VUT v Brně
Fakulta informačních technologií

ISA
Monitoring SSL spojenia

Peter Vinarčík
Login: xvinar00

V Brne
3.11.2020

Obsah

Úvod

V tomto projekte z predmetu „*Síťové aplikace a správa sítí*“ s názvom **Monitoring SSL spojení** sa zameriame na implementáciu SSL snifferu, ktorý bude po vygenerovaní spustiteľnej binárky schopný spracovať pcap súbor s koncovkou .pcapng, z ktorého spracuje dáta a informácie o daných SSL spojeniach.

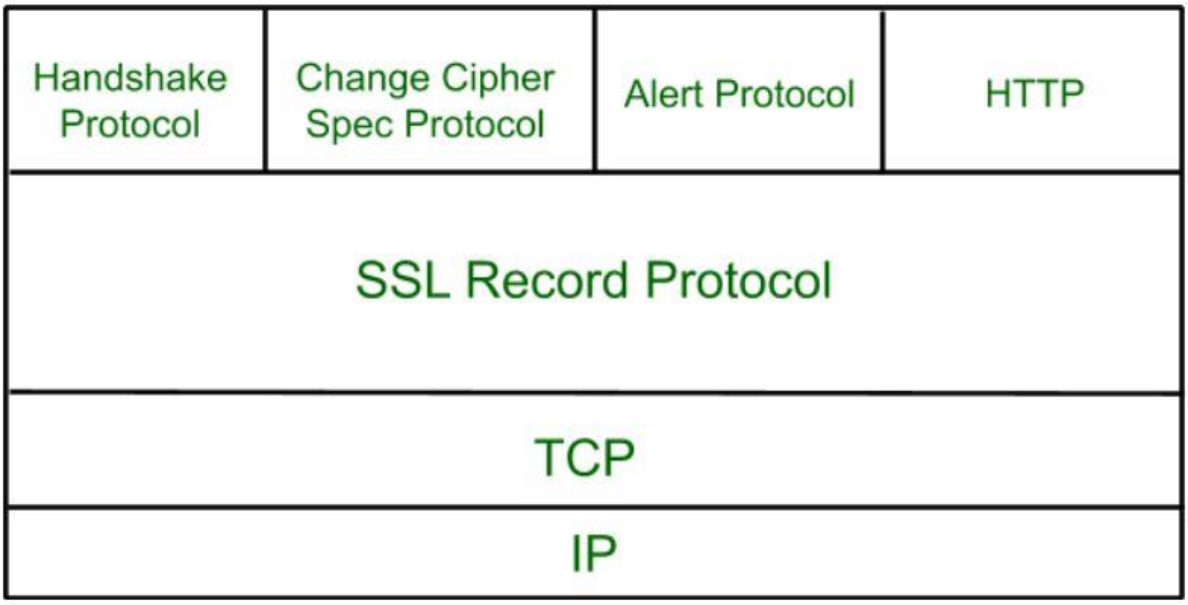
Zároveň tiež budeme schopný sniffer používať aj pre **live capture SSL spojení** na rozhraniach nachádzajúcich sa v našom zariadení.

Čo je to SSL?

Secure Sockets Layer, alebo skrátene **SSL** je protokol, ktorý zabezpečuje bezpečnosť komunikácie pri posielaní dát medzi komunikujúcim stranami *(resp. presnejšie povedané, je to vrstva, ktorá je vložená medzi vrstvu transportnú (TCP) a aplikačnú (HTTP))*[1] a **zabezpečuje komunikáciu šifrovaním a autentizáciou týchto strán**.

V dnešnej dobe je SSL zastúpené svojím následovníkom, ktorým je **Transport Layer Security** (ďalej v texte už len ako **TLS**).

Bližsie a detailnejšie sa na túto vrstvu pozrieme v ďalších kapitolách.



