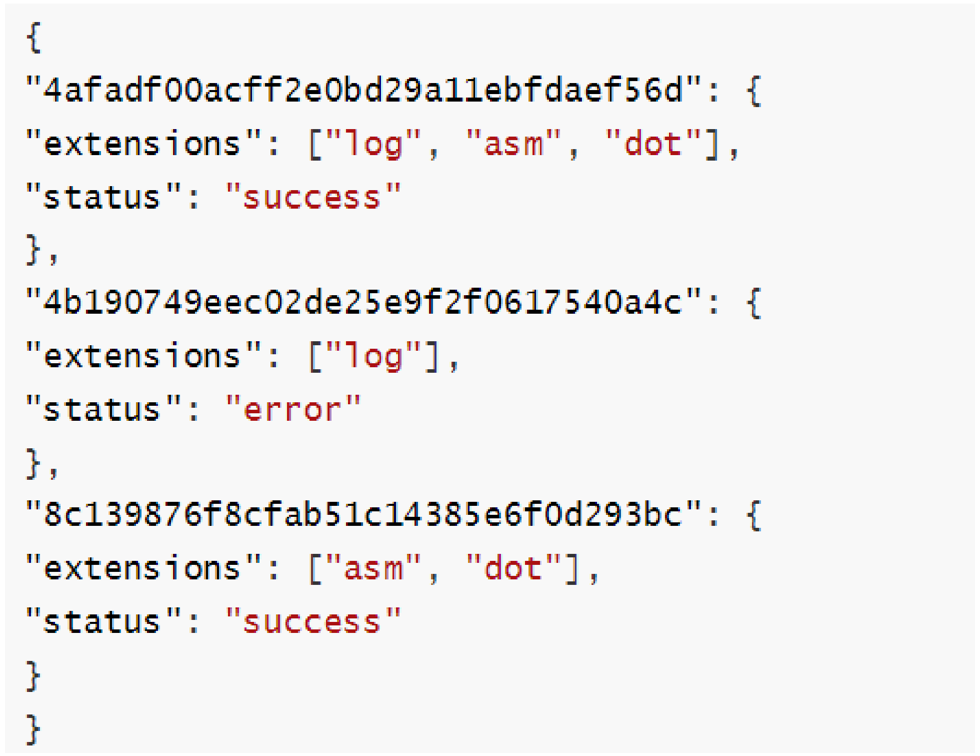
JSON Web Tokens (JWT) Attacks

Presenter: Thao Khong

Hanoi, 2025

Contents

1.1. JSON



```
{
"4afadf00acff2e0bd29a11ebfdaef56d": {
"extensions": ["log", "asm", "dot"],
"status": "success"
},
"4b190749eec02de25e9f2f0617540a4c": {
"extensions": ["log"],
"status": "error"
},
"8c139876f8cfab51c14385e6f0d293bc": {
"extensions": ["asm", "dot"],
"status": "success"
}
}
```

JSON (JavaScript Object Notation):

- a lightweight, language-independent data format
- is designed for data transmission and storage.
- Has a similar structure with JavaScript objects
- **key-value** pairs enclosed in curly braces { } separated by commas.

1.2. JWTs

- Traditional authentication methods: session – cookie
- Session: saves log in state in server side
- Cookie: contains session ID in client side (Session and Persistent)

➔Number of users authenticated ~ cost of storage

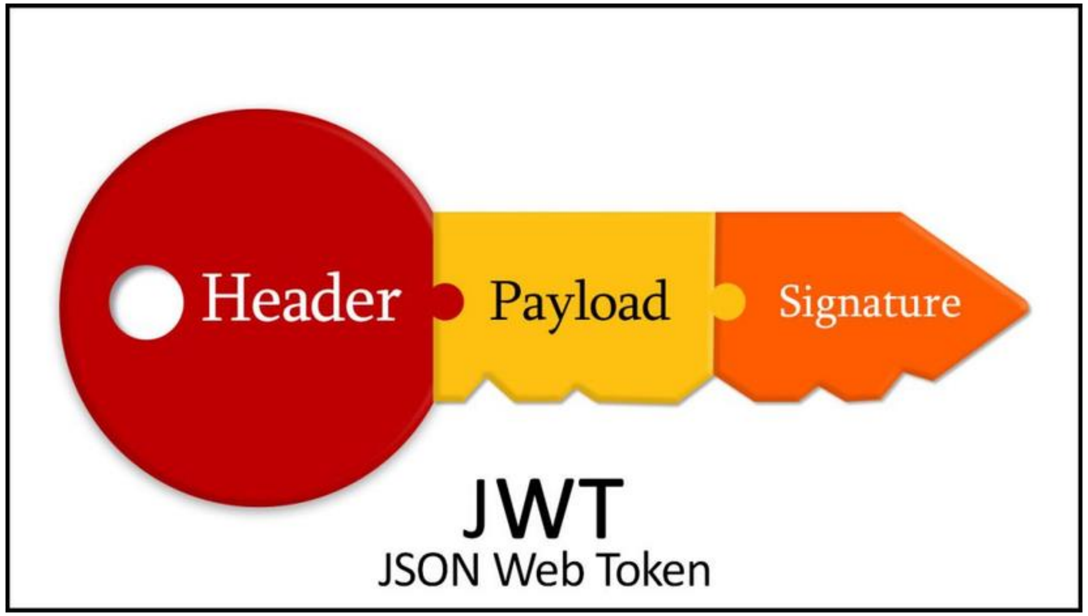
➔Session is recorded on a specific server ➔ user can only access on that server

