

Programowanie sieciowe

Miłosz Cieśla, Filip Ryniewicz, Aleksander Szymczyk

Sprawozdanie końcowe z projektu

15.01.2025

1 Treść zadania

Treść projektu

Celem projektu jest zaprojektowanie oraz implementacja szyfrowanego protokołu opartego na protokole TCP, tzw. mini TLS.

Założenia projektu

- Architektura klient-serwer.
- Serwer jest w stanie obsłużyć kilku klientów jednocześnie (liczba klientów powinna być podawana jako parametr uruchomienia, nie hardcodowana).
- Klient inicjuje połączenie z serwerem poprzez wysłanie wiadomości **ClientHello** (wiadomość nieszyfrowana), na którą serwer odpowiada wiadomością **ServerHello** (wiadomość nieszyfrowana).
- Sesja może zostać zakończona zarówno przez klienta, jak i przez serwer poprzez wysłanie wiadomości **EndSession**. Po odebraniu **EndSession** należy ponownie wysłać **ClientHello**, aby rozpocząć nową sesję.
- Wszystkie wiadomości poza **ClientHello** i **ServerHello** muszą być szyfrowane.
- Potencjalny napastnik po przechwyceniu wiadomości nie powinien być w stanie nic z niej odczytać.

Wymagania implementacyjne

- Wiadomości **ClientHello** i **ServerHello** służą do wymiany kluczy szyfrujących. Po ich wymianie klient oraz serwer muszą być w posiadaniu tego samego klucza szyfrującego, który będzie używany do szyfrowania kolejnych wiadomości.
- Użycie algorytmu wymiany kluczy, np. **Diffie-Hellman key exchange**. Samo szyfrowanie wiadomości może (ale nie musi) być zrealizowane prostym **OTP** (ang. One Time Pad).
- Komunikacja ma być sterowana z wiersza poleceń. Dostępne opcje:
 - Serwer:
 - * Zakończ połączenie dla wybranego klienta.
 - Klient:
 - * Zainicjuj połączenie z serwerem.
 - * Wyślij wiadomość (treść wiadomości jest nieistotna).
 - * Zakończ połączenie z serwerem.
- Serwer powinien wyświetlać listę wszystkich obecnie połączonych klientów oraz odbierane wiadomości.
- Komunikacja ma się odbywać w sieci **dockerowej**.
- Całość powinna być uruchamiana minimalną liczbą komend.

1.1 Wybrany wariant funkcjonalny

W1 – wykorzystanie mechanizmu **encrypt-then-mac** dla wysyłanych szyfrowanych wiadomości jako mechanizm integralności i autentyczności, implementacja w Pythonie.

2 Struktura wiadomości

- ClientHello:
 - niezaszyfrowany klucz publiczny klienta (domyślna długość 32 bajty)
 - zaszyfrowany przez klienta klucz wspólny
- ServerHello
 - niezaszyfrowany klucz publiczny serwera (domyślna długość 32 bajty)
 - zaszyfrowany przez serwer klucz wspólny
- Wiadomość
 - Bajty 0 - 31: MAC
 - Bajty 32+: szyfrogram
- EndSession - używa tej samej struktury co zwykła wiadomość, przesyłając na bajtach 32+ zaszyfrowaną wiadomość specjalną o treści "exit"

