

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Projektová dokumentácia k predmetu ISA
Monitoring SSL spojenia

Obsah

1	Úvod	2
2	Spustenie programu	2
3	Použité knižnice	3
3.1	Esenciálne pre jazyk C	3
3.2	Esenciálne pre prácu so sieťovými prvkami	3
4	Implementácia	4
4.1	Zostavenie komunikácie	4
4.2	Práca a analýza s paketmi	4
4.2.1	Spracovanie TCP	5
4.2.2	Spracovanie SSL spojenia	5
4.3	Výpis výstupu	6
5	Testovanie	7
6	Použité zdroje	8

1 Úvod

Zadaním projektu bolo vytvorenie nástroja na sledovanie SSL spojenia.

Projekt pozostáva hlavne z dvoch častí. Prvou časťou je sieťový adaptér, ktorý pripája program k danému sieťovému rozhraniu, prípadne je komunikácia načítaná zo súboru *pcap/pcapng*. Druhá časť uskutočňuje analýzu samotných zachytených paketov, filtruje pakety vhodné pre potreby funkcionality a získavajú sa informácie o danom spojení, ktoré sa následne zobrazí na obrazovke.

2 Spustenie programu

Program je kompatibilný s linuxovými systémami (*program bol vyvíjaný na Ubuntu 20.04.LTS*). K správnej kompilácii je vhodné disponovať prekladačom `gcc 7.5.0` a vyššie. Takisto je potrebný program `make`, testované na verzií `GNU Make 4.1` a knižnica `libpcap` testovaná vo verzií `0.8`.

V priečinku projektu sa nachádza Makefile, ktorý umožní projekt zostaviť použitím:

```
$ make
```

Pri zostavovaní projektu dochádza na referenčnom stroji alebo na serveri Merlin k warningu o nepoužívaní hodnoty parametru *args* vo funkcii *callback*, avšak nijako nebráni k správnej činnosti programu.

Vyčistenie zkompilovaného programu `sslsniff` je možné pomocou:

```
$ make clean
```

Projekt sa spúšťa pomocou:

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

Pokiaľ nie je možné projekt spustiť pri prepínači `-i` je potrebné mu poskytnúť administrátorské práva:

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

- `-i <rozhranie>` - určuje rozhranie, na ktorom bude program pracovať, môže byť zadaný menom rozhrania (maximálna dĺžka 40 znakov) alebo číslom označujúci poradie rozhrania.
- `-r <file>` - určuje súbor vo formáte *pcap* alebo *pcapng*, z ktorého sa načíta komunikácia. Voľba `<file>` môže byť s maximálnym počtom 1000 znakov. Môže byť zadaná relatívna aj absolútna cesta.
- `-help` | `-h` - zobrazí nápovedu.
- `žiadny argument` - zobrazí nápovedu a zobrazí dostupné rozhrania, na ktorých je možné zachytávať komunikáciu. V prípade chybných argumentov (chýbajúca voľba, prekročenie maximálneho počtu znakov), program skončí s návratovou hodnotou **1**. V prípade spoločne zadaných argumentov `-i` a `-r` sa berie do úvahy len argument `-i`, `-r` je ignorovaný.

3 Použité knižnice

V programe je použitých mnoho knižníc, dajú sa rozdeliť na dve kategórie.

3.1 Esenciálne pre jazyk C

Prvou kategóriou sú potrebné knižnice pre podporu funkcií jazyka C:

- `<stdio.h>`, `<stdlib.h>`, `<stdbool.h>`, `<string.h>` - štandardné funkcie ako *malloc*, práca s reťazcami.
- `<signal.h>` - na zachytenie signálu ukončenia programu pomocou CTRL+C.
- `<getopt.h>` - na spracovanie argumentov príkazového riadku.
- `<time.h>`, `<sys/types.h>` - na správnu prácu s časom.
- `"sslsniff.h"` - vlastný hlavičkový súbor, obsahujúci štruktúry `conn_info` a funkcie obojstranne viazaného zoznamu.