➔Do not contain much authentication data

➔Some vulnerabilities: session hijacking, CSRF,…

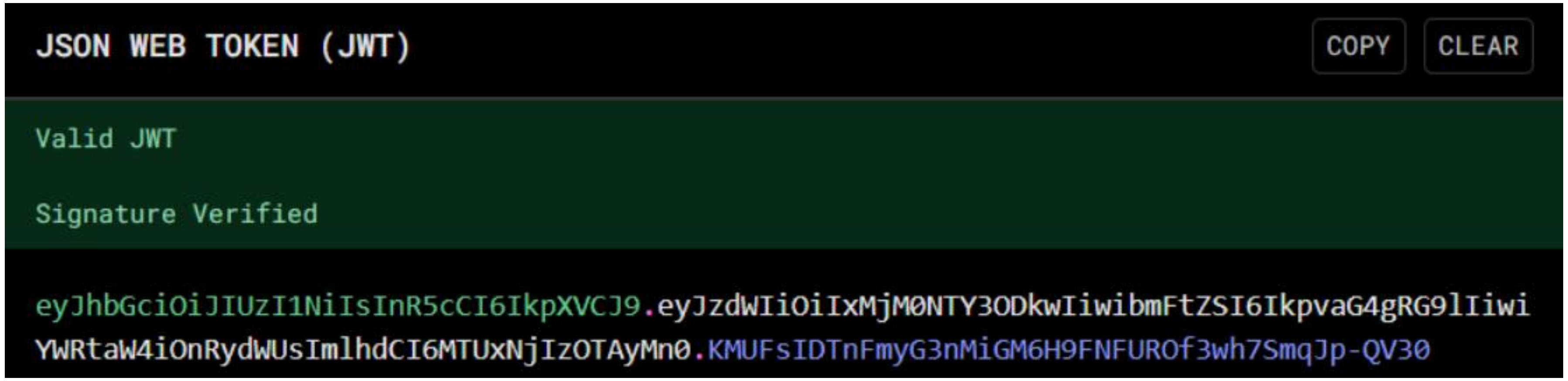
1.2. JWTs

JWT: an open standard (RFC 7519) for transmitting authentication information between a server and a client as a secure JSON string with a digital signature.



1.2. JWTs

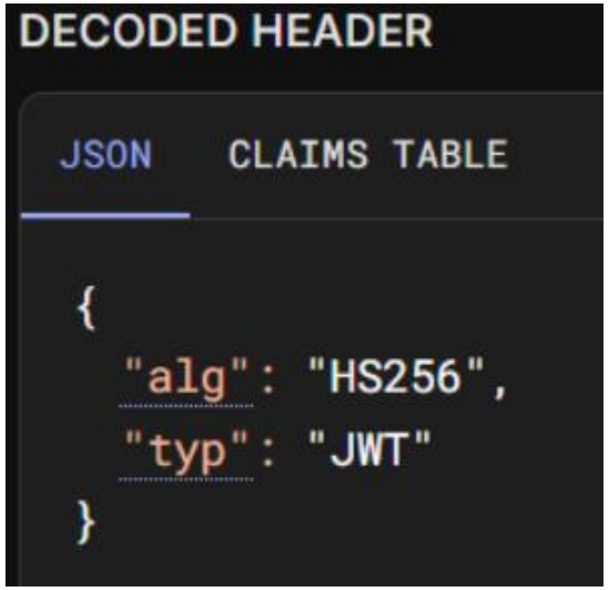
JWT structure:



