Data sporządzenia - 17.01.2026

Programowanie Sieciowe – Projekt

Zespół 32: **Katarzyna Kanicka,** Jan Mizera, Andrii-Stepan Pryimak

Treść zadania

Trzeba było zaprojektować i zaimplementować bezpieczny protokoł komunikacyjny oparty na TCP, realizujący architekturę klient-serwer. Protokół "mini TLS", zapewnia poufność, integralność oraz autentyczność przesyłanych danych.

Zgodnie z założeniami projektu, implementacja obejmuje:

- Wymianę kluczy (Handshake) za pomocą algorytmu Diffiego-Hellmana.
- Szyfrowanie danych strumieniowe.
- Weryfikację integralności mechanizmem HMAC (SHA-256) w schemacie *Encrypt-then-MAC*.
- Obsługę wielu klientów jednocześnie.

Opis rozwiązania problemu

Architektura i Narzędzia

Projekt zrealizowano w języku python + *Encrypt-then-MAC* (W1). Komunikacja sieciowa oparta jest na gniazdach (socket) w trybie blokującym.

- **Serwer (tcp-server.py):** Wykorzystuje wielowątkowość. Główny wątek nasłuchuje nowych połączeń, a dla każdego klienta tworzony jest osobny wątek obsługujący komunikację.
- **Klient (tcp-client.py):** Aplikacja konsolowa z interaktywnym menu, umożliwiająca nawiązywanie połączenia, wysyłanie wiadomości i zamykanie sesji na żądanie użytkownika.
- **Protokół (protocol.py):** Współdzielony moduł zawierający logikę pakowania wiadomości, szyfrowania oraz obsługi kryptograficznej.

Struktura Wiadomości i Protokół

Zgodnie z założeniami wstępnymi zaimplementowano

- ClientHello (0x01): Nieszyfrowana wiadomość inicjująca.
- ServerHello (0x02): Nieszyfrowana odpowiedź serwera.
- Wiadomość Szyfrowana (0x04): Kontener dla bezpiecznych danych

Algorytmy Kryptograficzne

Zgodnie z wariantem W1:

1. **Wymiana Kluczy:** Algorytm Diffie-Hellman. Po wymianie ClientHello i ServerHello obie strony obliczają wspólny sekret session_key.
2. **Szyfrowanie:** Zastosowano szyfr strumieniowy. Klucz szyfrujący dla danej wiadomości generowany jest jako XOR klucza sesji i licznika wiadomości, co służy jako ziarno generatora liczb pseudolosowych. Zapewnia to unikalność szyfrogramu dla każdej wiadomości.
3. **Integralność:** Schemat *Encrypt-then-MAC*. Do każdego pakietu dołączany jest skrót HMAC-SHA256, weryfikowany przez odbiorcę przed próbą deszyfrowania

Konfiguracja testowa(Doker)

projekt > client > $ run.sh

```
#!/bin/bash

docker rm -f z32-client-projekt
docker build -t z32-client-docker -f dockerfile ..
docker run -it --cap-add=NET_ADMIN --network z32_network --name z32-client-projekt z32-client-docker
```

```
HOST = "z32-server-projekt"
PORT = 5000
P = 2147483647
G = 2
```

projekt > server > $ run.sh

```
#!/bin/bash

N_CLIENT=${1:-1}

docker rm -f z32-server-projekt
docker build -t z32-server-docker -f dockerfile ..
docker run -it --network-alias z32-server-projekt --network z32_network --name z32-server-projekt z32-server-docker $N_CLIENT
```

```
HOST = "0.0.0.0"
PORT = 5000

connections = {}
connections_lock = threading.Lock()
client_threads = []
```

