JSON Web Tokens

For Fun and Profit

ChiPy June 2023 __main__ meeting · @eevelweezel

What is OAuth2, anyway?

RFC 6749… “The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.”

- Two flows:
  - Application, where we’ve already been granted permissions by a user or service
  - Redirect, where a user goes to a login portal to grant permissions, then gets redirected back (think “login with Google”)
- Both flows allow an application to interact with a service on a user’s behalf, under specific terms, without obtaining the user’s credentials

Authentication (authn)

Authentication is verifying the identity of a user. In Oauth2, this step is where you submit your credentials to request an access token, and specify the grants you're seeking (i.e., authorization code, refresh or ID token, etc.). If you aren't recognized, you'll get a 403.

Authorization (authz)

Authorization is verifying a user's privileges. In OAuth2, this step is where trade your access token for a bearer token, and possibly, refresh or id tokens. An authorization error is indicated by status 401.

I have a bearer token! Now what?

What is a JSON Web Token?

Base64URL encoded header, claims, and signature strings, delimited by ‘.’

xxxxxxxxxx . yyyyyyyyyyy . zzzzzzzzz

- JOSE Header (a.k.a., ‘JSON Object Signing and Encryption’) contains the type, signing algorithm, and optionally, the ID of the public key
  - {“typ”: “JWS”, “alg”: “RS256”, “kid”: “ab12cd34”}
  - Type can be JWS (signed), or JWE (encrypted), or just ‘JWT’. Large providers always sign these, but almost never encrypt them.
- Claims contain either things to be verified (issuer, audience, not-before, expiry), or information about the user (OIDC or Open ID Connect)
- Signature is the cryptographic signature of the payload and headers

Why would we want to use JSON Web Tokens?

- **Discoverability and ease of use:** large providers like Google or Github publish configs to a URI in the format "**https:**//hostname/**.well-known/…**"
- **Tamper-evident:** claims and headers are cryptographically signed. If it's been modified, signature validation will fail.
- **Claims are not secure:** while the spec supports encrypting them (type JWE), the majority of implementations send these as base64-encoded plaintext.

Provider Implementation Example: the Google Auth API

“Well-known” discoverable config:

https://accounts.google.com/.well-known/openid-configuration

… including current signing keys:

https://www.googleapis.com/oauth2/v3/certs

If it’s easy to consume JWTs, how easy is it to issue them?

- First, let’s do this the hard way, with cryptography.hazmat and __all__ the encodings

- Then, let’s look at a sane implementation using the pyJWT library, https://github.com/jpadilla/pyjwt

Things to Consider

- ID tokens can be used for transferring user data, usually in addition to what's included in a bearer token. Even if it's encrypted, **data you give out doesn't go away when the token expires**.
- When designing a signature scheme, it's **really important** to select an appropriate algorithm:
    - We all know we should use at least SHA256 for hashes. RS256, HS256, and ECDSA all satisfy this requirement.
    - Use an asymmetric (public/private) algorithm to sign your tokens. **Otherwise, anyone who can verify your signature can also sign your tokens**.
    - If you're going to **encrypt** claims, you need a symmetric encryption key per consumer. **These keys can't be published to your JWKS endpoint, and must be rotated manually.**

Further reading:

JWT Specification: https://datatracker.ietf.org/doc/html/rfc7519

User-friendly docs and debugging tools from Okta & auth0: https://jwt.io

Well-known config for the Google Accounts & GitHub:

https://accounts.google.com/.well-known/openid-configuration

https://token.actions.githubusercontent.com/.well-known/openid-configuration

OWASP guidelines for testing JWT implementations:

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/10-Testing_JSON_Web_Tokens

Heather White (Weezəl)

eevel.weezel@gmail.com
https://github.com/eevelweezel