Sprawozdanie końcowe

Działanie

Klient

Klient działa w następujący sposób:

1. Inicjacja połączenia TCP z serwerem
2. Utworzenie oddzielnego wątku na odbieranie wiadomości od serwera
3. W wątku głównym obsługiwana jest interakcja z terminalem tzn. klient ma opcję wysłać wiadomości: `ClientHello`, zwykłą wiadomość szyfrowaną i wiadomość `EndSession`. Wpisanie "EndSession" do zwykłej wiadomości szyfrowanej ma takie same skutki jak wybranie opcji wysłania `EndSession`. Przed wysłaniem wiadomości `ClientHello` nie można wysłać nic innego. Jest tak także w przypadku zakończenia sesji

```
Connected to server 172.21.36.2:8080
Available options:
-1 - Send Hello Message to server
-2 - Send message
-3 - Send EndSession
1
Sent Hello Message to server
Available options:
-1 - Send Hello Message to server
-2 - Send message
-3 - Send EndSession
Received Hello Message from server
2
Type message content: EndSession
Ended session - to initiate it again send Hello Message
Available options:
-1 - Send Hello Message to server
-2 - Send message
-3 - Send EndSession
```

```
Available options:
-1 - Send Hello Message to server
-2 - Send message
-3 - Send EndSession
1
Sent Hello Message to server
Available options:
-1 - Send Hello Message to server
-2 - Send message
-3 - Send EndSession
Received Hello Message from server
3
Ended session - to initiate it again send ClientHello
```

1. W pobocznym wątku odbieramy wiadomości od serwera. W zależności od tego czy zostały już wymienione wiadomości `Hello`, funkcja przyjmie tylko taką widomość jeżeli taka wymiana jeszcze nie nastąpiła. Po wymianie wątek nasłuchuje już tylko na wiadomości normalne. Jeżeli dotrze do niego wiadomość z zawartością "EndSession" sesja, jest kończona i można ją wznowić dopiero przez wysłanie `ClientHello`

```
Available options:
-1 - Send Hello Message to server
-2 - Send message
-3 - Send EndSession
Received Hello Message from server
2
Type message content: My first message to server
Sent normal message
Available options:
-1 - Send Hello Message to server
-2 - Send message
-3 - Send EndSession
Session ended by server. To initiate it again send ClientHello
```

```
Available options:
-1 - Send Hello Message to server
-2 - Send message
-3 - Send EndSession
Session ended by server. To initiate it again send ClientHello
2
Type message content: Why you ended session :C
Session is not active, message won't be sent
Available options:
-1 - Send Hello Message to server
-2 - Send message
-3 - Send EndSession
```

Serwer

Działanie serwera jest trochę bardziej złożone:

1. Tak jak w przypadku klienta, serwer najpierw wydziela osobny wątek na obsługę przychodzących połączeń, a w początkowym zostaje interakcja w terminalu
2. Dla każdego nowego klienta tworzony jest kolejny wątek, w którym obsługiwane są wiadomości.
3. W terminalu serwer ma opcję wysłać tylko normalną wiadomość na adres IP, który wysłał już `ClientHello`, w innym przypadku nie może tego zrobić

```
Server listening on 172.21.36.2:8080
If you want to print clients connected write `list`, else just type IP:PORT of client you want to connect
Client connected: 172.21.36.3:57874
list
User at address 172.21.36.3:57874 NOT READY for messages

172.21.36.3:57874
Enter message to send:
Hi you are my first client!
No active connection to 172.21.36.3:57874
Received Hello Message from 172.21.36.3:57874
Sent Hello Message response to 172.21.36.3:57874
list
User at address 172.21.36.3:57874 READY for messages

172.21.36.3:57874
Enter message to send:
HIIIII
Sent message to 172.21.36.3:57874: HIIIII
```

4. Po wysłaniu `EndSession` serwer ponownie nie może wysłać wiadomości

```
172.21.36.3:34902
Enter message to send:
EndSession
Ending connection with 172.21.36.3:34902
list
User at address 172.21.36.3:34902 NOT READY for messages

Received Hello Message from 172.21.36.3:34902
Sent Hello Message response to 172.21.36.3:34902
list
User at address 172.21.36.3:34902 READY for messages

Session ended by 172.21.36.3:34902. Cant send messages there
172.21.36.3:34902
Enter message to send:
Come back!
No active connection to 172.21.36.3:34902
```