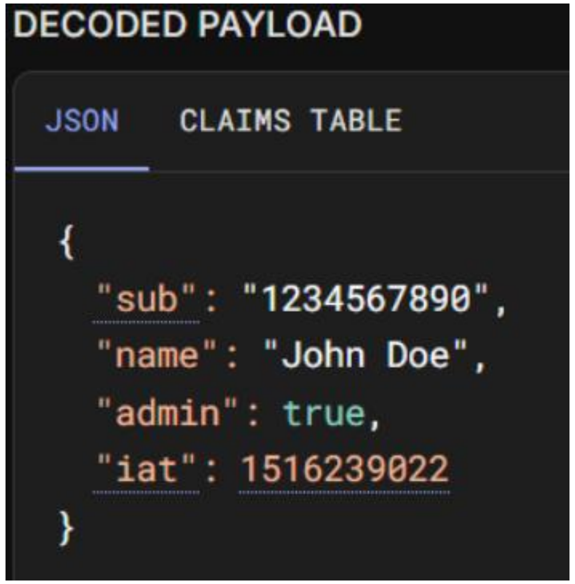
Valid JWT

Signature Verified

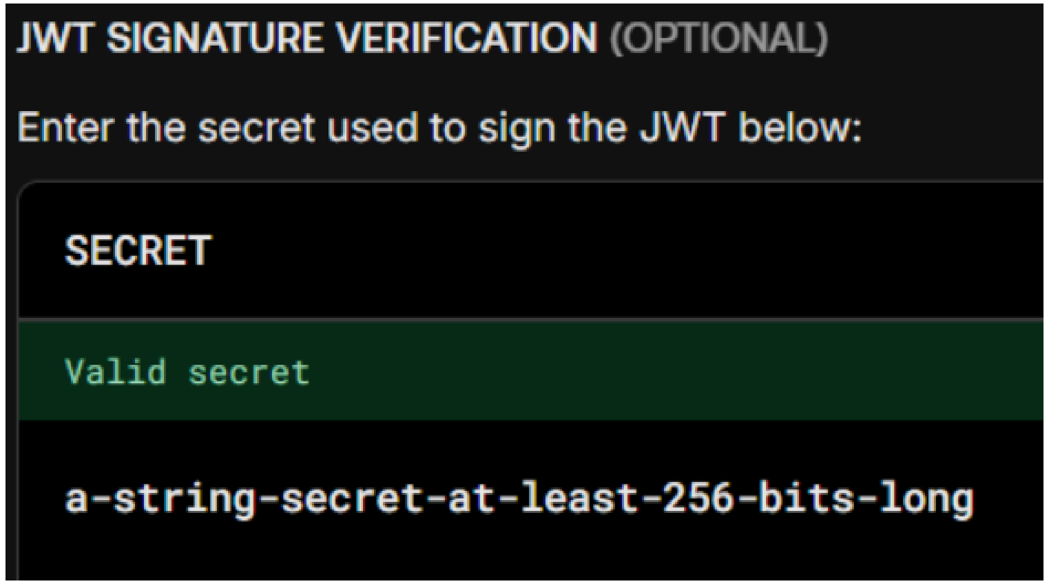
```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwi
YWRtaW4iOnRydWUsImlhdCI6MTUxNjIzOTAyMn0.KMUFsIDTnFmyG3nMiGM6H9FNFUROf3wh7SmqJp-QV30
```



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



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



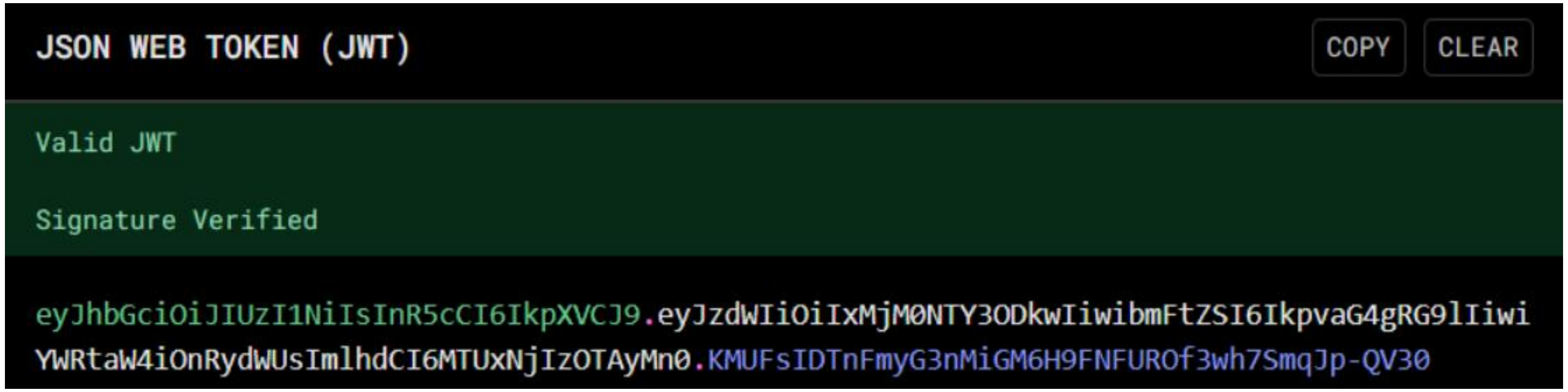
Enter the secret used to sign the JWT below:

Valid secret

a-string-secret-at-least-256-bits-long

1.2. JWTs

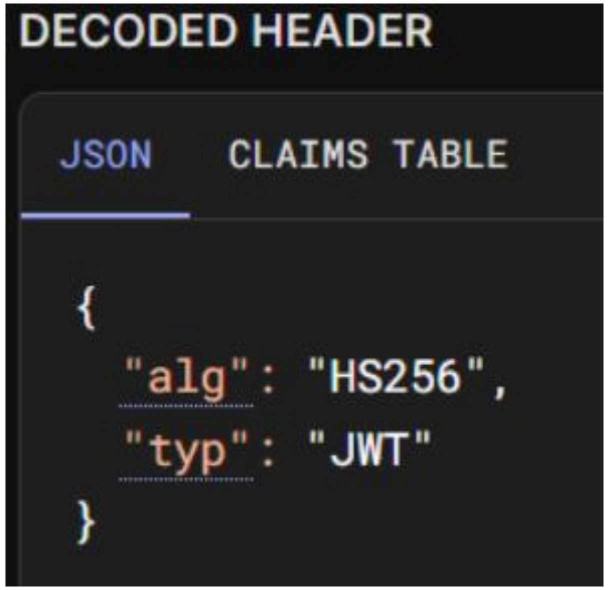
JWT header:



Valid JWT

Signature Verified

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwi
YWRtaW4iOnRydWUsImlhdCI6MTUxNjIzOTAyMn0.KMUFsIDTnFmyG3nMiGM6H9FNFUROf3wh7SmqJp-QV30
```

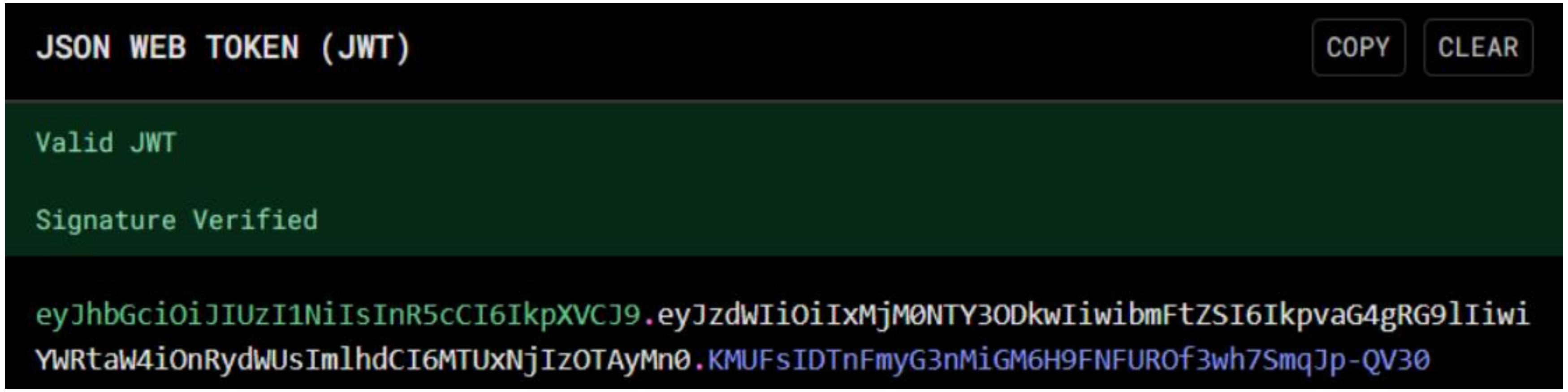


```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- contains info about **token type** and **encryption algorithms** used to create digital signatures for tokens
- Base64 encode JSON objects => encode string called header
- Often has 2 properties: alg and typ
- Ex: HMAC-SHA256 using secret key to sign JWT

1.2. JWTs

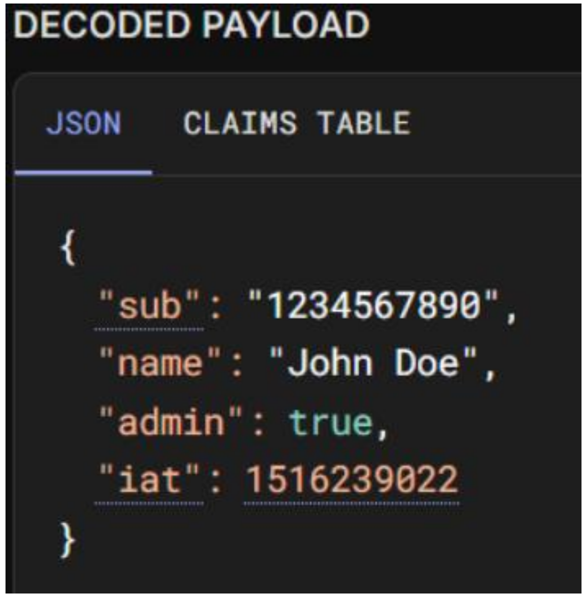
JWT payload:



Valid JWT

Signature Verified

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwi
YWRtaW4iOnRydWUsImlhdCI6MTUxNjIzOTAyMn0.KMUFsIDTnFmyG3nMiGM6H9FNFUROf3wh7SmqJp-QV30



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

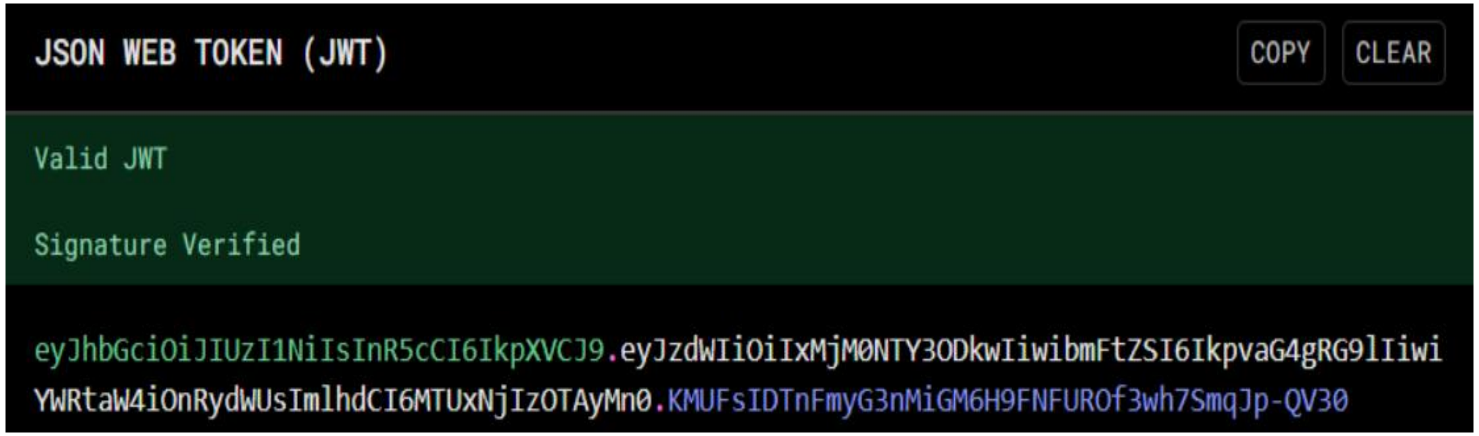
- Encoded in based64
- Often contains:
  - iss (issuer): the issuer of the token.
  - sub (subject): the owner of the token, often the user ID.
  - aud (audience): the application or API that uses the token.
  - exp (expiration time): the token's expiration time.
  - nbf (not before time): the time before which the token is not valid.
  - iat (issued at time): the time when the token was issued.
  - jti (JWT ID): a unique ID for each token.
  - …

