PSI dokumentacja

Paweł Spirydowicz, Hanna Zarzycka, Krzysztof Wojtaszko
18.01.2026 wersja 1

Treść zadania

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

Założenia

- Architektura klient serwer.
- Serwer jest w stanie obsłużyć kilku klientów jednocześnie (proszę nie hardcodować liczby klientów – oczekuję parametru uruchomienia).
- Klient inicjuje połączenie z serwerem poprzez wysłanie wiadomości ClientHello (nieszyfrowana),na którą serwer odpowiada wiadomością ServerHello (nieszyfrowana).
- Sesja może zostać zakończona zarówno przez klienta jak i przez serwer poprzez wysłanie wiadomości EndSession. Po odebraniu EndSession należy od nowa wysłać ClientHello.
- Wszystko poza ClientHello i ServerHello jest szyfrowane.
- Potencjalny napastnik po przechwyceniu wiadomości nie jest w stanie z nich nic odczytać.

Wariant

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

Struktura wiadomości

Typ wiadomości

Pierwsze dwa bity każdej wiadomości będzie zawierał informację o jej typie, według legendy:

- 0b00 - ClientHello
- 0b01 - ServerHello
- 0b10 - EncryptedMessage
- 0b11 - EndSession

ClientHello

Wiadomość ClientHello niesie informację o typie wiadomości, parametrach p i g algorytmu Diffiego-Hellmana oraz klucz publiczny klienta. Ma rozmiar 10 bajtów kolejno przydzielonych na dane informacje:

2 bity - typ
14 bitów - g
4 bajty - p
4 bajty - klucz publiczny

ServerHello

Wiadomość ServerHello niesie informację o typie wiadomości oraz kluczu publicznym serwera. Ma rozmiar 5 bajtów, kolejno przydzielnoych:

2 bity - typ (+**6** bitów dopełnienia)
4 bajty - klucz publiczny

EncryptedMessage

Zaszyfrowana wiadomość poprzedzona jest informacją o długości zaszyfrowanej części wiadomości i wektorem inicjalizacyjnym (IV). W zaszyfrowanej części wiadomości znajduje się informacja o typie wiadomości i treść wiadomości. Za treścią zaszyfrowanej wiadomości znajduje się MAC (Message Authentication Code). Cała struktura może mieć od 51 do 65586 bajtów:

2 bajty - długość wiadomości (n)
16 bajtów - wektor inicjalizacyjny
1 bajt - typ
n bajtów - zaszyfrowana wiadomość
32 bajty - MAC

EndSession

Wiadomość EndSession zawiera jawną informację o długości wiadomości (n) = 1, jawny wektor inicjalizacyjny, zaszyfrowaną informację o typie wiadomości i MAC, łącznie 51 bajtów:

2 bajty - długość wiadomości (n)
16 bajtów - wektor inicjalizacyjny
1 bajt - typ
32 bajty - MAC

Wymiana kluczy

W celu wymiany kluczy korzystamy z algorytmu Diffiego-Hellmana.

Przykładowy scenariusz

Dla czytelności przykładu użyjemy znacznie niższych wartości niż będą stosowane w programie.

Klient wybiera liczbę pierwszą

$p = 23$,

podstawę g względnie pierwszą do p

$g = 5$,

oraz klucz prywatny

$a = 8$

i na ich podstawie oblicza swój klucz publiczny z wzoru:

$A = g^a = 5^8 \bmod 23 = 16$.

Klient wysyła wiadomość ClientHello:

0x0005000000170000010

Serwer odczytuje

typ - 0b00

g - 0b00000000000101 = 5

p - 0x00000017 = 23

A - 0x00000010 = 16

Serwer wybiera klucz prywatny

$b = 13$

i na jej podstawie oblicza klucz publiczny:

$B = g^b = 5^{13} \bmod 23 = 21$

Serwer wysyła wiadomość ServerHello:

0x4000000015

Klient odczytuje

Typ - 0b01

B - 0x00000015 = 21

Obie strony muszą obliczyć wspólny sekret:

Dla klienta

$s = B^a \bmod p = 21^8 \bmod 23 = 3$

Dla serwera

$s = A^b \bmod p = 16^{13} \bmod 23 = 3$

Uzyskany wspólny sekret można wykorzystać do wygenerowania klucza do szyfrowania wiadomości oraz do obliczania MAC.

