Vysoké učení technické v Brně
Fakulta informačných technológií

Projektová dokumentácia k predmetu ISA

Reverse-engineering neznámeho protokolu

31.10.2021

Ján Homola (xhomol27)

Obsah

1. Úvod

Projekt pozostáva z troch hlavých častí. Prvá časť projektu sa zaoberá zachytením protokolovej komunikácie na virtuálnom stroji medzi referenčným klientom a serverom pomocou nástroja Wireshark. Následne sa táto komunikácia uloží do `pcap` súboru. Druhá časť projektu rieši implementáciu vlastného Wireshark dissectoru pre daný protokol v jazyku `Lua`. Posledná časť projektu sa zaoberá implementáciou náhradného referenčného klienta. Tento klient je implementovaní v jazyku `C`.

2. Spustenie programov

Pre správne spúšťanie servera a referenčného klienta bolo nutné pri prvom spustení použiť prepínač `-h`, aby bolo jasné ako sa dané programy majú spúšťať.

2.1. Spustenie servera

Server sa spúšťa nasledovne:

```
$ ./server [-a <adresa>] [-p <port>] [-h]
```

`-a <adresa>` - Určuje adresu na ktorej daný server počúva. Pokiaľ nie je tento argument zadaní server počúva na lokálne sieti.

`-p <port>` - Určuje na akom porte daný server počúva, pokiaľ tento argument nie je zadaný východzia hodnota portu je 32323.

`-h` - Vypíše nápovedu.

2.2. Spustenie referenčného klienta

Referenčný klient sa spúšťa nasledovne:

```
$ ./client [-a <adresa>] [-p <port>] <príkaz> [-h]
```

`-a <adresa>` - Určuje adresu alebo doménu servera na ktorú sa pripája, pokiaľ nie je tento argument zadaný klient sa pripojí na adresu lokálnej sieti.

`-p <port>` - Určuje na aký port servera sa pripojí, ak nie je tento argument zadaní klient sa pripojí na port 32323.

`-h` - Vypíše nápovedu.

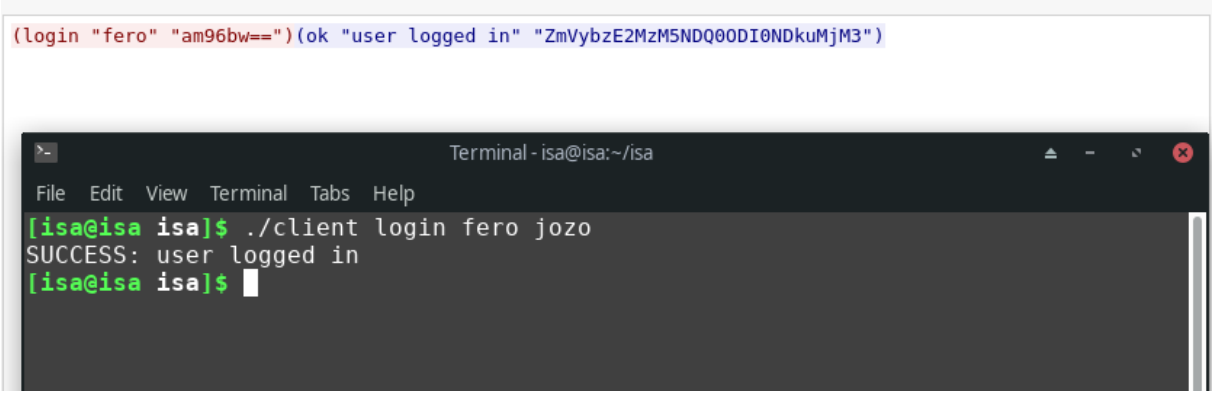
`<príkaz>` - Určuje aký príkaz sa má vykonať.
Podporuje príkazy ako : `register, login, list, send, fetch, logout.`

3. Zachytenie protokolovej komunikácie

Zachytávanie protokolovej komunikácie medzi referenčným klientom a serverom prebiehalo na virtuálnom stroji. Ako prvé bolo potrebné si zapnúť nastroj Wireshark a následne začať generovať komunikáciu medzi klientom a serverom. Generovanie komunikácie spočívalo vtom že bol zapnutý server na lokálnej adrese a na porte s predvoleným číslom 32323. Pomocou referenčného klienta boli posielané jednotlivé príkazy klienta na adresu a port servera. Takto vygenerovaná komunikácia bola zachytená v nástroji Wireshark. Jednalo sa o protokol `TCP` a keďže komunikácia až na heslá nebola šifrovaná bolo možné pomocou `TCP Stream` zobraziť požiadavku klienta na server a taktiež odpoveď serveru. V tejto komunikácií sú hesla šifrované pomocou `base64.` Následne po odchytení rôznych požiadavkou klienta a odpovedí serveru bola táto komunikácia uložená do súboru `isa.pcap`.

