Bezpieczeństwo systemów i usług informatycznych.
Budynek C-1, sala wirtualna wtorek 13:45
Prowadzący: Mgr inż. Przemysław Świerszcz

Sprawozdanie z laboratoriów nr 1

Mateusz Hyzicki, nr indeksu: 225947

Cel laboratoriów

Stworzyć komunikator w dowolnym języku i technologii do obsługi komunikacji klient-serwer. Komunikator ma wspierać protokół Diffiego-Hellmana oraz różne metody szyfrowania(brak szyfrowania, xor, cezar)

Protokół Diffiego-Hellmana - jest to uzgodniony schemat wymiany informacji przed przystąpieniem do faktycznej komunikacji, mający na celu ustalenie wspólnej, niepublicznej liczby będącej podstawą do wykorzystanych metod szyfrowania. Liczba ta jest obliczana po obu stronach na podstawie wymienionych danych dzięki czemu nie jest faktycznie nigdy udostępniana w sieci co utrudnić jej przechwycenie.

Szyfr cezara - sposób szyfrowania polegający na zastąpieniu każdego znaku innym, oddalonym o określoną odległość(tu: sekret obliczony w protokole Diffiego-Hellmana) w systemie ASCII

Szyfrowanie metodą XOR - sposób szyfrowania polegający na wykonaniu odwracalnej operacji XOR na każdym znaku wiadomości z określoną wartością(tu: sekret obliczony w protokole Diffiego-Hellmana)

Specyfikacja

Wybrany język: python3 - ze względu na własną znajomość języka, łatwość jego testowania oraz jego szerokie zastosowanie w poruszaniu się w systemach operacyjnych i korzystaniu z ich funkcji. Język ten świetnie nadaje się do prostych małych aplikacji, zachowując tym samym potencjał skalowalności
Wersja python 3.6

Wybrana biblioteka do obsługi kieszeni: socket - standardowa biblioteka pythonowa do obsługi socket'ów dostępna na większości obecnie rozpowszechnionych systemów(w tym Linux, Windows, MacOS)
Spis dodatkowych bibliotek zawarty w pliku requirements.txt.

Opis działania

Serwer - skrypt server.py uruchamia obsługę danej kieszeni.
Uruchamiany jest główny wątek serwera *run*. Nasłuchuje on nowych połączeń i dla każdego z nich tworzy osobny obiekt klasy typu object *Client*, składający się z połączenia, typu szyfrowania oraz wątku obsługującego go przez metodę *handle_client* - tam wykonywane są czynności protokołu Diffiego-Hellmana. Po przejściu wszystkich kroków wątek wchodzi w pętlę odbierającą i rozszyfrowywującą wiadomości przychodzące przez dane połączenie.
Po wystartowania głównego wątku serwer przechodzi w tryb interaktywny, z którego użytkownik może go zamknąć bądź napisać wiadomość do określonego przez identyfikator klienta według schematu:
${client_id}:${message_data}

Klient - skrypt client.py uruchamia obsługę kieszeni od strony klienta poprzez łączenie się z wystawionym na danej kieszeni serwerem.
Metoda *start_chat* rozpoczyna wykonywanie czynności sprecyzowane przez protokół Diffiego-Hellmana. Po ich sukcesywnym zakończeniu uruchamiany jest wątek z metodą *recv_data*, który odczytuje i rozszyfrowuje wiadomości przychodzące.
Następnie uruchamiana jest interaktywna pętla, podczas której użytkownik może pisać wiadomości, które wysyłane będą do serwera.

