JWT Attacks

Author: Rana Khalil (𝕏 @ranakhalilacad)

Agenda

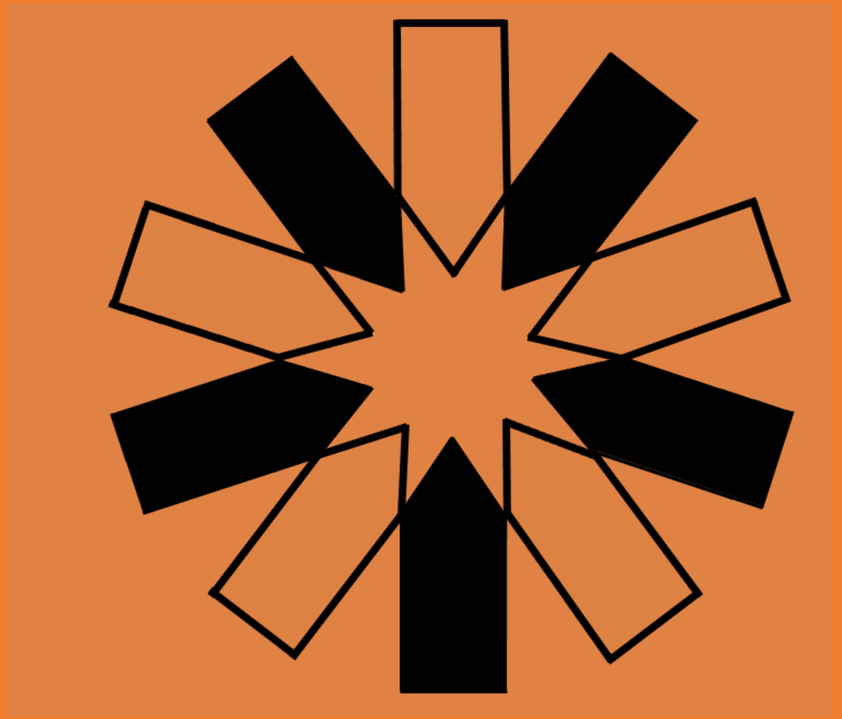
WHAT ARE JWTS?

WHAT ARE JWT VULNERABILITIES?

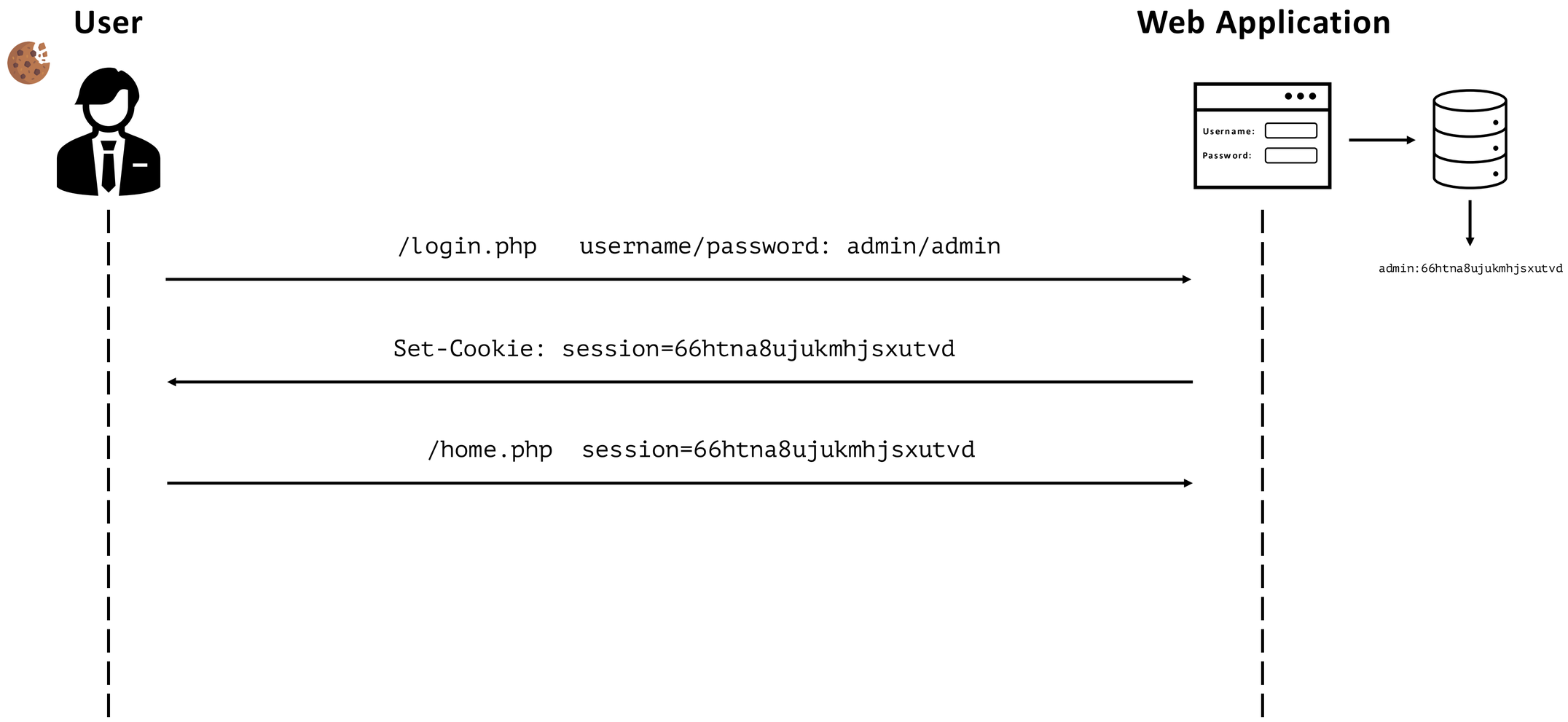
HOW DO YOU FIND AND EXPLOIT THEM?

HOW DO YOU PREVENT THEM?

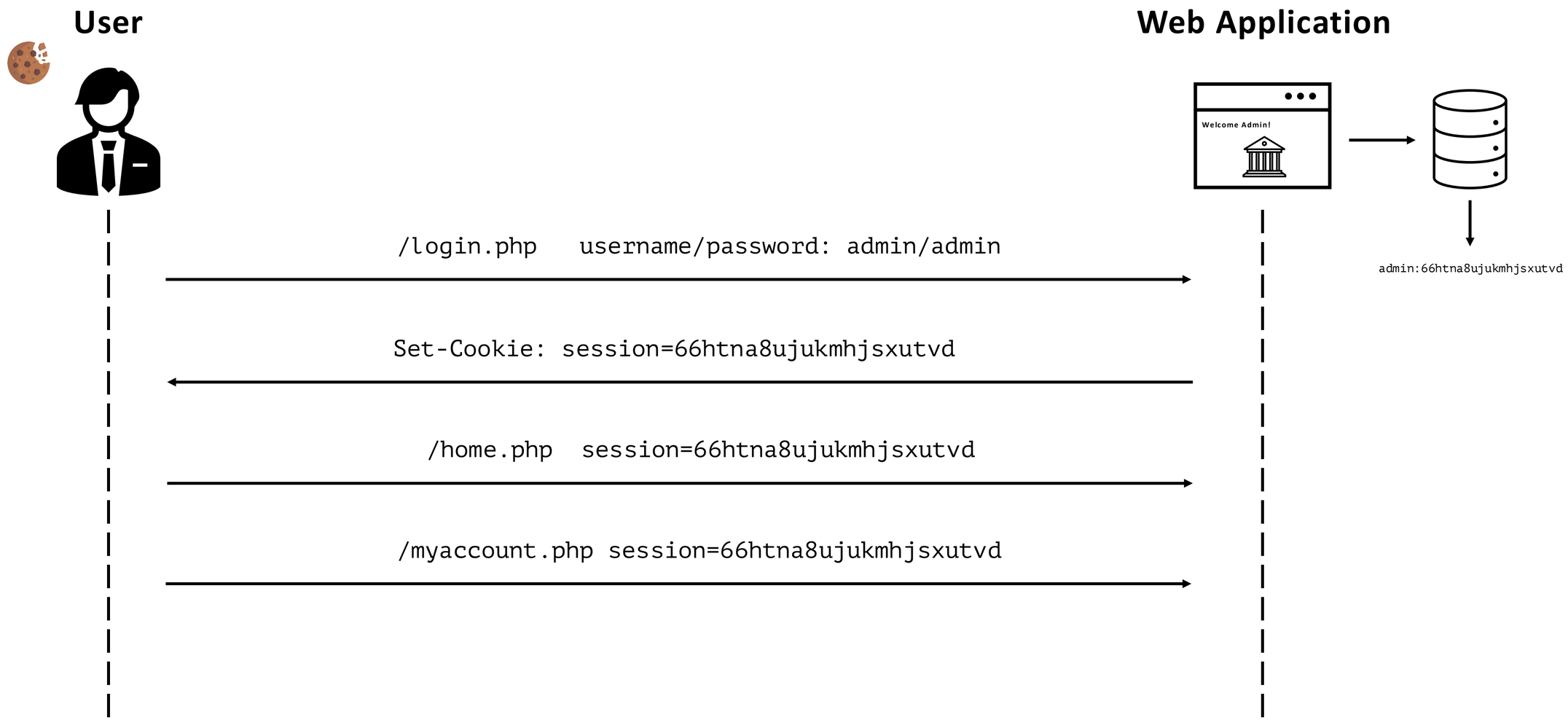
WHAT ARE JWTS?



Sessions

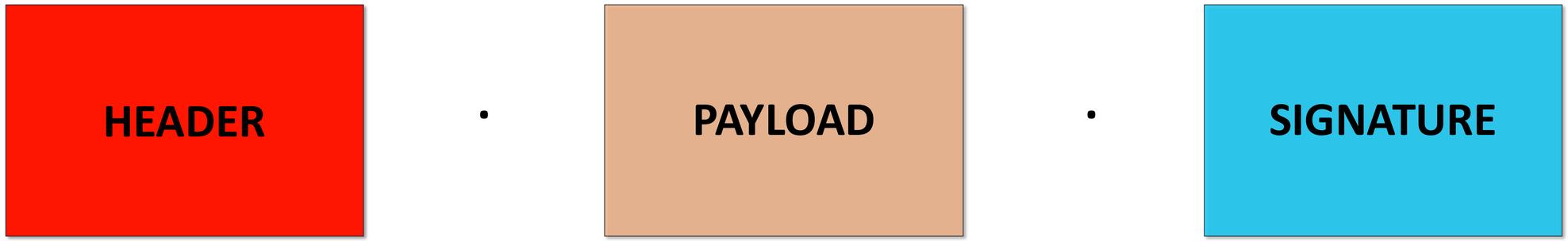

User
Web Application
Username:
Password:
admin:66htna8ujukmhjsxutvd
/login.php username/password: admin/admin
Set-Cookie: session=66htna8ujukmhjsxutvd
/home.php session=66htna8ujukmhjsxutvd

Sessions


User
Web Application
Welcome Admin!
admin:66htna8ujukmhjsxutvd
/login.php username/password: admin/admin
Set-Cookie: session=66htna8ujukmhjsxutvd
/home.php session=66htna8ujukmhjsxutvd
/myaccount.php session=66htna8ujukmhjsxutvd

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
3ZS1lYWRmNWE0NGM4YjUiLCJhbGciOiJSUzI1Ni
J9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsImV4cCI6
MTY0ODAzNzE2NCwibmFtZSI6IkNhcmxvcyBNb25
0b3lhIiwic3ViIjoiY2FybG9zIiwicm9sZSI6Im
Jsb2dfYXV0aG9yIiwiZW1haWwiOiJjYXJsb3NAY
2FybG9zLW1vbnRveWEubmV0IiwiaWF0IjoxNTE2
MjM5MDIyfQ.SYZBPIBg2CRjXAJ8vCER0LA_ENjI
I1JakvNQoP-
Hw6GG1zfl4JyngsZReIfqRvIAEi5L4HV0q7_9qG
hQZvy9ZdxEJbwTxRs_6Lb-
fZTDpW6lKYNdMyjw45_alSCZ1fypsMWz_2mTpQz
il0lOtps5Ei_z7mM7M8gCwe_AGpI53JxduQOaB5
HkT5gVrv9cKu9CsW5MS6ZbqYXpGyOG5ehoxqm8D
L5tFYaW3lB50ELxi0KsuTKEbD0t5BCl0aCR2MBJ
WAbN-
xeLwEenaqBiwPVvKixYleeDQiBEIylFdNNIMviK
RgXiYuAvMziVPbwSgkZVHeEdF5MQP1Oe2Spac-
6IfA
```


DECODED JWT

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "kid": "9136ddb3-cb0a-4a19-a07e-eadf5a44c8b5",
  "alg": "RS256"
}
```


PAYLOAD: DATA


```
  "exp": 1648037164,
  "name": "Carlos Montoya",
  "sub": "carlos",
  "role": "blog_author",
  "email": "carlos@carlos-montoya.net",
  "iat": 1516239022
}
```


```
RSASHA256( base64UrlEncode(header) + "." +
base64UrlEncode(payload), secret)
```