2.1 Przykładowy przebieg komunikacji

Krok	Klient	Serwer
1	<ul style="list-style-type: none">• Ustalanie klucza sesyjnego.	<ul style="list-style-type: none">• Utworzenie nowego wątku do obsługi połączenia.• Ustalanie klucza sesyjnego.
2	<ul style="list-style-type: none">• Klient wczytuje wiadomość z terminala.• Wiadomość jest szyfrowana za pomocą klucza sesyjnego.• Generowany jest kod MAC na podstawie szyfrogramu i klucza sesyjnego.• Wiadomość jest łączona w podaną wyżej strukturę i wysyłana do serwera.	<ul style="list-style-type: none">• Serwer przyjmuje wiadomość.• Dzieli ją na kod MAC oraz szyfrogram.• Oblicza kod MAC na podstawie klucza sesyjnego i szyfrogramu. Następnie sprawdza zgodność obliczonego kodu MAC z tym otrzymanym od klienta.• W zależności od zgodności kodów MAC:<ul style="list-style-type: none">– Odsyła wiadomość o treści: "ERROR: MAC verification failed" jeśli kody nie były zgodne.– Odsyła wiadomość o treści: "ACK: received" jeśli autentyczność klienta była pozytywna.
3	<ul style="list-style-type: none">• Klient przyjmuje odpowiedź serwera.• Wiadomość jest odszyfrowana i wyświetlana na terminalu.	<ul style="list-style-type: none">• W przypadku pozytywnej weryfikacji autentyczności i integralności wiadomość jest odszyfrowana i wyświetlana na terminalu.• Oczekiwanie na kolejne wiadomości.

Tabela 1: Sekwencja wymiany kluczy pomiędzy Klientem a Serwerem (handshake).

3 Opis wykorzystanych algorytmów

- Algorytm szyfrowania: Własna implementacja szyfru Vigenère’a, dla zachowania pierwotnej formy wiadomości używamy wszystkich znaków możliwych do wyświetlenia na konsoli jako słownika. Znaki wiadomości są przesuwane względem niego o wartość odpowiadającego im znaku klucza z uwzględnieniem cykliczności.
- Generowanie kodu MAC: wykorzystujemy gotową implementację HMAC opartą na funkcji skrótu SHA256 z biblioteki standardowej pythona.
- Generowanie klucza: Używając wszystkich znaków możliwych do wyświetlenia na konsoli, generujemy ich losowy ciąg o zadanej długości.

4 Sposób realizacji mechanizmu integralności i autentyczności dla szyfrowanych wiadomości

Skorzystaliśmy z podejścia Encrypt-then-MAC. Polega ono na najpierw zaszyfrowaniu wiadomości, a następnie generowaniu z niej kodu MAC. Zdecydowaliśmy się na to rozwiązanie, gdyż jest ono najpopularniejsze oraz jest uważane za najbezpieczniejsze.

4.1 Autentyczność

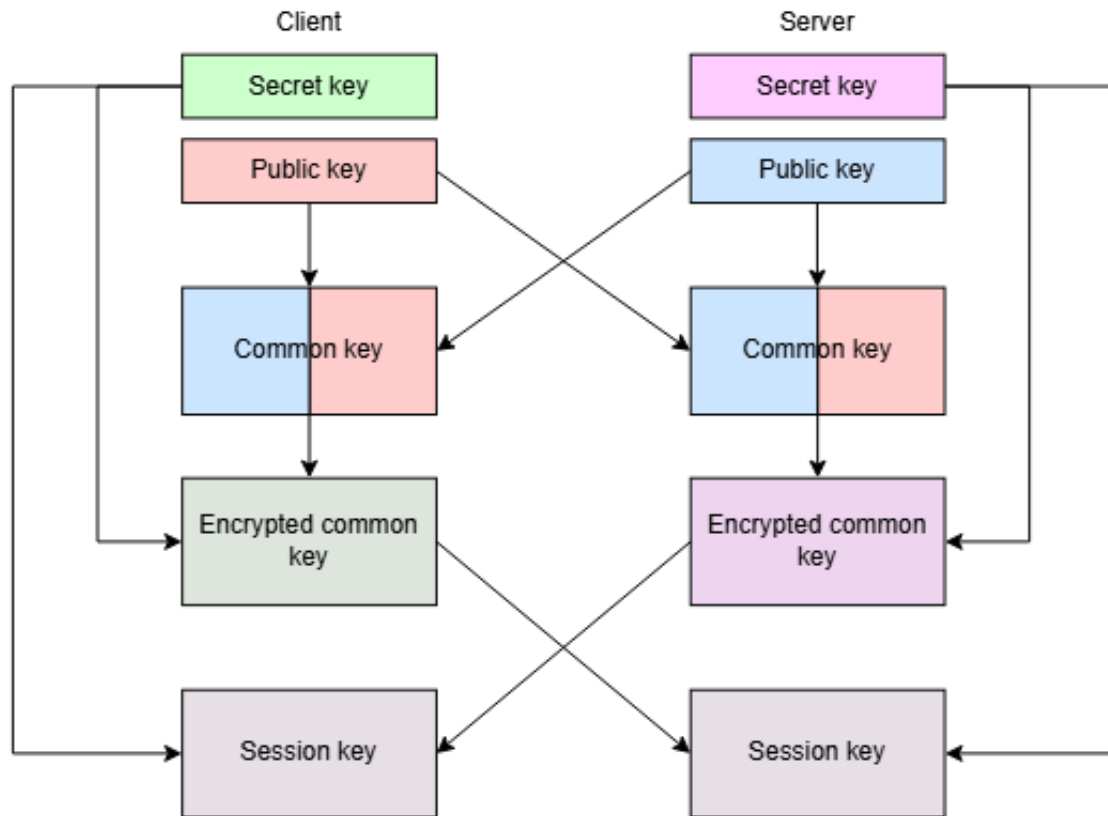
Zapewnienie autentyczności opiera się na kodzie MAC. Jeśli odbiorca, korzystając z tajnego klucza sesyjnego, wygeneruje MAC zgodny z tym, który został dołączony do wiadomości, można założyć, że wiadomość pochodzi od nadawcy znającego klucz sesyjny. W przeciwnym razie, próba sfałszowania lub modyfikacji wiadomości przez atakującego, który nie zna klucza sesyjnego, spowoduje niezgodność kodów MAC i odrzucenie wiadomości.

