

Bezpieczeństwo usług sieciowych

— laboratorium 1 —

komunikator JSON + Diffie Hellman

Adrian Frydmański

6 listopada 2017

1 Wstęp

Celem laboratorium było stworzenie komunikatora (serwera i klienta) przesyłającego wiadomości w zadeklarowanych JSONach i ustalającego klucz szyfrowania przez algorytm Diffiego Hellmana.

2 Opis projektu

Zarówno klient jak i serwer napisane zostały przy użyciu języka Python 2.7.

2.1 Sposób wykonania zadania

Projekt składa się z dwóch skryptów: `server.py` i `client.py` oraz dodatkowego pliku `message.py` zawierającego kod współdzielony przez oba skrypty.

Wiadomości użyte w komunikacji wyglądają następująco:

Funkcja	Treść wiadomości	Kierunek
Żądanie wymiany kluczy	{ "request": "keys" }	↑
Odpowiedź na żądanie	{ "p": 123, "g": 123 }	↓
Wymiana obliczonych a i b	{ "a": 123 }	↑
	{ "b": 123 }	↓
Ustawienie szyfrowania	{ "encryption": "none" }	↑
Wymiana wiadomości	{ "msg": "...", "from": "name" }	↕

Rysunek 1: Używane wiadomości

2.1.1 Serwer

Serwer należy uruchomić podając w argumencie numer portu (na przykład `./server.py 8000`).

Serwer uruchamia kilka wyspecjalizowanych wątków:

- wątek główny — odpowiada za nawiązanie połączenia. W przypadku połączenia przychodzącego tworzy wątek do obsługi go. Ponadto odpowiada za start wątku logera.
- logger — służy do wyświetlania zdarzeń, rozwiązuje problem pisania na stdout z wielu wątków. Uruchamiany jeden na instancję serwera.
- wątek kliencki — obsługuje połączenia przychodzące od klienta, uruchamiany dla każdego oddzielnie. Odpowiada za wymianę kluczy i ustawienie trybu szyfrowania. Uruchamia jeden „subwątek” wysyłający wiadomości przychodzące od innych klientów.
- „subwątek” kliencki — zawiera kolejkę wiadomości do wysłania podłączonemu klientowi. Jeden na klienta. Odczytuje skolejkowane wiadomości do wysłania i umieszcza w gniazdku po uprzednim zaszyfrowaniu.

2.1.2 Klient

Klienta należy uruchomić podając w argumentach adres ip serwera i numer portu (na przykład `./client.py localhost 8000`). Ponadto można określić początkową metodę szyfrowania w kolejnych argumentach (`none`, `cezar` lub `xor`) oraz uruchomić w trybie testowania, dodając argument `bot`.

Klient na początku próbuje nawiązać połączenie z serwerem. Wysła żądanie o liczby `p` i `g` z algorytmu Diffiego Hellmana, wylicza `A` czeka na wyliczoną przez serwer liczbę `B` i na ich podstawie oblicza klucz sesji.

Następnie wysyłana jest informacja o szyfrowaniu. Domyślne szyfrowanie to `none`, czyli jego brak — wiadomości są widoczne dla wszystkich.

Klient uruchamia wątek wyświetlający przychodzące wiadomości. Odpowiada on za deszyfrację ich i działa niezależnie od wprowadzanego tekstu.

W pętli głównej odczytywane są wiadomości wpisane na standardowe wejście. W trybie testów wysyłana jest wiadomość testowa z kolejnym numerem co stały okres wynoszący 2 sekundy.

Możliwa jest zmiana metody szyfrowania. Wystarczy wpisać w kliencie tę metodę poprzedzoną `@@@` (na przykład `@@@xor`, by zmienić szyfrowanie na `xor`). Wiadomości w tej postaci nie są wysyłane i są traktowane jako wiadomości sterujące.

2.1.3 Szyfrowanie i wiadomości

Szablony wiadomości, metody szyfrujące i deszyfrujące zostały umieszczone we współdzielonym pliku `message.py`.

Szyfrowanie i deszyfrowanie wygląda następująco:

- XOR — wykonanie operacji xor na każdym bajcie tekstu z najmłodszym bajtem klucza — zarówno przy szyfrowaniu jak i deszyfrowaniu;

- Szyfr Cezara — utworzenie słownika z przesunięciem o wartość klucza modulo połowa długości słownika (wynika to z faktu, że osobno przesuwane są małe i wielkie litery) i zamiana znaków zgodnie ze słownikiem. W przypadku deszyfrowania klucz ma wartość ujemną i operacja modulo daje jego dopełnienie do liczby będącej rozmiarem połowy słownika (liczby małych bądź wielkich liter).

2.2 Użyte rozwiązania

Wiadomości przesyłane między klientem, a serwerem są kodowane przy użyciu base64, by wpisany tekst nie naruszył struktury JSONa.

