API Security Bootcamp
Hands-On OWASP Top 10 for APIs

Dr. Sunny Wear

2023

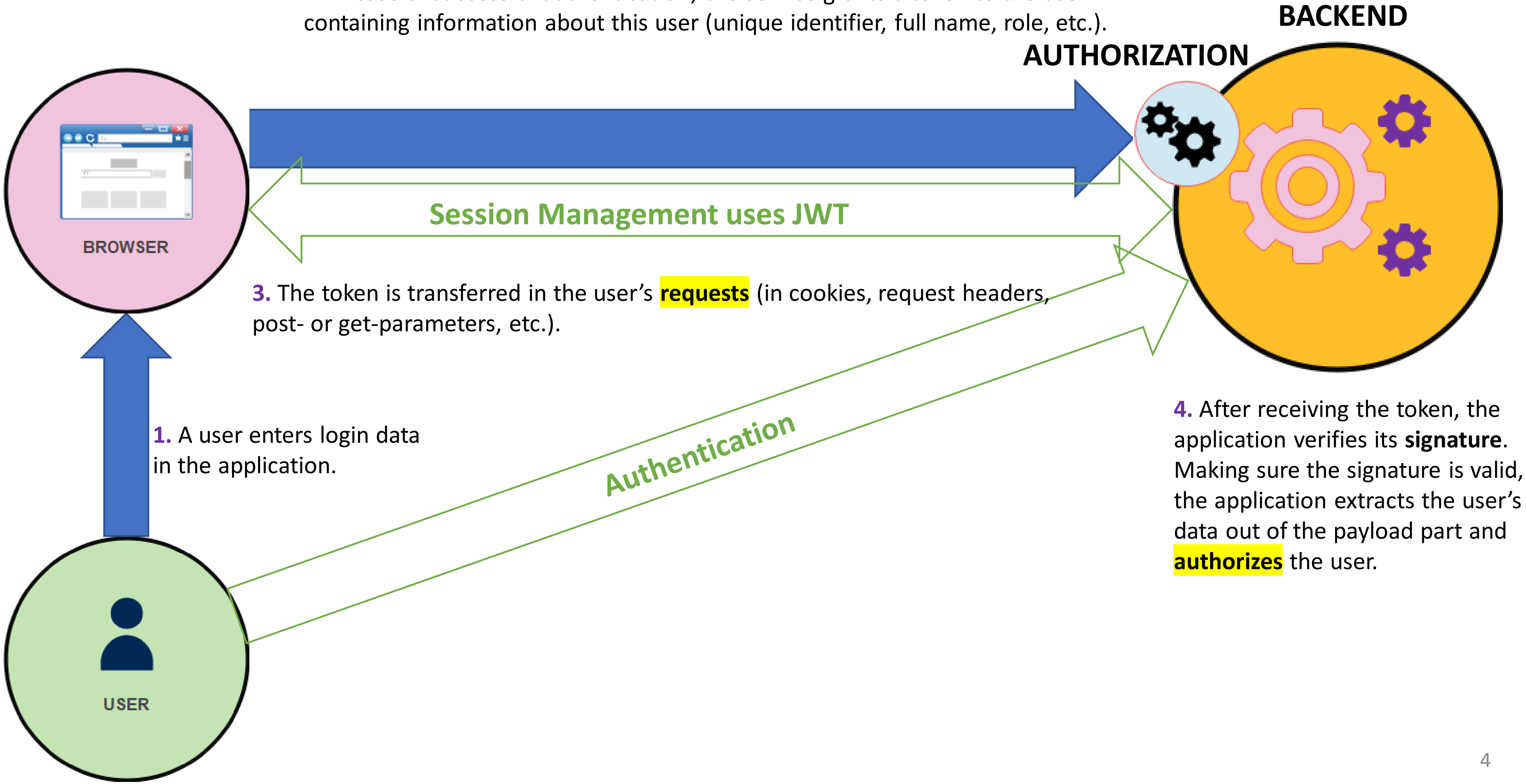
Hacking JWTs

JSON Web Tokens

A ‘self-described’ token typically used to hold session management information to identify an authenticated user and some authorized abilities.

What is a JSON Web Token?

JWT Authentication Flow



1. A user enters login data in the application.

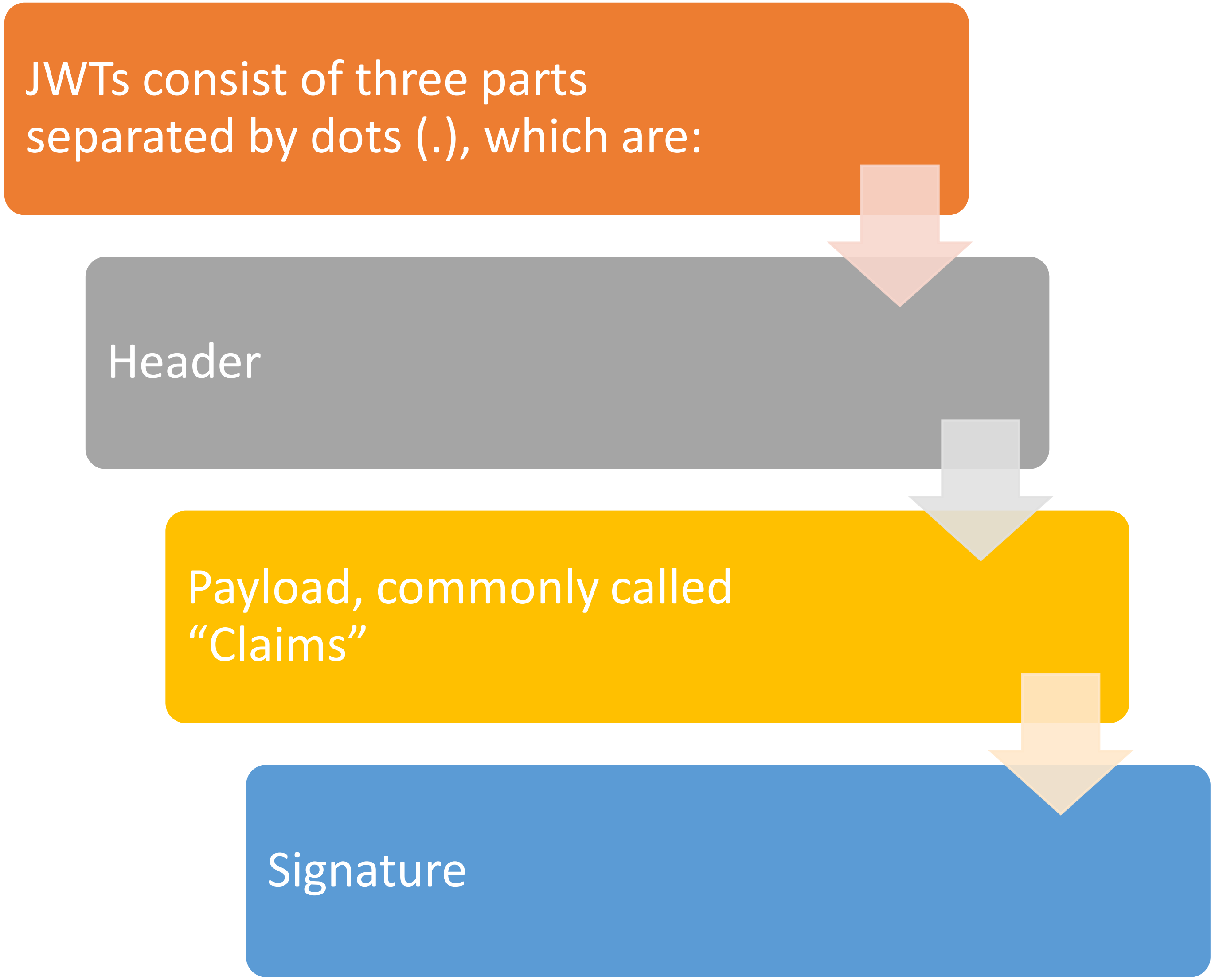
2. In case of successful authentication, the service grants a token to the user containing information about this user (unique identifier, full name, role, etc.).