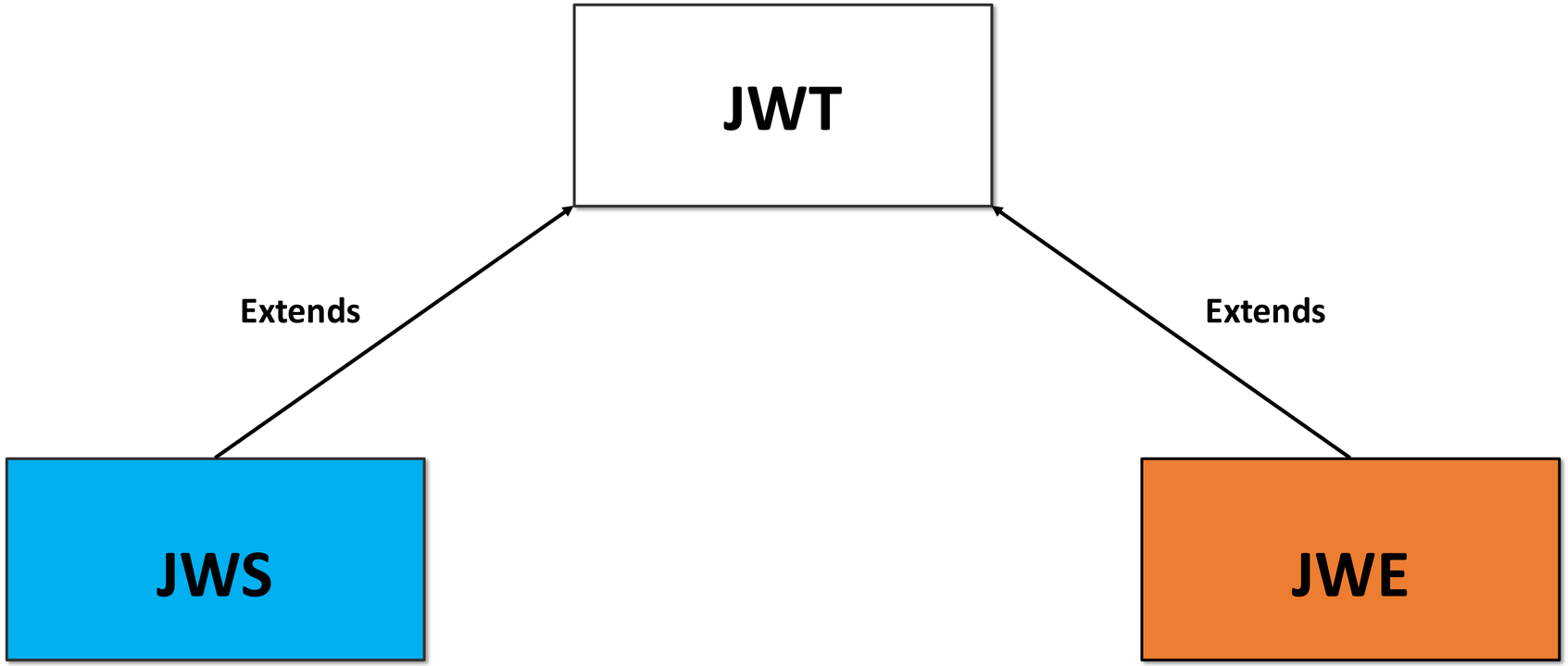
Sessions


User
Web Application
Username:
Password:
/login.php username/password: admin/admin
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9....
/home.php Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9....

Sessions


User
Web Application
Welcome Admin!
/login.php username/password: admin/admin
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9....
/home.php Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9....

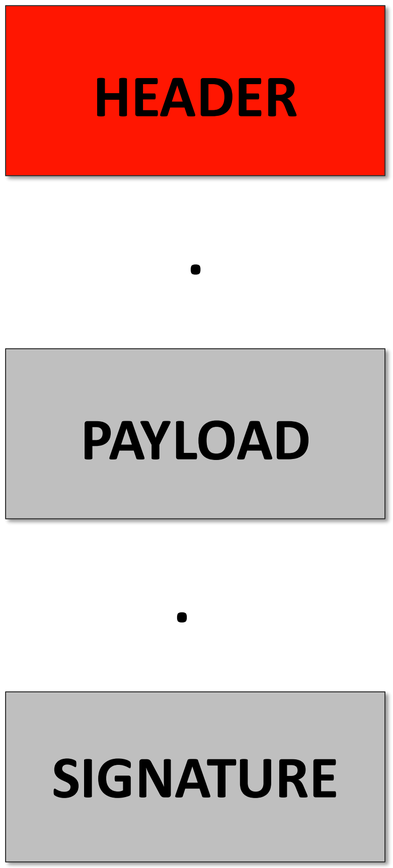
JWT vs JWS vs JWE



JWT Components

The first component is the header which contains metadata.

- `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.



JWT Components

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

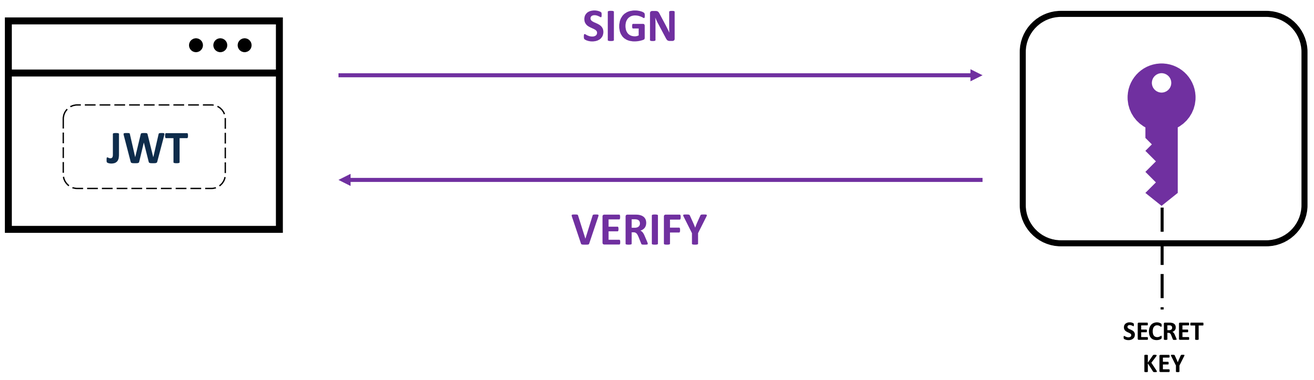
- `iss` — Identifies the issuer of the JWT.
- `sub` — Identifies the subject of the JWT.
- `aud` — Identifies the audience that the JWT is intended for.
- `exp` — A timestamp (in UNIX time format) indicating when the JWT expires and should no longer be accepted.
- `iat` — A timestamp indicating when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

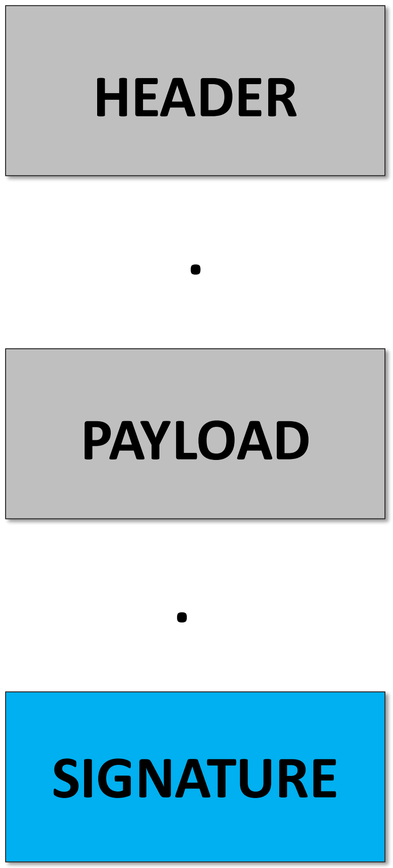
JWT Components

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.



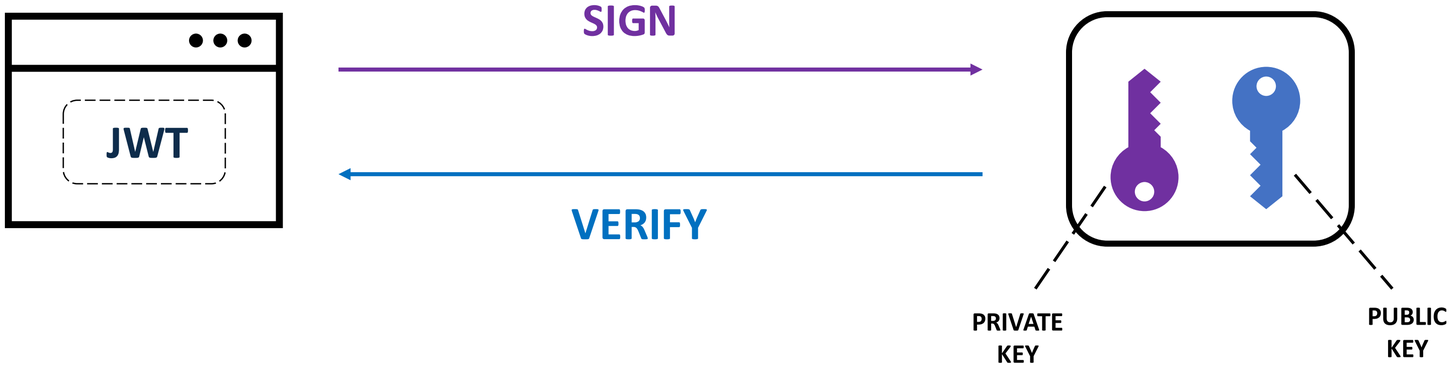


JWT Components

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.



HEADER

.

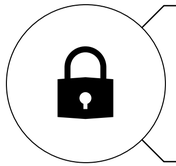
PAYLOAD

.

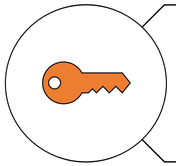
SIGNATURE

WHAT ARE JWT VULNERABILITIES?

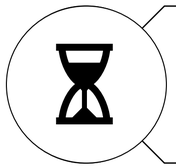
What Causes JWT Vulnerabilities?



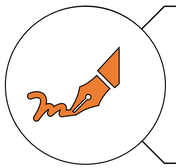
Weak or misconfigured cryptographic algorithms.



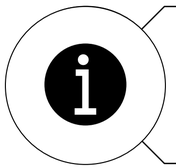
Poor key management.



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.

Source: https://github.com/advisories/GHSA-qwph-4952-7xr6

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 r
using a key that is embedded within the token. The vulnerability is due to node-jose following the JSON Web Signature (JWS
JSON Web Tokens (JWTs). This standard specifies that a JSON Web Key (JWK) representing a public key can be embedded w
of a JWS. This public key is then trusted for verification. An attacker could exploit this by forging valid JWS objects by remov
signature, adding a new public key to the header, and then signing the object using the (attacker-owned) private key associa
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 al