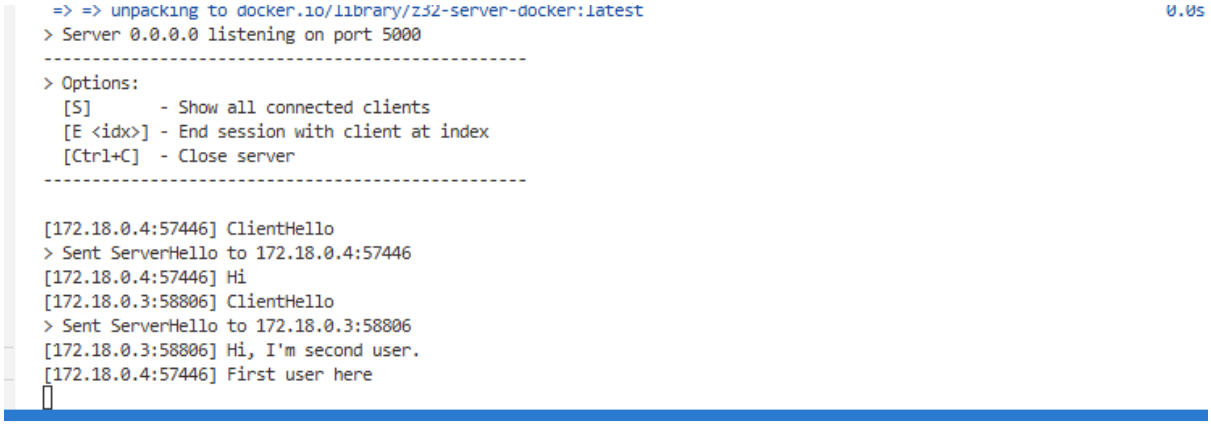
Serwer: z32-server-projekt (Docker) **Klient:** z32-client-project (Docker)

Sieć: Docker network z32_network **Port:** 5000

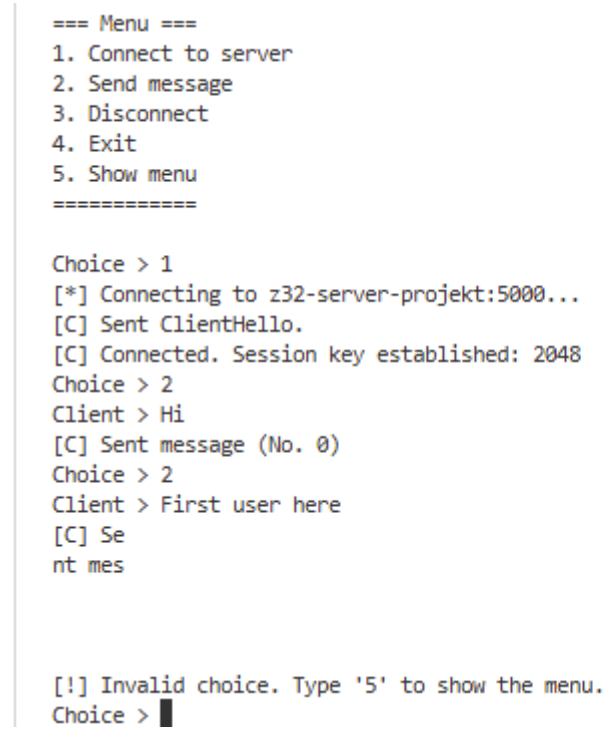
Testowanie i wyniki

Uruchomienie i Handshake

Logi Serwera: Widoczny start serwera, odebranie połączenia oraz wymiana kluczy. Widać też że serwer dobrze radzi z n klientami.



Client:



Zakończenie Sesji

Analiza w Wireshark

Serwer:

```
> Server 0.0.0.0 listening on port 5000
--------------------------------------------------
> Options:
  [S]       - Show all connected clients
  [E <idx>] - End session with client at index
  [Ctrl+C]  - Close server
--------------------------------------------------

[172.21.32.3:60180] ClientHello
> Sent ServerHello to 172.21.32.3:60180
S
> Current connections:
0. ('172.21.32.3', 60180)
[172.21.32.3:60180] test message
[172.21.32.3:60180] another test
[172.21.32.3:60180] EndSession (EndSession)
Disconnected 172.21.32.3:60180
S
> Current connections:
No active connections
^C
> Server shutting down...
```

Klient:

```
Choice > 1
[*] Connecting to z32-server-projekt:5000...
[C] Sent ClientHello.
[C] Connected. Session key established: 134217728
Choice > 5
=== Menu ===
1. Connect to server
2. Send message
3. Disconnect
4. Exit
5. Show menu
============

Choice > 2
Client > test message
[C] Sent message (No. 0)
Choice > 2
Client > another test
[C] Sent message (No. 1)
Choice > 3
[*] Closing session...
[C] Sent EndSession.
[C] Disconnected from server.
Choice > 4
Exiting...
```

Wireshark

No.	Time	Source	Destination	Protocol	Length	Info
34	13.567985	172.21.32.3	172.21.32.2	OML	79	OML
36	13.568763	172.21.32.2	172.21.32.3	RSL	71	unknown 0
44	55.366499	172.21.32.3	172.21.32.2	RSL	116	unknown 13
51	71.221677	172.21.32.3	172.21.32.2	RSL	116	unknown 13
54	84.261661	172.21.32.3	172.21.32.2	RSL	114	UNIT DATA INDication (DTAP) (TP)