3. The token is transferred in the user’s **requests** (in cookies, request headers, post- or get-parameters, etc.).

4. After receiving the token, the application verifies its **signature**. Making sure the signature is valid, the application extracts the user’s data out of the payload part and **authorizes** the user.

Bearer Token = Keys to the Kingdom

JWTs Components



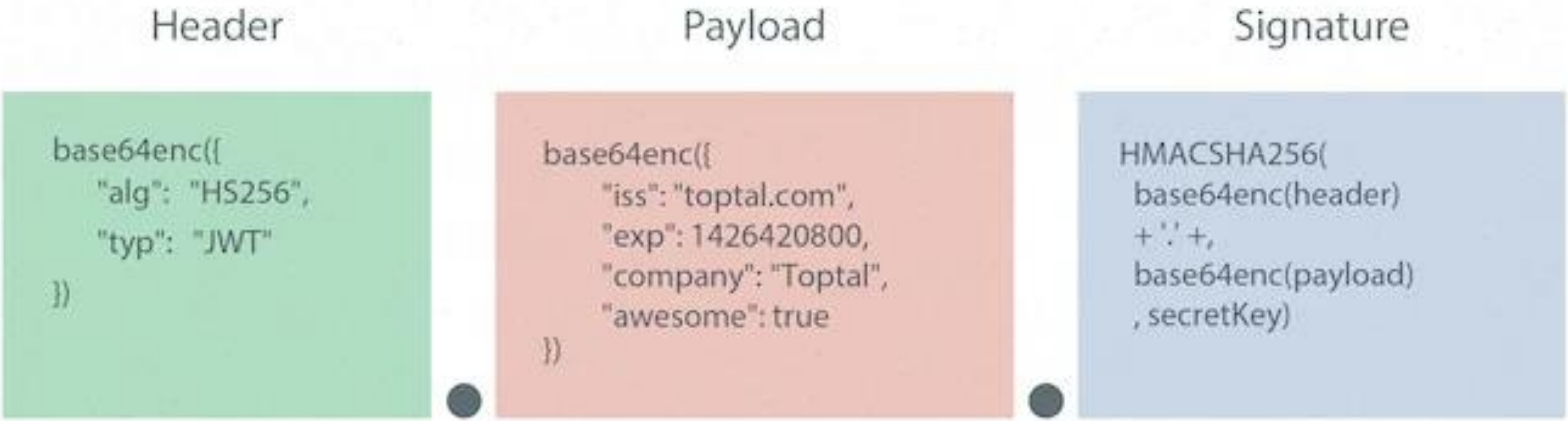
JWTs consist of three parts separated by dots (.), which are:

Header

Payload, commonly called “Claims”

Signature

To create the signature, you take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.



Header

```
base64enc({
     "alg":  "HS256",
     "typ":  "JWT"
})
```

Payload

```
base64enc({
       "iss": "toptal.com",
       "exp": 1426420800,
       "company": "Toptal",
       "awesome": true
})
```

Signature

```
HMACSHA256(
  base64enc(header)
  + '.' +,
  base64enc(payload)
  , secretKey)
```

Bearer of the token has access

Must examine expiration policy

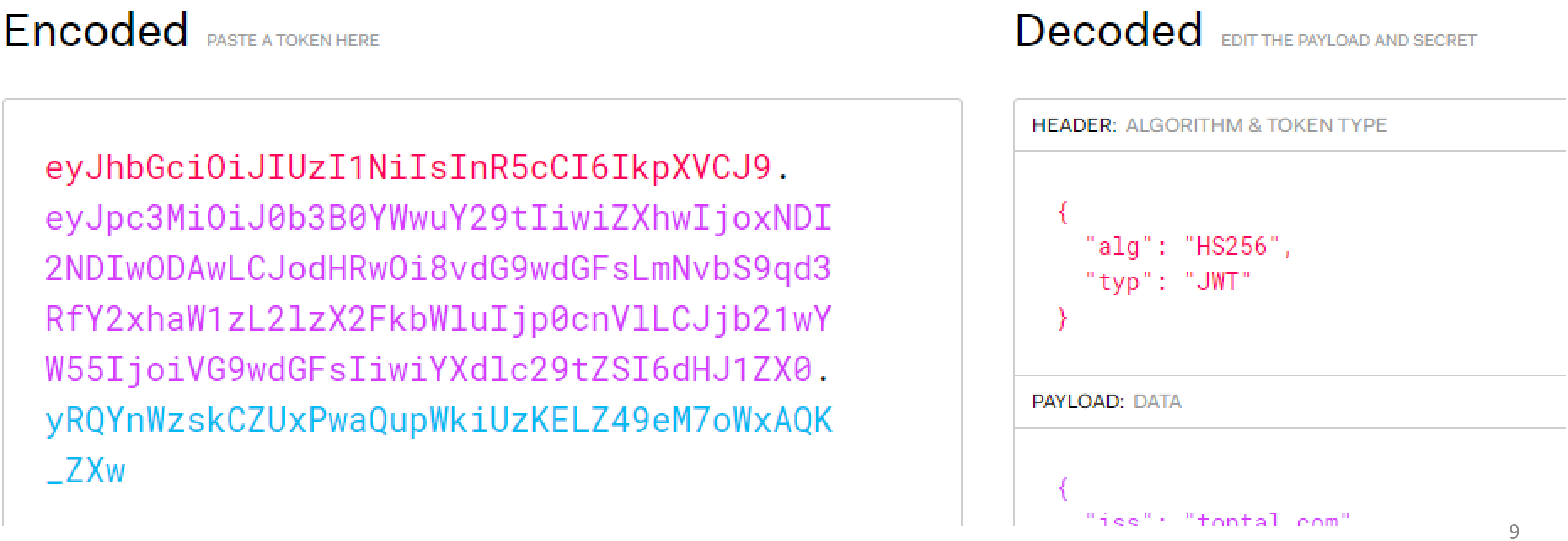
Try to re-use expired tokens

Mitigations:

- Short lifespan
- Randomness or nonce (e.g., jti field in claims)

Your New Best Friend!

