ISA Projekt

Reverse-engineering neznámého protokolu (Ing. Koutenský)

Manuál

Ondřej Sloup (xsloup02)

Obsah

O projektu

Cílem tohoto projektu bylo vytvořit disektor pro Wireshark v jazyku Lua a implementovat kompatibilního klienta pro připojení na poskytnutý server. Úkol projektu byl shrnut do následujících bodů:

1. Zachytit komunikaci mezi poskytnutým serverem a klientem
2. Implementovat vlastní Wireshark disektor pro daný protokol
3. Implementovat kompatibilního klienta pro daný protokol

Postup řešení

Prvním krokem v mém řešení bylo zprovoznit virtuální stroj na kterém je funkční implementace klienta i serveru a zjistit, jak vypadají posílané packety. Zjistil jsem, že se jedná o výměnu informací na TCP spojení podobné emailu, kde klient se registruje, přihlašuje a posílá zprávy jiným uživatelům.

Použil jsem program Wireshark pro sledování a zachycení komunikace. Na server jsem skrz klienta poslal všechny dostupné příkazy a celou komunikaci jsem si uložil jako „`.pcap`“ soubor. Na této zachycené komunikaci jsem se rozhodl vytvořit disektor, který mě více seznámil s klientem, který budu implementovat v následujícím kroku.

Wireshark disektor

Volba prostředí

Rozhodl jsem se psát disektor v jazyku Lua, kvůli rozsáhlým příkladům dostupných na oficiálních stránkách Wiresharku a zmíněným odkazům přímo v zadání – (1), (2), (3). Ačkoliv jsem neměl zkušenosti, jak s psaním disektorů, ani v psaním kódu v jazyku Lua, myslím, že jsem udělal správně rozhodnutí, jelikož zpracování této části projektu proběhlo poměrně hladce.

Použité zdroje

Před začátkem psaní kódu jsem si přečetl všechny dostupné příklady z oficiální dokumentace Wiresharku (1), (2), (3) a našel jsem YouTube video, které popisuje podobný příklad disektoru, který jsem implementoval (YouTube video – (4) a jeho zdrojový kód – (5)). Díky těmto zdrojům jsem byl schopný pochopit, jak script vytvořit a jaké jsou správné návrhové vzory.

Popis disektoru

Skript je rozdělen na 3 funkce – `parseMessage(), LaunchISAMaildissector() a Main().`

`Main()` je hlavní funkce, které je spuštěna na začátku scriptu a jejím jediným úkolem je spustit funkci `LaunchISAMaildissector()`.

Funkce `LaunchISAMaildissector()` spouští celý proces disektoru a definuje základní části pro rozpoznání protokolu. Nejprve definuje známé hodnoty jako jméno protokolu, proměnné označují úspěch a neúspěch odpovědi od serveru a možné výstupy do stromu Wiresharku. Dále je spuštěna vnitřní funkce `isamailproto.dissector()`, která již analyzuje zaslaná data.
Následuje strom podmínek, které rozpoznají, zda je správa od klienta či od serveru a zda je úspěšná. Jednotlivé zprávy jsou poté zařazeny a zpracovány tak, aby byly čitelné pro program Wireshark, a i pro jeho uživatele. Pro rozpoznání jednotlivých příkazů jsem použil pokročilé "Lua patterns", které mi usnadnili mapování zaslaných dat a rozpoznávání typů zprávy.

Jelikož se jedná o TCP protokol musel jsem implementovat znovu sestavení protokolu, pokud data přesáhnou délku 4096 bajtů. Pokud toto skript zjistí zavolá vnitřní funkci „`DESEGMENT_ONE_MORE_SEGMENT`", která se postará o to, aby skript dostal všechny data. Dále jsem implementoval zkontrolování délky packetu (7 bajtů), což odpovídá nejméně možné velikosti dat v packetu – prázdný list – „`(ok ())`". Při neúspěchu se packet nerozpoznává a je nezměněn skriptem.

Celý Wireshark disektor je namapován na stabilní port „32323", který je nastaven jako výchozí port na klientu. Rozhodl jsem se nerozpoznávat protokol heuristicky podle dat z důvodu možné kolize s ostatními, podobně vypadajícími protokoly, kde bych nemohl zaručit správnost. Uživatel toto nastavení bude muset mít na paměti.

Vše je popsáno a okomentováno ve zdrojovém kódu.

Vytváření klienta

Volba prostředí

Rozhodl jsem se plně použít veškerých výhod jazyka C++ a vypracovat celý projekt v něm. Chtěl jsem tento projekt takto zpracovat hlavně z toho důvodu, že to byl můj první projekt v jazyku C++ a díky disektoru, který jsem poměrně rychle implementoval jsem měl čas si promyslet celou strukturu a návrh klienta. Server pro klienta jsem si dokázal zprovoznit na svém počítači ve WSL2, což značně zjednodušilo debugování aplikace a její vypracování.

Použité knihovny

Knihovny, které jsem použil jsou ze standartní knihovny. Jmenovitě jsou to tyto:

- `sys/socket.h` – Knihovna pro vytvoření soketu a definici všech struktur, které jsou potřebné pro navázaní spojení se serverem
- `unistd.h` – Knihovna pro deklaraci struktur a typů, které usnadní práci s připojením
- `arpa/inet.h` – Knihovna pro deklaraci portů a IP adresy
- `netdb.h` – Knihovna pro deklaraci funkcí a typů potřebné k navázaní kontaktu
- `getopt.h` – Knihovna pro zpracování argumentů dle Unixu
- `regex` – Knihovna pro rozpoznání textových řetězců regulárním výrazem. Je použit pouze pro verifikaci packetů
- `iostream` – Knihovna pro standartní vstup a výstup
- `map` – Knihovna pro slovníky
- `vector` – Knihovna pro „pokročilá pole”
- `stdio.h`, `stdlib.h` – Knihovny pro makra a pro typy proměnných
- `string.h` – Knihovna pro definici textových řetězců
- `bitset` – Knihovna pro operace s bity používaná pro base64 konverzi
- `fstream` – Knihovna pro práci se soubory používaná pro vytváření souborů s tokenem

Použité zdroje

Při navrhování klienta jsem vycházel ze základní implementace TCP klienta, kterou jsem našel na internetu (6) a dále jsem ji přizpůsoboval svým potřebám. Velkou pomoc jsem našel na stránce (7), když jsem řešil rozpoznávání domén a IPv6. Řešil jsem problém s funkcemi `gethostbyname()`, která nepodporuje IPv6, `gethostbyname2()`, která podporuje IPv6, ale je složité ji implementovat a `getaddrinfo()`, kterou používám v projektu. (viz Zjišťování domén a IPv6)

Popis programu

Aplikace klient navazuje TCP spojení na server, aby na něm mohl vykonat předdefinované příkazy. Uživatel může specifikovat adresu i port, ale aplikace má i své výchozí nastavení (adresa "localhost" a port "32323"). Dále uživatel specifikuje jeden ze 6 příkazů, který provede danou akci:

1. `register` – registruje uživatele za pomocí specifikovaného jména a hesla. Heslo je převedeno do base64 podoby před odesláním na server
2. `login` – Přihlašuje uživatele za pomoci jeho jména a hesla, které je stejně jako v případě registrace převedeno do base64. Vytváří login-token.
3. `list` – Nemá žádné další parametry. Vypisuje všechny zprávy, které dostal přihlášený uživatel
4. `fetch` – Umožňuje zobrazit uživateli zprávu pomocí její ID. Uživatel toto ID musí zadat. ID odpovídá pořadí zpráv v listu. Je dostupný pouze přihlášenému uživateli.
5. `send` – Uživatel musí zadat komu je zpráva poslána, s jakým předmětem a samotný text zprávy. Je dostupný pouze přihlášenému uživateli a pouze přihlášený uživatel si může zprávu přečíst.
6. `logout` – Maže login-token a tím odhlašuje uživatele z klienta.

