UNIVERSITI TEKNIKAL MALAYSIA MELAKA

FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELEKTRONIK DAN KOMPUTER

BERR 2243

DATABASE & CLOUD SYSTEM

SEM 2 2024/2025

EXERCISE: WEEK 8

No	Name	Matrix No	Photo
1	NURUL ANIS HAFIFZA BINTI AMRAN	B122410321	
2	NUR AIN HIDAYAH BINTI ABDUL RAHIM	B122410323	

Submission :

1. **Wireshark Captures**:

o login.pcapng and admin-access.pcapng

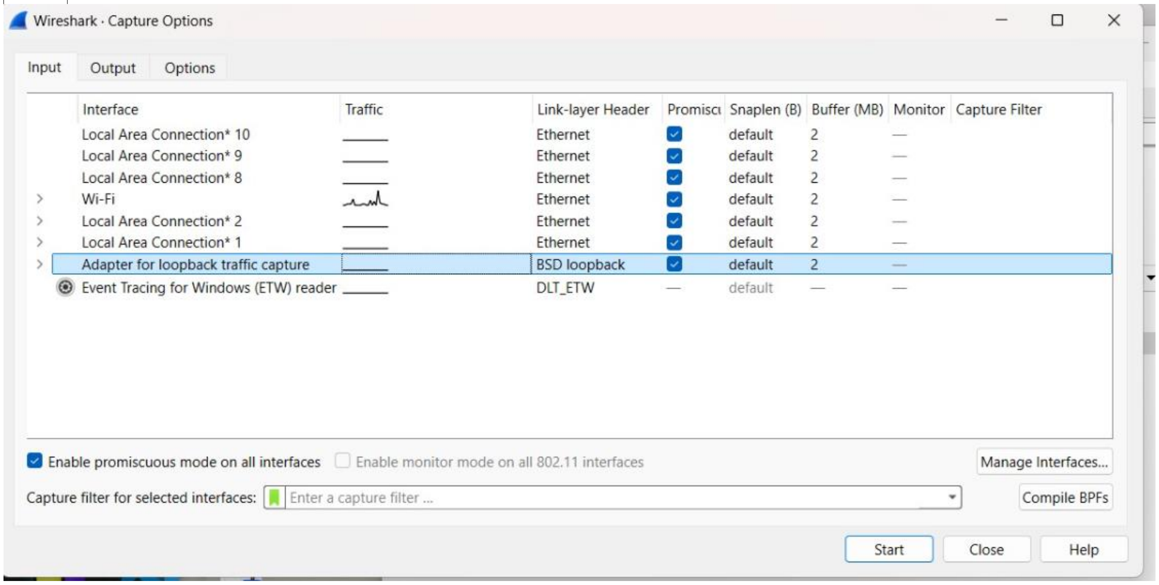
2. **Analysis Report**:

PART 1 : Wireshark Setup

TASK 1: Install Wireshark

1. Wireshark are downloaded and installed.

TASK 2: Configure Loopback Capture



Interface	Traffic	Link-layer Header	Promiscu	Snaplen (B)	Buffer (MB)	Monitor	Capture Filter
Local Area Connection* 10		Ethernet	☑	default	2	—	
Local Area Connection* 9		Ethernet	☑	default	2	—	
Local Area Connection* 8		Ethernet	☑	default	2	—	
Wi-Fi		Ethernet	☑	default	2	—	
Local Area Connection* 2		Ethernet	☑	default	2	—	
Local Area Connection* 1		Ethernet	☑	default	2	—	
Adapter for loopback traffic capture		BSD loopback	☑	default	2	—	
Event Tracing for Windows (ETW) reader		DLT_ETW	—	default	—	—	

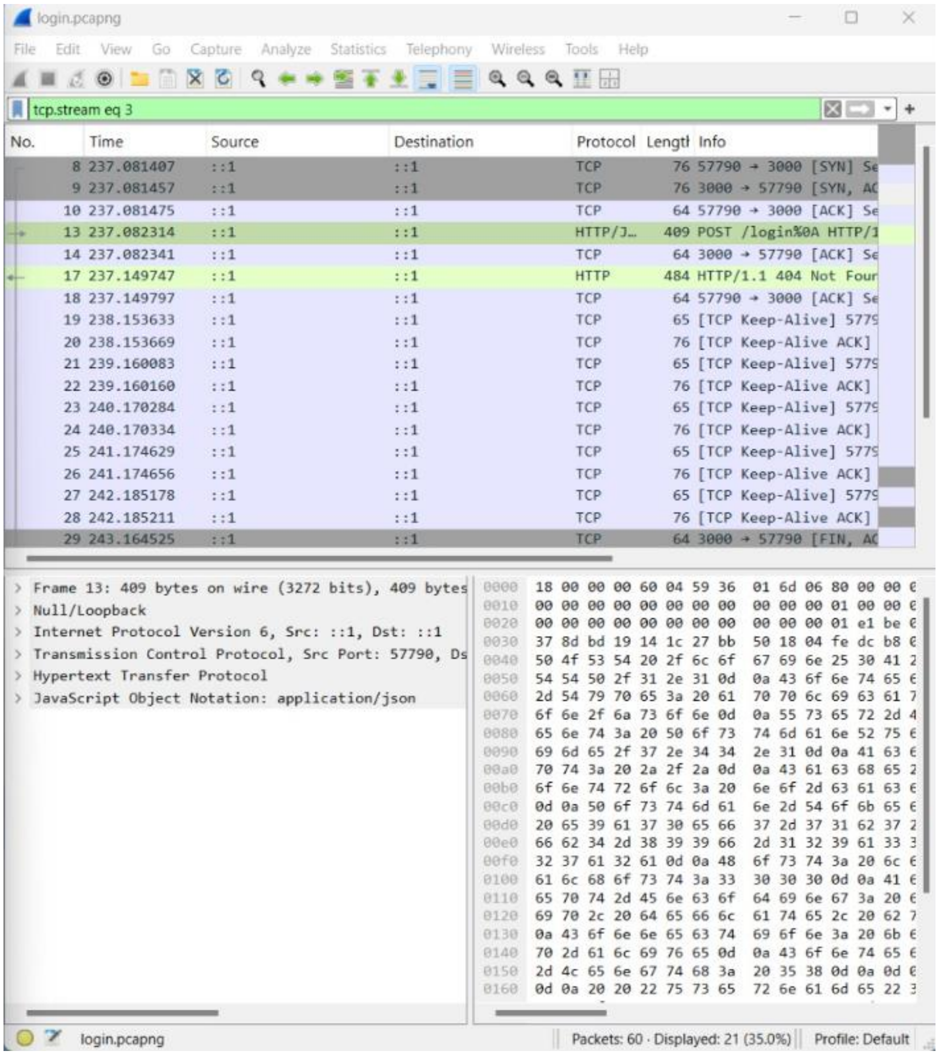
☑ Enable promiscuous mode on all interfaces ☐ Enable monitor mode on all 802.11 interfaces

