomto prípade rátame od prvého TCP paketu po druhý FIN paket (prebehlo ukončenie z oboch strán)). Po bližšom preskúmaní a manuálnom zrátaní sa počet bajtov v SSL dátach taktiež zhoduje v oboch programoch zhodujú sa aj časové údaje a SNI. Na základe týchto informácií vieme zhodnotiť, že program pracuje ako má

5. Referencie

1. Kurose, J. a Ross, K., 2014. *Počítačové Sítě*. Brno: Computer Press. ISBN 9788025138250.
2. Matoušek, P. 2014. *Síťové aplikace a jejich architektura*. VUTIUM. ISBN 9788021437661
3. Tools.ietf.org. 2020. *RFC 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0*. [online]. Dostupné na: https:/tools.ietf.org/html/rfc6101 [cit. 2020-11-15].
4. Tls.ulfheim.net. 2020. *The Illustrated TLS Connection: Every Byte Explained*. [online]. Dostupné na: https://tls.ulfheim.net [cit. 2020-11-15].
5. Tcpdump.org. 2020. *Manpage Of PCAP*. [online] Dostupné na: https://www.tcpdump.org/manpages/pcap.3pcap.html [cit. 2020-11-15].
6. Tcpdump.org. 2020. *Programming With Pcaptcpdump/LIBPCAP Public Repository.* [online] Dostupné na: https://www.tcpdump.org/pcap.html [cit. 2020-11-15].