3.1. Ukážka zachytenej protokolovej komunikácie

Zachytenie protokolovej komunikácie prebiehalo pomocou Wireshark a nástroja `TCP Stream.`



Obrázok 1 - TCP Stream na klientovu požiadavku login

Na obrázku je zobrazená komunikácia medzi klientom a serverom pomocou `TCP Stream`. Pomocou programu `./client` s príkazom `login` bola poslaná požiadavka na server, ktorá je v nástroji `TCP Stream` zobrazená červenou farbou. V požiadavke je vidieť že heslo je zakódované, jedná sa o kódovanie typu `base64.` V nástroji `TCP Stream` je taktiež vidno odpoveď

servera, ktorá je zobrazená modrou farbou. Na začiatku odpovede servera sa vždy nachádza status servera `ok` v prípade správne zaslanej požiadavky na server alebo `err` pri nesprávnom zaslaní požiadavky na server.

Rovnaký spôsobom prebiehalo skúmanie ďalších komunikácií medzi serverom a klientom a to na základe použitia ostatných podporovaných príkazov ako `register, send, list, fetch` a `logout.` Takto vygenerovaná komunikácia, ktorá bola zachytená pomocou nástroja Wireshark je uložená v súbore s názvom `isa.pcapng`.

4. Wireshark dissector

Dissector je podpora nástroja Wireshark pre sieťové protokoly. Umožňuje zobraziť protokolové dáta v takej forme, ktorá je jednoduchšia a zrozumiteľnejšie čitateľná pre užívateľa. Pri implementácií je potrebné určiť aký typ protokolu bude spracovávaný v tomto projekte ide o `TCP` protokol. Taktiež je nutné registrovať protokol na daný port v našom prípade to bol port s číslom `32323.`

4.1. Implementácia

Dissector je implementovaný v jazyku `Lua` v súbore s názvom `isa.lua`. Ako prvé bolo potrebné vytvoriť nový protokol s vlastným názvom. Môj protokol nesie názov `ISA`.

Hlavná časť programu sa nachádza vo funkcií `my_protokol.dissector().` Pre jednoduchšie rozpoznanie protokolu v nástroji Wireshark bolo potrebne nastaviť názov protokolu na `ISA`. Následne je kontrolované kto je odosielateľom správy a to na základe prvých štyroch znakov správy. Ak tento začiatok správy obsahuje `ok` alebo `err` jedná sa o správu od servera v opačnom prípade ide o správu od klienta. Vo Wiresharku sa do stĺpca s informáciami o pakete vypíše `Response` ak ide o server a pokiaľ ide o klienta vypíše sa `Request`. Podľa toho o akú správu išlo sa vypíšu potrebné dáta k tejto správe aby boli jednoduchšie čitateľne pre užívateľa. Pri odpovedi servera na príkaz `list` sa zavolá pomocná funkcia `get_number_of_msg()`, ktorá vráti hodnotu s počtom správ v danej schránke klienta. Táto hodnota sa pridá do stĺpca s informáciami o pakete.

4.2. Ukážka dissectoru

Pre správne spustenie dissectoru v nástroji Wireshark bolo potrebné pridať daný súbor s názvom `isa.lua` do zložky s názvom `Personal Lua Plugins`. Na ukážku jednotlivých komunikácií medzi klientom a severom bol otvorení v nástroji Wireshark súbor so zachytenou komunikáciou z úlohy číslo 1.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000264370	::1	::1	ISA	117	Request: register test1
6	0.000404899	::1	::1	ISA	116	Response: registered user test1
15	37.819406894	::1	::1	ISA	114	Request: login test1
17	37.819555937	::1	::1	ISA	144	Response: logged in dGVzdDExNjM1Nzc3NDQyNDgxLjc0NA==
24	106.684587195	::1	::1	ISA	179	Request: send [Recipient: test1, Subject: sprava]
26	106.684677601	::1	::1	ISA	107	Response: message sent
33	114.858091499	::1	::1	ISA	129	Request: list dGVzdDExNjM1Nzc3NDQyNDgxLjc0NA==
35	114.858179426	::1	::1	ISA	115	Response: listed 1 messages
42	127.041574916	::1	::1	ISA	132	Request: fetch 1
44	127.041685736	::1	::1	ISA	144	Response: fetch [From: test1, Subject: sprava]
51	138.623074284	::1	::1	ISA	131	Request: logout dGVzdDExNjM1Nzc3NDQyNDgxLjc0NA==
53	138.623185742	::1	::1	ISA	105	Response: logget out

