Sieťové aplikácie a správa sietí

Reverse-engineering neznámeho protokolu

Vanessa Jóriová (xjorio00)
14.11.2021

Obsah

1. Úvod

Tento dokument je dokumentáciou k projektu predmetu Sieťové aplikácia a správa sietí – Reverse engineering neznámeho protokolu. Jeho náplňou bolo zachytiť v programe Wireshark komunikáciu referenčného klienta s referenčným serverom. Pre prehľadnejšie zobrazenie komunikácie bolo nutné implementovať pre daný protokol dissector v jazyku Lua. Na základe zistených informácií o formáte protokolu bol následne implementovaný klient v jazyku C++, ktorý slúži ako náhrada referenčného klienta.

Referenčný klient podporuje registráciu, prihlasovanie, odhlasovanie užívateľov, posielanie správ a ich zobrazovanie. Prídavkom je dissector v jazyku Lua, ktorý prehľadne zobrazí významné časti protokolu.

2. Zachytená komunikácia a dissector

Wireshark využíva pre prehľadné zobrazovanie informácií zo zachytených paketov dissectory. Dissectory pre známe protokoly sú už vstavané, v tomto projekte implementovaná komunikácia však využíva vlastný, na mieru vytvorený protokol, vhodný dissector teda bolo nutné doimplementovať.

2.1 Zachytená komunikácia

Prvým krokom bolo zachytenie komunikácie referenčného klienta a serveru. Na danom porte boli zachytené TCP packety zahajujúce a ukončujúce spojenie, rovnako ako TCP packety obsahujúce protokol aplikačnej vrstvy. Ten je textový, jeho štruktúra je teda identifikovateľná voľným okom.

Zachytené boli dva druhy komunikácie – žiadosti zo strany klienta a odpovede zo strany serveru. Všetky správy sú obalené v zátvorke, pričom žiadosti zo strany klienta začínajú požadovanou akciou (register/login/list/send/fetch/logout), odpovede zo strany servera zasa statusom ok alebo err a následnou odpoveďou (správou o úspechu/neúspechu, popr. inými dodatočnými informáciami).

Prenášané hodnoty sú vyznačené úvodzovkami. Úvodzovky, ktoré vystupujú ako súčasť reťazca, nie ako identifikácia začiatku/konca položky, sú od identifikačných úvodzoviek odlíšené tzv. escapovaním, alebo prítomnosťou znaku \ pred úvodzovkami.

Nižšie sú uvedené príklady žiadostí o vykonanie podporovanej akcie a následné odpovede servera.

Register

Súčasťou žiadosti je užívateľské meno a heslo zakódované pomocou kódovania base64 (táto informácia bola poskytnutá v zadaní projektu), odpoveď zase pozostáva zo statusu a príslušnej hláške o úspešnosti a neúspešnosti

```
(register "vaness" "aGVzbG8=")(ok "registered user vaness")
```

```
(register "vaness" "dmFuZXNz")(err "user already registered")
```

Login

Súčasťou žiadosti je opäť užívateľské meno a heslo zakódované pomocou kódovania base64, odpoveď pozostáva popri statuse a hláške v prípade úspechu aj s reťazcom, tzv. *login token*, ktorý predstavuje identifikáciu konkrétneho sedenia.

```
(login "vaness" "emxlaGVzbG8=")(err "incorrect password")
```

```
(login "vaness" "aGVzbG8=")(ok "user logged in" "dmFuZXNzMTYzNjgzMDcyNjQ0MS43MDc1")
```

List

Súčasťou žiadosti je login token, odpoveďou je status a trojica id – odosielať – predmet pre každú prijatú správu, obalená v zátvorke.

```
(list "dmFuZXNzMTYzNjgzMTEyNzE3MS45MTE0")(ok ((1 "vaness2" "test") (2 "vaness2" "pokus rozbit") (3 "vaness2" "newline test") (4 "vaness2" "dolezity oznam")))
```

