AUTH BY-PASS AND JWT VULNERABILITY

Mahya Jamshidian

Secure Software Development

Spring 2020

SIMPLE AUTHENTICATION BYPASS

- By interpreting the packets between server and user, one might get a hand on the session id transferred via http header
- This gives the attacker the opportunity to hijack the users session given only the session_id
- Even though we used hashing in saving the session id.

DEMONSTRATE

JASON WEB TOKEN

- JSON Web Tokens (JWT) are an open, industry standard RFC 7519 method for representing claims ***securely*** between two parties.
- A string that is encoded in a JWS or JWE, enabling the claims to be digitally signed or MACed and/or encrypted.

WHEN SHOULD YOU USE JWTS?

- Authentication
  - Most common scenario
    - Once the user is logged in, each subsequent request will include the JWT
    - Allowing the user to access routes, services, and resources permitted
  - Single Sign-On
    - Widely uses JWT since it has a small overhead and is compatible across different domains

WHEN SHOULD YOU USE JWTS?

- Information Exchange
  - Securely transmitting the claims between parties
    - Using public/private keys

  - Integrity is guaranteed since the content is signed with the payload

JWT STRUCTURE

- Three parts separated by dots (.) which are:
  - Header
  - Payload
  - Signature

JWT STRUCTURE

1 2

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NT
Y3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5M
DIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o
```

3

1 Header

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

2 Payload

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

3 Signature

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

JWT AS API KEYS

Security Problems:

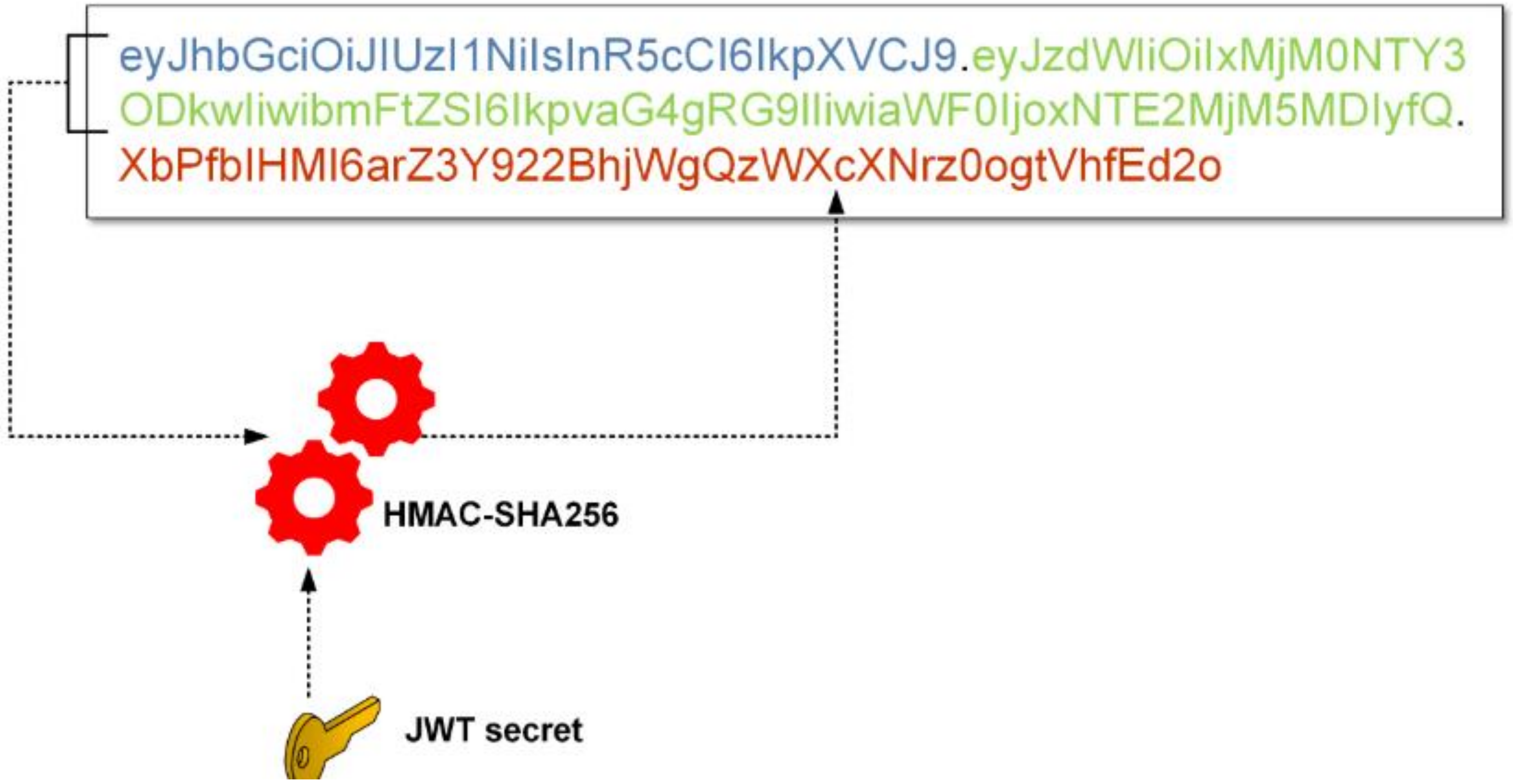
1) Lack of Confidentiality
2) Authorization Bypass

Header:

```
{
“alg ”: “HS256 ”,
“typ ”: “JWT ”
}
```

Payload:

```
{
“iat”: “1416929061”,
“jti”: “802057ff9b5b4eb7fbb8856b6eb2cc5b”,
“scopes”: {
“users”: {
“actions”: [
“read”,
“create”
]
},
“users_app_metadata”: {
“actions”: [
“read”,
“create”
]
}
}
}
```



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3
ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o
```

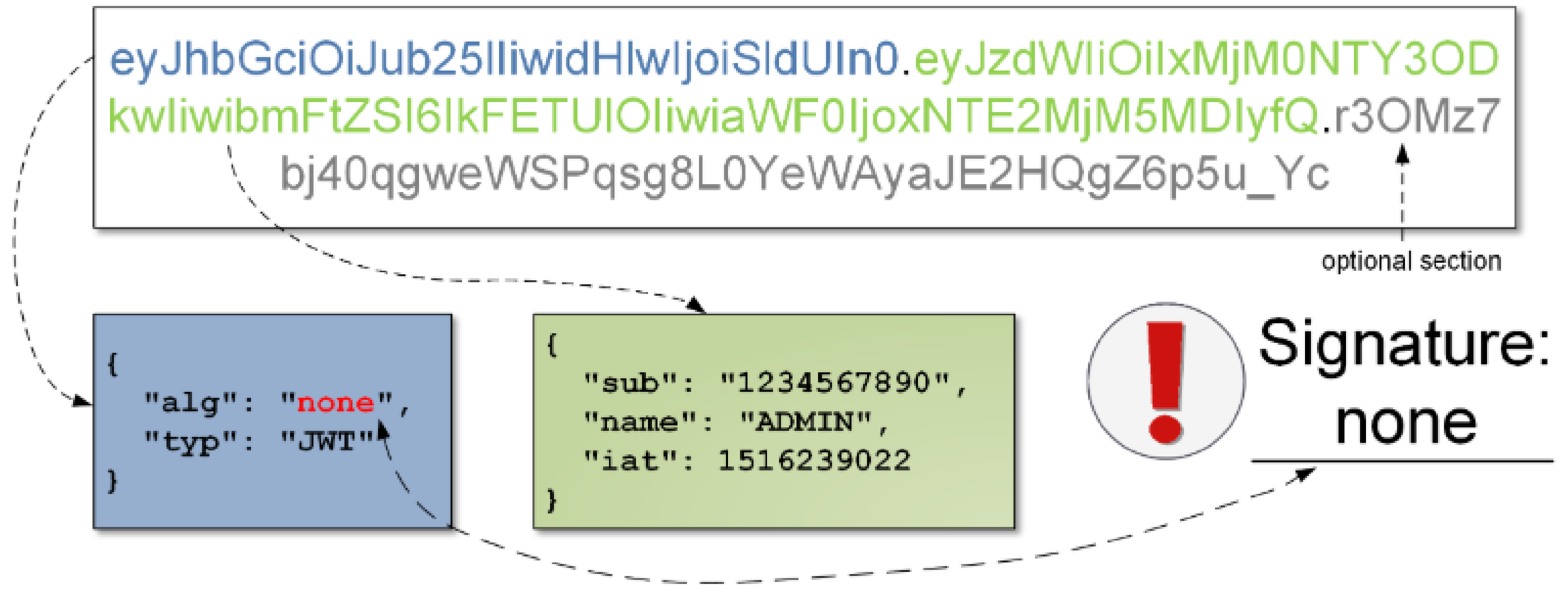
SO, WHY WOULD IT BE NOT SECURE?

