Sprawozdanie końcowe – projekt PSI

Projekt: Programowanie Sieciowe
Autorzy: Marek Dzięcioł, Jakub Mieczkowski, Tomasz Nejman
Data sporządzenia: 18.01.2026
Wersja: 1.0

Treść polecenia:

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

Struktura plików i sposób uruchomienia

Pliki, w których znajduje się implementacja tego projektu to:

- server/tcp_server.py
- client/tcp_client.py
- proj_lib.py – tutaj zostały wydzielone funkcje, które są uniwersalne dla obu stron, korzystających z tego samego protokołu (naszego)

Oprócz tego, dwa pliki Dockerfile oraz plik run.sh.

Plik run.sh – służy on do uruchomienia programu. Automatycznie uruchamia on 1 kontener zawierający serwer i *n* kontenerów, każdy z jednym klientem.

- pierwszy argument wywołania to maksymalna liczba połączeń serwera
- drugi argument wywołania to liczba kontenerów klienckich.

run.sh automatycznie wykrywa czy jest odpalony na natywnym Linuxie czy na Windowsie z zainstalowanym WSL. Plik ten przyczepia w trybie interaktywnym terminal kontenera zawierającego serwer i uruchamia *n* terminali, po jednym dla każdego klienta, gdzie są one doczepiane w trybie interaktywnego terminala.

Sposób działania i użycia programu

Połączenie inicjowane jest automatycznie przez każdego klienta i rozpoczyna się interaktywna sesja. Zarówno klient, jak i serwer mogą wysyłać wiadomości do siebie nawzajem. Jeżeli któraś ze stron wyśle wiadomość z zawartością “ENDSSION” (nasza sygnatura zakończenia sesji), która jest nierozróżnialna od zwykłej przed rozszyfrowaniem, sesja jest kończona dla danego klienta, a serwer usuwa go z listy połączeń. Serwer ma dostępne komendy (w interaktywnym programie):

- list (wyświetla obecne połączenia)
- exit (wysyła “ENDSSION” i kończy połączenie z każdym z klientów)
- help (wyświetla legalne polecenia)

- <ID> <zawartość> – to struktura polecenia, służącego do wysyłania wiadomości do klienta. ID to indeks połączenia z listy drukowanej przez “list”, a zawartość to treść wiadomości. Nie trzeba dodawać cudzysłowów, a spacje są legalne w treści.

Klient może wpisać tekst wiadomości, która jest wysyłana. Po wpisaniu sygnatury końca sesji “ENDSSION”, w terminalu klienta wystarczy nacisnąć “Enter”, aby go zamknąć.

Analogicznie dzieje się w sytuacji, gdy to serwer zakończy połączenie: należy zatwierdzić pustą wiadomość (nie zostanie ona wysłana - połączenie już nie istnieje) i nacisnąć “Enter”, aby zamknąć terminal kliencki.

Opis użytych algorytmów

W ramach projektu zaimplementowano prosty protokół szyfrowanej komunikacji po TCP, wzorowany na koncepcji TLS (tzw. mini TLS). Po zestawieniu połączenia tcp klient-serwer wykonywane jest bezpieczne losowanie kluczy i wymiana klucza sesyjnego, a następnie wszystkie komunikaty aplikacyjne przesyłane są w postaci zaszyfrowanej(zarówno nagłówek jak i ładunek). W projekcie wykorzystano następujące mechanizmy kryptograficzne: wymianę kluczy Diffiego–Hellmana, szyfrowanie strumieniowe OTP oraz uwierzytelnianie i kontrolę integralności poprzez HMAC-SHA224 w trybie mac-then-encrypt.

Wymiana kluczy – Diffie–Hellman (DH)

W celu uzgodnienia wspólnego klucza sesji klient i serwer stosują algorytm wymiany kluczy Diffie–Hellman. Mechanizm ten umożliwia uzyskanie tego samego klucza po obu stronach komunikacji bez konieczności przesyłania klucza prywatnego przez sieć. Czyli nawet w przypadku podsłuchu komunikacji, atakujący nie będzie w stanie odtworzyć klucza sesji w rozsądnym czasie.