Mechanizm integralności i autentyczności

Encrypt-then-MAC

Zdecydowaliśmy się na użycie mechanizmu integralności i autentyczności encrypt-then-MAC. Ta metoda polega na w pierwszej kolejności zaszyfrowaniu wiadomości kluczem szyfrującym, a następnie obliczeniu kodu uwierzytelniania wiadomości (MAC) na podstawie zaszyfrowanego tekstu. MAC stanowi rodzaj podpisuj pod wiadomością i jest w praktyce wyliczony przez funkcję haszującą lub szyfr blokowy z domieszanym kluczem tajnym. Zapewnia integralność, ponieważ każda próba zmiany treści zaszyfrowanej wiadomości skutkuje zmianą wartości MAC przez co odbiorca odrzuca wiadomość. Zapewnia autentyczność, ponieważ wymaga znajomości tajnego współdzielonego klucza, aby uzyskać ten sam MAC, jak i odszyfrować wiadomość.

W implementacji tej metody planujemy zastosować HMAC, czyli typ kodu uwierzytelniającego, wykorzystujący funkcję haszującą. Jako funkcję haszującą użyjemy SHA256, która pozwoli nam otrzymać MAC o długości 32 bajtów. Funkcje potrzebne do realizacji tego zadania weźmiemy z modułów hmac i hashlib Pythona.

Działanie

Struktura

Projekt zawiera plik docker-compose.yaml, w którym określony jest sposób budowania trzech kontenerów - serwera, klienta i sieci.

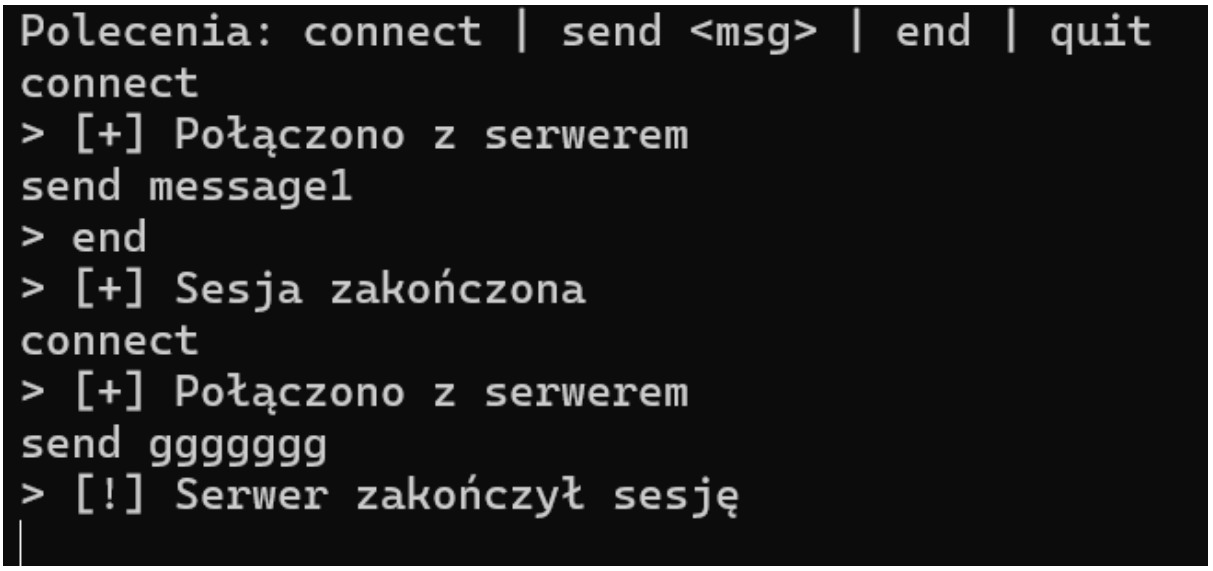
W folderze server znajduje się kod serwera w pythonie, plik dockerfile oraz endpoint.sh zawierający skrypt bashowy, który ma wykonać się w kontenerze. Poza odpaleniem programu uruchamia on tcpdump, które zapisuje ruch w sieci do pliku. Folder client zawiera kod klienta w pythonie oraz plik dockerfile.

Dostępny jest skrypt app4.sh, który pozwala na proste przetestowanie połączenia ze strony klienta, jednak dla większej interaktywności programu najlepiej otworzyć kilka terminali, aby móc również korzystać z komend serwera albo utworzyć wielu klientów.