3.2 Esenciálne pre prácu so sieťovými prvkami

Druhou kategóriou sú knižnice potrebné pre pripojenie sa k sieťovému adaptéru, prípadne parsovaniu pcapng súboru, alebo k používaniu štruktúr paketov:

- `<arpa/inet.h>` funkcie *inet_pton(...)*, *inet_pton(...)*, *inet_pton(...)* pre prácu s IP adresami IPv4, IPv6.
- `<pcap.h>` funkcie z pcap knižnice slúžiace k zostaveniu sieťového adaptéra, ktorý sa pripojí k existujúcemu pripojeniu a taktiež k zachytávaniu paketov.
- `<netinet/ip.h>`, `<netinet/ip6.h>` - štruktúry hlavičiek IPv4 a IPv6 paketov.
- `<netinet/tcp.h>` - štruktúra hlavičky TCP.
- `<netinet/if_ether.h>` - štruktúra ethernetovej hlavičky.

4 Implementácia

Program je implementovaný v jazyku C v súbore `sslsniff.c`. `sslsniff.c` je rozdelený do niekoľkých funkcií. Na začiatku sa do premenných načítajú vstupné argumenty uvedené v sekcii 2 pomocou funkcie `args_parse(int argc, char *argv[], char *iface, char *rfile)`. Jednotlivé časti sú v komentári kódu ozdrojované a okomentované.

4.1 Zostavenie komunikácie

Implementácia sieťového adaptéru pripajajúceho sa na existujúcu sieť alebo parsovanie pcapng súboru je vo funkciách `pcap_t set_up(int mode, char *iface, char *rfile)` využívajú funkcie knižnice `pcap.h`. Po správnom nakonfigurovaní funkcií `pcap_open_offline`, prípadne `pcap_open_live`, je možné zostaviť filter, ktorý je nastavený na prepúšťanie TCP paketov, nakoľko len tieto pakety sú schopné prenášať SSL pakety [5]. Funkciou `pcap_compile` s parametrami `pcap_compile(sniff, &fp, "tcp ", 0, pNet)` skompilujeme náš adaptér a následne ho môžeme aplikovať pomocou `pcap_setfilter(sniff, &fp)`, v úspešnom prípade funkcia vráti štruktúru typu `pcap_t`, ak zlyhá ukončí program s návratovou chybou -10 [2].

Takto zostavený adaptér teraz môžeme pomocou `pcap_loop(sniff, -1, callback, NULL)` uviesť do „nekonečného cyklu“, kedy sa zachytávajú pakety a pri každom zachytenom pakete sa volá funkcia `callback` [1]. V prípade načítavania z pcapng súboru funkcia končí pri poslednom pakete v zachytenej komunikácii inak je ju potrebné ukončiť pomocou `ctrl+c`.

4.2 Práca a analýza s paketmi

Funkcia `callback(u_char *args, const struct pcap_pkthdr* pkthdr, const u_char* buffer)` spracováva každý zachytený paket. Pomocou `struct ether_header *p` sa zisťuje či daný paket používa IPv4, alebo IPv6, v prípade, že `ntohs(p->ether_type) == ETHERTYPE_IPV6` nastaví sa `bool ipv6 = true`; . Timestamp paketu - teda čas, kedy bol daný paket zachytený je uložený v `pkthdr->ts.tv_sec` a mikro sekundy v `pkthdr->ts.tv_usec`. Program je možné kedykoľvek ukončiť pomocou **CTRL+C**.

Dôležitá štruktúra v projekte je `struct conn_info`:

```
typedef struct conn_info
{
    char src_addr[40];    //contains source IPv4/6 , 40 is the longest it could
                        get
    char dest_addr[40];   //contains destination IPv4/6
    unsigned long src_port; //contains source PORT
    char sni[1025];       //SNI - server name indication of max size 1024
                        characters
    long start_time;      //the timestamp of the first packet in connection (TCP
                        SYN)
    long start_microsec;   //the same as above but the microseconds
    int packets;          //packets count in connection from TCP SYN to TCP FIN
    int size;             //SSL data in Bytes transfered in connection
    bool has_ssl;         // flag to recognize from basic tcp
    bool has_syn_ack;     // flag connection established
    bool second_fin;      // flag finished connection
    int overflow;         // signaling when TLS load is bigger than actual TCP
                        payload
} conn_info;
```

