SEARCH GUARD

JSON WEB TOKEN AUTHENTICATION

Search Guard

01.

WHY TOKEN BASED AUTHENTICATION

- **Traditional approach: Session-based**
  - → User logs in to an application
  - → The server validates credentials and sets up a server-side session
  - → The browser saves session ID in a cookie and sends it with each request
- **Drawbacks**
  - → Scalability: Sessions are server-bound
  - → Overhead: Sessions are kept in-memory
  - → Security: CSRF
  - → Portability: CORS

02.

WHY TOKEN BASED AUTHENTICATION

- **Token-based authentication**
  - → User logs in to an application or Identity Provider (IdP)
  - → Application or IdP creates a signed token containing all user information
  - → Client application saves token and sends it with each request
- **Advantages**
  - → Scalability: Stateless architecture
  - → Overhead: No state is stored on the server
  - → Standards-based: JWT RFC 7519
  - → Portability: Token can be used for multiple applications

03.

JSON WEB TOKENS

- **JSON-based open standard for creating access tokens**
- **Access tokens assert a number of claims**
  - → Standard claims: subject (user), issuer, expiration date, etc.
  - → Non-standard claims: roles, permissions, etc.
- **Base64 encoded JSON string**
- **Verified by a message authentication hash**
  - → Symmetric: HMAC
  - → Asymmetric: RSA and ECDSA
- **Self-contained, all user information stored in the token**

04.

ANATOMY OF A JSON WEB TOKEN

- **Token consists of three parts**
  - → Header
  - → Payload
  - → Signature
- **Base64 encoded, separated by a dot**

Header	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
Payload	eyJsb2dnZWRJbkFzIjoiYWRtaW4iLCJpYXQiOjE0MjI3Nzk2Mzh9.
Signature	gzSraSYS8EXBxLN_oWnFSRgCzcmJmMjLiuyu5CSpy

05.

HEADER

- Information about the used signing mechanism
- Algorithm / cryptographic hash function
- Type (“static”, JWT)

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

06.

PAYLOAD

- Consists of claims

→ Any piece of information that was verified by the server / IdP

- Standard claims (“registered claims”)
- Non-standard claims

```
{
  "iss": "search-guard.com",
  "sub": "1234567890",
  "name": "John Doe",
  "username": "jdoe",
  "roles": ["devops", "it"]
}
```

07.

SIGNATURE

- **Calculated by a cryptographic hash function using:**
  - → Symmetric: Shared secret
  - → Asymmetric: Private key
- **Appended to header and payload**

```
Pseudo-code (symmetric):
encoded = base64UrlEncode(header) + "." + base64UrlEncode(payload)
signature = HMACSHA256(encoded, 'secretkey');
jwt = encoded + "." + base64UrlEncode(signature)
Result:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dnZWRJbkFzIjoiYWRtaW4iLCJpYXQiOjE0MjI3Nzk2
Mzh9.gzSraSYS8EXBxLN_oWnFSRgCzcmJmMjLiuyu5CSpyHI
```

08.

HTTP BEARER AUTHENTICATION

- HTTP authentication schema

→ "Grant access to the bearer of this token"

- The token is added to each request as HTTP header
- Default header name: "Authorization"
- Token is prepended with "Bearer"

```
Authorization: Bearer <token>
```

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dnZWRJbkFz …
```

09.

SEARCH GUARD CONFIGURATION

- **JWT is self-contained**
  - → All user information present in JWT
  - → No authentication backend necessary
  - → No authorization backend necessary
- **Simple configuration**
  - → Symmetric: shared secret
  - → Asymmetric: public key
  - → Claims containing username (mandatory) and roles (optional)

EXAMPLE: SYMMETRIC KEY

```
jwt_auth_domain:
  enabled: true
  order: 0
  http_authenticator:
    type: jwt
    challenge: false
    config:
      signing_key: "bjBkNDBjYjg0LWJlZTMtMTFlNi1hZjdjLWNiOWFiYTM1YWJjNQ=="
      jwt_header: "Authorization"
      roles_key: roles
      subject_key: username
  authentication_backend:
    type: noop
```

12.

EXAMPLE: ASYMMETRIC KEY

```
jwt_auth_domain:
  enabled: true
  order: 0
  http_authenticator:
    type: jwt
    challenge: false
    config:
      signing_key: |-
        -----BEGIN PUBLIC KEY-----
        MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA2WDCucZF9dVw9j0T6Sp ...
        -----END PUBLIC KEY-----
      jwt_header: "Authorization"
      roles_key: roles
      subject_key: username
  authentication_backend:
    type: noop
```

13.

EXAMPLE: CURL CALL

```
{
  "name": "John Doe",
  "username": "jdoe",
  "roles": ["devops", "it"]
}
```

```
curl -Ss \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdW…” \
  -XGET https://sgssl-0.example.com:9200/_searchguard/authinfo?pretty
```

```
{
  user: "User [name=jdoe, roles=[devops, it], requestedTenant=null]",
  user_name: "jdoe",
  backend_roles: ["devops", "it"],
  sg_roles: ["sg_devops", "sg_it"],
  ...
}
```

ALTERNATIVE: URL PARAMETER

- The token is added to each request as URL parameter
- Non-standard
- Parameter name can be chosen freely

```
https://sgssl-0.example.com:9200/_searchguard/authinfo?urltoken=eyJhbGciOiJIUzI1 …
```

EXAMPLE: URL PARAMETER

```
jwt_auth_domain:
  enabled: true
  order: 0
  http_authenticator:
    type: jwt
    challenge: false
    config:
      signing_key: "bjBkNDBjYjg0LWJlZTMtMTFlNi1hZjdjLWNiOWFiYTM1YWJjNQ=="
      jwt_url_parameter: "urltoken"
      roles_key: roles
      subject_key: username
  authentication_backend:
    type: noop
```

16.

RESOURCES

- Search Guard website

→ https://search-guard.com/

- Documentation

→ https://docs.search-guard.com/latest/json-web-tokens

- Community Forum

→ https://groups.google.com/d/forum/search-guard

- GitHub

→ https://github.com/floragunncom

SEARCH GUARD

SEND US A MESSAGE

info@search-guard.com

Search Guard

Search Guard