Capture filter for selected interfaces: Enter a capture filter ...

Npcap Loopback Adapter have been selected and tcp port 3000 are filtered.

PART 2 : Capturing Authentication Flow

TASK 3: Capture Login Request.


login.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.stream eq 3

```
No.   Time         Source   Destination   Protocol   Lengtl Info
    8 237.081407   ::1      ::1           TCP           76 57790 → 3000 [SYN] Se
    9 237.081457   ::1      ::1           TCP           76 3000 → 57790 [SYN, AC
   10 237.081475   ::1      ::1           TCP           64 57790 → 3000 [ACK] Se
   13 237.082314   ::1      ::1           HTTP/J…      409 POST /login%0A HTTP/1
   14 237.082341   ::1      ::1           TCP           64 3000 → 57790 [ACK] Se
   17 237.149747   ::1      ::1           HTTP         484 HTTP/1.1 404 Not Four
   18 237.149797   ::1      ::1           TCP           64 57790 → 3000 [ACK] Se
   19 238.153633   ::1      ::1           TCP           65 [TCP Keep-Alive] 5779
   20 238.153669   ::1      ::1           TCP           76 [TCP Keep-Alive ACK]
   21 239.160083   ::1      ::1           TCP           65 [TCP Keep-Alive] 5779
   22 239.160160   ::1      ::1           TCP           76 [TCP Keep-Alive ACK]
   23 240.170284   ::1      ::1           TCP           65 [TCP Keep-Alive] 5779
   24 240.170334   ::1      ::1           TCP           76 [TCP Keep-Alive ACK]
   25 241.174629   ::1      ::1           TCP           65 [TCP Keep-Alive] 5779
   26 241.174656   ::1      ::1           TCP           76 [TCP Keep-Alive ACK]
   27 242.185178   ::1      ::1           TCP           65 [TCP Keep-Alive] 5779
   28 242.185211   ::1      ::1           TCP           76 [TCP Keep-Alive ACK]
   29 243.164525   ::1      ::1           TCP           64 3000 → 57790 [FIN, AC
```

```
> Frame 13: 409 bytes on wire (3272 bits), 409 bytes
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 57790, Ds
> Hypertext Transfer Protocol
> JavaScript Object Notation: application/json
```

