Single Sign-On im Web mittels JWT

Marjan Bachtiari

Aufbau

1. Wofür das Ganze?
2. Grundlagen JWT
    - 2.1. Überblick
    - 2.2. Struktur eines Tokens
    - 2.3. Zwei Arten von JWT
3. Single Sign-On
    - 3.1. Im Allgemeinen
    - 3.2. Mittels JWT
        - 3.2.1. Erstellung
        - 3.2.2. Validierung
4. Kleines Code Beispiel
5. Zusammenfassung

1. Wofür das Ganze?

In welchem Bereich bewegen wir uns?

Wieso ist das interessant?

2. Grundlagen
JWT

2.1 Überblick

Was ist JWT

- offener Industriestandard [RFC7519]
- kompakt
- URL-safe
- “nur” ein Mittel, um Claims auszutauschen

Kompakt

- HTTP Authorisierungsheader geeignet (begrenzter Platz)
- erreicht durch JSON-Format

```
{
  "name": "Marjan",
  "student": true,
  "semester": 6,
  "attended_lectures": [
    "Mathematische Grundlagen",
    "Grundlagen der Informatik",
    "Datenbanken",
    "Logik und Berechenbarkeit",
    ...
  ]
}
```

URL-safe

- also URL-encoded
  - nur in URL darstellbare ASCII-Zeichen verwendet

Repräsentation von Claims

- JWT repräsentiert Set von Claims
- Claim = Name/ Value-Paar

2.2 Struktur eines Tokens

Grundsätzlicher Aufbau

- Header
  - “typ” und “alg” idR enthalten
- Payload
  - “iss”, “exp” und “iat” vor allem interessant
- Signatur
  - berechnet über Hash von Header und Payload

2.3 Zwei Arten von JWTs

JWTs codiert als JWE oder JWS

JWS - JSON Web Signature

- durch digitale Signatur oder MAC gesicherte JSON Struktur
- Aufbau:

```
BASE64URL(UTF8(JWS Protected Header)) || '.' ||
BASE64URL(JWS Payload) || '.' ||
BASE64URL(JWS Signature)
```

JWE - JSON Web Encryption

- durch Verschlüsselung geschütztes JSON Objekt
- Aufbau:

```
BASE64URL(UTF8(JWE Protected Header)) || '.' ||
BASE64URL(JWE Encrypted Key) || '.' ||
BASE64URL(JWE Initialization Vector) || '.' ||
BASE64URL(JWE Ciphertext) || '.' ||
BASE64URL(JWE Authentication Tag)
```

Paar Unterschiede

- Aufbau augenscheinlich
- Verschiedene Algorithmen-Arten in “alg” hinterlegt
- JWS hat payload, JWE ciphertext

3. Single Sign-On

3.1 Im Allgemeinen

3.2 Mittels JWT

3.2.1 Erstellung

1. JWT Claim Set mit gewünschten Claims definieren
2. Nachricht = BASE64URL(UTF8(JWT Claim Set))
3. Header generieren mit gewünschten Params (typ und alg)
4. BASE64URL(UTF8(JWS Protected Header))

3.2.1 Erstellung

5. JWS Signatur gemäß gewähltem Algorithmus berechnen

6. BASE64URL(JWS Signatur)

7. Wenn geschachtelt, wieder bei 3.) anfangen (Payload = jetzige JWS, in Header cty:JWT)

8. erhaltenes JWT als JWS-Repräsentation

3.2.2 Validierung

Wenn einer der folgenden Schritte fehl schlägt, muss JWT abgelehnt werden

1. Verifizieren, dass JWT mind. 1 Punkt enthält

2. Base64url decode den encoded Header

3. Verifizieren, dass resultierende Bytes UTF-8 codierte Repräsentation von gültigem JSON Objekt sind

4. Wenn geschachtelte Signaturen, alle einzeln prüfen, ansonsten Ergebnis 3. = Header

3.2.2 Validierung

5. Verifizieren, dass resultierender Header nur Params und Werte enthält, die verstanden und unterstützt werden

6. Base64url decode den JWS Payload

7. Base64url decode die JWS Signatur

8. Signatur gegen eigene berechnete Signatur validieren

9. Wenn geschachtelt, wiederhole 4 - 8

10. Ausschließlich wenn alle Signaturen/ MACs gestimmt haben, JWT validiert

4. Kleines Code Beispiel

Token Generierung in Rails

5. Zusammenfassung

- kleiner Overhead
- einfache Nutzung über verschiedene Domains hinweg
- Zustand User nicht auf Server gespeichert (zustandslose Authentifizierung)
- kann den Bedarf von DB-Abfragen senken, da self-contained

Vielen Dank für Eure Aufmerksamkeit!

Quellenangaben

* RFC7519
* RFC7515
* RFC7159
* RFC4949
* RFC2104
* https://jwt.io/
* https://github.com/jwt/ruby-jwt
* https://securedb.co/community/jwt-vs-jws-vs-jwe/
* http://jose.readthedocs.org/en/latest/
* Folien Hübner IT-Sicherheit WS15/16
* https://developer.atlassian.com/static/connect/docs/latest/concepts/understanding-jwt.html