1.2. JWTs

JWT signature:

$$\begin{tikzcd} \text{Header + Payload + Secret key} \arrow[d, "\text{HMAC/RSA}"] \\ \text{JWT Signature string} \end{tikzcd}$$

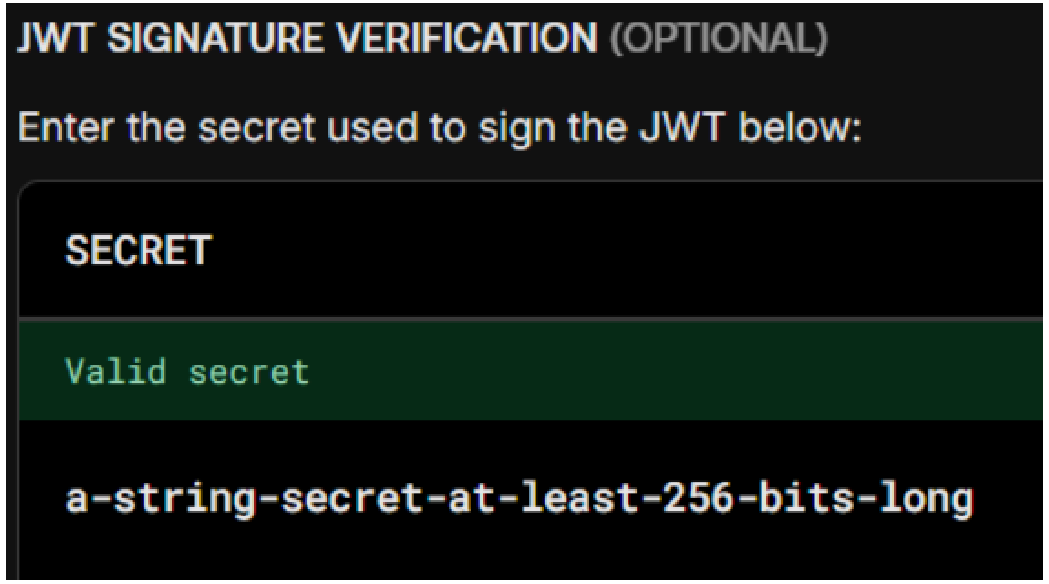
➔ Ensure the integrity of the payload



Valid JWT

Signature Verified

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwi
YWRtaW4iOnRydWUsImlhdCI6MTUxNjIzOTAyMn0.KMUFsIDTnFmyG3nMiGM6H9FNFUROf3wh7SmqJp-QV30
```



Enter the secret used to sign the JWT below:

SECRET

Valid secret