4.2 Integralność

Integralność jest również zapewniona dzięki użyciu kodu MAC. Nadawca generuje kod MAC dla szyfrogramu, bazując na tajnym kluczu sesyjnym, i dołącza go do wiadomości. Odbiorca, po otrzymaniu wiadomości, samodzielnie oblicza MAC dla otrzymanego szyfrogramu i porównuje go z dołączonym kodem. Jeśli kody są zgodne, można mieć pewność, że treść wiadomości nie uległa modyfikacji w trakcie przesyłu.

5 Schemat ustalania klucza sesyjnego

Wymiana kluczy to nieco zmieniona wersja klasycznego Diffiego–Helmana. W opisanym rozwiązaniu klucze są ze sobą najpierw łączone (konkatenowane), a następnie szyfrowane kluczami prywatnymi Klienta i Serwera, co wymaga dodatkowych kroków, ale nadal służy temu samemu celowi: uzgodnieniu wspólnego klucza sesyjnego.



Rysunek 1: Diagram ustalania klucza sesyjnego (handshake)

Krok	Klient	Serwer
1	<ul style="list-style-type: none"> Generuje swój <code>secret_key</code> oraz <code>public_key</code>. Wysyła <code>public_key</code> do serwera. 	<ul style="list-style-type: none"> Odbiera <code>client_public_key</code> (klucz publiczny Klienta). Generuje swój <code>secret_key</code> oraz <code>public_key</code>. Wysyła do Klienta swój klucz publiczny <code>server_key</code>.
2	<ul style="list-style-type: none"> Odbiera <code>server_public_key</code> (klucz publiczny Serwera). Tworzy <code>common_key</code> = $(\text{server_public_key} + \text{client_public_key})$. 	<ul style="list-style-type: none"> Tworzy <code>common_key</code> = $(\text{server_public_key} + \text{client_public_key})$.
3	<ul style="list-style-type: none"> Szyfruje <code>common_key</code> swoim <code>secret_key</code> i wysyła wynik do Serwera. 	<ul style="list-style-type: none"> Szyfruje <code>common_key</code> swoim <code>secret_key</code> i wysyła wynik do Klienta.
4	<ul style="list-style-type: none"> Odbiera od Serwera <code>common_encrypted_server</code>. 	<ul style="list-style-type: none"> Odbiera od Klienta <code>common_encrypted_client</code>.
5	<ul style="list-style-type: none"> Szyfruje <code>common_encrypted_server</code> swoim <code>secret_key</code>. Otrzymany klucz jest wspólnym kluczem sesji. 	<ul style="list-style-type: none"> Szyfruje <code>common_encrypted_client</code> swoim <code>secret_key</code>. Otrzymany klucz jest wspólnym kluczem sesji.

