



# ISA 2021/2022 – Projektová dokumentace

Reverse-engineering neznámého protokolu

Vojtěch Fiala (xfiala61)

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Analýza zachycené komunikace a popis protokolu</b>	<b>2</b>
2.1	Společná část zpráv . . . . .	3
2.2	Příkazy, které program podporuje . . . . .	3
2.3	Chybová hlášení na jednotlivé zprávy . . . . .	4
2.4	Analýza příkazů a úspěšných odpovědí na ně . . . . .	4
2.4.1	Příkaz <code>register</code> . . . . .	5
2.4.2	Příkaz <code>login</code> . . . . .	6
2.4.3	Příkaz <code>logout</code> . . . . .	7
2.4.4	Příkaz <code>send</code> . . . . .	7
2.4.5	Příkaz <code>list</code> . . . . .	9
2.4.6	Příkaz <code>fetch</code> . . . . .	10
<b>3</b>	<b>Implementace Wireshark dissectoru</b>	<b>11</b>
3.1	Popis dissectoru . . . . .	11
3.2	Limitace . . . . .	12
3.3	Formát paketů . . . . .	13
3.3.1	Chybové zprávy . . . . .	13
3.3.2	Jednotlivé příkazy . . . . .	14
3.3.3	Příkaz <code>list</code> . . . . .	15
<b>4</b>	<b>Klient</b>	<b>16</b>

# 1 Úvod

Cílem tohoto projektu bylo vytvoření klienta v jazyce C / C++, který bude komunikovat s poskytnutým serverem, přičemž jeho funkcionality zároveň bude kompatibilní s referenčním klientem. Tato komunikace měla být dále zachycena a pro její popis měl být vytvořen dissector v jazyce C a nebo Lua, fungující jako plugin pro program Wireshark<sup>1</sup>.

## 2 Analýza zachycené komunikace a popis protokolu

Zachytávání a následná analýza komunikace probíhaly za pomoci programu Wireshark. Formát protokolu bude nejlepší vysvětlit názorně na jednom ze zachycených paketů –

0000	00 00 00 00 00 00 00 00	00 00 00 00 86 dd 60 08	.....`.
0010	f1 9f 00 33 06 40 00 00	00 00 00 00 00 00 00 00	...3.@.....
0020	00 00 00 00 00 01 00 00	00 00 00 00 00 00 00 00	.....
0030	00 00 00 00 00 01 7e 43	9a 92 ae 87 b3 02 07 b8	.....~C.....
0040	ba f7 80 18 02 00 00 3b	00 00 01 01 08 0a 69 a2	.....;.....i.
0050	b8 b8 69 a2 b8 b8 28 6f	6b 20 22 6d 65 73 73 61	..i...{o k "messa
0060	67 65 20 73 65 6e 74 22	29	ge sent" )

Obrázek 1: Paket s odpovědí serveru na příkaz `send`

Zajímavá část paketu, kterou server tvoří, je uložena v poslední části TCP paketu, konkrétně jako tzv. „TCP Payload“. Všechny zprávy, poslané jak klientem, tak serverem, jsou v payloadu ohraničeny kulatými závorkami - „(“ a „)“, jak je ostatně možno vidět i na ukázkovém paketu 1.

V tomto konkrétním případě byl klientem na server zaslán příkaz `send`, který slouží k odeslání zprávy specifikovanému uživateli. Zobrazený paket pak reprezentuje odpověď serveru na tento příkaz. Tato odpověď slouží jako potvrzení, jestli byl příkaz úspěšně vykonán, či jestli došlo k chybě. V zachyceném případě byla zpráva úspěšně odeslána (v paketu 1 je hned na začátku TCP Payload zpráva `ok`), server ji zpracoval a zasílá klientovi zpět odpověď s potvrzením, že klientovi se zprávu povedlo odeslat (`message sent`).

<sup>1</sup><https://www.wireshark.org/>

## 2.1 Společná část zpráv

Formáty jednotlivých zpráv se liší. Co je však společné, je již výše zmíněné ohrazení zprávy závorkami. V případě, že se jedná o odpověď ze strany serveru, je také společný formát určení typu odpovědi (`ok` pro úspěch a `err` pro neúspěch – oba typy odpovědí se nacházejí vždy na začátku zprávy, hned za ohraňující závorkou). Příklad úspěchu bylo možno vidět na [paketu 1](#). Příklad neúspěchu lze vidět zde –