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": "n4EPtAOCc9AlkeQHPzHStgAbgs7bTZLwUBZdR8_KuKPEHLd4rHVTeT-O-XV2jRojdNhxJWTDvNd7nqQ0
       "e": "AQAB"
     }
   },
   "protected_b64u": "eyJhbGciOiJQUzI1NiIsImp3ayI6eyJrdHkiOiJSU0EiLCJraWQiOiJiaWxiby5iYWdnaW5
   "sig-input": "eyJhbGciOiJQUzI1NiIsImp3ayI6eyJrdHkiOiJSU0EiLCJraWQiOiJiaWxiby5iYWdnaW5zQGhv
   "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":"n4EPtAOCc9Alke..........rNP5zw",
          "e":"AQAB"
          }
```

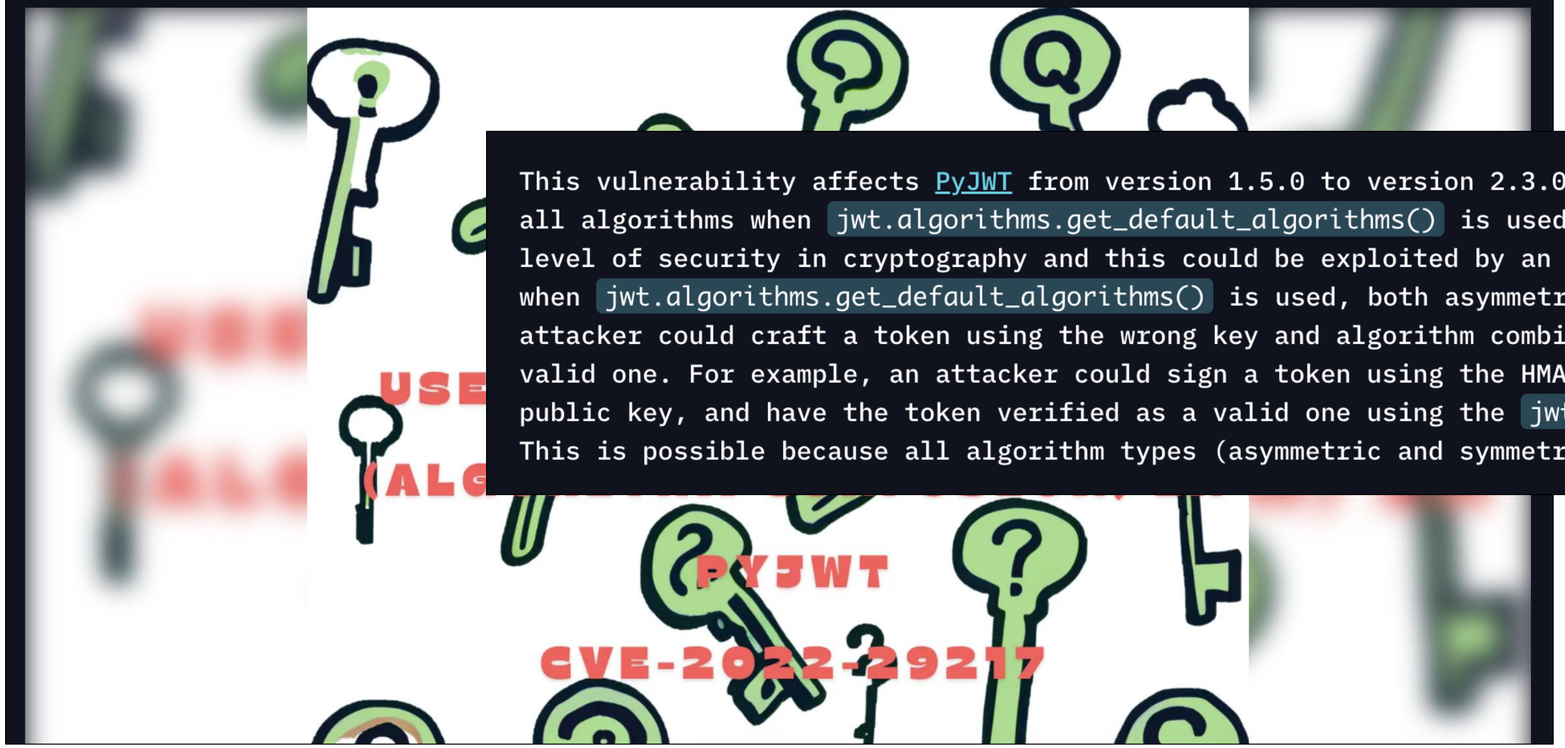
Sources: https://nvd.nist.gov/vuln/detail/CVE-2018-0114 & https://github.com/j4k0m/CVE-2018-0114?tab=readme-ov-file

PyJWT Algorithm Confusion

by @Emmharnuhe rl 02 Aug 2024 0 1 0 191

Risky algorithms (algorithm confusion) in PyJWT (CVE-2022-29217)

Screenshots from the blog posts

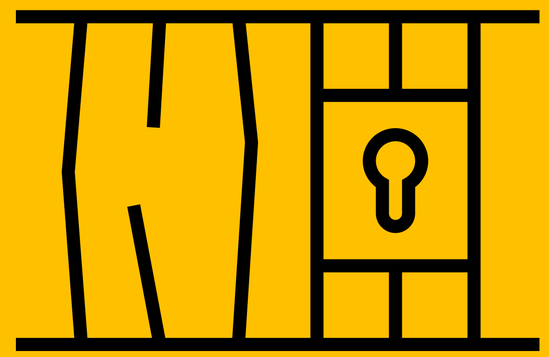


This vulnerability affects PyJWT from version 1.5.0 to version 2.3.0. These vulnerable versions of PyJWT support all algorithms when `jwt.algorithms.get_default_algorithms()` is used during decoding. Algorithms determine the level of security in cryptography and this could be exploited by an attacker. Since all algorithms are supported when `jwt.algorithms.get_default_algorithms()` is used, both asymmetric and symmetric algorithms are supported. An attacker could craft a token using the wrong key and algorithm combination and have the JWT checker verify it as a valid one. For example, an attacker could sign a token using the HMAC algorithm; a symmetric algorithm with a public key, and have the token verified as a valid one using the `jwt.algorithms.get_default_algorithms()` call. This is possible because all algorithm types (asymmetric and symmetric) are supported using that call.

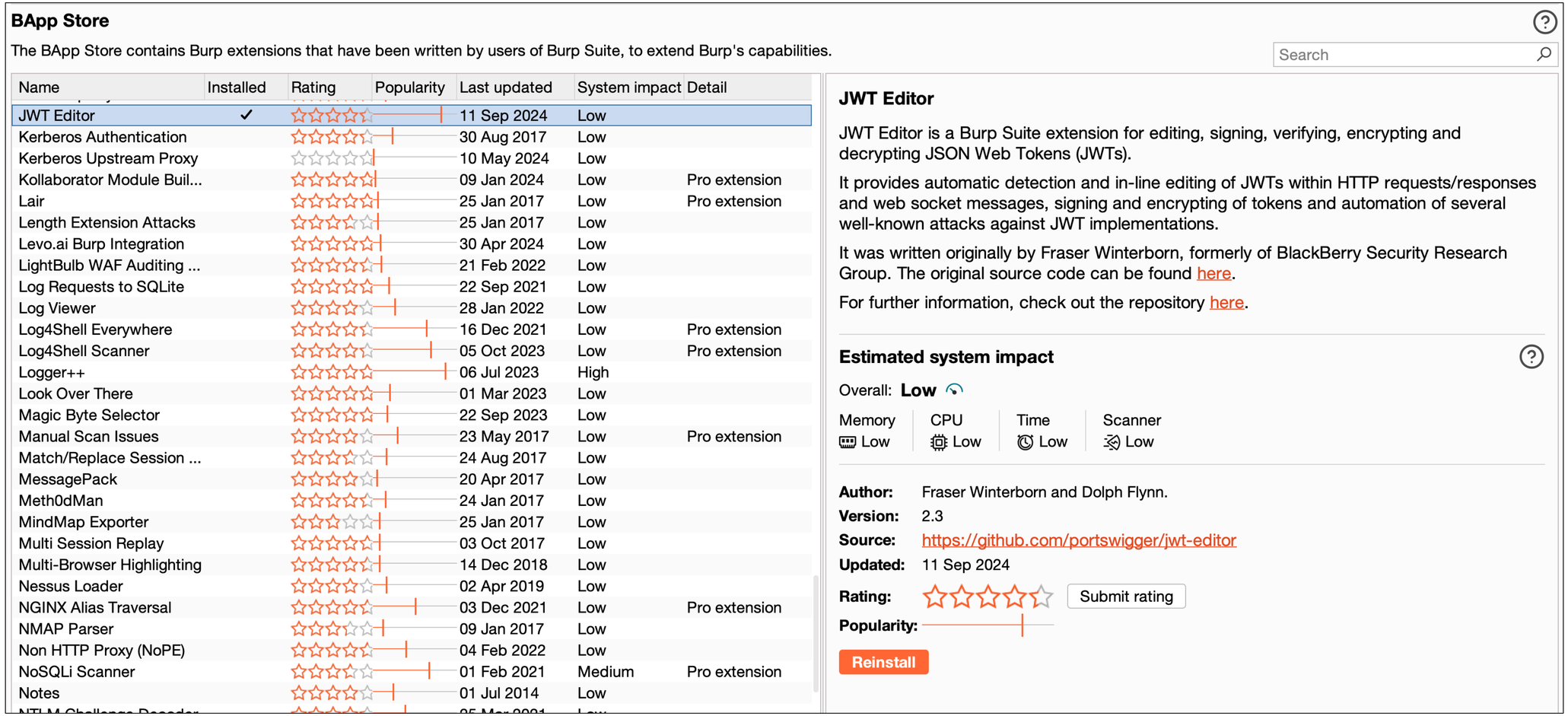
Source: https://www.vicarius.io/vsociety/posts/risky-algorithms-algorithm-confusion-in-pyjwt-cve-2022-29217

Author: Rana Khalil (@ranakhalilacad)

HOW TO FIND & EXPLOIT JWT VULNERABILITIES?



JWT Editor Burp Extension

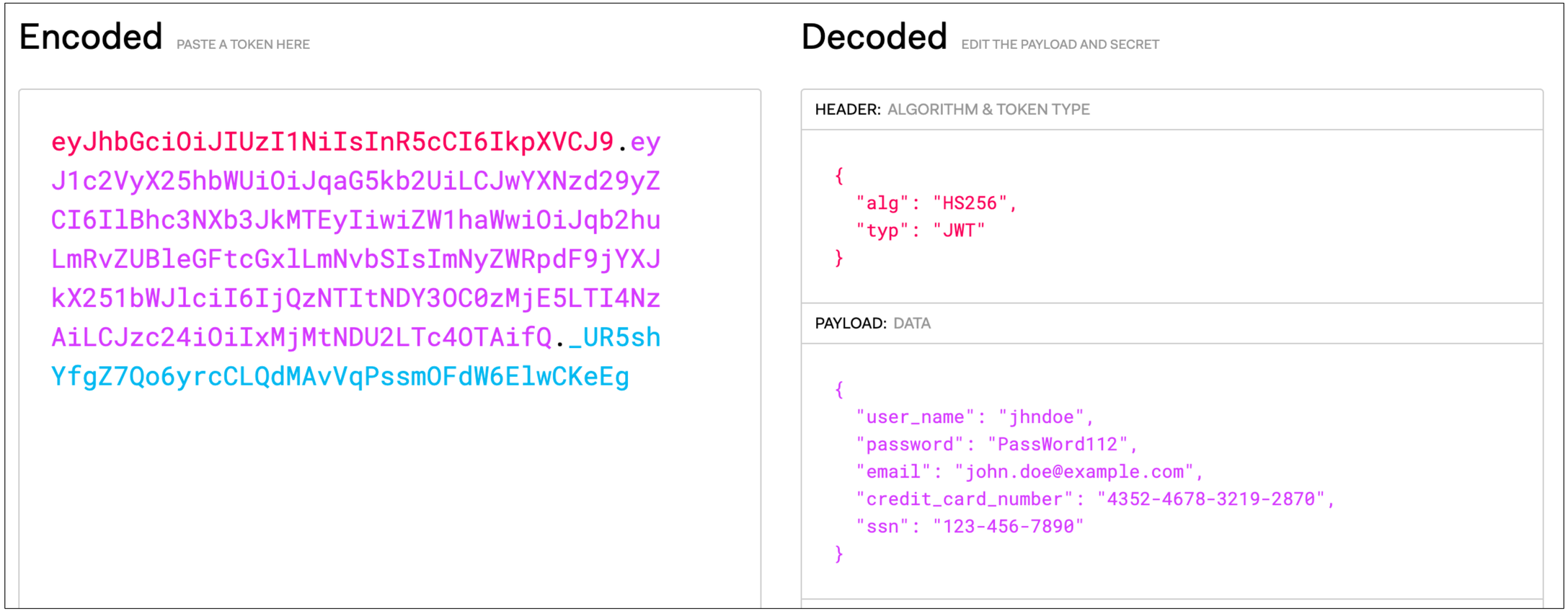


JWT Vulnerabilities

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

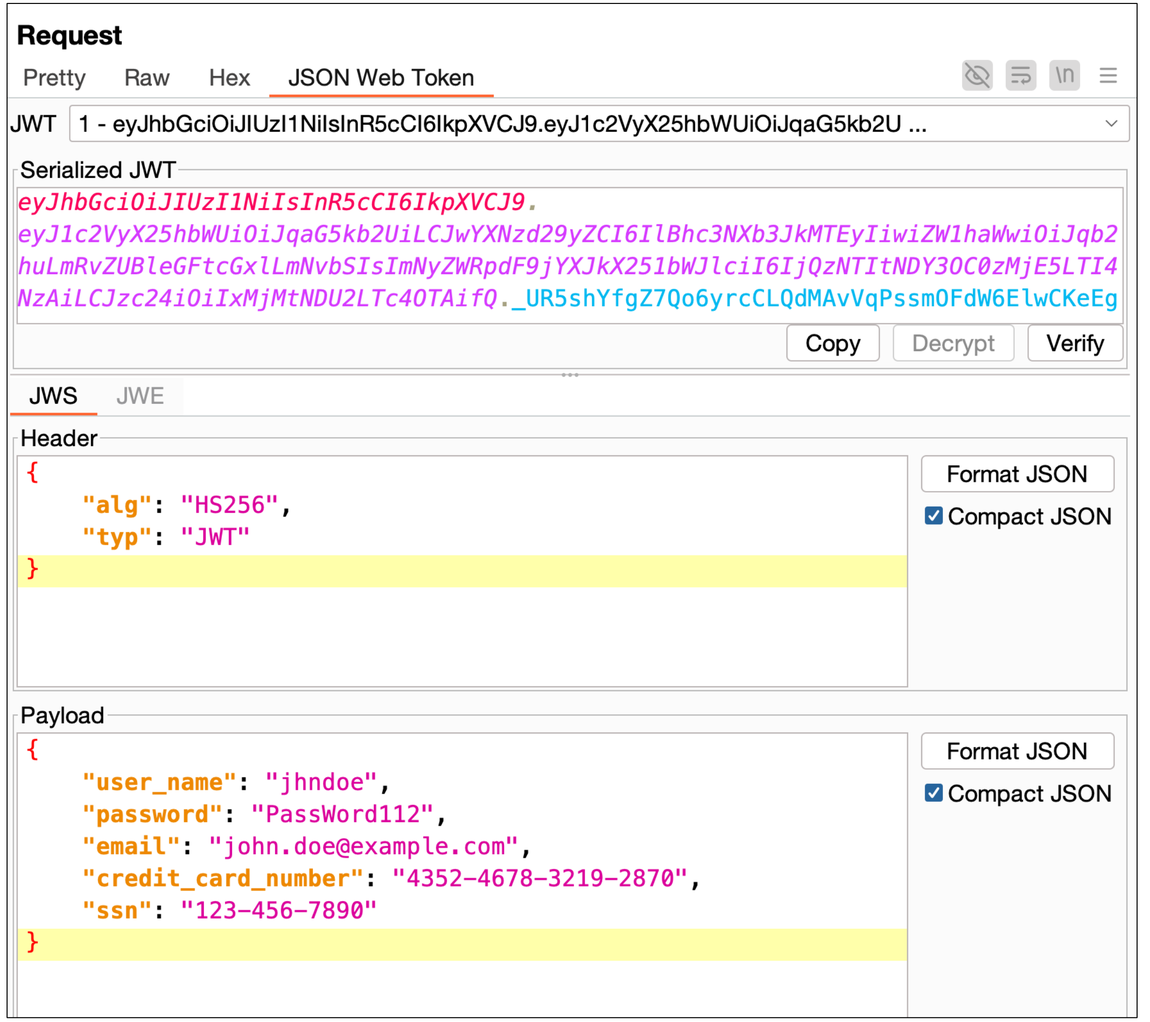
Sensitive information (like passwords, credit card numbers, SSNs, etc.) are included in JWT.



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.



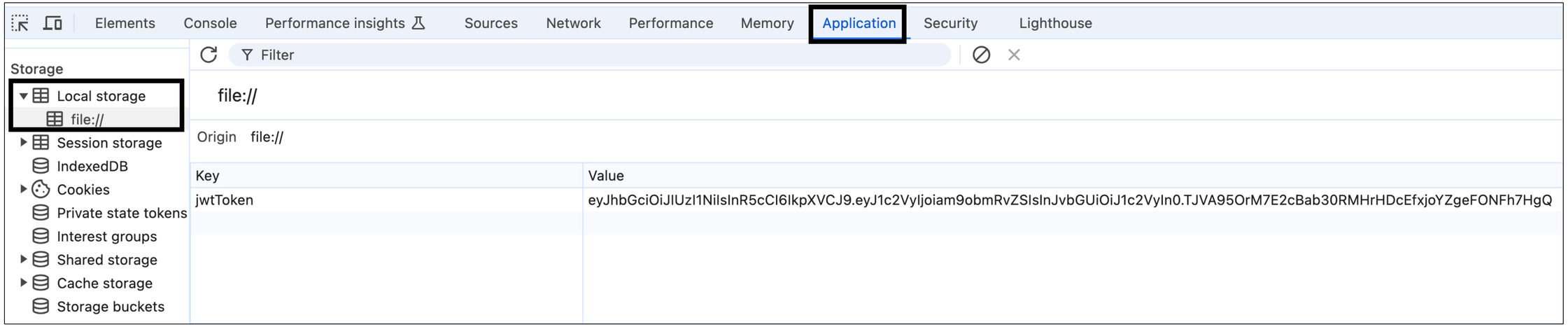
V2 - JWT Stored in Insecure Location

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;
```

