JWT Security

Dr. Philippe De Ryck

https://Pragmatic Web Security.com
Abstract

A JSON Web Key (JWK) is a JavaScript Object Notation (JSON) data structure that represents a cryptographic key. This specification also defines a JWK Set JSON data structure that represents a set of JWKs. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification and IANA registries established by that specification.
Abstract
A JSON data structure that represents a set of cryptographic keys. This specification also defines identifiers for use with this specification. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification and IANA registries established by that specification.
I am Dr. Philippe De Ryck

Founder of Pragmatic Web Security
Google Developer Expert
Auth0 Ambassador / Expert
SecAppDev organizer

I help developers with security

- Academic-level security training
- Hands-on in-depth online courses
- Security advisory services

https://pragmaticwebsecurity.com
1. JWT Signature Schemes
2. JWT Key Management
3. Ridiculous JWT vulnerabilities
4. Quiz & Summary
JWT Signature Schemes
By default, JWTs are ...

A. Base64 encoded
B. Signed
C. Encrypted
eyJhbGciOjI1NzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTQxNiwiaWF0IjoxNDEwMjQ0MzA1fQ.s0D9ZvIhWf3VQ8S
LIPoH_ytKd1sDhVPmT7gP6H

<table>
<thead>
<tr>
<th>HEADER: ALGORITHM &amp; TOKEN TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{</td>
</tr>
</tbody>
</table>
|   "alg": "HS256",
|   "typ": "JWT"
| } |

<table>
<thead>
<tr>
<th>PAYLOAD: DATA</th>
</tr>
</thead>
</table>
| "user": "1",
| "tenant": "d8cf3fa301a34c968502a7051bfdc0a8",
| "restaurant": "5e4fd699d6b84cd8b1bee5f0428c0918",
| "tenant_name": "The Burger Group",
| "restaurant_name": "Burger Master"
| |

<table>
<thead>
<tr>
<th>VERIFY SIGNATURE</th>
</tr>
</thead>
</table>
| HMACSHA256(
|   base64UrlEncode(header) + "." +
|   base64UrlEncode(payload),
|   SuperSecretHMACkey |
| ) | □ secret base64 encoded |
Using the `java-jwt` library to decode a JWT

```java
String token = getTokenFromUrl(); //"eyJhbGciOiJIU...";
try {
    DecodedJWT jwt = JWT.decode(token);
} catch (JWTDecodeException exception) { //Invalid token
}
```

The `decode` function returns the claims of the JWT, but does not verify the signature.

Using the `java-jwt` library to verify the HMAC and decode a JWT

```java
String token = getTokenFromUrl(); //"eyJhbGciOiJIU...";
try {
    Algorithm algorithm = Algorithm.HMAC256("secret");
    JWTVerifier verifier = JWT.require(algorithm).build();
    DecodedJWT jwt = verifier.verify(token);
} catch (JWTVerificationException exception) { //Invalid signature/claims
}
```

The verify function on a verifier will only return the claims when the signature is valid.
**Generate HMAC**

`data` → [Process] → `yxzN...sFno=`

**SECRET KEY**

`SECRET KEY` → **HMAC**

**Verify HMAC**

`data` → [Process] → `yxzN...sFno=` → `= =`

Message differs from the one that was signed

Message is the same as the one that was signed
Your secret should be more random, and should not be published on a Powerpoint slide
Your secret should be more random, and should not be published on a Powerpoint slide.

A key of the same size as the hash output (for instance, 256 bits for "HS256") or larger MUST be used with this algorithm.
**Asymmetric JWT Signatures**

**Generate Signature**

1. **Data**: Input the data to be signed.
2. **Private Key**: Use the private key to generate the signature.
3. **Signature**: The generated signature is appended to the data.

**Verify Signature**

1. **Public Key**: Use the public key to verify the signature.
2. **Verification Result**:
   - If the signature is valid, the message is the same as the one that was signed.
   - If the signature is invalid, the message differs from the one that was signed.

---

@PhilippeDeRyck
A DISTRIBUTED JWT USE CASE

1. Request security token
2. Generate a JWT and sign with the private key
3. A JWT security token
4. Call API with the JWT
5. Use public key to verify JWT signature and make authorization decision
6. Response
JWT Signature Schemes

JWT Key Management

Ridiculous JWT vulnerabilities

Quiz & Summary
JWT Key Management
Which of these key distribution mechanisms are used by JWTs?

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Static deployment (e.g., in an environment file)</td>
</tr>
<tr>
<td>B</td>
<td>Embedding the key in a JWT</td>
</tr>
<tr>
<td>C</td>
<td>Embedding the location of the key in a JWT</td>
</tr>
<tr>
<td>D</td>
<td>Not using keys at all</td>
</tr>
</tbody>
</table>
How does the receiver know which key to use to verify the signature?