Send

Súčasťou žiadosti je login token, prijímateľ, predmet a text správy, odpoveďou je status a potvrdzujúca správa.

```
(send "dmFuZXNzMjE2MzY4MzA3Nzg1MjkuMTA2Mg==" "vaness" "dolezity oznam" "Som celkom hladna")(ok "message sent")
```

Fetch

Súčasťou žiadosti je login token a číslo správy, odpoveďou status a v prípade úspechu v zátvorkách obalená trojica odosielateľ – predmet – obsah správy, popr. správa o neúspechu akcie.

```
(fetch "dmFuZXNzMTYzNjgzMTEyNzE3MS45MTE0" 4)(ok ("vaness2" "dolezity oznam" "Som celkom hladna"))
```

```
(fetch "dmFuZXNzMTYzNjgzMTEyNzE3MS45MTE0" 10)(err "message id not found")
```

Logout

Súčasťou žiadosti je session token, odpoveďou je status a správa o úspechu.

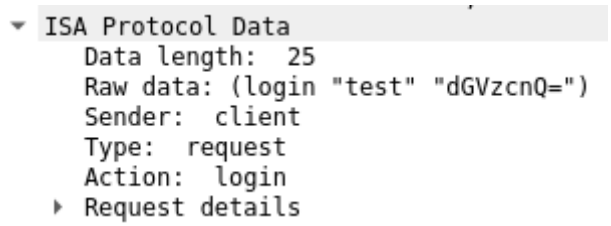
```
(logout "dmFuZXNzMTYzNjgzMTEyNzE3MS45MTE0")(ok "logged out")
```

2.2 Popis návrhu dissectoru

Na základe odchytenej komunikácie bol následne navrhnutý dissector, popis ktorého častí je uvedený nižšie.

2.2.1 Základné informácie

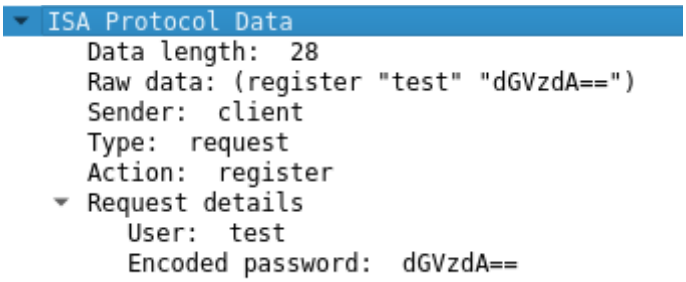
Nasledujúce informácie sú zobrazované pri každom pakete, s miernymi modifikáciami v závislosti od toho, či sa jedná o žiadosť klienta alebo odpoveď servera. Medzi tieto informácie patrí veľkosť dát paketu, „surová“ (nespracovaná) forma protokolu, odosielateľ (klient/server) a typ (žiadosť, odpoveď). V prípade, že sa jedná o odpoveď, je zobrazený aj status (ok, error), v prípade, že sa jedná o žiadosť klienta, sa zobrazí aj typ akcie (send, fetch atď.)