https://jwt.io



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpc3MiOiJ0b3B0YWwuY29tIiwiZXhwIjoxNDI
2NDIwODAwLCJodHRwOi8vdG9wdGFsLmNvbS9qd3
RfY2xhaW1zL2lzX2FkbWluIjp0cnVlLCJjb21wY
W55IjoiVG9wdGFsIiwiYXdlc29tZSI6dHJ1ZX0.
yRQYnWzskCZUxPwaQupWkiUzKELZ49eM7oWxAQK
_ZXw
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "iss": "toptal.com"
```

Exercise 4-1: Reading Your First JWTs

- Copy and paste these into the site https://jwt.io

Token 1:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ0b3B0YWwuY29tIiwiZXhwIjoxNDI2NDIwODAwLCJodHRwOi8vdG9wdGFsLmNvbS9qd3RfY2xhaW1zL2lzX2FkbWluIjp0cnVlLCJjb21wYW55IjoiVG9wdGFsIiwiYXdlc29tZSI6dHJ1ZX0.yRQYnWzskCZUxPwaQupWkiUzKELZ49eM7oWxAQK_ZXw

Token 2:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjEzMzciLCJ1c2VybmFtZSI6ImJpem9uZSIsImlhdCI6MTU5NDIwOTYwMCwicm9sZSI6InVzZXIifQ.ZvkYYnyM929FM4NW9_hSis7_x3_9rymsDAx9yuOcc1I

JWT Validation

- For Hashes
  - Need secret in order to verify and/or modify contents of the JWT
  - Secret is shared with client and server OOB
- For RSA
  - Need public key in order to validate issuer’s digital signature
  - Need private key in order to modify contents of JWT

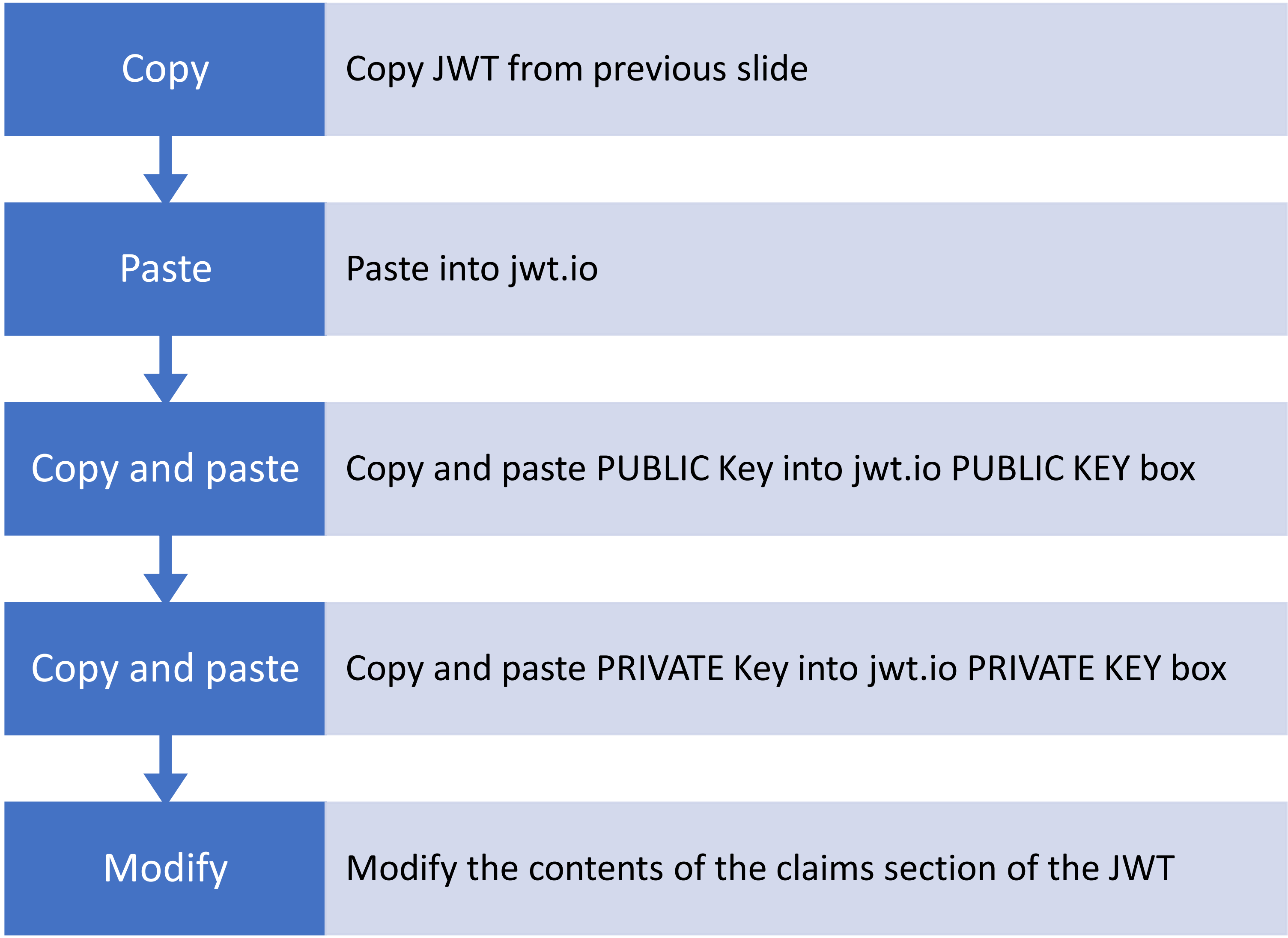
- eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJBUEkgSGFja2luZyBXb3Jrc2hvcCIsIm5hbWUiOiJTdW5ueSBXZWFyIiwiaWF0IjoxNTE2MjM5MDIyfQ.mOqSnYiF6yfPG-hYHL5dyFB7jhN3Y6xKQ4AXC1pSLRGzLPsIXUmq2hM4Acp8ub67OBIcEDQG8LpxsH6iGllJyPCvnNfgMHvOFryyqlLTE5Lh74r9rzGe_sE8rcldpriKqsyo9rYr_Wyyn9O8E5I7HJS_qUuklRSCFMC9GfaZcaWWkVfkc2rzwUD9grU7XU0zNebofziaeZIkzhlcJNBcABgL6uoCcxiwYxCNZRMGw9Q7LofyvwEaJT-f1p536CMj_K1362ZprayDDUchK7-UcCh5ZcrlHDgI14URdUXLrHIJu6ovYs91tGukDv02eY9Jcybb2ME6uGa3aOFXy1EmKg

Practice: JWT Validation

- Copy JWT from previous slide
- Paste into jwt.io
- Copy and paste PUBLIC Key into jwt.io PUBLIC KEY box
- See the Digital Signature is verified (BLUE)

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAungIJYymAQvXgjgp7BdS
H8I7rPC6LCCwCRq8jG5FUEpyq+7pb/xDF/bOB46PIxH1QHwfCsbiaG9YhGnB4meV
gWJCU6FDfRuAqz0kqf6JuifwaKuVvWlSg7R7kAqzb9M9jvH97qH9/78EJ3u4/d37
vzmHGPOro1r3DV2B0oy4/Fd4+12+K7fxWPIWspzm+gBvPDkkvyRfC02DzG4V8vhS
0TqfAs7MEAFVGHMOlcpDtD2thZOkjKxs1PayomPgZrs4aRRbBb4jG2lpPHCYEkIi
93xdJAeXXSbzROdnU9IsT/ZbSrF/8v7ClpNbF1CQtzddLe7eGNXkSJjNExmKB/O2
vQIDAQAB
-----END PUBLIC KEY-----
```