Wymiana realizowana jest poprzez wiadomości “ClientHello” zawierającą: p, g (parametry grupy) oraz liczbę A; “ServerHello” zawierającą liczbę B

Dokładne działanie przedstawia się następująco:

1. Klient wysyła parametry p, g oraz swój klucz publiczny A = g^a mod p.
2. Serwer odsyła swój klucz publiczny B = g^b mod p.
3. Obie strony wyliczają wspólny sekret: klient: K = B^a mod p i serwer: K = A^b mod p
4. Wartość K stanowi klucz sesji i jest wykorzystywana w kolejnych etapach (szyfrowanie OTP + HMAC).

Szyfrowanie wiadomości – OTP (One-Time Pad)

Do szyfrowania danych zastosowano algorytm OTP czyli szyfrowanie strumieniowe oparte o operację XOR. W praktyce, zamiast prawdziwie losowego jednorazowego klucza (co byłoby niepraktyczne), strumień szyfrujący (keystream) jest deterministycznie generowany na podstawie klucza sesji K i kierunku wymiany informacji(c2s, s2c).

Proces szyfrowania przebiega następująco:

1. generowany jest strumień bajtów keystream o długości równej długości szyfrowanego bloku,
2. następnie wykonywana jest operacja bitowa XOR: ciphertext[i] = plaintext[i] XOR keystream[i]
3. odszyfrowanie po przeciwnej stronie jest identyczne: plaintext[i] = ciphertext[i] XOR keystream[i]

Zaletą tego podejścia jest prostota implementacji oraz łatwość ręcznego sprawdzenia poprawności działania.

Integralność i autentyczność – HMAC-SHA224

Aby zapewnić ochronę przed modyfikacją danych oraz podstawową integralność po stronie odbiorcy, zastosowano mechanizm HMAC-SHA224. HMAC (Hash-based Message Authentication Code) to kod uwierzytelniający wyliczany na podstawie klucza sesji K i danych jawnych, które mają być chronione

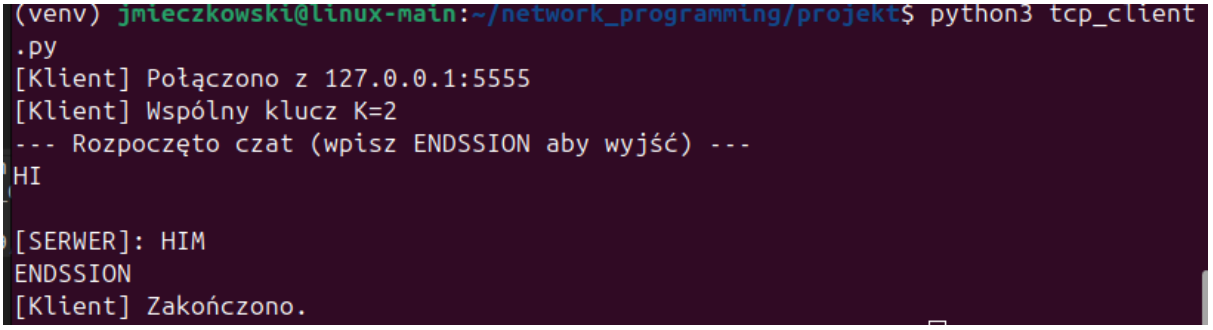
W projekcie przyjęto długość równą długości funkcji skrótu SHA-224 czyli 28 bajtów.

HMAC obliczany jest dla danych: typ wiadomości, rozmiar wiadomości i zawartość wiadomości.