```
a-string-secret-at-least-256-bits-long
```

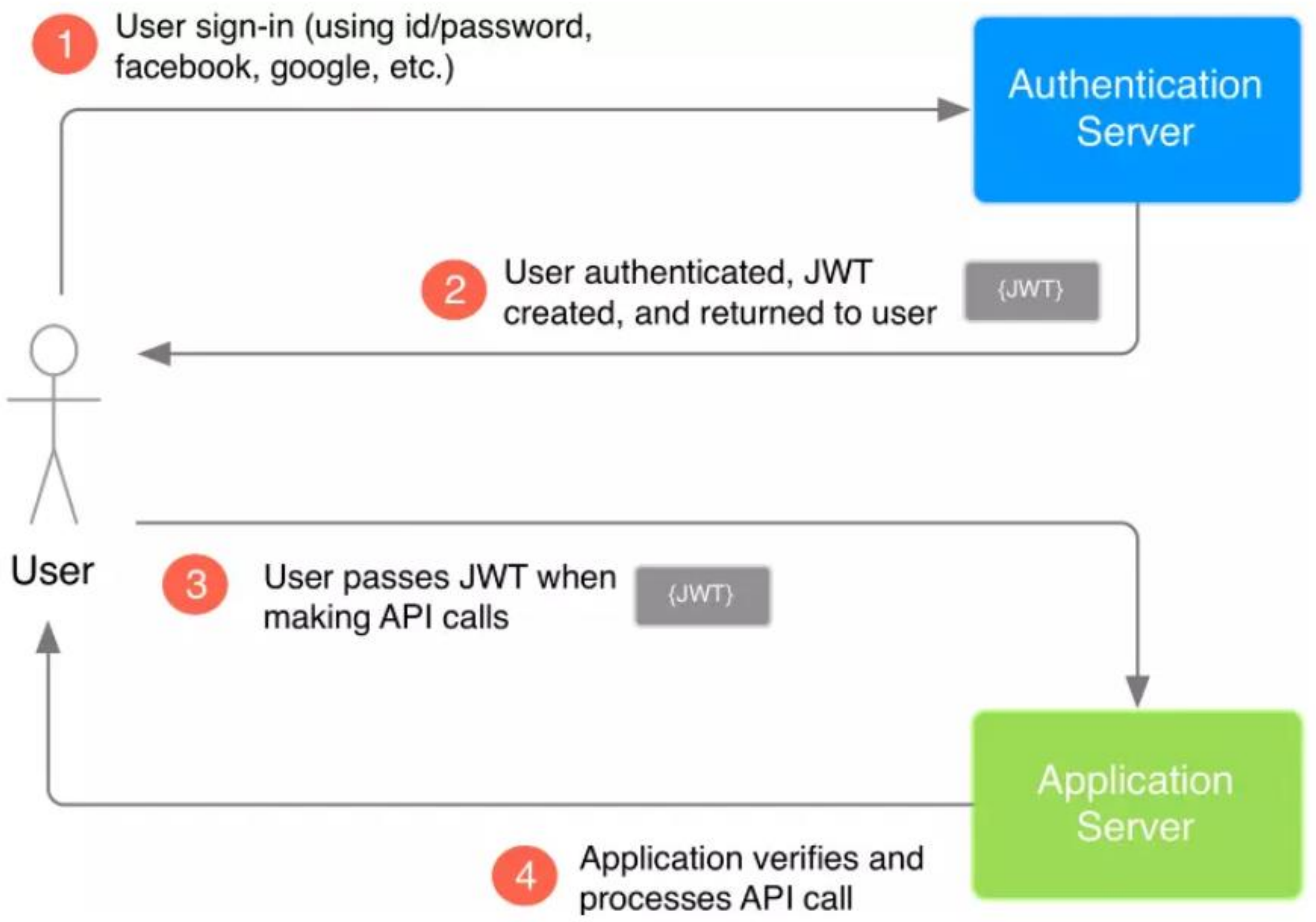
1.2. JWTs

When should JWTs be used:

- User authentication: granting access to the web pages or apis
- User account management: storing account-related information
- Info exchange between apps/ internal micro services: auth.mahcompani.com; mail.mahcompani.com,…

1.2. JWTs

Processing flow:


1
User sign-in (using id/password, facebook, google, etc.)
Authentication Server
2
User authenticated, JWT created, and returned to user
{JWT}
User
3
User passes JWT when making API calls
{JWT}
Application Server
4
Application verifies and processes API call

1.2. JWTs

Pros:

- No need to store session data on server, language-independent,…
- Compact (Base64-encoded) ➔ easily sent in HTTP headers, URLs, or cookies
- Ideal for microservices or single sign-on (SSO) systems
- Scales well across distributed systems and load-balanced environments
- Flexible structure: add or remove properties easily

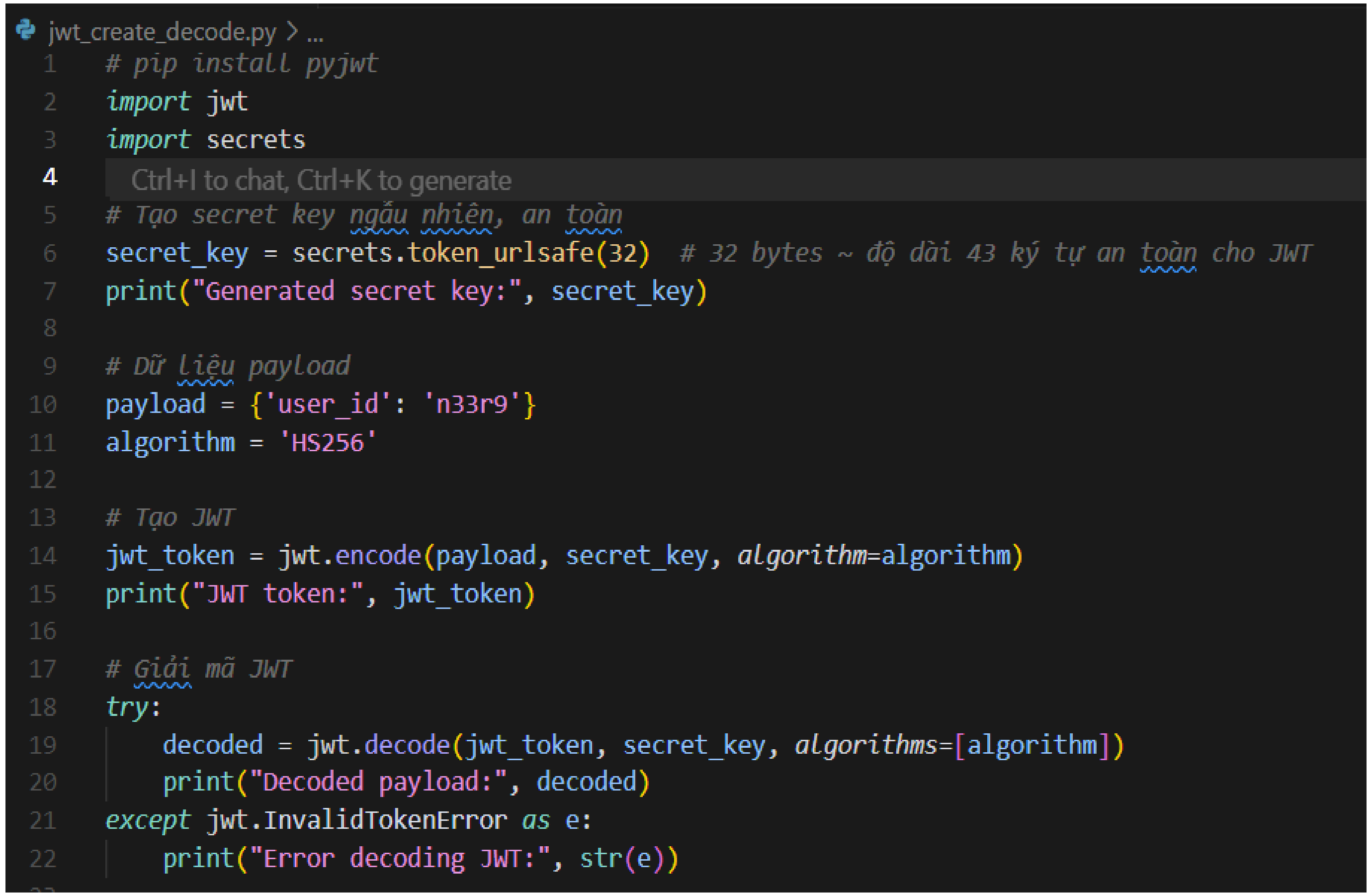
1.2. JWTs

Cons:

- Payloads are Base64-encoded, not encrypted ➔ risk of data exposure
- Non-revocable, large-size token
- Can be abused if not handled securely

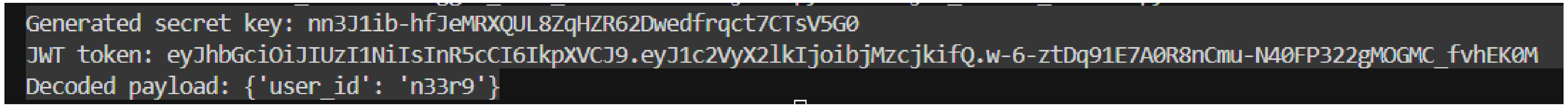
1.2. JWTs

Create a JWT:



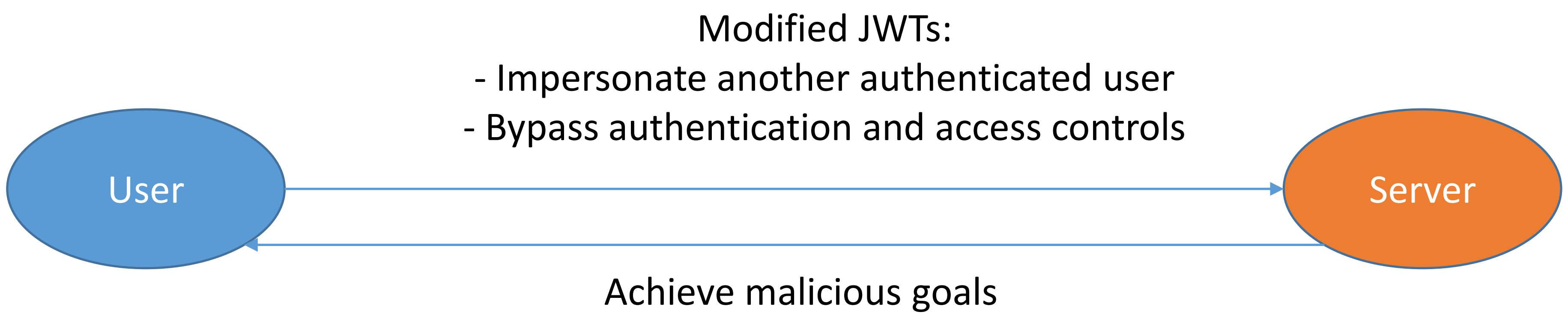
jwt_create_decode.py > ...

