VYSOKÉ UČENIE TECHNICKÉ V BRNE
FAKULTA INFORMAČNÝCH TECHNOLOGIÍ

Sieťové aplikácie a správa sietí – projekt

Reverse-engineering neznámeho protokolu

5.11.2021

Samuel Valaštín

Obsah

1 Cieľ a dekompozícia projektu

Projekt sa dá ako celok rozdeliť na niekoľko častí. Prvou časťou je analýza a zoznámenie sa s neznámym protokolom pomocou nástroja `Wireshark`. Následne aplikácia nadobudnutých znalostí a implementácia dissectoru v jazyku `Lua`. Poslednou časťou je implementácia klienta. Pre implementáciu klienta neznámeho protokolu bol zvolený jazyk `C++`.

2 Analýza protokolu

Komunikácia medzi klientom a serverom prebieha nad protkolom `TCP`. Po naviazaní spojenia klient odošle požiadavku na server a ten mu následne odpovie. Predvolená adresa komunikácie je `localhost` a predvolený port klienta je `32323`. Klient aj server podporujú `IPv4` a rovnako aj `IPv6`.

2.1 Klient - odoslanie požiadavky

Pri odosielaní dát, ktoré obsahujú dôverné informácie sa využíva kódovanie `Base64` [6]. Toto kódovanie zabezpečuje, že v našom prípade heslá nie sú posielané vo svojej jednoduchej forme, ale sú posielané zakódované v podobe ASCII znakov.

2.2 Klient - formát odoslaných požiadaviek

Pri odosielaní požiadaviek na registráciu a prihlásenie je heslo posielané vo vyššie spomenutom formáte Base64. Pri prihlásení obdrží klient od serveru "login-token", ktorý odosiela serveru ako argument pre požiadavky na odoslanie správy, načítanie správy a odhlásenie.

- Požiadavka na registráciu – `(register "<uživateľské meno>" "<heslo>")`
- Požiadavka na prihlásenie – `(login "<uživateľské meno>" "<heslo>")`
- Požiadavka na zoznam správ – `(list "<token>" )`
- Požiadavka na odoslanie správy – `(send "<token>" "<příjemca>" "<predmet>" "<telo>")`
- Požiadavka na načítanie správy – `(fetch "<token>" <id>)`
- Požiadavka na odhlásenie – `(logout "<token>")`

2.3 Server - formát odoslaných odpovedí

Každá odpoveď serveru začína otváracou zátvorkou, po ktorej nasleduje `status` odpovede. Množina hodnôt, ktoré nadobúda status sa rozdeľuje na `ok` pri správnej požiadavke a `err` pri nesprávnej požiadavke. Väčšina odpovedí (odpovede na – žiadosť o registráciu, žiadosť o odhlásenie a všetky odpovede s chybovým statusom) má formát – `(status "správa")`. Ostatné odpovede serveru, ktoré majú špecifický formát odpovede, sú zhrnuté nižšie.