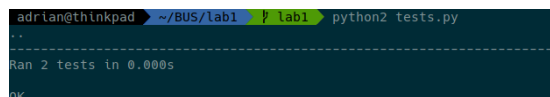
Liczby `p` i `g` są generowane dzięki funkcji `Crypto.Util.number.getPrime()`.

3 Testy

Poniżej opisano wykonane testy mające na celu sprawdzenie poprawności działania skryptów.

3.1 Testy jednostkowe

Przeprowadzono testy jednostkowe z użyciem wbudowanego w Pythona `unittest`. Uruchamianie testów jednostkowych odbywa się poprzez uruchomienie skryptu `tests.py`. Testy jednostkowe sprawdzają poprawność szyfrowania i deszyfrowania wiadomości.

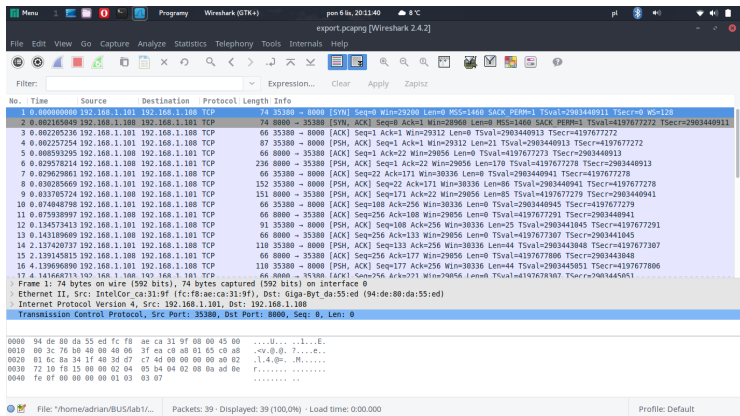


```
adriang@thinkpad ~ /BUS/lab1$ python2 tests.py
..
Ran 2 tests in 0.000s
OK
```

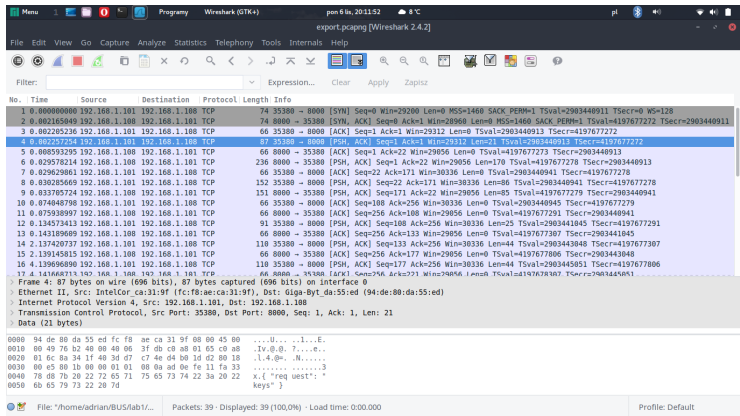
Rysunek 2: Uruchamianie testów jednostkowych.

3.2 Przesyłanie i odbieranie wiadomości

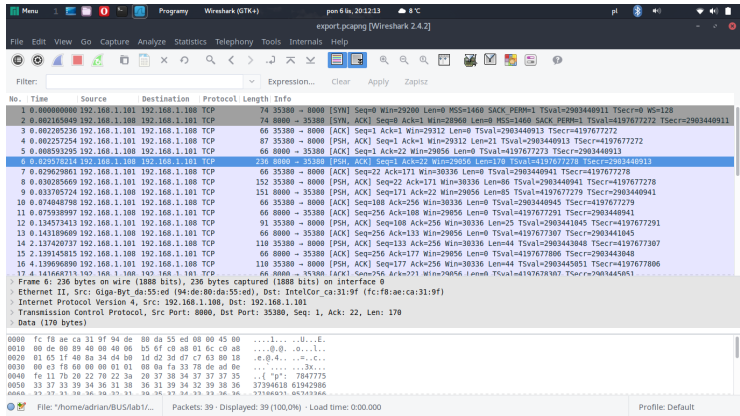
Podstawowym testem jest sprawdzenie poprawności wysyłania i odbierania wiadomości przez klienta i serwer. Test został wbudowany jako funkcjonalność klienta. Należy uruchomić go z argumentem `bot`. W wyniku tego klient zacznie wysyłać wiadomości z kolejnymi numerami do serwera. Serwer odbierze je i roześle pozostałym podłączonym klientom.