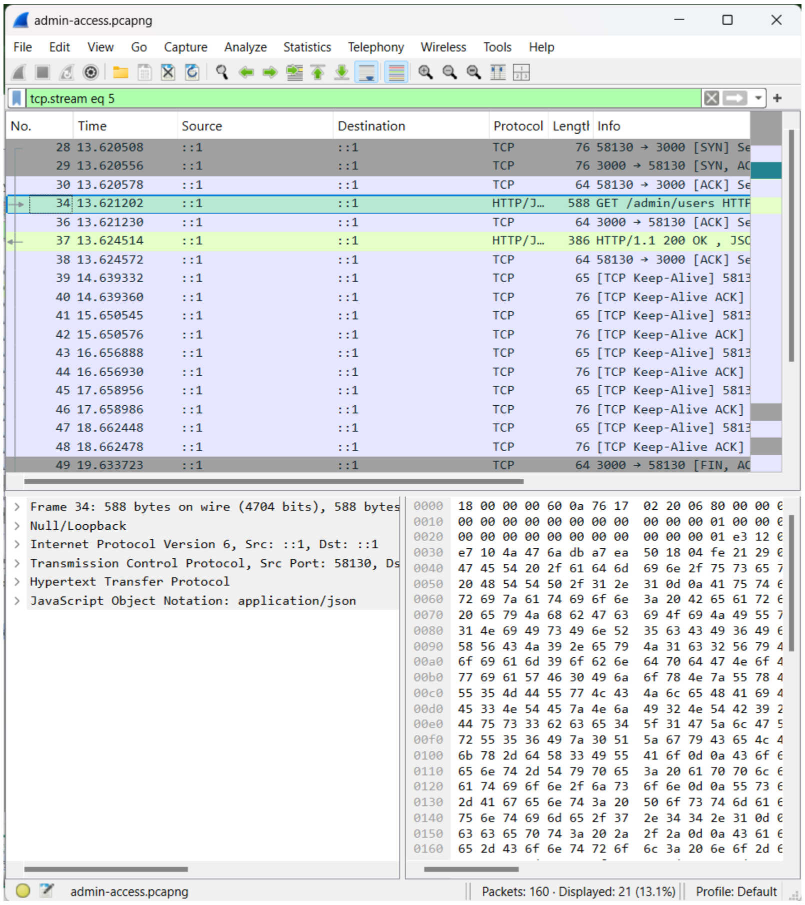
```
0000  18 00 00 00 60 04 59 36  01 6d 06 80 00 00
0010  00 00 00 00 00 00 00 00  00 00 00 01 00 00
0020  00 00 00 00 00 00 00 00  00 00 00 01 e1 be
0030  37 8d bd 19 14 1c 27 bb  50 18 04 fe dc b8
0040  50 4f 53 54 20 2f 6c 6f  67 69 6e 25 30 41
0050  54 54 50 2f 31 2e 31 0d  0a 43 6f 6e 74 65
0060  2d 54 79 70 65 3a 20 61  70 70 6c 69 63 61
0070  6f 6e 2f 6a 73 6f 6e 0d  0a 55 73 65 72 2d
0080  65 6e 74 3a 20 50 6f 73  74 6d 61 6e 52 75
0090  69 6d 65 2f 37 2e 34 34  2e 31 0d 0a 41 63
00a0  70 74 3a 20 2a 2f 2a 0d  0a 43 61 63 68 65
00b0  6f 6e 74 72 6f 6c 3a 20  6e 6f 2d 63 61 63
00c0  0d 0a 50 6f 73 74 6d 61  6e 2d 54 6f 6b 65
00d0  20 65 39 61 37 30 65 66  37 2d 37 31 62 37
00e0  66 62 34 2d 38 39 39 66  2d 31 32 39 61 33
00f0  32 37 61 32 61 0d 0a 48  6f 73 74 3a 20 6c
0100  61 6c 68 6f 73 74 3a 33  30 30 30 0d 0a 41
0110  65 70 74 2d 45 6e 63 6f  64 69 6e 67 3a 20
0120  69 70 2c 20 64 65 66 6c  61 74 65 2c 20 62
0130  0a 43 6f 6e 6e 65 63 74  69 6f 6e 3a 20 6b
0140  70 2d 61 6c 69 76 65 0d  0a 43 6f 6e 74 65
0150  2d 4c 65 6e 67 74 68 3a  20 35 38 0d 0a 0d
0160  0d 0a 20 20 22 75 73 65  72 6e 61 6d 65 22
```

login.pcapng | Packets: 60 · Displayed: 21 (35.0%) | Profile: Default

Send login request to Postman then capture using Wireshark and save it as login.pcapng.

TASK 4: Capture Protected Resource Access

Access admin endpoint in Postman then capture using Wireshark and save it as



admin-access.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.stream eq 5

```
No.  Time       Source   Destination  Protocol Length Info
  28 13.620508  ::1      ::1          TCP        76 58130 → 3000 [SYN] Se
  29 13.620556  ::1      ::1          TCP        76 3000 → 58130 [SYN, AC
  30 13.620578  ::1      ::1          TCP        64 58130 → 3000 [ACK] Se
  34 13.621202  ::1      ::1          HTTP/J…   588 GET /admin/users HTTP
  36 13.621230  ::1      ::1          TCP        64 3000 → 58130 [ACK] Se
  37 13.624514  ::1      ::1          HTTP/J…   386 HTTP/1.1 200 OK , JSC
  38 13.624572  ::1      ::1          TCP        64 58130 → 3000 [ACK] Se
  39 14.639332  ::1      ::1          TCP        65 [TCP Keep-Alive] 5813
  40 14.639360  ::1      ::1          TCP        76 [TCP Keep-Alive ACK]
  41 15.650545  ::1      ::1          TCP        65 [TCP Keep-Alive] 5813
  42 15.650576  ::1      ::1          TCP        76 [TCP Keep-Alive ACK]
  43 16.656888  ::1      ::1          TCP        65 [TCP Keep-Alive] 5813
  44 16.656930  ::1      ::1          TCP        76 [TCP Keep-Alive ACK]
  45 17.658956  ::1      ::1          TCP        65 [TCP Keep-Alive] 5813
  46 17.658986  ::1      ::1          TCP        76 [TCP Keep-Alive ACK]
  47 18.662448  ::1      ::1          TCP        65 [TCP Keep-Alive] 5813
  48 18.662478  ::1      ::1          TCP        76 [TCP Keep-Alive ACK]
  49 19.633723  ::1      ::1          TCP        64 3000 → 58130 [FIN, AC
```

```
> Frame 34: 588 bytes on wire (4704 bits), 588 bytes
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 58130, Ds
> Hypertext Transfer Protocol
> JavaScript Object Notation: application/json
```

```
0000  18 00 00 00 60 0a 76 17  02 20 06 80 00 00
0010  00 00 00 00 00 00 00 00  00 00 00 01 00 00
0020  00 00 00 00 00 00 00 00  00 00 00 01 e3 12
0030  e7 10 4a 47 6a db a7 ea  50 18 04 fe 21 29
0040  47 45 54 20 2f 61 64 6d  69 6e 2f 75 73 65
0050  20 48 54 54 50 2f 31 2e  31 0d 0a 41 75 74
0060  72 69 7a 61 74 69 6f 6e  3a 20 42 65 61 72
0070  20 65 79 4a 68 62 47 63  69 4f 69 4a 49 55
0080  31 4e 69 49 73 49 6e 52  35 63 43 49 36 49
0090  58 56 43 4a 39 2e 65 79  4a 31 63 32 56 79
00a0  6f 69 61 6d 39 6f 62 6e  64 70 64 47 4e 6f
00b0  77 69 61 57 46 30 49 6a  6f 78 4e 7a 55 78
00c0  55 35 4d 44 55 77 4c 43  4a 6c 65 48 41 69
00d0  45 33 4e 54 45 7a 4e 6a  49 32 4e 54 42 39
00e0  44 75 73 33 62 63 65 34  5f 31 47 5a 6c 47
00f0  72 55 35 36 49 7a 30 51  5a 67 79 43 65 4c
0100  6b 78 2d 64 58 33 49 55  41 6f 0d 0a 43 6f
0110  65 6e 74 2d 54 79 70 65  3a 20 61 70 70 6c
0120  61 74 69 6f 6e 2f 6a 73  6f 6e 0d 0a 55 73
0130  2d 41 67 65 6e 74 3a 20  50 6f 73 74 6d 61
0140  75 6e 74 69 6d 65 2f 37  2e 34 34 2e 31 0d
0150  63 63 65 70 74 3a 20 2a  2f 2a 0d 0a 43 61
0160  65 2d 43 6f 6e 74 72 6f  6c 3a 20 6e 6f 2d
```

