
Síťové aplikace a správa sítí
2020 / 2021

# Monitoring SSL spojení

Vypracoval: Jan Lorenc (xloren15) Datum: 21. 10. 2020

# Obsah

## Zadání

Úkolem projektu bylo rozpoznat zabezpečené spojení SSL/TLS v síťové komunikaci. Požadovaná podporovaná verze TLS byla alespoň TLS 1.2. Síťová komunikace se musela buď odposlechnout z uživatelem zadaného síťového rozhraní, nebo ze zadaného .pcapng souboru. Řešení muselo být implementováno v jazyce C/C++ a spustitelné na různých platformách, primárně však na školním serveru merlin a referenčním virtuálním stroji.

## Jak funguje SSL/TLS?

Protokol pracuje mezi transportní a aplikační vrstvou, kde zabezpečuje přenos dat, která jsou šifrovaná a zajišťuje integritu těchto dat, důvěrnost a autentizaci. To, že se používá tento protokol, se často pozná přidaným „s“ za běžnými protokoly, tedy http – https, pop – pops atd.

Ssl pakety se přenáší přes tcp protokol, který zde slouží jako takový zprostředkovatel. Z toho důvodu před samotným ssl spojením musí stále proběhnou klasický tcp handshake. Následuje ssl handshake, který vypadá následovně (C – klient, S – server):

C – S: Client Hello – Zahájení spojení požadavkem klienta, který serveru předá verzi protokolu, nějaká náhodná data, seznam sad šifrovacích algoritmů, seznam kompresních metod, rozšíření a id sezení.

S – C: Server Hello – Odpověď serveru, který předá stejné informace o sobě

S – C: Server Certificate – Poskytne certifikát obsahující název serveru, veřejný klíč a důkaz toho, že server vlastní privátní klíč k tomuto veřejnému, poskytnutý důvěryhodnou třetí stranou. (nepovinný)

S – C: Server Key Exchange – Informace potřebné pro výměnu klíčů. Klient i server mají vlastní privátní a veřejný klíč a k tomu veřejný klíč toho druhého. Díky tomu se pak vygeneruje společný klíč kombinací vlastního privátního klíče a veřejného klíče druhé strany. (nepovinný)

S – C: Certificate Request – Poslán, pokud je zapnutá klientská autentikace. (nepovinný)

S – C: Server Hello Done – Server indikuje, že skončil se svou půlkou handshaku.

C – S: Client Certificate – Poslán, pokud si ho server vyžádal přes Certificate Request. (nepovinný)

C – S: Client Key Exchange – Klientské informace pro výměnu klíčů.

C – S: Certificate Verify – Slouží k potvrzení, že vlastní privátní klíč ke svému veřejnému certifikátu. (nepovinný)

C – S: Client Change Cipher Spec – Indikace, že společné klíče jsou spočítané a nadále šifruje svým zapisovacím klíčem.

C – S: Client Handshake Finished – Odešle serveru zašifrovaný celý handshake se speciálním číslem pro identifikaci klienta.

S – C: Server Change Cipher Spec – Indikace, že společné klíče jsou spočítané a nadále šifruje svým zapisovacím klíčem.

S – C: Server Handshake Finished – Odešle klientovi zašifrovaný celý handshake se speciálním číslem pro identifikaci serveru.

Následně si klient a server vyměňují zašifrovaná data, dokud klient nepošle Close Notify, kterým ukončuje spojení.

Pro potřeby projektu je ještě důležité znát ssl hlavičku, která obsahuje tři položky a celkem má 5B:

- Typ - 1B … hodnoty: 20(ChangeCipherSpec), 21(Alert), 22(Handshake), 23(Application data)
- Verze - 2B … hodnoty: 300(SSL 3.0), 301(TLS 1.0), 302(TLS 1.1), 303(TLS 1.2)
- Délka dat - 2B

## Návrh aplikace

Knihovna libpcap dovoluje poměrně snadno zachytávat pakety jak ze síťového rozhraní, tak i ze souboru, a to téměř stejným způsobem. Navíc obsahuje struktury pro hlavičky různých typů paketů, proto lze bez větších problémů rozpoznat, o který paket se aktuálně jedná. Tím řečeno, bez potřeby cokoliv vymýšlet dokážeme určit, zda máme v ruce tcp paket. Jediné, co je pro tuto část aplikace potřeba, je mít na začátku dvě cesty na získání pcap dat, jednu pro soubor a jednu pro síťové rozhraní. Dále je již vše společné.