Táto štruktúra je elementom v poli `tDLinkedList connection_list`, ktorého funkcionlita je implementovaná v súbore `list.c`. Štruktúra `conn_info` uchováva informácie o spojení ako IP adresy zdroja a cieľa, port zdroju, SNI (*server name indication*), počiatočný čas komunikácie, počet paketov v spojení, počet bytov prenesených TLS a príznaky informujúce stav komunikácie. Každé spojenie je jasne dané svojou IP adresou a

číslo portu. Pri spracovávaní konkrétneho spojenia sa vďaka týmto vlastnostiam vie určiť, o ktoré spojenie v liste `tDList connection_list` sa jedná.

4.2.1 Spracovanie TCP

Pri zachytenom TCP pakete sa volá funkcia `int tcp_packet(long time, long microsec, const u_char *buffer, bool ipv6, unsigned int data_len)`. V tejto funkcii sa získavajú informácie o IP adresách (funkcie `readIPv4`, `readIPv6`), portoch `tcph->source`, `tcph->dest` (zo štruktúry `struct tcphdr` [7] [6]). Pri zlyhaní funkcie `malloc` vráti hodnotu `-10`. Kontrolujú sa príznaky **SYN**, **ACK** a **FIN**, ktoré sú pre nás významné, pretože vieme určiť stav komunikácie. Pri vzniku TCP komunikácie musí prebehnúť TCP handshake, kedy sa najskôr nastavujú príznaky SYN, odpoveď SYN ACK a potom ACK od zdroja a spojenie je nadviazané. Pri ukončení spojenia sa nastavujú príznaky FIN a potvrdenie ukončenia FIN ACK [8]. Sledovaním týchto príznakov sa mení volanie funkcie `tcp_connection(...)`.

Funkcia `void tcp_connection(const u_char *buffer, unsigned data_len, int tcphdr_len, uint16_t src_port, uint16_t dst_port, char *src_addr, char *dst_addr, long time, long microsec, int tcp_syn)` slúži k správaniu aktuálneho spojenia prípadne volanie funkcií na spracovanie SSL/TLS paketov. Pri volaní funkcie s hodnotou premennej `int tcp_syn 1` (nastavený príznak SYN), sa použije vetva spracovania nového TCP spojenia, kedy sa vytvára plný premenná `conn_info header`, doplnia sa informácie o porte, IP adresách, timestamp, nastaví sa príznak a počet paketov a následne sa premenná header vloží do listu `connection_list` pomocou funkcie `DLInsertLast`. Pokiaľ bola premenná `int tcp_syn` s hodnotou 2 použije sa vetva `else` vyhladá sa pomocou IP adresy a portu dané spojenie v liste `connection_list` a pokiaľ sa tam spojenie nachádza nastaví sa príznak `has_syn_ack` na `true`, čím sme potvrdili, že dané spojenie sa skutočne nadviazalo. Ak bola funkcia volaná s `int tcp_syn == 0` testuje sa vo vedľajšej vetve či sa v pakete nenachádza hlavička SSLv3 až TLS1.2 paketu. Ak bol zachytený príznak FIN nastavuje sa príznak `second_fin` na `true`, vďaka tomu pri druhom zachytenom FIN príznaku daného spojenia je spojenie vypísané na štandardný výstup vo funkcii `void print_data(...)`.

4.2.2 Spracovanie SSL spojenia

SSL/TLS (*Secure Socket Layer/Transport Layer Security*) je nezávislé zabezpečenie dát nad transportnou vrstvou v TCP paketoch. Každý mechanizmus vyžaduje vytvorenie a distribúciu kľúčov, zabezpečenie a overenie [5]. Zastaralé verzie SSL sa už dnes nepoužívajú, preto aj pre potreby projektu je implementovaná podpora pre SSLv3 a vyššie, teda TLS1.0, TLS1.1 a TLS1.2.

