

PSI - Projekt, sprawozdanie finalne

16.01.2025

Zespół Z39 w składzie:

Adrian Murawski

Kacper Straszak

Michał Brzeziński

Cel projektu

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

Wybrany wariant funkcjonalny:

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

Struktura wiadomości

Typ wiadomości - struktura:

ClientHello - $\text{ClientHello}\{A, \{p\}, \{g\}$

ServerHello - $\text{ServerHello}\{B\}$

MessageData - $\{\text{HMAC}\}\{\text{IV}\}\text{MessageData}\{\text{treść wiadomości}\}$

EndSessionX - $\{\text{HMAC}\}\{\text{IV}\}\text{EndSessionX}$

** nazwy w nawiasach klamrowych oznaczają, że w ich miejsce wstawiana jest odpowiednia wartość*

*** liczby A, B, g, p są zgodne z opisem algorytmu Diffiego-Hellmana*

**** w wiadomości EndSessionX , X jest podmieniany na S lub C (w zależności która ze stron kończy połączenie)*

***** IV to wektor inicjalizacyjny dla algorytmu AES*

Wiadomości ClientHello i ServerHello są nieszyfrowane, więc nie jest przesyłany hash dla tych wiadomości.

Wykorzystane algorytmy

Do wymiany wspólnego sekretu użyto algorytmu Diffiego-Hellmana. Po wymianie wiadomości ClientHello i ServerHello, obie strony posiadają wspólny sekret s .

Do szyfrowania użyto algorytmu AES, a do utworzenia 32-bajowego klucza tego algorytmu używany jest hash SHA-256 wyliczony dla sekretu s .

Jako funkcja skrótu używany jest HMAC, który wykorzystuje funkcję skrótu SHA-256 oraz sekret *s*.

HMAC jest obliczany dla pozostałej części wiadomości, np. w przypadku typu wiadomości EndSessionX, dla IV oraz EndSessionX.

Implementacja

System działa poprawnie, klient poprawnie nawiązuje połączenie z serwerem (o ile serwer nie obsługuje już maksymalnej ilości klientów). Do ustanowienia połączenia wykorzystane są wiadomości ClientHello i ServerHello. Po tym procesie obie strony mają wyliczony sekret *s* oraz klucz AES. Następnie klient ma możliwość przesyłania zaszyfrowanych wiadomości do serwera lub też zakończenia połączenia. Na serwerze z kolei można wyświetlić podłączonych klientów, zakończyć połączenie z wybranym klientem lub zatrzymać serwer. Wiadomości przychodzące do serwera są odpowiednio odszyfrowywane i wyświetlane.

Dodatkowe informacje:

- uruchamiając serwer, należy podać host i port na którym będzie on uruchomiony
- uruchamiając klienta, należy podać host i port uruchomionego serwera
- można manipulować ilością wyświetlanych informacji poprzez ustawienie wyższego poziomu logowania (domyślnie w plikach ustawiony jest poziom *debug*, oznaczający największą ilość wyświetlanych informacji)
- maksymalna ilość klientów przekazywana jest jako parametr wywołania dla serwera

Uruchomienie

Zgodnie z wymaganiami, serwer oraz klienta można uruchomić w dockerze.

Najpierw należy wykonać polecenie **docker compose up -d**, aby utworzyć kontenery. Następnie można uruchomić serwer i klienta.

Uruchomienie instancji serwera:

```
docker exec -it z39_server python server.py <host> <port> <max_clients>
```

Uruchomienie instancji klienta:

```
docker exec -it z39_client python client.py <server_host> <server_port>
```

Przykładowo jako host do serwera można podać server, a jako port - 12345. Te same wartości należy podać przy uruchamianiu klienta.

Aby uzyskać wielu klientów, można uruchomić wiele instancji klienta w kontenerze client (w oddzielnych terminalach).

Z uwagi na to, że aplikacje wyświetlają dane i jednocześnie są interaktywne (czekają na polecenie użytkownika - np. wysłanie wiadomości, czy zakończenie połączenia), to czasem zapytanie użytkownika o opcję może zostać "zasypane" innymi informacjami, stąd może nie być jasne, jakie użytkownik ma opcje - wtedy wystarczy kliknąć przycisk enter, co spowoduje ponowne wyświetlenie dostępnych opcji.

Wyniki

Logi z przykładowego uruchomienia serwera są dostępne w katalogu `docs/example/logs`. W katalogu `docs/example/wireshark` umieszczono dane z programu *wireshark*, który został uruchomiony w momencie przesyłania wiadomości między klientem a serwerem.