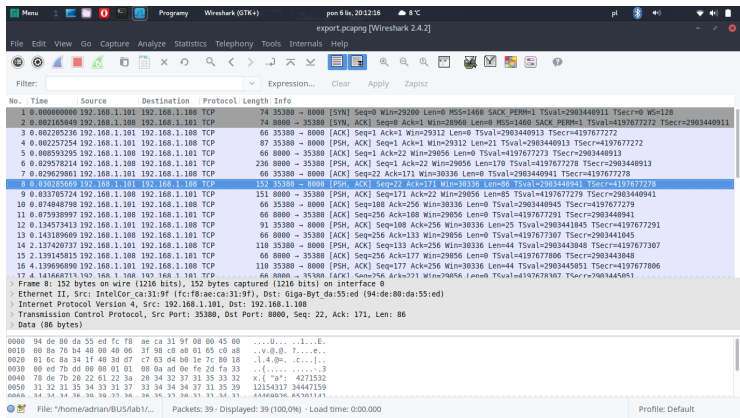
Rysunek 5: Wireshark — handshake



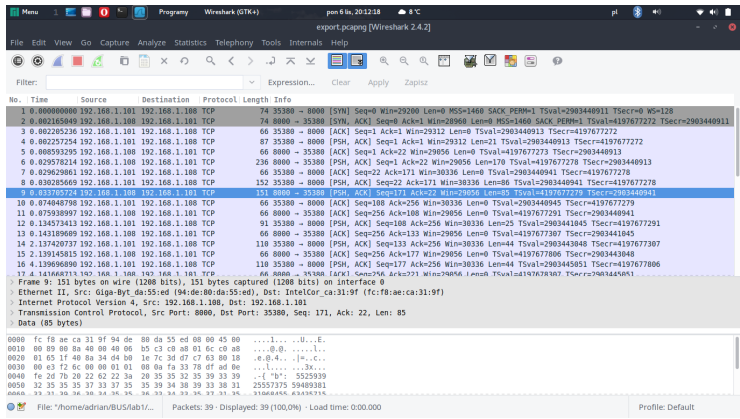
Rysunek 6: Wireshark — żądanie wymiany kluczy



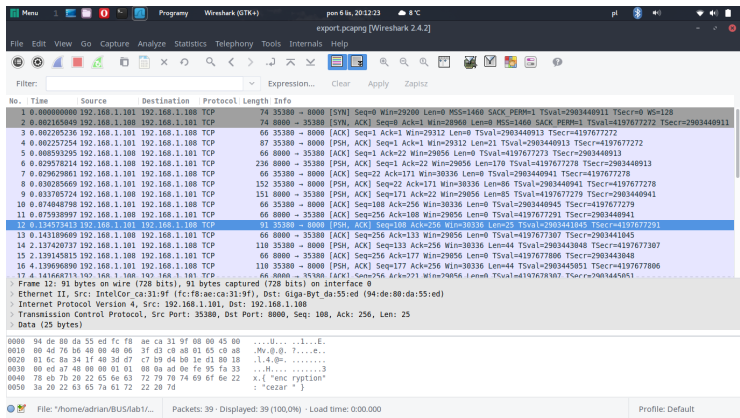
Rysunek 7: Wireshark — liczby p i q od serwera



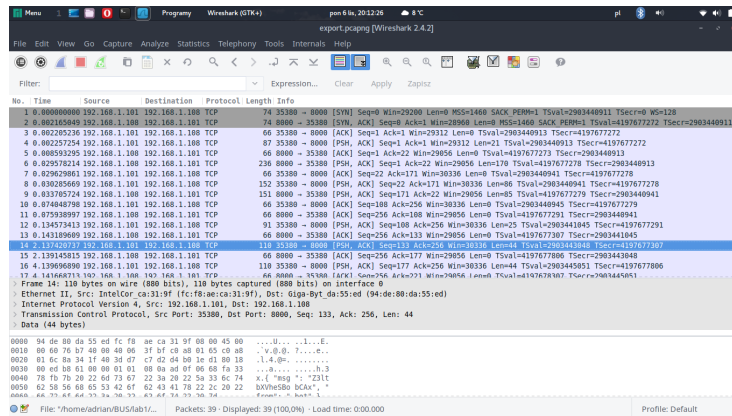
Rysunek 8: Wireshark — wysłanie liczby A do serwera



Rysunek 9: Wireshark — odebranie liczby B z serwera



Rysunek 10: Wireshark — wysłanie informacji o szyfrowaniu



Rysunek 11: Wireshark — wysłanie wiadomości

Można zauważyć, że przesyłana wiadomość jest zakodowana, a po zdekodowaniu jej z base64 nadal trzeba ją odszyfrować zgodnie z przyjętym szyfrowaniem.

4 Wnioski

Prosty komunikator uruchamiany w terminalu mógłby zostać rozbudowany o graficzne międzymordzie użytkownika. W prosty sposób rozwiązałoby to problem usuwania wpisywanej wiadomości podczas otrzymywania innej z serwera i wyświetlania jej na ekranie. W konsoli wymagałoby to użycia dodatkowych bibliotek (jak Curses).