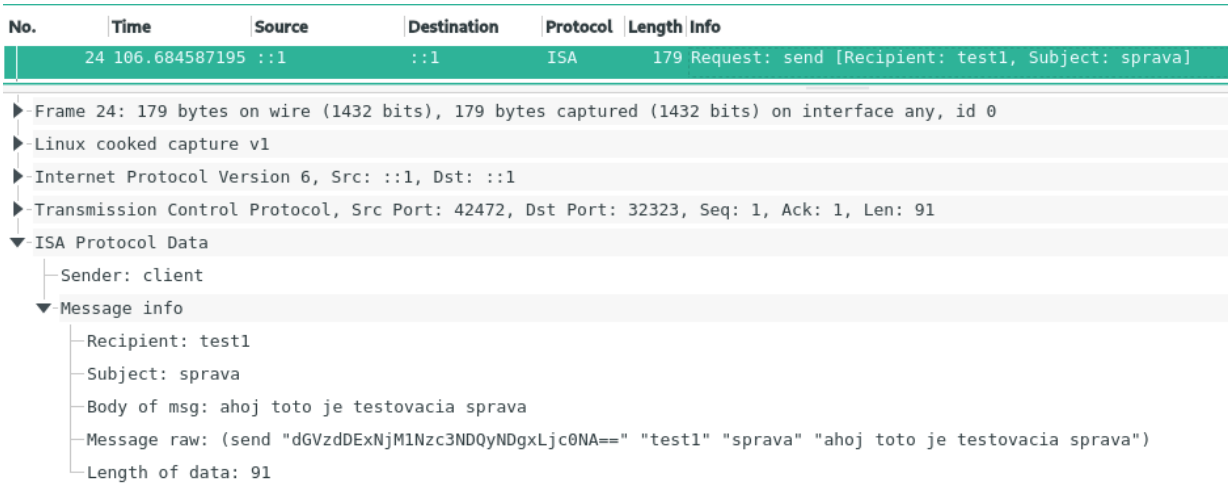
Obrázok 2 - komunikácia klienta so serverom

Na obrázku č. 2 je možné vidieť komunikáciu medzi klientom a serverom. Klient posiela požiadavky s jednotlivými príkazmi serveru, a následne server posiela odpoveď klientovi. Základné informácie o pakete sú zobrazené v stĺpci s názvom `Info`. Kde sa nachádza o aký typ správy sa jedna, ak ide o požiadavku od klienta zobrazí sa tam `Request` a daný príkaz. Pri príkaze `send` sa vypíše kto je príjemca správy a aký je predmet danej správy. Pokiaľ ide o odpoveď servera zobrazí sa tam `Response` a daná odpoveď servera. Pri odpovedi na požiadavku s príkazom `list` sa vypíše koľko správ má užívateľ vo svojej schránke.

`ISA Protocol Data` vždy obsahuje informáciu o tom kto danú správu poslal buď klient alebo server. Pokiaľ ide o server obsahuje dodatočnú informáciu o tom aký je status odpovede `ok` alebo `err`.