- **It’s rather complicated**
  - a multitude of cryptographic algorithms
  - two different ways of encoding (serialization)
  - Compression
  - the possibility of more than one signature
  - encryption to multiple recipients
  - All JWT related specifications have 300+ pages!

The *complexity* is certainly not a friend of *security*.

- **ALG: None**
  - according to the formal specification of JWT, a **signature is not mandatory**

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



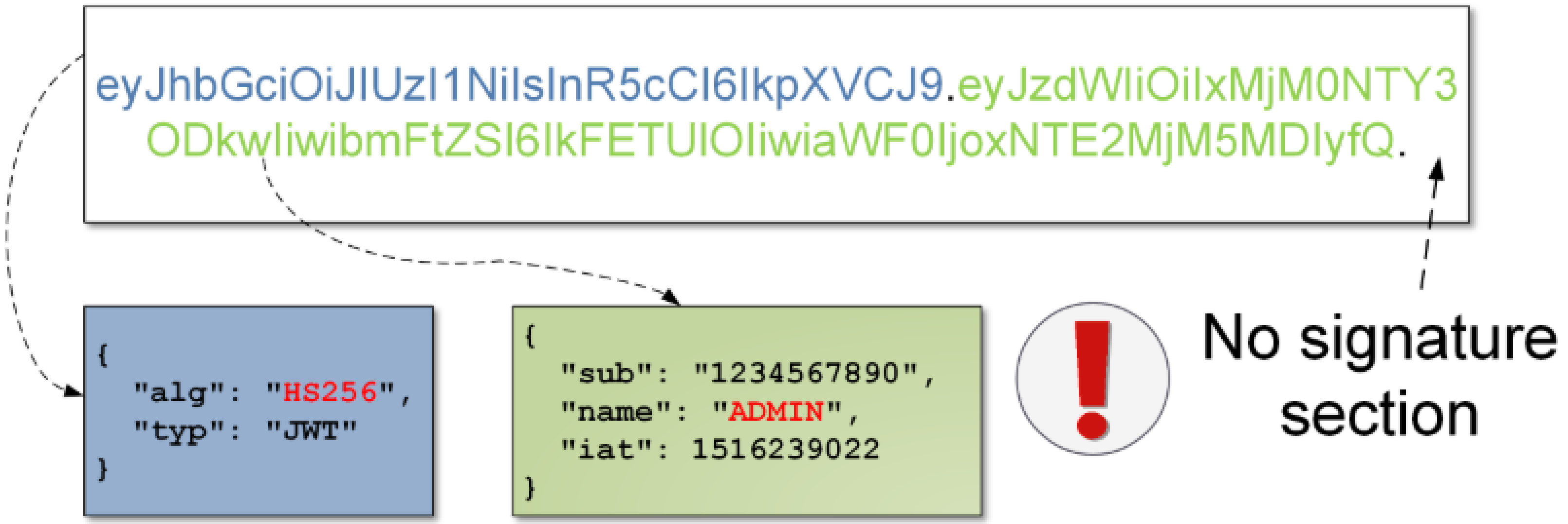
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdWIiOiIxMjM0NTY3OD
kwIiwibmFtZSI6IkFETUlOIiwiaWF0IjoxNTE2MjM5MDIyfQ.r3OMz7
bj40qgweWSPqsg8L0YeWAyaJE2HQgZ6p5u_Yc
optional section
Signature: none

