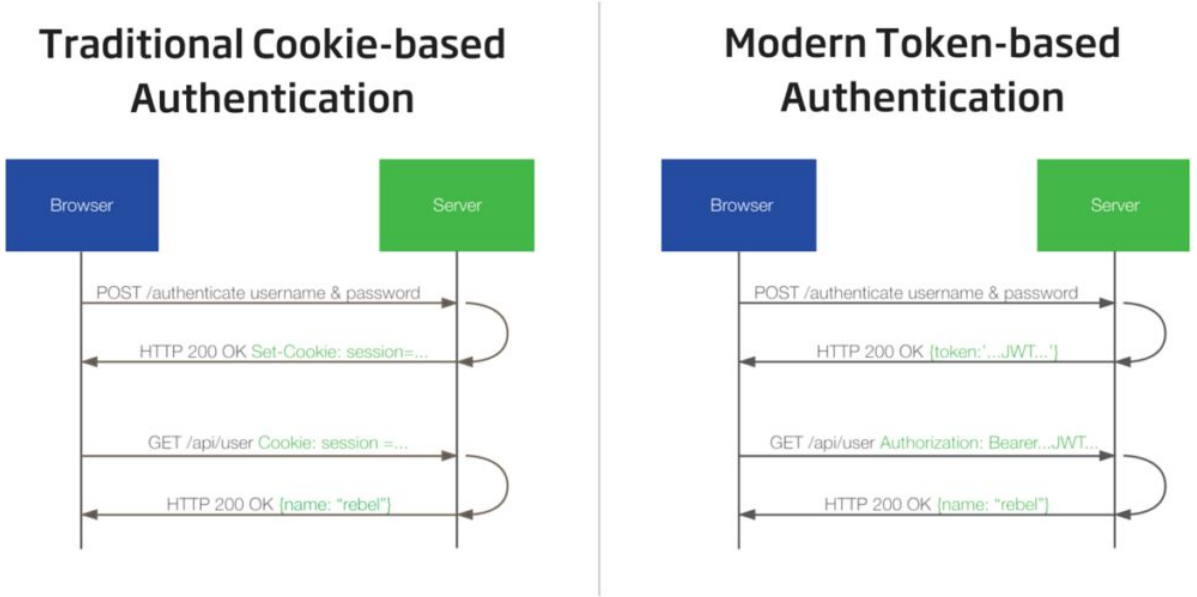
REST Security

REST Security

- REST is stateless
- Requests are complete and self-contained
- Security challenges

Token Based Authentication

- One time use of username and password for the purpose of obtaining a login token



Token Characteristics

- Time limited
- Revocable
- Trusted
- Self-contained
- Compact

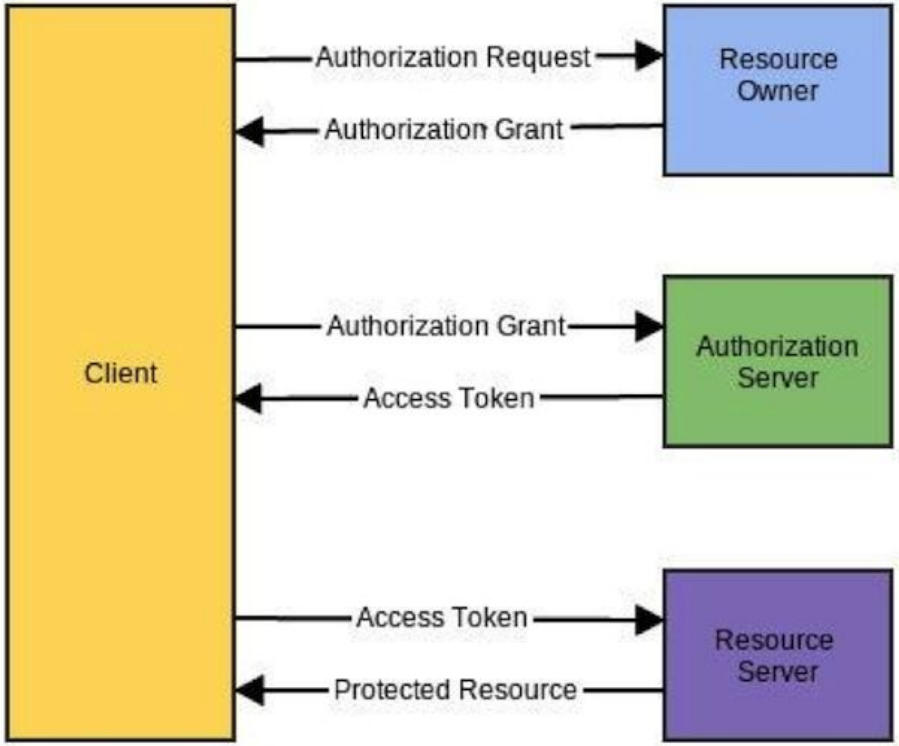
OAuth2

- OAuth 2 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean

- It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account

- OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.