Wiadomości typu `Hello` nie są w żaden sposób szyfrowane (Będzie pokazane w dalszej części dokumentu), natomiast w momencie gdy klient i serwer wymienią klucze i obliczą klucz wspólny, wtedy każda wiadomość jest sprawdzana pod kątem poprawności MAC i następnie odszyfrowywana

Implementacja w kodzie

Klient

Inicjalizacja połączenia i tworzenie wątku na obsługę wiadomości

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    try:
        s.connect((HOST, TCP_PORT))
    except ConnectionRefusedError:
        print(f"Could not connect to server {HOST}:{TCP_PORT}")
        sys.exit(1)

    print(f'Connected to server {HOST}:{TCP_PORT}')
    server_thread = threading.Thread(target=handle_server_messages, args=(s,))
    server_thread.daemon = True
    server_thread.start()
```

Obsługa terminala - wysłanie wiadomości Client Hello

```
    if command_id == "1":
        with HELLO_LOCK:
            if RECEIVED_HELLO:
                print("Session is active, no need to send that message again")
            else:
                message = {
                    "type": "Hello message",
                    "public_key": PUBLIC_KEY,
                    "base": BASE,
                    "modulus": MODULUS
                }
                s.sendall(json.dumps(message).encode('utf-8'))
                print(f"Sent Hello Message to server")
                RECEIVED_HELLO = False
```

Odbieranie wiadomości serwera

```python
def handle_server_messages(server_socket: socket.socket):
    global RECEIVED_HELLO
    global SHARED_KEY
    while True:
        response = server_socket.recv(1024)

        if not response:
            print("Server has closed the connection. Exiting client.")
            server_socket.close()
            sys.exit(0)

        with HELLO_LOCK:
            if not RECEIVED_HELLO:
                try:
                    response_message = json.loads(response.decode('utf-8'))
                    if "type" in response_message and response_message["type"] == "Hello message":
                        RECEIVED_HELLO = True
                        SHARED_KEY = (response_message["public_key"]**PRIVATE_KEY) % MODULUS
                        print("Received Hello Message from server")
                        # print("Shared_key: ", SHARED_KEY)
                    else:
                        print("Expected Hello Message, got something else.")
                except json.JSONDecodeError:
                    print("Expected Hello Message, got something else.")
            else:
                message = decrypt_message(response)
                if message == "EndSession":
                    print("Session ended by server. To initiate it again send ClientHello")
                    RECEIVED_HELLO = False
                else:
                    print(f'Received normal message from server: {message}')
```

`RECEIVED_HELLO` jest zmienną `True/False` i to dzięki niej klient wie, czy nastąpiła już wymiana kluczy czy nie. Jest odpowiednio zmieniana w przypadku otrzymania lub wysłania `EndSession`. Dodatkowo została zastosowana blokada, ponieważ oba wątki korzystają z tej zmiennej. Wprowadziliśmy to, bo nie wiedzieliśmy czy problemy ze współbieżnością w ogóle by wystąpiły, a tak nie musieliśmy się tym przejmować. Dodatkowo nie jest to bardzo zaawansowany mechanizm a mógł oszczędzić trochę czasu na debuggowanie.

Serwer

Implementacja po stronie serwera jest prawie symetryczna poza kilkoma szczegółami

Zamiast `RECEIVED_HELLO` jest słownik `CLIENTS`, który przetrzymuje takie wartości jak klucz symetryczny, status połączenia i socket clienta

Inicjalizacja serwera

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((IP, TCP_PORT))
    s.listen()
    print(f"Server listening on {IP}:{TCP_PORT}")

    serving_clients_thread = threading.Thread(target=handle_serving_clients, args=(s,))
    serving_clients_thread.daemon = True
    serving_clients_thread.start()
```

Tutaj widać, że od razu oddelegowujemy obsługę połączeń gdzie indziej

Obsługa połączeń

```
def handle_serving_clients(server_socket):
    while True:
        conn, addr = server_socket.accept()
        addr_str = f"{addr[0]}:{addr[1]}"
        print(f"Client connected: {addr_str}")
        with CLIENTS_LOCK:
            CLIENTS[addr_str] = [conn, False, -1]
        client_thread = threading.Thread(target=handle_client, args=(conn, addr_str, PRIVATE_KEY,))
        client_thread.daemon = True
        client_thread.start()
```