Keďže je spojenie šifrované, informácie o pakete môžeme dostať len z SSL/TLS hlavičky. Sú viaceré možnosti ako je možné hlavičku nájsť, jednou je otestovať postupnosť bajtov ihneď na začiatku TCP payloadu [4].

```
if ((buffer[tcphdr_len] == 0x14) || //SSLv3 - TLS1.2
    (buffer[tcphdr_len] == 0x15) ||
    (buffer[tcphdr_len] == 0x16) ||
    (buffer[tcphdr_len] == 0x17)) &&
    (buffer[tcphdr_len + 1] == 0x03 && (buffer[tcphdr_len + 2] < 0x04)))
```

- `0x14` - `CHANGE_CIPHER_SPEC` - informuje že je potrebné zmeniť šifru.
- `0x15` - `ALERT` - upozornenie o udalostiach, ktoré nastali
- `0x16` - `HANDSHAKE` - prebieha SSL/TLS handshake
- `0x17` - `APPLICATION_DATA` - paket nesúci dáta

Premenná `tcphdr_len` značí pozíciu v bytes array `u_char buffer` kde začína TCP payload. Na tejto pozícii sa musia nachádzať 0x14, 0x15, 0x16, 0x17 bajty, ktoré značia úlohu prenášaného paketu [3]. Nasleduje verzia SSL/TLS protokolu, ktorej prvý bajt začína 0x03 a druhý opisuje konkrétnu verziu (SSL3.0 = 0x00, TLS1.0 = 0x01, TLS1.1 = 0x02, TLS1.2 = 0x03). Za týmto bajtom sa ďalej nachádza informácia o dĺžke, ktorú SSL/TLS nesie. Pomocou funkcie `ntohs` sa táto 2-bajtová veľkosť spracuje do desiatkovej sústavy a pričíta sa do celkovej veľkosti (`connection_list.Act->data.size`), ktorú spojenie prenieslo. Avšak nie vždy sa hlavička nachádza hneď na začiatku TCP payload, ale môže sa nachádzať aj inde. V takom prípade je potrebné použiť funkciu `int loop_packet (const u_char *buffer, int tcphdr_len, unsigned int data_len)`. Prehľadáva payload TCP paketu a nachádza postupnosť SSL/TLS hlavičky spomenutú vyššie. Ak ju nájde preskočí sa o nájdenú veľkosť na miesto, kde by sa mala nachádzať ďalšia hlavička. Ak sa tam nenachádza funkcia vráti doposiaľ nájdenú veľkosť, inak vráti 0. Zisťuje sa aj prípadné pretečenie. To znamená, že pokiaľ sa nájde veľkosť hlavičky, ktorá prevyšuje dĺžku spracovávaného TCP paketu, pretekajúca veľkosť sa uloží do `connection_list.Act->data.overflow`. Ak je vrátená z funkcie nenulová hodnota, prípadne bola hlavička priamo nájdená na začiatku TCP payloadu, je volaná funkcia `void ssl_connection(...)`.

Funkcia zabezpečujúca správne spracovanie náležitostí SSL/TLS spojenia.

```
void ssl_connection(const u_char *buffer, unsigned data_len,
    int tcphdr_len, int segmented_packet)
```

Významná udalosť pre nadviazanie SSL/TLS spojenia je SSL handshake. O prebiehajúcom handshaku nás informuje bajt s hodnotou 0x16. Handshake je iniciovaný takzvaným **Client-Hello**, pýta sa v ňom serveru, ktoré z ponúknutých šifier a metód je možné použiť. Informácia o typu handshaku sa nachádza na `buffer[tcphdr_len + 5]` pokiaľ je rovný 0x01 jedná sa o Client-Hello a vo funkcii `ssl_connection` sa následne daný paket spracováva. Z tohto paketu je možné vyčítať info **SNI** - server name indication, značí, ktorému hostname sa klient snaží pripojiť v handshaku [9]. V Client-Hello sa nachádzajú rôzne informácie v *extensions* v jednom z nich sa nachádza aj informácia o SNI.