Practice: JWT Modification



Copy — Copy JWT from previous slide

Paste — Paste into jwt.io

Copy and paste — Copy and paste PUBLIC Key into jwt.io PUBLIC KEY box

Copy and paste — Copy and paste PRIVATE Key into jwt.io PRIVATE KEY box

Modify — Modify the contents of the claims section of the JWT

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAungIJYymAQvXgjgp7BdSH8I7rPC6LCCwCRq8jG5FUEpyq+7p
b/xDF/bOB46PIxH1QHwfCsbiaG9YhGnB4meVgWJCU6FDfRuAqz0kqf6JuifwaKuV
vWlSg7R7kAqzb9M9jvH97qH9/78EJ3u4/d37vzmHGPOro1r3DV2B0oy4/Fd4+12+
K7fxWPIWspzm+gBvPDkkvyRfC02DzG4V8vhS0TqfAs7MEAFVGHMOlcpDtD2thZOk
jKxs1PayomPgZrs4aRRbBb4jG2lpPHCYEkIi93xdJAeXXSbzROdnU9IsT/ZbSrF/
8v7ClpNbF1CQtzddLe7eGNXkSJjNExmKB/O2vQIDAQABAoIBAA1xECNGzYOnSyB5
tHnun26nJX6ctsruC0qIuR1FaK02RKhkt0KplFuSoLz2N5a/WWbN273+4rzFBAQ9
jGqp7WLPhrj5F8CZvi953536eYqoDOI6tjdac4aHeN3EC7XMrDQU+Sld1427Qw0m
k8oHGbnp53VywUVsDgGSY0SefMpB03DjsRh8TxVxpTMZQ/AayssvzKpvQzCdZWiP
X1SWCyHQuKa/Z9Ab4yg/an+MAqlV2y9KrGU5jxVDH2+kzc5NLBjfd839WDt6ADf/
J9avayy6Jfi/RXZ6G0qCKzgmIc+hKe8NC0fPziew6sefWNx9FDZCqDxOQoCnFGjB
6bg5ReECgYEA5EOLilvwZy0Pkyw9+SJr9unutsLABUZtaL0Nt9SFXfuOQYVeqelY
L2mkZxeiLF5XRFBZAIB24JPLUsFeV1JcLPA/ikCP5UjdYUHU3K+QHULQkGzpIqGl
yTDPTIXawxFeQWn1Wt7O/xqR9XZEYDhYWJyKKB2PWkGmE+S8K9oFuSECgYEA0SBl
0Ap/c6R0kjJd86I++sgJjAHXgg0hkGvSAcHENrjTbGFFPWusJ15xFaIHZ3mi6wkZ
qC9hRlxxMv8Yxm3FGc1qgGk29C0qJIJiaNMOCrzEsEIqwkoxUQ+Vy86/Ju5C1QGW
garvL697CccG4T2CcIGvko7N0U2laBVTYoi//h0CgYAZ2OArKZ/+Pub9lkvqMxCg
o/qo7UKLFl97NbUg9MqpSrvgBXcjrE2VCNRZ7B4sAf7FuIdrfCB566JhW44QOz4+
xHGdeRQSNX8D7U1qM+MQvSkawYpgpoc8Ue+XTazo28WdiJ8EzCgKUProHZ0+fALc
