

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

Zalecane jest utworzenie środowiska wirtualnego, w którym zostaną zainstalowane niezbędne pakiety. Można to zrobić (będąc w katalogu `src/`) przy pomocy polecenia *poetry shell* oraz *poetry install*.

Uwaga!

Wymagane jest, aby na komputerze był zainstalowany Python oraz poetry.

Będąc już w wirtualnym środowisku można uruchomić serwer oraz klienta w podany sposób:

Uruchomienie serwera:

```
python server.py <host> <port> <max_clients>
```

Uruchomienie klienta:

```
python client.py <server_host> <server_port>
```

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.

a) wiadomość ClientHello

```
((tcp.dstport == 12345 && tcp.srcport == 33524) || (tcp.
```

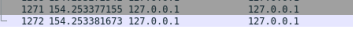
[illegible]

```
((tcp.dstport == 12345 && tcp.srcport == 33524) || (tcp.dst
```

```
(tcp.dstport == 12345 && tcp.srport == 33524) || (tcp.dstport == 33524 && tcp.srport == 12345))
```

No.	Time	Source	Destination	Protocol	Length	Info
98	3.279990832	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	3.279994824	127.0.0.1	127.0.0.1	TCP	74	12345 → 33524 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3907754523 TSecr=3907754523 WS=128
100	3.279999283	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3907754523 TSecr=3907754523
101	3.279993472	127.0.0.1	127.0.0.1	TCP	83	33524 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=17 TSval=3907754523 TSecr=3907754523
102	3.279994802	127.0.0.1	127.0.0.1	TCP	66	12345 → 33524 [ACK] Seq=1 Ack=18 Win=65536 Len=0 TSval=3907754523 TSecr=3907754523
103	3.28031570	127.0.0.1	127.0.0.1	TCP	70	12345 → 33524 [PSH, ACK] Seq=1 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	25.063056784	127.0.0.1	127.0.0.1	TCP	162	33524 → 12345 [PSH, ACK] Seq=18 Ack=14 Win=65536 Len=96 TSval=390777550 TSecr=3907754524
746	25.064061639	127.0.0.1	127.0.0.1	TCP	162	12345 → 33524 [PSH, ACK] Seq=14 Ack=114 Win=65536 Len=96 TSval=390777552 TSecr=390777550
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=390777552 TSecr=390777552
1267	154.252765236	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=390777552
1268	154.253272542	127.0.0.1	127.0.0.1	TCP	66	12345 → 33524 [FIN, ACK] Seq=118 Ack=178 Win=65536 Len=0 TSval=3907911769 TSecr=3907911768
1271	154.253977155	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	154.253381673	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

Timestamp value: 3907754524
Timestamp echo reply: 3907754523

- [Timestamp]
- [SEQ/ACK analysis]
- TCP payload (13 bytes)
- Data (13 bytes)


```
((tcp.dstport == 12345 && tcp.srcport == 33524) || (tcp.ds
```

[illegible]

d) EndSessionC

(((tcp.dstport == 12345 && tcp.srcport == 33524) (tcp.dstport == 33524 && tcp.srcport == 12345)))									
No.	Time	Source	Destination	Protocol	Length	Info			
98	3.279980332	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	3.279984824	127.0.0.1	127.0.0.1	TCP	74	12345 → 33524 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=3907754523 TSecr=3907754523 WS=128			
100	3.279989243	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3907754523 TSecr=3907754523			
101	3.279963472	127.0.0.1	127.0.0.1	TCP	83	33524 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=17 TSval=3907754523 TSecr=3907754523			
102	3.279964802	127.0.0.1	127.0.0.1	TCP	66	12345 → 33524 [ACK] Seq=1 Ack=18 Win=65536 Len=0 TSval=3907754523 TSecr=3907754523			
103	3.280371570	127.0.0.1	127.0.0.1	TCP	79	12345 → 33524 [PSH, ACK] Seq=1 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	25.063056784	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	25.064961639	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	25.064965039	127.0.0.1	127.0.0.1	TCP	66	33524 → 12345 [ACK] Seq=114 Ack=110 Win=65536 Len=0 TSval=3907777552 TSecr=390777552			
1267	154.252765236	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=390777552			
1268	154.253272542	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			
1271	154.253377155	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	154.253381673	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			

Timestamp value: 3907911768	
Timestamp echo reply: 390777552	
[Timestamps]	
[SEQ/ACK analysis]	
TCP payload (64 bytes)	
Data (64 bytes)	
Data: 70efaf737083096647a980fe5513f638bcbab571e16dd0e46ec280b9ced0ad2eeeeebc5...	
0000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 45 00
0010	00 74 0b 18 40 00 49 06 31 6a 7f 00 00 01 7f 00
0020	00 01 82 f4 30 39 3d 7b 68 63 d8 96 8e 90 80 18
0030	02 00 fe 68 00 00 01 01 08 0a e8 ee 00 58 e8 eb
0040	f4 10 70 67 07 57 00 30 92 64 7e 90 6f e5 53 37
0050	05 3c ba b5 71 e1 6d df 6e 46 c5 28 0b 9c 6d
0060	0a d2 ee ee c5 41 6e 43 16 8c 08 27 9c 6e 9b
0070	88 90 f2 bd e4 20 10 87 f3 d3 ff 93 7b 63 89
0080	28 1a

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 dostępnego w pliku `src/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 przysyłając 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/>