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

```
{
  "sub": "1234567890",
  "name": "ADMIN",
  "iat": 1516239022
}
```

DEMONSTRATE

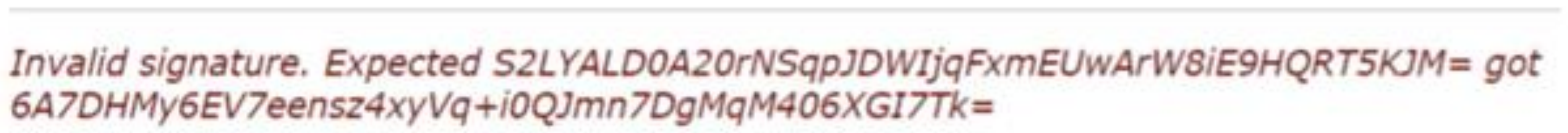
- A valid signature is returned on exception



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkFETUlOIiwiaWF0IjoxNTE2MjM5MDIyfQ.
```

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

```
{
  "sub": "1234567890",
  "name": "ADMIN",
  "iat": 1516239022
}
```



```
Invalid signature. Expected S2LYALD0A20rNSqpJDWIjqFxmEUwArW8iE9HQRT5KJM= got 6A7DHMy6EV7eensz4xyVq+i0QJmn7DgMqM406XGI7Tk=
```

2

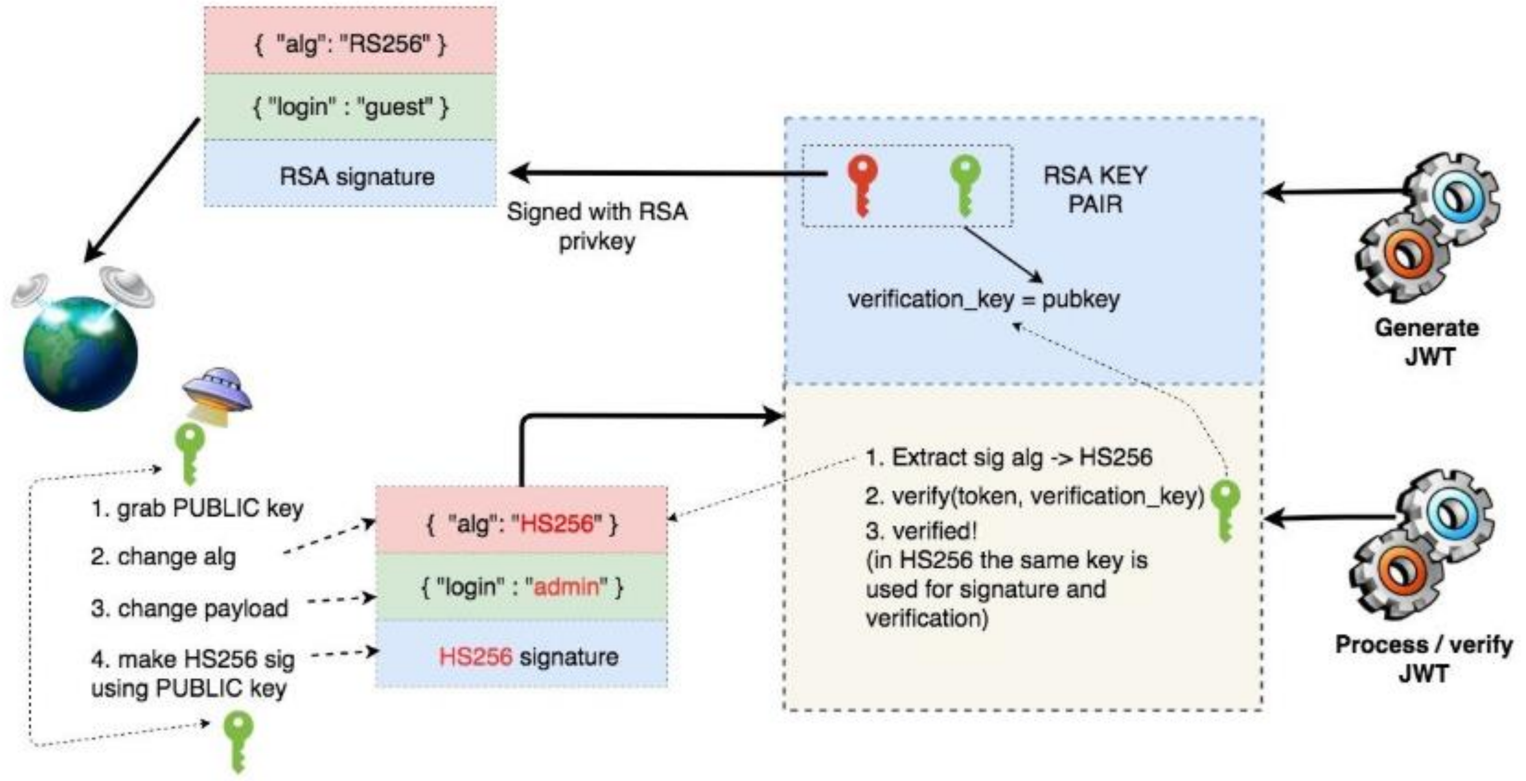
- Cracking the HMAC
  - It can be done offline
  - Hashcat library has a built-in feature for jwt