Popis vypracování

Aplikace je rozdělená na několik souborů, kde každý zastává určitou část.

- main.cpp – Vstupní část programu, která spouští jednotlivé části programu
- client.cpp/hpp – Zpracovává vstupní parametry a vytváří soket, který je použit připojení na server
- handler.cpp/hpp – Na základě předaných vstupních parametrů vytvoří třídu, která ze zpracovaných příkazů vytvoří textové řetězce, který je odeslán na server. Poté zpravuje příchozí zprávu ze serveru.
- requestmsg.cpp/hpp – Soubor obsahující jednotlivé třídy, které zpracovávají zprávy ze serveru a vytváří odesílané zprávy
- errorcodes.hpp – Seznam chybových kódů

Třídy

V projektu jsem vytvořil dvě základní třídy:

- RequestHandler – Třída, která rozpoznává, která třída se má vytvořit podle zadaného parametru
- RequestMsg – Abstraktní třída, která má pod sebou dalších šest tříd odpovídající každému příkazu, který uživatel může zadat. Jednotlivé třídy mají stejné abstraktní funkce, které přímo zpracují zprávu, která se odešle na server.

Funkce

main.cpp

- `int main()` – vstupní funkce projektu, která spouští všechny funkce tohoto programu

client.cpp/hpp

- `printHelpMessage()` – Funkce obsahující textový řetězec, který je vypsán, pokud uživatel se snaží vypsat pomocnou zprávu
- `parseArgs()` – Funkce, která se stará o pasování argumentů pomocí getopt a ověřuje jejich platnost
- `createSocketAndConnect()` – Funkce, která vytváří soket na kterém bude vytvořeno spojení. Zde jsem použil tento zdroj (7).

handler.cpp/hpp

Tento soubor obsahuje pouze zmíněnou třídu RequestHandler()

- `buildClientString()` – Funkce pro vytvoření textového řetězce, který bude zaslán serveru. Funkce nejdříve zjistí podle definované mapy, pod kterou třídu příkaz spadá
- `exchangeData()` – Funkce, která se pomocí vytvořeného soketu připojuje na server. Dokáže rozpoznat doménová jména a umí IPv4 i IPv6.

requestmsg.cpp/hpp

- třída `RequestMsg` – abstraktní třída, která zpracovává obsah příchozích a odchozích zpráv. Obsahuje definici funkcí, které pomáhají zpracovat zprávy serveru. Dále obsahuje informace o počtu argumentů a název příkazů
    - `buildString()` – abstraktní funkce, které je děděná všemi příkazovými třídami. Vytváří textový řetězce a zajišťuje veškeré funkce, které jsou potřebné pro jeho vytvoření.
    - `handleOutput()` – abstraktní funkce, které je děděná všemi příkazovými třídami. Zpracuje textový řetězec, který je přijat od serveru a zavolá všechny potřebné funkce.
    - `getError()` – Vypíše specifickou chybovou hlášku, pokud je příkaz zadán se špatnými požadavky
    - `getNumArg()` – Navrátí počet argumentů potřebných pro zavolání příkazu
    - `toBase64()` – Konvertuje heslo do base64 podoby
    - `getToken()` – Navrátí token z uloženého souboru, který je potřeba pro interakci se serverem při přihlášeném uživateli
    - `createToken()` – Vytvoří token a uloží ho do souboru pro pozdější použití
    - `removeToken()` – Smaže soubor při odhlášení uživatele
    - `resultParse()` – Zjistí, zda data v packetu jsou ve správném formátu a jestli je možné je dále zpracovávat
    - `printResult()` – Vypíše na standartní výstup z vytvořený textový řetězec s hlavičkou “ERROR” nebo “SUCCESS”.
    - `escapeChars()` – Zajistí, aby znaky „\n”, „\t”, „\”, „“ byly escapované při posílání na server
    - `unescapeChars()` – Zajistí, aby znaky „\n”, „\t”, „\”, „“ byly převedeny při vypisování na standartní výstup
    - `isNumber()` – Zjistí, jestli daný textový řetězec je číslo
    - `splitByRegex()` – Funkce, jejíž podobnou verzi používám i v disektoru, zpracovává příchozí zprávy podle vytvořeného konečného automatu, který zjišťuje podle uvozovek, o jakou část zprávy se jedná
    - třídy `Register`, `Login`, `List`, `Fetch`, `Send`, `Logout` – obsahují abstraktní funkce `buildString()`, `handleOutput()` a `getError()`, které jsou upraveny specificky pro jejich potřeby

Problémy s implementací

Zjišťování domén a IPv6

Při vypracovávání této části projektu jsem se snažil navrhnout klienta, tak aby podporoval vše jako referenční klient, a to IPv6 podporu a podporu doménových jmen. Na fóru jsem narazil na funkci `gethostbyname()` z knihovny `netdb.h`, kterou jsem poměrně rychle a jednoduše naimplementoval. V průběhu řešení jsem ale zjistil, že funkce nemá podporu pro IPv6, a proto jsem začal hledat další ekvivalenty a jako jeden z možných se naskytla funkce `gethostbyname2(),` která sice tuto podporu má, ale bylo složité ji naimplementovat tak aby jednoduše dosáhla mého žádaného výsledku. Při řešení problémů s touto funkcí jsem objevil funkci `getaddrinfo()` (7). Tato funkce podporovala vše, co jsem chtěl naimplementovat.

Zpracovávání zpráv od serveru

Zpracování packetů od serveru bylo nejsložitější, a i nejvíc časově náročné na celém projektu. Pokusil jsem se o 3 různé způsoby, jak navrhnout zpracování dat, tak, abych neudělal nikde chybu. Mým prvním nápadem bylo rozdělovat jednotlivé argumenty zprávy pomocí textového řetězce „“ “” – mezera ohraničená uvozovkami. Bohužel toto řešení nebylo správné, jelikož zde nastalo mnoho chyb, pokud uživatel chtěl zapsat jako poslední dva znaky právě uvozovku a mezeru.

Proto jsem se rozhodl zpracovat regulární výraz který by mi pomohl s tímto problémem a který fungoval skvěle, než jsem se pustil do rozsáhlého testování, kde jsem zjistil že kompilátor gcc má v sobě chybu při zpracovávání regulárních řetězců nad textových řetězcem větším než 27KiB a dochází k „stack overflow“ což je fatální chyba. (8) (9) (10)

Mým posledním a funkčním řešením bylo vytvořit funkci, která prochází řetězcem a kontroluje, zda se text nachází v uvozovkách či ne a přeskakuje znaky které jsou serverem (nebo klientem) escapovány.