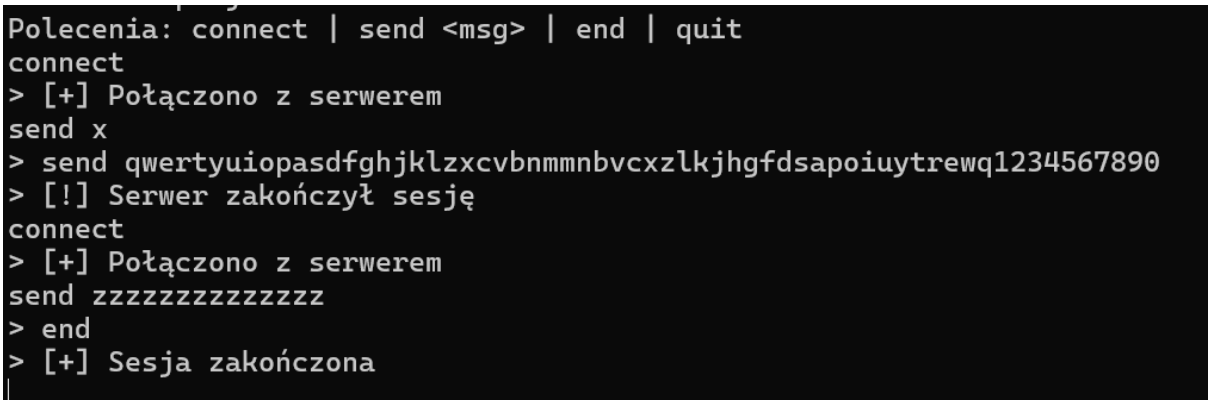
Klienci

Program klienta pozwala na wysłanie ClientHello poprzez komendę connect. Może wysłać dowolną wiadomość poprzez send, a także zakończyć połączenie z serwerem poprzez end. Na terminalu zostaje wyświetlona informacja, gdy serwer zakończy sesję.

Klient 1



Klient 2



Serwer

Serwer automatycznie odbiera wiadomości ClientHello i odsyła Server Hello. Wyświetla na terminalu informację o każdym kliencie, który próbuje się połączyć i każdej zakończonej sesji przez klienta. Z poziomu serwera są dostępne dwa polecenia: list - wypisanie listy klientów wraz z numerami przypisanymi im przez program oraz end [numer klienta] służący do zakończenia sesji z klientem.

```
psi-server  | tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
psi-server  | Serwer nasłuchuje na porcie 1234
psi-server  | >
psi-server  | Dostępne polecenia: list | end <client_id> | quit
psi-server  | > list
psi-server  | Brak podłączonych klientów
psi-server  | > [+] Klient ('172.23.0.4', 50414)
psi-server  | [1] message1
psi-server  | [+] Klient ('172.23.0.3', 39306)
psi-server  | [2] x
psi-server  | list
psi-server  | Klient 1 ('172.23.0.4', 50414)
psi-server  | Klient 2 ('172.23.0.3', 39306)
psi-server  | > [1] Sesja zakończona przez klienta
psi-server  | [-] Klient 1 rozłączony
psi-server  | [+] Klient ('172.23.0.4', 39200)
psi-server  | [3] ggggggg
psi-server  | [2] qwertyuiopasdfghjklzxcvbnmmnbvcxzlkjhgfdsapoiuytrewq1234567890
psi-server  | list
psi-server  | Klient 2 ('172.23.0.3', 39306)
psi-server  | Klient 3 ('172.23.0.4', 39200)
psi-server  | > end 2
psi-server  | [+] Zakończono sesję klienta 2
psi-server  | > [2] Błąd:
psi-server  | [-] Klient 2 rozłączony
psi-server  | [+] Klient ('172.23.0.3', 51160)
psi-server  | [4] zzzzzzzzzzzzzz
psi-server  | [4] Sesja zakończona przez klienta
psi-server  | [-] Klient 4 rozłączony
psi-server  | list
psi-server  | Klient 3 ('172.23.0.4', 39200)
psi-server  | > end 3
psi-server  | [+] Zakończono sesję klienta 3
psi-server  | > [3] Błąd:
psi-server  | [-] Klient 3 rozłączony
```

Przesył pakietów

W zaobserwowanym ruchu sieci możemy znaleźć przykłady przesyłu wiadomości ClientHello, ServerHello, EncryptedMessage i EndSession. Najłatwiej ich szukać po długości przesyłanych danych. ClientHello powinien mieć co najmniej 10 bajtów, ServerHello 5, EncryptedMessage, EndSession 51. Poniższe przykłady odpowiadają fragmentom wymiany informacji ze zrzutów ekranu powyżej.

