Vytvořil : Michal Řezník
xrezni28

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Projektová dokumentace

Reverse-engineering neznámeho protokolu

15. listopadu 2021

1. Úvod

Tento dokument slouží jako dokumentace k projektu do předmětu ISA – Reverse-engineering neznámeho protokolu. Budou se zde popisovat stěžejní části mé práce společně s návodem použití.

2. Zadání

Zadáním toho projektu bylo několik úkonů, pomocí kterých jsme na konci byli schopni vytvořit klienta, který bude schopen sloužit jako drop-in náhrada toho referenčního. Samotné body, kterými jsme museli projít byly :

1. Zachytit pomocí referenčního klientu komunikaci se serverem a obeznámit se s formátem komunikace a protokolu.
2. Vytvořit dissector, který bude prezentovat zachycená data ve formě, která je příjemnější pro uživatele.
3. Pomocí nabylých informací vytvořit klienta.

3. Návrh a implementace

Po prostudování aplikace serveru a klienta jsem zjistil, že klient podporuje několik příkazů a zároveň pracuje se souborem login-token, který vždy obsahuje token, který vrací server po přihlášení. Po příkazu odhlášení je následně token odstraněn. Server pak podporuje změnu portu a adresy ve formátu jak IPV4, tak i IPV6.

3.1 Zachycení komunikace

K zachycení samotné komunikace jsem použil wireshark. Zachytával jsem jednotlivé zprávy a zjišťoval, jak samotné příkazy přicházejí a jak server odpovídá. Postupně jsem zjistil, že velkou roli hraje samotný login-token, jež se odesílá pokaždé s příkazem(výjimkou je registrace a přihlášení). Předělání informací z příkazové řádky na řetězec pro server má za úkol klient. Pro příklad:

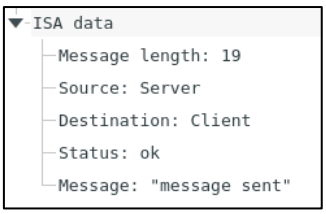
. /client send user subject body *->* (*send "login-token" "user" "subject" "body"*)
. /client fetch 1 -> *(fetch "login-token" 1)*

3.2 Disektor pro wireshark

Ve volbě mezi jazykem Lua a jazykem C k tvorbě disektoru jsem po radách zkušenějších rádců zvolil Lua. Z počátku bylo velmi náročné zorientovat se v přístupu k jednotlivým informacím, které wireshark vrací. Nakonec jsem však našel vše, co jsem považoval za relevantní pro uživatele, a konečný výstup vypadal takto:

. /client send user subject body

Odpověď od serveru :



3.3 Referenční klient

Stejně jako u disektoru, tak i v tomto případě jsem se jako první radil. Nakonec díky tomu, že jazyk C++ nabízí větší komfort v rámci práce s daty a funkcemi, jsem se rozhodl právě pro něj. Práce na referenčním klientu jsem považoval za hlavní úkol samotného projektu, tudíž bych se mu chtěl i nejvíce věnovat v této dokumentaci. Samotný klient bych rozdělil na několik fází:

3.3.1 Práce s argumenty

Pro práci s argumenty po spuštění jsem se rozhodl, kvůli velkému množství kombinací, napsat vlastní funkci *"parse_arguments,"* která má za úkol rozpoznat daný argument, posoudit pomocí pozice ostatních argumentů a jejich počtu správnost spouštění. K posouzení používám několik globálních proměnných, jako například "*ComPosition*", v níž je uložen index argumentu obsahující příkaz(řeší problém "--" switche).

Celá funkce je volána na začátku ve funkci "main" z cyklu, který postupně projde celý spouštěcí příkaz. Celá první část funkce "main" je věnována kontrole. Je zde ještě nastavení základních hodnot pro port a adresu a příprava na samotné spojení.

3.3.2 Sestavení připojení

U této části mi byl velmi nápomocen internet a stránky, které uvádím ve zdrojích, protože mi pomohly nejen vysvětlením, ale i praktickými ukázkami, které vedly k tomu, že mi tato část zabrala nejkratší dobu ze všech zde popsaných.

Pokud sesbírání informací o serveru na dané adrese proběhlo úspěšně, spouštíme cyklus, který má za úkol najít daný výsledek a připojit se k němu.

3.3.3 Sestavení zprávy a přijetí odpovědi

Následuje druhá část, která pracuje s argumenty. Nyní již se zkontrolovaným vstupem sestavíme zprávu ve tvaru jako jsme mohli pozorovat při práci s programem wireshark. Pro příkaz login zde máme funkci "*encode*", která nám před odesláním heslo zakóduje kódováním base64.

Na konci cyklu máme ještě jeden sloužící k přijetí přesné velikosti dané odpovědi, zajišťujeme tak neplýtvání pamětí.

3.3.4 Práce s odpovědí serveru a její výpis

Posledním úkonem, který náš program má za úkol před konečným úklidem dat je převzít zprávu od serveru a předat ji přes klienta do příkazového řádku opět ve tvaru přizpůsobeném pro klienta. Jednalo se o různorodý úkol. Zatím co někdy se jednalo pouze o drobnou úpravu, jindy bylo nutné procházet odpověď znak po znaku, díky čemuž jsem zamezil chybám spojenými s takzvanými escape sekvencemi.

3.3.5 Úklid

Samotný úklid již byl jen tečkou za celým programem. Je zde volána funkce clear(), která uvolňuje paměť z proměnných. Dále uvolňujeme informace z připojení a uzavíráme socket.

4. Spouštění programu

Pro vytvoření spustitelného souboru client je nejprve nutno soubory přeložit pomocí příkazu *make.* Pro následné smazání přebytečných souborů je možno použít *make clean.*

. /client [-h] nebo . /client [--help] => vypíše nápovědu

. /client [-a <adress>] [-p <port>] [--] [command] …

- Switche -a a -p je možno nahradit jejich delší variantou(-adress a -port).
- Po argumentu -- musí následovat příkaz-
- Za třemi tečkami již následují argumenty příkazu, který chceme odeslat na server.

5. Odchylka od zadání

Jedinou odchylkou od zpracování referenčního klienta a mého je reakce, kterou považuji za chybnou a ve své verzi jsem ji proto eliminoval. Jednalo se například různé spouštění nápovědy i při chybě zadaném vstupu. Například:

. /client -aha -> vede ke spuštění nápovědy
./ client -ap -> referenční klient bere jako korektní spuštění

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

6. Návratové kódy klienta

0 = Korektní ukončení programu bez chyb v průběhu.
10 = Nesprávný počet argumentů.
16 = Nesprávný typ dat, či duplicitní deklarace.
18 = Neznámý příkaz.
24 = Problém během kontaktování serveru.
30 = Nedokončena komunikace.

7. Zdroje

Uvádím pouze zdroje, které mi byly opravdovou inspirací, nebo z nichž jsem převzal funkce. Během zjišťování funkcionality jsem využil mnoho stránek, ale žádná z nezmíněných mi nebyla nápomocná jako tyto:

https://mika-s.github.io/wireshark/lua/dissector/2017/11/04/creating-a-wireshark-dissector-in-lua-1.html - Pomoc při tvorbě disektoru.

https://riptutorial.com/cplusplus/example/24000/hello-tcp-client - Pomoc při vytváření tcp komunikace v c++.

https://beej.us/guide/bgnet/html/#client-server-background – Převzata funkce "get_in_addr" pro podporu ipv4 a ipv6.

https://renenyffenegger.ch/notes/development/Base64/Encoding-and-decoding-base-64-with-cpp/ - Převzata funkce pro encoding hesla na base64.