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 08 .....`
0010 79 ab 00 3a 06 40 00 00 00 00 00 00 00 00 00 00 y.:@.....
0020 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 7e 43 9a 8e a8 f2 93 38 ec 0e .....~C....8..
0040 39 10 80 18 02 00 00 42 00 00 01 01 08 0a 69 a2 9.....B.....i
0050 58 e8 69 a2 58 e8 28 65 72 72 20 22 69 6e 63 6f X.i.X.(e rr "inco
0060 72 72 65 63 74 20 70 61 73 73 77 6f 72 64 22 29 rrect pa ssword")
```

Obrázek 2: Paket s odpovědí serveru značící neúspěch na příkaz `login`

V tomto případě byl klientem odeslán příkaz `login`, který slouží k přihlášení uživatele. Bylo však zadáno špatné heslo a proto přihlášení nemohlo proběhnout. To lze vidět v těle zprávy - `incorrect password`.

Všechny zprávy odeslané serverem v reakci na příkaz zaslaný klientem mají v případě neúspěchu (`err`) také společný formát chybové zprávy. Ta je vždy, jak lze vidět i na [paketu 2](#), ohrazena uvozovkami (").

## 2.2 Příkazy, které program podporuje

Již jsem zmínil 2 příkazy, se kterými klient a server pracují. Pro doplnění kontextu a přehlednost si však myslím, že by bylo vhodné všechny tyto příkazy a jejich popis vypsát. Jedná se o příkazy následující:

- `register` – Registrace nového uživatele
- `login` – Přihlášení registrovaného uživatele
- `logout` – Odhlášení přihlášeného uživatele
- `send` – Odeslání zprávy konkrétnímu uživateli
- `list` – Zobrazení všech zpráv, které přihlášený uživatel obdržel
- `fetch` – Vypsání těla konkrétní zprávy

## 2.3 Chybová hlášení na jednotlivé zprávy

V předchozí sekci 2.2 byly představeny příkazy, které program podporuje. Již v sekci 2.1 bylo možno vidět paket s chybovým hlášením ze strany serveru. Nyní se na jednotlivé možné chyby v závislosti na příkaz, na který reagují, podíváme podrobněji. Problémy, ke kterým může dojít v reakci na jednotlivé příkazy, jsou následující (Ve formátu `příkaz – chyba`):

- `register – user already registered` (Uživatel s tímto jménem je již zaregistrován.)
- `login – incorrect password` (Nesprávné heslo.)
- `login – unknown user` (Uživatele se zadaným uživatelským jménem se nepovedlo najít.)
- `send – unknown recipient` (Cílový uživatel neexistuje.)
- `fetch – message id not found` (ID zprávy, kterou chtěl uživatel zobrazit, nebylo nalezeno.)
- `fetch – wrong arguments` (Zadáno neplatné číslo zprávy (není kladný integer).)
- `logout, fetch, send, list – incorrect login token` (Žádný uživatel není přihlášen, přesto je přítomen login-token soubor indikující opak. Může nastat v případě, že je server vypnut, aniž by se uživatel odhlásil.)

Chyb, ke kterým může dojít (př. snaha zobrazit si zprávy příkazem `list` v případě, že uživatel není přihlášen), je samozřejmě více. Jsou však už lokálního charakteru, tzn. nedochází k nim v reakci na korektně zadaný příkaz, kdy je vše pro úspěšné odeslání příkazu splněno. Tyto chyby tedy již nepřichází ze strany serveru, ale jsou vypisovány pouze ze strany klienta.

## 2.4 Analýza příkazů a úspěšných odpovědí na ně

Chyby již byly shrnuty a nyní je čas se zaměřit opačným směrem, tedy na úspěchy. Společně s nimi se také podíváme na formát jednotlivých příkazů posílaných na server klientem. Pro názornou ukázkou paketů budou k dispozici vždy paket zaslaný klientem a poté ukázkou odpovědi na tento paket ze strany serveru.

Všechny příkazy, které aplikace podporuje, již byly zmíněny v sekci 2.2 a ve stejném pořadí, v jakém jsou zmíněny tam, budou zpracovány i zde.

Na paketech lze poznat, že se jedná o příkaz zaslaný na server klientem tak, že začátek příkazu neobsahuje reakci `err` ani `ok`, ale naopak název zasílaného příkazu.

### 2.4.1 Příkaz `register`

Jako první se tedy podíváme na příkaz `register` –

```

0000 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 09 .....`
0010 d5 72 00 40 06 40 00 00 00 00 00 00 00 00 00 00 .r.@. @.....
0020 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 9a 88 7e 43 50 79 68 28 1a 90 .....~CPyh(..
0040 23 35 80 18 02 00 00 48 00 00 01 01 08 0a 69 a1 #5.....H.....i.
0050 eb 11 69 a1 eb 11 28 72 65 67 69 73 74 65 72 20 ..i...(r egister
0060 22 75 73 65 72 6e 61 6d 65 22 20 22 63 47 46 7a "usernam e" "cGFz
0070 63 77 3d 3d 22 29 cw==" )

```

Obrázek 3: Paket zaslaný klientem po zadání příkazu `register`

Na paketu lze vidět zaslaný příkaz (`register`), jenž je následován v uvozovkách jménem uživatele (v tomto případě se uživatel jmenuje doslova `username`). Poté je uvedeno heslo, které si uživatel zvolil. Toto heslo však není v klasické podobě, ale je zakódováno algoritmem `base64`<sup>2</sup>.

Obě tyto položky, tedy uživatelské jméno i heslo, mají společné, že se nacházejí uvnitř uvozovek, stejně jako tělo např. odpovědi serveru v případě neúspěchu uživatelem zadaného příkazu (viz sekce 2.1). Tohoto faktu je později využito jak při tvorbě dissectoru, tak při tvorbě klienta.

Přejďeme k odpovědi na příkaz `register` –