4	0.000272	172.23.0.4	172.23.0.2	TCP	76 50414 → 1234 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=10 TSval=138934697…
6	0.001055	172.23.0.2	172.23.0.4	TCP	72 1234 → 50414 [PSH, ACK] Seq=1 Ack=11 Win=65152 Len=6 TSval=244929726…
8	28.493058	172.23.0.4	172.23.0.2	TCP	132 50414 → 1234 [PSH, ACK] Seq=11 Ack=7 Win=64256 Len=66 TSval=13893754…

172.23.0.4 - klient 1
172.23.0.2 - serwer

- 4 - ClientHello
- 6 - ServerHello
- 8 - Encrypted message

13	35.512969	172.23.0.3	172.23.0.2	TCP	76 39306 → 1234 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=10 TSval=322178688…
15	35.513562	172.23.0.2	172.23.0.3	TCP	72 1234 → 39306 [PSH, ACK] Seq=1 Ack=11 Win=65152 Len=6 TSval=8191348 T…
17	54.310226	172.23.0.3	172.23.0.2	TCP	132 39306 → 1234 [PSH, ACK] Seq=11 Ack=7 Win=64256 Len=66 TSval=32218056…
19	87.117736	172.23.0.4	172.23.0.2	TCP	132 50414 → 1234 [PSH, ACK] Seq=77 Ack=7 Win=64256 Len=66 TSval=13894340…

172.23.0.3 - klient 2
172.23.0.2 - serwer

- 13 - ClientHello

- 15 - ServerHello
- 17 - Encrypted message
- 19 - Encrypted message

```
43 158.401615    172.23.0.3          172.23.0.2          TCP        76 51160 → 1234 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=10 TSval=322190977…
45 158.402288    172.23.0.2          172.23.0.3          TCP        72 1234 → 51160 [PSH, ACK] Seq=1 Ack=11 Win=65152 Len=6 TSval=8314237 T…
47 173.621611    172.23.0.3          172.23.0.2          TCP       132 51160 → 1234 [PSH, ACK] Seq=11 Ack=7 Win=64256 Len=66 TSval=32219249…
49 189.086578    172.23.0.3          172.23.0.2          TCP       132 51160 → 1234 [PSH, ACK] Seq=77 Ack=7 Win=64256 Len=66 TSval=32219404…
54 200.184248    172.23.0.2          172.23.0.4          TCP       132 1234 → 39200 [PSH, ACK] Seq=7 Ack=77 Win=65152 Len=66 TSval=24494974…
```

172.23.0.4 - klient 1
172.23.0.3 - klient 2
172.23.0.2 - serwer

- 43 - ClientHello od klienta 2
- 45 - ServerHello do klienta 2
- 47 - EncryptedMessage od klienta 2
- 49 - EndSession od klienta 2
- 54 - EndSession od serwera do klienta 1

Szyfrowanie

Przykładowy pakiet z EncryptedMessage

```
da 9e 65 51 15 ff 96 b2  5a 3f b5 3a 08 00 45 00   ··eQ···· Z?·:··E·
00 76 85 22 40 00 40 06  5d 2b ac 17 00 04 ac 17   ·v·"@·@· ]+······
00 02 c4 ee 04 d2 58 f4  82 03 98 4e aa f3 80 18   ·····X·· ···N····
01 f6 58 9d 00 00 01 01  08 0a 52 d0 2f ea 91 fd   ··X····· ··R·/···
4f 74 00 20 14 ac 63 d6  15 ba 47 b5 86 f8 21 04   Ot·· ··c· ··G···!·
e3 da bf d2 be 38 02 ff  af 49 52 2b cf 98 51 f3   ·····8·· ·IR+··Q·
45 82 0e 07 d2 94 46 9e  cb 91 3e e9 42 5d 5a fc   E·····F· ··>·B]Z·
63 6c f4 3b 47 35 a8 b5  5c cc 56 03 4d 79 65 ae   cl·;G5·· \·V·Mye·
7c dc af 45                                        |··E
```

