Vysoké učení technické v Brně
Fakulta informačních technologií

Síťové aplikace a správa sítí

Reverse-engineering neznámeho protokolu

xnepra01
Kateřina Neprašová

13. listopadu 2021

Obsah

1 Zadání projektu

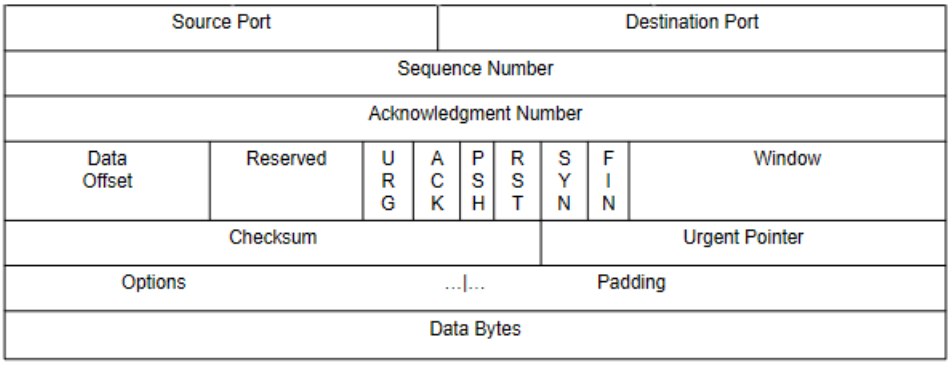
Cílem projektu je implementace komunikující aplikace pomocí síťové knihovny BSD sockets v jazyce `C/C++`. Celkově zadání sestává ze 3 úkolů:

1. Zachycení komunikace mezi poskytnutým serverem a klientem
2. Implementace vlastního Wireshark dissectoru pro daný protokol
3. Implementace kompatibilního klienta pro daný protokol

2 Zachycená komunikace mezi poskytnutým serverem a klientem

Klient a server komunikují pomocí TCP protokolu.

TCP (Transmission Control Protocol): *Hlavním účelem TCP je spolehlivé připojení, má uspořádanou povahu (pakety jsou zpracovávány v pevné posloupnosti). TCP se používá pro různé typy komunikace, jako je e-mail, přenos souborů a jakákoliv jiná operace, kde jsou bezchybná data důležitější než samotná rychlost přenosu.*[1]



Obrázek 1: Struktura TCP paketů

Paket: *Paket je malý segment větší zprávy odesílané přes počítačové sítě. Pakety se skládají ze záhlaví a užitečného obsahu. V záhlaví paketu nalezneme informace o obsahu, původu a cíli paketu.*[2]

Celková funkcionalita spočívá v registraci uživatele u klienta, jeho přihlášení, posílání zpráv, zobrazení seznamu poslaných zpráv a výpis zvolené zprávy. Ze zachycené komunikace můžeme tušit, že se jedná o nějaký *mailing* protokol - protokol pro korespondenci. Jednotlivé pakety obsahují informace podobné běžným TCP paketům.

Asi není velkým překvapením, že existuje jistá podobnost mezi touto zachycenou komunikací (a jejím principem) a běžnými protokoly, které se dnes používají pro korespondenci: Simple Mail Transfer Protocol (SMTP), Post Office Protocol (POP) a Internet Message Access Protocol (IMAP). Všechny tři též samozřejmě využívají TCP.[3]

```
No.  Time           Source      Destination  Protocol Length Info
   4 0.000714646    10.0.2.15   10.0.2.15    TCP      95     46240 → 32323 [PSH, ACK] Seq=1 Ack=1 Win=65536
                                                               Len=29 TSval=4036918474 TSecr=4036918473
   5 0.000739084    10.0.2.15   10.0.2.15    TCP      66     32323 → 46240 [ACK] Seq=1 Ack=30 Win=65536 Len=0
                                                               TSval=4036918474 TSecr=4036918474
   6 0.000924815    10.0.2.15   10.0.2.15    TCP      94     32323 → 46240 [PSH, ACK] Seq=1 Ack=30 Win=65536
                                                               Len=28 TSval=4036918474 TSecr=4036918474
   7 0.000930579    10.0.2.15   10.0.2.15    TCP      66     46240 → 32323 [ACK] Seq=30 Ack=29 Win=65536 Len=0
                                                               TSval=4036918474 TSecr=4036918474
   8 0.000965311    10.0.2.15   10.0.2.15    TCP      66     32323 → 46240 [FIN, ACK] Seq=29 Ack=30 Win=65536
                                                               Len=0 TSval=4036918474 TSecr=4036918474
   9 0.001135565    10.0.2.15   10.0.2.15    TCP      66     46240 → 32323 [FIN, ACK] Seq=30 Ack=30 Win=65536
                                                               Len=0 TSval=4036918474 TSecr=4036918474
  10 0.001146411    10.0.2.15   10.0.2.15    TCP      66     32323 → 46240 [ACK] Seq=30 Ack=31 Win=65536 Len=0
                                                               TSval=4036918474 TSecr=4036918474
  11 13.005006433   10.0.2.15   10.0.2.15    TCP      74     46242 → 32323 [SYN] Seq=0 Win=65495 Len=0
                                                               MSS=65495 SACK_PERM=1 TSval=4036931478 TSecr=0 WS=128
  12 13.005018281   10.0.2.15   10.0.2.15    TCP      74     32323 → 46242 [SYN, ACK] Seq=0 Ack=1 Win=65483
                                                               Len=0 MSS=65495 SACK_PERM=1 TSval=4036931478
                                                               TSecr=4036931478 WS=128
  13 13.005027482   10.0.2.15   10.0.2.15    TCP      66     46242 → 32323 [ACK] Seq=1 Ack=1 Win=65536 Len=0
                                                               TSval=4036931478 TSecr=4036931478
```

