Sprawozdanie końcowe

Projekt: Mini TLS – wariant W1 (Encrypt-then-MAC)

Alesia Filinkova, Weronika Maślana, Diana Pelin

Dowód że program działa

Serwer

```
[+] Server listening on port 4444
server> [+] Client connected: ('172.18.0.3', 54554)
[+] Client ID=1 from ('172.18.0.3', 54554)
[1] secretMessage
EndSession 1
[+] Sent EndSession to client 1
server> [-] Client 1 disconnected
```

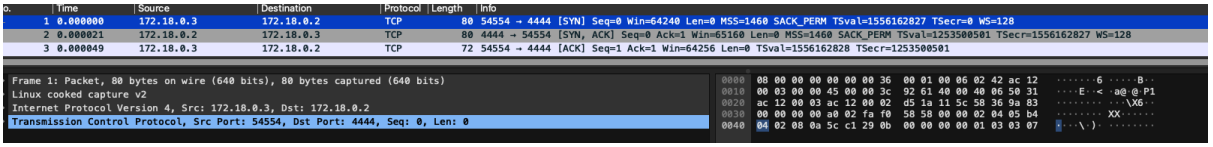
Klient

```
[+] Secure session established
>> secretMessage
>>
[!] Session ended by server

[+] Client terminated
```

Komunikacja między serwerem i klientem

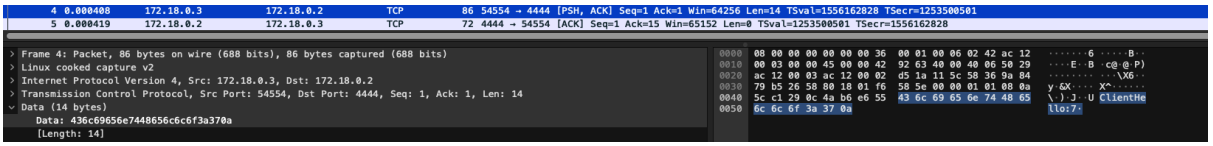
	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.18.0.3	172.18.0.2	TCP	80	54554 → 4444 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1556162827 TSecr=0 WS=128
2	0.000021	172.18.0.2	172.18.0.3	TCP	80	4444 → 54554 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1253500501 TSecr=1556162827 WS=128
3	0.000049	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1556162828 TSecr=1253500501
4	0.000408	172.18.0.3	172.18.0.2	TCP	86	54554 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=14 TSval=1556162828 TSecr=1253500501
5	0.000419	172.18.0.2	172.18.0.3	TCP	72	4444 → 54554 [ACK] Seq=1 Ack=15 Win=65152 Len=0 TSval=1253500501 TSecr=1556162828
6	0.000956	172.18.0.2	172.18.0.3	TCP	86	4444 → 54554 [PSH, ACK] Seq=1 Ack=15 Win=65152 Len=14 TSval=1253500501 TSecr=1556162828
7	0.001011	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=15 Ack=15 Win=64256 Len=0 TSval=1556162828 TSecr=1253500501
8	22.165231	172.18.0.3	172.18.0.2	TCP	117	54554 → 4444 [PSH, ACK] Seq=15 Ack=15 Win=64256 Len=45 TSval=1556184993 TSecr=1253500501
9	22.206076	172.18.0.2	172.18.0.3	TCP	72	4444 → 54554 [ACK] Seq=15 Ack=60 Win=65152 Len=0 TSval=1253522707 TSecr=1556184993
10	54.904899	172.18.0.2	172.18.0.3	TCP	114	4444 → 54554 [PSH, ACK] Seq=15 Ack=60 Win=65152 Len=42 TSval=1253555406 TSecr=1556184993
11	54.905051	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=60 Ack=57 Win=64256 Len=0 TSval=1556217733 TSecr=1253555406
12	57.844635	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [FIN, ACK] Seq=60 Ack=57 Win=64256 Len=0 TSval=1556220672 TSecr=1253555406
13	57.844872	172.18.0.2	172.18.0.3	TCP	72	4444 → 54554 [FIN, ACK] Seq=57 Ack=61 Win=65152 Len=0 TSval=1253558346 TSecr=1556220672
14	57.844927	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=61 Ack=58 Win=64256 Len=0 TSval=1556220673 TSecr=1253558346