```

0000 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 05 .....`
0010 3c 40 00 3f 06 40 00 00 00 00 00 00 00 00 00 00 <@.?.@.....
0020 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 7e 43 9a 88 1a 90 23 35 50 79 .....~C...#5Py
0040 68 48 80 18 02 00 00 47 00 00 01 01 08 0a 69 a1 hH.....G.....i.
0050 eb 11 69 a1 eb 11 28 6f 6b 20 22 72 65 67 69 73 ..i...(o k "regis
0060 74 65 72 65 64 20 75 73 65 72 20 75 73 65 72 6e tered us er usern
0070 61 6d 65 22 29 ame")

```

Obrázek 4: Paket zaslaný serverem jako reakce na příkaz `register`

<sup>2</sup><https://en.wikipedia.org/wiki/Base64>

V odpovědi můžeme vidět očekávanou reakci serveru `ok`, tedy registrace proběhla korektně. Za ní následuje, opět v uvozovkách, odpověď serveru `registered user username`. Ta potvrzuje, že registrace uživatele `username` byla úspěšná.

## 2.4.2 Příkaz `login`

```

0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 07 .....`
0010 df 11 00 3d 06 40 00 00 00 00 00 00 00 00 00 00 ...=@.....
0020 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 9a 90 7e 43 98 df 96 5c b4 22 .....~C...\
0040 65 44 80 18 02 00 00 45 00 00 01 01 08 0a 69 a2 eD....E.....i
0050 5f 5a 69 a2 5f 5a 28 6c 6f 67 69 6e 20 22 75 73 _Zi_Z(l ogin "us
0060 65 72 6e 61 6d 65 22 20 22 63 47 46 7a 63 77 3d _ername" "cGFzcw=
0070 3d 22 29      =")

```

Obrázek 5: Paket zaslaný klientem po zadání příkazu `login`

Jako první můžeme vidět název zaslaného příkazu, v tomto případě tedy `login`. Následuje v uvozovkách uvedené jméno uživatele, který má být přihlášen (stejně jako u příkazu `register` v sekci 2.4.1 se jedná o uživatele `username`). Po jméně uživatele následuje uživatelské heslo, které je zakódováno algoritmem `base64` stejně jako u příkazu `register`, jak můžeme vidět v paketu 3.

Nyní se podíváme na odpověď serveru na tento paket. –

```

0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 08 .....`
0010 ac 83 00 5c 06 40 00 00 00 00 00 00 00 00 00 00 ...\.@.....
0020 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 7e 43 9a 90 b4 22 65 44 98 df .....~C..."eD
0040 96 79 80 18 02 00 00 64 00 00 01 01 08 0a 69 a2 .y....d.....i
0050 5f 5b 69 a2 5f 5a 28 6f 6b 20 22 75 73 65 72 20 _[i_Z(o k "user
0060 6c 6f 67 67 65 64 20 69 6e 22 20 22 64 58 4e 6c _logged i n" "dXNl
0070 63 6d 35 68 62 57 55 78 4e 6a 4d 32 4d 7a 41 35 cm5hbWUx NjM2MzA5
0080 4f 44 51 79 4f 44 41 30 4c 6a 63 32 4d 77 3d 3d ODQyODAw Ljc2Mw==
0090 22 29      ")

```

Obrázek 6: Paket zaslaný serverem jako reakce na příkaz `login`

Odpověď je opět `ok`, potvrzující, že přihlášení proběhlo v pořádku. Následuje zpráva serveru, `user logged in`, potvrzující úspěšné přihlášení. Poté následuje token, který reprezentuje přihlášeného uživatele. Tento token se také, stejně jako všechny části zprávy (mimo stav odpovědi), nachází v uvozovkách. Zde je vhodné dodat, že tento token je dále zpracován klientem a umístěn do souboru „`login-token`“, který je využíván dalšími příkazy.





Paket s ním vypadá následovně –