Navazování spojení při zjištění chyby

Při zkoumání referenčního klienta jsem zjistil, že klient navazuje spojení i přestože nedochází ke komunikaci z důvodu chyby. Toto jsem se rozhodl neimplementovat abych nezatěžoval server.

Base64 konverze

Při vypracovávání klienta jsem si všiml že všechna hesla, se kterými se uživatel přihlašuje na server jsou konvertována do base64 pro základní bezpečnost. Tuto funkci jsem také implementoval s pomocí webové stránky (11), odkud jsem pochopil její základní princip a přepsal ho do jazyka C++.

Návratové kódy

Soubor errorcodes.hpp obsahuje veškeré návratové kódy včetně vysvětlení jejich významů.

Testování

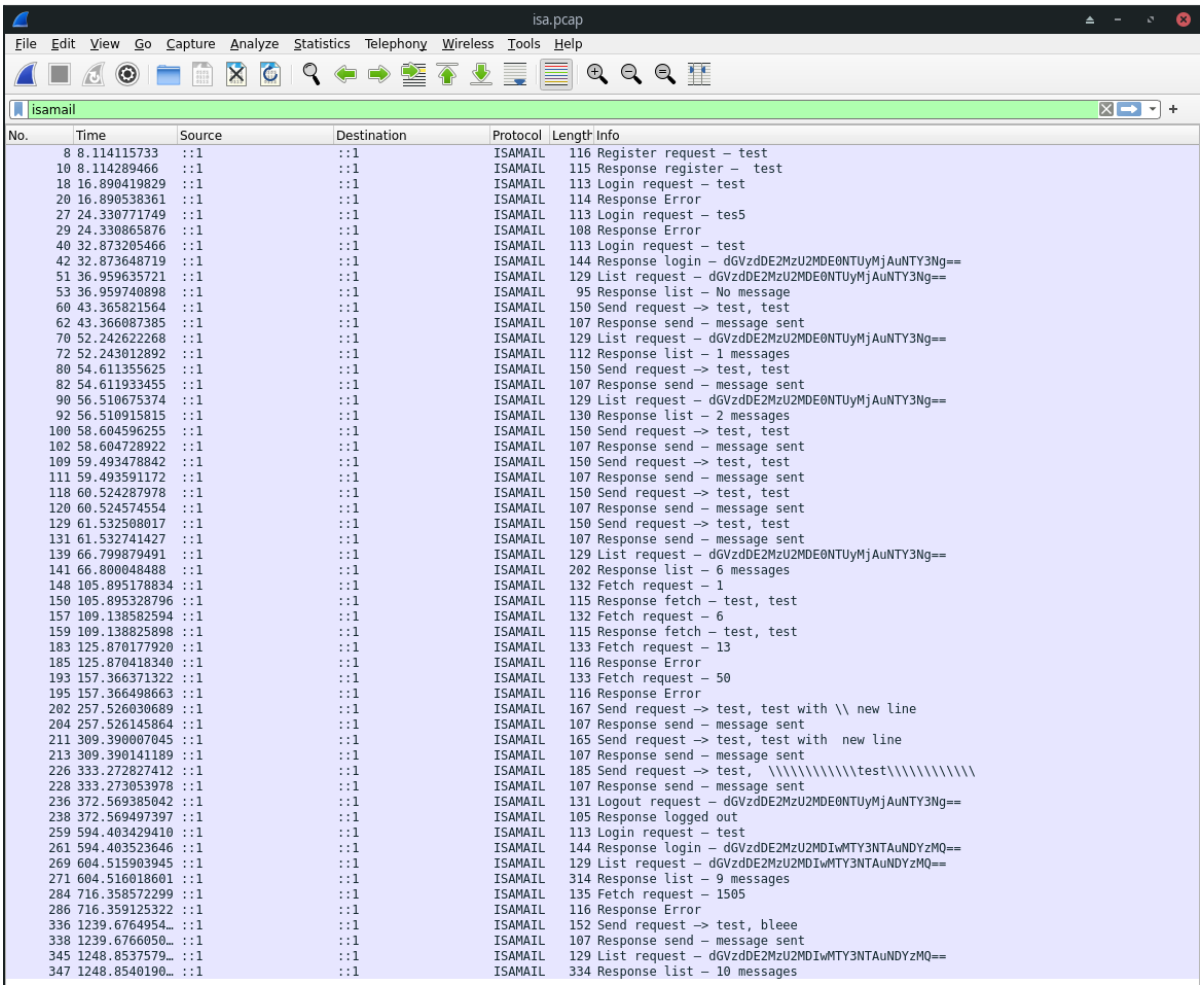
Aplikace jsem otestoval na těchto systémech:

1. Ubuntu 20.04.3 LTS on Windows 10 x86_64 – (5.4.72-microsoft-standard-WSL2)
2. Manjaro Linux x86_64, 5.13.19-2-MANJARO – referenční počítač

Bylo testováno samostatné sestavení projektu, a i jeho spuštění

Wireshark disektor

Wireshark disektor jsem byl schopen otestovat i na systému Windows 10 Pro 21H1 – 19043.1288. Skript je nutné přidat do konfigurační Wireshark složky. Na Linuxu do „`/root/.config/wireshark/`" (na referenčním virtuálním stroji je nutné spustit wireshark jako root) a na Windows do „`%APPDATA%\Roaming\Wireshark`". Je testován s referenčním clientem.



Příklady spuštění

Kompletní Wireshark příklad spuštění na referenčním počítači na výchozím portu a adrese.

Login

Request a response:

```
Frame 40: 113 bytes on wire (904 bits), 113 bytes captured (904 bits)
Linux cooked capture v1
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 57614, Dst Port: 32323, Seq: 1, Ack: 1, Len: 25
ISAMAIL Protocol Data
    Request: login
    Login: test
    Password: dGVzdA==
    Length: 25
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ·· ····· ·· ·····
0010  60 05 1d a9 00 39 06 40  00 00 00 00 00 00 00 00   `····9·@ ·· ·····
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ·· ····· ·· ·····
0030  00 00 00 00 00 00 00 01  e1 0e 7e 43 9e 6a 97 98   ·· ····· ··~C·j··
0040  86 10 0a f0 80 18 02 00  00 41 00 00 01 01 08 0a   ·· ····· ·A ·····
0050  15 4c 8e f3 15 4c 8e f2  28 6c 6f 67 69 6e 20 22   ·L···L·· (login "
0060  74 65 73 74 22 20 22 64  47 56 7a 64 41 3d 3d 22   test" "d GVzdA=="
0070  29                                                )
```

```
Frame 42: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits)
Linux cooked capture v1
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 32323, Dst Port: 57614, Seq: 1, Ack: 26, Len: 56
ISAMAIL Protocol Data
    Token: dGVzdDE2MzU2MDE0NTUyMjAuNTY3Ng==
    Status: ok
    Message: "user logged in" "dGVzdDE2MzU2MDE0NTUyMjAuNTY3Ng=="
    Length: 56
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ·· ····· ·· ·····
0010  60 04 81 44 00 58 06 40  00 00 00 00 00 00 00 00   `··D·X·@ ·· ·····
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ·· ····· ·· ·····
0030  00 00 00 00 00 00 00 01  7e 43 e1 0e 86 10 0a f0   ·· ····· ·· ~C····
0040  9e 6a 97 b1 80 18 02 00  00 60 00 00 01 01 08 0a   ·j····· ·`·· ····
0050  15 4c 8e f3 15 4c 8e f3  28 6f 6b 20 22 75 73 65   ·L···L·· (ok "use
0060  72 20 6c 6f 67 67 65 64  20 69 6e 22 20 22 64 47   r logged  in" "dG
0070  56 7a 64 44 45 32 4d 7a  55 32 4d 44 45 30 4e 54   VzdDE2Mz U2MDE0NT
0080  55 79 4d 6a 41 75 4e 54  59 33 4e 67 3d 3d 22 29   UyMjAuNT Y3Ng==")
```