Po obdržení tcp paketu nám již však libpcap moc nepomůže a zjistit, zda se jedná o ssl, již musíme sami na základě obsahu tcp zprávy. V první řadě je však potřeba rozpoznat zahájení spojení. Jakmile zachytneme tcp paket s příznakem SYN, vytvoříme spojení a kontrolujeme, jestli je validní, tedy čekáme na odpověď s příznaky SYN a ACK a pak ještě potvrzení druhého SYN jednou ACK zprávou. V tomto bodě máme úspěšný tcp handshake a jedná-li se o ssl spojení, následuje ssl handshake. Čekáme tedy postupně na Client Hello a Server Hello, které již musíme rozpoznat z obsahu tcp zprávy, která by měla na začátku obsahovat ssl hlavičku. Pokud se úspěšně rozpozná i ssl handshake, dostaneme se do stavu zachytávání dat a de facto čekání na FIN, v opačném případě spojení mažeme, neboť se nejedná o ssl.

Během tohoto stavu každý paket bereme jako potenciální ssl a pokud jím je, tak navýším počet bytů spojení o jeho délku avšak pozor, může nastat situace, kdy je tato větší než je délka aktuálního paketu. V tomto případě si je třeba uložit velikost tohoto přetečení a v následujícím paketu tuto použít jako offset, odkud číst další data.

Jestliže nám již přijde FIN paket, očekáváme dále ACK a FIN od druhé strany, nicméně tato druhá strana ještě s komunikací skončit nemusela a stále nám může něco poslat, proto ještě nekončíme se zpracováváním příchozích ssl paketů. Až nám dojde druhý FIN, tak již pouze čekáme na závěrečný ACK a zde již skutečně ignorujeme vše ostatní. Po posledním ACK paketu vypíšeme informaci o spojení a toto smažeme z našeho seznamu.

## Implementace

Projekt sestává z několika souborů, kde každý obsahuje vlastní část řešení. Prvním z nich je tento soubor manual.pdf obsahující dokumentaci. Dále je zde Makefile, který je poměrně jednoduchý, obsahující jediný příkaz na sestavení programu, jehož výstupem je pouze spustitelný soubor, žádné .o soubory, proto není nutný ani make příkaz clean. Posledním souborem neobsahující zdrojový kód je sslsniff.1 nahrazující soubor README z obecného zadání, ve kterém lze najít stručný popis programu ve formátu linuxové manuálové stránky.

Zdrojových souborů je vícero z důvodu čistoty kódu a konvencí jazyků C a C++. V souboru sslsniff.cpp se nachází velmi krátká funkce main(), jež pouze zpracuje vstupní argumenty programu pomocí třídy Args a spustí zachytávání paketů přes třídu Sniffer.

Soubory error.h a error.cpp obsahují deklaraci a definici chybové funkce, jež vypíše chybu a ukončí program.

V args.h a args.cpp pak lze najít třídu Args. Ta si ve svých atributech uchovává hodnoty vstupních argumentů programu, metodou Parse() zvaliduje argumenty a pokud jsou v pořádku, uloží si vstup do svých atributů a nechá program pokračovat. V opačném případě ukončí program s popisem problému.

Dále řešení obsahuje connection.h a connection.cpp se třídou Connection reprezentující ssl spojení. Třída obsahuje všechny informace o spojení, které máme zpracovávat a vypisovat, tedy zdrojovou adresu a port, cílovou adresu, počet paketů a bytů, SNI, začátek a konec spojení. Mimo to udržuje svůj stav, k čemuž slouží enum ExpectedPacket definovaný taktéž v connection.h, a výše zmíněnou hodnotu přetečení velikosti paketu. Metodu má jen jednu a tou jen Print(), jež vypíše informace o spojení.