/dTTGA3MfZSNuh/oo0Z0oQKBgQDCBuIPnL3ViH7TacG7mwv913zsFoVh11cNzMui
76lh8CfRBYqdSvoF6NzY6mUePz/F+8J/Rb5l7rzkSMQuzoexweGPVI81C3ZvOz2c
7jy3/54pvqo3a2jIQcKEvsShlSwSvw9qLTMQNircznj2oAAW7Gv+eVpCWF0f7bFY
2XHwjQKBgQCXlqk4NH8OQSiPxREdQtzTwuPqES21SS5RecMkOGwE0KFKOxfzeWf/
+1RAmkQtULsI5pcTIvgBRIG/yrqX0MPbi7/S8O+44X8SNSEnVrguFoyfa+NLTI7i
jh0rKYlK3SBNE7p+7yGzLlNfl8GHR9929lp9bqsFQ+QLVqmIR0ah8A==
-----END RSA PRIVATE KEY-----
```

Burp Plugin for JWTs

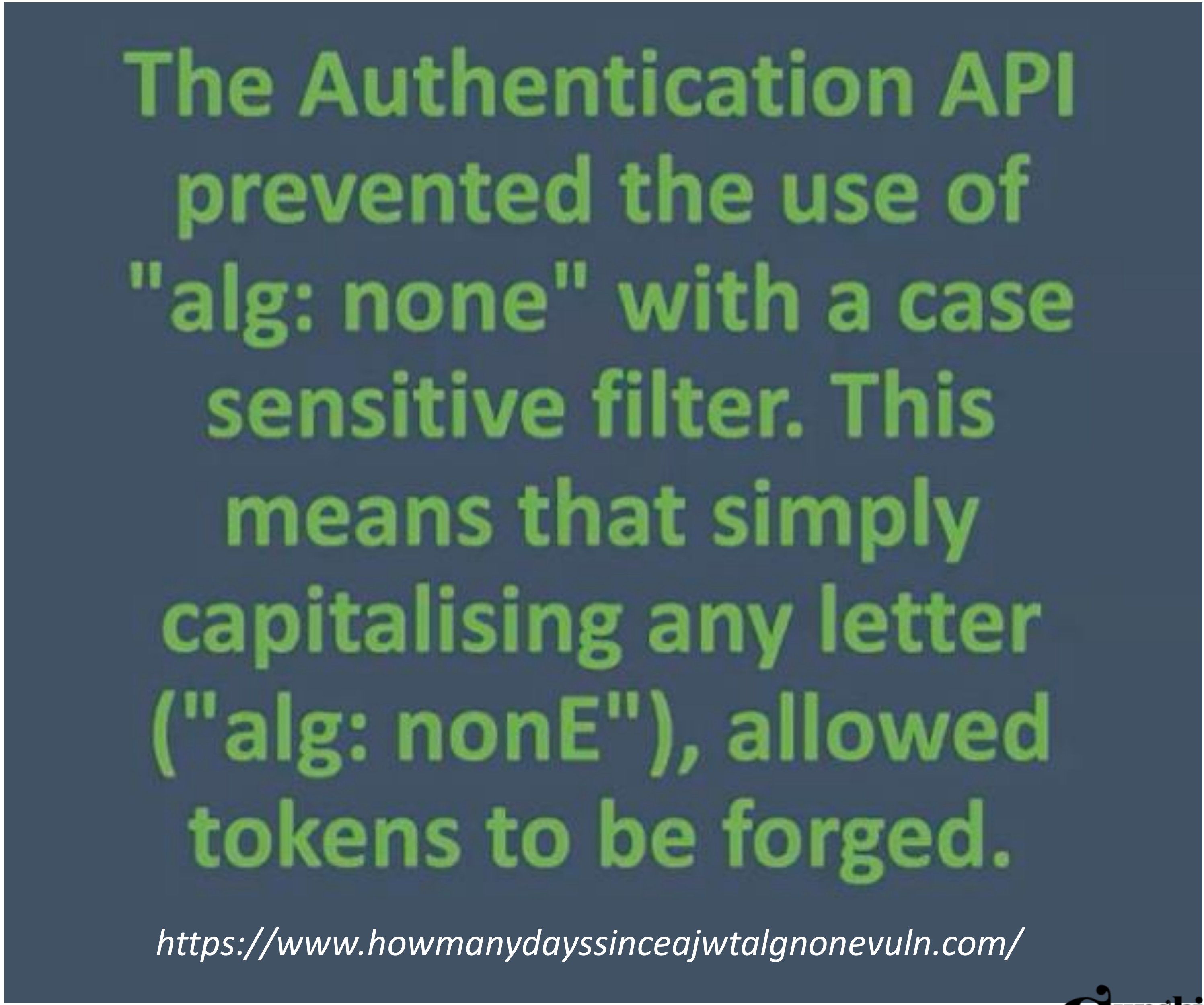
JWT Editor

- BApp Store

What is the None Algorithm?

- Included in original RFC specification
- Intended for debugging/testing only
- Many libraries implemented None algo and treated tokens using this algo as signed!
- When crafted by an attacker, can allow arbitrary account access

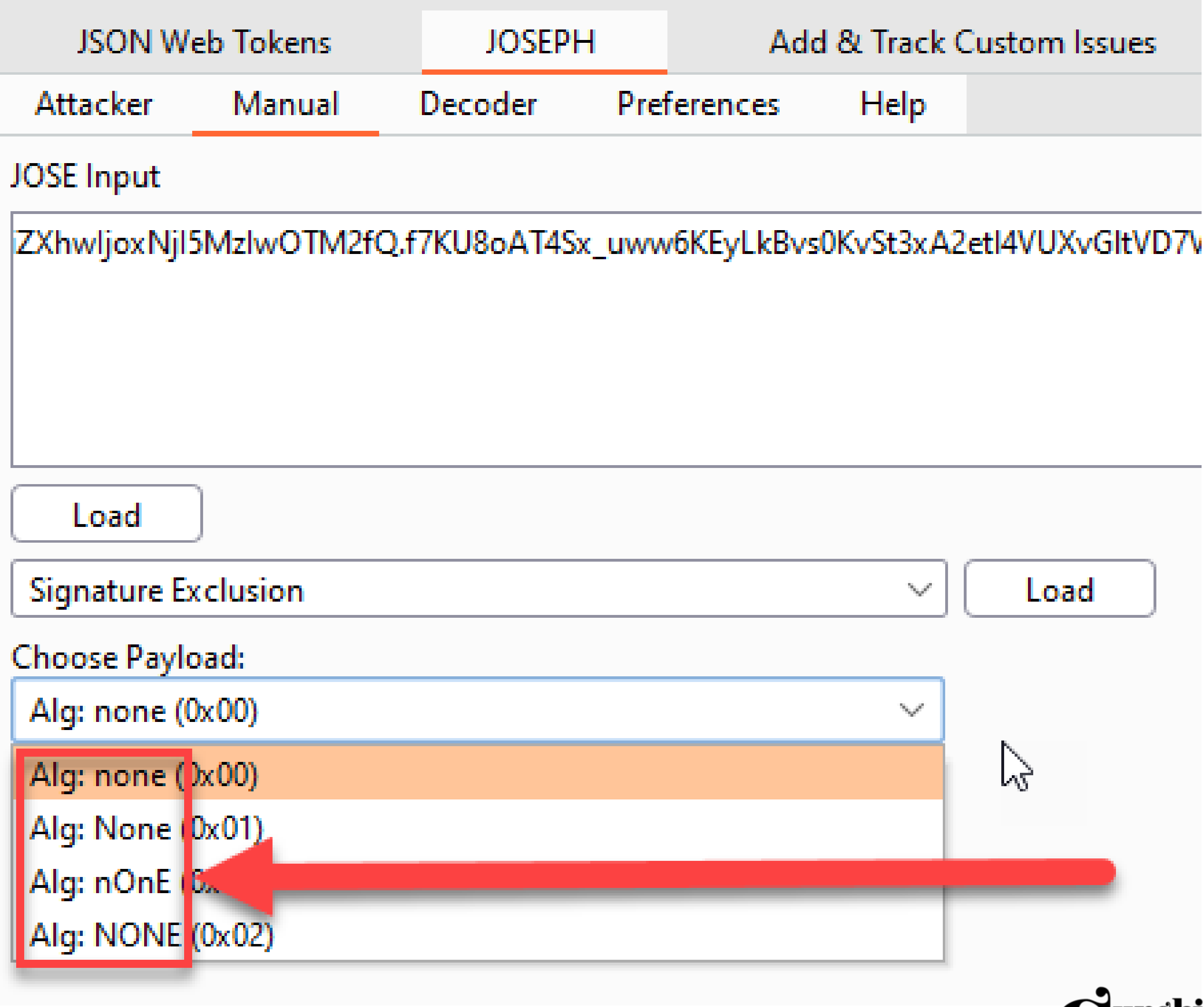
Does the None Attack still happen today?



The Authentication API prevented the use of "alg: none" with a case sensitive filter. This means that simply capitalising any letter ("alg: nonE"), allowed tokens to be forged.

https://www.howmanydayssinceajwtalgnonevuln.com/

Flavors of ‘None’ algo



ZXhwIjoxNjI5MzIwOTM2fQ.f7KU8oAT4Sx_uww6KEyLkBvs0KvSt3xA2etl4VUXvGItVD7V

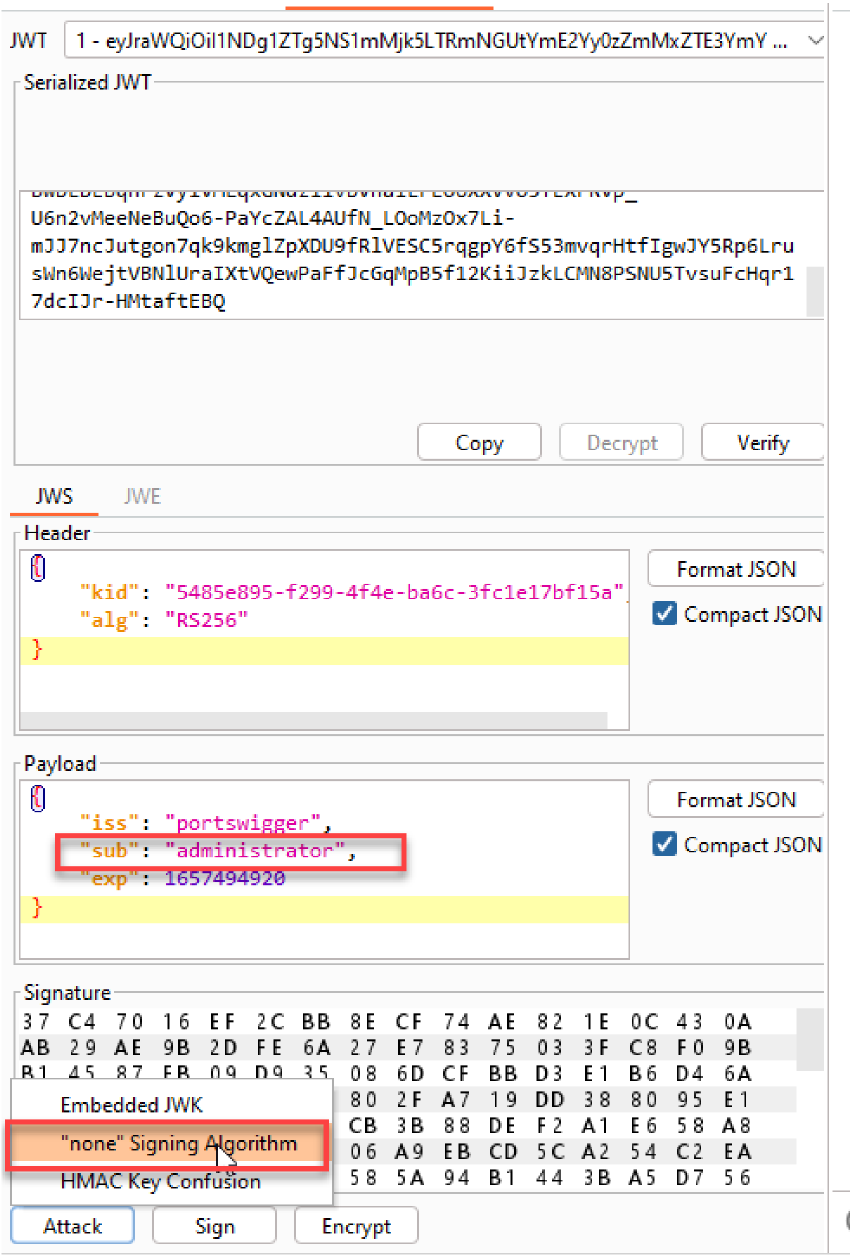
Signature Exclusion

Choose Payload: Alg: none (0x00)

JWT Pitfalls: Weak Validation

- **Unverified signature** – mods w/ no private key
- **Flawed signature verification** – none algo

Exercise 4-2: JWT authentication bypass via flawed signature verification



JWT 1 - eyJraWQiOil1NDg1ZTg5NS1mMjk5LTRmNGUtYmE2Yy0zZmMxZTE3YmY ...

Serialized JWT