What if there are multiple possible keys?
The reserved `kid` claim represents a key identifier, helping the receiver to find the right key. Useful to retrieve a key from a centralized key store.

```json
{
    "alg": "RS256",
    "typ": "JWT",
    "kid": "d8cf3fa301a34c968502a7051bfdc0a8"
}
```
The reserved jku claim represents a URL pointing to a set of public keys that can be used to verify the signature. Since these keys are publicly available, the receiver can retrieve them from this location.

The kid claim can be used to select the right key from the key set.
The reserved \texttt{x5u} claim represents the location of an X.509 certificate (TLS certificate).

Since the certificate is publicly available, the receiver can retrieve it from this location.
Without proper verification, a gullible backend will retrieve the attacker's keys and use them to verify a malicious JWT token. This setup allows an attacker to provide arbitrary JWT tokens that will be considered valid, causing a major vulnerability.
.well-known/openid-configuration

```json
{
  "issuer": "https://sts.restograde.com/",
  "authorization_endpoint": "https://sts.restograde.com/authorize",
  "token_endpoint": "https://sts.restograde.com/oauth/token",
  "device_authorization_endpoint": "https://sts.restograde.com/oauth/device/code",
  "userinfo_endpoint": "https://sts.restograde.com/userinfo",
  "mfa_challenge_endpoint": "https://sts.restograde.com/mfa/challenge",
  "registration_endpoint": "https://sts.restograde.com/oidc/register",
  "revocation_endpoint": "https://sts.restograde.com/oidc/revoke",
  "scopes_supported": [...],
  "response_types_supported": [...],
  "code_challenge_methods_supported": [...],
  "response_modes_supported": [...],
  "subject_types_supported": [...],
  "id_token_signing_alg_values_supported": [...],
  "token_endpoint_auth_methods_supported": [...],
  "claims_supported": [...]
}
```
RIDICULOUS JWT VULNERABILITIES
{ "alg": "none", 
"typ": "JWT", 
"kid": "Ae42SFaYAECQQ"
}

{ "file_id": "502a7051bfdc0a8d8cf3fa301a34c968", 
"sub": "5e4fd699d6b84cd8b1bee5f0428c0918", 
"iss": "https://sts.restograde.com", 
"aud": "https://files休息格雷幼", 
"iat": 1521314123, 
"exp": 1621314123 }
Critical Vulnerabilities Affect JSON Web Token Libraries

Author:
Chris Brook
April 1, 2015 / 2:58 pm

3:30 minute read

Share this article:

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.

JSON Web Token Validation Bypass in Auth0 Authentication API

Ben discusses a JSON Web Token validation bypass issue disclosed to Auth0 in their Authentication API.

https://insomniasec.com/blog/auth0-jwt-validation-bypass
It has been 90 days since the last alg=None JWT vulnerability.

The UK NHS COVID-19 contact tracing app for Android was accepting alg=None tokens in venue check-in QR codes. Write-up here.
JOSEPH - JavaScript Object Signing and Encryption Pentesting Helper

This extension helps to test applications that use JavaScript Object Signing and Encryption, including JSON Web Tokens.

Features

- Recognition and marking
- JWS/JWE editors
- (Semi-)Automated attacks
  - Bleichenbacher MMA
  - Key Confusion (aka Algorithm Substitution)
  - Signature Exclusion
- Base64url en-/decoder
- Easy extensibility of new attacks

Author  Dennis Detering
Version  1.0.2
Rating  ★★★★★
Popularity  

Last updated  08 February 2019

You can install BAApps directly within Burp, via the BAApp Store feature in the Burp Extender tool. You can also download them from here, for offline installation into Burp.
JWT Signature Schemes

JWT Key Management

Ridiculous JWT vulnerabilities

Quiz & Summary
SUMMARY
Internet Engineering Task Force (IETF)
Request for Comments: 8725
BCP: 225
Updates: 7519
Category: Best Current Practice
ISSN: 2070–1721

JSON Web Token Best Current Practices

Abstract

JSON Web Tokens, also known as JWTs, are URL-safe JSON-based security tokens that contain a set of claims that can be signed and/or encrypted. JWTs are being widely used and deployed as a simple security token format in numerous protocols and applications, both in the area of digital identity and in other application areas. This Best Current Practices document updates RFC 7519 to provide actionable guidance leading to secure implementation and deployment of JWTs.
BEST PRACTICES JWT SECURITY

• Choose the proper signature algorithm
  • HMACs are only useful internally in an application
  • All other scenarios should rely on asymmetric signatures
    • Make sure you have a secure way to obtain the public keys of the sender

• Follow JWT security recommendations
  • Explicitly type your JWTs
  • Use strong signature algorithms
  • Use reserved claims and their meaning

• Explicitly verify the security of the backend application
  • Libraries should be actively supported and up to date
  • JWTs with none signatures should be rejected case-insensitively
  • JWTs with invalid signatures should be rejected
This online course condenses dozens of confusing specs into a crystal-clear academic-level learning experience.
Thank you for watching!

Connect on social media for more in-depth security content

@PhilippeDeRyck
/in/PhilippeDeRyck