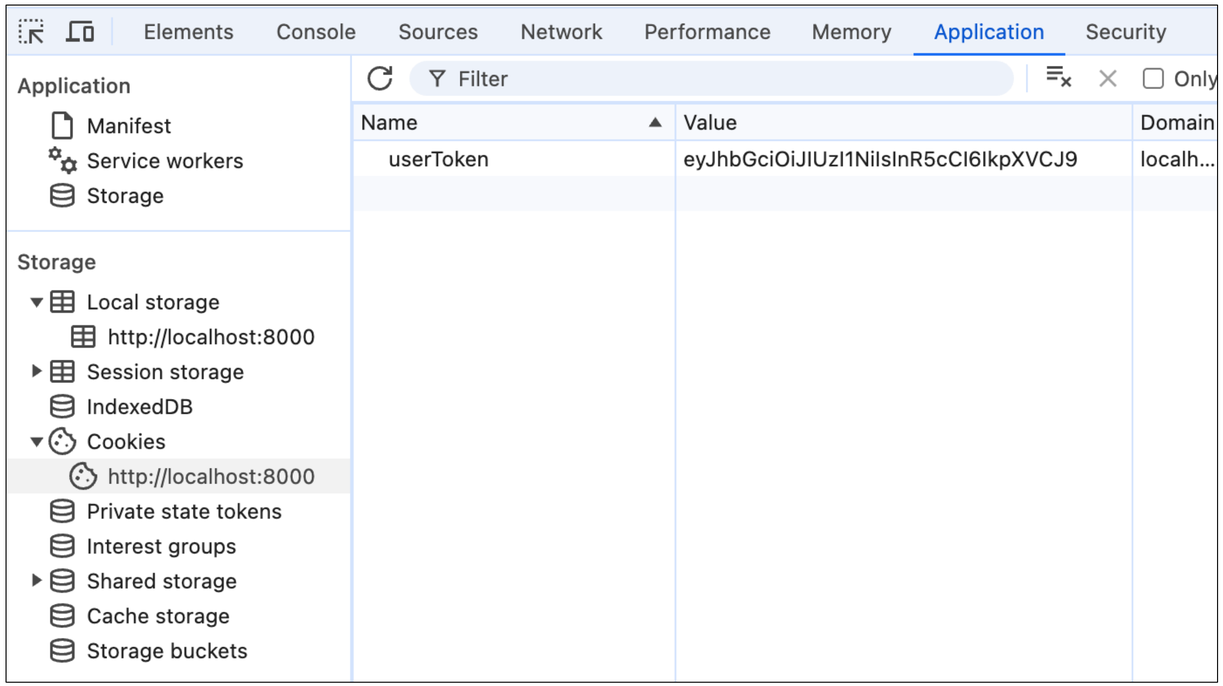
JWT Stored in Local Storage



V2 - JWT Stored in Insecure Location

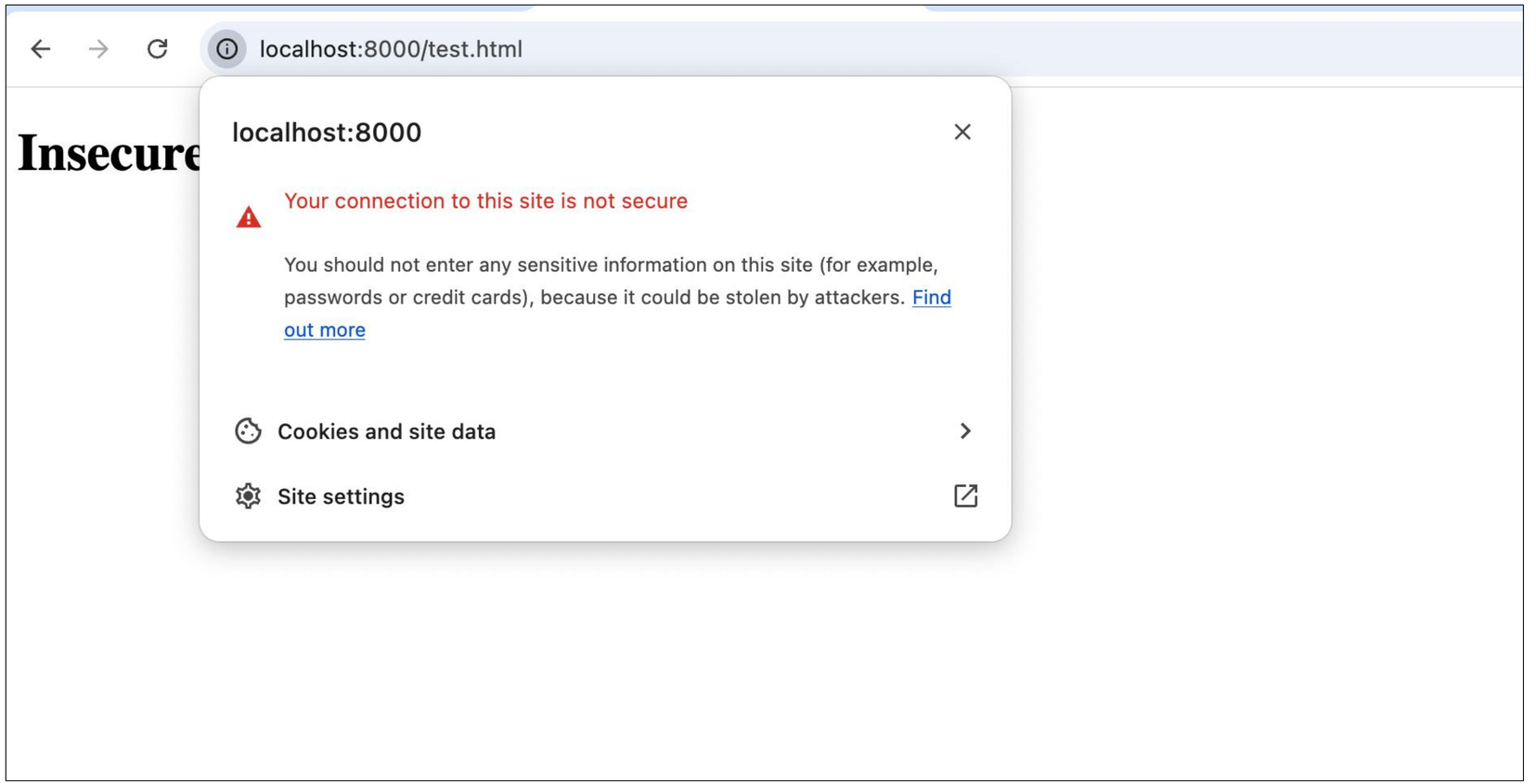
Steps to Find & Exploit:

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**.