Obrázok 1 - základné informácie zobrazené vo Wiresharku

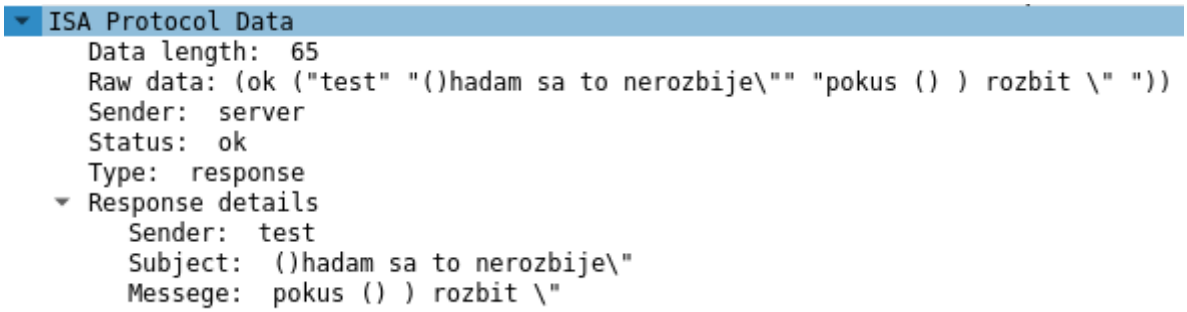
2.2.2 Detaily žiadosti a odpovede

Pre viac informácií má užívateľ možnosť zobraziť detaily žiadosti/odpovede. Tie sa líšia podľa typu danej žiadosti/odpovede. V prípade jednoduchej odpovedi serveru alebo chybovom hlásení sa jedná o zobrazenie danej správy, štandardne sa ale zobrazujú všetky relevantné informácie viditeľné v protokole. V prípade akcie *register* a *login* to je užívateľské meno a zakódované heslo, v odpovedi serveru na žiadosť *login* v prípade úspechu je mimo príslušného hlásenia zobrazený aj priradený tzv. session token, alebo identifikácia daného sedenia.



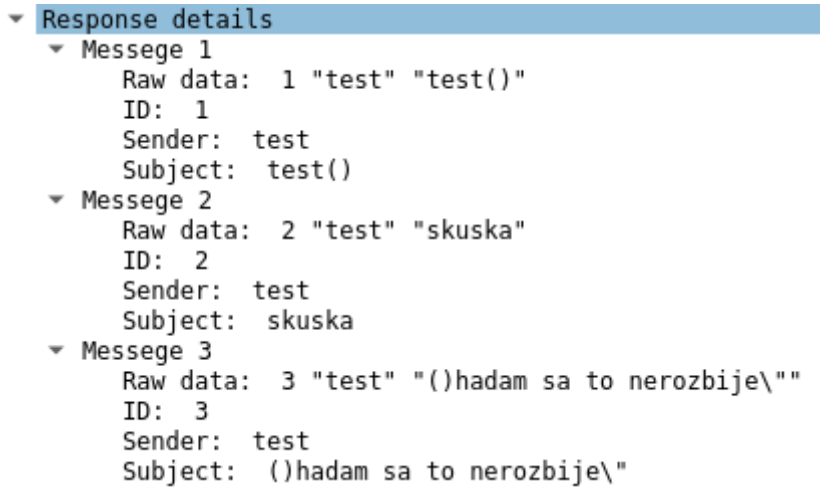
Obrázok 2 - Demonštrácia zobrazenia žiadosti o registráciu

Pri žiadosti *send* je zobrazený session token daného odosielateľa, rovnako ako aj všetky požadované údaje (prijímateľ, predmet, telo správy). Odpoveďou servera je hlásenie o úspechu/neúspechu akcie s príslušným popisom. Žiadosť *fetch* popri session tokene obsahuje aj poradové číslo žiadanej správy, odpoveď od serveru na túto správu v prípade úspechu zahrňuje odosielateľa, predmet a telo správy.



Obrázok 3 - Demonštrácia zobrazenie odpovede serveru na žiadosť fetch

Najkomplexnejšie zobrazenie má odpoveď na *list* v prípade viacerých prijatých správ na strane aktívneho užívateľa. V takom prípade je vytvorený samostatný podstrom pre každú prijatú správu, zobrazujúci odosielateľa a predmet.

