Bezpieczeństwo Systemów Komputerowych

Zadanie 2: Uwierzytelnianie

Państwa zadanie będzie polegało na implementacji protokołu Otway-Rees. Zgodnie ze schematem podanym na wykładzie:

1. Klient wysyła serwerowi komunikat jako krotkę[1]

$$(\{1\}, \{2\}, \{3\}, \{4\})$$

gdzie

{1} losowa wartość liczbowa wylosowana przez klienta, oznaczmy ją jako m_1

{2} identyfikator klienta

{3} identyfikator serwera

{4} zaszyfrowana kluczem K_{KT}[2] wiadomość w postaci napisu:

$$\{1\} : \{2\} : \{3\} : \{4\}$$

gdzie

{1} nonce wylosowany przez klienta, którą oznaczymy przez n_K

{2} losowa wartość, którą wcześniej oznaczyliśmy jako m_1

{3} identyfikator klienta

{4} identyfikator serwera

2. Serwer odbiera komunikat od klienta i wysyła do zaufanego serwera krotkę:

$$(\{1\}, \{2\}, \{3\}, \{4\}, \{5\})$$

gdzie

{1} losowa wartość, którą wcześniej oznaczyliśmy jako m_1

{2} identyfikator klienta

{3} identyfikator serwera

{4} zaszyfrowana kluczem K_{KT} wiadomość w postaci napisu:

$$\{1\} : \{2\} : \{3\} : \{4\}$$

gdzie

{1} nonce wylosowany przez klienta, którą oznaczyliśmy przez n_K

{2} losowa wartość, którą wcześniej oznaczyliśmy jako m_1

{3} identyfikator klienta

{4} identyfikator serwera

[1] Przez krotkę mamy rozumieć *tuple*

[2] klucz współdzielony przez klienta i zaufany serwer

{5} zaszyfrowana kluczem K_{ST}[3] wiadomość postaci napisu:

$$\{1\} : \{2\} : \{3\} : \{4\}$$

gdzie

{1} nonce wylosowany przez serwer, którą oznaczymy przez n_S
{2} losowa wartość, którą wcześniej oznaczyliśmy jako m_1
{3} identyfikator klienta
{4} identyfikator serwera

3. Zaufany serwer wysyła do serwera krotkę:

$$(\{1\}, \{2\}, \{3\})$$

gdzie

{1} losowa wartość, którą wcześniej oznaczyliśmy jako m_1
{2} zaszyfrowana kluczem K_{KT} wiadomość w postaci napisu:

$$\{1\} : \{2\}$$

gdzie

{1} nonce wylosowany przez klienta, którą oznaczyliśmy przez n_K
{2} Klucz sesji K_{KS}

{2} zaszyfrowana kluczem K_{ST} wiadomość w postaci napisu:

$$\{1\} : \{2\}$$

gdzie

{1} nonce wylosowany przez serwer, którą oznaczyliśmy przez n_S
{2} Klucz sesji K_{KS}

4. Serwer wysyła do klienta krotkę:

$$(\{1\}, \{2\})$$

gdzie

{1} losowa wartość, którą wcześniej oznaczyliśmy jako m_1
{2} zaszyfrowana kluczem K_{KT} wiadomość w postaci napisu:

$$\{1\} : \{2\}$$

gdzie

{1} nonce wylosowany przez klienta, którą oznaczyliśmy przez n_K
{2} Klucz sesji K_{KS}

W ramach rozwiązania, mają państwo wysłać na system BaCa archiwum ZIP, które będzie zawierało 4 pliki (archiwum nie może zawierać katalogu, który będzie zawierał te pliki):

- `Server.py`
- `TrustedServer.py`
- `Client.py`
- `Utils.py`

I żadnego pliku więcej!

[3] klucz współdzielony przez serwer i zaufany serwer

1 Zawartość pliku TrustedServer.py

W pliku `TrustedServer.py` ma być zdefiniowana klasa `TrustedServer` (dziedzicząca po klasie `threading.Thread`), która będzie implementowała funkcjonalność zaufanego serwera trzeciej strony. Klasa ma posiadać pola:

input_queue - kolejka[4], która będzie służyła przychodzącym klientom (w tym wypadku klientem będzie serwer) do nawiązania połączenia. Podobnie jak na zajęciach, do nawiązania połączenia wystarczy wysłanie dowolnego komunikatu przez tą kolejkę - przesyłana wartość nie musi być sprawdzana. Gdy klient się zgłosi, ma być utworzony nowy wątek, który będzie obsługiwał tylko jednego klienta.

output_queue - kolejka, która będzie służyła zaufanemu serwerowi do wysłania przychodzącym klientom krotki zawierającej dwie kolejki:

- pierwsza służąca do pisania do nowo utworzonego wątku, który ma obsłużyć jednego przychodzącego klienta
- druga służąca do czytania od nowo utworzonowego wątku, który ma obsłużyć jednego przychodzącego klienta.

Klasa ma posiadać metody:

__init__ , która ma przyjmować argumenty:

keys - słownik kluczy, w którym kluczami są identyfikatory wszystkich użytkowników w systemie, a wartościami dla są klucze używane przy szyfrowaniu symetrycznym.

max_connections - liczba całkowita wyrażająca limit działających równocześnie wątków obsługujących klientów.

run - metoda implementująca nasłuchiwanie na przychodzących klientów, tworzenie nowych wątków i wysyłanie odpowiednich rzeczy do klientów. Uwaga: proszę pamiętać o zadbanie o to, żeby nie tworzyło się zbyt wiele wątków - będzie to testowane

finish - metoda do kończenia wątku, ważne jest, aby można było ładnie zakończyć wątek po zakończonym teście.