Ústřední třídou celého programu je Sniffer nacházející se v souborech sniffer.h a sniffer.cpp. Jejími atributy jsou:

- Type - určuje, zda se jedná o soubor nebo síťové rozhraní
- Source - jméno zdroje, ať už soubor nebo rozhraní
- Connections - seznam aktuálně zpracovávaných spojení ve formě instancí třídy Connection
- EthHeaderSize - konstanta o hodnotě 14 udávající velikost ethernetové hlavičky
- TcpHeaderLengthOffset - konstanta o hodnotě 12 udávající pozici délky tcp hlavičky
- SslHeaderSize - konstanta o hodnotě 5 udávající velikost ssl hlavičky

Metody (pro zkrácení bez parametrů):

- Sniff() - jediná veřejná metoda. Slouží k zahájení zpracování paketů. Pouze na základě typu zavolá buď SniffFromInterface(), nebo SniffFromFile().
- SniffFromInterface() - zkontroluje existenci síťového rozhraní zavoláním metody CheckInterfaceExistence(), poté rozhraní otevře pro poslech, čímž získá strukturu pcap_t a tuto zpracuje zavoláním metody SniffFromPcap().
- SniffFromFile() - zkontroluje existenci a správný typ zadaného souboru zavoláním metody CheckFileExistence(), poté ze souboru načte data do struktury pcap_t a tuto zpracuje zavoláním metody SniffFromPcap().
- CheckInterfaceExistence() - najde všechna dostupná síťová rozhraní s nimiž porovná uživatelem zadané. Pokud existuje, tak nechá program pokračovat, jinak ukončí program a vypíše dostupná rozhraní.
- CheckFileExistence() - zkontroluje existenci souboru a jestli je skutečně typu .pcapng. Pokud validace neprojde, končí program s chybovou hláškou.
- SniffFromPcap() - metoda je de facto jeden velký cyklus načítající pakety z pcap. U každého paketu se ověří zda se vůbec jedná o ip paket a z něj se uloží zdrojová a cílová adresa, dále se ověří, zda se jedná o tcp paket a z něj se uchová zdrojový a cílový port.

  V tuto chvíli víme, že se jedná o tcp paket a pokusíme se ho přiřadit k existujícímu spojení. Pokud spojení neexistuje, tak pokud má paket příznak SYN, spojení vytvoří, pokud ne, tak patří spojení, jehož začátek aplikace nezachytila, například proto, že již bylo navázáno, když se teprve spustila.

  Dále následuje přepínač dle aktuálního stavu spojení. V případě tcp handshake se pouze nastaví následující požadovaný stav (SYN – ACK+SYN – ACK) a pokud se tak nestane, spojení se smaže. Dalšími stavy jsou prvky ssl handshaku, kdy se ověří přítomnost ssl metodou ProcessSSL(), která vrací počet ssl zabezpečených přenesených bytů. O ssl se jedná, pokud tato vrátí nenulový počet bytů. V případě Client Hello se vyhledá i SNI metodou FindSNI(). Mimo to se stejně jako u tcp handshaku jen nastaví další očekávaný stav a pokud něco nesedí, spojení se maže jako neplatné. Ve stavu WAITING_FOR_FIN se každý paket testuje jako ssl, dokud nedojde paket s příznakem FIN, kdy se pak přechází do stavů dle ukončení tcp spojení (FIN – FIN+ACK – ACK). Dokud však nedojde druhý FIN, stále zpracováváme ssl pakety, které ještě mohou dojít od druhé strany a až na ACK druhého FINu spojení ukončíme, vypíšeme a smažeme.
- CreateConnection() - vytvoří a inicializuje novou instanci třídy Connection a přidá ji do aktuálních spojení.
- FindConnection() - vyhledá dle parametrů konkrétní aktuální spojení a vrátí na něj ukazatel.
- RemoveConnection() - smaže zadané spojení ze seznamu.
- ProcessSSL() - v cyklu prochází data tcp paketu. Vyhledá hlavičku ssl, navýší počet bytů spojení o délku ssl paketu a o tuto navýší offset v pcap paketu. Takto v cyklech najde všechny ssl pakety. Konec nastává, pokud nebyla detekována ssl hlavička, nebo se překročila velikost paketu. V druhém případě se pod podmínkou, že k překročení došlo ssl daty, velikost přetečených dat uloží a použije jako offset v příštím volání funkce.

- FindSNI() – Client Hello paket obsahuje SNI na specifickém místě, ze kterého ho funkce přečte. Toto místo však není statické a musí se k němu dopočítat. Začne s počátečním offsetem 43B (ssl hlavička (5B), handshake hlavička (6B), random(32B)), poté si přečte délku Session Id a posune se dál, kde si přečte délku Cipher Suites, stejně pak Compression Methods až se dostane k Extensions, ve kterých prohledává, dokud nenajde typ o hodnotě 0x00 0x00 značící SNI, přečte jeho délku, a pak přečte právě tolik bytů, čímž získá SNI, které vrátí.