admin-access.pcapng | Packets: 160 · Displayed: 21 (13.1%) | Profile: Default

o Marked screenshots of TCP streams

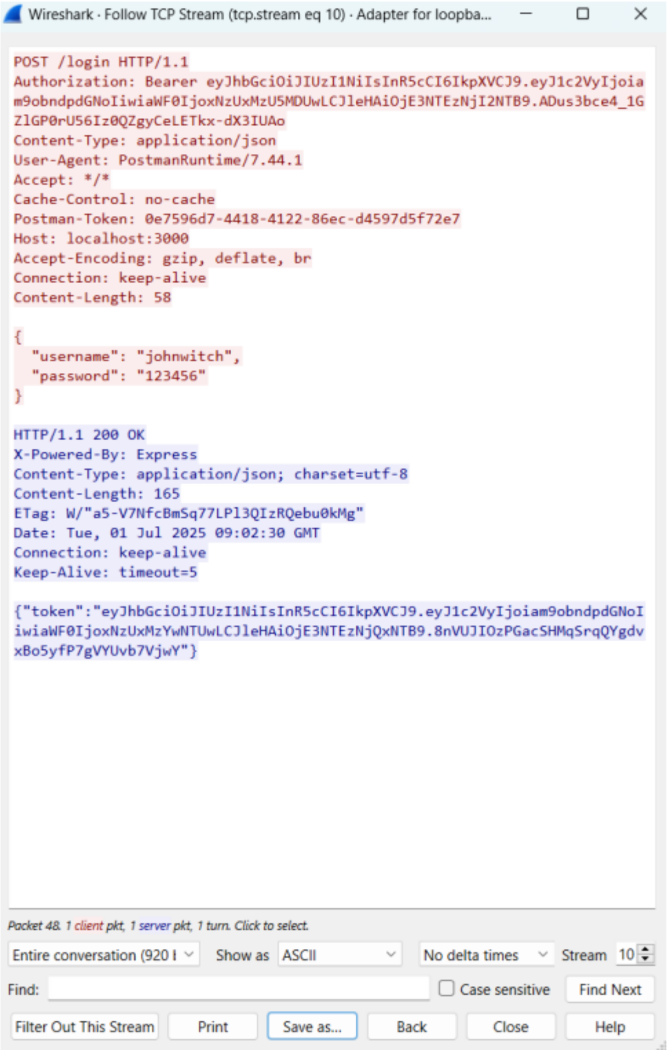
PART 3: TCP Stream Analysis

TASK 5: Analyze Login Request

1 In login.pcapng then Follow and TCP Stream. After that, identify components:

2 HTTPS Request: Method, path, headers, JSON body.

3. HTTPS Respond: Status code, JWT token in body.



Wireshark · Follow TCP Stream (tcp.stream eq 10) · Adapter for loopba...

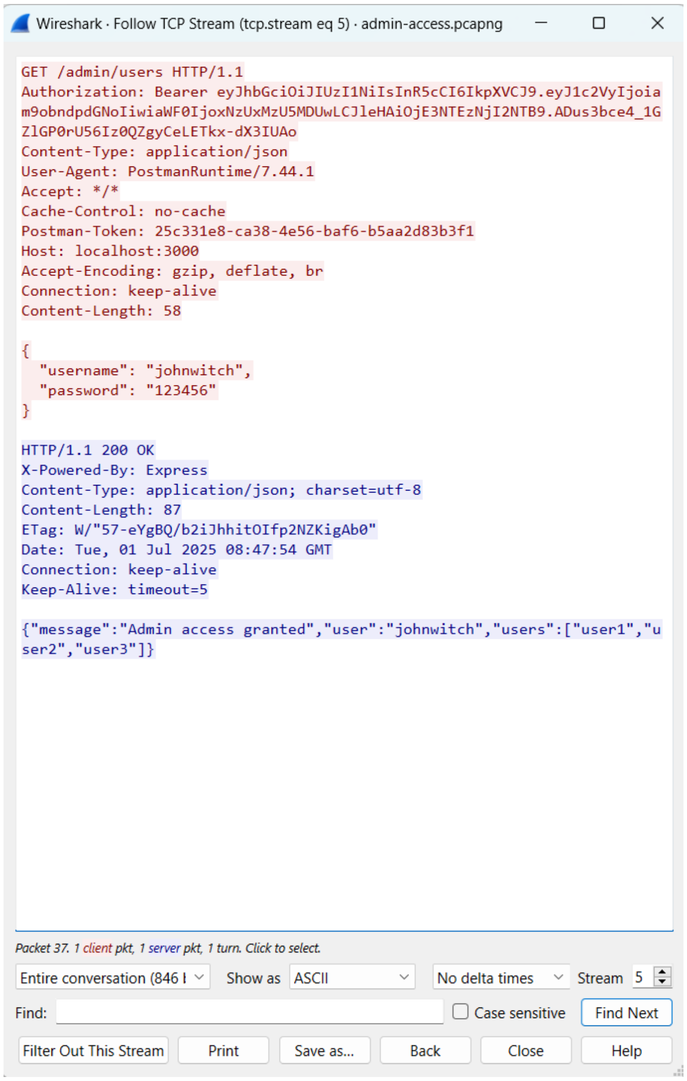
```
POST /login HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoia
m9obndpdGNoIiwiaWF0IjoxNzUxMzU5MDUwLCJleHAiOjE3NTEzNjI2NTB9.ADus3bce4_1G
Z1GP0rU56Iz0QZgyCeLETkx-dX3IUAo
Content-Type: application/json
User-Agent: PostmanRuntime/7.44.1
Accept: */*
Cache-Control: no-cache
Postman-Token: 0e7596d7-4418-4122-86ec-d4597d5f72e7
Host: localhost:3000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 58

{
  "username": "johnwitch",
  "password": "123456"
}

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 165
ETag: W/"a5-V7NfcBmSq77LPl3QIzRQebu0kMg"
Date: Tue, 01 Jul 2025 09:02:30 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiam9obndpdGNoI
iwiaWF0IjoxNzUxMzYwNTUwLCJleHAiOjE3NTEzNjQxNTB9.8nVUJIOzPGacSHMqSrqQYgdv
xBo5yfP7gVYUvb7VjwY"}
```