Chybové stavy:

```
Frame 20: 114 bytes on wire (912 bits), 114 bytes captured (912 bits)
Linux cooked capture v1
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 32323, Dst Port: 57610, Seq: 1, Ack: 26, Len: 26
ISAMAIL Protocol Data
    Status: err
    Message: "incorrect password"
    Length: 26
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ·· ····· ·· ·····
0010  60 08 c6 41 00 3a 06 40  00 00 00 00 00 00 00 00   `··A·:·@ ·· ·····
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ·· ····· ·· ·····
0030  00 00 00 00 00 00 00 01  7e 43 e1 0a f5 1c 18 3c   ·· ····· ·· ~C····<
0040  1c 74 cb 08 80 18 02 00  00 42 00 00 01 01 08 0a   ·t····· ·B ·····
0050  15 4c 50 84 15 4c 50 84  28 65 72 72 20 22 69 6e   ·LP··LP· (err "in
0060  63 6f 72 72 65 63 74 20  70 61 73 73 77 6f 72 64   correct  password
0070  22 29                                             ")
```

```
▸ Frame 29: 108 bytes on wire (864 bits), 108 bytes captured (864 bits)
▸ Linux cooked capture v1
▸ Internet Protocol Version 6, Src: ::1, Dst: ::1
▸ Transmission Control Protocol, Src Port: 32323, Dst Port: 57612, Seq: 1, Ack: 26, Len: 20
▾ ISAMAIL Protocol Data
    Status: err
    Message: "unknown user"
    Length: 20
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 06 d8 ea 00 34 06 40  00 00 00 00 00 00 00 00   `....4.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  7e 43 e1 0c a7 83 6b 51   ........ ~C....kQ
0040  dd d7 c0 81 80 18 02 00  00 3c 00 00 01 01 08 0a   ........ .<......
0050  15 4c 6d 94 15 4c 6d 94  28 65 72 72 20 22 75 6e   .Lm..Lm. (err "un
0060  6b 6e 6f 77 6e 20 75 73  65 72 22 29               known us er")
```

Register

Request a response:

```
▸ Frame 8: 116 bytes on wire (928 bits), 116 bytes captured (928 bits)
▸ Linux cooked capture v1
▸ Internet Protocol Version 6, Src: ::1, Dst: ::1
▸ Transmission Control Protocol, Src Port: 57608, Dst Port: 32323, Seq: 1, Ack: 1, Len: 28
▾ ISAMAIL Protocol Data
    Request: register
    Login: test
    Password: dGVzdA==
    Length: 28
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 0c f7 1d 00 3c 06 40  00 00 00 00 00 00 00 00   `....<.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  e1 08 7e 43 b1 53 a8 db   ........ ..~C.S..
0040  59 f4 13 f4 80 18 02 00  00 44 00 00 01 01 08 0a   Y....... .D......
0050  15 4c 2e 3b 15 4c 2e 37  28 72 65 67 69 73 74 65   .L.;.L.7 (registe
0060  72 20 22 74 65 73 74 22  20 22 64 47 56 7a 64 41   r "test"  "dGVzdA
0070  3d 3d 22 29                                       ==")
```

```
▸ Frame 10: 115 bytes on wire (920 bits), 115 bytes captured (920 bits)
▸ Linux cooked capture v1
▸ Internet Protocol Version 6, Src: ::1, Dst: ::1
▸ Transmission Control Protocol, Src Port: 32323, Dst Port: 57608, Seq: 1, Ack: 29, Len: 27
▾ ISAMAIL Protocol Data
    Login:  test
    Status: ok
    Message: "registered user test"
    Length: 27
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 0c db 91 00 3b 06 40  00 00 00 00 00 00 00 00   `....;.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  7e 43 e1 08 59 f4 13 f4   ........ ~C..Y...
0040  b1 53 a8 f7 80 18 02 00  00 43 00 00 01 01 08 0a   .S...... .C......
0050  15 4c 2e 3c 15 4c 2e 3b  28 6f 6b 20 22 72 65 67   .L.<.L.; (ok "reg
0060  69 73 74 65 72 65 64 20  75 73 65 72 20 74 65 73   istered  user tes
0070  74 22 29                                          t")
```

List

Request a response:

```
▸ Frame 51: 129 bytes on wire (1032 bits), 129 bytes captured (1032 bits)
▸ Linux cooked capture v1
▸ Internet Protocol Version 6, Src: ::1, Dst: ::1
▸ Transmission Control Protocol, Src Port: 57616, Dst Port: 32323, Seq: 1, Ack: 1, Len: 41
▾ ISAMAIL Protocol Data
    Request: list
    Token: dGVzdDE2MzU2MDE0NTUyMjAuNTY3Ng==
    Length: 41
```