Tabela 2: Sekwencja wymiany kluczy pomiędzy Klientem a Serwerem (handshake).

- `common_key` - konkatenacja obydwu kluczy (`server_public_key` + `client_public_key`)
- `common_encrypted_server` - `common_key` zaszyfrowany kluczem prywatnym serwera
- `common_encrypted_client` - `common_key` zaszyfrowany kluczem prywatnym klienta
- `session_key` - `common_key` zaszyfrowany kluczem prywatnym serwera i kluczem prywatnym klienta

6 Śledzenie komunikacji za pomocą Wireshark

6.1 Ustalenie klucza sesyjnego

1	0.000000	127.0.0.1	127.0.0.1	TCP	68	58283 → 8000 [SYN, Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=114612013 TSecr=0 SACK_PERM
2	0.000065	127.0.0.1	127.0.0.1	TCP	68	8000 → 58283 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=515066995 TSecr=114612013 SACK_PERM
3	0.000086	127.0.0.1	127.0.0.1	TCP	56	58283 → 8000 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=114612013 TSecr=515066995
4	0.000101	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 8000 → 58283 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=515066995 TSecr=114612013
5	0.000148	127.0.0.1	127.0.0.1	TCP	88	58283 → 8000 [PSH, ACK] Seq=1 Ack=1 Win=408256 Len=32 TSval=114612013 TSecr=515066995
6	0.000177	127.0.0.1	127.0.0.1	TCP	56	8000 → 58283 [ACK] Seq=1 Ack=33 Win=408256 Len=0 TSval=515066995 TSecr=114612013
7	0.000448	127.0.0.1	127.0.0.1	TCP	88	8000 → 58283 [PSH, ACK] Seq=1 Ack=33 Win=408256 Len=32 TSval=515066995 TSecr=114612013
8	0.000472	127.0.0.1	127.0.0.1	TCP	56	58283 → 8000 [ACK] Seq=33 Ack=33 Win=408256 Len=0 TSval=114612013 TSecr=515066995
9	0.000484	127.0.0.1	127.0.0.1	TCP	120	8000 → 58283 [PSH, ACK] Seq=33 Ack=33 Win=408256 Len=64 TSval=515066995 TSecr=114612013
10	0.000497	127.0.0.1	127.0.0.1	TCP	56	58283 → 8000 [ACK] Seq=33 Ack=97 Win=408192 Len=0 TSval=114612013 TSecr=515066995
11	0.000526	127.0.0.1	127.0.0.1	TCP	120	58283 → 8000 [PSH, ACK] Seq=33 Ack=97 Win=408192 Len=64 TSval=114612013 TSecr=515066995
12	0.000541	127.0.0.1	127.0.0.1	TCP	56	8000 → 58283 [ACK] Seq=97 Ack=97 Win=408192 Len=0 TSval=515066995 TSecr=114612013

Rysunek 2: Nawiązanie połączenia i wymiana kluczy

1-4. Standardowe nawiązanie połączenia TCP między klientem a serwerem.