Funkcia `char *get_TLS_SNI(const u_char *buffer, int tcphdr_len)` sa stará o jej získanie. SNI je uložené do premennej `connection_list.Act->data.sni`, v prípade, že sa v Client-Hello nenachádza táto informácia (napríklad SSL3 SNI nepoužíva) je uložený do premennej reťazec „(Could not find SNI)“. Na potvrdenie Hand-Shake je potrebné dostať odpoveď aj od serveru. V **Server-Hello** sa potvrdí informácia o použitej šifre a metóde. Informáciu o Server-Hello je možné získať vďaka `buffer[tcphdr_len + 5] == 0x02`, zároveň sa aj nastaví príznak `has_ssl = true`. Znamená to, že zabezpečené spojenie je uskutočnené a je možné začať posielať zašifrované dáta [3]. V každom zachytenom pakete v danom spojení je pripočítaný počet paketov a v prípade SSL/TLS paketu je potom pripočítaná aj veľkosť v bajtoch prenesených dát.

4.3 Výpis výstupu

Výstup dát zabezpečuje funkcia,

```
void print_data(uint16_t src_port, uint16_t dst_port,
    char *src_addr, char *dst_addr,
    long time, long microsec)
```

ktorá je volaná z funkcie `tcp_packet` pri zachytenom príznaku FIN. Zabezpečuje výpis ukončeného SSL/TLS spojenia. Za ukončené sa pokladá spojenie, ktoré zaznamenalo dvakrát príznak FIN. Dáta vypisuje na štandardný výstup vo forme určenom v zadaní. Je tu počítaná aj dĺžka trvania spojenia od prvého TCP SYN paketu až po posledný TCP FIN paket v sekundách s presnosťou na mikro sekundy. Po úspešnom vypísaní sa zmaže štruktúra `conn_info` s daným spojením z listu `connection_list`. Taktiež sú tu zmazané aj spojenia, ktoré neobsahovali zabezpečenú komunikáciu.

Výstup programu má nasledovnú štruktúru.

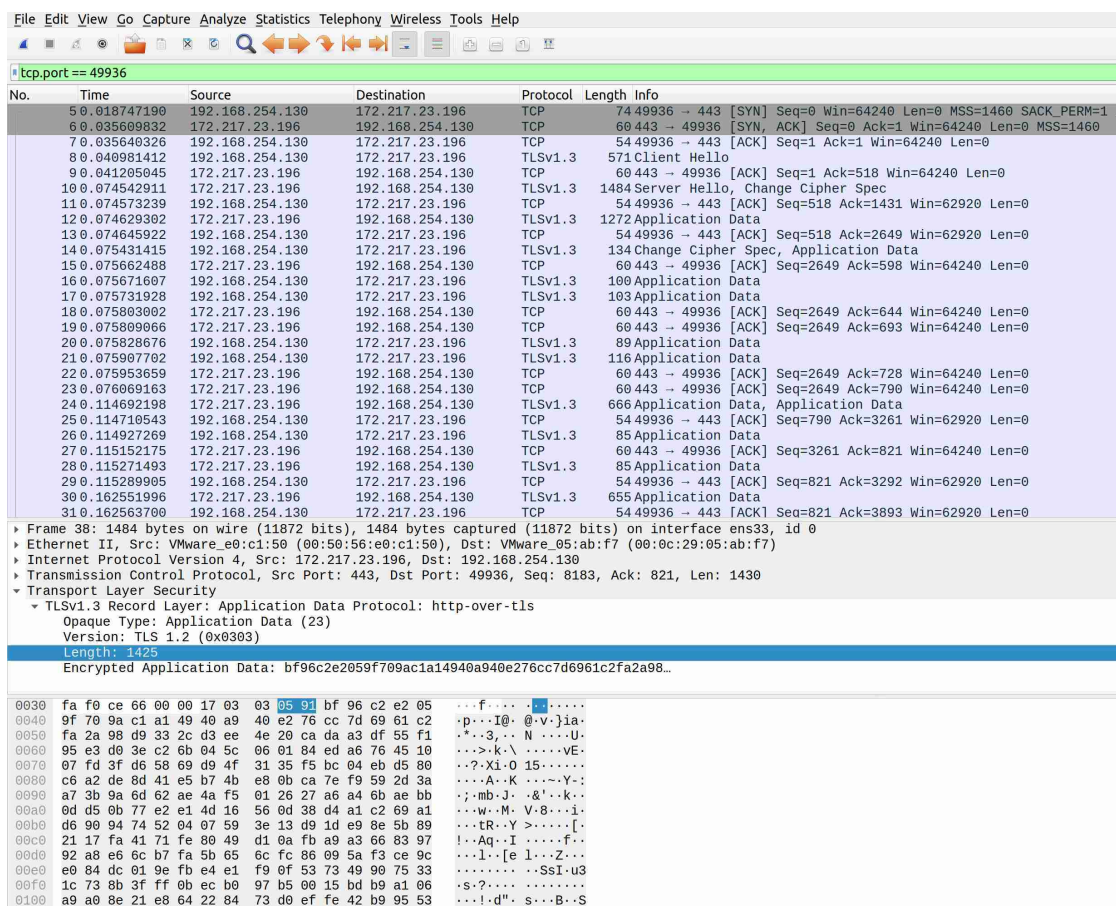
<timestamp>, <client ip>, <client port>, <server ip>, <SNI>, <bytes>, <packets>, <duration sec>

