SANS

JWT Security Issues

Adrien de Beaupré

About Me

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…

What are JWT

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.

JWT Parts

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
CJleHAiOjE2MDI2OTc4MzR9.vqY5OE1oMPF6MlN98vDk0osrtTWWTF
y68O1zf1oxcvw
```

Creating JWT in Python

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)
```

JWT Security Issues

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

Identifying the Algorithm

If the server is signing the JTW 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 None Algorithm

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

eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VyX2lkIjoxLCJleHAiOjE2MDI2MTA4OTJ9.

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

Cracking HS256 Secret

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
```

Algorithm Substitution

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

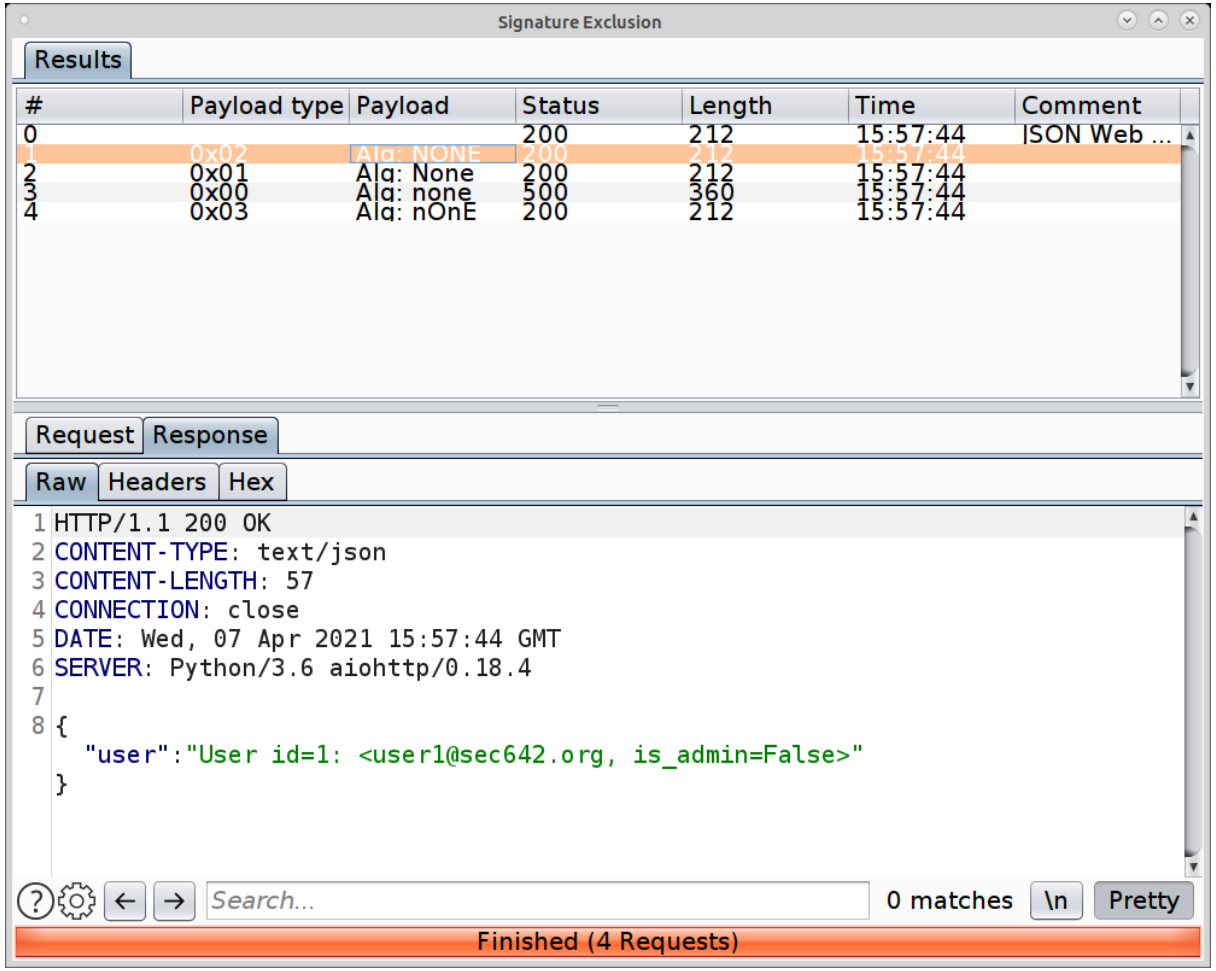
Or use Python to create a new JWT with the public key

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

Signature Exclusion Attacks

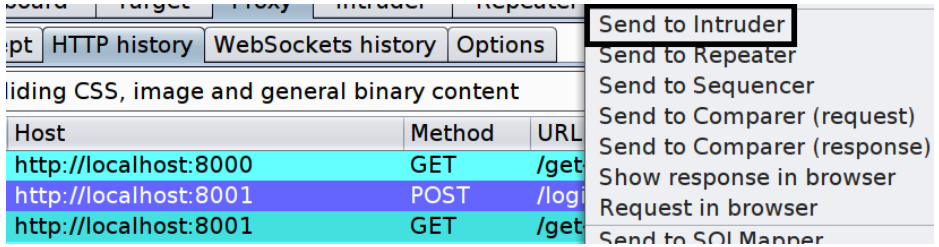
The Burp extension called JOSEPH (JSON Web Token Attacker in Burp Store) can perform what it calls a Signature Exclusion attack