Strumień TCP:

```
00000000  01 7f ff ff ff 00 00 00  02 02 00 00 00           ........ .....
    00000000  02 00 00 08 00                                    .....
0000000D  04 00 00 00 0d 01 e2 aa  15 8c 72 5f 3e f0 a2 f9   ........ ..r_>...
0000001D  cb 27 a9 7a 42 97 fa f2  73 33 74 bd 35 19 71 d4   .'.zB... s3t.5.q.
0000002D  96 04 a8 3d 4f 11 04 bb  03 16 2c c6 7f 74 e2 89   ...=O... ..,..t..
0000003D  c7 a2                                             ..
0000003F  04 00 00 00 0d 01 0e 79  ca ea e0 1b be fa 99 1d   .......y ........
0000004F  57 bb af d7 72 42 5f 5f  3a 83 17 55 f8 8d a2 2b   W...rB__ :..U...+
0000005F  3f fd e8 38 95 2f 50 15  19 a2 e2 5f 46 7c 24 a8   ?..8./P. ..._F|$.
0000006F  13 b4                                            ..
00000071  04 00 00 00 0b 02 b0 71  41 1d 14 e0 5f 4e a3 71   .......q A..._N.q
00000081  a6 e9 ff b2 3c 2d 45 dd  bf 5e 5b 29 07 0b 57 0b   ....<-E. .^[)..W.
00000091  6c dc 3b 0d 0b f5 a3 20  e2 b9 18 08 1c 8e db 0e   l.;....  ........
```

Logi z programu Wireshark zgadzają się z odbytą komunikacją między **klientem** (172.21.32.3) a **serwerem** (172.21.32.2).

Można zauważyć po kolei:

- ClientHello

header	module p	base g	client public key
0x01	0x7FFFFFFF	0x00000002	0x02000000

- ServerHello – 02 00 00 08 00

header	server public key
0x02	0x00000800

- Trzy szyfrowane wiadomości (dwie zwykłe i EndSession)

header	length	type	message	MAC
0x04	0x000000 0d	0x01	0xe2aa158c725f3ef 0a2f9cb27	0xa97a4297faf2733374bd3519 71d49604a83d4f1104bb03162 cc67f74e289c7a2
0x04	0x000000 0d	0x01	0x0e79caeae01bbe fa991d57bb	0xafd772425f5f3a831755f88da 22b3ffde838952f501519a2e25f 467c24a813b4
0x04	0x000000 0b	0x02	0xb071411d14e05f 4ea371	0xa6e9ffb23c2d45ddbf5e5b29 070b570b6cdc3b0d0bf5a320e 2b918081c8edb0e

Weryfikacja szyfrowanych wiadomości

Klucz sesyjny: 134217728

Wiadomość 0.

- Seed generatora losowego = 134217728 + 0 = 134217728
- Wygenerowany klucz = '\x96\xcff\xf8R2[\x83\xd1\x98\xacB'
- Treść = „test message”
- Wynik szyfrowania OTP: e2 aa 15 8c 72 5f 3e f0 a2 f9 cb 27

Wiadomość 1.

- Seed generatora losowego = 134217728 + 1 = 134217729
- Wygenerowany klucz = ‘o\x17\xa5\x9e\x88~\xcc\xda\xedx$\xcf’
- Treść = „another test”
- Wynik szyfrowania OTP: 0e 79 ca ea e0 1b be fa 99 1d 57 bb

Wiadomość 2.

- Seed generatora losowego = 134217728 + 2 = 134217730
- Wygenerowany klucz = ‘\xf5\x1f%Nq\x93,'\xcc\x1fa\x12’
- Treść = „EndSession”
- Wynik szyfrowania OTP: b0 71 41 1d 14 e0 5f 4e a3 71

Wnioski: Porównując otrzymany wynik z tym z programu Wireshark można stwierdzić, że protokół poprawnie szyfruje wiadomości.

Problemy napotkane i rozwiązania

W trakcie realizacji zadania nie wystąpiły istotne problemy techniczne ani trudności implementacyjne.

Wnioski

Zrealizowany projekt spełnia wszystkie postawione wymagania. Protokół skutecznie zabezpiecza transmisję przed pasywnym nasłuchem (szyfrowanie) oraz aktywną modyfikacją (HMAC). Zastosowanie architektury wielowątkowej pozwala na stabilną obsługę wielu klientów. Mechanizm *Encrypt-then-MAC* sprawdził się jako skuteczna metoda ochrony integralności.