Poniżej prezentowane są zrzuty ekranu z najważniejszych uchwyconych pakietów:

a) wiadomość ClientHello

No.	Time	Source	Destination	Protocol	Length	Info
98	3.279908332	127.0.0.1	127.0.0.1	TCP	74	33524 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=3907754523 TSecr=0 WS=128
99	3.279904824	127.0.0.1	127.0.0.1	TCP	74	12345 → 33524 [SYN, ACK] Seq=14 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3907754523 TSecr=0 WS=128
100	3.279909283	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=14 Ack=1 Win=65536 Len=0 TSval=3907754523 TSecr=3907754523
101	3.279905472	127.0.0.1	127.0.0.1	TCP	8	12345 → 33524 [PSH, ACK] Seq=18 Ack=14 Win=65536 Len=1 TSval=3907754523 TSecr=3907754523
102	3.279904802	127.0.0.1	127.0.0.1	TCP	66	12345 → 33524 [ACK] Seq=14 Ack=18 Win=65536 Len=0 TSval=3907754523 TSecr=3907754523
103	3.280371578	127.0.0.1	127.0.0.1	TCP	79	12345 → 33524 [PSH, ACK] Seq=14 Ack=18 Win=65536 Len=13 TSval=3907754524 TSecr=3907754523
104	3.280373778	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=18 Ack=14 Win=65536 Len=0 TSval=3907754524 TSecr=3907754524
745	25.063956784	127.0.0.1	127.0.0.1	TCP	162	33524 → 12345 [PSH, ACK] Seq=18 Ack=14 Win=65536 Len=96 TSval=3907777550 TSecr=3907754524
746	25.064061630	127.0.0.1	127.0.0.1	TCP	162	12345 → 33524 [PSH, ACK] Seq=14 Ack=114 Win=65536 Len=96 TSval=3907777552 TSecr=3907777550
747	25.064965839	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=114 Ack=110 Win=65536 Len=0 TSval=3907777552 TSecr=3907777552
12671	54.252765236	127.0.0.1	127.0.0.1	TCP	130	33524 → 12345 [PSH, ACK] Seq=114 Ack=110 Win=65536 Len=64 TSval=3907911768 TSecr=3907777552
12672	54.253272542	127.0.0.1	127.0.0.1	TCP	66	12345 → 33524 [FIN, ACK] Seq=110 Ack=178 Win=65536 Len=0 TSval=3907911769 TSecr=3907911768
12673	54.253377155	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [FIN, ACK] Seq=178 Ack=111 Win=65536 Len=0 TSval=3907911769 TSecr=3907911769
1272	54.253381673	127.0.0.1	127.0.0.1	TCP	66	12345 → 33524 [ACK] Seq=111 Ack=179 Win=65536 Len=0 TSval=3907911769 TSecr=3907911769

b) wiadomość ServerHello

No.	Time	Source	Destination	Protocol	Length	Info
98	3.279998032	127.0.0.1	127.0.0.1	TCP	74	33524 → 12345 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 TCP_Perm=1 TSecr=390775452 TSrc=0 MSS=128
99	3.2799984824	127.0.0.1	127.0.0.1	TCP	74	12345 → 33524 [SYN, ACK] Seq=18 Ack=1 Win=65483 Len=0 MSS=65495 TCP_Perm=1 TSecr=390775452 TSrc=390775452 MSS=128
100	3.2799989283	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSeq=3907754523 TSecr=3907754523
101	3.2799993472	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [PSH, ACK] Seq=18 Ack=1 Win=65536 Len=0 TSeq=3907754523 TSecr=3907754523
102	3.2799984802	127.0.0.1	127.0.0.1	TCP	66	12345 → 33524 [ACK] Seq=1 Ack=18 Win=65536 Len=0 TSeq=3907754523 TSecr=3907754523
103	3.280311576	127.0.0.1	127.0.0.1	TCP	70	12345 → 33524 [FIN, ACK] Seq=18 Ack=18 Win=65536 Len=0 TSeq=3907754524 TSecr=3907754523
104	3.2803137178	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=18 Ack=14 Win=65536 Len=0 TSeq=3907754524 TSecr=3907754524
145	25.0630586784	127.0.0.1	127.0.0.1	TCP	162	33524 → 12345 [PSH, ACK] Seq=18 Ack=14 Win=65536 Len=96 TSeq=390777550 TSecr=3907754524
146	25.064061693	127.0.0.1	127.0.0.1	TCP	162	12345 → 33524 [PSH, ACK] Seq=14 Ack=114 Win=65536 Len=96 TSeq=390777752 TSecr=390777550
147	25.064065839	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=114 Ack=11 Win=65536 Len=0 TSeq=390777752 TSecr=390777552
149	25.0640776523	127.0.0.1	127.0.0.1	TCP	120	33524 → 12345 [PSH, ACK] Seq=14 Ack=110 Win=65536 Len=96 TSeq=390771968 TSecr=390777552
1268	154.253277542	127.0.0.1	127.0.0.1	TCP	66	12345 → 33524 [FIN, ACK] Seq=118 Ack=178 Win=65536 Len=0 TSeq=3907719769 TSecr=39077168
1271	154.253377155	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=178 Ack=111 Win=65536 Len=0 TSeq=3907719769 TSecr=39077168

```

Timestamp value: 3907754524
Timestamp echo reply: 3907754523
  > [Timestamps]
  > [SEQ/ACK analysis]
TCP payload (13 bytes)
  > Data (13 bytes)
    Data: 53657276657248656c66f3134
0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 E
0010 60 41 ad 74 40 00 00 00 8f 40 7f 00 00 01 7f 00 A te @ @ ---
0020 00 01 39 82 f4 08 06 0e e3 3d 7b 08 03 80 18 00 - # (h)
0030 02 00 fe 35 00 00 01 01 08 0a e8 eb 9a 1c e8 eb . 5 ---
0040 9a 1b 53 05 72 76 65 72 48 65 6c 6c 6f 31 34 . Server Hall014