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

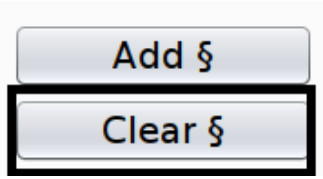


None Algorithm With Intruder

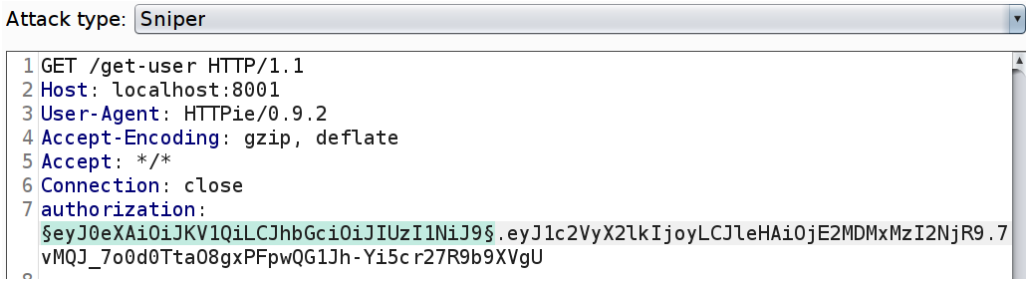
1 ➡



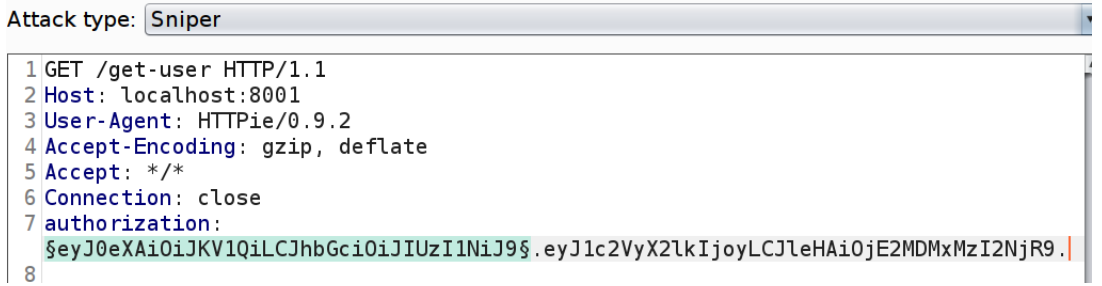
2 ➡



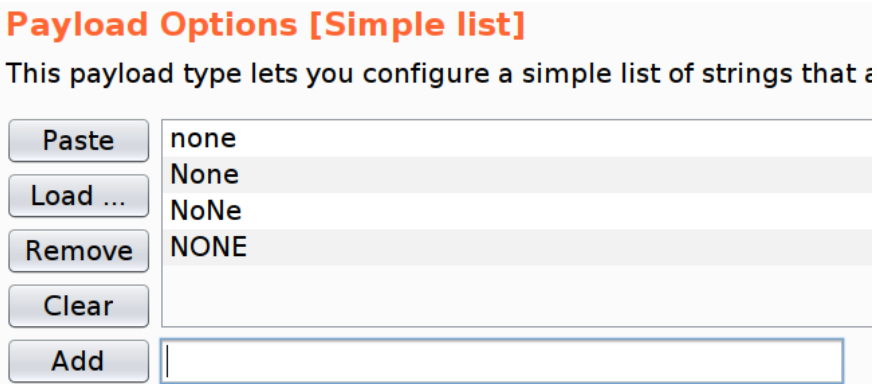
3 ➡



4 ➡



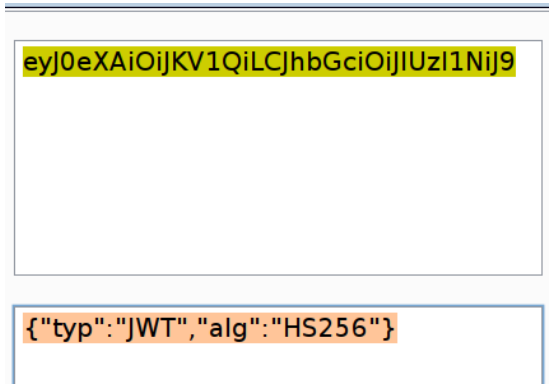
← 5



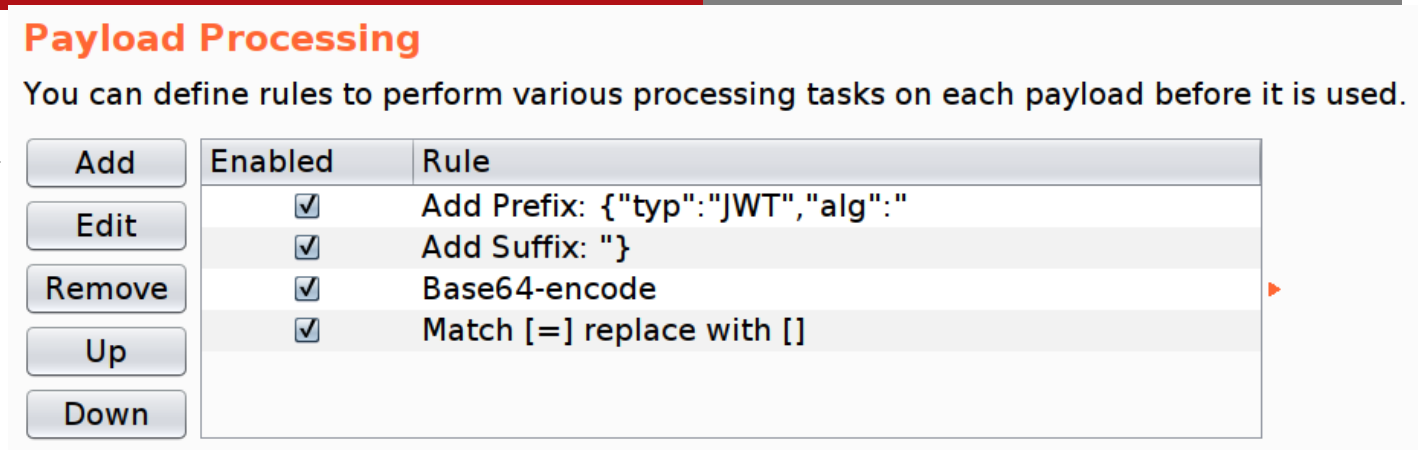
← 6