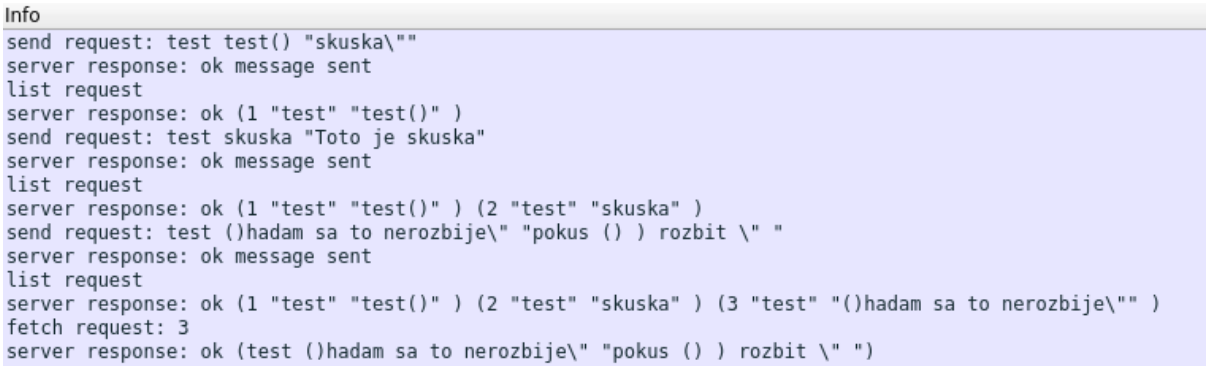


Obrázok 4 - Demonštrácia podstromov pri zobrazení odpovede na žiadosť list

Poslednou akciou je žiadosť *logout* sprevádzaná session tokenom a odpoveď servera ohlasujúca úspech/neúspech.

2.2.3 Informácie pri zobrazovaní všetkých zachytených paketov

Väčšina vstavaných dissectorov v rámci svojho fungovania zobrazuje relevantné informácie aj v stĺpci info pri zobrazení zachytených paketov, ktoré poskytnú užívateľovi prehľad o obsahu bez rozkliknutia daného paketu. Na tento základný prehľad som sa zamerala v svojom riešení aj ja. Zobrazuje sa, či sa jedná o žiadosť alebo odpoveď, rovnako ako aj najrelevantnejšie informácie o komunikácii, ako je demonštrované na obrázku nižšie.



Obrázok 5 - zobrazenie relevantných informácií v stĺpci info

2.3 Implementácia dissectora

Dissector je implementovaný v jazyku Lua. Na zoznámenie sa s tvorbou dissectorov v jazyku Lua som využila odporučené zdroje [3] [4], ďalšími konkrétnymi ukážkami a zrozumiteľným popisom mi pomohla aj stránka [2].

Dissector je priradený ku konkrétnemu portu (port 32323), na ktorom klient a server implicitne bežia, pri nastavení iného portu teda nebude volaný. Hoci je možná aj heuristická detekcia daného protokolu, jej problematika je nad rámec tohto projektu a preto nebola implementovaná.

Spracovanie paketu a rozoznávanie jeho jednotlivých častí je spravené formou konečného automatu. To, či je odosielateľom server alebo klient, je rozhodované podľa úvodu správy (informácia o spracovaní žiadosti *ok/err* značí, že sa jedná o odpoveď serveru). V prípade žiadosti je najprv načítaný typ žiadosti (*send, fetch,...)* a podľa tohto typu načítané a spracované očakávané polia. V prípade odpovede je rozhodovanie o niečo zložitejšie, keďže tá explicitne neobsahuje, o akú odpoveď sa jedná – to je určované podľa formátu danej odpovede. Najprv sa načíta status odpovede a potom sa podľa nasledujúcich znakov vetví, o aký typ odpovede sa jedná – napr. prítomnosť zátvorky za statusom ok indikuje, že sa jedná o response na žiadosť *fetch* alebo *list*, nasledujúce znaky presnejšie upresňujú typ.

2.3.1 TCP reassembly

[3] poukazuje na to, že TCP paket nemusí obsahovať celú žiadosť alebo odpoveď, ale len jej časť. Pri štandardnom využití referenčného klienta a servera sa mi to nestalo, pri vložení „umelo“ veľkého obsahu napr. do tela správy pri žiadosti *send* alebo obmedzení veľkosti zasielaných/prijímaných dát vo vlastnej implementácii klienta sa mi podarilo túto situáciu (rozdelenie requestu/response na viacero packetov) simulovať.