1. Nawiązanie połączenia



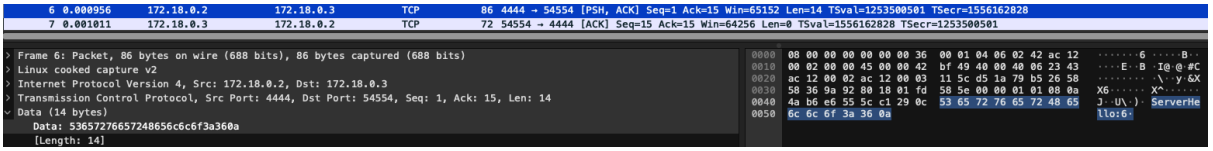
Nawiązanie połączenia pomiędzy klientem a serwerem odbywa się zgodnie z mechanizmem TCP three-way handshake. Jak pokazano na zrzucie z Wiresharka, klient inicjuje połączenie pakietem SYN, serwer odpowiada pakietem SYN-ACK, a następnie klient potwierdza połączenie pakietem ACK. Dopiero po poprawnym zestawieniu połączenia TCP rozpoczyna się właściwa komunikacja

2. Wysłanie wiadomości ClientHello



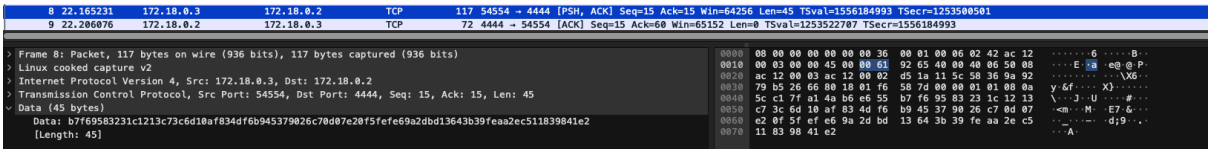
Po zestawieniu połączenia TCP klient wysyła wiadomość ClientHello. Wiadomość ta przesyłana jest w postaci jawnej. Kolejny pakiet zawiera jedynie flagę ACK i stanowi potwierdzenie odebrania ClientHello przez serwer na poziomie protokołu TCP, bez przesyłania danych aplikacyjnych

3. Wysłanie wiadomości ServerHello



W odpowiedzi na wiadomość ClientHello serwer przesyła wiadomość ServerHello. Wiadomość jest w postaci jawnej i zawiera klucz publiczny serwera. Następnie klient potwierdza odebranie wiadomości ServerHello pakietem ACK.

4. Wysłanie dowolnej wiadomości



EncryptedMessage:

b7f69583231c1213c73c6d10af834df6b945379026c70d07e20f5fefe69a2dbd13643b39feaa2ec511839841e2

Kluczy użyte podczas szyfrowanie:

OTP_KEY=c493f6f146685f76b44f0c77ca88120c

MAC_KEY=b8bc89f534fe69b6828827b974e68849

Do odszyfrowania wiadomości użyto prostego kodu (innego niż używa serwer podczas odszyfrowania wiadomości)

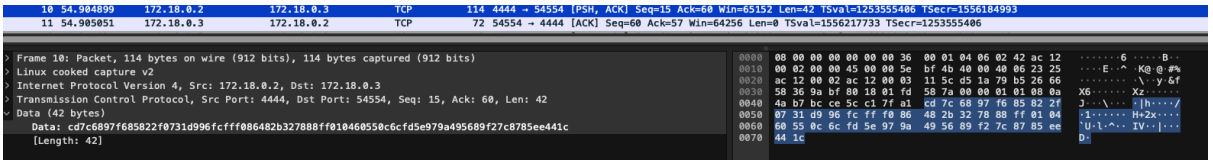
```
msg_hex = "b7f69583231c1213c73c6d10af834df6b945379026c70d07e20f5fefe69a2dbd13643b39feaa2ec511839841e2"
otp_hex = "c493f6f146685f76b44f0c77ca88120c"
mac_key_hex = "b8bc89f534fe69b6828827b974e68849"

msg = unhexlify(msg_hex)
otp_key = unhexlify(otp_hex)
mac_key = unhexlify(mac_key_hex)

ciphertext = msg[:-32]
mac = msg[-32:]

plaintext = bytes(c ^ otp_key[i % len(otp_key)] for i, c in
                  enumerate(ciphertext))
print("PLAINTEXT:", plaintext.decode())
```