```

0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 0b .....`
0010 37 19 00 c6 06 40 00 00 00 00 00 00 00 00 00 7....@..
0020 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 9a 92 7e 43 07 b8 ba 51 ae 87 .....~C...Q..
0040 b3 02 80 18 02 00 00 ce 00 00 01 01 08 0a 69 a2 .....i.
0050 b8 b8 69 a2 b8 b8 28 73 65 6e 64 20 22 64 58 4e ..i...(s end "dXN
0060 6c 63 6d 35 68 62 57 55 78 4e 6a 4d 32 4d 7a 41 lcm5hbWU xNm2MzA
0070 35 4f 44 51 79 4f 44 41 30 4c 6a 63 32 4d 77 3d 50DQy0DA 0Ljc2Mw=
0080 3d 22 20 22 75 73 65 72 6e 61 6d 65 22 20 22 74 =" "user name" "t
0090 65 73 74 6d 61 69 6c 22 20 22 41 20 6c 6f 6e 67 estmail" "A long
00a0 20 6d 65 73 73 61 67 65 20 77 69 74 68 20 61 20 message with a
00b0 5c 5c 6e 20 61 6e 64 20 5c 5c 74 20 61 6e 64 20 \\n and \\t and
00c0 5c 5c 20 61 6e 64 20 5c 22 20 61 6e 64 20 28 29 \\ and \ " and (
00d0 29 29 28 29 5c 6e 61 6e 64 20 6e 6f 74 20 74 6f ))())\nan d not to
00e0 20 66 6f 72 67 65 74 20 61 6e 20 61 63 74 75 61 forget an actua
00f0 6c 20 6e 65 77 6c 69 6e 65 21 22 29 l newlin e!")

```

Obrázek 9: Paket zaslaný klientem po zadání příkazu `send`

Jako první vidíme, jak je již zvykem, název příkazu. Následuje token reprezentující aktuálního přihlášeného uživatele (odesílatele) a po něm můžeme vidět jméno příjemce, komu je zpráva určena (pozn. v tomto specifickém případě posílá uživatel `username` zprávu sám sobě).

Následuje předmět zprávy, který je zde zvláštní tím, že se v něm nacházejí uvozovky, které nejsou uvozovkami ohraničujícími jednotlivé části paketu. Byly vloženy jako součást předmětu zprávy, na což klient reaguje tak, že před uvozovky vloží zpětné lomítko.

Podobný jev můžeme vidět i v samotné zprávě, ve které se nachází znak `\n` reprezentující nový řádek. Ten však již před sebou žádné zdvojené zpětné lomítko nemá. Ten je klientem v případě vypsání zprávy příkazem `fetch` interpretován jako nový řádek, stejně jako uvozovky s lomítkem před nimi jsou interpretovány jako pouze uvozovky.

Více se problematikou nahrazování speciálních znaků zabývá sekce 4.

Nyní již k odpovědi na příkaz `send` –

```

0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 08 .....`
0010 f1 9f 00 33 06 40 00 00 00 00 00 00 00 00 00 ...3..@..
0020 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 7e 43 9a 92 ae 87 b3 02 07 b8 .....~C .....
0040 ba f7 80 18 02 00 00 3b 00 00 01 01 08 0a 69 a2 .....; .....i.
0050 b8 b8 69 a2 b8 b8 28 6f 6b 20 22 6d 65 73 73 61 ..i...(o k "messa
0060 67 65 20 73 65 6e 74 22 29 ge sent" )

```

Obrázek 10: Paket zaslaný serverem jako reakce na příkaz `send`

Odpověď serveru je, na rozdíl od zprávy ze strany klienta, velmi jednoduchá - jedná se o pouhé potvrzení typu ok a zprávu message sent, která značí, že se zprávu klientovi povedlo úspěšně odeslat.

## 2.4.5 Příkaz list

Příkaz list je, co se délky týče, přesným opakem příkazu send ve smyslu, že zatímco u příkazu send byla dlouhá zpráva ze strany klienta, zde může být dlouhá odpověď ze strany serveru (záleží na počtu odeslaných zpráv). Paket při tomto příkazu vypadá následovně –

```

0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 09 .....`
0010 4a a3 00 4d 06 40 00 00 00 00 00 00 00 00 00 00 J..M.@.....
0020 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 00 01 9a 96 7e 43 d4 82 f0 07 79 c3 .....~C...y
0040 f6 30 80 18 02 00 00 55 00 00 01 01 08 0a 69 a3 .0....U.....i
0050 ea 00 69 a3 ea 00 28 6c 69 73 74 20 22 64 58 4e ..i...{l ist "dXN
0060 6c 63 6d 35 68 62 57 55 78 4e 6a 4d 32 4d 7a 41 lcm5hbWU xNm2MzA
0070 35 4f 44 51 79 4f 44 41 30 4c 6a 63 32 4d 77 3d 50DQy0DA 0Ljc2Mw=
0080 3d 22 29      =")

```

Obrázek 11: Paket zaslaný klientem po zadání příkazu list

Po názvu příkazu zde můžeme vidět akorát token reprezentující přihlášeného uživatele. Zajímavější je odpověď na tento příkaz, nacházející se níže –