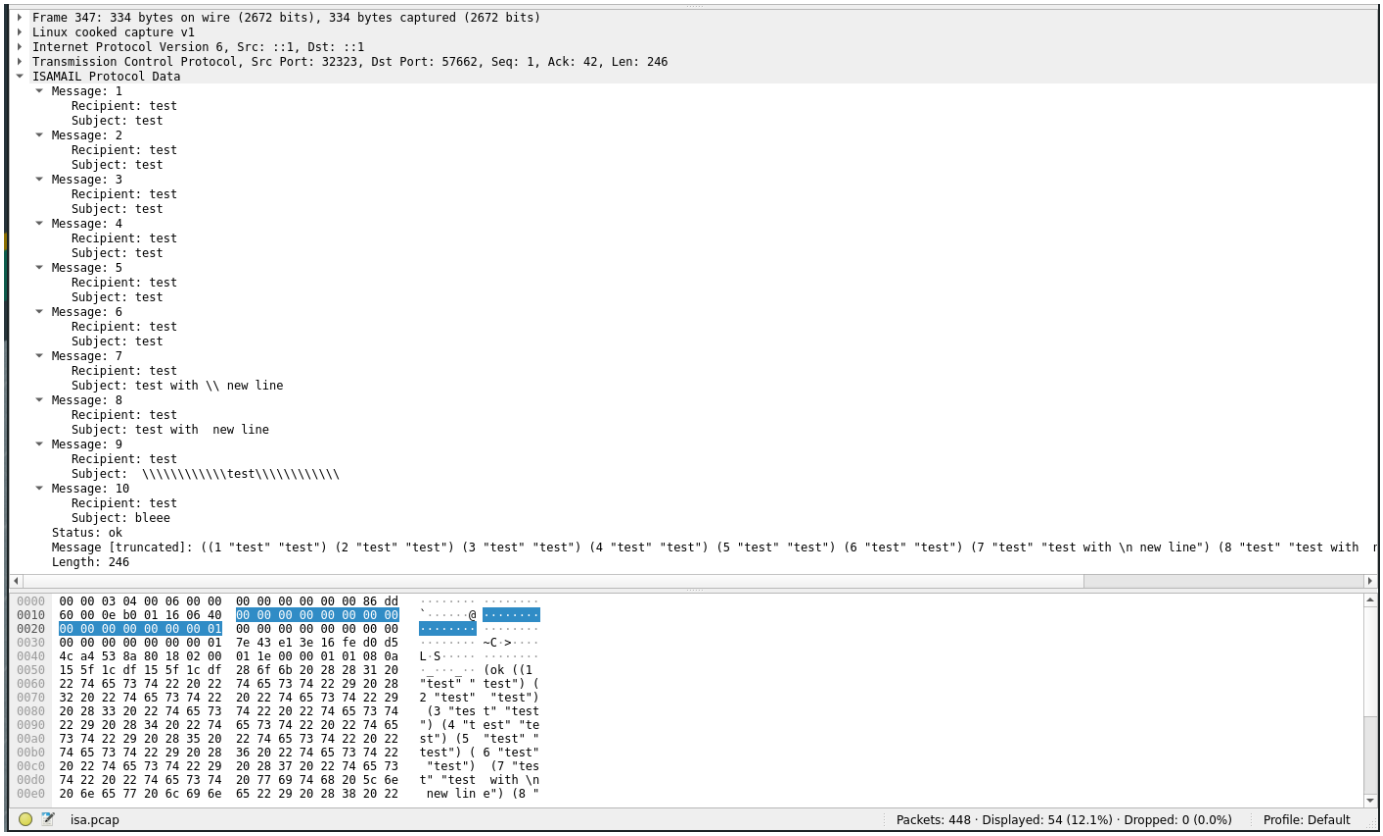
```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 04 0d 0f 00 49 06 40  00 00 00 00 00 00 00 00   `....I.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  e1 10 7e 43 e8 a8 c3 9a   ........ ..~C....
0040  ae 4b 83 2c 80 18 02 00  00 51 00 00 01 01 08 0a   .K.,.... .Q......
0050  15 4c 9e e9 15 4c 9e e9  28 6c 69 73 74 20 22 64   .L...L.. (list "d
0060  47 56 7a 64 44 45 32 4d  7a 55 32 4d 44 45 30 4e   GVzdDE2M zU2MDE0N
0070  54 55 79 4d 6a 41 75 4e  54 59 33 4e 67 3d 3d 22   TUyMjAuN TY3Ng=="
0080  29                                                )
```

```
▸ Frame 72: 112 bytes on wire (896 bits), 112 bytes captured (896 bits)
▸ Linux cooked capture v1
▸ Internet Protocol Version 6, Src: ::1, Dst: ::1
▸ Transmission Control Protocol, Src Port: 32323, Dst Port: 57620, Seq: 1, Ack: 42, Len: 24
▾ ISAMAIL Protocol Data
  ▾ Message: 1
      Recipient: test
      Subject: test
    Status: ok
    Message: ((1 "test" "test"))
    Length: 24
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 86 dd 86 dd   ........ ........
0010  60 07 ac e8 00 38 06 40  00 00 00 00 00 00 00 00   `....8.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  7e 43 e1 14 63 18 99 30   ........ ~C..c..0
0040  63 e8 e9 a4 80 18 02 00  00 40 00 00 01 01 08 0a   c....... .@......
0050  15 4c da 9c 15 4c da 9c  28 6f 6b 20 28 28 31 20   .L...L.. (ok ((1
0060  22 74 65 73 74 22 20 22  74 65 73 74 22 29 29 29   "test" " test")))
```

```
▸ Frame 53: 95 bytes on wire (760 bits), 95 bytes captured (760 bits)
▸ Linux cooked capture v1
▸ Internet Protocol Version 6, Src: ::1, Dst: ::1
▸ Transmission Control Protocol, Src Port: 32323, Dst Port: 57616, Seq: 1, Ack: 42, Len: 7
▾ ISAMAIL Protocol Data
    Status: ok
    Message: ()
    Length: 7
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 0c b4 47 00 27 06 40  00 00 00 00 00 00 00 00   `..G.'.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  7e 43 e1 10 ae 4b 83 2c   ........ ~C...K.,
0040  e8 a8 c3 c3 80 18 02 00  00 2f 00 00 01 01 08 0a   ........ ./......
0050  15 4c 9e e9 15 4c 9e e9  28 6f 6b 20 28 29 29      .L...L.. (ok ())
```


Frame 347: 334 bytes on wire (2672 bits), 334 bytes captured (2672 bits)
Linux cooked capture v1
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 32323, Dst Port: 57662, Seq: 1, Ack: 42, Len: 246
ISAMAIL Protocol Data
Message: 1
Recipient: test
Subject: test
Message: 2
Recipient: test
Subject: test
Message: 3
Recipient: test
Subject: test
Message: 4
Recipient: test
Subject: test
Message: 5
Recipient: test
Subject: test
Message: 6
Recipient: test
Subject: test
Message: 7
Recipient: test
Subject: test with \\ new line
Message: 8
Recipient: test
Subject: test with new line
Message: 9
Recipient: test
Subject: \\\\\\\\\\\\test\\\\\\\\\\\\
Message: 10
Recipient: test
Subject: bleee
Status: ok
Message [truncated]: ((1 "test" "test") (2 "test" "test") (3 "test" "test") (4 "test" "test") (5 "test" "test") (6 "test" "test") (7 "test" "test with \n new line") (8 "test" "test with
Length: 246
0000 00 00 03 04 00 06 00 00 00 00 00 00 00 00 86 dd
0010 60 00 0e b0 01 16 06 40 00 00 00 00 00 00 00 00
0020 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 01 7e 43 e1 3e 16 fe d0 d5
0040 4c a4 53 8a 80 18 02 00 01 1e 00 00 01 01 08 0a
0050 15 5f 1c df 15 5f 1c df 28 6f 6b 20 28 28 31 20
0060 22 74 65 73 74 22 20 22 74 65 73 74 22 29 20 28
0070 32 20 22 74 65 73 74 22 20 22 74 65 73 74 22 29
0080 20 28 33 20 22 74 65 73 74 22 20 22 74 65 73 74
0090 22 29 20 28 34 20 22 74 65 73 74 22 20 22 74 65
00a0 73 74 22 29 20 28 35 20 22 74 65 73 74 22 20 22
00b0 74 65 73 74 22 29 20 28 36 20 22 74 65 73 74 22
00c0 20 22 74 65 73 74 22 29 20 28 37 20 22 74 65 73
00d0 74 22 20 22 74 65 73 74 20 77 69 74 68 20 5c 6e
00e0 20 6e 65 77 20 6c 69 6e 65 22 29 20 28 38 20 22
isa.pcap
Packets: 448 · Displayed: 54 (12.1%) · Dropped: 0 (0.0%)
Profile: Default