Keďže protokol neobsahuje správu o veľkosti očakávaných dát, je nutné zisťovať dynamicky, či sa jedná o správu celú. Na začiatku sa teda prekontroluje, či je protokol štandardne ukončený (znakom „)“). Keďže i rozsegmentovaná správa sa môže javiť ako korektne ukončená v prípade, že je posledný znak „)“, ale správa pri tom nie je celá, rozhodla som sa implementovať aj dynamické zisťovanie, či je správa celá.

Pokiaľ je správa ukončená nesprávnym znakom, alebo sa počas spracovávania zistí, že správa neobsahuje očakávané znaky/index presiahne veľkosť správy, pomocou *pktinfo.desegment_len = DESEGMENT_ONE_MORE_SEGMENT* sa k aktuálnemu paketu pridá nasledujúci a dissector sa zavolá opäť, tentoraz na oba segmenty súčasne. V praxi to vyzerá tak, že sa dissector zavolá len na posledný segment správy, ktorá však obsahuje všetky predošlé a je korektne zobrazená.

2.4 Limitácie riešenia

Objavené limitácie riešenia súvisia s výnimočnými prípadmi, ktoré môžu nastať počas TCP reassembly. Vo výnimočných prípadoch sa finálny paket korektne nezložil, popr. nastala pri volaní dissectoru chyba. Hoci sa väčšina paketov pri testovaní vyskladala korektne, nemožno vylúčiť prípadné chyby spojené s nasegmentovaním správ do viacerých častí.

Dissector tiež predpokladá, zachytená komunikácia je kompletná – zachytenie len časti správy povedie na chybu.

3. Implementácia kompatibilného klienta

K riešeniu projektu bol k dispozícii referenčný klient, úlohou zasa podľa dostupných informácií zo sekcií uvedených vyššie reimplementovať kompatibilného klienta, ktorý bude slúžiť ako náhrada referenčného. Tento úsek nemá za úlohu popisovať funkcionalitu referenčného ani implementovaného

klienta, nasledujúce odseky teda budú zamerané predovšetkým na popis implementácie, význačné problémy a prípadné zaujímavosti riešenia.

3.1. Implementácia klienta

Hoci je sieťová komunikácia implementovaná v štandardoch jazyka C, klient je implementovaný v jazyku C++, predovšetkým kvôli podpore objektovo orientovaného programovania a stringov. Tvoria ho moduly arg_parser.cpp, client.cpp, request_parser.cpp a response_parser.cpp.

Arg_parser.cpp najprv načíta zadané argumenty, priradí klientovi adresu a port, získa užívateľom požadovanú akciu a jej parametre. Následne je v module client.cpp inicializovaná sieťová komunikácia pripojením sa na server.

Pri študovaní teórie a princípov sieťovej komunikácie som využila [5], pri konkrétnej implementácii vychádzala z [1]. Po úspešnom pripojení sa je serveru zaslaná žiadosť vytvorená modulom request_parser.cpp, ktorý z načítaných argumentov poskladá príslušnú žiadosť. Za zmienku stojí funkcia *void parseString(string * s)*, ktorá implementuje escapovanie špeciálnych znakov z dôvody jednoznačnosti protokolu – konkrétne sa jedná o znaky ‘ \“ ‘, ‘ \\ ‘, ktoré treba „manuálne“ escapnúť (teda pridať pred ne znak ‘ \ ‘), inak by neumožnili korektné spracovanie žiadosti serverom. Escapovaný je aj znak nového riadku.

Následne je obdržaná odpoveď od servera. Tá je v module response_parser.cpp upravená do požadovaného výstupu. Predošlé escapnuté znaky sú funkciou *string * outputParse(string * old_str)* upravené do štandardnej podoby vhodnej pre výstup a klient ukončí svoju činnosť.