```

c) szyfrowana wiadomość

The image shows a Wireshark packet capture of a TCP connection. The top pane displays a list of packets, with packet 1272 selected. The middle pane shows the details of the selected packet, including the TCP header and the data payload. The data payload is marked as 'Data (96 bytes)' and is shown in hexadecimal and ASCII. The bottom pane shows the raw data in hexadecimal and ASCII. The data is encrypted, as indicated by the 'Data (96 bytes)' label and the non-readable ASCII output.

d) EndSessionC

The image shows a Wireshark packet capture of a TCP connection. The top pane displays a list of packets, with packet 1272 selected. The middle pane shows the details of the selected packet, including the TCP header and the data payload. The data payload is marked as 'Data (64 bytes)' and is shown in hexadecimal and ASCII. The bottom pane shows the raw data in hexadecimal and ASCII. The data is encrypted, as indicated by the 'Data (64 bytes)' label and the non-readable ASCII output.

Jak widać, wiadomości ClientHello i ServerHello nie są szyfrowane, stąd można bez problemu odczytać ich zawartość. Z kolei zawartości szyfrowanych wiadomości nie są możliwe do odczytania z poziomu wiresharka. Aby je odszyfrować wymagane jest posiadanie odpowiedniego klucza AES i wektora IV.

Za pomocą kodu dedykowanego w pliku manual_decode.py, po wstawieniu klucza AES i wektora IV (z logów) oraz wiadomości (jako ciąg heksadecymalny otrzymany z wiresharka) można odszyfrować wiadomość:

```
• → src git:(project) X p3 manual_decode.py
Odszyfrowana wiadomość: MessageDatazaszyfrowana wiadomość do serwera
```

Treść wiadomości jest zgodna z wpisaną przez klienta wiadomością, co widać także po logach:

```
12  Dostępne opcje:
13  1. Wyślij wiadomość
14  2. Zakończ połączenie
15  Wybierz opcję: 1
16  Message: zaszyfrowana wiadomość do serwera
17
```

Postępując analogicznie dla wektora IV i odczytanej następnej przesyłanej wiadomości z wiresharka można otrzymać:

```
• → src git:(project) X p3 manual decode.py
Odszyfrowana wiadomość: EndSessionC
```

Jest to wynik działania operacji nr 2, którą może wykonać klient, czyli zakończenia połączenia, co istotnie zostało wykonane przez klienta.

Wnioski

W ramach projektu zaprojektowano i zaimplementowano szyfrowany protokół mini TLS oparty na TCP. Po wymianie wiadomości ClientHello i ServerHello, obie strony wyliczają sekret, korzystając przy tym z algorytmu Diffiego-Hellmana. Ten sekret jest następnie używany do generowania klucza AES potrzebnego później do szyfrowania danych. Integralność i autentyczność wiadomości zapewnia HMAC z funkcją skrótu SHA-256.

Po ustanowieniu połączenia klient może wysyłać zaszyfrowane wiadomości do serwera, a serwer zarządza aktywnymi połączeniami, umożliwiając zakończenie sesji z klientem. Wiadomości ClientHello i ServerHello nie są szyfrowane. Zastosowanie mechanizmu encrypt-then-mac zapewnia integralność przesyłanych danych, a protokół wykorzystując wszystkie swoje parametry i funkcje działa prawidłowo, zabezpieczając transmisję przed atakami i przesyłając zgodne z wymogami wiadomości.

Bibliografia

https://pl.wikipedia.org/wiki/Protok%C3%B3%C5%82_Diffiego-Hellmana
<https://pypi.org/project/pycryptodome/>
<https://pl.wikipedia.org/wiki/SHA-2>
https://pl.wikipedia.org/wiki/Advanced_Encryption_Standard
<https://www.wireshark.org/>