Send

Request a response:

```
▸ Frame 62: 107 bytes on wire (856 bits), 107 bytes captured (856 bits)
▸ Linux cooked capture v1
▸ Internet Protocol Version 6, Src: ::1, Dst: ::1
▸ Transmission Control Protocol, Src Port: 32323, Dst Port: 57618, Seq: 1, Ack: 63, Len: 19
▾ ISAMAIL Protocol Data
    Status: ok
    Message: "message sent"
    Length: 19
```

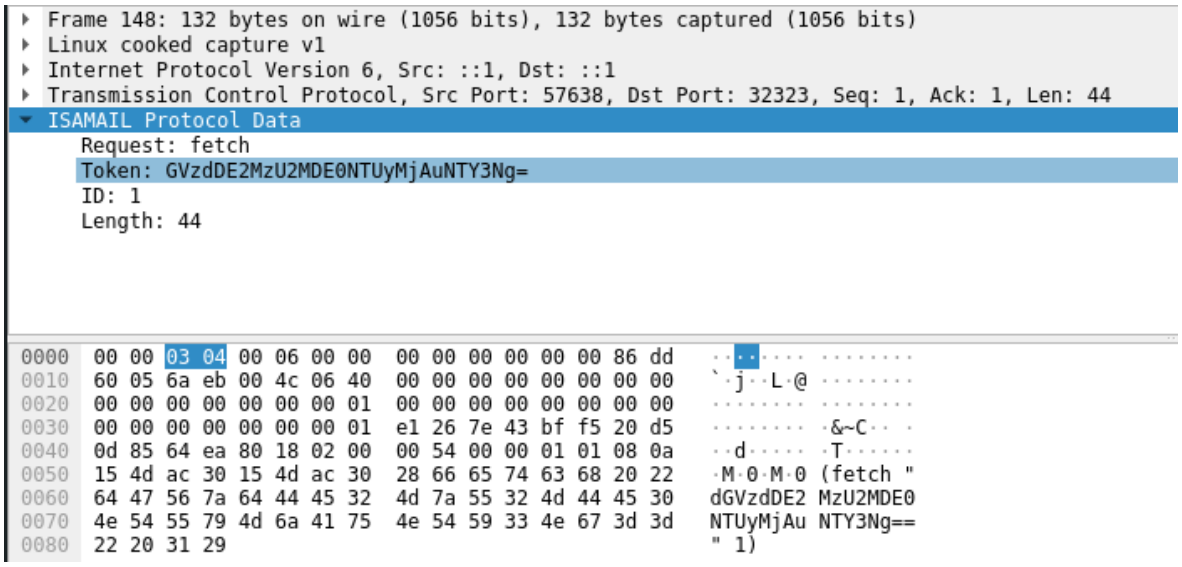
```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 0d a7 eb 00 33 06 40  00 00 00 00 00 00 00 00   `....3.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  7e 43 e1 12 a3 77 19 7c   ........ ~C...w.|
0040  57 65 1b 70 80 18 02 00  00 3b 00 00 01 01 08 0a   We.p.... .;......
0050  15 4c b7 ef 15 4c b7 ef  28 6f 6b 20 22 6d 65 73   .L...L.. (ok "mes
0060  73 61 67 65 20 73 65 6e  74 22 29                 sage sen t")
```

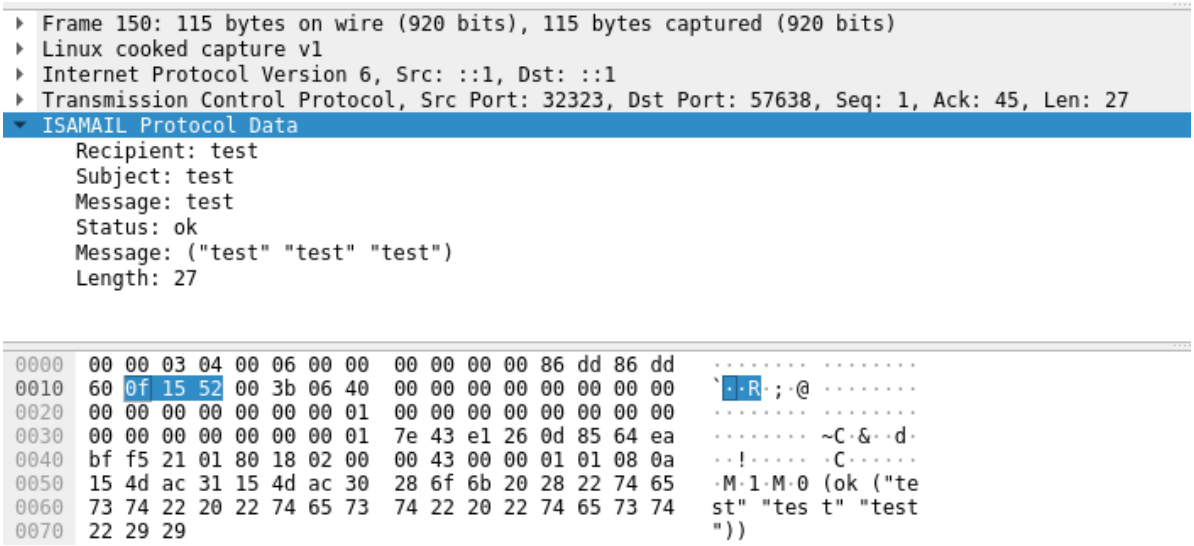
```
▸ Frame 60: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits)
▸ Linux cooked capture v1
▸ Internet Protocol Version 6, Src: ::1, Dst: ::1
▸ Transmission Control Protocol, Src Port: 57618, Dst Port: 32323, Seq: 1, Ack: 1, Len: 62
▾ ISAMAIL Protocol Data
    Request: send
    Token: dGVzdDE2MzU2MDE0NTUyMjAuNTY3Ng==
    Recipient: test
    Subject: test
    Message: test
    Length: 62
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 0c cc 9e 00 5e 06 40  00 00 00 00 00 00 00 00   `....^.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  e1 12 7e 43 57 65 1b 32   ........ ..~CWe.2
0040  a3 77 19 7c 80 18 02 00  00 66 00 00 01 01 08 0a   .w.|.... .f......
0050  15 4c b7 ef 15 4c b7 ef  28 73 65 6e 64 20 22 64   .L...L.. (send "d
0060  47 56 7a 64 44 45 32 4d  7a 55 32 4d 44 45 30 4e   GVzdDE2M zU2MDE0N
0070  54 55 79 4d 6a 41 75 4e  54 59 33 4e 67 3d 3d 22   TUyMjAuN TY3Ng=="
0080  20 22 74 65 73 74 22 20  22 74 65 73 74 22 20 22    "test"  "test" "
0090  74 65 73 74 22 29                                  test")
```