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

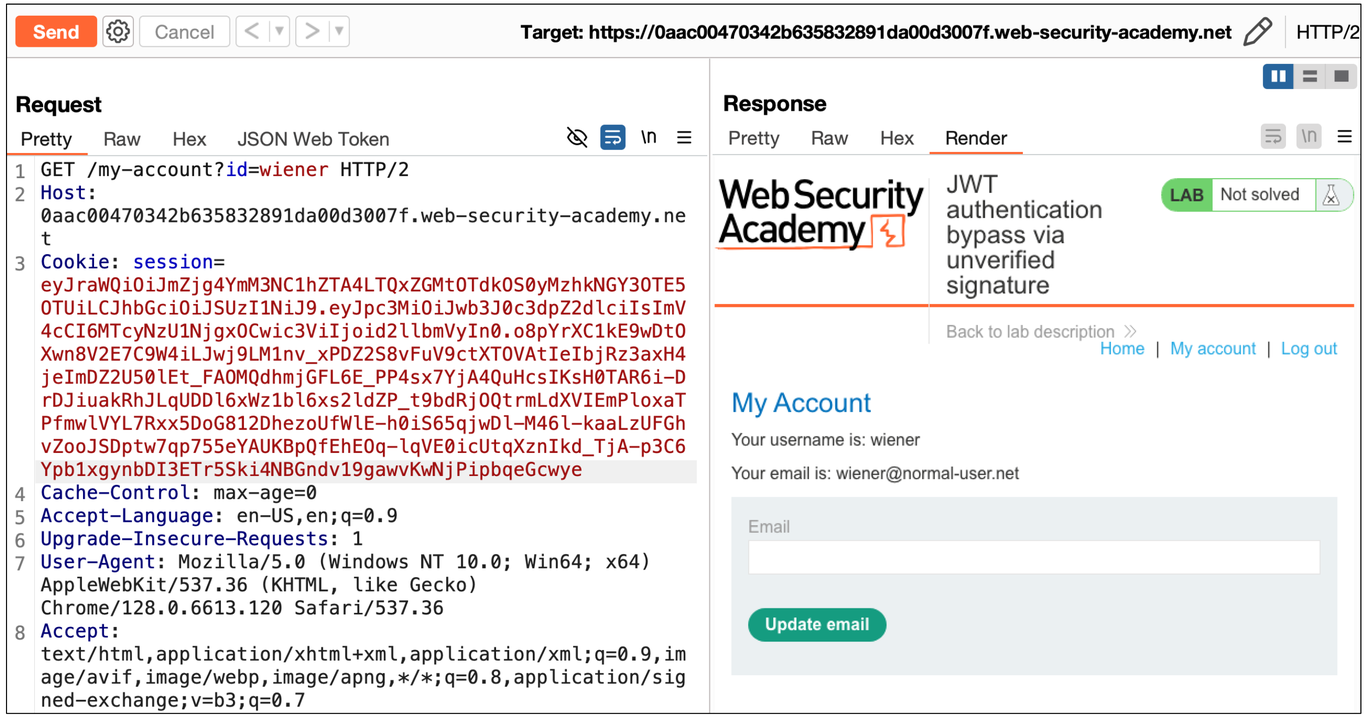
The application does not verify the signature of the JWT / accepts arbitrary signatures. This is sometimes possible because developers confuse the verify() and decode() methods and only pass incoming tokens to the decode function.

```
{
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

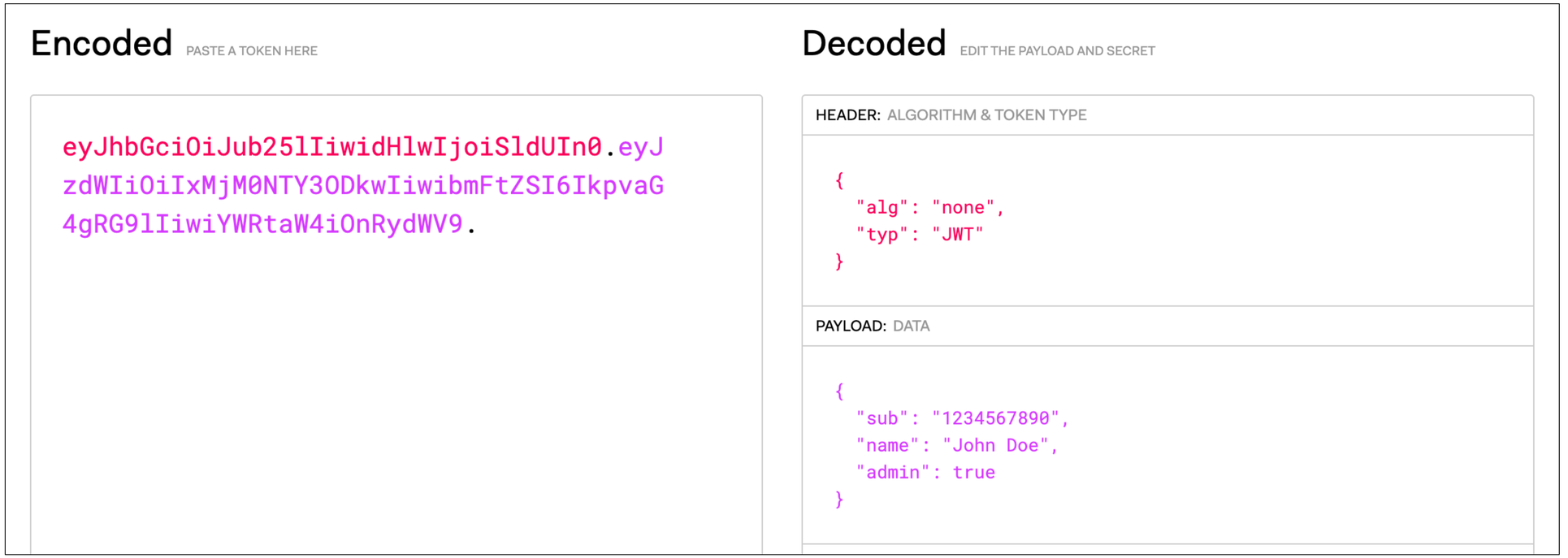
Steps to Find & Exploit:

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**.



V7 – None Algorithm Accepted

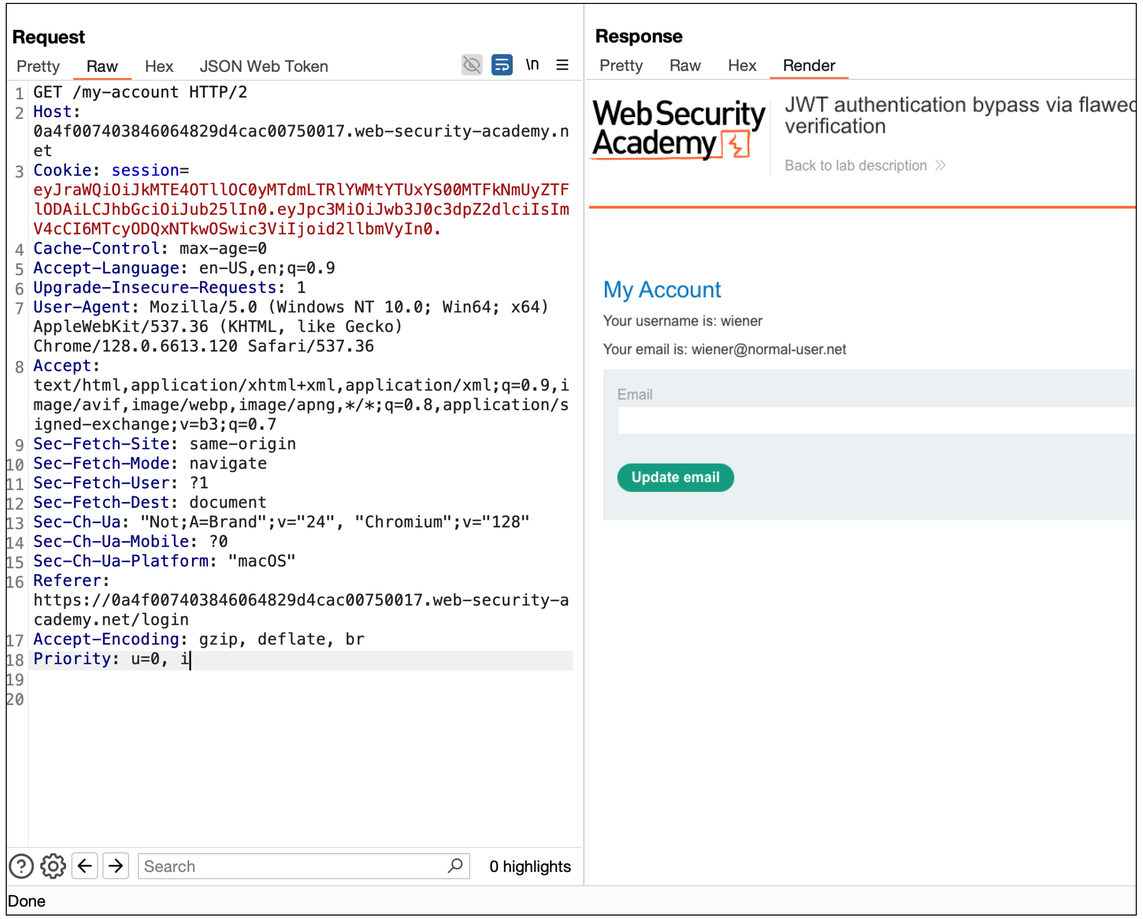
The application accepts a token with the “none” algorithm and does not properly verify the algorithm specified in the JWT header.



V7 – None Algorithm Accepted

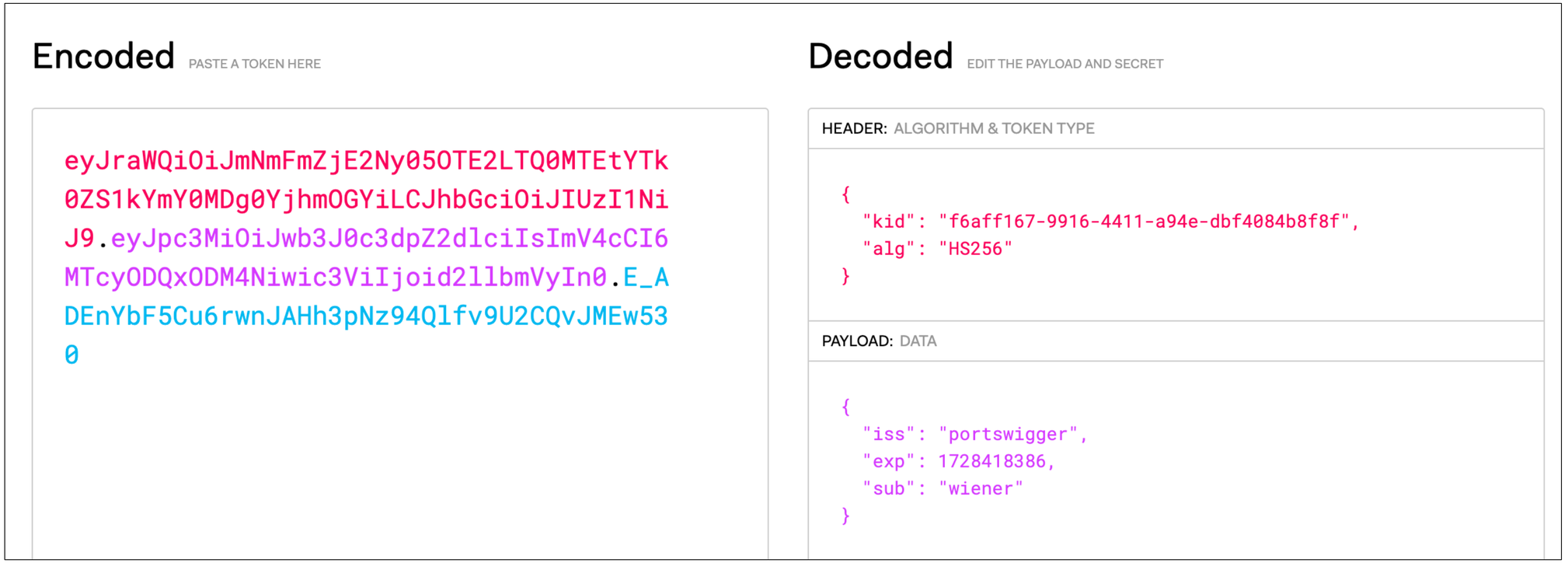
Steps to Find & Exploit:

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**.



V8 – Weak / Brute-forceable Signing Key

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



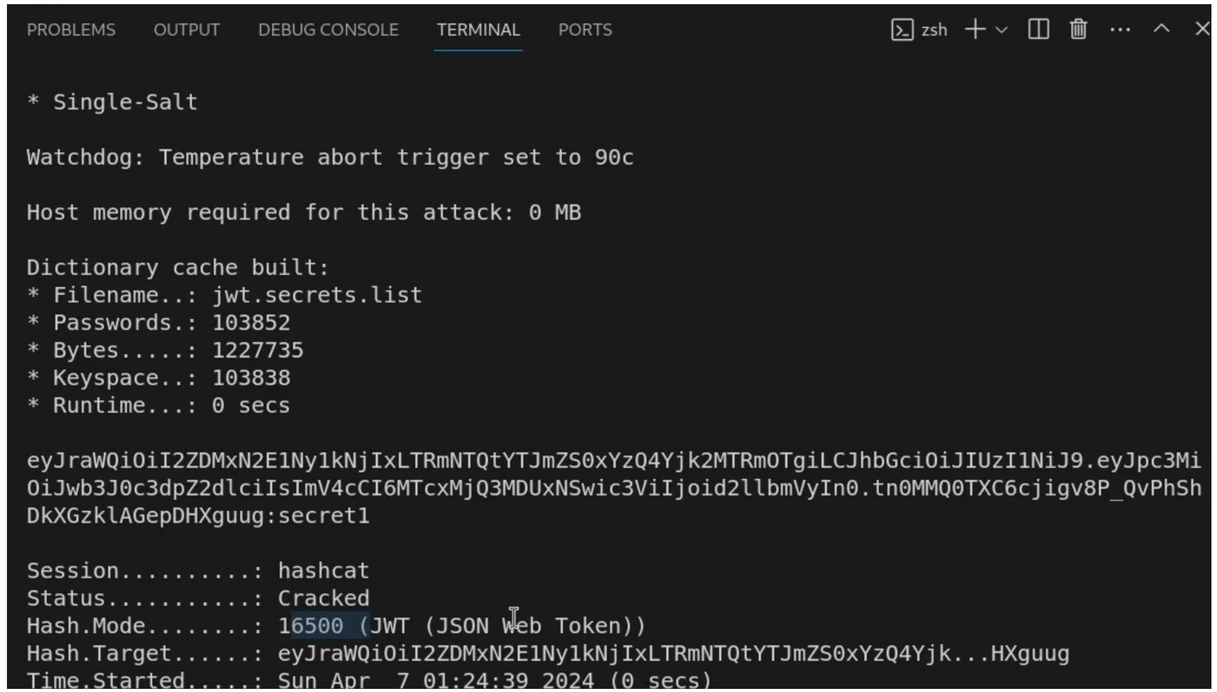
V8 – Weak / Brute-forceable Signing Key

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
```