5. Początek wymiany kluczy. Klient wysła swój klucz publiczny serwerowi (ClientHello).

6. ACK od serwera

7. Serwer wysła swój klucz publiczny.

8. ACK od klienta

9. Serwer wysła `server_common_encrypted`.

10. ACK od serwera

11. Klient `client_common_encrypted`.

12. ACK od klienta

```
0000 02 00 00 00 45 00 00 54 00 00 40 00 40 06 00 00 .....E..t..@...
0010 7f 00 00 01 7f 00 00 01 e3 ab 1f 40 4b 03 60 ac .....@...dK...
0020 98 a3 64 4b 80 18 18 eb fe 48 00 00 01 01 08 0a .....LS.....H.....
0030 06 64 d7 2d 1e b3 4c 73 4b 4e 50 00 4b 57 39 22 .....LS KNP@KW9"
0040 67 2e 3f 71 3d 62 7a 4c 39 6c 4a 77 48 47 2a 65 g..?q=bzL 9LJwHG.e
0050 2b 56 41 78 7b 47 33 45 +VAX{G3E
Establishing connection...
[SENT] Client_key b'KNP@KW9"q.?q=bzL9LJwHG.e+VAX{G3E' l
ngth: 32
[RECEIVED] Server_key :6EVk&(<T+j
UW VI::44,ywCu3hnx2U length: 32
[SENT] Common encrypted b'+uic~|PbZlBWV-|><7|x0c/ 6yHXk
```

5. Przesłanie `client_key` przez klienta do serwera

```
0000 02 00 00 00 45 00 00 54 00 00 40 00 40 06 00 00 .....E..t..@...
0010 7f 00 00 01 7f 00 00 01 1f 40 e3 ab 98 a3 64 4b .....@...dK...
0020 4b 03 60 cc 80 18 18 eb fe 68 00 00 01 01 08 0a .....H.....
0030 1e b3 4c 73 06 d4 d7 2d 3a 47 45 56 6b 26 28 3c .....LS.....:GEVk&(<
0040 54 2b 6a 0a 55 57 20 56 49 2d 3a 34 34 2c 79 77 T+j UW V I::44,yw
0050 43 75 33 68 6e 78 32 55 Cu3hnx2U
Choose an option:
[RECEIVED] Client_key KNP@KW9"q.?q=bzL9LJwHG.e+VAX{G3E l
ngth: 32
[SENT] Server_key b':6EVk&(<T+j\UW VI::44,ywCu3hnx2U' l
ngth: 32
[SENT] Common encrypted b'"/AEoR1le{#}.L\rm\NQwW\x0b7^
```

7. Przesłanie `server_key` przez serwer do klienta

```
0000 02 00 00 00 45 00 00 74 00 00 40 00 40 06 00 00 .....E..t..@...
0010 7f 00 00 01 7f 00 00 01 1f 40 e3 ab 98 a3 64 6b .....@...dK...
0020 4b 03 60 cc 80 18 18 eb fe 68 00 00 01 01 08 0a .....H.....
0030 1e b3 4c 73 06 d4 d7 2d 27 5e 2f 41 45 6f 52 31 .....LS....."/AEoR1
0040 7c 65 5b 23 50 2e 4c 0d 6d 5c 4e 51 77 57 00 37 |e{#}.L\rm\NQwW/7
0050 5e 00 74 30 50 71 29 4b 42 20 5e 21 25 66 7e 5c ^t0Pq|KB ^!%f~\
0060 51 68 4c 20 61 73 5f 5e 5e 77 68 3d 2a 72 44 60 QhL as ^!%f~\
0070 6c 6d 21 67 69 7a 2a 75 lmgiz+u
[SENT] Server_key b':6EVk&(<T+j\UW VI::44,ywCu3hnx2U' l
ngth: 32
[SENT] Common encrypted b'"/AEoR1le{#}.L\rm\NQwW\x0b7^
tt0Pq|KB ^!%f~\QhL as ^!%f~\rD`lmgiz+u' length: 64
[RECEIVED] Encrypted_client +ujc~|PbZlBWV-J|><7
(,6yHXkS9
&.FBuCj?| m$0_Tr|>Kq.r868g:|R|/'X length: 64
```