5 Testovanie

Testovanie prebiehalo na stroji so systémom Ubuntu 20.04 a na serveri Merlin. Testovanie prebiehalo manuálnou kontrolou výstupu z programu `sslsniff` so zachytenou komunikáciou s open source softvérom **Wireshark**. Pomocou display filtra `tcp.port == PORT_SPOJENIA` bolo možné zobrazíť práve kontrolované spojenie a manuálne sa odkontroloval počet paketov spojenia a počet prenesených bajtov SSL paketmi.

```
dominik@dominikZenBook:~/Desktop/skola/ISA/VUT-FIT-ISA$ ./sslsniff -r google.pcapng
2020-10-13 15:31:33.663920,192.168.254.130,49936,172.217.23.196,www.google.com,17509,50,0.161493
```

Obrázek 1: Pcapng súbor spracovaný pomocou sslsniff



No.	Time	Source	Destination	Protocol	Length	Info
50	0.018747190	192.168.254.130	172.217.23.196	TCP	74	49936 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 ...
60	0.035609832	172.217.23.196	192.168.254.130	TCP	60	443 → 49936 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
70	0.035640326	192.168.254.130	172.217.23.196	TCP	54	49936 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
80	0.040981412	192.168.254.130	172.217.23.196	TLSv1.3	571	Client Hello
90	0.041205045	172.217.23.196	192.168.254.130	TCP	60	443 → 49936 [ACK] Seq=1 Ack=518 Win=64240 Len=0
100	0.074542911	172.217.23.196	192.168.254.130	TLSv1.3	1484	Server Hello, Change Cipher Spec
110	0.074573239	192.168.254.130	172.217.23.196	TCP	54	49936 → 443 [ACK] Seq=518 Ack=1431 Win=62920 Len=0
120	0.074629302	172.217.23.196	192.168.254.130	TLSv1.3	1272	Application Data
130	0.074645922	192.168.254.130	172.217.23.196	TCP	54	49936 → 443 [ACK] Seq=518 Ack=2649 Win=62920 Len=0
140	0.075431415	192.168.254.130	172.217.23.196	TLSv1.3	134	Change Cipher Spec, Application Data
150	0.075662488	172.217.23.196	192.168.254.130	TCP	60	443 → 49936 [ACK] Seq=2649 Ack=598 Win=64240 Len=0
160	0.075671607	192.168.254.130	172.217.23.196	TLSv1.3	100	Application Data
170	0.075731928	192.168.254.130	172.217.23.196	TLSv1.3	103	Application Data
180	0.075803002	172.217.23.196	192.168.254.130	TCP	60	443 → 49936 [ACK] Seq=2649 Ack=644 Win=64240 Len=0
190	0.075809066	172.217.23.196	192.168.254.130	TCP	60	443 → 49936 [ACK] Seq=2649 Ack=693 Win=64240 Len=0
200	0.075828676	192.168.254.130	172.217.23.196	TLSv1.3	89	Application Data
210	0.075907702	192.168.254.130	172.217.23.196	TLSv1.3	116	Application Data
220	0.075953659	172.217.23.196	192.168.254.130	TCP	60	443 → 49936 [ACK] Seq=2649 Ack=728 Win=64240 Len=0
230	0.07609163	172.217.23.196	192.168.254.130	TCP	60	443 → 49936 [ACK] Seq=2649 Ack=790 Win=64240 Len=0
240	0.114692198	172.217.23.196	192.168.254.130	TLSv1.3	666	Application Data, Application Data
250	0.114719543	192.168.254.130	172.217.23.196	TCP	54	49936 → 443 [ACK] Seq=790 Ack=3261 Win=62920 Len=0
260	0.114927269	192.168.254.130	172.217.23.196	TLSv1.3	85	Application Data
270	0.115152175	172.217.23.196	192.168.254.130	TCP	60	443 → 49936 [ACK] Seq=3261 Ack=821 Win=64240 Len=0
280	0.115271493	172.217.23.196	192.168.254.130	TLSv1.3	85	Application Data
290	0.115289905	192.168.254.130	172.217.23.196	TCP	54	49936 → 443 [ACK] Seq=821 Ack=3292 Win=62920 Len=0
300	0.162551996	172.217.23.196	192.168.254.130	TLSv1.3	655	Application Data
310	0.162563700	192.168.254.130	172.217.23.196	TCP	54	49936 → 443 [ACK] Seq=821 Ack=3893 Win=62920 Len=0