Wynik: PLAINTEXT: secretMessage

Odszyfrowana wiadomość przez serwera

```
[+] Server listening on port 4444
server> [+] Client connected: ('172.18.0.3', 54554)
[+] Client ID=1 from ('172.18.0.3', 54554)
[1] secretMessage
EndSession 1
[+] Sent EndSession to client 1
server> [-] Client 1 disconnected
```

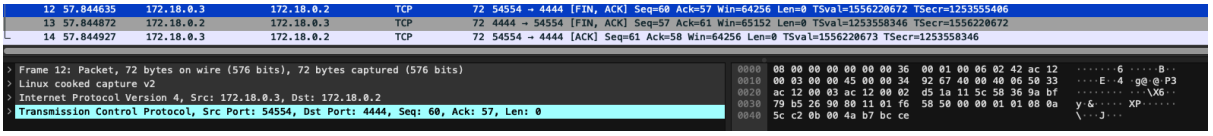
5. Wysłanie wiadomości EndSession



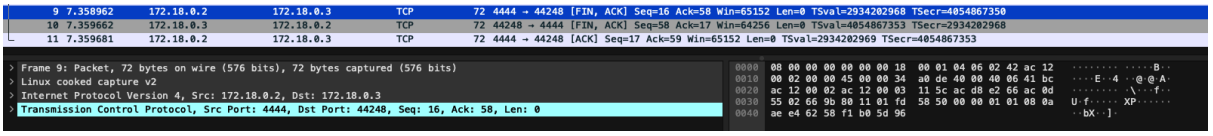
Zakończenie sesji realizowane jest poprzez wysłanie przez serwer zaszyfrowanej wiadomości EndSession. Treść wiadomości nie jest czytelna, ponieważ podlega szyfrowaniu z wykorzystaniem klucza sesyjnego. Następnie klient potwierdza odbiór wiadomości pakietem ACK, co kończy komunikację na poziomie protokołu aplikacyjnego.

6. Zakończenie połączenia

Wariant 1: serwer wywołuje endSession



Wariant 2: klient wywołuje endSession



Po zakończeniu komunikacji na poziomie protokołu aplikacyjnego następuje poprawne zamknięcie połączenia TCP. Klient/serwer inicjuje zakończenie sesji pakietem FIN, ACK, na co serwer/klient odpowiada własnym pakietem FIN, ACK. Ostatecznie klient/serwer potwierdza zamknięcie połączenia pakietem ACK, co odpowiada klasycznemu mechanizmowi four-way handshake protokołu TCP

Opis użytych algorytmów

Algorytm wymiany kluczy(Diffie–Hellman key exchange)

```
9   def dh_generate_private():
10      return int.from_bytes(os.urandom(2), "big")
11
12
13  def dh_generate_public(priv):
14      return pow(G, priv, P)
```

Generowanie kluczy

Klient:

```python
# --- ClientHello ---
priv = dh_generate_private()
pub = dh_generate_public(priv)
s.sendall(f"ClientHello:{pub}\n".encode())

# --- ServerHello ---
data = b""
while not data.endswith(b"\n"):
    chunk = s.recv(1)
    if not chunk:
        print("[!] Connection closed by server")
        return
    data += chunk
```

Serwer:

```python
# --- ClientHello ---
data = recv_line(conn)
if not data or not data.startswith("ClientHello"):
    conn.close()
    return

client_pub = int(data.split(":")[1])

# --- ServerHello ---
server_priv = dh_generate_private()
server_pub = dh_generate_public(server_priv)
conn.sendall(f"ServerHello:{server_pub}\n".encode())

shared = dh_compute_shared(client_pub, server_priv)
otp_key, mac_key = derive_keys(shared)
```

W projekcie zastosowano algorytm wymiany kluczy Diffie–Hellman. Klient i serwer generują własne klucze prywatne oraz odpowiadające im klucze publiczne, które są wymieniane w wiadomościach ClientHello i ServerHello. Następnie obie strony obliczają wspólny/dzielony klucz przy użyciu otrzymanego klucza publicznego i własnego klucza prywatnego. Uzyskany sekret wykorzystywany jest jako podstawa do wyprowadzenia kluczy sesyjnych używanych do szyfrowania danych oraz generowania MAC.