TASK 6: Analyze Protected Request

1 In admin-access.pcapng then Follow and TCP Stream. After that examine:

2 Authorization: Bearer <token> header

3 Server response with user data


Wireshark · Follow TCP Stream (tcp.stream eq 5) · admin-access.pcapng

```
GET /admin/users HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoia
m9obndpdGNoIiwiaWF0IjoxNzUxMzU5MDUwLCJleHAiOjE3NTEzNjI2NTB9.ADus3bce4_1G
ZlGP0rU56Iz0QZgyCeLETkx-dX3IUAo
Content-Type: application/json
User-Agent: PostmanRuntime/7.44.1
Accept: */*
Cache-Control: no-cache
Postman-Token: 25c331e8-ca38-4e56-baf6-b5aa2d83b3f1
Host: localhost:3000
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 58

{
  "username": "johnwitch",
  "password": "123456"
}

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 87
ETag: W/"57-eYgBQ/b2iJhhitOIfp2NZKigAb0"
Date: Tue, 01 Jul 2025 08:47:54 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{"message":"Admin access granted","user":"johnwitch","users":["user1","u
ser2","user3"]}
```

Packet 37. 1 client pkt, 1 server pkt, 1 turn. Click to select.

Entire conversation (846 I Show as ASCII No delta times Stream 5

Find: □ Case sensitive

Exercise Questions – Wireshark Analysis

1. Protocol Analysis

• What is the exact sequence of TCP packets during the 3-way handshake?
  - SYN: Client → Server
  - SYN-ACK: Server → Client
  - ACK: Client → Server

• How many packets are exchanged for a successful login request?
  - Approximately 6–8 packets, including 3-way handshake, login POST request, server response, and TCP session management packets (e.g., ACK, FIN).

2. Header Inspection

• What headers does Postman include that a browser might omit?
  - Postman-Token
  - User-Agent: PostmanRuntime/...
  - Cache-Control: no-cache
  - Accept-Encoding: gzip, deflate, br

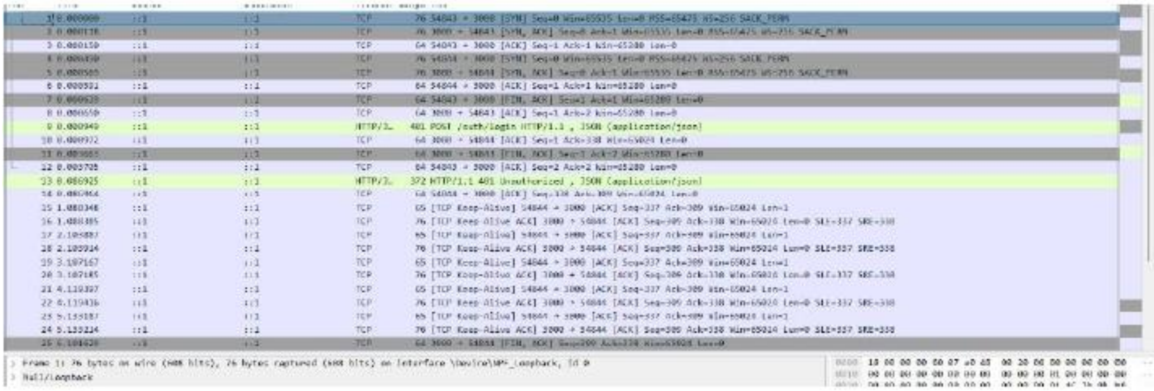
• How is the Content-Length header calculated?
  - It equals the number of bytes in the HTTP request body.
  - For example, the JSON body {"username": "johnwitch", "password": "123456"} has 58 characters, so Content-Length is 58.

3. JWT Transmission

• At which OSI layer is the JWT token visible? Why is this dangerous?
  - Visible at the Application Layer (Layer 7).
  - Dangerous because it is sent in plaintext over HTTP, allowing attackers to steal and reuse the token (session hijacking).