```

0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 0c .....`
0010 db ad 00 40 06 40 00 00 00 00 00 00 00 00 00 00 ...@.@.....
0020 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 00 01 7e 43 9a 96 79 c3 f6 30 d4 82 .....~C..y..0..
0040 f0 34 80 18 02 00 00 48 00 00 01 01 08 0a 69 a3 .4....H.....i
0050 ea 00 69 a3 ea 00 28 6f 6b 20 28 28 31 20 22 75 ..i...{o k ((l "u
0060 73 65 72 6e 61 6d 65 22 20 22 74 65 73 74 6d 61 sername" "testma
0070 69 6c 22 29 29 29      il"))

```

Obrázek 12: Paket zaslaný serverem jako reakce na příkaz list

V tomto případě je zpráva relativně krátká (proto, aby se vůbec dala celá v rozumném rozlišení zobrazit) a server nám tentokrát vrací pouze 1 zprávu, která byla uživateli zaslána. Jedná se o stejnou zprávu, kterou si uživatel sám poslal v sekci 2.4.4.

Odpověď serveru začíná reakcí ok, která je následována dvěma závorkami. První z těchto závorek značí začátek vypisování jednotlivých zpráv a druhá pak

odděluje začátek každé zprávy. První částí konkrétní zprávy je její identifikační číslo – v tomto případě je zpráva pouze 1, tedy její ID je 1.

Následuje již klasicky v uvozovkách nejprve název odesílatele a poté předmět zprávy. Zpráva je pak závorkou zakončena. V případě, že by zpráv bylo víc, by následovala mezera a opět závorkami ohraničená další zpráva. Zde však další zpráva není a tedy následují další 2 závorky, které celý paket zakončují.

Bylo by ještě vhodné zde zmínit, že v případě, že uživateli nikdo žádnou zprávu nikdy nenapsal, bude odpověď serveru stále `ok` a celý payload bude následující:  
(`ok` ) ) ).

## 2.4.6 Příkaz `fetch`

Posledním příkazem je příkaz `fetch`, který zobrazuje jednotlivé zprávy podle ID získaného na základě odpovědi serveru na příkaz `list`. Jeho obsah vypadá následovně –

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 04 .....`
0010 f4 15 00 50 06 40 00 00 00 00 00 00 00 00 00 ...P.@...
0020 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 9a 9a 7e 43 fa e1 f0 ec 7c 8d .....~C...|
0040 45 06 80 18 02 00 00 58 00 00 01 01 08 0a 69 a4 E.....X.....i
0050 04 6e 69 a4 04 6e 28 66 65 74 63 68 20 22 64 58 .ni..n(f fetch "dX
0060 4e 6c 63 6d 35 68 62 57 55 78 4e 6a 4d 32 4d 7a Nlcm5hbW UxNjM2Mz
0070 41 35 4f 44 51 79 4f 44 41 30 4c 6a 63 32 4d 77 A5ODQyOD A0Ljc2Mw
0080 3d 3d 22 20 31 29 == " 1)
```

Obrázek 13: Paket zaslaný klientem po zadání příkazu `fetch`

Požadavek na server je v tomto případě podobný jako u příkazu `list` (viz [2.4.5](#)), kdy je paket tvořen pouze zadaným příkazem, uživatelským tokenem, ale na rozdíl od příkazu `list` je zde za uživatelským tokenem ještě číslo – toto číslo udává ID zprávy, kterou by si klient chtěl přečíst. V tomto případě se jedná o jedinou zprávu, kterou uživatel obdržel, a to o zprávu s ID 1, což je stejná zpráva, jaká byla poslána v sekci [2.4.4](#).

Následuje odpověď serveru –

```
0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 0b .....`
0010 07 d9 00 9f 06 40 00 00 00 00 00 00 00 00 00 00 .....@.....
0020 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 7e 43 9a 9a 7c 8d 45 06 fa e1 .....~C..|.E...
0040 f1 1c 80 18 02 00 00 a7 00 00 01 01 08 0a 69 a4 .....i.
0050 04 6f 69 a4 04 6e 28 6f 6b 20 28 22 75 73 65 72 .oi..n(o k ("user
0060 6e 61 6d 65 22 20 22 74 65 73 74 6d 61 69 6c 22 name" "t estmail"
0070 20 22 41 20 6c 6f 6e 67 20 6d 65 73 73 61 67 65 "A long message
0080 20 77 69 74 68 20 61 20 5c 5c 6e 20 61 6e 64 20 with a \\n and
0090 5c 5c 74 20 61 6e 64 20 5c 5c 20 61 6e 64 20 5c \\t and \\ and \
00a0 22 20 61 6e 64 20 28 29 29 29 28 29 5c 6e 61 6e " and () )())\nan
00b0 64 20 6e 6f 74 20 74 6f 20 66 6f 72 67 65 74 20 d not to forget
00c0 61 6e 20 61 63 74 75 61 6c 20 6e 65 77 6c 69 6e an actua l newlin
00d0 65 21 22 29 29 e!"))
```

Obrázek 14: Paket zaslaný serverem jako reakce na příkaz `fetch`

V odpovědi můžeme vidět status, jenž je opět `ok` a je následován závorkou značící začátek zprávy (podobně jako u příkazu `list`). Uvnitř závorek poté vidíme uvozovkami ohraničené jednotlivé položky. První z nich je jméno odesílatele, následuje předmět zprávy a nakonec tělo zprávy.