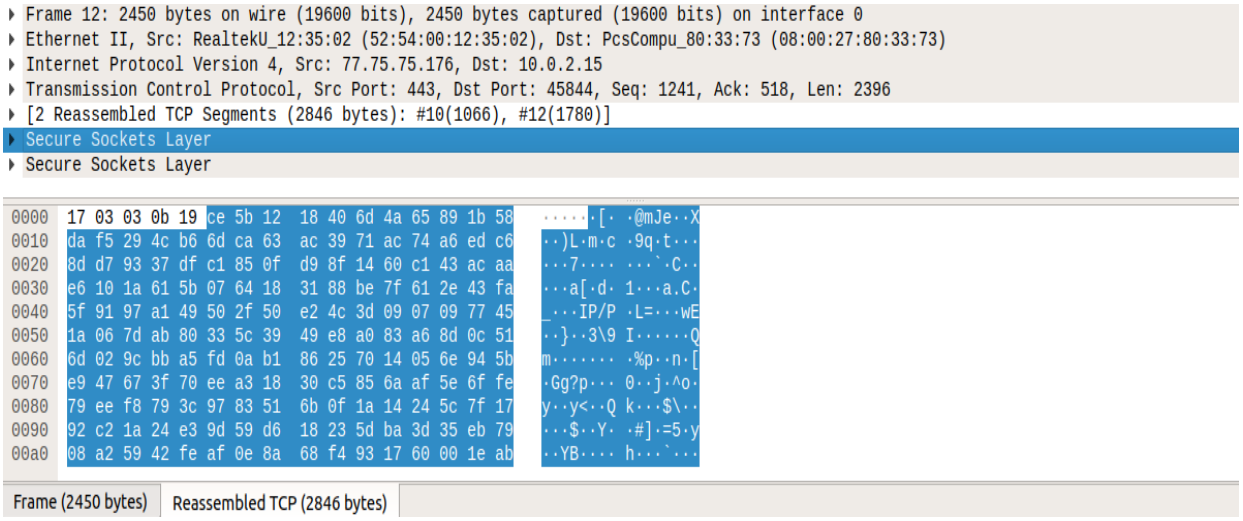
Obr. 1 – vizualizácia [1] z úvodného textu

Návrh aplikácie

Návrh aplikácie prebiehal v dvoch etapách. V prvej časti bolo nutné sa zoznámiť **s problematikou týkajúcou sa SSL respektíve TLS protokolu**.

Bolo nevyhnutné **preskúmať jednotlivé TLS pakety** a ich samotnú štruktúru pomocou programu **wireshark**, ktorý nám detailne popísal byty nachádzajúce sa na špecifických pozíciách tak, ako je to uvedené v štandardoch daného protokolu.

Po načrtnutí týchto informácií sa konečne môžeme presunúť do ďalšej kapitoly, kde si detailne popíšeme význam jednotlivých pozičných bytov podľa už spomínaného štandardu nášho implementovaného SSL protokolu resp. jeho snifferu.



Obr. 2 – štruktúra TLS packetu, vyznačený payload a hlavička TLS

Implementácia

Po zoznámení sa so štruktúrou a problematikou TLS / SSL protokolu môžeme prejsť k samotnej implementácii.

Ako sme si mohli v prechádzajúcej kapitole povšimnúť, TLS packet má na začiatku svojho payloadu (prvý byte payloadu sa klasicky nachádza na fixnej pozícii pre TCP packet) tzv. „Record Type“ teda v preklade typ záznamu, ktorý nám špecifikuje, čo máme v danom TLS packete očakávať.

Poznáme 4 typy záznamov:

- Handshake (0x16)
- Change Cipher Spec (0x14)
- Alert (0x15)
- Application Data (0x17)

Po vyššie zmienenom byte nasledujú ďalšie 2 byty, ktoré špecifikujú o akú verziu sa jedná. Naša implementácia konkrétne počíta s verziami (resp. hodnotami bytov)

- Druhý byte: 0x03
- Tretí byte: 0x01, 0x02, 0x03, 0x04

Tretí byte payloadu napísaný vyššie označuje verziu TLS resp. SSL protokolu.

Aktuálne sme opísali prvé 3 dôležité byty payloadu (môžeme porovnať s obrázkom vyššie kde payload začína čísla 17 – application Data, 03 03 – verzia TLS protokolu.

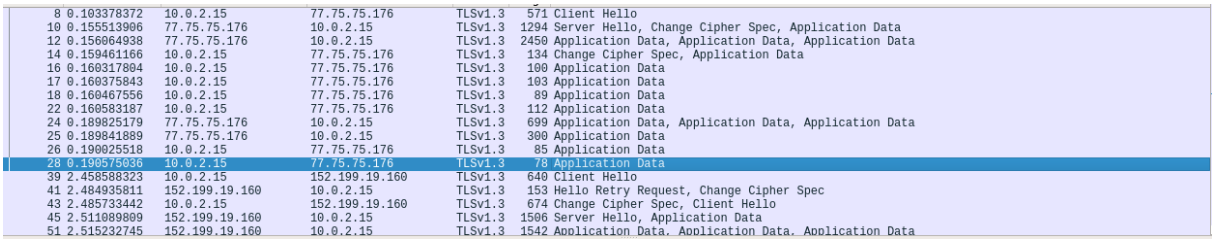
Ďalšie nasledujúce 2 byty sa po ich konkatenácii (poďla obrázku vyššie 0b 19 -> 0b19) a následnom prevedení na decimálnu hodnou pomocou vlastnoručne implementovanej funkcie convertHexLength() zmenia na výslednú dĺžku SSL packetu (do ktorej sa nepočíta spomínaných prvých 5 bytov [record type 1 byte, verzia tls 2 byty, dĺžka SSL packetu 2 byty).