• How would HTTPS change what you see in Wireshark?
  - With HTTPS, the HTTP content is encrypted.
  - You would only see encrypted TLS packets in Wireshark, making the JWT and headers unreadable.

4. Error Handling



```
 1 0.000000  ::1  ::1  TCP       76 54843 → 3000 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
 2 0.000118  ::1  ::1  TCP       76 3000 → 54843 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
 3 0.000150  ::1  ::1  TCP       64 54843 → 3000 [ACK] Seq=1 Ack=1 Win=65280 Len=0
 4 0.000450  ::1  ::1  TCP       76 54844 → 3000 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
 5 0.000505  ::1  ::1  TCP       76 3000 → 54844 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
 6 0.000531  ::1  ::1  TCP       64 54844 → 3000 [ACK] Seq=1 Ack=1 Win=65280 Len=0
 7 0.000628  ::1  ::1  TCP       64 54843 → 3000 [FIN, ACK] Seq=1 Ack=1 Win=65280 Len=0
 8 0.000650  ::1  ::1  TCP       64 3000 → 54843 [ACK] Seq=1 Ack=2 Win=65280 Len=0
 9 0.000949  ::1  ::1  HTTP/J…  401 POST /auth/login HTTP/1.1 , JSON (application/json)
10 0.000972  ::1  ::1  TCP       64 3000 → 54844 [ACK] Seq=1 Ack=338 Win=65024 Len=0
11 0.003665  ::1  ::1  TCP       64 3000 → 54843 [FIN, ACK] Seq=1 Ack=2 Win=65280 Len=0
12 0.003705  ::1  ::1  TCP       64 54843 → 3000 [ACK] Seq=2 Ack=2 Win=65280 Len=0
13 0.086925  ::1  ::1  HTTP/J…  372 HTTP/1.1 401 Unauthorized , JSON (application/json)
14 0.086964  ::1  ::1  TCP       64 54844 → 3000 [ACK] Seq=338 Ack=309 Win=65024 Len=0
15 1.080346  ::1  ::1  TCP       65 [TCP Keep-Alive] 54844 → 3000 [ACK] Seq=337 Ack=309 Win=65024 Len=1
16 1.080385  ::1  ::1  TCP       76 [TCP Keep-Alive ACK] 3000 → 54844 [ACK] Seq=309 Ack=338 Win=65024 Len=0 SLE=337 SRE=338
17 2.105887  ::1  ::1  TCP       65 [TCP Keep-Alive] 54844 → 3000 [ACK] Seq=337 Ack=309 Win=65024 Len=1
18 2.105914  ::1  ::1  TCP       76 [TCP Keep-Alive ACK] 3000 → 54844 [ACK] Seq=309 Ack=338 Win=65024 Len=0 SLE=337 SRE=338
19 3.107167  ::1  ::1  TCP       65 [TCP Keep-Alive] 54844 → 3000 [ACK] Seq=337 Ack=309 Win=65024 Len=1
20 3.107185  ::1  ::1  TCP       76 [TCP Keep-Alive ACK] 3000 → 54844 [ACK] Seq=309 Ack=338 Win=65024 Len=0 SLE=337 SRE=338
21 4.119397  ::1  ::1  TCP       65 [TCP Keep-Alive] 54844 → 3000 [ACK] Seq=337 Ack=309 Win=65024 Len=1
22 4.119436  ::1  ::1  TCP       76 [TCP Keep-Alive ACK] 3000 → 54844 [ACK] Seq=309 Ack=338 Win=65024 Len=0 SLE=337 SRE=338
23 5.133187  ::1  ::1  TCP       65 [TCP Keep-Alive] 54844 → 3000 [ACK] Seq=337 Ack=309 Win=65024 Len=1
24 5.133214  ::1  ::1  TCP       76 [TCP Keep-Alive ACK] 3000 → 54844 [ACK] Seq=309 Ack=338 Win=65024 Len=0 SLE=337 SRE=338
25 6.101620  ::1  ::1  TCP       64 3000 → 54844 [FIN, ACK] Seq=309 Ack=338 Win=65024 Len=0

> Frame 1: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
```

• Capture an invalid login attempt. What status code is returned?
  - The status code returned for an invalid login attempt is 401 Unauthorized. This indicates that the request was made with invalid credentials.

• How does the TCP stream differ for a 401 vs 200 response?
  - A 200 OK response includes a JSON payload with a JWT token or user data, indicating successful authentication.

- A 401 Unauthorized response generally has a smaller payload and often includes a message indicating authentication failure. Both may have similar TCP stream structures, but the content length and response body differ.

5. Performance

• Measure time between POST /login request and response.

- From Wireshark, the POST /login request starts at time 19.826420 and the response (200 OK) arrives at 19.827433. The time difference is approximately 1.013 milliseconds.

• What contributes to this latency?

- Factors include:
  - • Server processing time to validate credentials and generate a token.
  - • Network stack processing (even on localhost).
  - • Overhead from Postman runtime and JSON encoding/decoding.
  - • Any artificial delay in the backend API for testing or debugging.