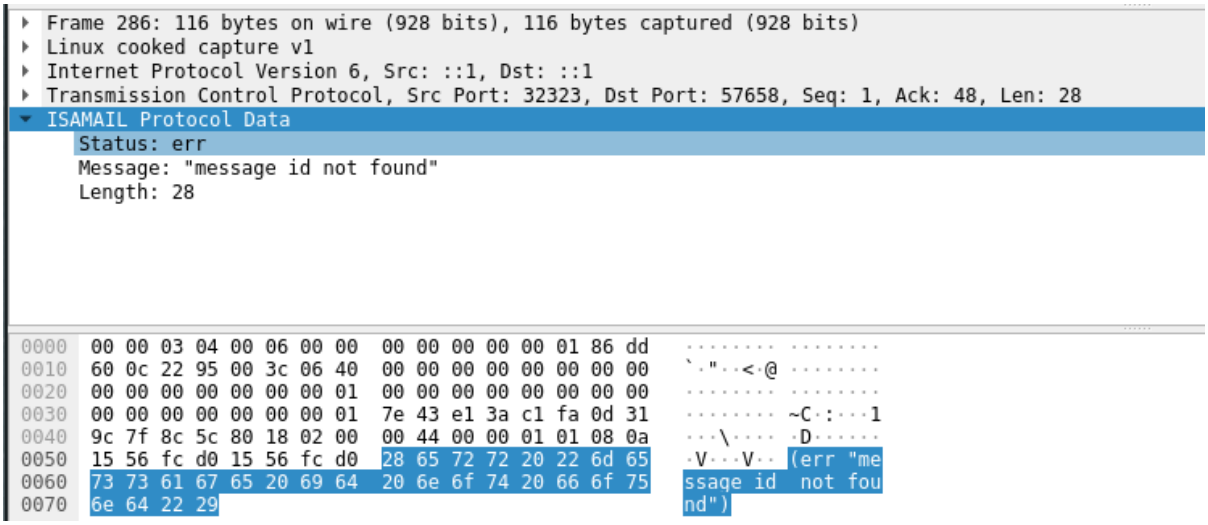
Fetch

Request a response:





Chybové stavy:



Logout

Request a response:

```
Frame 236: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits)
Linux cooked capture v1
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 57652, Dst Port: 32323, Seq: 1, Ack: 1, Len: 43
ISAMAIL Protocol Data
    Request: logout
    Token: dGVzdDE2MzU2MDE0NTUyMjAuNTY3Ng==
    Length: 43
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 00 91 65 00 4b 06 40  00 00 00 00 00 00 00 00   `..e.K.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  e1 34 7e 43 b4 a9 5d 13   ........ .4~C..].
0040  e8 af 6d 2b 80 18 02 00  00 53 00 00 01 01 08 0a   ..m+.... .S......
0050  15 51 bd e3 15 51 bd e2  28 6c 6f 67 6f 75 74 20   .Q...Q.. (logout
0060  22 64 47 56 7a 64 44 45  32 4d 7a 55 32 4d 44 45   "dGVzdDE 2MzU2MDE
0070  30 4e 54 55 79 4d 6a 41  75 4e 54 59 33 4e 67 3d   0NTUyMjA uNTY3Ng=
0080  3d 22 29                                          ="")
```

```
Frame 238: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)
Linux cooked capture v1
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 32323, Dst Port: 57652, Seq: 1, Ack: 44, Len: 17
ISAMAIL Protocol Data
    Status: ok
    Message: "logged out"
    Length: 17
```

```
0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 86 dd   ........ ........
0010  60 04 af 41 00 31 06 40  00 00 00 00 00 00 00 00   `..A.1.@ ........
0020  00 00 00 00 00 00 00 01  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 01  7e 43 e1 34 e8 af 6d 2b   ........ ~C.4..m+
0040  b4 a9 5d 3e 80 18 02 00  00 39 00 00 01 01 08 0a   ..]>.... .9......
0050  15 51 bd e3 15 51 bd e3  28 6f 6b 20 22 6c 6f 67   .Q...Q.. (ok "log
0060  67 65 64 20 6f 75 74 22  29                       ged out" )
```

Klient

Klient se kompiluje pomocí vytvořeného makefile. Jsou dostupné příkazy „`build`" a „`clean`", které vytvoří či smažou soubor „`./client`".

```
[isa@isa client]$ make
rm -f ./client
g++ -std=c++11  -o client main.cpp client.cpp handler.cpp requestmsg.cpp
[isa@isa client]$
```

Příklady spuštění

Moje aplikace je ve složce „client" a referenční ve složce „isa".

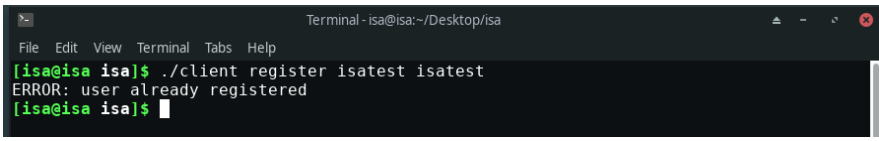
Login

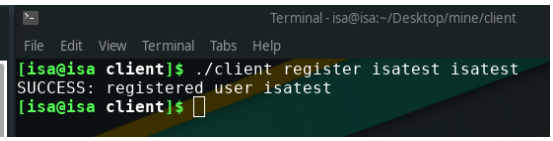
```
[isa@isa isa]$ ./client login test test
SUCCESS: user logged in
[isa@isa isa]$ ./client login test1 test
ERROR: unknown user
[isa@isa isa]$
```

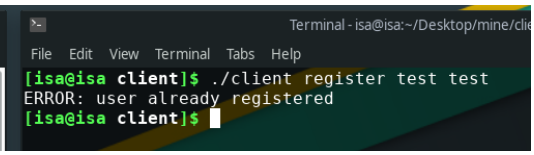
```
[isa@isa client]$ ./client login test test
SUCCESS: user logged in
[isa@isa client]$ ./client login test1 test
ERROR: unknown user
[isa@isa client]$
```

Register

Tento příklad nebylo možné porovnat jelikož 2 uživatelé nemůžou být na server registrovaní 2x.

