JSON Web Token (JWT)

A security perspective

Tim Kent

TSS Cyber

January 2019

About me

- Senior Security Consultant at TSS Cyber
- Member of CrikeyCon CTF Creation Team
- SecTalks Slack: `timk`
- Twitter: `@__timk`

JWT primer/anatomy

Three `Base64` (well, *technically* "`Base64-URL`") encoded chunks separated by full stops:

```
Authorization: Bearer xxx.yyy.zzz
```

- `xxx` = headers
- `yyy` = payload (`key:value` claims)
- `zzz` = signature

Decoded example:

```
Headers = {
    "alg" : "HS256",
    "typ" : "JWT"
}
Payload = {
    "sub" : "timk",
    "jti" : "ef804cf0-a95b-4a07-ac54-174101284507",
    "nbf" : 1545265273,
    "exp" : 1545267073
}
Signature = "TP3KspMyxHtaxjoQl4UlzqKOay9dmGp_zEgRW69NfqU"
```

HMAC vs pub/priv (asymmetric)

HMAC (static key):

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

RSA pub/priv (asymmetric):

```
Headers = {
    "alg" : "RS256",
    "typ" : "JWT"
}
```

Stateless vs stateful

Stateless JWTs:

- Stateless JWTs can scale well horizontally (no need for state database).
- Stateless also means you can't log out or invalidate a token.
- Good use case - single use (and short lived) token to access a file from one of hundreds of download servers in a CDN.
- Bad use case - session tokens.

Stateful JWTs:

- Contain a unique identifier, state is stored server side much like traditional session tokens.

Cookies vs local storage

This section is not specific to JWT!

Cookies

- When using JWTs in a cookie with `HttpOnly`, the same rules apply as any other session identifier.
- Still need CSRF token to prevent replaying (even with `HttpOnly`).
- Consider the `SameSite` flag as second line of defence against CSRF.

The following would not be available via AJAX, so XSS is not possible:

```
Set-Cookie: Auth=xxx.yyy.zzz; Secure; HttpOnly
```

Local storage

- Claims that local storage is more secure than non-`HttpOnly` cookies are incorrect.
- Local storage is accessed using JavaScript, so vulnerable to XSS in same way.

```
> localStorage.jwt = 'xxx.yyy.zzz'
> alert(localStorage.jwt)
```

- You can't use CORS (Cross-Origin Resource Sharing) for `localStorage` access.

Be careful using JWT

- Use a mature validation library rather than rolling your own.
- Always explicitly set/check algorithm.
- Validate all your claims - use not before (`nbf`) claim to prevent scenarios where issuer had time set 5 years in the future, for example.
- Take care using JWTs for different services from same issuer.
- RSASSA private key can also verify.
- Use RSASSA/ECDSA over HMAC (anyone with HMAC secret can create JWTs).
- Use a very long random key if you do use HMAC (they can be cracked offline with Hashcat).
- Use JWE if claims contain sensitive info.

Reasons against JWT

- JWTs give up more info than server side sessions (expiry time plus whatever is in your private claims).
- Lots of parsing has to happen server side.
- Session IDs are usually much smaller than JWTs.
- Less mature/less tested tech.

JWT as session token

Using JWTs for sessions is no easier:

- You need to either keep track of valid JTIs, or a blacklist of invalidated JTIs.
- Then you need to run Redis or something else to keep state, which defeats your advantages.
- Check your session invalidation method actually works!
- Strongly consider sticking to your web framework's tried and tested session handling.

Walkthroughs

- Three challenges.
- All real-world scenarios.
- Doing this live so please cross your fingers!

Walkthrough of JWTease 1

Signature stripping

Some early JWT libraries accepted this:

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

This is still a “valid” JWT.

Walkthrough of JWTease 2

Algorithm confusion

Handing back literal public key as HMAC secret:
Receive `"alg" : "RS256"` but send back `"alg" : "HS265"`

Walkthrough of JWTease 3

Weak secret

This time using HMAC mode with a weak secret.

Further reading

https://tools.ietf.org/html/rfc7519
http://cryto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/
Further exercises here:
https://pentesterlab.com/exercises/