Tutaj akceptowane są połączenia i tworzone kolejne wątki

Obsługa klienta

```
def handle_client(conn, addr_str, private_key):
    while True:
        data = conn.recv(1024)
        if not data:
                print(f'Client {addr_str} has closed the connection.')
                conn.close()
                with CLIENTS_LOCK:
                    del CLIENTS[addr_str]
                break
        with CLIENTS_LOCK:
            if not CLIENTS[addr_str][1]:
                try:
                    message = json.loads(data.decode('utf-8'))
                    if "type" in message and message["type"] == "Hello message":
                        print(f"Received Hello Message from {addr_str}")
                        CLIENTS[addr_str][1] = True
                        public_key = (message["base"] ** private_key) % message["modulus"]
                        response = {
                            "type": "Hello message",
                            "public_key": public_key
                        }
                        CLIENTS[addr_str][2] = (message["public_key"]**PRIVATE_KEY) % message['modulus']
                        # print("Shared_key: ", CLIENTS[addr_str][2])
                        conn.sendall(json.dumps(response).encode('utf-8'))
                        print(f"Sent Hello Message response to {addr_str}")
                    else:
                        print(f"Received message from {addr_str}, but ignoring until Hello Message.")
                except json.JSONDecodeError:
                    print(f"Received message from {addr_str}, but ignoring until Hello Message.")
            else:
                message = decrypt_message(data, CLIENTS[addr_str][2])
                if message == "EndSession":
                    print(f'Session ended by {addr_str}. Can''t send messages there')
                    CLIENTS[addr_str][1]= False
                    CLIENTS[addr_str][2]= -1
                else:
                    print(f'Received normal message from {addr_str}: {message}')
```

Tak samo jak w przypadku komunikacji po stronie klienta - pierwsze w kolejności to sprawdzamy czy połączenie jest aktywne, jeżeli nie to próbujemy uzyskać `ClientHello`, jeżeli tak to odkodowujemy wiadomość.

W obu przypadkach jeżeli dotrze wiadomość, ale nie jest to `Hello Message` to odbiorca dostaje komunikat, że coś przyszło ale nie jest to oczekiwana wiadomość. Normalnie nie powinno się zdarzyć, bo po obu stronach blokujemy możliwość wysłania zwykłej wiadomości przed `Hello Message` ale na fazie implementacji było pomocne, bo był to znak że coś nie działa.

Szyfrowanie i odszyfrowywanie

Po obu stronach przebiega tak samo.

```python
def encrypt_message(message):
    expanded_key = SHA256.new(str(SHARED_KEY).encode('utf-8')).digest()
    # print(SHARED_KEY)


    iv = get_random_bytes(16)
    cipher = AES.new(expanded_key, AES.MODE_CBC, iv)

    padding_length = 16 - (len(message) % 16)
    padded_message = message + chr(padding_length) * padding_length

    ciphertext = cipher.encrypt(padded_message.encode('utf-8'))

    hmac = HMAC.new(expanded_key, digestmod=SHA256)
    hmac.update(iv + ciphertext)

    # print(f"IV: {iv}")
    # print(f"Ciphertext: {ciphertext}")
    # print(f"Computed HMAC: {hmac.digest()}")

    final_message = iv + ciphertext + hmac.digest()
    # print(final_message)

    return final_message
```

1. Klucz wspólny rozszerzamy do 256 bitów
2. Generujemy losowy wektor początkowy - nie było to wspomniane w Sprawozdaniu Wstępnym ale jest potrzebne do działania algorytmu
3. Wyrównujemy wiadomość tak żeby mogły powstać pełne bloki
4. Szyfrujemy wiadomość algorytmem AES w trybie CBC
5. Algorytmem HMAC-SHA256 generujemy kod MAC
6. Ostateczną wiadomość tworzymy z niezaszyfrowanego wektora początkowego (nie jest to potrzebne), zaszyfrowanej wiadomości i kod MAC