3. **Security Recommendations**:

o 3 risks of unencrypted JWT transmission

o 2 ways to enhance API security

API Security Summary

3 Risks of Unencrypted JWT Transmission

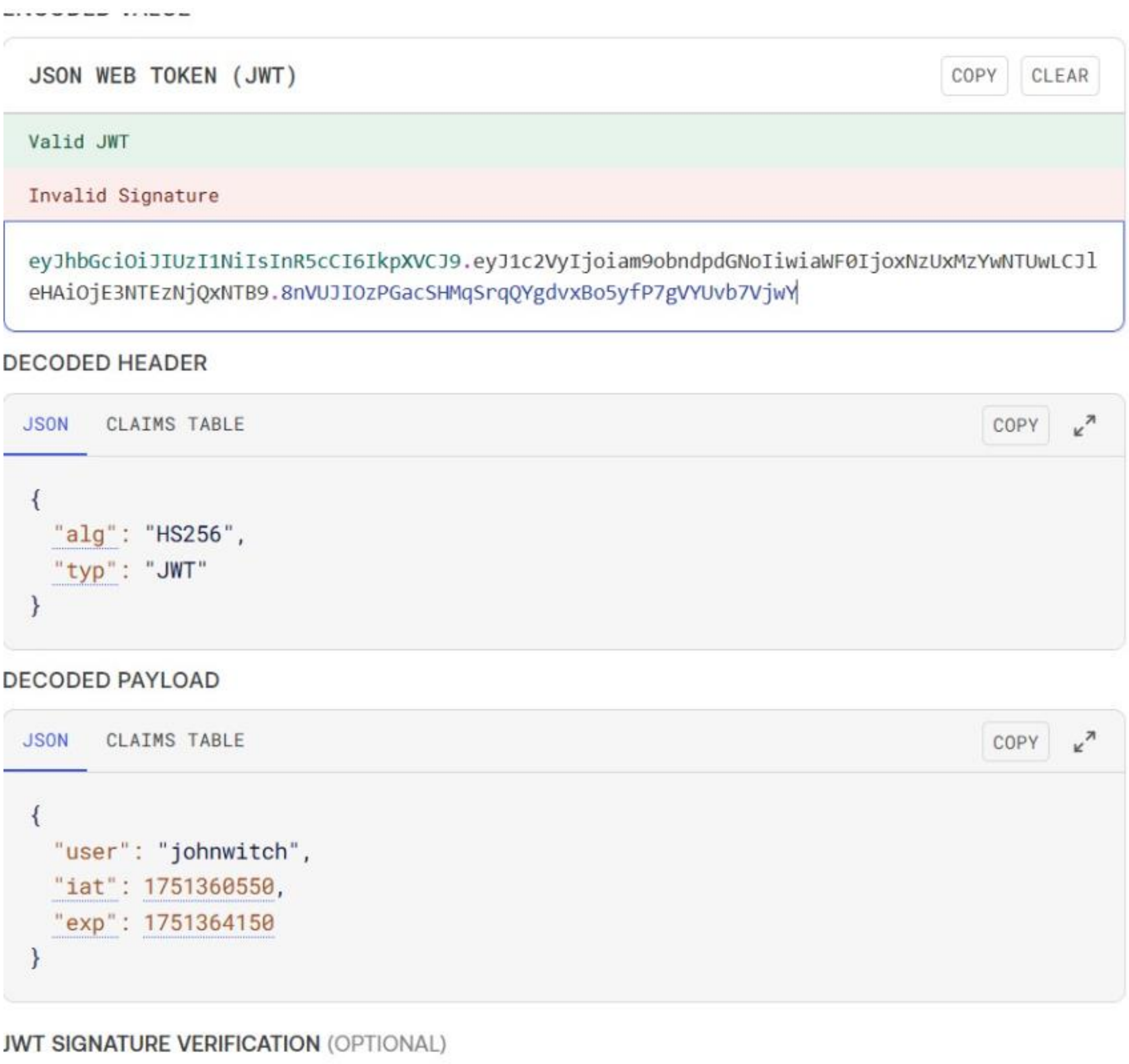
1. Token Theft via Network Sniffing:
   JWTs transmitted over HTTP can be intercepted using tools like Wireshark or tcpdump. Attackers can replay stolen tokens to impersonate users.
2. Sensitive Information Exposure:
   JWT payloads are only Base64-encoded, not encrypted. User data (e.g., usernames, roles, email) can be read in plain text.
3. Session Hijacking and Privilege Escalation:
   If an attacker modifies a JWT (e.g., changing "user": "user1" to "user": "admin") and the server fails to validate the signature properly, it can lead to unauthorized access.

2 Ways to Enhance API Security

4. Use HTTPS (TLS Encryption):
   Always enforce HTTPS to encrypt tokens in transit and prevent man-in-the-middle (MITM) attacks.

5. Validate JWT Signature and Expiry Strictly:
   Ensure the server verifies the JWT signature using a secure, secret key, and checks `exp` (expiry) and `iat` (issued-at) fields to reject old or tampered tokens.

Security Exercise – JWT Token Vulnerability



JSON WEB TOKEN (JWT)

Valid JWT

Invalid Signature

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoiam9obndpdGNoIiwiaWF0IjoxNzUxMzYwNTUwLCJl
eHAiOjE3NTEzNjQxNTB9.8nVUJIOzPGacSHMqSrqQYgdvxBo5yfP7gVYUvb7VjwY
```

DECODED HEADER

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

DECODED PAYLOAD

```
{
  "user": "johnwitch",
  "iat": 1751360550,
  "exp": 1751364150
}
```

JWT SIGNATURE VERIFICATION (OPTIONAL)

1. What user information is exposed in the JWT?
By capturing and decoding the JWT from Wireshark using jwt.io, the following payload (Base64-decoded) is revealed:

```
{
  "user": "johnwitch",
  "iat": 1751360550,
  "exp": 1751364150
}
```

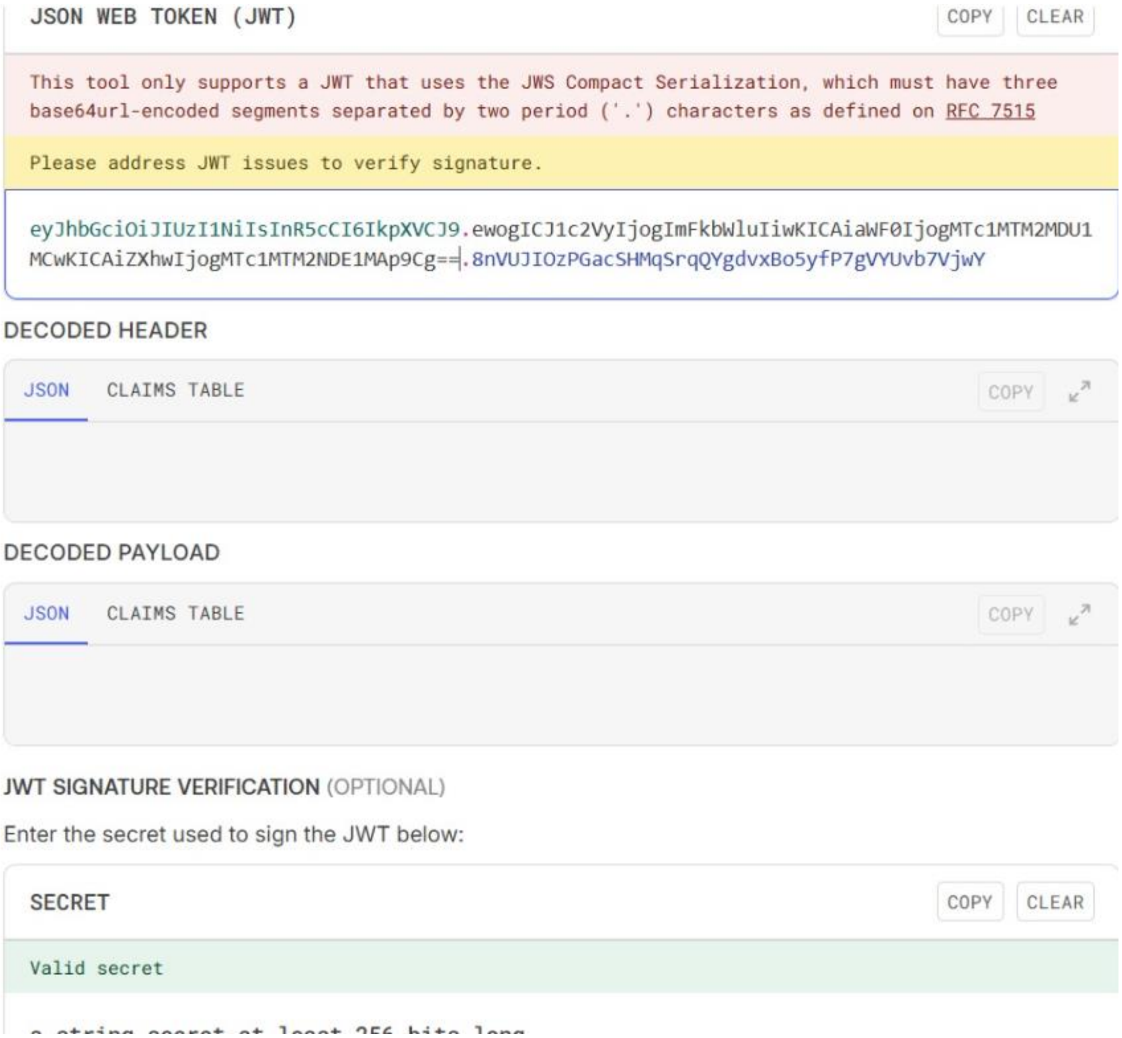
This means the JWT exposes:
- The username: johnwitch
- The issued-at time (iat) and expiry time (exp) in Unix timestamp format

Note: JWTs are only Base64-encoded, not encrypted, so sensitive user information can be exposed if traffic is not protected by HTTPS.

2. What happens when the JWT is modified (e.g., change user to admin)?
When attempting to modify the payload, for example:

```
{
  "user": "admin",
  "iat": 1751360550,
  "exp": 1751364150
```

}



This tool only supports a JWT that uses the JWS Compact Serialization, which must have three base64url-encoded segments separated by two period ('.') characters as defined on RFC 7515

Please address JWT issues to verify signature.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ewogICJ1c2VyIjogImFkbWluIiwKICAiaWF0IjogMTc1MTM2MDU1
MCwKICAiZXhwIjogMTc1MTM2NDE1MAp9Cg==.8nVUJIOzPGacSHMqSrqQYgdvxBo5yfP7gVYUvb7VjwY
```

Enter the secret used to sign the JWT below:

Valid secret

a string secret at least 256 bits long

...and re-encode the token, the following occurs:
- The signature becomes invalid, because the new payload doesn't match the original HMAC signature generated by the server's secret key.
- As a result, the server rejects the request with a 401 Unauthorized or similar response.

3. Why doesn't this work in production?
Because:
- JWT tokens are signed with a secret key known only to the server.
- When the payload is tampered with, the signature no longer matches.
- Production servers verify the signature before trusting the payload.

This mechanism ensures data integrity and prevents unauthorized access, even if the token is intercepted.

4. Why is this dangerous without HTTPS?

- Without HTTPS, JWTs are transmitted in plaintext, making them visible in tools like Wireshark.
- Attackers can capture, replay, or attempt to modify tokens.
- Even though tampering is blocked (due to signature verification), token theft (token hijacking) remains a risk.

5. How would HTTPS change what you see in Wireshark?

- With HTTPS, the entire HTTP traffic (including JWTs in headers or body) is encrypted.
- Wireshark would not be able to read the contents of the token or any user information.
- You would see encrypted TLS packets, not readable JSON.