Obrázek 2: Zachycená komunikace

Na začátku komunikace probíhá standardní *Three-Way Handshake* = proces, který se používá v síti TCP/IP k navázání spojení mezi serverem a klientem. V komunikaci lze vypozorovat tyto typy TCP zpráv:

- SYN: Slouží k navázání a vytvoření spojení.
- ACK: Potvrzení druhé straně, že bylo obdrženo SYN.
- PSH: Flag určující, že data byla odeslána (i když vyrovnávací paměť není zaplněna).[4]
- FIN: Slouží k ukončení spojení.

Dále si můžeme všimnout informací o zdrojové a cílové adrese i portu, polí časových razítek, informací o délce zprávy, rámcích apod.

```
Transmission Control Protocol, Src Port: 323, Dst Port: 53468, Seq: 1, Ack: 59, Len: 19
    Source Port: 323
    Destination Port: 53468
    [Stream index: 2]
    [TCP Segment Len: 19]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3110624858
    [Next Sequence Number: 20    (relative sequence number)]
    Acknowledgment Number: 59    (relative ack number)
    Acknowledgment number (raw): 2596902283
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x018 (PSH, ACK)
```

Obrázek 3: Ukázka z rozpisu informací o paketu

3 Implementace Wireshark dissectoru

Dissectory jsou určeny k analýze určité části dat paketu, můžeme je chápat jako parser protokolu. Struktura dissectorů a postdissectorů je odlišná, pracují odlišně. Postdissector je dissector registrovaný k volání poté, co již byl zavolán dissector – všechna pole protokolu jsou již k dispozici.[5] V tomto projektu se věnujeme implementaci jednoduchého dissectoru.

Dissector z paketů extrahuje potřebná data a umožňuje jejich formátování a výpis, přidání do stromu disekce v programu Wireshark – analyzátor síťových protokolů, umožňující pozorování dění v síti, více informací viz [6]. Soustředíme se zejména na IP adresu, port, délku dat a samotná data.

IP adresa: *IPv4 adresa se obvykle vyjadřuje v desítkovém formátu s tečkami. IP adresa se skládá z čísla sítě a čísla hostitele. Adresa IPv6 jev nezkrácené formě reprezentována jako osm skupin čtyř hexadecimálních číslic, přičemž každá skupina představuje 16 bitů. IP adresa v kombinaci s portem definuje adresu konkrétní služby v konkrétním systému.*[7]

Port: *Port je koncovým bodem komunikace, specifická čísla portů jsou vyhrazena pro identifikaci konkrétních služeb.*[8]

Dissector začíná vytvořením objektu Proto (protokol). Konstruktor tabulky přijímá dva argumenty: jméno a popis, v mém případě jsem náš protokol nazvala ISAproto. Následně vytvářím a přiřazuji pole do ISAproto pomocí třídy `ProtoField` (viz [9]), pomocí třídy `Field` pak deklaruji pole zdrojového a cílového portu pro čtení. Protokol vyžaduje tabulku polí a funkci dissector. Funkce dissector se volá jednou pro každý paket našeho typu.

Funkce dissector má tři parametry: `buffer`, `pinfo` a `tree`. `Buffer` obsahuje vyrovnávací paměť paketu a je to objekt `Tvb`. Obsahuje data, která chceme dissectovat. `Pinfo` obsahuje sloupce seznamu paketů, je objektem `Pinfo`. `Tree` je kořen stromu a je to objekt `TreeItem`.[10]

Ve funkci `isa_proto.dissector(buffer, pinfo, tree)` nejprve zjišťuji délku zprávy, následně řeším opětovné sestavení protokolu TCP. Zde mohou dle [5] nastat následující problémy:

1. Segment paketu TCP může obsahovat pouze první část zprávy.
2. Segment paketu TCP může obsahovat více zpráv.
3. Paket TCP může být uprostřed zprávy, protože předchozí segment nebyl zachycen.
4. Paket TCP může být odříznutý, protože uživatel nastavil Wireshark tak, aby omezil velikost zachycených paketů.
5. Jakákoli kombinace výše uvedených možností.