Analogicznie przebiega odszyfrowywanie

```python
def decrypt_message(encrypted_message):
    iv = encrypted_message[:16]
    ciphertext = encrypted_message[16:-32]
    received_hmac = encrypted_message[-32:]


    expanded_key = SHA256.new(str(SHARED_KEY).encode('utf-8')).digest()


    hmac = HMAC.new(expanded_key, digestmod=SHA256)
    hmac.update(iv + ciphertext)
    try:
        hmac.verify(received_hmac)
    except ValueError:
        raise ValueError("Invalid MAC")

    cipher = AES.new(expanded_key, AES.MODE_CBC, iv)


    decrypted_data = unpad(cipher.decrypt(ciphertext), 16)
    return decrypted_data.decode('utf-8')
```

1. Rozdzielamy otrzymaną wiadomość - ustaliliśmy, że szyfrowane wiadomości będą zawsze miały ten sam schemat
2. Rozszerzamy klucz wspólny
3. Sprawdzamy kod MAC, w przypadku nieprawidłowego informujemy o tym
4. Odszyfrowujemy wiadomość
5. Usuwamy wypełnienie jeżeli takie istnieje

Wszystko kluczowe funkcje zostały zapewnione przez bibliotekę `pycryptodome` moduł `Crypto`.

Monitoring pakietów

Do monitorowania wymiany pakietów użyty został `tcpdump`, a konkretnie komenda

```
tcpdump -i eth0 -X tcp
```

Wywołana została na kontenerze serwera i pozwoliła na wyfiltrowanie tylko niezbędnych pakietów razem z ich zawartością

Wymiana wiadomości Hello Message

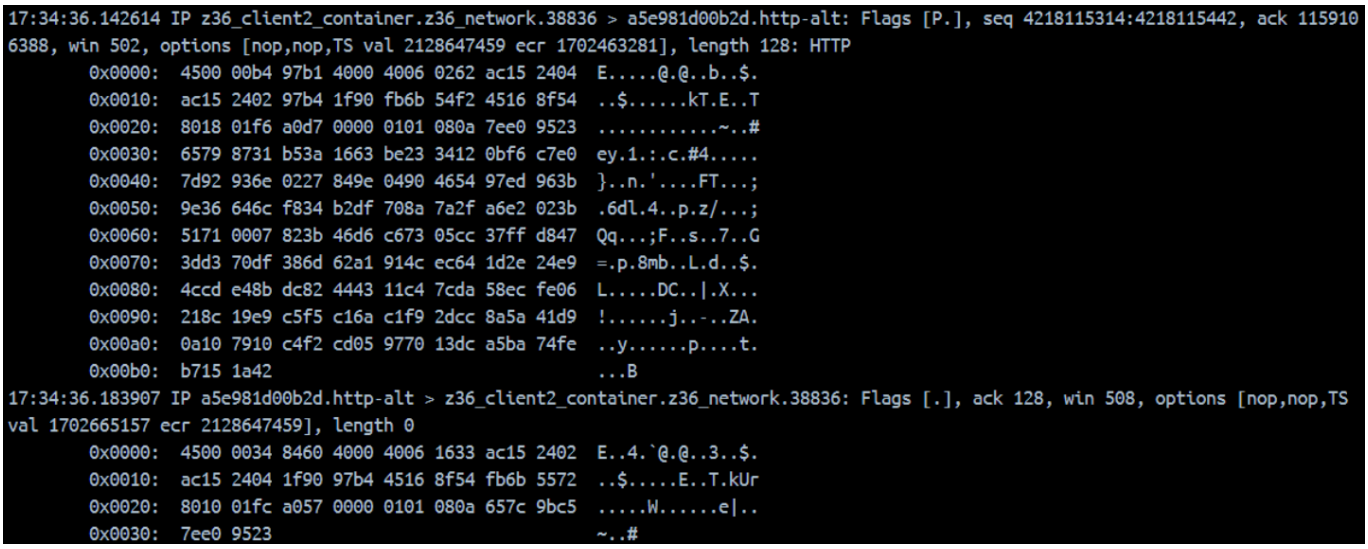
Widać, że kontener klienta najpierw wysyła `Client Hello`, a serwer od razu odpowiada mu potwierdzeniem

Następnie

Serwer odpowiada `Server Hello` i klient odsyła mu potwierdzenie.

Na tym etapie komunikacja przebiega prawidłowo. Kolejność widać w logach.