# O programu

Program zachytává ssl síťová spojení a upozorňuje na ně uživatele přes CLI. Uživatel může specifikovat, ze kterého síťového rozhraní či .pcapng souboru má pakety zachytávat. Při ukončení spojení se informace vypíše do příkazové řádky v následujícím formátu.

<timestamp>,<client ip>,<client port>,<server ip>,<SNI>,<bytes>,<packets>,<duration sec>

Kde „bytes“ je počet bytů přenesených ssl pakety, tedy součet hodnot „length“ políček v ssl hlavičkách a „packets“ je počet všech paketů týkajících se spojení od prvního SYN paketu až k ACK druhého FINu, tedy ne pouze ssl pakety.

Program končí, dočetl-li .pcapng soubor nebo v případě poslechu na rozhraní dokud ho uživatel sám neukončí.

## Použití

Po rozbalení odevzdaného .tar souboru stačí zadat příkaz „make“, kterým se program sestaví a vytvoří spustitelný soubor sslsniff. Pro informace o programu se můžeme podívat do manuálových stránek příkazem „man -l sslsniff.1“.

Program má 3 možné kombinace argumentů:

| | |
|---|---|
| -h | Vypíše nápovědu o používání aplikace |
| -r <file> | Program bude číst pakety ze zadaného soubor |
| -i <interface> | Program bude poslouchat síťový provoz na zadaném rozhraní |

Uživatel musí program spustit s právě jednou z uvedených možností. Žádný argument není přímo povinný, nicméně některý se zadat musí. Zároveň není možné argumenty kombinovat.

Ukázkový vstup a výstup:

```
./sslsniff -r pcap.pcapng
2020-10-22 20:33:24.963783,100.120.36.107,51133,91.198.174.192,en.wikipedia.org,1221,11,0.111
```

# Testování

Aplikaci jsem testoval několika způsoby. Nejprve jsem testoval samostatná spojení, které jsem vytvořil pomocí programů openssl či curl a zároveň zachytával programem Wireshark. Výstupy jsem porovnal a tímto způsobem jsem mohl ověřit korektnost všech informací, které máme zaznamenávat. Čas, klient a server jsou zjevné, počet paketů jsem si taktéž jednoduše spočítal. Počet bytů byl náročnější, byť stále bezproblémový. Svým programem jsem si vypsal počet ssl bytů každého paketu a porovnával s délkou v ssl hlavičkách ve Wiresharku. Takto jsem zkoušel spojení různých verzí ssl a různých velikostí požadavků, stejně tak i naopak ignorování spojení, které nejsou ssl.

Dále jsem považoval za důležité otestovat vícenásobné spojení ke stejnému serveru, čili změna je jen v klientském portu. De facto jsem postupoval stejně, jen jsem vytvořil více spojení zároveň, ukončoval a znovu připojoval. Takto spojení běžela zároveň na jiných portech nebo při ukončení jednoho a připojení dalšího se znovu použil původní port. Aplikace úspěšně zvládala, neboť spojení správně oddělila a rozeznala právě díky různým portům.

Pro ověření stejného chování pro čtení ze souboru a z rozhraní jsem aktivně poslouchal z rozhraní a zároveň zachytával Wiresharkem. Posléze jsem .pcapng z Wiresharku dal aplikaci na vstup a výstup porovnal s jejím výstupem při aktivním zachytávání.

Na závěr jsem testoval větší zátěž. Spustil jsem aplikaci i Wireshark a navštěvoval různé webové stránky. U tohoto způsobu nešlo ověřit počet ssl bytů, pokud bych nechtěl strávit hodiny porovnáváním všech paketů mezi Wiresharkem a aplikací, ale test sloužil k tomu, jestli správně zachytím pouze ssl spojení, jestli je zachytím všechny, zda je správný zdroj, cíl a čas a jestli aplikace nespadne pod plnohodnotným síťovým provozem.

# Reference


Driscoll, M. *The Illustrated TLS Connection.* Dostupné z: https://tls.ulfheim.net

Bandara, S. (2019-6-27). *TLS Handshake: Under the Hood*. Dostupné z: https://medium.com/@technospace/tls-handshake-under-the-hood-79d20c0020de

Dierks, T., Independent, Rescorla, E. & RTFM, Inc (2008, srpen). *The Transport Layer Security (TLS) Protocol Version 1.2.* Dostupné z: https://tools.ietf.org/html/rfc5246

Carstens, T. & Harris, G. *Programming with Pcap.* Dostupné z: https://www.tcpdump.org/pcap.html