```
Session..........: hashcat
Status...........: Running
Hash.Type........: JWT (JSON Web Token)
Hash.Target......: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMj...
Guess.Mask.......: ?1?2?2?2?2?2?2 [7]
Guess.Charset....: -1 ?l?d?u, -2 ?l?d, -3 ?l?d*!$@_, -4 Undefined
Guess.Queue......: 7/15 (46.67%)
Speed.Dev.#1.....: 198.0 MH/s (9.68ms) @ Accel:32 Loops:8 Thr:512 Vec:1
Recovered........: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.........: 17964072960/134960504832 (13.31%)
Rejected.........: 0/17964072960 (0.00%)
Restore.Point....: 0/1679616 (0.00%)
Candidates.#1....: U7veran -> a2vbj14
```

RFC: Keysize >= hmacsize

DEMONSTRATE

- too many cooks spoil the JWT broth



OTHER THORNS!

- JWT Header allows to include pub-key used for signing the token!
- JWE implementation has flaws!
- Some libraries decode() function lacks validation
- Advantage: Using same authorization/authentication in several domains! Also a drawback since a simple leakage could cause catastrophic damage.
- Replay attacks
- Timing attacks
- Multitudes of libraries: Multitudes of bugs

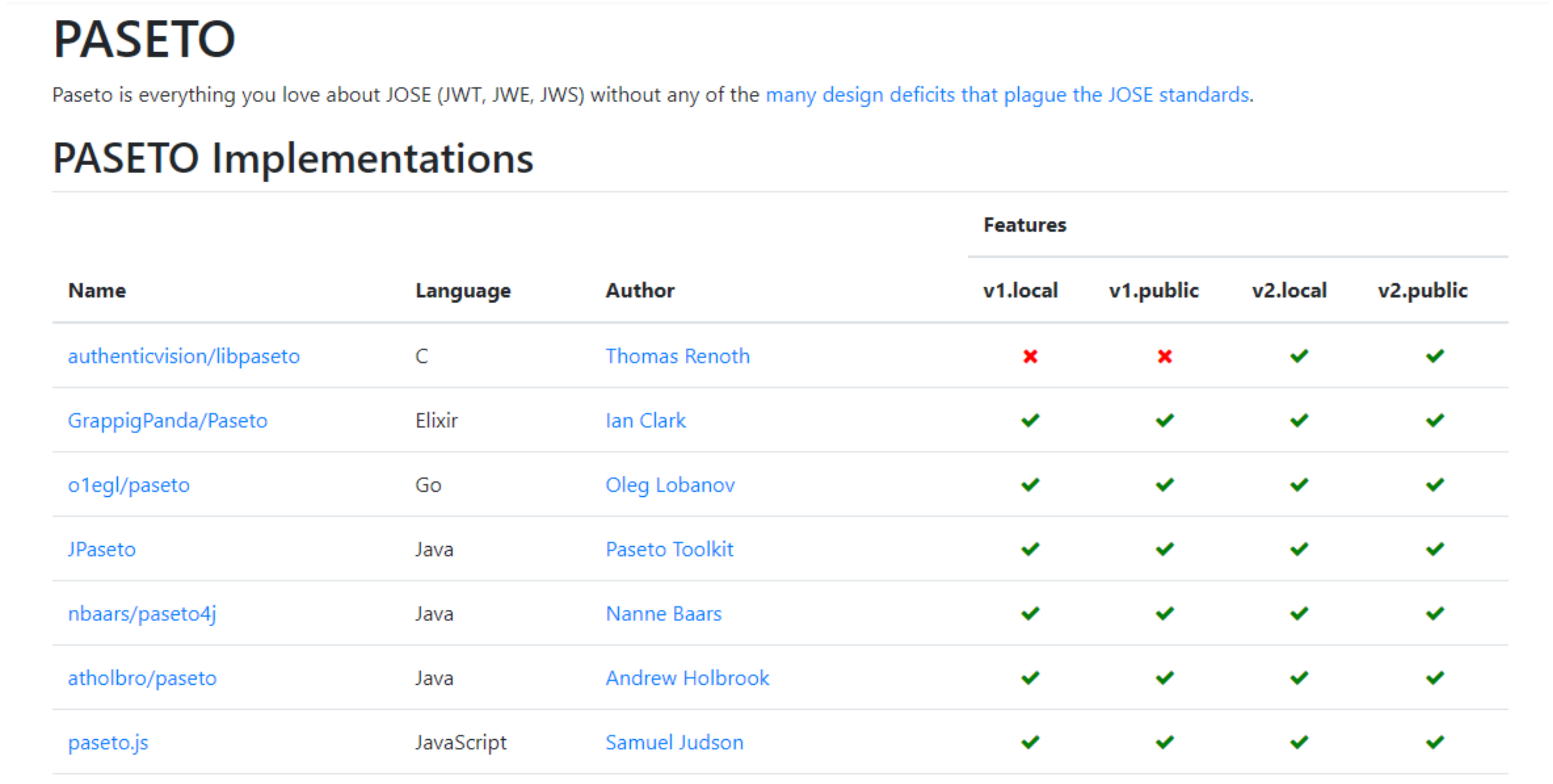
MAKE IT SAFER

Alternative

JWT Hardening

ALTERNATIVE

- PASETO



PASETO

Paseto is everything you love about JOSE (JWT, JWE, JWS) without any of the many design deficits that plague the JOSE standards.

PASETO Implementations

			Features			
Name	Language	Author	v1.local	v1.public	v2.local	v2.public
authenticvision/libpaseto	C	Thomas Renoth	✗	✗	✓	✓
GrappigPanda/Paseto	Elixir	Ian Clark	✓	✓	✓	✓
o1egl/paseto	Go	Oleg Lobanov	✓	✓	✓	✓
JPaseto	Java	Paseto Toolkit	✓	✓	✓	✓
nbaars/paseto4j	Java	Nanne Baars	✓	✓	✓	✓
atholbro/paseto	Java	Andrew Holbrook	✓	✓	✓	✓
paseto.js	JavaScript	Samuel Judson	✓	✓	✓	✓

MAKE JWT SAFER

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3
ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o
```