None Algorithm With Intruder Continued

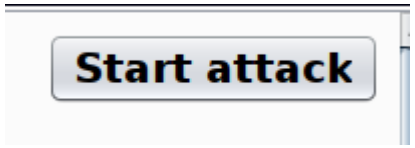
7 ➡



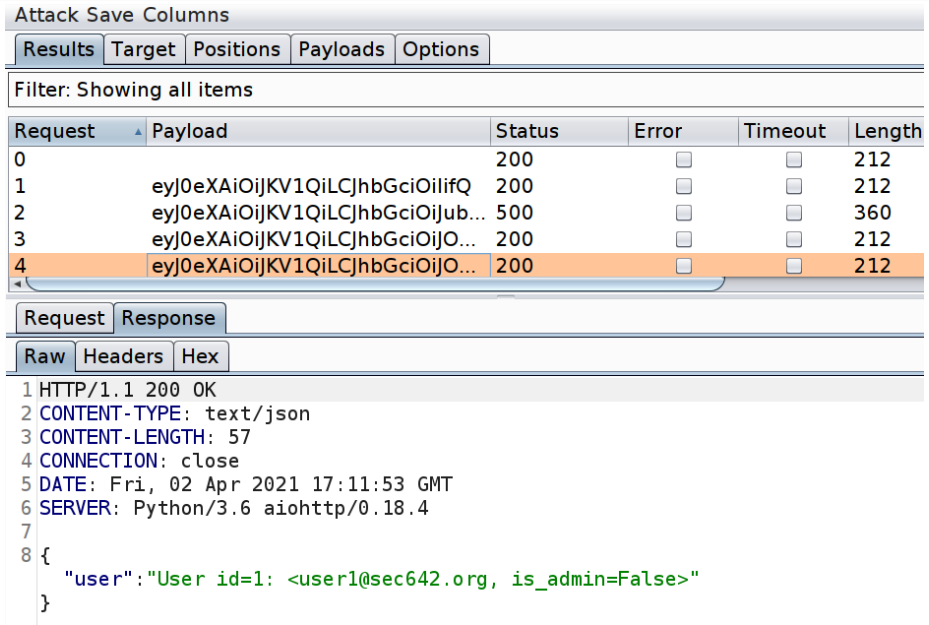
8-12 ➡



13 ➡

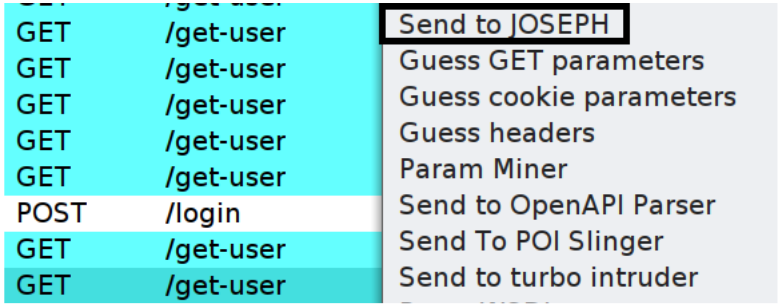


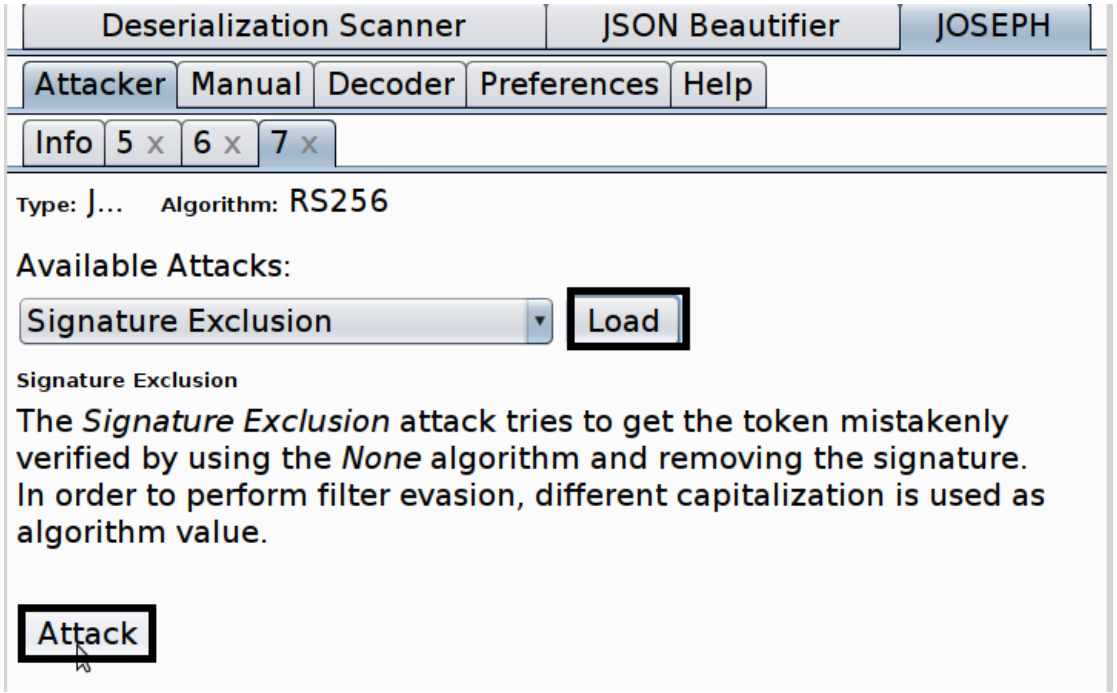
Success ➡



JOSEPH Signature Exclusion

- 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





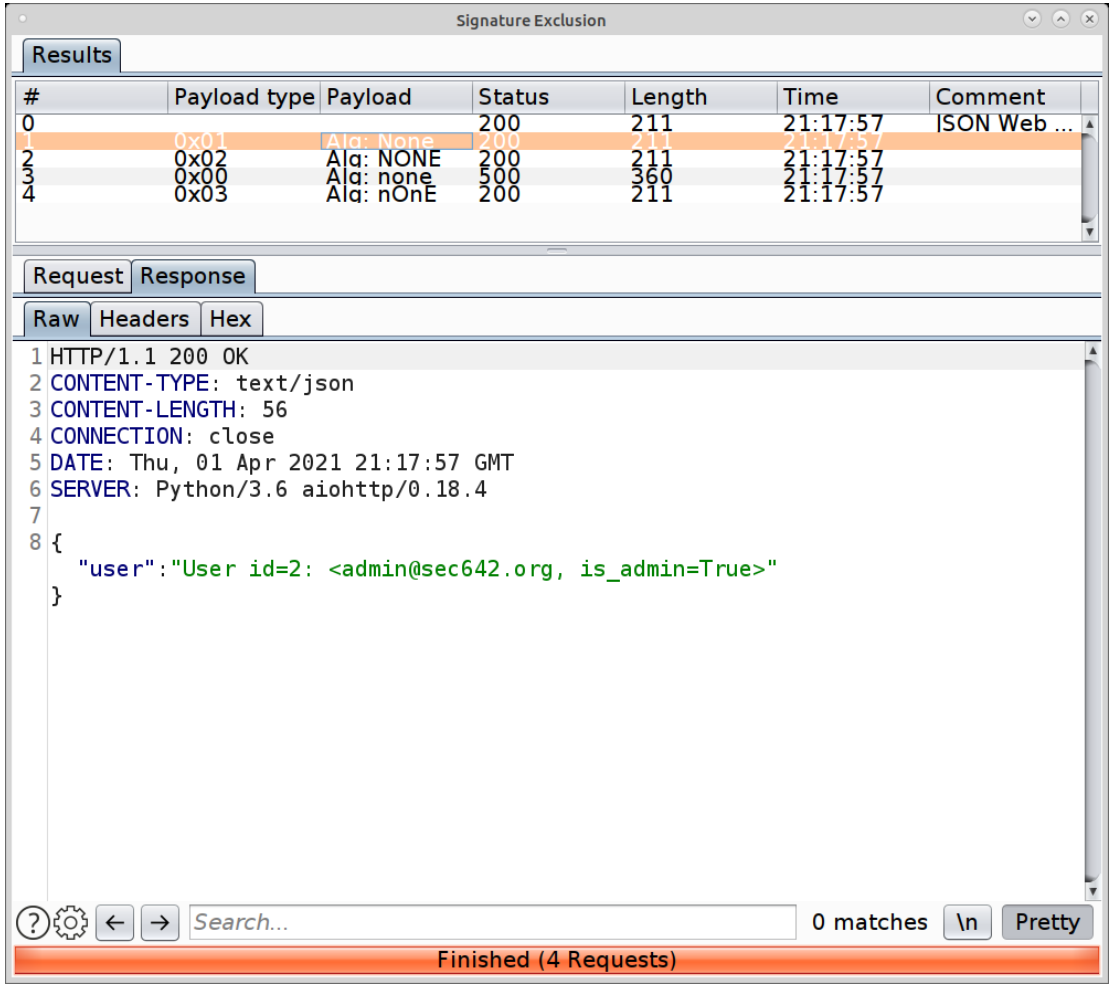
Examining Signature Exclusion Attacks