Frame 38: 1484 bytes on wire (11872 bits), 1484 bytes captured (11872 bits) on interface ens33, id 0

Ethernet II, Src: VMware_05:c1:50 (00:50:56:e0:c1:50), Dst: VMware_05:ab:f7 (00:0c:29:05:ab:f7)

Internet Protocol Version 4, Src: 172.217.23.196, Dst: 192.168.254.130

Transmission Control Protocol, Src Port: 49936, Dst Port: 443, Seq: 8183, Ack: 821, Len: 1430

Transport Layer Security

TLShv1.3 Record Layer: Application Data Protocol: http-over-tls

Opaque Type: Application Data (23)

Version: TLS 1.2 (0x0303)

Length: 1425

Encrypted Application Data: bf96c2e2059f709ac1a14940a940e276cc7d6961c2fa2a98...

Obrázek 2: Pcapng súbor spracovaný pomocou programu Wireshark

6 Použité zdroje

Použitá literatura

- [1] ARORA, H.: How to Perform Packet Sniffing Using Libpcap with C Example Code. [online], 2002, [vid. 2020-10-14].
URL <https://www.thegeekstuff.com/2012/10/packet-sniffing-using-libpcap/>
- [2] CARSTENS, T.: Programming with pcap. [online], rev. 25. október 2012, [vid. 2020-10-14].
URL <https://www.tcpdump.org/pcap.html>
- [3] Álvaro Castro-Castilla: Traffic Analysis of an SSL/TLS Session. [online], rev. 23. decembra, 2014, [vid. 2020-10-18].
URL <http://blog.fourthbit.com/2014/12/23/traffic-analysis-of-an-ssl-slash-tls-s>
- [4] Jschauma: Capturing specific SSL and TLS version packets using tcpdump. [online], rev. 8. marca, 2019, [vid. 2020-10-20].
URL <https://www.netmeister.org/blog/tcpdump-ssl-and-tls.html>
- [5] MATOUŠEK, P.: Sít'ové aplikace a správa sítí. [Univerzitná prednáška], 2020.
- [6] VESELÝ, V.: IPv6 Sít'ová vrstva. [Univerzitná prednáška], 2017.
- [7] VESELÝ, V.: Sít'ová vrstva. [Univerzitná prednáška], 2018.
- [8] VESELÝ, V.: Transportní vrstva. [Univerzitná prednáška], 2018.
- [9] Wikipedia: Server Name Indication. [online], rev. 30. október 2020, [vid. 2020-10-30].
URL https://en.wikipedia.org/wiki/Server_Name_Indication