Po tomto prevedení, môžeme výslednú dĺžku pripočítať k celkovej veľkosti daného spojenia (ktoré je kontrolované pomocou zdrojovej a cieľovej IP adresy respektíve portov).

Každé spojenie je uchovávané v globálnom liste vo forme štruktúry, ktorá obsahuje spomínané IP adresy zdroju a cieľu a to isté platí o portoch daného spojenia. Ďalej sa tu uchováva počet všetkých TCP packetov daného spojenia (ako je počet vypočítavaný si povieme neskôr) a výsledný počet bytov daného spojenia. Veľkosť (v bytoch) je pripočítavaná iba v prípade, že doručený paket je SSL paket (nie iba TCP paket daného spojenia).

V určitých situáciách, kedy je paket obzvlášť veľký, môžeme pozorovať tzv. REASSEMBLED PAKETY priamo vo Wiresharku. Tieto pakety fungujú na princípe, že ich časť je poslaná v payloade nasledujúceho paketu, čo má za následok, že payload nemusí striktne začínať vyššie spomínanými kombináciami.

Na riešenie tejto situácie tu máme pripravené tzv. prechádzania po pakete cez offset, ktorý nám umožňuje celý paket prehľadať a prípadné „skryté“ TLS / SSL objaviť a pripočítať počet bytov do výslednej komunikácie. V krátkosti povedané, po objavení dĺžky aktuálneho paketu, ktorý sme vyčítali z SSL hlavičky, skočíme v payloade o offset. Offset je stanovený ako dĺžka aktuálneho paketu, ktorý sme vyčítali zo SSL hlavičky + 5 bytov hlavičky a až na tomto mieste začneme hľadať ďalší možný paket. Nie je samozrejme zaručené, že sa tu bude nachádzať, ale pre optimálnosť a predovšetkým PRESNOSŤ riešenia je nevyhnutné túto skutočnosť takouto formou kontrolovať.



8	0.103378372	10.0.2.15	77.75.75.176	TLSv1.3	571 Client Hello
10	0.155513906	77.75.75.176	10.0.2.15	TLSv1.3	1294 Server Hello, Change Cipher Spec, Application Data
12	0.156064938	77.75.75.176	10.0.2.15	TLSv1.3	2450 Application Data, Application Data, Application Data
14	0.159461166	10.0.2.15	77.75.75.176	TLSv1.3	134 Change Cipher Spec, Application Data
16	0.160317804	10.0.2.15	77.75.75.176	TLSv1.3	100 Application Data
17	0.160375843	10.0.2.15	77.75.75.176	TLSv1.3	103 Application Data
18	0.160467556	10.0.2.15	77.75.75.176	TLSv1.3	89 Application Data
22	0.160583187	10.0.2.15	77.75.75.176	TLSv1.3	112 Application Data
24	0.189825179	77.75.75.176	10.0.2.15	TLSv1.3	699 Application Data, Application Data, Application Data
25	0.189841889	77.75.75.176	10.0.2.15	TLSv1.3	300 Application Data
26	0.190025518	10.0.2.15	77.75.75.176	TLSv1.3	85 Application Data
28	0.190575036	10.0.2.15	77.75.75.176	TLSv1.3	78 Application Data
39	2.458588323	10.0.2.15	152.199.19.160	TLSv1.3	640 Client Hello
41	2.484935811	152.199.19.160	10.0.2.15	TLSv1.3	153 Hello Retry Request, Change Cipher Spec
43	2.485733442	10.0.2.15	152.199.19.160	TLSv1.3	674 Change Cipher Spec, Client Hello
45	2.511089809	152.199.19.160	10.0.2.15	TLSv1.3	1506 Server Hello, Application Data
51	2.515232745	152.199.19.160	10.0.2.15	TLSv1.3	1542 Application Data, Application Data, Application Data