2 Zawartość pliku Server.py

W pliku `Server.py` ma być zdefiniowana klasa `Server` (dziedzicząca po klasie `threading.Thread`), która będzie implementowała funkcjonalność serwera przed którym mają się uwierzytelniać klienci. Klasa ma posiadać pola:

input_queue - kolejka, która będzie służyła przychodzącym klientom (w tym wypadku klientem będzie rzeczywiście klient) do nawiązania połączenia. Podobnie jak na zajęciach, do nawiązania połączenia wystarczy wysłanie dowolnego komunikatu przez tą kolejkę - przesyłana wartość nie musi być sprawdzana. Gdy klient się zgłosi, ma być utworzony nowy wątek, który będzie obsługiwał tylko jednego klienta.

output_queue - kolejka, która będzie służyła serwerowi do wysłania przychodzącym klientom krotki zawierającej dwie kolejki:

- pierwsza służąca do pisania do nowo utworzonego wątku, który ma obsłużyć jednego przychodzącego klienta
- druga służąca do czytania od nowo utworzonowego wątku, który ma obsłużyć jednego przychodzącego klienta.

Klasa ma posiadać metody:

__init__ , która ma przyjmować argumenty:

server_id - identyfikator serwera, pod którym serwer będzie znany wszystkim uczestnikom komunikacji

server_key - klucz służący serwerowi do bezpiecznego szyfrowanego symetrycznie porozumiewania się z zaufanym serwerem trzeciej strony

[4] przez kolejkę mamy rozumieć obiekt typu Queue.Queue

max_connections - liczba całkowita wyrażająca limit działających równocześnie wątków obsługujących klientów.

trusted_server - referencja na działający zaufany serwer trzeciej strony

run - metoda implementująca nasłuchiwanie na przychodzących klientów, tworzenie nowych wątków i wysyłanie odpowiednich rzeczy do klientów. Uwaga: proszę pamiętać o zadbanie o to, żeby nie tworzyło się zbyt wiele wątków - będzie to testowane

finish - metoda do kończenia wątku, ważne jest, aby można było ładnie zakończyć wątek po zakończonym teście.

3 Zawartość pliku Client.py

W pliku `Client.py` ma być zdefiniowana klasa `Client` (dziedzicząca po klasie `threading.Thread`), która będzie implementowała funkcjonalność klienta, który ma się uwierzytelnić przed serwerem. Klasa ma posiadać metody:

__init__ , która ma przyjmować argumenty:

client_id - identyfikator klienta, pod którym klient będzie znany wszystkim uczestnikom komunikacji

client_key - klucz służący klientowi do bezpiecznego szyfrowanego symetrycznie porozumiewania się z zaufanym serwerem trzeciej strony

server - referencja na działający serwer, przed którym klient ma się uwierzytelnić.

server_id - identyfikator serwera, przed którym klient ma się uwierzytelnić.

run - metoda implementująca wszystkie działania klienta przewidziane dla niego przez protokół. Wątek po uwierzytelnieniu się powinien wypisać na standardowe wyjście komunikat "OK", a w przypadku braku uwierzytelnienia powinien wypisać komunikat "ERROR". Po wypisaniu komuniaktu na stardowe wyjście powinien zakończyć swoje działanie.

4 Zawartość pliku Utils.py

W pliku `Utils.py` mają się znaleźć 3 funkcje:

- `encrypt(message, key)` - służąca do szyfrowania wiadomości `message` przy użyciu klucza `key`
- `decrypt(message, key)` - służąca do deszyfrowania wiadomości `message` przy użyciu klucza `key`
- `generate_random_key()` - służąca do generowania losowych kluczy

5 Uwagi implementacyjne

- Zadanie będzie uruchamiane w interpreterze pythona w wersji 2.6
- Każdy komunikat może być stringiem "ERROR", w przypadku gdy coś w procesie uwierzytelniania nie będzie się zgadzało, na przykład jeżeli w wiadomości od zaufanego serwera serwer dostanie inną wartość n_S niż się spodziewał. W przypadku jakiegoś błędu uwierzytelniania, klient ma wypisać na standardowe wyjście komunikat "ERROR".

6 Uwagi odnośnie testowania

- Państwa klasy będą w różnych konfiguracjach testowane z klasami napisanymi przez autora zadania.
- W czasie testowania Państwa kod będzie analizowany za pomocą narzędzie `pylint` z dodatkowymi opcjami:

```
pylint --disable=C0111,R0922 --max-args=10 --max-locals=20
```

7 Termin i punktacja

- Termin na oddanie zadania mija 3 tygodnie po ogłoszeniu zadania na stronie BaCa (jednocześnie zostanie umieszczony post na forum wykładu na platformie pegaz)
- Za zadanie można uzyskać 40 punktów, z czego 10 punktów będzie wynikiem działania programu `pylint` na przesłanym kodzie źródłowym.

8 Test jawny

```
from Utils import generate_random_key
from TrustedServer import TrustedServer
from Server import Server
from Client import Client


id_for_alice = 'alice'
id_for_bob = 'bob'
key_for_alice = generate_random_key()
key_for_bob = generate_random_key()

trusted_server = TrustedServer(keys={id_for_alice: key_for_alice, id_for_bob: key_for_bob},max_connections=10)
trusted_server.start()

server = Server(server_id=id_for_bob, server_key=key_for_bob, max_connections=10, trusted_server=trusted_server)
server.start()

client = Client(client_id=id_for_alice, client_key=key_for_alice, server=server, server_id=id_for_bob)
client.start()

client.join()
server.finish()
trusted_server.finish()
```