```
U6n2vMeeNeBuQo6-PaYcZAL4AUfN_LOoMzOx7Li-
mJJ7ncJutgon7qk9kmglZpXDU9fRlVESC5rqgpY6fS53mvqrHtfIgwJY5Rp6Lru
sWn6WejtVBNlUraIXtVQewPaFfJcGqMpB5f12KiiJzkLCMN8PSNU5TvsuFcHqr1
7dcIJr-HMtaftEBQ
```

Header

```
{
    "kid": "5485e895-f299-4f4e-ba6c-3fc1e17bf15a",
    "alg": "RS256"
}
```

☑ Compact JSON

Payload

```
{
    "iss": "portswigger",
    "sub": "administrator",
    "exp": 1657494920
}
```

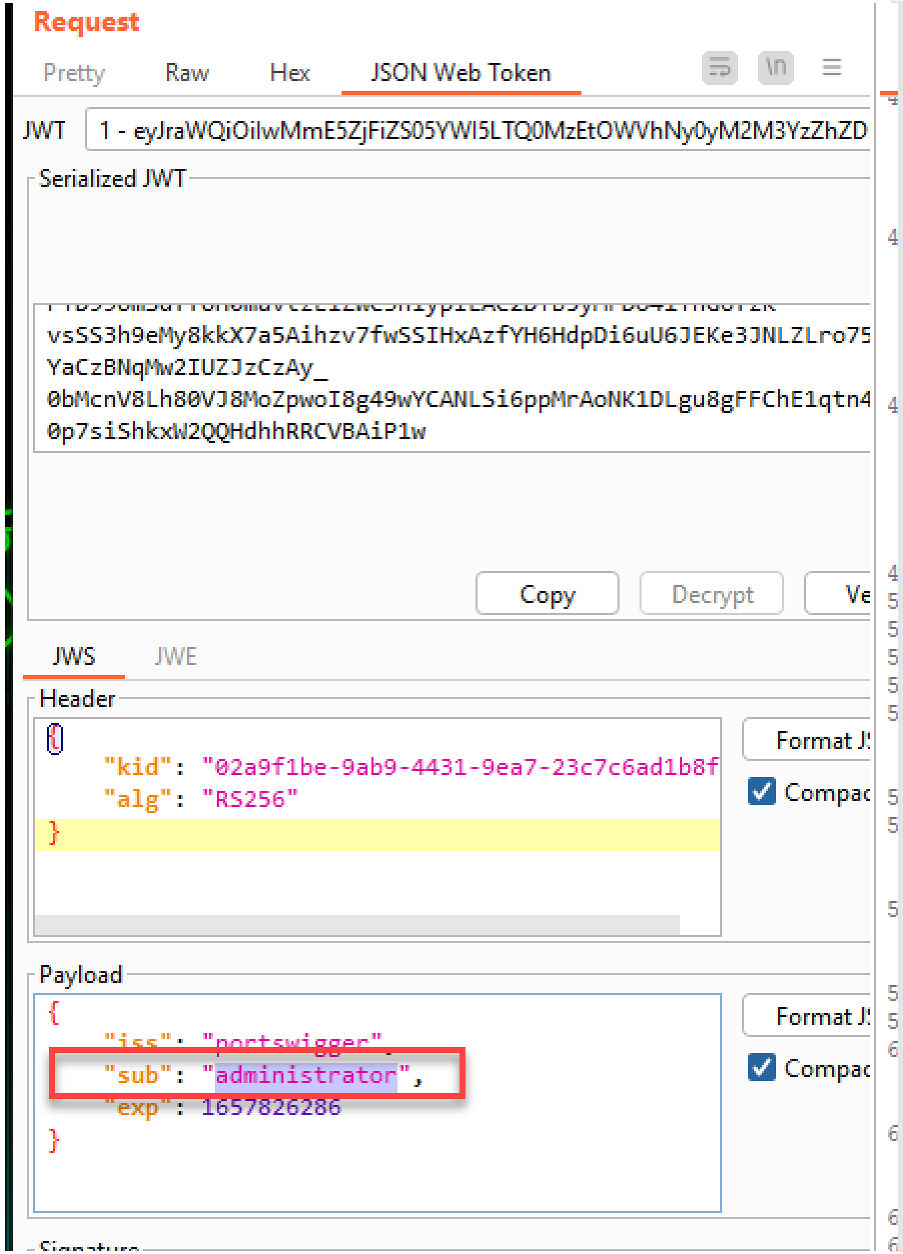
☑ Compact JSON

Signature

```
37 C4 70 16 EF 2C BB 8E CF 74 AE 82 1E 0C 43 0A
AB 29 AE 9B 2D FE 6A 27 E7 83 75 03 3F C8 F0 9B
B1 45 87 FB 09 D9 35 08 6D CF BB D3 E1 B6 D4 6A
                     80 2F A7 19 DD 38 80 95 E1
                     CB 3B 88 DE F2 A1 E6 58 A8
                     06 A9 EB CD 5C A2 54 C2 EA
                     58 5A 94 B1 44 3B A5 D7 56
```

- Look under **JWT Labs**
- Portswigger Web Security Lab: "**JWT authentication bypass via flawed signature verification**"

Exercise 4-3: JWT authentication bypass via unverified signature



JWT 1 - eyJraWQiOilwMmE5ZjFiZS05YWI5LTQ0MzEtOWVhNy0yM2M3YzZhZD

Serialized JWT

```
vsSS3h9eMy8kkX7a5Aihzv7fwSSIHxAzfYH6HdpDi6uU6JEKe3JNLZLro75
YaCzBNqMw2IUZJzCzAy_
0bMcnV8Lh80VJ8MoZpwoI8g49wYCANLSi6ppMrAoNK1DLgu8gFFChE1qtn4
0p7siShkxW2QQHdhhRRCVBAiP1w
```

Header

```
{
    "kid": "02a9f1be-9ab9-4431-9ea7-23c7c6ad1b8f
    "alg": "RS256"
}
```

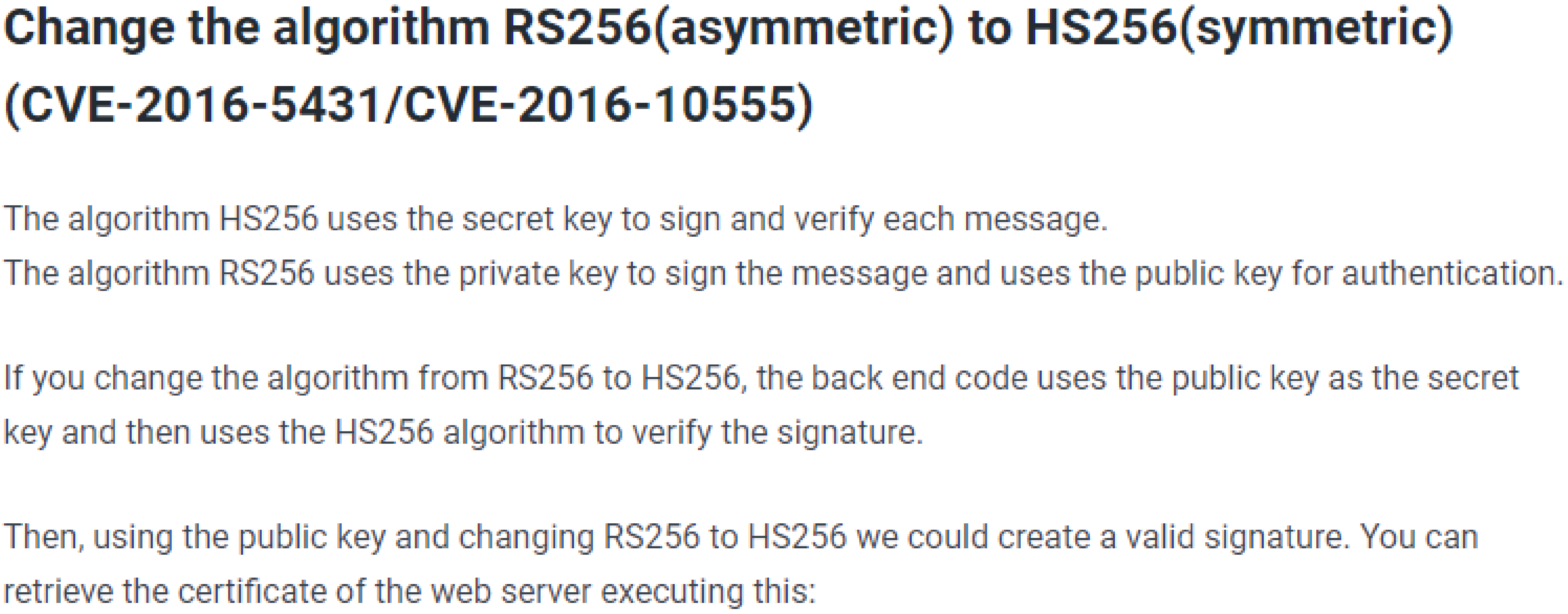
Payload