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

- `none` algorithm
- Insecure accepting of too many signature algorithms

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

- Potentially confidential data
- No automatic token expiry
- Replay attacks
- No validations of certain JWT claims

```
HMACSHA256(
BASE64URL(header)
.
BASE64URL(payload),
secret)
```

- Weak JWT key
- Invalid token handling in case of missing signature section
- No signature check by decode() function
- Timing attacks on signature
- Insecure storage of JWT key

- Vulnerabilities in libraries
- Libraries in debug mode
- Problems with token invalidation
- Token allowing access to too many endpoints

MAKE JWT SAFER

To begin with

- Understand what you want to use:
  - consider whether you need JWS or JWE,
  - choose the appropriate algorithms,
  - understand their purpose (at least on a general level – e.g. HMAC, public key, private key).
  - Find out what exactly offers the JWT library you have chosen. Maybe there is a ready-made, more straightforward mechanism you can use?

MAKE JWT SAFER

Keys

- Use appropriately complex symmetric/asymmetric keys.
- Have a scenario prepared in case of compromise (disclosure) of one of the keys.
- Keep the keys in a safe place (e.g. do not hardcode them permanently in the source code).
- Ideally do not allow to set arbitrary signature algorithm by the sending party (it is best to force a specific signature algorithm(s) on the server side).

MAKE JWT SAFER

Signature

- Check if your implementation does not accept the *none* signature algorithm.
- Check if your implementation doesn't accept an empty signature (i.e. the signature is not checked).
- If you use JWE, check that you are using safe algorithms and that you are using safe implementation of these algorithms.
- Distinguish between verify() and decode(). In other words, check if you are sure you are verifying the signature.

MAKE JWT SAFER

Signature

- Check if your implementation does not accept the *none* signature algorithm.
- Check if your implementation doesn’t accept an empty signature (i.e. the signature is not checked).
- If you use JWE, check that you are using safe algorithms and that you are using safe implementation of these algorithms.
- Distinguish between verify() and decode(). In other words, check if you are sure you are verifying the signature.

MAKE JWT SAFER

General rules

- Check if the token generated in one place cannot be used in another to gain unauthorized access.
- Check that the debug mode is turned off and that it cannot be activated with a simple trick (e.g. ?debug=true).
- Avoid sending tokens in URLs (this might leak sensitive data – e.g. such tokens are then written to web server logs).

MAKE JWT SAFER

Payload

- Check whether you are placing confidential information in JWS payload (not recommended).
- Make sure you are protected against a replay attack (resending a token).
- Make sure that the tokens have a sufficiently short validity period (e.g. by using the "exp" claim).
- Make sure that the "exp" is actually checked. Think about whether you need to invalidate a specific token(s) (the standard does not give tools for this, but there are several ways to implement this type of mechanism).

MAKE JWT SAFER

Libraries

- Read the library's documentation carefully.
- Check the vulnerabilities in the library you use (e.g. in the service: cvedetails.com or on the project website).
- Check that your previous projects do not use a vulnerable library; check if you are monitoring new bugs in the library (they may show up, e.g. after a month of implementation).
- Track new vulnerabilities in libraries that support JWT. Perhaps, in the future, someone will find a vulnerability in another project, which exists in the same form in the library you are using.

REFERENCES

- JSON Web Token Best Current Practices:

https://tools.ietf.org/html/draft-ietf-oauth-jwt-bcp-04

- JWT Handbook:

https://auth0.com/resources/ebooks/jwt-handbook

- Discussion on vulnerabilities of JWT:

https://lobste.rs/s/r4lv76/jwt_is_bad_standard_everyone_should_avoid

- JWT Cheat Sheet for Java (OWASP).

https://www.owasp.org/index.php/JSON_Web_Token_(JWT)_Cheat_Sheet_for_Java

- A couple of ideas on how to use JWT safer:

https://dev.to/neilmadden/7-best-practices-for-json-web-tokens

- A set of arguments against using JWT to create a session:

http://cryto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/

- Comparison of JWTs with session IDs and advice on relevant security features:

http://by.jtl.xyz/2016/06/the-unspoken-vulnerability-of-jwts.html

THANK YOU!