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



JOSEPH Key Confusion

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

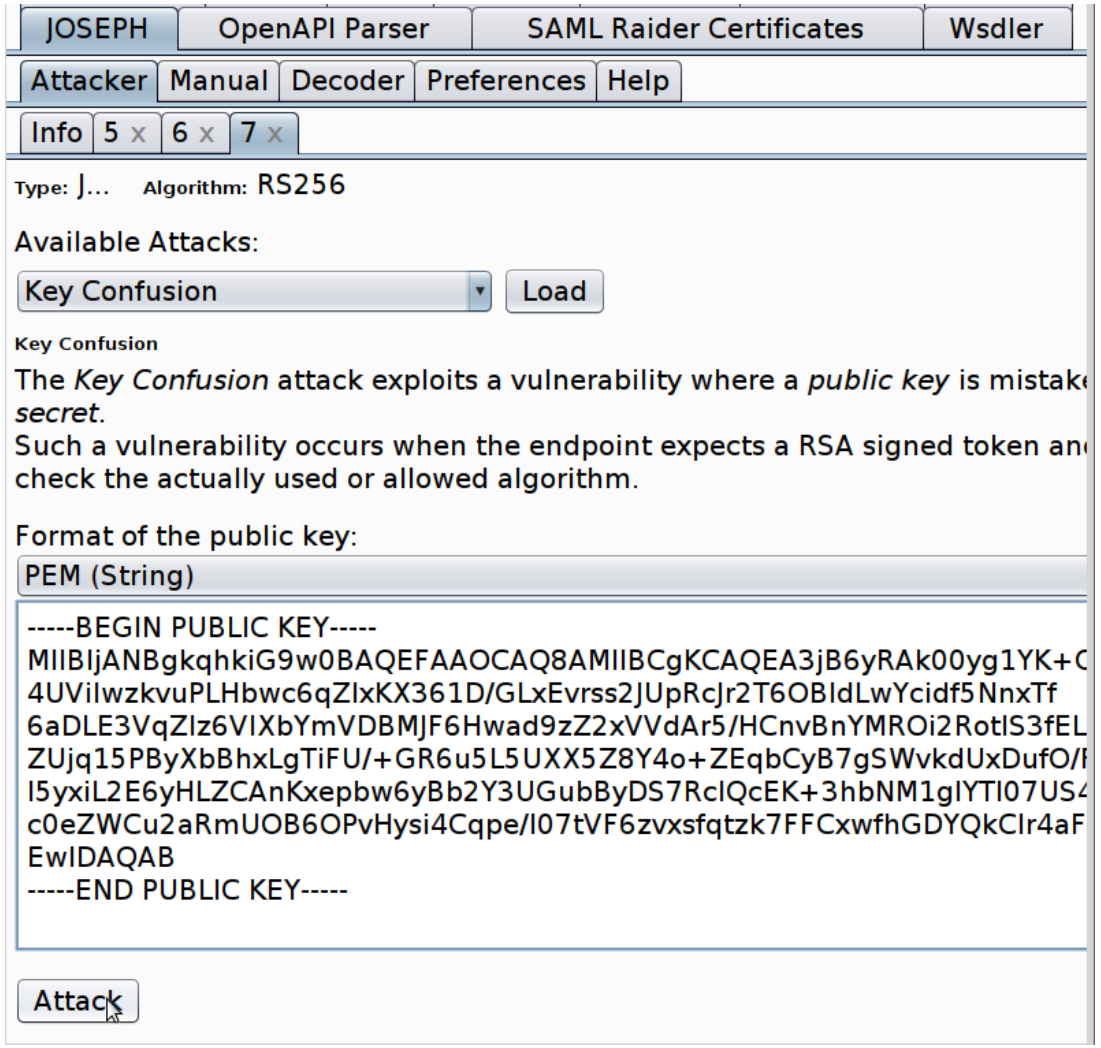
On the Attacker tab select Key Confusion

Click on Load

Paste the public key

Then click on Attack

The Attacker window will open up, similar to Intruder



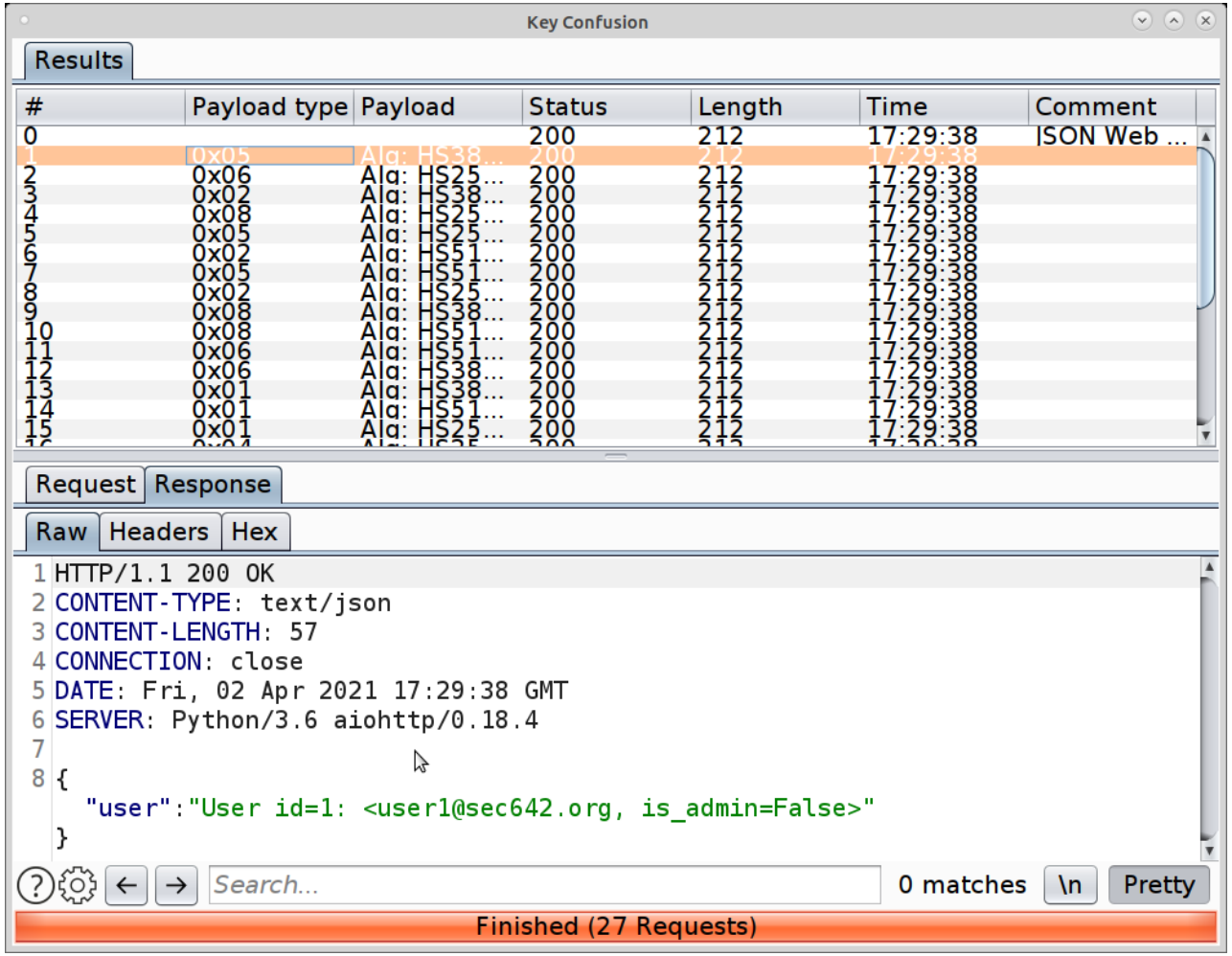
Key Confusion Attacks

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



Conclusion

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.

References

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

References

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

Course Resources and Contact Information

AUTHOR CONTACT
Adrien de Beaupré
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
REGISTRATION:
registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org