## 3 Implementace Wireshark dissectoru

### 3.1 Popis dissectoru

Nyní přichází na řadu popis dissectoru, který pakety popsané v minulé sekci [2](#) rozděluje na jednotlivé části pro lepší uživatelskou přívětivost.

Chtěl bych zde také zmínit, že dissector je inspirován [\[4\]](#).

Dissector naslouchá na portu **32323**, což je výchozí port používaný serverem v případě, že uživatel nespecifikuje port jiný.

Možností jak udělat, aby dissector fungoval bez ohledu na port, je udělat jej heuristickým způsobem, kdy bude každý paket kontrolován, jestli odpovídá zadaným požadavkům. Tento přístup jsem však při implementaci nezvolil z důvodu, že nejenom že ztěžuje implementaci, ale je i náročnější na systémové prostředky (Musí projít každý paket, nejen ty na určitém portu).

Dissector tedy naslouchá pouze na zmíněném portu **32323**, kde každé zprávě přiřadí ve Wiresharku protokol **ISA**, podle kterého poté lze pakety filtrovat a zobrazit si tedy pouze ty pakety, které obsahují užitečné informace (př. pro tvorbu klienta / dissectoru).

Následuje ukázka zachycených paketů s rozlišením ISA protokolu

1	0.000000	::1	::1	TCP	94	39558 → 32323 [SYN]
2	0.000008	::1	::1	TCP	94	32323 → 39558 [SYN]
3	0.000014	::1	::1	TCP	86	39558 → 32323 [ACK]
4	0.000107	::1	::1	ISA	133	39558 → 32323 [PSH]
5	0.000110	::1	::1	TCP	86	32323 → 39558 [ACK]
6	0.003416	::1	::1	ISA	115	32323 → 39558 [PSH]
7	0.003419	::1	::1	TCP	86	39558 → 32323 [ACK]
8	0.003435	::1	::1	TCP	86	32323 → 39558 [FIN]
9	0.003521	::1	::1	TCP	86	39558 → 32323 [FIN]
10	0.003526	::1	::1	TCP	86	32323 → 39558 [ACK]

Obrázek 15: Pakety zachycené Wiresharkem s rozlišením ISA protokolu

Komunikace probíhá formou TCP paketů. Jak můžeme vidět, nejprve mezi klientem a serverem proběhne tzv. 3-Way handshake. Poté je klientem odeslán na server požadavek, na který poté pošle server klientovi zpět odpověď.

Zprávy spadající do „ISA“ protokolu jsou nejprve dissektorem rozlišeny na základě zprávy nacházející se hned za hraniční závorkou. Podle toho poté probíhá klasifikace na požadavek ze strany klienta nebo odpověď ze strany serveru.

Odpovědi jsou dále děleny podle typu – ok a err.

Jednotlivé zprávy jsou následně zpracovány primárně funkcí, která v těle zprávy najde v uvozovkách (a nebo v závorkách) jednotlivé položky, které potom přidává do stromové struktury obsahu paketu.

V případě, že byla originální zpráva moc dlouhá a server ji musel rozdělit mezi vícero paketů, pokusí se ji dissektor znovu spojit. Bohužel se mi nepovedlo odhalit podle čeho server určuje na kolik paketů bude zpráva rozdělena, a tedy jsem určil, že žádost o spojení více paketů do jednoho proběhne pouze v případě, kdy je paket delší než 32731 bytů. Dissektor se tedy u paketů delších než tato hodnota pokusí zpracovat i další segment. Kompletní výpis informací o paketu poté proběhne v posledním přijatém paketu. O problémech tohoto řešení se dále zmiňuje následující sekce [3.2](#).

## 3.2 Limitace

Jak už to tak v životě bývá, nic není dokonalé, a ani můj dissektor není výjimkou a má tedy svoje limitace. Hlavním „nedostatkem“ jsou případy, kdy je Wireshark zachytávání spuštěno uprostřed komunikace - v takovém případě dissektor nerozpozná, o jaké zprávy se jedná, a pokusí se tom uživatele informovat v popisu Payloadu. Další „limitací“ by se dal označit případ, kdy byla originální zpráva moc

dlouhá a nevešla se do jednoho paketu, jak je již popsáno o sekci výše. Problémem může být, že na poskytnutém virtuálním stroji Wireshark tyto rozdělené pakety neoznačuje pod ISA protokolem, ale jako TCP (přestože po zadání vyhledávacího filtru „ISA“ se tyto pakety zobrazí). V tomto případě, tedy kdy je zpráva rozdělena na více paketů, ale může dojít k ještě jednomu problému a to tehdy, když je zpráva rozdělena a velikost dílčích zpráv nepřekročí již zmíněnou hranici 32731 bytů. V takovémto případě dissector pouze informuje, že paket může být součástí větší zprávy.

