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 - ClientHello{A},{p},{g}
ServerHello - ServerHello{B}
MessageData - {HMAC}{IV}MessageData{treść wiadomości}
EndSessionX - {HMAC}{IV}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-bajtowego 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).

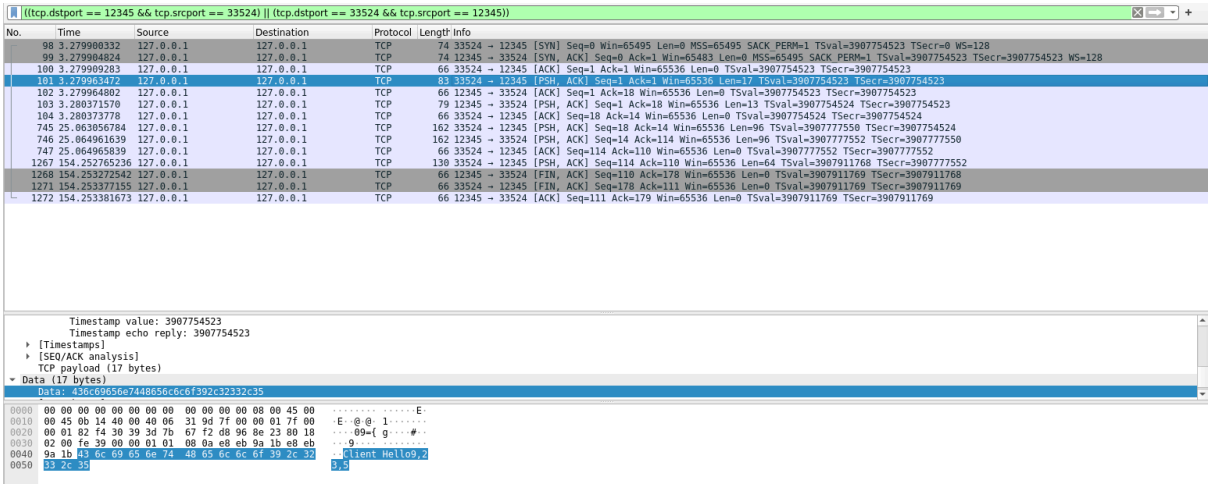
Z uwagi na to, że aplikacje wyświetlają dane i jednocześnie są interaktywne (czekają na polecenie użytkownika - np. wysłanie wiadomości, czy zakończenie połączenia), to czasem zapytanie użytkownika o opcję może zostać “zasypane” innymi informacjami, stąd może nie być jasne, jakie użytkownik ma opcje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

Wyniki

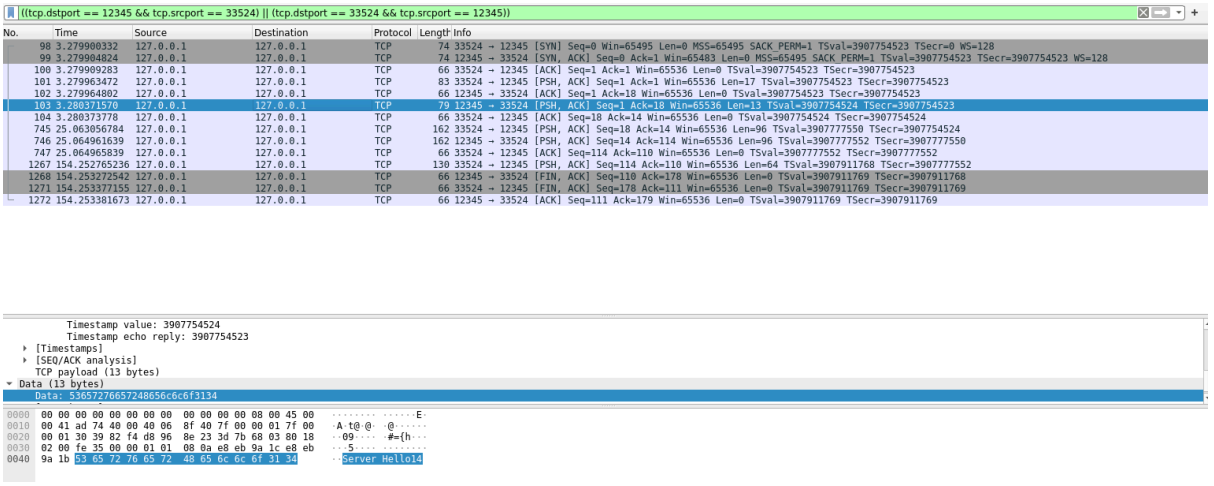
Logi z przykładowego uruchomienia serwera są dostępne w katalogu docs/example/logs. W katalogu docs/example/wireshark umieszczono dane z programu *wireshark*, który został uruchomiony w momencie przesyłania wiadomości między klientem a serwerem.