Hoci je formát komunikácie predom určený, modul response_parser.cpp je implementovaný tak, aby kontroloval prítomnosť očakávaných znakov a v prípade ich chýbania (alebo neočakávaného formátu) ohlásil chybný formát odpovede.

3.2. Limitácie klienta

Limitácie alebo nekorektné chovanie klienta mi z môjho testovanie nie je známe.

4. Návod na použitie

4.1. Dissector

Po spustení Wiresharku sa zobrazí v ľavom hornom rohu lišta. Poslednou položkou je Help, po rozkliknutí sa objaví menu s poslednou položkou About Wireshark. Po rozkliknutí je v záložke Folders zobrazená cesta k Personal Lua Plugins a Global Lua Plugins, do jednej zo zložiek je nutné premiestniť zdrojový kód *isa.lua*, následne Wireshark reštartovať alebo využiť skratku Ctrl + Shift + L. Po tomto procese by mal byť dissector automaticky volaný na všetky relevantné pakety s komunikáciou klienta a serveru na porte 32323.

4.2. Preklad klienta

K programu je priložený Makefile, ktorý umožní príkazom *make* projekt preložiť. Príkaz *make clean* slúži na zmazanie všetkých binárnych súborov.

4.3. Použitie klienta

Program je použiteľný vo formáte **client [<option> ...] command [<args> ...] ...**

Medzi **<option>** patrí:

- **-a <addr>** alebo **- -address <addr>**
  pričom <addr> predstavuje adresu, na ktorú sa má klient pripojiť
- **-p <port>** alebo **- -port <port>**
  pričom <port> predstavuje port, ku ktorému sa má klient pripojiť
- **-h** alebo **- -help**
  vypíše nápovedu spojenú s používaním programu

Medzi podporované príkazy (**command)** patrí:

- **register <username> <password>**
  registrácia, pričom <username> je zvolené používateľské meno
  a <password> heslo
- **login <username> <password>**
  prihlásenie, pričom <username> je zvolené používateľské meno
  a <password> heslo
- **list**
  výpis prijatých správ
- **send <recipient> <subject> <body>**
  odoslanie správy, pričom <recipient> je užívateľské meno prijímateľa,
  <subject> je predmet správy a <body> obsah správy
- **fetch <id>**
  zobrazenie správy, pričom <id> je identifikačné číslo vyžadovanej správy
- **logout**
  odhlásenie sa

Príklad použitia:

client -a localhost -p 32323 register username secretpassword
pripojenie k adrese localhost a portu 32323
a registrácia používateľa „username“ s heslom „secretpassword“

client -a localhost -p 32323 send username important_subject „Hello.“
pripojenie k adrese localhost a portu 32323
odoslanie používateľovi „username“ správu s predmetom important_subject
a obsahom správy „Hello.“

Literatúra

[1] Brian “Beej Jorgensen” Hall.: Beej’s Guide to Network Programming. [online], november 2020, [vid. 2021-11-1]. Dostupné z: https://beej.us/guide/bgnet/html/#client-server-background

[2] Creating a Wireshark dissector in Lua. [online], november 2017, [vid. 2021-10-20]. Dostupné z: https://mika-s.github.io/wireshark/lua/dissector/2017/11/04/creating-a-wireshark-dissector-in-lua-1.html

[3] Lua dissectors. [online], júl 2015, [vid. 2021-10-20]. Dostupné z: https://wiki.wireshark.org/Lua/Dissectors

[4] Lua dissectors examples. [online], august 2020, [vid. 2021-10-20]. Dostupné z: https://wiki.wireshark.org/Lua/Dissectors

[5] MATOUŠEK, Petr. Síťové aplikace a jejich architektura. Brno: VUTIUM, 2014. ISBN 978-80-214-3766-1.