Przebieg implementacji

Implementację rozpoczęto od stworzenia klasy zawierającą podstawowe metody obsługi socketów. Miała ona służyć zarówno dla serwera jak i klienta. Szybko jednak okazało się, że warto rozdzielić te dwie funkcjonalności, w samej klasie macierzystej zostawiając tylko wspólne metody do wysyłania, odbierania wiadomości, parametry oraz importowane moduły. Po ukończeniu szkieletu tych modułów możliwe było wzajemne połączenie i wysłanie pojedynczej wiadomości.
Następnie dodany został protokół i przy okazji został poprawiony sposób przesyłania wiadomości by mogło się to dziać w dwie strony. Wciąż jednak tylko w ustalonym porządku, kolejności.
W dalszym etapie dodana została obsługa wielowątkowości w aplikacji przez co po ustanowieniu połączenia możliwa była niezależna komunikacja dwustronna. Serwer natomiast zaczął wykorzystywać je do obsługi wielu połączeń jednocześnie.
Ostatnim głównym etapem było dodanie modułu encrypter.py i usprawnienie komunikacji o szyfrowanie i deszyfrowanie przychodzących wiadomości. Przy okazji naprawiony został błąd w ustalaniu sekretu wynikający ze zbyt dużych wartości kluczy(podczas obliczeń wartości przekraczały zakres typu *integer* w python'ie i bez zgłaszania błędu fałszowały wynik.
Na koniec zaktualizowany został README o aktualne informacje dotyczące aplikacji, która sama została uzupełniona o komentarze zwięźle opisujące funkcje, metody i klasy.

Ostateczna struktura tak wykonanego projektu wygląda następująco:

```
├── clinet.py
├── __init__.py
├── modules
│   ├── clientUI.py
│   ├── diffiHellman.py
│   ├── encrypter.py
│   ├── serverUI.py
│   └── socketUI.py
├── README.md
└── server.py
```

Nieukończone usprawnienia i błędy

Ze względu na niedostatek czasu aplikacja nie posiada wielu(niekiedy niezbędnych) usprawnień.
Głównym problemem jest jej trudność w jej konfiguracji - skrypty nie przyjmują żadnych argumentów i wszelkie atrybuty jak nazwa, adres serwera, numer portu, sposób szyfrowania czy poziom logowania są “zahardcodowane” wewnątrz skryptów.
Mało interaktywne jest też przerywanie działania aplikacji - z założeniu miało się to dziać na życzenie użytkownika by bezpiecznie zakończyć wszystkie wątki i zamknąć połączenia. Niestety wstępnie zaimplementowany mechanizm mający to robić nie funkcjonuje i jedyną możliwością jest zabicie procesu aplikacji. Skutkuje to niekiedy w zawieszeniu się procesu obsługi kieszeni i zablokowaniem do niej dostępu na jakiś czas mimo że aplikacja powinna już być zakończona.
Do mniej znaczących problemów zalicza się obsługa odbioru wiadomości - na wzór instruktażu z dokumentacji biblioteki socket miała ona się odbywać tzw. chunk’ami, blokami. Niestety po odebraniu pierwszego, przy próbie sprawdzeniu wejścia aplikacja zawiesza się i oczekuje nowego inputu. Zostało to jednak ominięte poprzez zdefiniowanie dużego rozmiaru bloku i usunięciu pętli sprawdzającej czy coś jeszcze czeka na wejściu.
Inną niedokończoną funkcjonalnością jest jednoczesne wzajemne dzielenie się wynikami jeszcze podczas dopełniania się protokołu Diffiego-Hellmana
Brakuje też implementacji szyfrowania typu XOR

Wnioski

Mimo ograniczonych zasobów czasowych udało się stworzyć funkcjonujący według większości założeń komunikator. Komunikacja przez kieszeń okazała się dużo łatwiejsza w implementacji niż początkowo się spodziewałem. Protokół Diffiego-Hellmana sprytnie wykorzystuje matematyczne właściwości do ukrycia klucza użytego w szyfrowaniu. Nie wydaje się to jednak być zabezpieczeniem nie do złamania, gdyż wszelkie dane potrzebne do jego obliczenia są jawnie przesyłane w początkowej fazie komunikacji. Dla informatyka zaznajomionego z aplikacją i wykorzystywanymi przez nią metodami rozszyfrowanie wiadomości nie powinno stanowić trudności.