```
{
    "iss": "portswigger",
    "sub": "administrator",
    "exp": 1657826286
}
```

- Look under **JWT Labs**
- Portswigger Web Security Lab: “**JWT authentication bypass via unverified signature**”

What is the Key Confusion Attack?

- Two types:
    1. Using HS256 with server Public Key
    2. Using RS256 with Attacker crafted Public Key and Private Key



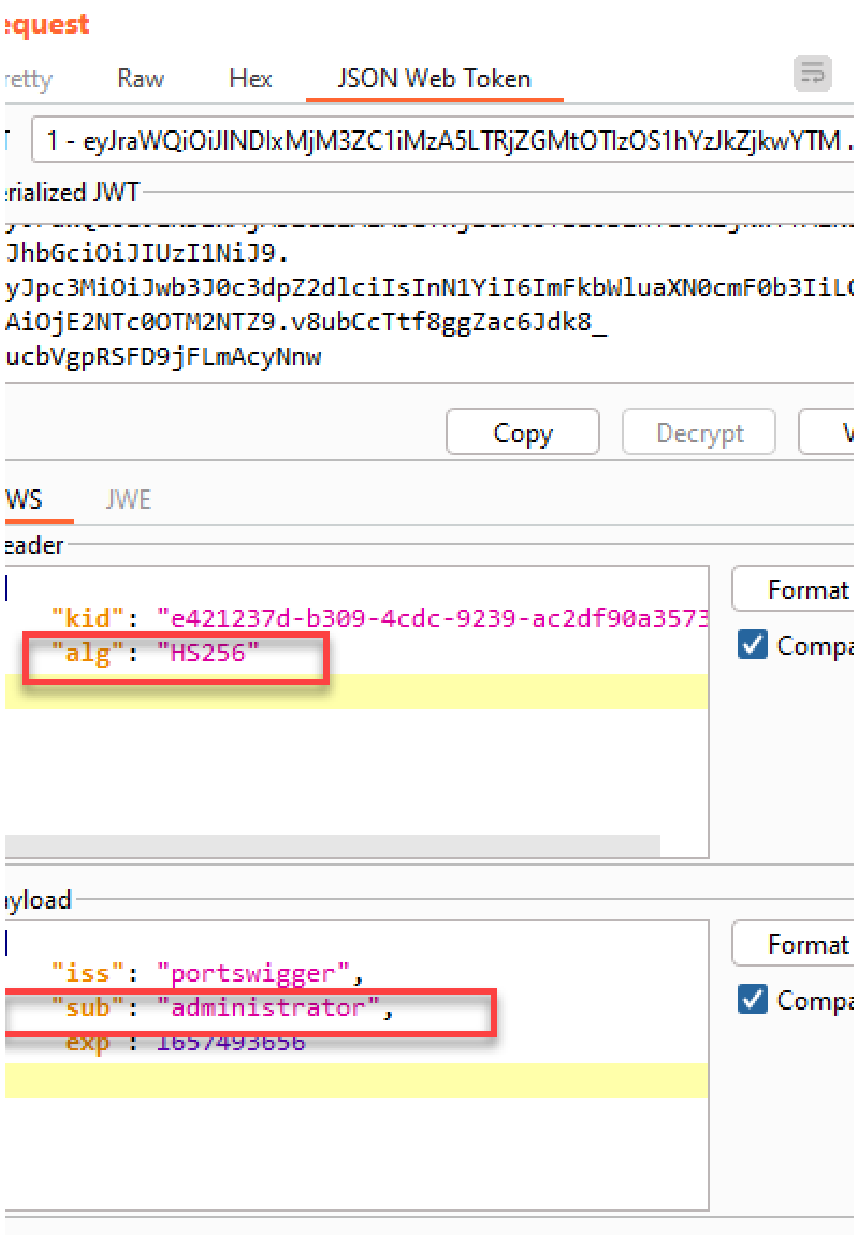
Change the algorithm RS256(asymmetric) to HS256(symmetric) (CVE-2016-5431/CVE-2016-10555)

The algorithm HS256 uses the secret key to sign and verify each message.
The algorithm RS256 uses the private key to sign the message and uses the public key for authentication.

If you change the algorithm from RS256 to HS256, the back end code uses the public key as the secret key and then uses the HS256 algorithm to verify the signature.

Then, using the public key and changing RS256 to HS256 we could create a valid signature. You can retrieve the certificate of the web server executing this:

Exercise 4-4: JWT authentication bypass via algorithm confusion



```
1 - eyJraWQiOiJINDIxMjM3ZC1iMzA5LTRjZGMtOTIzOS1hYzJkZjkwYTM .
```

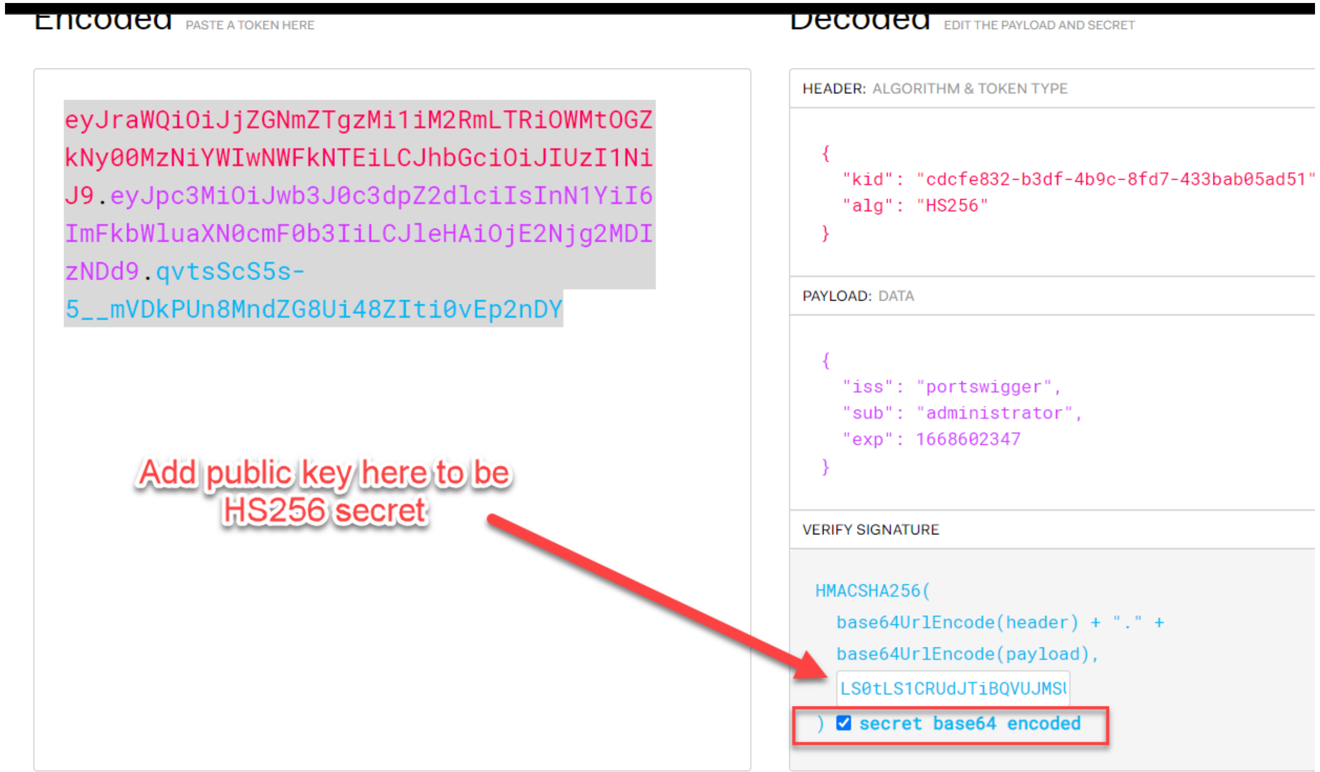
```
JhbGciOiJIUzI1NiJ9.
yJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6ImFkbWluaXN0cmF0b3IiL
AiOjE2NTc0OTM2NTZ9.v8ubCcTtf8ggZac6Jdk8_
ucbVgpRSFD9jFLmAcyNnw
```