Obr. 3 – Application Data príklad

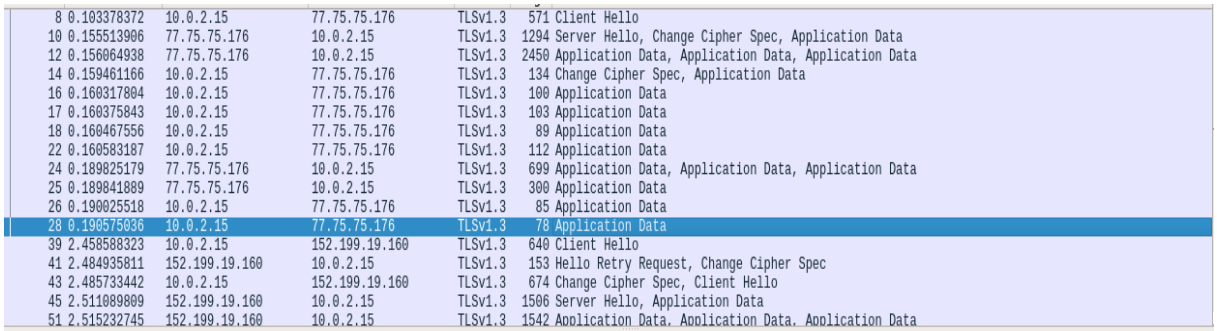
Na obrázku, ktorý môžeme vidieť vyššie môžeme pozorovať vyznačený paket, ktorý má už spomínaný názov Application Data, čo znamená, že jeho payload, respektíve v určitých situáciách (v prípade reassembled paketu) niekde vnútri jeho payloadu sa nachádza kombinácia čísel 17 03 03. Pre zopakovanie si to prejdime ešte raz. 17 ➔ Record Type, 03 03 ➔ TLS vo verzii 1.3.

Dôležité časti implementácie

Ako sa uvádza aj v manuálovej príručke pre program (sslsniff.1), implementácia počíta s niektorými „obmedzeniami“, resp. upresnením určitých Častí, ktoré v samotnom zadaní neboli detailne upresnené resp. boli neskôr upresnené na fóre týkajúce sa daného projektu.

Implementácia počíta s tým, že:

- za **VALÍDNU SSL KOMUNIKÁCIU**, ktorá môže byť VYPÍSANÁ NA ŠTANDARDNÝ VÝSTUP je považovaná iba komunikácia, v ktorej prebehli obidve verzie handshaku (01, 02) to znamená, že komunikáciu **musí obsahovať ako CLIENT HELLO, tak aj SERVER HELLO.**
- Ráta sa s tým, **že spojenie končí po paketoch, ktoré obsahovali FIN flag** (2x) a tento **flag musí prísť jedenkrát od klienta a druhýkrát od serveru** – tým sa zabezpečí 100 percentná presnosť pri ukončovaní spojenia, nakoľko FIN flag môže prísť aj 3, resp 4-krát, pretože prvýkrát sa posielanie nemusí podariť a tak musí prísť k znovuodoslaniu.
- Ráta sa **len s verziami TLS 1.0 a vyššie**!



8	0.103378372	10.0.2.15	77.75.75.176	TLSv1.3	571 Client Hello
10	0.155513906	77.75.75.176	10.0.2.15	TLSv1.3	1294 Server Hello, Change Cipher Spec, Application Data
12	0.156064938	77.75.75.176	10.0.2.15	TLSv1.3	2450 Application Data, Application Data, Application Data
14	0.159461166	10.0.2.15	77.75.75.176	TLSv1.3	134 Change Cipher Spec, Application Data
16	0.160317804	10.0.2.15	77.75.75.176	TLSv1.3	100 Application Data
17	0.160375843	10.0.2.15	77.75.75.176	TLSv1.3	103 Application Data
18	0.160467556	10.0.2.15	77.75.75.176	TLSv1.3	89 Application Data
22	0.160583187	10.0.2.15	77.75.75.176	TLSv1.3	112 Application Data
24	0.189825179	77.75.75.176	10.0.2.15	TLSv1.3	699 Application Data, Application Data, Application Data
25	0.189841889	77.75.75.176	10.0.2.15	TLSv1.3	300 Application Data
26	0.190025518	10.0.2.15	77.75.75.176	TLSv1.3	85 Application Data
28	0.190575036	10.0.2.15	77.75.75.176	TLSv1.3	78 Application Data
39	2.458588323	10.0.2.15	152.199.19.160	TLSv1.3	640 Client Hello
41	2.484935811	152.199.19.160	10.0.2.15	TLSv1.3	153 Hello Retry Request, Change Cipher Spec
43	2.485733442	10.0.2.15	152.199.19.160	TLSv1.3	674 Change Cipher Spec, Client Hello
45	2.511089809	152.199.19.160	10.0.2.15	TLSv1.3	1506 Server Hello, Application Data
51	2.515232745	152.199.19.160	10.0.2.15	TLSv1.3	1542 Application Data, Application Data, Application Data

Obr. 4 – demonštrácia validného handshaku

Na obrázku vyššie je demonštrované ako je očakávaný handshake v rámci spojenia (na prvých dvoch paketoch obrázku). Vidíme, že po príchode Client Hello bol nasledovaný Server Hello. Nie je nutná podmienka aby boli nevyhnutne za sebou avšak stále musí platiť, že HANDSHAKE musí PREBEHNÚŤ CELÝ.

Literatúra