Poniżej prezentowane są zrzuty ekranu z najważniejszych uchwyconych pakietów:

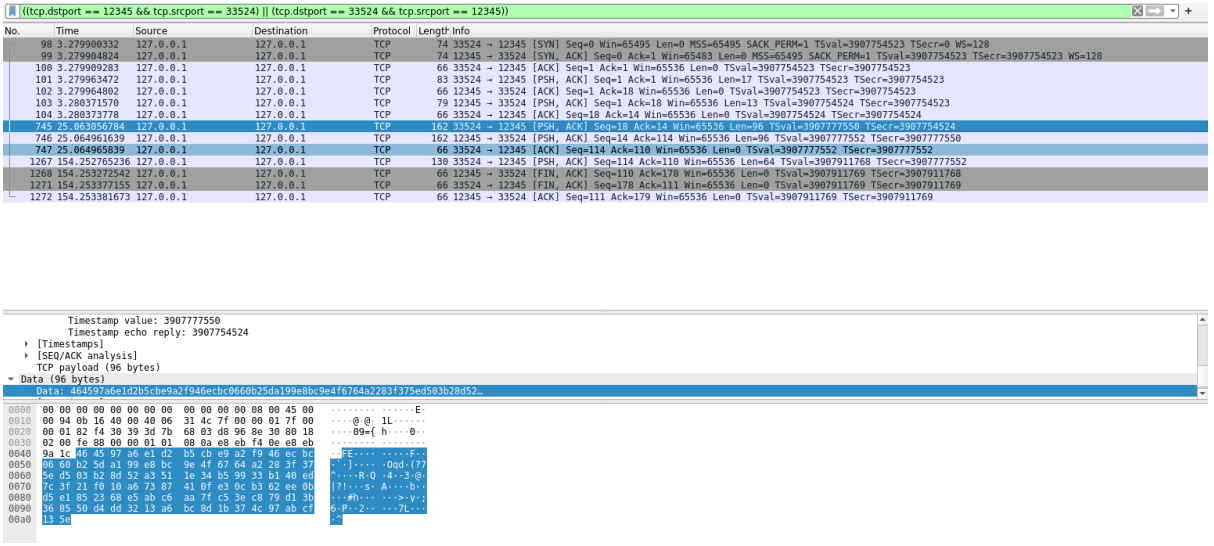
a) wiadomość ClientHello



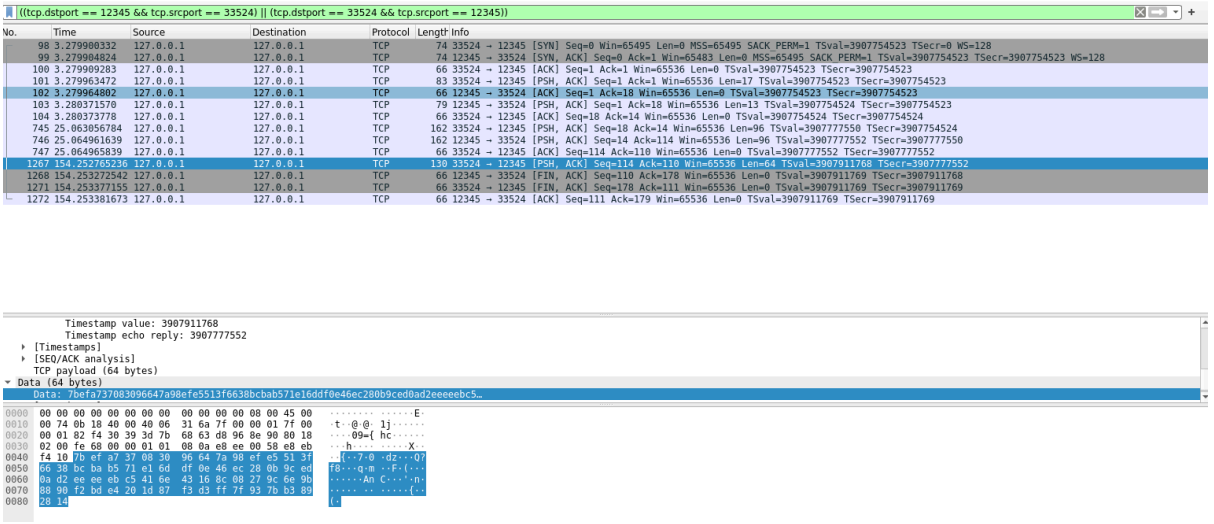
b) wiadomość ServerHello



c) szyfrowana wiadomość



d) EndSessionC



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 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:

```
Dostępne opcje:
1. Wyślij wiadomość
2. Zakończ połączenie
Wybierz opcję: 1
Message: zaszyfrowana wiadomość do serwera
```

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/