LU05e - Authentifizierungsservice erstellen

info box

Planung

Für die Realisierung eines Authentifizierungsservice müssen wir zunächst einige Punkte festlegen.

Benutzerkonten

Unser Authentifizierungsservice benötigt Zugriff auf ein Verzeichnis aller Benutzerkonten. Je nach Anzahl Benutzer und Umfang der Daten können wir die Benutzer in einer Datei (z.B. JSON), einer Datenbank (z.B. MySQL, MongoDB) oder einem LDPA-Verzeichnisdienst erfolgen. In einer Organisation richten wir uns soweit möglich nach den bereits vorhandenen Lösungen.

Die gespeicherten Daten eines Benutzerkontos richten sich nach den Anforderungen der Applikationen, welche den Authentifizierungsservice nutzen wollen. Typische Beispiele sind:

- Benutzername
- Emailadresse
- Passwort (**als Hashwert gespeichert**)
- Weitere Authentifikationsfaktoren wie Authenticatorcode, Public Key oder Telefonnummer.
- Vor- und Nachname
- Benutzergruppe(n) und Berechtigungen

Requests

Ein Authentifizierungsservice muss zumindest einen Service für die Anmeldung anbieten. Zusätzlich können weitere Services für das Abmelden und das Zurücksetzen des Passworts in Frage kommen.

Response

Falls die Authentifizierung erfolgreich war, müssen wir dem Client eine Art von digitaler Identitätskarte, ein sogenanntes Token, ausstellen. Mit diesem Token kann sich der Client danach bei anderen Services ausweisen, um Zugriff auf die Ressourcen zu erhalten. Dazu werden im Token alle für die Autorisation relevanten Daten gespeichert, z.B.

- Benutzername
- Berechtigungen

Solche Tokens müssen natürlich gut geschützt werden. Ein Angreifer könnte mit einem gestohlenen oder manipulierten Token grossen Schaden anrichten. Um dies zu verhindern, treffen wir einige

Sicherheitsmassnahmen.

Gültigkeit

Ein Token ist nur für wenige Minuten gültig. Nach Ablauf dieser Zeit muss der Client ein neues Token anfordern.

Signatur

Eine Signatur lässt uns erkennen, ob ein Token manipuliert wurde. Jedes Token wird durch ein Public-Key-Verfahren mit einem geheimen Schlüssel signiert. Nur der Authentifizierungsservice kennt diesen geheimen Private Key. Um die Signatur zu prüfen nutzen die anderen Services den entsprechenden Public Key.

Verschlüsselung

Eine Verschlüsselung des Tokens verhindert, dass ein Angreifer die Daten auslesen kann. Um das Token zu entschlüsseln, müssen alle Services einen gemeinsamen Schlüssel verwenden.

Umsetzung

Für die Umsetzung des Tokens verwenden wir JSON Web Token (JWT). Die Bibliothek PyJWT stellt uns die nötigen Funktionen bereit.

Schlüssel

Die geheimen Schlüssel dürfen selbstverständlich nicht im Sourcecode unserer Applikation stehen. Andernfalls könnte auf GitHub jeder den Schlüssel sehen. Stattdessen verwenden wir eine `.env`-Datei. Diese soll von git ignoriert werden, damit sie nicht versehentlich im öffentlichen Repository landet.

```
ACCESS_TOKEN_KEY="a04f2af45d3a4e161a7dd2d17fdae47f"
TOKEN_DURATION=600
```

- Der `ACCESS_TOKEN_KEY` dient zum Verschlüsseln des Tokens.
- Die `TOKEN_DURATION` legt fest, wieviele Sekunden das Token gültig ist.

Um die Daten aus der `.env`-Datei zu lesen, musst du `current_app` von `flask` importieren. Beim Erstellen der Flask-Applikation lädst du die Daten aus der Datei:

app.py

```
...
def create_app():
    app = Flask(__name__)
    CORS(app)
    app.config.from_pyfile('./.env')
    api = Api(app)
```

Um einen bestimmten Wert zu lesen, verwendest du den Schlüssel aus der `.env`-Datei:

```
from flask import current_app
def example:
  signature_key = current_app.config['ACCESS_TOKEN_KEY']
  print (signature_key)
```

Signieren und Verschlüsseln

Die Funktion `jwt.encode()` übernimmt diese Aufgaben für uns. Für den Aufruf geben wir diese Parameter mit:

- `payload`: Ein Dictionary mit den Daten, die im Token gespeichert werden.
- `key`: Der geheime Schlüssel. Dies kann ein einfacher String sein oder z.B. ein RSA Private Key
- `algorithm`: Der Verschlüsselungsalgorithmus; Default ist 'HS256''

M321-LU05