Szyfrowana konwersacja



Tutaj nie da się już z logów uzyskać treści wiadomości

Ręczne odszyfrowanie

Do osobnego pliku `manual_description.py` skopiowana została wiadomość wyciągnięta z logów.

```
hex_message =
"b53a1663be2334120bf6c7e07d92936e0227849e0490465497ed963b9e36646cf834b2df708a7a2fa
6e2023b51710007823b46d6c67305cc37ffd8473dd370df386d62a1914cec641d2e24e94ccde48bdc8
2444311c47cda58ecfe06218c19e9c5f5c16ac1f92dcc8a5a41d90a107910c4f2cd05977013dca5ba7
4feb7151a42"
```

Jako, że mieliśmy informację z logów, że wiadomość ma 128 bajtów to wiadomo było, że trzeba wziąć ostatnie 128 bajtów. Początkowe bajty to nagłówki.

Następnie wystarczyło przekonwertować to z postaci hexadecymalnej do bajtów i odszyfrować wiadomość tak samo jak odszyfrowywana jest u klienta i serwera

```python
encrypted_message = bytes.fromhex(hex_message)

iv = encrypted_message[:16]
ciphertext = encrypted_message[16:-32]
received_hmac = encrypted_message[-32:]
expanded_key = SHA256.new(str(shared_key).encode('utf-8')).digest()

hmac = HMAC.new(expanded_key, digestmod=SHA256)

hmac.update(iv + ciphertext)
print(f"IV: {iv}")
print(f"Ciphertext: {ciphertext}")
print(f"Received HMAC: {received_hmac}")
print(f"Computed HMAC: {hmac.digest()}")
try:
    hmac.verify(received_hmac)
except ValueError:
    raise ValueError("Invalid MAC")

cipher = AES.new(expanded_key, AES.MODE_CBC, iv)


decrypted_data = unpad(cipher.decrypt(ciphertext), 16)
print(decrypted_data.decode('utf-8'))
```

W wyniku działania programu uzyskujemy następującą wiadomość

```
PS C:\Users\Mateusz\Documents\GitHub\wut-network-programming-lab\Projekt> python .\manual_decryption.py
Server public key: 9
Client public key: 19
Shared key calculated by server: 6
Shared key calculated by client: 6
IV: b'\xb5:\x16c\xbe#4\x12\x0b\xf6\xc7\xe0}\x92\x93n'
Ciphertext: b"\x02'\x84\x9e\x04\x90FT\x97\xed\x96;\x9e6d1\xf84\xb2\xdfp\x8az/\xa6\xe2\x02;Qq\x00\x07\x82;F\xd6\xc6s\x05\xcc7\xff\xd8G=\xd3p\xdf8mb\xa1\x91L\xecd\x1
d.$\xe9L\xcd\xe4\x8b\xdc\x82DC\x11\xc4|\xdaX\xec\xfe\x06!\x8c\x19\xe9"
d.$\xe9L\xcd\xe4\x8b\xdc\x82DC\x11\xc4|\xdaX\xec\xfe\x06!\x8c\x19\xe9"
Received HMAC: b'\xc5\xf5\xc1j\xc1\xf9-\xcc\x8aZA\xd9\n\x10y\x10\xc4\xf2\xcd\x05\x97p\x13\xdc\xa5\xbat\xfe\xb7\x15\x1aB'
Received HMAC: b'\xc5\xf5\xc1j\xc1\xf9-\xcc\x8aZA\xd9\n\x10y\x10\xc4\xf2\xcd\x05\x97p\x13\xdc\xa5\xbat\xfe\xb7\x15\x1aB'
Computed HMAC: b'\xc5\xf5\xc1j\xc1\xf9-\xcc\x8aZA\xd9\n\x10y\x10\xc4\xf2\xcd\x05\x97p\x13\xdc\xa5\xbat\xfe\xb7\x15\x1aB'
This is encypted message - if you managed to decode it - congrats!
PS C:\Users\Mateusz\Documents\GitHub\wut-network-programming-lab\Projekt>
```

```
This is encypted message - if you managed to decode it - congrats!
```

W programie umieszczone były dodatkowo wartości użyte do obliczenia klucza wspólnego

Dodatkowe informacje

Sposób uruchomienia środowiska w pliku `README.md`

Logi z `tcpdump` pochodzą z Docker Desktop chodzącym na prywatnym sprzęcie

Parametry użyte w tych testach to

```
SERVER_PRIVATE_KEY = 10
CLIENT_PRIVATE_KEY = 15
BASE = 5
MODULUS = 23
```