```
{
  "kid": "e421237d-b309-4cdc-9239-ac2df90a3573
  "alg": "HS256"
```

```
{
  "iss": "portswigger",
  "sub": "administrator",
  "exp": 1657493656
```

- Portswigger Web Security Lab: “**JWT authentication bypass via algorithm confusion**”

If Burp Plugin is not working, do manually:



Encoded PASTE A TOKEN HERE

```
eyJraWQiOiJjZGNmZTgzMi1iM2RmLTRiOWMtOGZ
kNy00MzNiYWIwNWFkNTEiLCJhbGciOiJIUzI1Ni
J9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6
ImFkbWluaXN0cmF0b3IiLCJleHAiOjE2Njg2MDI
zNDd9.qvtsScS5s-
5__mVDkPUn8MndZG8Ui48ZIti0vEp2nDY
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid": "cdcfe832-b3df-4b9c-8fd7-433bab05ad51"
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "iss": "portswigger",
  "sub": "administrator",
  "exp": 1668602347
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  LS0tLS1CRUdJTiBQVUJMSU
) ☑ secret base64 encoded
```

Add public key here to be HS256 secret

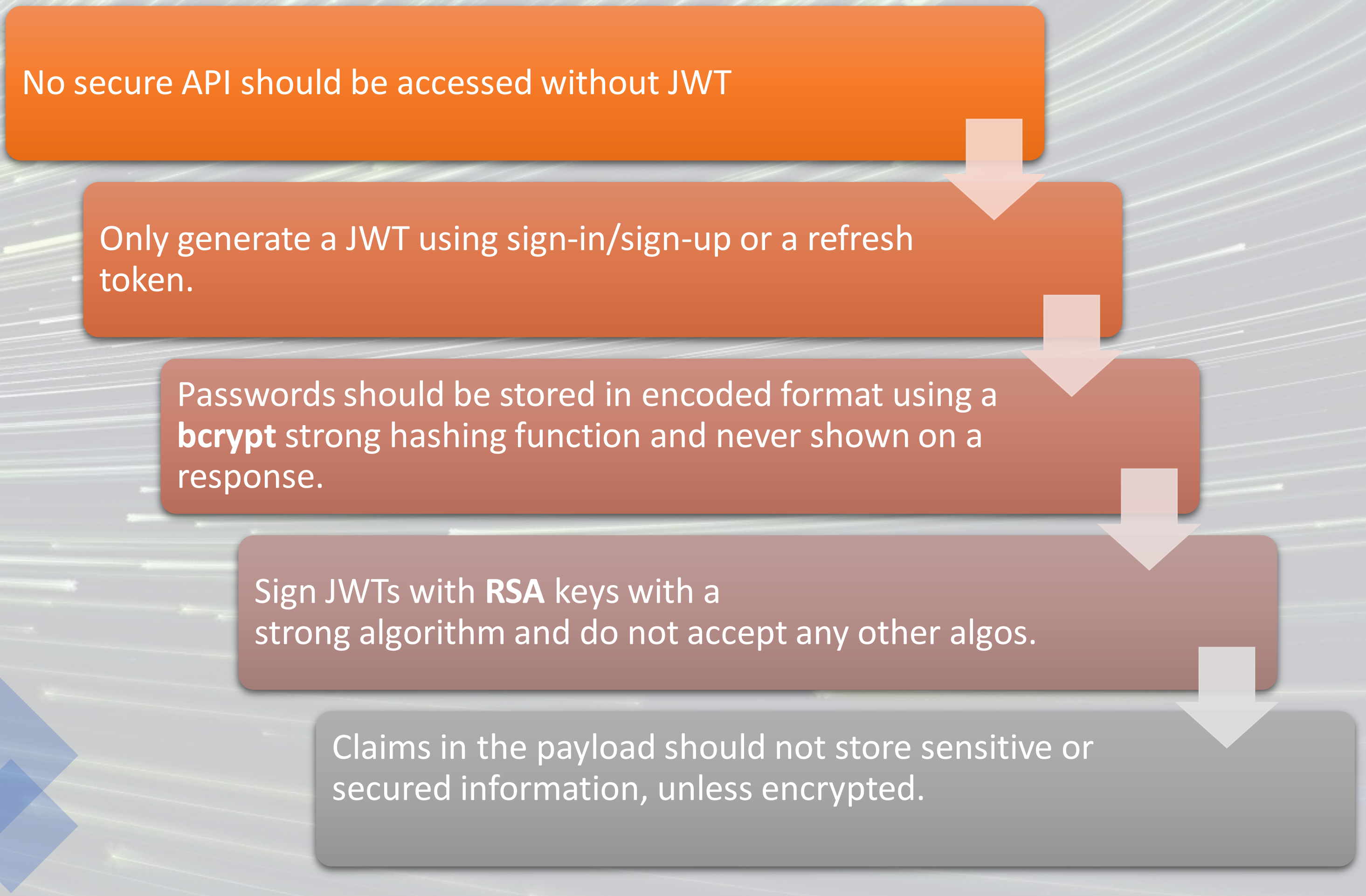
- Copy token from Raw
- Paste into jwt.io
- Copy/paste base64 public key into secret textbox
- Check the box for base64 secret
- Copy token into Raw request

Case Study: $23,000 for Authentication Bypass & File Upload & Arbitrary File Overwrite

- **JSON Web Token (JWT)** for the authentication mechanism
- JavaScript code references an administrative realm called “**test-dashboard”;** he changed the default setting of realm to be **test-dashboard** enabling access to the admin console.

https://medium.com/@h4x0r_dz/23000-for-authentication-bypass-file-upload-arbitrary-file-overwrite-2578b730a5f8

Rules for Securing REST APIs using JWTs



1. No secure API should be accessed without JWT
2. Only generate a JWT using sign-in/sign-up or a refresh token.
3. Passwords should be stored in encoded format using a **bcrypt** strong hashing function and never shown on a response.
4. Sign JWTs with **RSA** keys with a strong algorithm and do not accept any other algos.
5. Claims in the payload should not store sensitive or secured information, unless encrypted.