Przykładowy pakiet z EndSession

```
96 b2 5a 3f b5 3a da 9e  65 51 15 ff 08 00 45 00   ··Z?·:·· eQ····E·
00 76 dd 18 40 00 40 06  05 35 ac 17 00 02 ac 17   ·v··@·@· ·5······
00 04 04 d2 99 20 be c0  68 94 7c 75 a9 b3 80 18   ····· ·· h·|u····
01 fd 58 9d 00 00 01 01  08 0a 92 00 5d 6c 52 d1   ··X····· ····]lR·
3f 11 00 20 b3 3d 98 b8  aa cb 6b 16 7f ac ee e5   ?·· ·=·· ··k·····
d8 48 d5 ba 6a 93 6f 88  2e 34 05 ab 0f 9d e3 88   ·H··j·o· .4······
9e f7 2b 33 3b c1 2c a2  90 26 a5 1d 37 2b d1 fc   ··+3;·,· ·&··7+··
57 de 32 0f 72 ea 1e ab  ab b6 f4 de 97 3d 93 47   W·2·r··· ·····=·G
74 d0 6f 01                                        t·o·
```

Użyte algorytmy

Algorytm Diffiego-Hellmana

Algorytm Diffiego-Hellmana służy do uzgadniania kluczy szyfrujących w taki sposób, żeby nie dało się ich zdobyć poprzez podsłuchiwanie. Jest oparty na dużych liczbach pierwszych i problemie obliczania logarytmów dyskretnych, który jest nie do rozwiązania w sensownym czasie na współczesnych komputerach. Dokładne działanie algorytmu na przykładzie zawarliśmy wyżej w dokumentacji.

SHA256

Sha256 to funkcja skrótu obliczająca hash o długości 32-bitów. Hash jest zawsze taki sam dla identycznych danych oraz ma niewiele kolizji dla różnych danych, więc można go wykorzystać do weryfikacji integralności i autentyczności otrzymanych danych poprzez HMAC.

AES

AES to symetryczny i bardzo bezpieczny szyfr blokowy, którym szyfrujemy wiadomości. Dzięki wspólnemu sekretowi, który klient i serwer uzyskuje przy pomocy algorytmu Diffiego-Hellmana obie strony połączenia mogą zaszyfrować i odszyfrować wiadomość.

Napotkane problemy

Głównym problemem, który napotkaliśmy był pomiar ruchu w sieci dockerowej. Nie byliśmy w stanie go zarejestrować bezpośrednio Wiresharkiem, ponieważ nie był widoczny poza kontenerami. Rozwiązaliśmy to tak, że zainstalowaliśmy w kontenerze tcpdump, który mierzy ruch w tej sieci i zapisuje do pliku. Można go potem ściągnąć do lokalnego środowiska i otworzyć za pomocą Wireshark dla większej czytelności.

Przy włączaniu kontenera komendą *docker-compose up [container]* nie działa w pełni wczytywanie danych z konsoli. Nie był to problem przy kliencie, ale serwer nie mógł być otwarty komendą docker-compose run, ponieważ klient nie był w stanie go wykryć. Rozwiązaliśmy ten problem używając komendy *docker-compose attach psi-server* w oddzielnym terminalu, dzięki czemu można wprowadzać komendy dla serwera.

Wnioski

Naszymi głównymi celami w projekcie były: ustalenie prostych struktur wiadomości, szyfrowanie danych oraz zapewnienie integralności i autentyczności.

Każdemu rodzajowi wiadomości przypisaliśmy oznaczenie typu, które zajmuje tylko dwa pierwsze bity. ClientHello i ServerHello zgodnie z założeniem udało nam się przekazać

niewielką ilością bajtów. Pozostałe typy wiadomości są znacznie dłuższe ze względu na użyte metody zapewnienia integralności i autentyczności.

Jako algorytm szyfrowania wykorzystaliśmy AES - bardzo bezpieczny, ale szybki szyfr blokowy. AES jest algorytmem symetrycznym, więc wymaga ustalenia wspólnego klucza przez klienta i serwer. To zadanie wykonaliśmy poprzez algorytm Diffiego-Hellmana, Wymieniając potrzebne parametry w wiadomościach ClientHello i ServerHello.

Integralność i autentyczność zapewnia wykorzystanie mechanizmu encrypt-then-mac. Zapewnia on integralność, ponieważ próba manipulacji danymi sprawia, że MAC odbiorcy będzie inny niż zawarty w wiadomości. Autentyczność wynika z faktu, że wspólny klucz został ustalony w taki sposób, że jedynie klient i serwer znają jego wartość.