Problém z prvního bodu řeším nastavením `Pinfo desegment_len` na předdefinovanou hodnotu `DESEGMENT_ONE_MORE_SEGMENT` v případě, že poslaná zpráva nemá očekávané ukončení – pravou kulatou závorku. Druhý a třetí problém moje implementace neřeší. Pro ošetření čtvrtého problému porovnávám délku `Tvb` a `reported_len()`, a pokud se liší, znamená to, že byl paket odříznut. V takovém případě daný paket ignoruji.[5]

Pro získání statusu poslané zprávy používám cyklus while, kdy si načítám znaky do první mezery, znaky pak spojím ve slovo za využití `table.concat`. Dále dle hodnoty zdrojového a cílového portu rozlišuji, kterým směrem mezi serverem a klientem paket putuje, a dle toho ke statusu přidávám, zda se jedná o zprávu ze serveru, či od klienta.

Následně vytvořím strom protokolu a přidám do něj dříve definovaná pole a jejich hodnoty dle [11]. Nakonec načtu tcp.port disekční tabulku a zaregistruji náš protokol pro zpracování portu 32323.[1]

[1] Tento dissector tedy funguje pouze pro komunikaci na portu 32323.

```
ISA Protocol Data
    Message length: 23
    State: Client request: register
    Data: register "isa" "aXNh"
```

Obrázek 4: Výpis ve stromu disekce

```
No.  Time          Source     Destination  Protocol Length Info
  4  0.000073      127.0.0.1  127.0.0.1    ISAPROTO 94     Client request: register
  6  0.000425      127.0.0.1  127.0.0.1    ISAPROTO 93     Server response: ok
 24  27.474674     127.0.0.1  127.0.0.1    ISAPROTO 91     Client request: login
 26  27.475033     127.0.0.1  127.0.0.1    ISAPROTO 122    Server response: ok
 34  41.050522     127.0.0.1  127.0.0.1    ISAPROTO 107    Client request: list
 36  41.050682     127.0.0.1  127.0.0.1    ISAPROTO 73     Server response: ok
 54  70.614517     127.0.0.1  127.0.0.1    ISAPROTO 129    Client request: send
 56  70.614711     127.0.0.1  127.0.0.1    ISAPROTO 85     Server response: ok
 84  160.330442    127.0.0.1  127.0.0.1    ISAPROTO 110    Client request: fetch
 86  160.330716    127.0.0.1  127.0.0.1    ISAPROTO 94     Server response: ok
116  239.992869    127.0.0.1  127.0.0.1    ISAPROTO 109    Client request: logout
118  239.993047    127.0.0.1  127.0.0.1    ISAPROTO 83     Server response: ok
```

Obrázek 5: Zachycená komunikace s využitím implementovaného dissectoru

4 Implementace kompatibilního klienta

Má implementace klienta je rozdělena do třech tříd:

1. `ArgsParser` – třída pro analýzu argumentů programu
2. `CommunicationBase` – třída zajišťující vše okolo připojení k serveru
3. `Client` – třída zapouzdřující funkce klienta

Pro usnadnění zpracování argumentů využívám knihovnu `getopt.h`. Při zpracování argumentů rovnou ověřuji správnost IP adresy a portu pomocí k tomu určených funkcí. V případě ukládání jména a hesla uživatele ukládám heslo rovnou v jeho zakódované podobě. Kódování do base64 provádím za využití volně dostupného kodéru od autora Megumi Tomita viz [12].

Ke zprostředkování spojení využívám především knihoven `arpa/inet.h`, `netdb.h` a `sys/socket.h`. Dle flagu předaného při konstrukci třídy rozlišuji, zda se jedná o socket IPv4, či IPv6 socket. Po vytvoření socketu přiřazuji struktuře pro zpracování internetových adres patřičné hodnoty dle [13]. Navážeme spojení, dochází k poslání zprávy serveru a přijetí odpovědi, ať už jako jednoho celku, nebo v případě rozkouskovaných zpráv po částech. Následně je socket uzavřen a spojení je ukončeno.

Konstruktor třídy `Client` bere jako parametr instanci třídy `ArgsParser`. S jejím využitím postupně volá funkce třídy `CommunicationBase` a ovládá tak spojení se serverem. Součástí třídy `Client` jsou funkce zpracovávající zprávu od serveru i data posílaná na server.