9. Przesłanie `server_common_encrypted` z serwera do klienta

```
0000 02 00 00 00 45 00 00 74 00 00 40 00 40 06 00 00 .....E..t..@...
0010 7f 00 00 01 7f 00 00 01 e3 ab 1f 40 4b 03 60 cc .....@...dK...
0020 98 a3 64 4b 80 18 18 eb fe 68 00 00 01 01 08 0a .....dK...h.....
0030 06 d4 d7 2d 1e b3 4c 73 2b 75 6a 63 7e 7c 50 62 .....LS +ujc~|Pb
0040 5a 21 42 57 76 2d 4a 7c 3e 3c 37 0c 28 2c 26 79 ZlBWV-J|><7,(6y
0050 48 58 6b 53 39 26 5f 2e 46 42 75 43 6a 3f 7c 09 HXKS9&.FBuCj?|
0060 6d 24 30 5f 54 72 5d 3e 4b 71 2e 72 38 47 38 67 m$0_Tr|>Kq.r868g
0070 3a 5d 52 29 2f 2f 60 58 j|R|/'X
[RECEIVED] Server_key :6EVk&(<T+j
UW VI::44,ywCu3hnx2U length: 32
[SENT] Common encrypted b'+ujc~|PbZlBWV-J|><7|x0c/ 6yHXk
S9&.FBuCj?|tm$0_Tr|>Kq.r868g:|R|/'X length: 64
m\NQwW|VED encrypted_server "/AEoR1le{#}.L
7^ t0Pq|KB ^!%f~\QhL as ^!%f~\rD`lmgiz+u l
length: 64
```

11. Przesłanie `client_common_encrypted` z klienta do serwera

6.2 Przesyłanie wiadomości

No.	Time	Source	Destination	Protocol	Length	Info
2	1.260960	127.0.0.1	127.0.0.1	TCP	97	58332 → 8000 [PSH, ACK] Seq=1 Ack=1 Win=6378 Len=41 TSval=2558535485 TSecr=3043822815
3	1.261032	127.0.0.1	127.0.0.1	TCP	56	8000 → 58332 [ACK] Seq=1 Ack=42 Win=6377 Len=0 TSval=3043837749 TSecr=2558535485
4	1.764238	127.0.0.1	127.0.0.1	TCP	90	8000 → 58332 [PSH, ACK] Seq=1 Ack=42 Win=6377 Len=34 TSval=3043838252 TSecr=2558535485
5	1.764323	127.0.0.1	127.0.0.1	TCP	56	58332 → 8000 [ACK] Seq=42 Ack=35 Win=6377 Len=0 TSval=2558535988 TSecr=3043838252

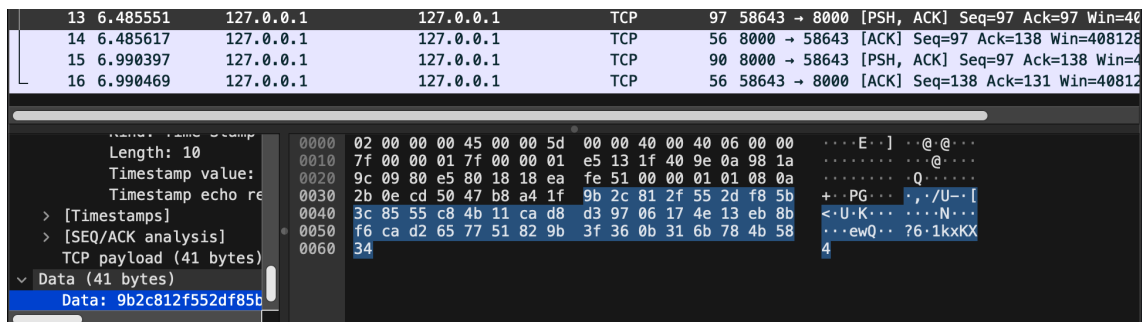
Rysunek 3: Przesyłanie i odbiór

1. Wysłanie wiadomości przez klienta do serwera
2. ACK od serwera
3. Przesłanie z powrotem potwierdzenia odbioru wiadomości w postaci: "ACK: addr received"
4. ACK od klienta

6.3 Weryfikacja szyfrowania

W celu manualnej weryfikacji skuteczności deszyfrowania wiadomości przez serwer wykonaliśmy następujące kroki:

1. Dodaliśmy w kodzie serwera funkcję wypisującą klucz sesyjny zakodowany w formacie Base64.
2. Skopiowaliśmy zakodowany klucz sesyjny.
3. Wysłaliśmy z klienta wiadomość o treści: `wiadomosc`.
4. Z przechwyconego pakietu w programie *Wireshark* skopiowaliśmy sekcję `Data`, zawierającą kod MAC oraz szyfrogram.
5. Odkodowaliśmy klucz sesyjny do jego pierwotnej postaci oraz zdekodowaliśmy wiadomość z formatu heksadecymalnego.
6. Użyliśmy klucza sesyjnego do odszyfrowania wiadomości.
7. Jak widać, odszyfrowana wiadomość ma treść zgodną z oryginałem.