```python
# pip install pyjwt
import jwt
import secrets
Ctrl+I to chat, Ctrl+K to generate
# Tạo secret key ngẫu nhiên, an toàn
secret_key = secrets.token_urlsafe(32)  # 32 bytes ~ độ dài 43 ký tự an toàn cho JWT
print("Generated secret key:", secret_key)

# Dữ liệu payload
payload = {'user_id': 'n33r9'}
algorithm = 'HS256'

# Tạo JWT
jwt_token = jwt.encode(payload, secret_key, algorithm=algorithm)
print("JWT token:", jwt_token)

# Giải mã JWT
try:
    decoded = jwt.decode(jwt_token, secret_key, algorithms=[algorithm])
    print("Decoded payload:", decoded)
except jwt.InvalidTokenError as e:
    print("Error decoding JWT:", str(e))
```



```
Generated secret key: nn3J1ib-hfJeMRXQUL8ZqHZR62Dwedfrqct7CTsV5G0
JWT token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoibjMzcjkifQ.w-6-ztDq91E7A0R8nCmu-N40FP322gMOGMC_fvhEK0M
Decoded payload: {'user_id': 'n33r9'}
```

2.1. JWT attack definition:


Modified JWTs:
- Impersonate another authenticated user
- Bypass authentication and access controls
User
Server
Achieve malicious goals

2.2. JWT attack types and defense:

Type 1: Modify the JWT info value

- **Issue**: decodes a JWT token without verifying its signature = trusts any token's payload even if it’s invalid, expired, or tampered
- **Risk**: An attacker can modify the payload (e.g., change ***"isAdmin": false*** to ***"isAdmin": true***) and send the forged token.
- **Mitigate**: **validate the token's signature** before trusting or decoding its content.

2.2. JWT attack types and defense:

Modify the JWT info value:

- **Lab 1:** https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-unverified-signature
- **Lab 2:** https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-flawed-signature-verification

2.2. JWT attack types and defense:

Type 1: Modify the JWT info value:

- Some security-related questions:
  - Does the website verify the signature?
  - Does the website trust the algorithm specified in the token?

2.2. JWT attack types and defense:

Type 2: Bruteforce the security key

- **Lab 3:** https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-weak-signing-key
  - Does the website verify the signature? Yes
  - Does the website trust the algorithm specified in the token? Yes
  - Is the security key strong enough? Can we guess it, thencreate the new payload remake the JWT signature? (The goal is to impersonate the ***administrator***)

2.2. JWT attack types and defense:

*** JOSE Headers and Self-signed JWTs**

- **JOSE headers:** Besides ‘alg’ and ‘typ’

JWK (JSON Web Key): an object JSON to represent a key.

JKU (JWK Set URL): URL contains a set of public courses in the JWK

KID (Key ID): an ID for the public key used to verify the JWT signature...

2.2. JWT attack types and defense:

*** JOSE Headers and Self-signed JWTs**

- **JOSE headers:**

```
{
    "kid": "ed2Nf8sb-sD6ng0-scs5390g-fFD8sfxG",
    "typ": "JWT",
    "alg": "RS256",
    "jwk": {
        "kty": "RSA",
        "e": "AQAB",
        "kid": "ed2Nf8sb-sD6ng0-scs5390g-fFD8sfxG",
        "n": "yy1wpYmffgXBxhAUJzHHocCuJolwDqql75ZWuCQ_cb33K2vh9m"
    }
}
```

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

2.2. JWT attack types and defense:

*** JOSE Headers and Self-signed JWTs**

- **Self-signed JWTs:**

Issuer & Verifier all have the same secret key

2.2. JWT attack types and defense:

Type 3: Attack on JWK parameter in Self-Signed JWTs

Lab 4: https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-jwk-header-injection

Lab 5: https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-jku-header-injection

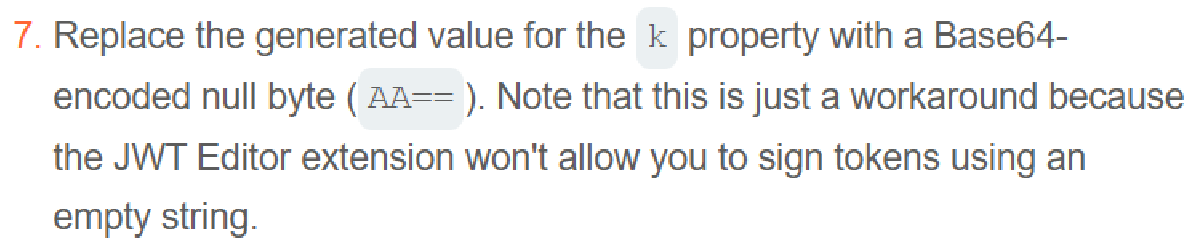
Lab 6: https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-kid-header-path-traversal

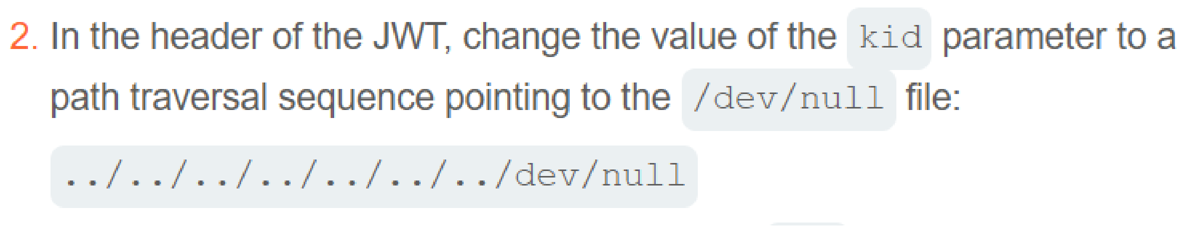
2.2. JWT attack types and defense:

Type 3: Attack on JWK parameter in Self-Signed JWTs

?1 Why Lab 5, we have to generate RSA key, whereas, using the symmetric key for Lab 6?

?2 Lab 6:


7. Replace the generated value for the `k` property with a Base64-encoded null byte (`AA==`). Note that this is just a workaround because the JWT Editor extension won't allow you to sign tokens using an empty string.


2. In the header of the JWT, change the value of the `kid` parameter to a path traversal sequence pointing to the `/dev/null` file:

```
../../../../../../../dev/null
```

How to define this path? Change to this "../../../dev/null", get the same result

Thank you!