Algorytm wyprowadzania kluczy (Key derivation)

```python
def derive_keys(shared_secret: int):
    digest = hashlib.sha256(shared_secret.to_bytes(2, "big")).digest()
    otp = digest[:16]
    mac = digest[16:]
    return otp, mac
```

W celu uzyskania kluczy sesyjnych zastosowano uproszczony mechanizm wyprowadzania kluczy (Key Derivation Function). Funkcja derive_keys przyjmuje jako wejście wspólny klucz uzyskany w wyniku algorytmu Diffie–Hellman i deterministycznie wyprowadza z niego dwa klucze: klucz szyfrujący (OTP_KEY) oraz klucz do generowania kodu MAC (MAC_KEY).

Algorytm szyfrowania danych (One-Time Pad (OTP) / XOR stream)

```python
def otp_encrypt(key: bytes, plaintext: bytes, counter: int = 0) -> bytes:
    ciphertext = bytearray()
    key_len = len(key)
    for i, b in enumerate(plaintext):
        k = key[(i + counter) % key_len]
        ciphertext.append(b ^ k)
    return bytes(ciphertext)


def otp_decrypt(key: bytes, ciphertext: bytes, counter: int = 0) -> bytes:
    return otp_encrypt(key, ciphertext, counter)
```

Do szyfrowania danych zastosowano algorytm One-Time Pad w postaci strumienia XOR. Szyfrowanie polega na wykonaniu operacji XOR pomiędzy kolejnymi bajtami wiadomości a bajtami klucza sesyjnego (OTP_KEY). W przypadku gdy długość wiadomości przekracza długość klucza, klucz jest powtarzany cyklicznie do długości danych.

Ten sam mechanizm wykorzystywany jest zarówno do szyfrowania, jak i do odszyfrowywania wiadomości, ponieważ operacja XOR jest odwracalna. Klucz szyfrujący wykorzystywany w algorytmie OTP pochodzi z procesu wymiany kluczy Diffie–Hellman i jest wspólny dla obu stron sesji.

Schemat zabezpieczenia wiadomości (Encrypt-then-MAC)

```python
41  def encrypt_then_mac(
42      otp_key: bytes, mac_key: bytes, plaintext: bytes, counter: int = 0
43  ) -> bytes:
44      ciphertext = otp_encrypt(otp_key, plaintext, counter)
45      mac = hmac.new(mac_key, ciphertext, hashlib.sha256).digest()
46      return ciphertext + mac
```

W projekcie zastosowano schemat zabezpieczenia wiadomości typu Encrypt-then-MAC. Oznacza to, że treść wiadomości jest w pierwszej kolejności szyfrowana przy użyciu algorytmu OTP, a następnie dla powstałego ciphertextu obliczany jest kod MAC z wykorzystaniem algorytmu HMAC-SHA256.

Po stronie odbiorcy najpierw weryfikowana jest poprawność kodu MAC, a dopiero po jej pozytywnym zakończeniu następuje odszyfrowanie wiadomości.

Napotkane problemy

Podczas realizacji projektu pojawiły się trudności związane z poprawnym zrozumieniem kolejności komunikatów protokołu oraz rozróżnieniem wiadomości aplikacyjnych od pakietów sterujących TCP, takich jak ACK. Wyzwaniem była również analiza ruchu sieciowego w Wiresharku, szczególnie w środowisku Docker, gdzie pakiety pochodzą z wielu interfejsów. Dodatkowo problematyczne okazało się ręczne odszyfrowywanie przechwyconych danych i prawidłowe wydzielenie ciphertextu oraz MAC. Wszystkie te trudności zostały rozwiązane poprzez testy oraz stopniową analizę przechwyconych pakietów.

Wnioski

Zrealizowany projekt potwierdza możliwość implementacji uproszczonego, szyfrowanego protokołu komunikacyjnego opartego na TCP, zgodnie z założeniami zadania. Zastosowanie algorytmu Diffie–Hellman umożliwiło bezpieczne uzgodnienie klucza sesyjnego, a schemat encrypt-then-MAC zapewnił integralność i autentyczność przesyłanych danych. Analiza ruchu sieciowego wykazała, że treść zaszyfrowanych wiadomości nie jest możliwa do odczytania bez znajomości klucza sesyjnego. Projekt pozwolił na praktyczne zrozumienie mechanizmów stosowanych w protokołach bezpieczeństwa, takich jak TLS