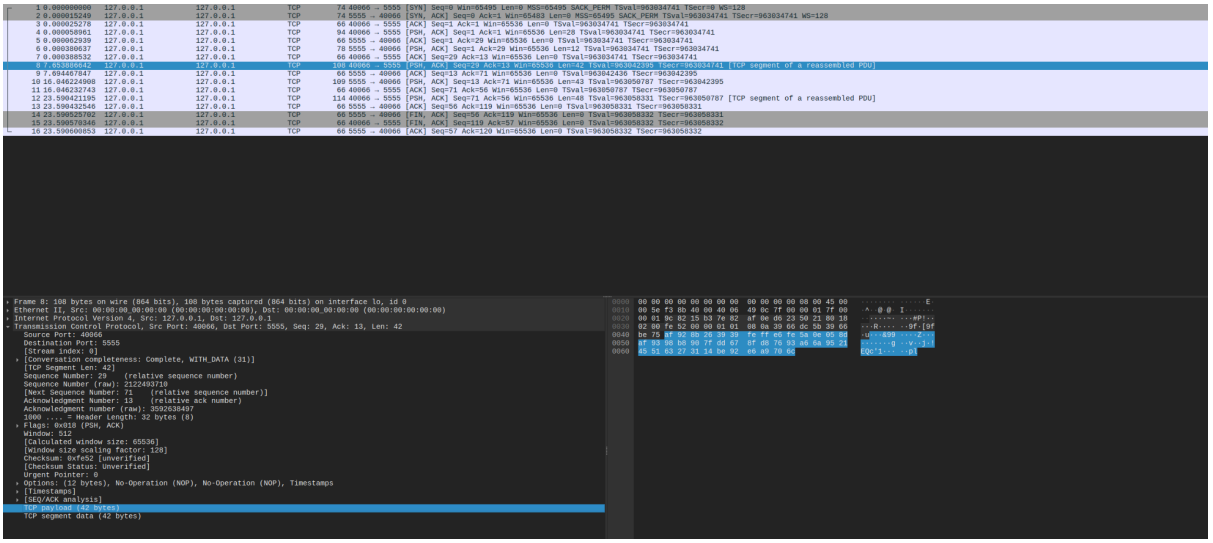
Algorytm ten pozwala wykryć:

- zmianę treści wiadomości,
- podmianę typu wiadomości,
- manipulację deklarowanym rozmiarem.

Dowód, że protokół działa



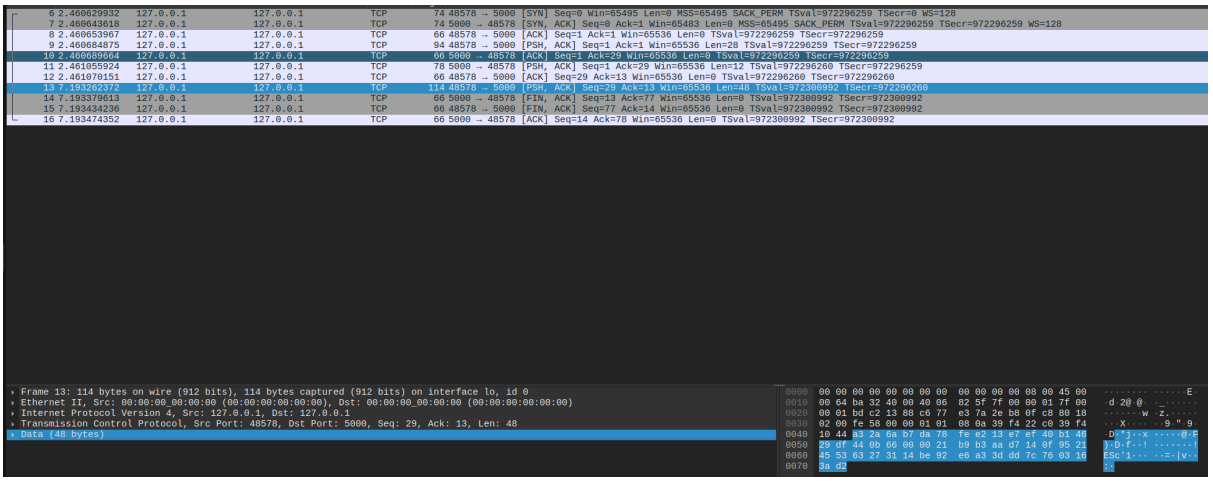
Przechwycono w wireshark:



Wiadomość ma 42 bajty, z czego nagłówek zajmuje 40, więc 2 ostatnie to wiadomość: 70 6c.
Zapisany do pliku keystream wyliczony automatycznie z klucza: 38 25
Po wykonaniu operacji xor: 70^38=48 6c^25=49
Z tablicy ASCII: 48=H, 49=I
Wiadomość: HI

ENDSSION jest nierozróżnialne od innych wiadomości

Payload ENDSSION jest nierozróżnialny względem zwykłej komendy, ponieważ różni się jedynie typem w nagłówku i zawartością. Jak widać na załączonym zrzucie ekranu dalej następuje tylko zamknięcie sesji



Napotkane problemy

Podczas implementacji i testowania napotkaliśmy kilka problemów technicznych związanych zarówno z kryptografią, jak i komunikacją sieciową i obsługą procesów.