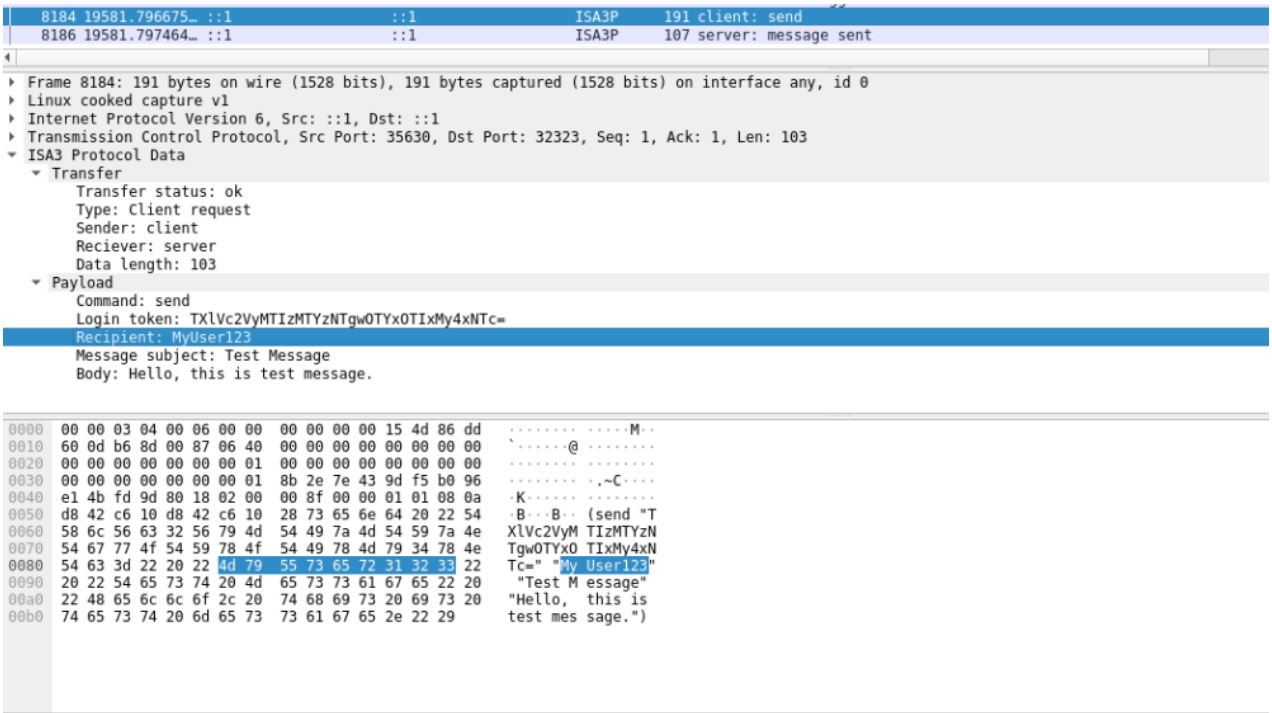
- Odpoveď na úspešné prihlásenie – `(ok "správa" "token")`
- Odpoveď na úspešné načítanie správy – `(ok ("prijemca" "predmet" "telo"))`
- Odpoveď na úspešný zoznam správ – `(ok ([("prijemca" "predmet" "telo")..]))`
- Odpoveď na prázdny zoznam správ – `(ok ())`

3 Návrh a implementácia dissectoru

Dissectoru je priradený `TCP` port `32323` [4]. Ak prichádzajúci paket protokolu neobsahuje celú správu, dissector sa snaží túto udalosť detekovať [1] . V prípade ak správa obsahuje nepárny počet úvodzoviek a zároveň ak nekončí správa ukončovacou zátvorkou `)` , požiada o desegmentáciu ďalšieho paketu s očakávaním, že nebola doručená celá správa. Rovnako prebieha kontrola, ktorá sa snaží rozoznať kľúčové slová, ktorými začína formát správ klienta a serveru. Týmto spôsobom je na tomto porte odfiltrovávaná komunikácia, ktorá nepatrí nášmu protokolu.

3.1 Položky dissectoru

Strom dissectoru je rozdelený na 2 podstromy. Podstrom `Transfer` ukladá informácie o prenose údajov. Detekuje status paketu, typ paketu (žiadosť klienta, odpoveď serveru), odosieľatela paketu, príjmateľa paketu a dĺžku aplikačných dát paketu. Podstrom `Payload` ukladá užitočné informácie protokolu. Pre pakety od klienta detekuje príkaz aj jeho argumenty. Pre pakety od serveru detekuje dáta odpovede, ktoré nie sú ďalej rozdelené. Výnimku má odpoveď serveru na žiadosť o prihlásenie, ktorá ako položku ukladá aj prihlasovací token.



Obrázek 1: Ukážka dissectoru.

4 Návrh a implementácia klienta

Implementácia klienta rozlišuje dve návratové hodnoty. Klient vracia návratovú hodnotu `0` pri úspechu a `1` pri neúspechu.