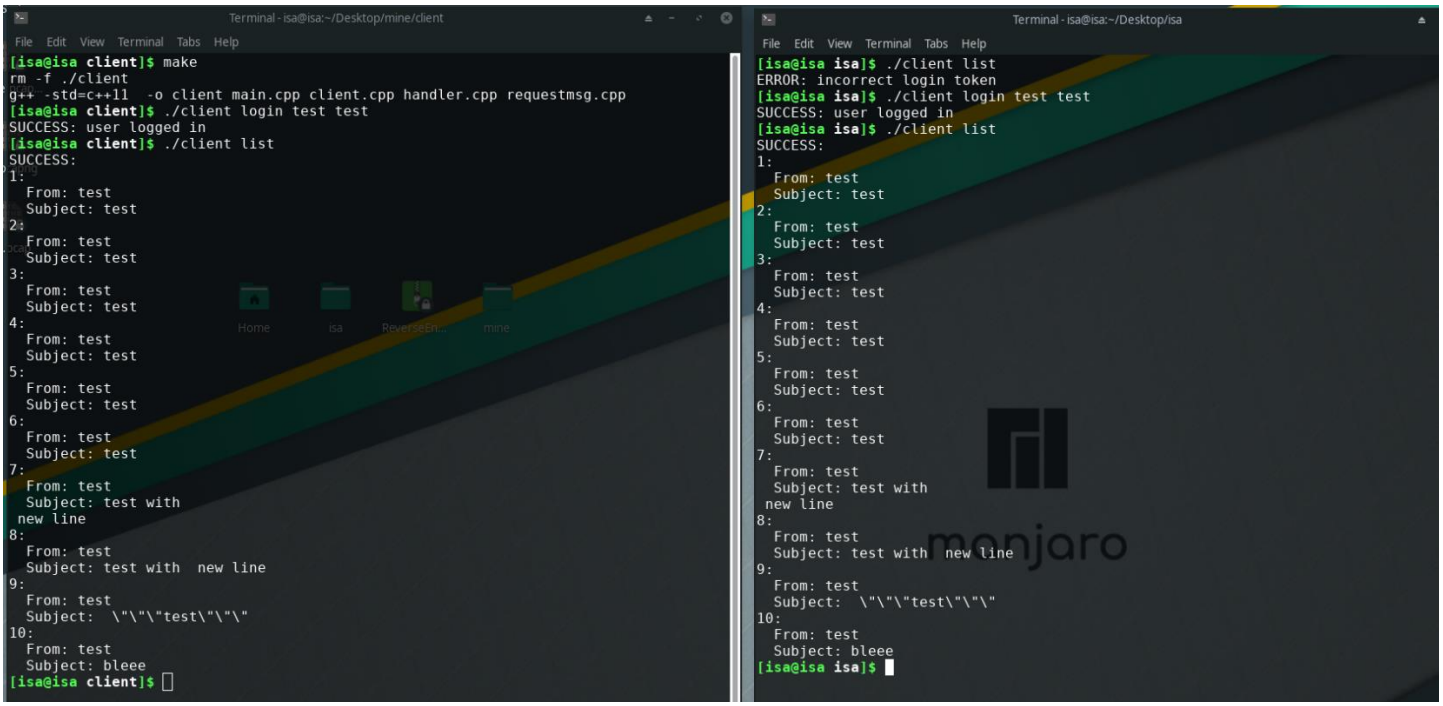




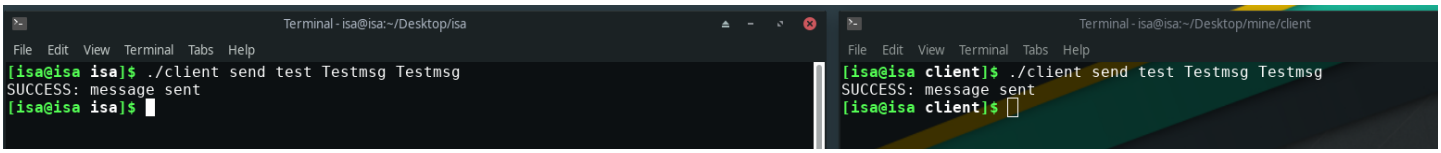


List

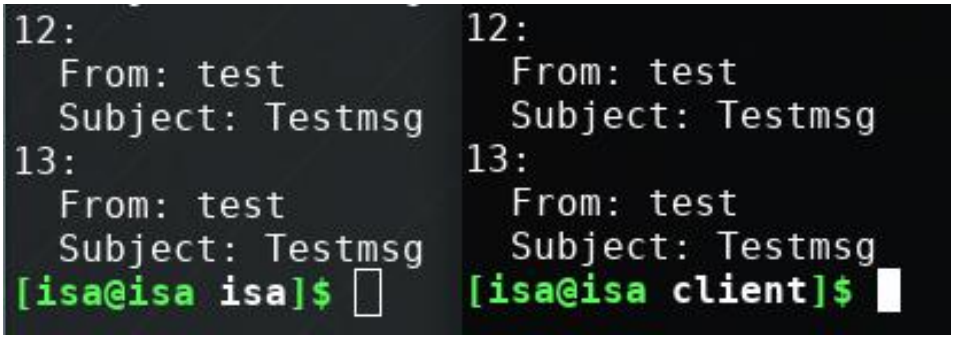
Tento fetch byl vytvořen send příkazy při testování disektoru.



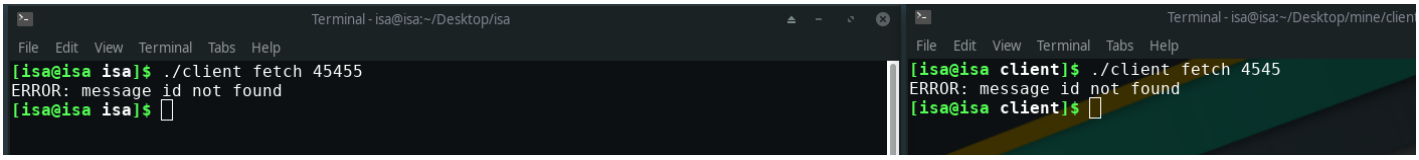
Send



Validace, že obě zprávy byly poslány.



Fetch



Logout

Reference

[1] 1. **Wireshark Foundation.** https://gitlab.com. *Wireshark dissector – Examples.* [Online] 12. August 2020. [Citace: 26. October 2021.] https://gitlab.com/wireshark/wireshark/-/wikis/Lua/Examples.

[2] 2. —. gitlab.com. *Wireshark wiki – Lua.* [Online] 8. October 2021. [Citace: 26. October 2021.] https://gitlab.com/wireshark/wireshark/-/wikis/Lua.

[3] 3. —. https://gitlab.com. *Wireshark wiki – Dissectors.* [Online] Wireshark Foundation, 1. September 2020. [Citace: 26. October 2021.] https://gitlab.com/wireshark/wireshark/-/wikis/Lua/Dissectors.

[4] 4. **Stevens (dist67), Didier.** https://www.youtube.com/. *Packet Class: Wireshark - Lua Protocol Dissectors.* [Online] 11. April 2014. [Citace: 26. October 2021.] https://www.youtube.com/watch?v=I4nf23HywmI.

[5] 5. **Didier (dist67), Stevens.** https://didierstevens.com. *Didier Stevens Blog.* [Online] 10. August 2014. [Citace: 26. October 2021.] https://didierstevens.com/files/workshops/botnet01-dissector.lua.

[6] 6. **Shukla 1, Yogesh.** https://www.geeksforgeeks.org. *GeeksForGeeks – TCP Server-Client implementation in C.* [Online] 23. September 2021. [Citace: 26. October 2021.] https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/.

[7] 7. **Roymunson.** https://stackoverflow.com/. *Client in C++, use gethostbyname or getaddrinfo – StackOverflow.* [Online] 13. October 2018. [Citace: 26. 10 2021.] https://stackoverflow.com/questions/52727565/client-in-c-use-gethostbyname-or-getaddrinfo.

[8] 8. **chaoskeeper.** https://gcc.gnu.org. *GCC Bugzilla – Bug 70459.* [Online] 30. March 2016. [Citace: 26. October 2021.] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=70459.

[9] 9. **Seelig, Holger.** https://gcc.gnu.org. *GCC Bugzilla – Bug 86164.* [Online] 15. June 2018. [Citace: 26. October 2021.] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=86164.

[10] 10. **Arciemowicz, Maksymilian.** https://gcc.gnu.org/. *GCC Bugzilla – Bug 61582.* [Online] 23. June 2014. [Citace: 26. October 2021.] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=61582.

[11] 11. **Victor.** https://base64.guru. *Base64 Encode Algorithm.* [Online] 2019. [Citace: 26. October 2021.] https://base64.guru/learn/base64-algorithm/encode.