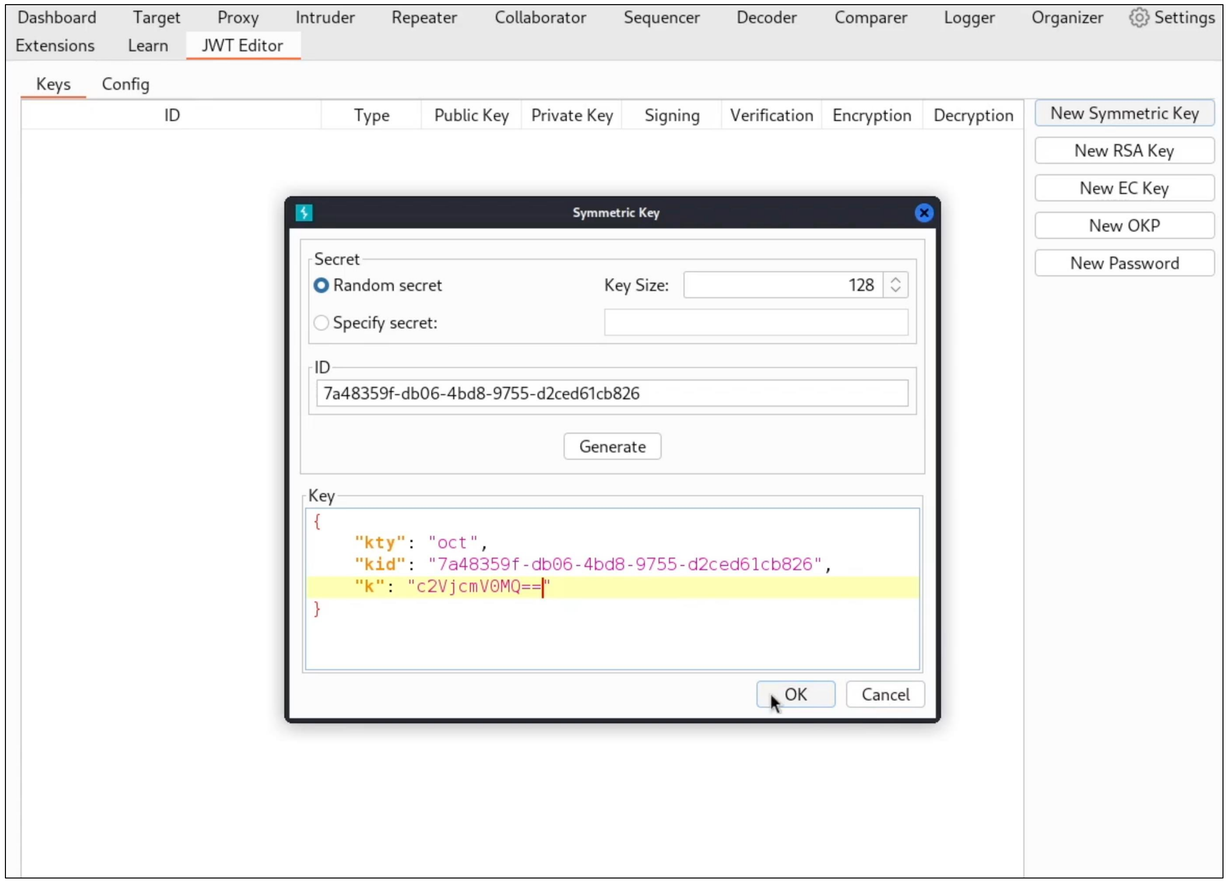


V8 – Weak / Brute-forceable Signing Key

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.

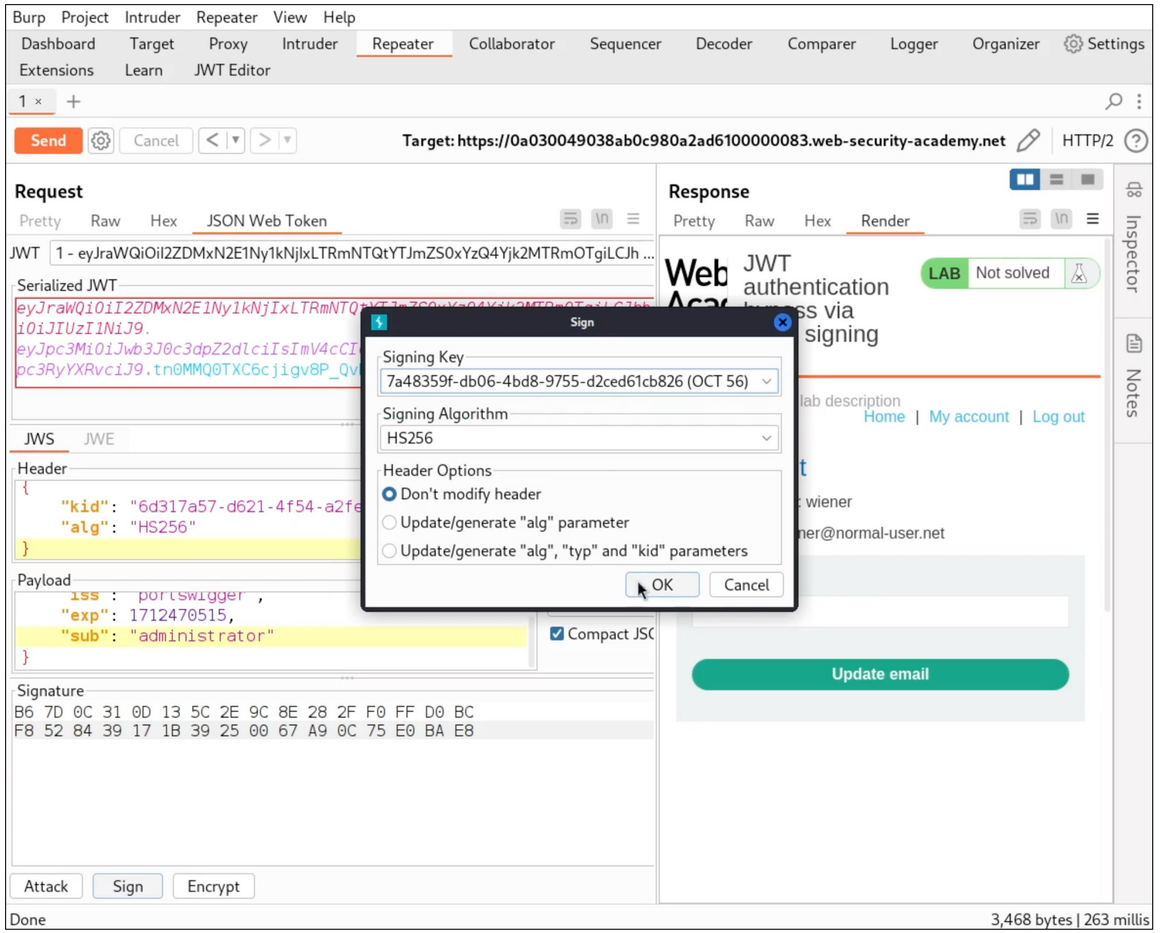


V8 – Weak / Brute-forceable Signing Key

Steps to Find & Exploit:

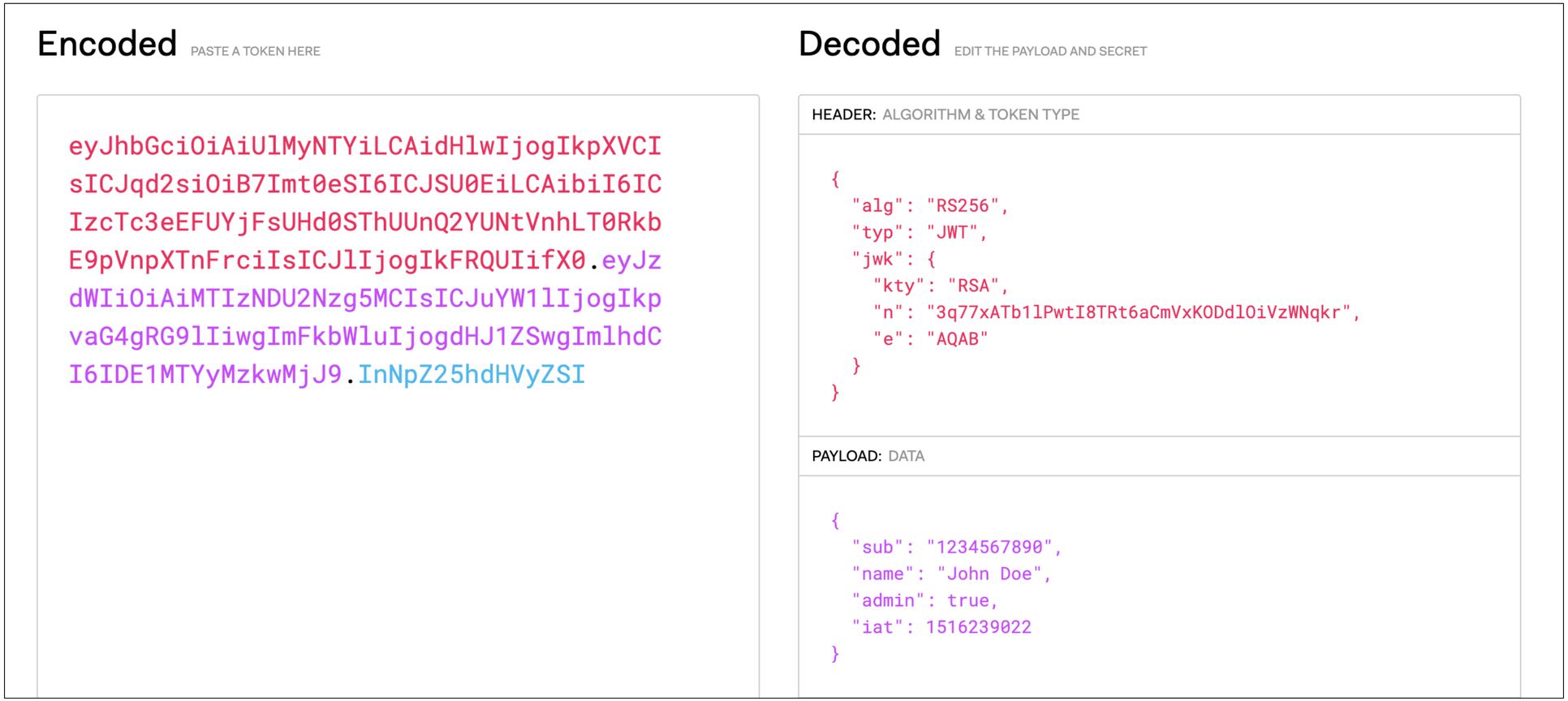
Part 3: Modify and sign the JWT

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

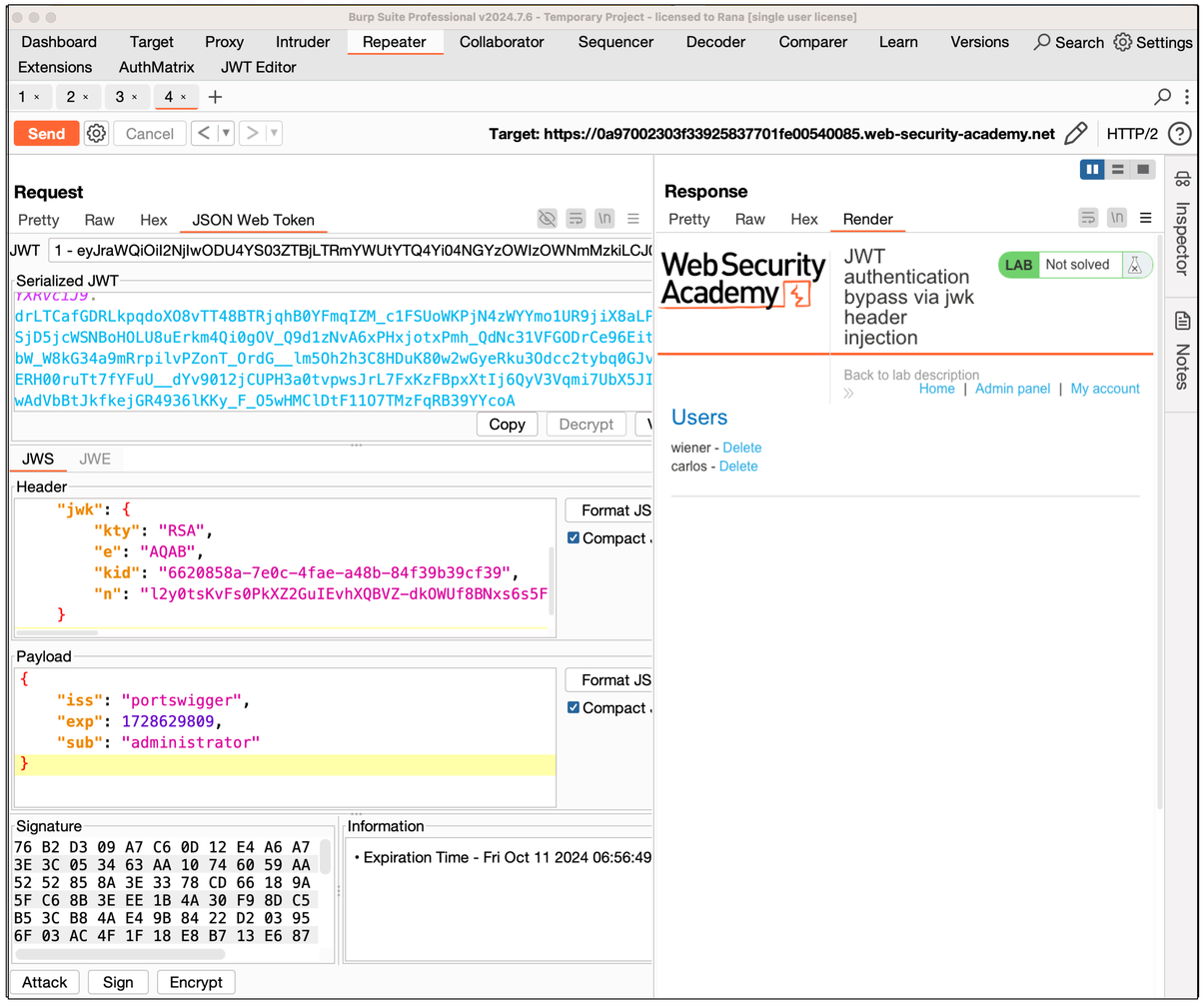
The application trusts any key that is embedded in the jwk header.