Vesměs se jedná o funkce parsující text jako například funkce řešení escapování speciálních znaků (implementováno pouze escapování `"` a `/` ve zprávě od klienta serveru, opačně i znak nového řádku), ořezávání hlaviček, extrakce zprávy z poslaných dat apod.

Program končí s návratovou hodnotou 0 po úspěšném splnění příkazu, v jiném případě je návratová hodnota rovna jedné a na výstup je vypsán problém, který nastal.

5 Spuštění programu client

K sestavení projektu slouží přiložený soubor Makefile, spuštění sestavení je potřeba provést příkazem `make`, je tedy nutné, aby na daném zařízení byl k dispozici `GNU Make`. Po sestavení se vytvoří vytvoří spustitelný klient s názvem client. Vyčištění lze provést příkazem `make clean`.

Spuštění programu se pak provádí pomocí příkazu[2]

```
sudo ./client [ <option> ... ] <command> [<args>] ...
```

Nápovědu lze získat zadáním argumentu `-h`, či `--help`, nápověda ilustruje použití programu a význam jednotlivých argumentů a vypadá následovně:

```
usage: client [ <option> ... ] <command> [<args>] ...
Options:
[-h | --help]
  Show this help
[-a | --adress]  <address>
  Server hostname or address to connect to (default localhost)
[-p | --port]    <port>
  Server port to connect to (default 32323)
--
Do not treat any remaining argument as a switch (at this level)
Supported commands:
 register <username> <password>
 login <username> <password>
 list
 send <recipient> <subject> <body>
 fetch <id>
 logout
```

Výchozí hodnota adresy odpovídá IPv6 localhost adrese a port je defaultně nastaven na 32323.

6 Testování

Projekt byl testován na referenční image – isa-2021-koutensky.ova pracující na systému manjaro linux. Pro porovnání výsledků testování byla využita komunikace referenčního a klienta zachycená v aplikaci Wireshark.

[2]Bez příkazu `sudo` nemusí program fungovat správně

Literatura

[1] KUROSE, J. F.; ROSS, K. W.: *Computer networking: A top-down approach*. Pearson, sedmé vydání, 2017, ISBN 978-0-13-359414-0.

[2] What is a packet?, Cloudflare, Inc., [online], 2021.
Dostupné z: `https://www.cloudflare.com/learning/network-layer/what-is-a-packet/`

[3] GORALSKI, W.: Chapter 25 - SMTP and Email. In *The Illustrated Network (Second Edition)*, Boston: Morgan Kaufmann, second edition vydání, 2017, ISBN 978-0-12-811027-0, s. 637–659.
Dostupné z: `https://www.sciencedirect.com/science/article/pii/B9780128110270000254`

[4] *Understanding PSH ACK TCP Flags*. howtouselinux, stretch, [online], Nov 2021.
Dostupné z: `https://www.howtouselinux.com/post/psh-ack-tcp-flags#viewer-ceudf`

[5] *Dissectors*. GitLab, wiki.wireshark.org, [online], Aug 2020.
Dostupné z: `https://gitlab.com/wireshark/wireshark/-/wikis/Lua/Dissectors`

[6] *Wireshark Download*. Wireshark · Go Deep., wireshark.org, [online].
Dostupné z: `https://www.wireshark.org/`

[7] PARZIALE, L.: *TCP/IP tutorial and technical overview*. IBM International Technical Support Organization, 2006, ISBN 978-0130676108.

[8] *Port (computer networking)*. Wikipedia, Wikimedia Foundation, [online], Apr 2021.
Dostupné z: `https://en.wikipedia.org/wiki/Port_(computer_networking)`

[9] *11.2. Obtaining Dissection Data*. wireshark.org, [online].
Dostupné z: `https://www.wireshark.org/docs/wsdg_html_chunked/lua_module_Field.html#lua_class_Field`

[10] SUNDLAND, M.: *Creating a Wireshark dissector in Lua - part 1 (the basics)*. Mika's tech blog, github.com, [online], Nov 2017.
Dostupné z: `https://mika-s.github.io/wireshark/lua/dissector/2017/11/04/creating-a-wireshark-dissector-in-lua-1.html`

[11] *11.7. Adding Information To The Dissection Tree*. wireshark.org, [online].
Dostupné z: `https://www.wireshark.org/docs/wsdg_html_chunked/lua_module_Tree.html`

[12] Tomita, M.: *C single header base64 decode/encoder*. Gist, [online], 2016.
Dostupné z: `https://gist.github.com/tomykaira/f0fd86b6c73063283afe550bc5d77594`

[13] *struct sockaddr_in, struct in_addr*. [online].
Dostupné z: `https://www.gta.ufrj.br/ensino/eel878/sockets/sockaddr_inman.html`

Seznam obrázků