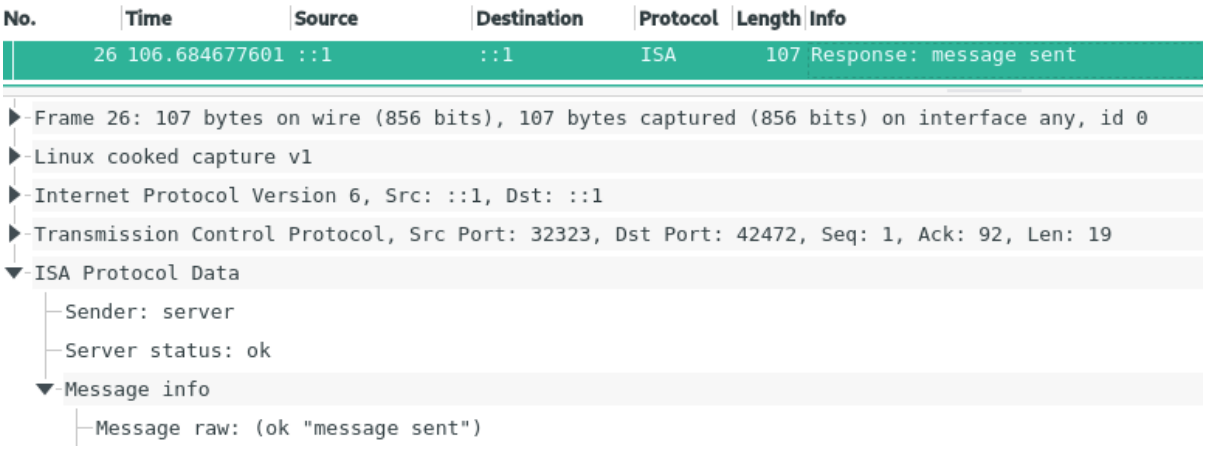
4.2.1. Ukážka príkazu send

Vybral som si jeden príkaz `send`, ktorý detailnejšie opíšem. Pri tomto príkaze som zobrazil najviac informácií preto som sa rozhodol ho bližšie opísať. Požiadavka klienta na príkaz `send` obsahuje v `ISA Protocol Data` informácie ako kto je odosielateľom paketu. Ďalej tam je uvedené kto je príjemca danej správy, aký ma predmet správa a konkrétny obsah správy. Taktiež tam je zobrazené ako vyzerá samotná požiadavka pre príkaz `send`. Nakoniec je uvedená informácia o dĺžke dát, ktoré sa posielajú.



Obrázok 3- požiadavka klienta na príkaz send

Odpoveď servera taktiež obsahuje informáciu kto je odosielateľom paketu a zároveň tu je pridaná informácia o statuse servera. Tento status servera hovorí o tom či požiadavka na server bola úspešná. Na konci `ISA Protocol Data` je taktiež zobrazená odpoveď servera a dĺžka dát, ktoré sa posielajú.



Obrázok 4- odpoveď servera na príkaz send

5. Kompatibilný klient

Poslednou úlohou bolo neimplementovať kompatibilného klienta. Tento klient ma slúžiť ako náhrada referenčného klienta vo forme spustenia aj jednotlivých výstupov. Spúšťanie programu je rovnaké ako pri referenčnom klientovi.

5.1. Implementácia

Kompatibilný klient je implementovaní v jazyku `C` v súbore s názvom `client.c`.

Hlavná časť programu sa nachádza vo funkcií `main()`. Na začiatku dochádza k spracovaniu vstupných argumentov a do premennej `command` sa uloží o aký typ príkazu sa jedná. Následne sa zistí aká bude veľkosť správy pomocou funkcie `length_of_msg()`, aby bolo možné vytvoriť premennú typu `char` o tejto veľkosti. Na základe toho aký bol príkaz zadaný sa vytvorí správa, ktorá bude poslaná na server. Ak sa jedná o príkaz `register` alebo `login` heslo užívateľa sa zakóduje pomocou funkcie `base64_encoding()`. Ďalej sa prostredníctvom funkcie `inet_pton()` zistí aká adresa bola zadaná ako argument, pokiaľ sa nejedna o IPv4 adresu a ani o IPv6 adresu skontroluje či sa náhodou nejedná o doménové meno prostredníctvom pomocnej funkcie `get_ip_address()`. Táto funkcia vráti ip adresu danej domény pokiaľ existuje. Následne sa vytvorí `socket` buď pre IPv4 alebo IPv6. Pokúsi sa pripojiť na daný `socket` pomocou funkcie `connect()`, pokiaľ sa nepodarilo pripojiť na `socket` program sa ukončí a vráti hodnotu 1. Ak pripojenie bolo úspešné pošle sa správa na server pomocou funkcie `send()`, následne sa čaká na odpoveď od servera pomocou funkcie `recv()` a odpoveď sa ukladá do premennej s názvom `all_msg_server`. Nakoniec sa odpoveď servera vypíše na štandardný výstup prostredníctvom pomocnej funkcie `print_server_msg()`.

5.2. Testovanie

Testovanie kompatibilného klienta spočívalo v spúšťaní jednotlivých príkazov pomocou referenčného klienta a mnou vytvoreného kompatibilného klienta. Následne boli kontrolované výstupy oboch klientov. Počas tohto testovania som narazil nato že referenčný klient podporuje aj zadanie domény namiesto ip adresy. Z tohto dôvodu vnímam testovanie ako prínosne a podporu doménových mien som doimplementoval.

6. Použitá literatúra

1. Hadriel Kaplan: Lua/Dissectors [online], rev. 7.2.2015, [vid. 2021-10-11]. Dostupné z: https://wiki.wireshark.org/Lua/Dissectors
2. Guy Harris: Lua/Examples [online], rev. 28.9.2018, [vid. 2021-10-11]. Dostupné z: https://wiki.wireshark.org/Lua/Examples
3. Inet_pton(3) Linux manual page [online], rev. 22.3.2021, [vid. 2021-10-13]. Dostupné z: https://man7.org/linux/man-pages/man3/inet_pton.3.html
4. Getaddrinfo(3) Linux manual page [online], rev. 27.8.2021, [vid. 2021-10-21]. Dostupné z: https://man7.org/linux/man-pages/man3/getaddrinfo.3.html