V9 – Header Injection via the jwk Parameter

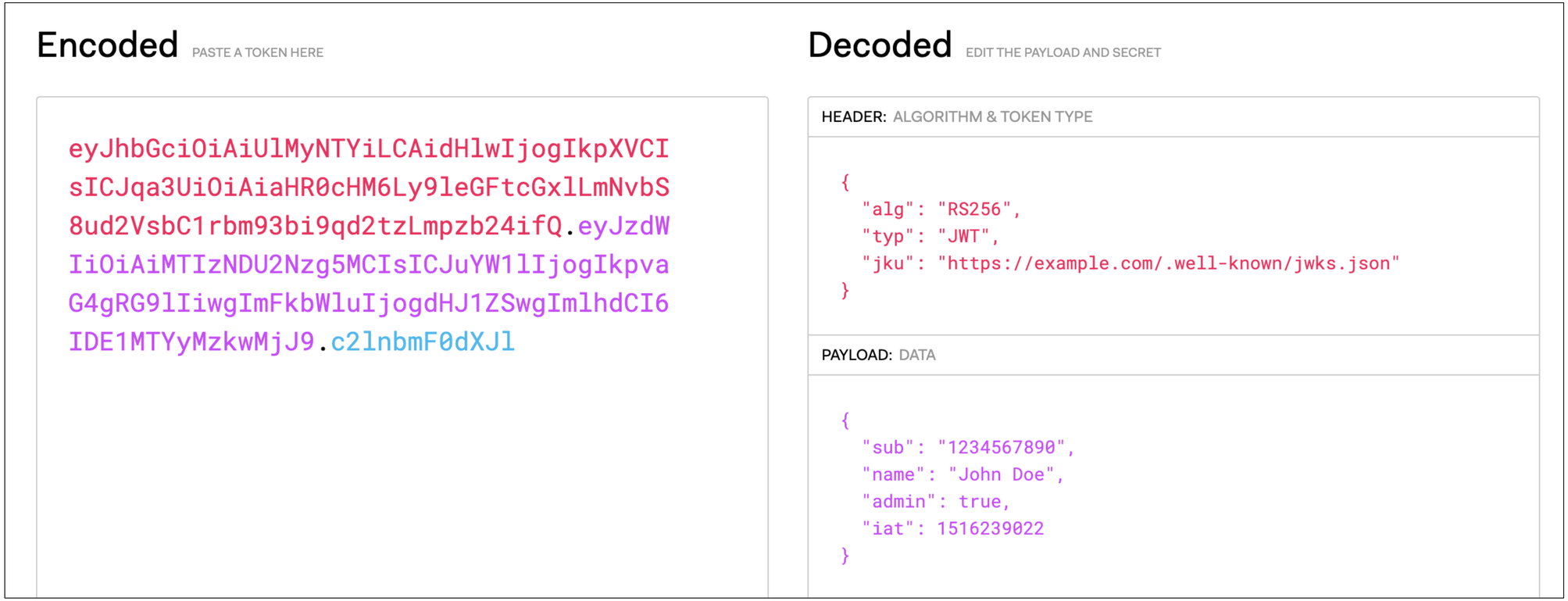
Steps to Find & Exploit:

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.



V10 – Header Injection via the jku Parameter

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



V10 – Header Injection via the jku Parameter

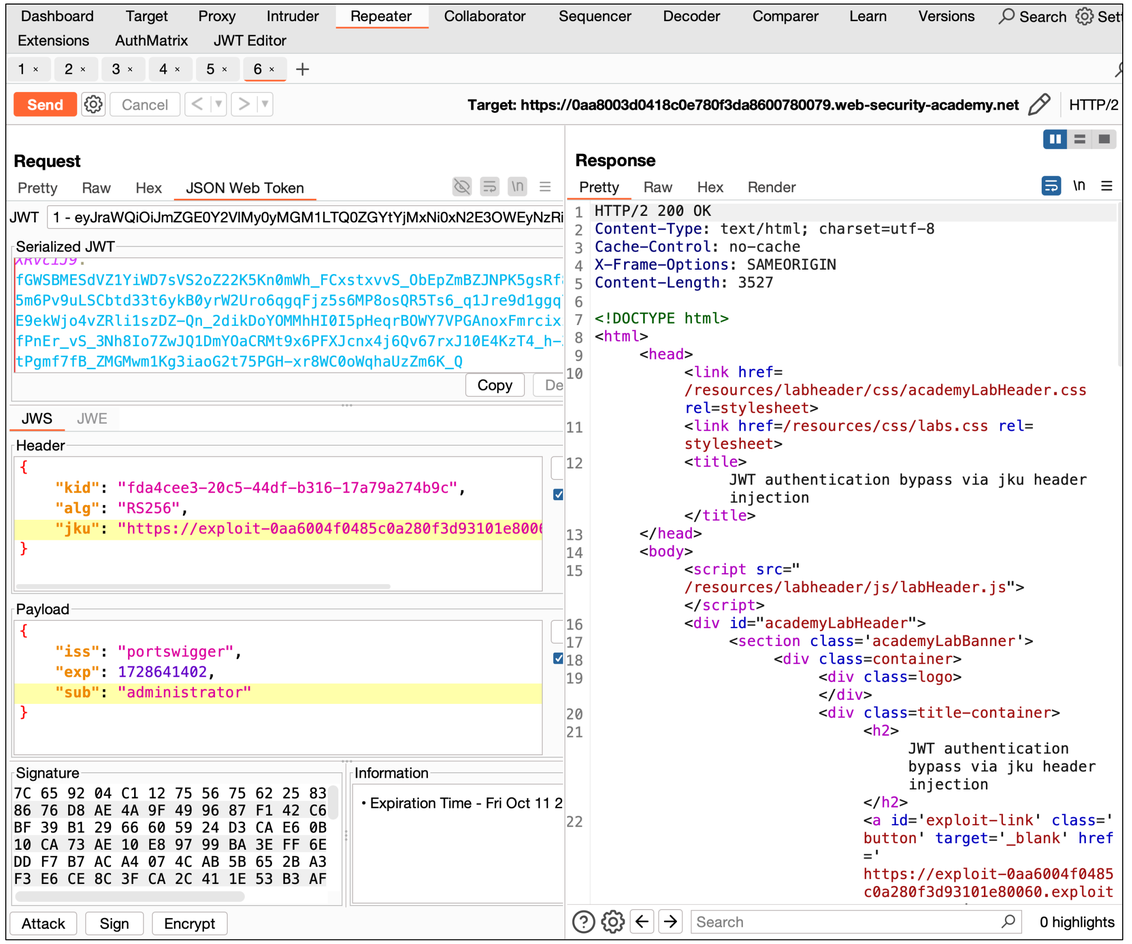
Steps to Find & Exploit:

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.

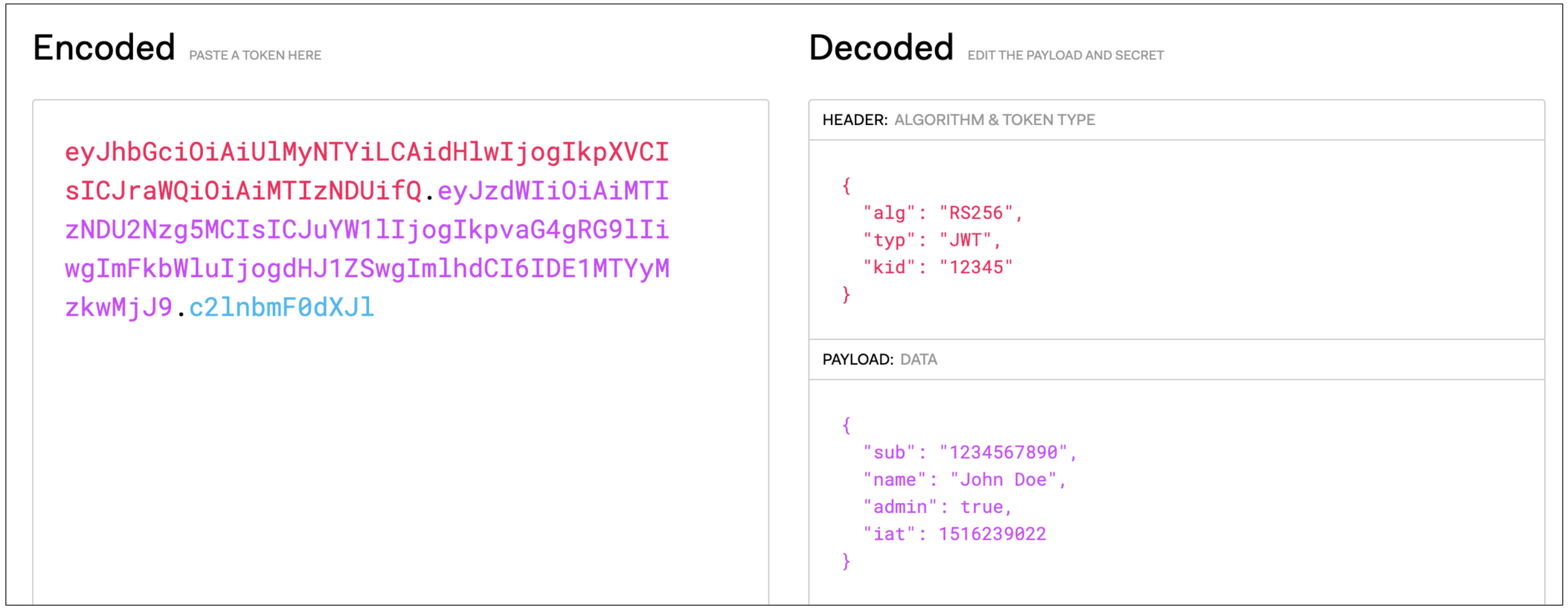
Part 2 - Modify and sign the JWT

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.