- Pierwszym problemem było generowanie wartości losowych. Początkowo zastosowany mechanizm losowania był niewystarczająco losowy, przez co mogły występować powtarzalne klucze sesyjne. Ostatecznie zastosowano bezpieczniejszą metodę losowania dostępną w Pythonie, aby uniknąć przewidywalności kluczy.
- Kolejnym problemem była kwestia doboru rozmiarów liczb. Ponieważ w protokole DH przesyłane są liczby p, g, A, B, konieczne było przyjęcie konkretnego formatu (int64) i konsekwentne stosowanie go po obu stronach. Jednakże, takie rozmiary liczb są niewystarczające dla faktycznych zastosowań kryptograficznych.
- Kolejnym problemem jest użycie w kolejnych wiadomościach tego samego ziarna generującego losowe bajty. Z tego powodu może nastąpić sytuacja w której poznanie treści pozwoli uzyskać klucz K. Wynika to ze sposobu działania funkcji xor(a xor b = c, c xor a = b). Można to rozwiązać np. poprzez dodanie wektora inicjalizacji.
- Podczas uruchamiania kilku klientów w Dockerze (jeden klient = jeden kontener) pojawił się problem polegający na tym, że po zakończeniu sesji kontener klienta pozostaje uruchomiony. Użytkownik musi go ręcznie zakończyć.
- W trakcie implementacji kończenia sesji pojawiła się sytuacja, w której jedna ze stron kończyła połączenie wysyłając ENDSSION, ale druga strona nadal oczekiwała potwierdzenie typu OK. Skutkowało to zakleszczeniem: proces pozostawał w stanie oczekiwania, mimo że połączenie logicznie powinno zostać zamknięte.
- Podczas testów serwera często przerywaliśmy program za pomocą Ctrl+C. Powodowało to pozostawienie portu w stanie TIME_WAIT, przez co przy szybkim ponownym uruchomieniu serwera pojawiał się błąd: Errno 98: Address already in use
- Ostatnim problemem, który utrudniał testowanie, było buforowanie wyjścia. Podczas pracy w kontenerach oraz przy podpiętym terminalu niektóre komunikaty debugujące lub informacyjne nie pojawiały się od razu, tylko dopiero po czasie lub po zakończeniu programu. Rozwiązaniem było wymuszenie opróżniania bufora (flush).

Wnioski

W ramach projektu udało się zaprojektować oraz zaimplementować działający szyfrowany protokół komunikacyjny (mini TLS) oparty na TCP. Protokół jest stosunkowo bezpieczny, pomijając użyte zbyt małe wartości liczb pierwszych i sekretów oraz brak wektora inicjalizacji.
Poprawność działania implementacji została potwierdzona testami praktycznymi, w tym ręczną analizą przechwyconych pakietów w programie Wireshark. Umożliwiło to weryfikację, że przesyłane dane są rzeczywiście zaszyfrowane.
Oczywiście, naszemu protokołowi brakuje bardzo dużo do pełnoprawnego protokołu TLS (np. certyfikatów, uwierzytelnienia stron, ochrony przed różnymi typami ataków, mechanizmów renegocjacji).
Podsumowując, nasz edukacyjny projekt spełnił postawione przed nim wymagania. Opracowano działający protokół szyfrowanej komunikacji TCP, zaimplementowano klienta i serwera, oraz potwierdzono działanie.

Realizacja projektu pozwoliła nam lepiej zrozumieć praktyczne aspekty tworzenia protokołów sieciowych takie jak kryptografia, synchronizacja czy testowanie.