13	6.485551	127.0.0.1	127.0.0.1	TCP	97	58643 → 8000 [PSH, ACK] Seq=97 Ack=97 Win=408128
14	6.485617	127.0.0.1	127.0.0.1	TCP	56	8000 → 58643 [ACK] Seq=97 Ack=138 Win=408128
15	6.990397	127.0.0.1	127.0.0.1	TCP	90	8000 → 58643 [PSH, ACK] Seq=97 Ack=138 Win=408128
16	6.990469	127.0.0.1	127.0.0.1	TCP	56	58643 → 8000 [ACK] Seq=138 Ack=131 Win=408128

Details of packet 14:

- Length: 10
- Timestamp value: 0020
- Timestamp echo received: 0030
- [Timestamps]
- [SEQ/ACK analysis]
- TCP payload (41 bytes)
- Data (41 bytes)
- Data: 9b2c812f552df85b

Rysunek 4: Komunikacja przedstawiona w WireShark (na niebiesko zaznaczone pole Data)


```
manual_decryption.py > ...
1 import base64
2 from utils import decrypt
3
4 base64_session_key = "T19fXwpibXZ9bFpSMjJMLiFAJXdVT0pgJCg2Y3oiP21LMyIiTzplcVBgDWY9biN9bm5yem13PSgKfC5cXC1gZw=="
5 message_hex = "9b2c812f552df85b3c8555c84b11cad8d39706174e13eb8bf6cad2657751829b3f360b316b784b5834"
6
7 session_key = base64.b64decode(base64_session_key).decode('ascii')
8
9 bin_message = bytes.fromhex(message_hex)
10 encoded_msg = bin_message[32:]
11 decoded_msg = str(encoded_msg.decode('ascii'))
12
13 decrypted_msg = decrypt(decoded_msg, session_key)
14
15 print(decrypted_msg)
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS COMMENTS

Python + - [] [] ... ^ x

```
(3.13.1)
PSI2024Z_Z40/projekt main*
• /Users/pekoraptor/.pyenv/versions/3.13.1/bin/python /Users/pekoraptor/PSI2024Z_Z40/projekt/manual_decryption.py
?6
1kxKX4
wiadomosc
(3.13.1)
PSI2024Z_Z40/projekt main*
○ [ ]
```

Rysunek 5: Skrypt do manualnego deszyfrowania

7 Napotkane problemy

- Problem z asynchronicznym zakończeniem połączenia, rozwiązaliśmy go poprzez dodanie dodatkowych wątków zarówno po stronie serwera, jak i klienta. Wątki te odpowiadają za odbieranie informacji o zakończeniu komunikacji i bezpieczne zamknięcie połączenia.
- Problem z UI serwera, wyświetlające się logi z komunikacji wprowadzały chaos w interfejsie. Aby temu zapobiec, chcieliśmy aby wyświetlanie otrzymanych logów wykonywało się dopiero po przejściu w dedykowaną zakładkę. Rozwiązaliśmy ten problem dodając kolejkę logów.

8 Wnioski

- Udało nam się zaimplementować wszystkie wymagane funkcje, co pozwoliło na przeprowadzenie wygodnej w obsłudze symulacji bezpiecznej komunikacji między klientem a serwerem.
- Dzięki analizie pakietów Wireshark mogliśmy potwierdzić bezpieczeństwo naszej implementacji.
- Mechanizm Encrypt-then-MAC był prosty w implementacji, a jednocześnie okazał się skuteczny w zapewnieniu bezpieczeństwa przesyłanych wiadomości.
- Wykorzystanie konteneryzacji Docker znacząco uprościło proces wdrożenia i testowania aplikacji, eliminując problemy z zależnościami środowiskowymi między członkami zespołu.