V11 – Header Injection via the kid Parameter

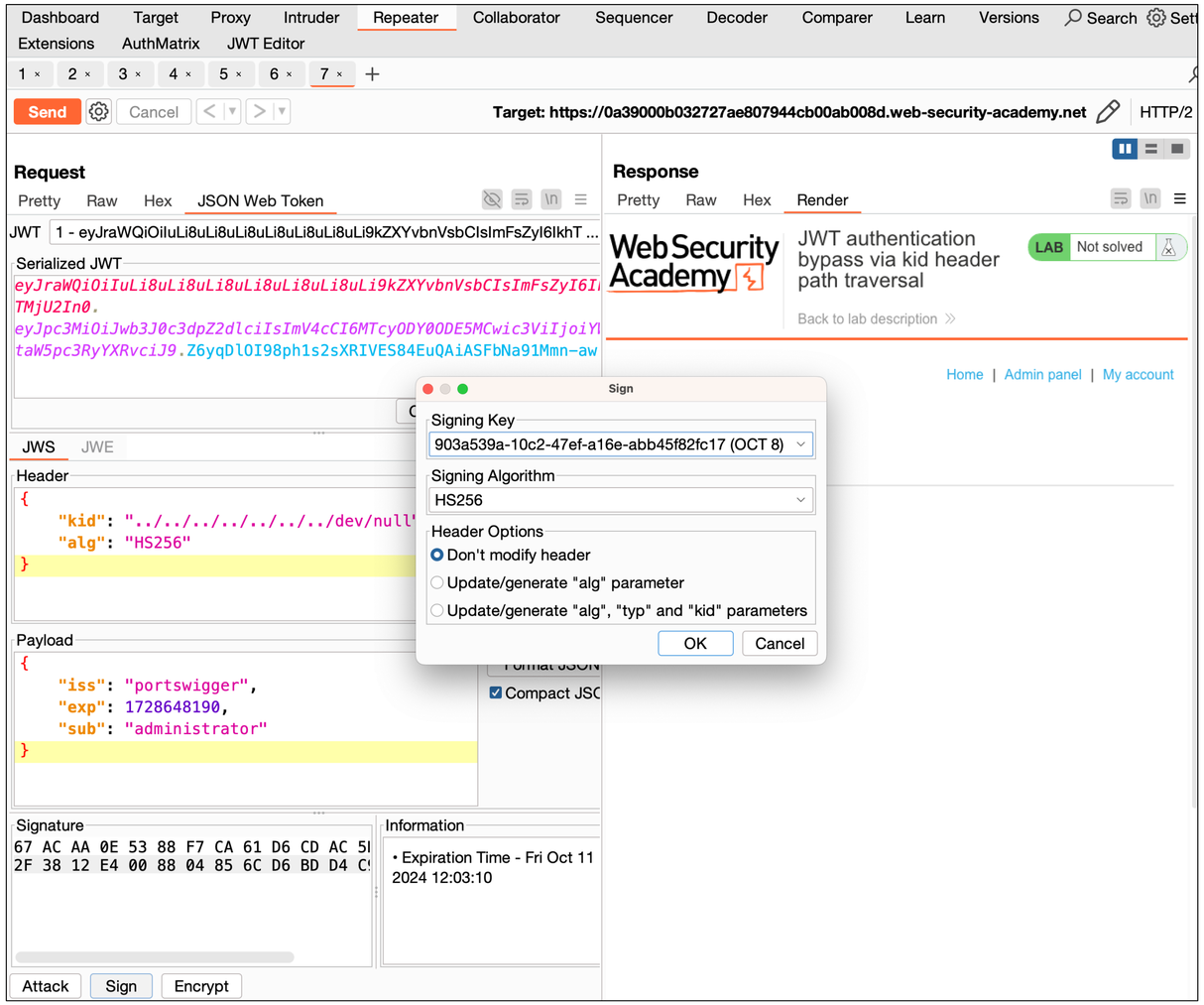
Steps to Find & Exploit:

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**.

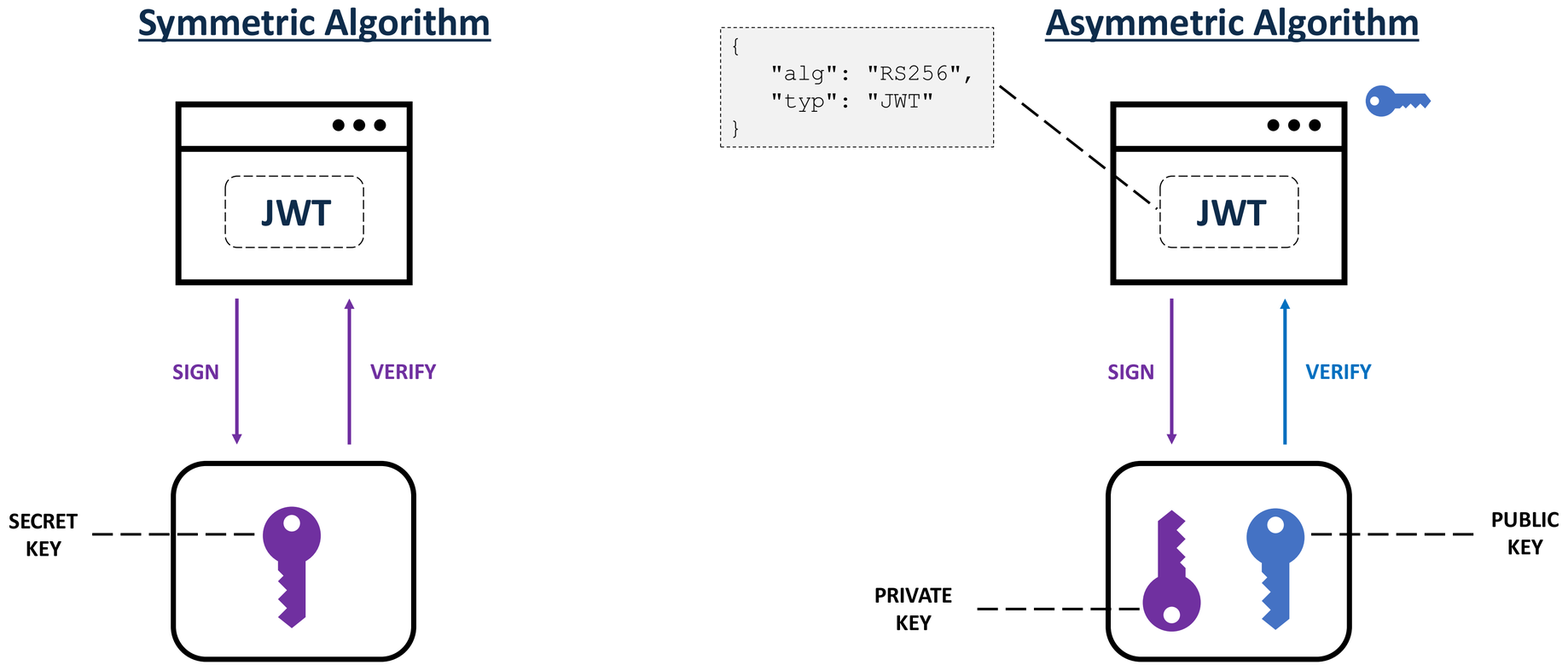
Part 2 – Modify and sign the token

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.



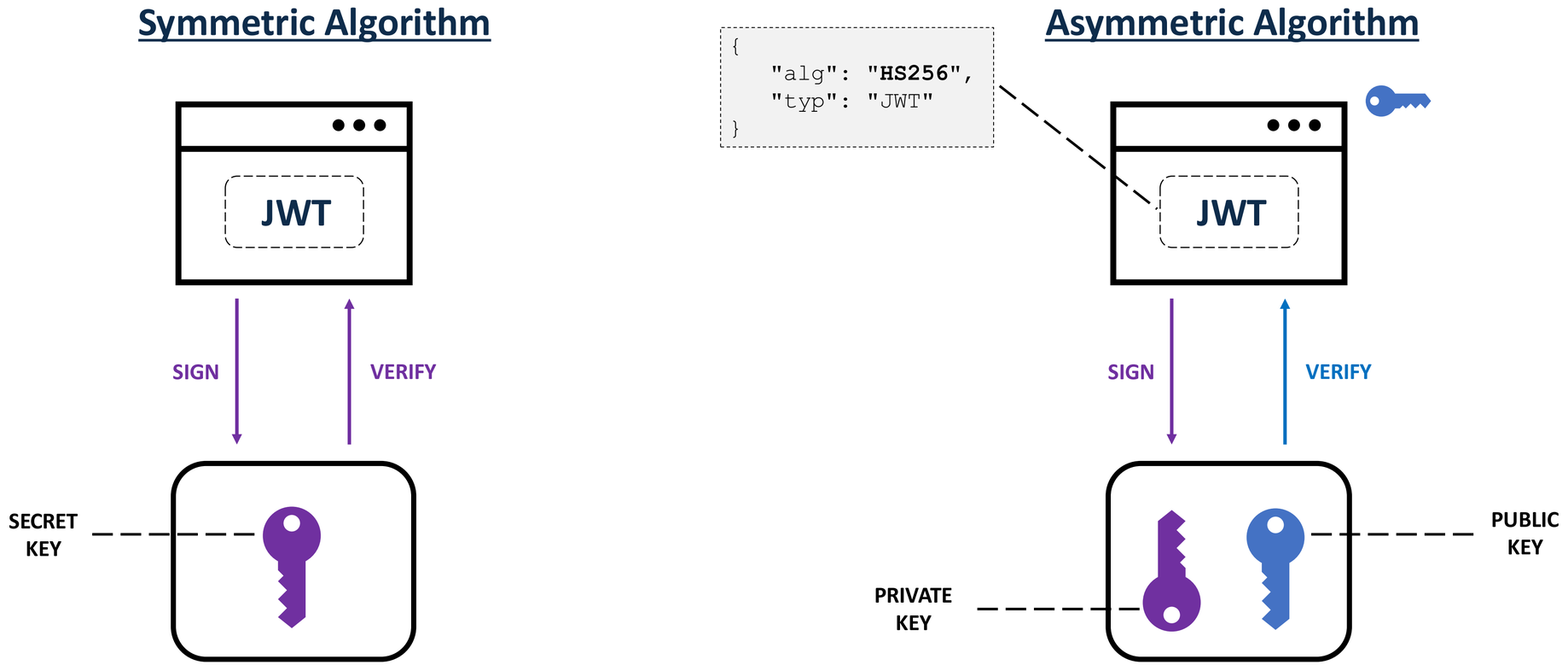
V12 – Algorithm Confusion Attack

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

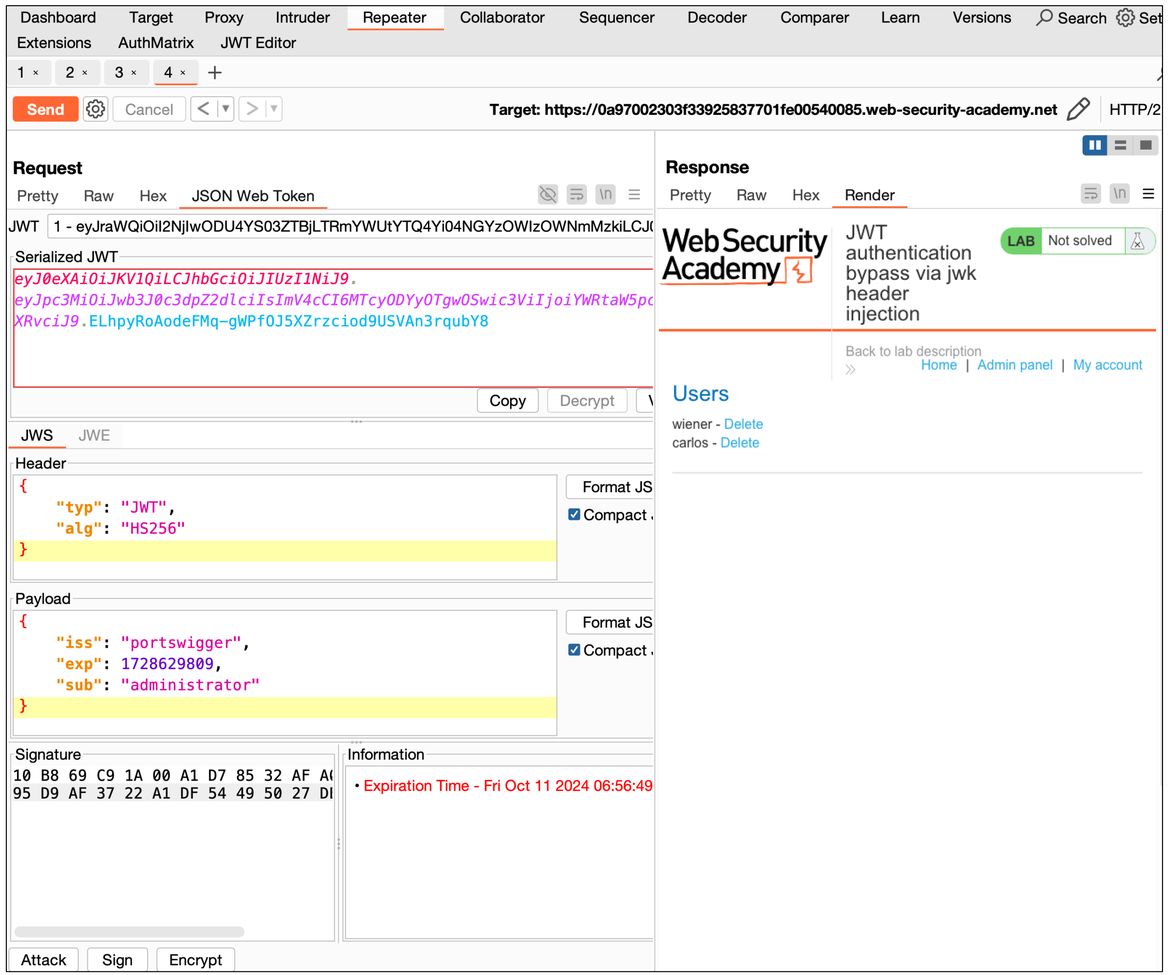
1) Obtain the server's public key

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.



JWT Attacks Labs

LAB APPRENTICE JWT authentication bypass via unverified signature →

LAB APPRENTICE JWT authentication bypass via flawed signature verification →

LAB PRACTITIONER JWT authentication bypass via weak signing key →

LAB PRACTITIONER JWT authentication bypass via jwk header injection →

LAB PRACTITIONER JWT authentication bypass via jku header injection →

LAB PRACTITIONER JWT authentication bypass via kid header path traversal →

LAB EXPERT JWT authentication bypass via algorithm confusion →

LAB 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*