4.1 Spracovanie argumentov

Argumenty sú spracované ručne bez použitia bežne využívaných knižníc pre spracovanie argumentov. Ešte pred samotným spracovaním sa escapujú špeciálne znaky(spätné lomítko, nový riadok, tabulátor a úvodzovky).
Pri špecifikácii portu prebieha kontrola, či je zadaný port v rozsahu `<0, 65535>`. Pre všetky príkazy je kontrolovaný správny počet argumentov. Zadanie viacerých príkazov pri jednom spustení klienta nie je podporované. Pri nesprávnom zadaní argumentov sa vypíše špecifická nápoveda, ktorá upresňuje použitie daného príkazu a jeho argumenty. Pri nesprávnom zadaní príkazov sa vypíše všeobecná nápoveda.

4.2 Vytvorenie požiadavky

Po escapovaní špecialných znakov a následnom spracovaní a validácii argumentov sa na základe zadaných argumentov vytvorí obsah správy, ktorý bude odoslaný serveru. Pri spracovaní hesla, ktoré sa využíva na registráciu a prihlásenie sa používa vlastná implementácia algoritmu `base64` [3] , ktorá využíva bitové posuny a logické bitové operácie. Pre príkazy, ktoré vyžadujú prácu s "login-tokenom" sa využíva štandardná knižnica jazyka C pre prácu so súbormi.

4.3 Sieťová infraštruktúra

Riešenie podporuje rovnako ako referenčný klient `IPv4` a `IPv6`. Pre identifikáciu a preklad adries sa využíva funkcia `getaddrinfo()` [2] , ktorá vracia zoznam položiek na ktoré sa klient postupným prechodom snaží pripojiť. Po úspešnom pripojení klient posiela dáta s požiadavkou serveru, ktorý mu vráti náležitú odpoveď. Po prijatí celej odpovede sa spojenie so serverom ukončí.

4.4 Spracovanie odpovede

Pre jednoduchšie spracovanie klient najskôr skontroluje status správy, ktorý obdržal od serveru. Pri vypisovaní informácii na štandardný výstup sú špeciálne znaky v dátach pred vypísaním unescapované (tabulátor, nový riadok, úvodzovky). Postupnosť spätných lomítok je pre správny výpis nahradená postupnosťou s polovičným počtom spätných lomítok.

5 Známe limitácie

- Dissector nie je implementovaný ako heuristický dissector [5]. Pri špecifikácii portu iného ako je predvolenený nie je možné dissectorom zachytávať komunikáciu medzi serverom a klientom.

6 Odkazy na referencie

Reference

[1] A Mutable Log: Dealing with segmented data in a Wireshark dissector written in Lua. [online], 2013, Dostupné z: https://tewarid.github.io/2013/05/21/dealing-with-segmented-data-in-a-wireshark-dissector-written-in-lua.html, navštívené 2021-10-20.

[2] GNU: getaddrinfo(3) – Linux manual page. [online], 2021, Dostupné z: https://man7.org/linux/man-pages/man3/getaddrinfo.3.html, navštívené 2021-10-24.

[3] KUMAR, A.: What Is Base64 Encoding? [online], 2001, Dostupné z: https://levelup.gitconnected.com/what-is-base64-encoding-4b5ed1eb58a4, navštívené 2021-10-22.

[4] SUNLAND, M.: Creating a Wireshark dissector in Lua - part 1 (the basics). [online], 2017, Dostupné z: https://mika-s.github.io/wireshark/lua/dissector/2017/11/04/creating-a-wireshark-dissector-in-lua-1.html, navštívené 2021-10-17.

[5] SUNLAND, M.: Creating port-independent (heuristic) Wireshark dissectors in Lua. [online], 2018, Dostupné z: https://mika-s.github.io/wireshark/lua/dissector/2018/12/30/creating-port-independent-wireshark-dissectors-in-lua.html, navštívené 2021-10-18.

[6] TSCHABITSCHER, H.: How Base64 Encoding Works. [online], Dostupné z: https://www.lifewire.com/base64-encoding-overview-1166412, navštívené 2021-10-22.