### 3.3 Formát paketů

V této sekci se zaměřím na zpracování jednotlivých paketů dissectorem. Zde by bylo vhodné zmínit položky, které jsou pro pakety, nehledě na typ, společné. Jako první bych zmínil (mimo zprávy rozdělené na vícero paketů, viz 3.2) pole `Type`. To může nabývat 2 hodnot – `Response` a `Request`. U zpráv začínajících `err` nebo `ok` se bude vždy jednat o odpověď ze strany serveru, tedy typ bude `Response`. V ostatních případech bude na prvním místě zprávy příkaz, který zadal uživatel a bude se tedy jednat o `Request`.

Všechny zprávy, které jsou určeny jako patřící do ISA protokolu, mají poslední položku vždy `Length`, která určuje velikost `Payload` části paketu v bytech.

Následující položky jsou pak společné pro zprávy typu `Response`, tzn. zprávy zaslané serverem klientovi. Jedná se o `Response status` a `Response to Opcode`. `Response status` Může nabývat 2 hodnot – `ok` a `err`. `Response to Opcode` pak určuje, na jaký příkaz server odpovídá. To je určeno na základě formátu zprávy v paketu.

#### 3.3.1 Chybové zprávy

Nyní bych se rád zaměřil na zobrazení error zpráv, o kterých pojednává sekce 2.3.

Zpracování chyb ukážu na paketu, kdy server odpovídá klientovi na příkaz `login`, kdy se klient pokusil přihlásit na neexistujícího uživatele.

Můžeme vidět stromovou strukturu, která je společná pro všechny zachycené zprávy, kdy jsou jednotlivé položky schované ve stromu „ISA Protocol Payload“.

Paket je typu `Response`, tedy odeslaný serverem klientovi v reakci na požadavek klienta.



ISA Protocol Payload		
Type: Response		
Response status: err		
Response to Opcode: login		
Message: unknown user		
Length: 20		
0000	00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 0e	.....`.
0010	a7 04 00 34 06 40 00 00 00 00 00 00 00 00 00 00	...4@.....
0020	00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00	.....
0030	00 00 00 00 00 01 7e 43 9a 8c 82 ab 5f 90 ee 2b	.....~C...+.
0040	2d ce 80 18 02 00 00 3c 00 00 01 01 08 0a 69 a2	-.....<...i.
0050	49 1e 69 a2 49 1e 28 65 72 72 20 22 75 6e 6b 6e	I·i·I·(e rr "unkn
0060	6f 77 6e 20 75 73 65 72 22 29	own user ")

Obrázek 16: Odpověď serveru na pokus o přihlášení jako neexistující uživatel

Následuje `Response status`, který bude v tomto případě samozřejmě `err`. Jelikož se jedná o odpověď serveru, je pro uživatele také vhodné vědět o odpověď na jaký příkaz se jedná. V tomto případě byl příkaz `login`.

Poté je vypsán obsah chybové zprávy, což je tentokrát, jak již bylo zmíněno, `unknown user` – uživatel se pokusil přihlásit pod neexistujícím jménem.

Poslední položkou je `Length`, která je zde 20 bytů.

Formát chybových zpráv je pro všechny chyby stejný a tedy se jimi dále již nebudu zabývat.

### 3.3.2 Jednotlivé příkazy

Formát jednotlivých paketů je k vidění v sekci 2.4.

Na ukázkou výpisu dissectoru je zde k dispozici paket s žádostí o registraci.

ISA Protocol Payload									
Type: Request									
Operation code: register									
Username: username									
Obfuscated password: cGFzcw==									
Length: 32									
0000	00	00	00	00	00	00	00	00	00
0010	d5	72	00	40	06	40	00	00	00
0020	00	00	00	00	00	01	00	00	00
0030	00	00	00	00	00	01	9a	88	7e
0040	23	35	80	18	02	00	00	48	00
0050	eb	11	69	a1	eb	11	28	72	65
0060	22	75	73	65	72	6e	61	6d	65
0070	63	77	3d	3d	22	29			

Obrázek 17: Žádost klienta o registraci uživatele

U příkazu `register` se jedná samozřejmě o žádost, tedy typ `Request`. Zadaný příkaz byl `register`, dále dissector vypíše uživatelské jméno, zakódované heslo a délku payload části paketu.

Odpověď obsahuje typ, status, příkaz na který odpovídá, zprávu a délku zprávy.

Ostatní příkazy jsou pak zpracovány obdobně jako paket výše – jednotlivé informace jsou vypsány srozumitelně pod sebou ve stromové struktuře.

### 3.3.3 Příkaz `list`

Za zmínku nicméně stojí příkaz `list`, jehož zpracování bylo ze všech nejnáročnější.

ISA Protocol Payload									
Type: Request									
Operation code: list									
User Token: dXNlcm5hbWUxNjM2MzA5ODQyODA0Ljc2Mw==									
Length: 45									
0000	00	00	00	00	00	00	00	00	00
0010	4a	a3	00	4d	06	40	00	00	00
0020	00	00	00	00	00	01	00	00	00
0030	00	00	00	00	00	01	9a	96	7e
0040	f6	30	80	18	02	00	00	55	00
0050	ea	00	69	a3	ea	00	28	6c	69
0060	6c	63	6d	35	68	62	57	55	78
0070	35	4f	44	51	79	4f	44	41	30
0080	3d	22	29						

Obrázek 18: Žádost klienta o vypsání zpráv

V žádosti o vypsání zpráv je pouze typ (`Request`), jelikož se jedná o žádost ze strany klienta, příkaz (v tomto případě `list`), uživatelský token a délka textové části paketu.



Podstatně zajímavější a „hezčí“ částí je pak ale odpověď serveru, která je k vidění níže.

```

ISA Protocol Payload
Type: Response
Response status: ok
Response to Opcode: list
Message ID: 1
Message sender: username
Message subject: testmail
Message ID: 2
Message sender: username
Message subject: helloworld
Message ID: 3
Message sender: username
Message subject: hello2
Length: 84
0000 00 00 00 00 00 00 00 00 00 00 00 00 86 dd 60 0d .....`
0010 2b 96 00 74 06 40 00 00 00 00 00 00 00 00 00 00 +..t.@.....
0020 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 01 7e 43 9a a6 3e 9e 97 7d dc 0c .....~C..>..}..
0040 6e 6d 80 18 02 00 00 7c 00 00 01 01 08 0a 69 b4 nm....|.....i.
0050 3c e1 69 b4 3c e1 28 6f 6b 20 28 28 31 20 22 75 <.i.<.(o k ((1 "u
0060 73 65 72 6e 61 6d 65 22 20 22 74 65 73 74 6d 61 sername" "testma
0070 69 6c 22 29 20 28 32 20 22 75 73 65 72 6e 61 6d il") (2 "usernam
0080 65 22 20 22 68 65 6c 6c 6f 77 6f 72 6c 64 22 29 e" "hell oworld")
0090 20 28 33 20 22 75 73 65 72 6e 61 6d 65 22 20 22 (3 "use rname" "
00a0 68 65 6c 6c 6f 32 22 29 29 29 hello2") ))

```

Obrázek 19: Odpověď serveru se zprávami, které uživatel obdržel

V odpovědi můžeme vidět 5 různých zpráv, které uživateli byly napsány (V tomto případě uživatelem samotným). Jednotlivé zprávy jsou pak vypsány pod sebou, kdy je vypsáno ID každé zprávy, její odesílatel a předmět. Vše je pak zakončeno výpisem délky textové části paketu.

## 4 Klient

Poslední sekci dokumentace je popis klienta. Jelikož však není dle zadání důvod popisovat funkcionality referenčního klienta, zmíním zde jenom zdroje, které jsem při jeho tvorbě použil a také pár drobných odlišností.

Co se týče převzatého obsahu, konkrétně se jedná o funkci pro načtení obsahu souboru do stringu (viz [1]), která je v mém programu použita pro načtení obsahu již zmíněného souboru `login-token`. Dále se pak jedná o funkci pro přeložení hostname na IP adresu a potažmo o funkcionality posílání paketů na server a čtení odpovědi (viz [2]).

Poslední přejatou funkcí, která je z přejatých funkcí také nejdůležitější a nejméně změněná, je funkce na konverzi hesla do jeho base64 podoby. Tato funkce je umístěna v hlavičkovém souboru `base64.h` a zdrojovém souboru `base64.cpp` (viz [3]).

A co se odlišností týče, tak se jedná o vlastní chybové hlášky, které se v některých případech mírně odlišují od těch referenčních.

# Literatura

- [1] <https://stackoverflow.com/users/179910/jerry-coffin>: Read whole ASCII file into C++ std::string. [Stack Overflow], 8.4.2010, <https://stackoverflow.com/questions/2602013/read-whole-ascii-file-into-c-stdstring> [Cit. 27. 10. 2021]. URL: <https://stackoverflow.com/a/2602258>
- [2] Kerrisk, M.; Drepper, U.: getaddrinfo(3) - Linux manual page. [Online], 2008, [Cit. 27. 10. 2021]. URL: <https://man7.org/linux/man-pages/man3/getaddrinfo.3.html>
- [3] Lihocký, M.: Base64 decode snippet in C++. [Online], 18.12.2012, <https://stackoverflow.com/questions/180947/base64-decode-snippet-in-c> [Cit. 27. 10. 2021]. URL: <https://stackoverflow.com/a/13935718>
- [4] Sundland, M.: Creating a Wireshark dissector in Lua - part 1 (the basics). [Online], 2017, [Cit. 27. 10. 2021]. URL: <https://mika-s.github.io/wireshark/lua/dissector/2017/11/04/creating-a-wireshark-dissector-in-lua-1.html>