OAuth2


Client
Authorization Request
Authorization Grant
Resource Owner
Authorization Grant
Access Token
Authorization Server
Access Token
Protected Resource
Resource Server

JWT

- JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

JWT

- Open standard
- Securely transfer data between nodes
- Digitally signed information is verified and trusted

JWT

- Compact
  - URL, POST, Header
  - Fast transmission
- Self-contained
- Information about the user
- Decreases DB query need

JWT Token Example

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJiYWRqZXZpYy5tIiwiZXhwIjoxNTI3NDk0MzQ3LCJpYXQiOjE1MjY4ODk1NDd9.ohdqtLd9P12T-AzuPBssfs5kkkQD2D2ddBKw_9D1Epfjv6EV7HwtuehWAyzIT3-KFkGpmQWW7AaDR-2ZV2S0Wg

https://jwt.io

JWT Structure

Header: eyJhbGciOiJIUzUxMiJ9

```
{  "alg": "HS512" }
```

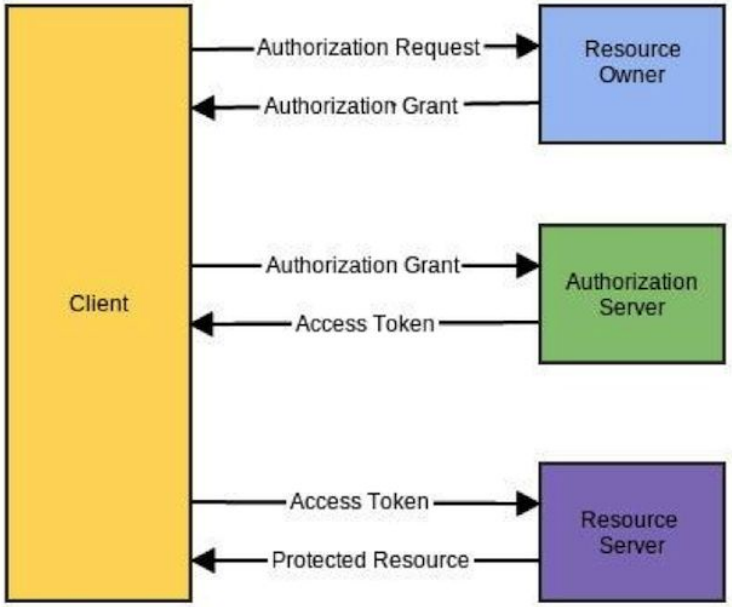
Payload:eyJzdWIiOiJiYWRqZXZpYy5tIiwiZXhwIjoxNTI3NDk0MzQ3LCJpYXQiOjE1MjY4ODk1NDd9

{ "sub": "badjevic.m", "exp": 1527494347, "iat": 1526889547}

Signature:ohdqtLd9P12T-AzuPBssfs5kkkQD2D2ddBKw_9D1Epfjv6EV7HwtuehWAyzIT3-KFkGpmQWW7AaDR-2ZV2S0Wg

OAuth2 and JWT

- OAuth2 is a protocol
- JWT is a token format



JWT SAGA Example

- First step - Authentication with username and password

```
@RequestMapping(value = "${jwt.route.authentication.path}", method = RequestMethod.POST)
public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtAuthenticationRequest authenticationRequest)
throws AuthenticationException {

   authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());

   // Reload password post-security so we can generate the token
   final UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());
   final String token = jwtTokenUtil.generateToken(userDetails);

   // Return the token
   return ResponseEntity.ok(new JwtAuthenticationResponse(token));
}
```

Token Generation

- https://java.jsonwebtoken.io/

```
public String generateToken(UserDetails userDetails) {
   Map<String, Object> claims = new HashMap<>();
   return doGenerateToken(claims, userDetails.getUsername());
}

private String doGenerateToken(Map<String, Object> claims, String subject) {
   final Date createdDate = clock.now();
   final Date expirationDate = calculateExpirationDate(createdDate);

   return Jwts.builder()
        .setClaims(claims)
        .setSubject(subject)
        .setIssuedAt(createdDate)
        .setExpiration(expirationDate)
        .signWith(SignatureAlgorithm.HS512, secret)
        .compact();
}
```

Storing Token on the Client Side

- Browser Local Storage

```
// client.js
function getJwtToken() {
   return localStorage.getItem(TOKEN_KEY);
}

function setJwtToken(token) {
   localStorage.setItem(TOKEN_KEY, token);
}
```

- Adding as header with every request

```
// client.js
$.ajax({...   headers: createAuthorizationTokenHeader() …}
```

Authorization:

```
Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJuaW5vdmljLm4iLCJleHAiOjE1Mjc1MDA3MTgsImlhdCI6MTUyNjg5NTkxOH0.9hsp4dq61cO9n0zmxLfsZ5H2K4UBlUyw
Nnb2fHTV9fXtQDC2tBCT5OPgMSWcOptqwtrpDzn9Sa8gmR8DK6wCBw
```

Security Config

```
httpSecurity
        // we don't need CSRF because our token is invulnerable
        .csrf().disable()
        .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
        // don't create session
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
        .authorizeRequests()
        .antMatchers("/auth/**").permitAll()
        .anyRequest().authenticated();



// Custom JWT based security filter
JwtAuthorizationTokenFilter authenticationTokenFilter = new JwtAuthorizationTokenFilter(userDetailsService(),
jwtTokenUtil, tokenHeader);
httpSecurity
        .addFilterBefore(authenticationTokenFilter, UsernamePasswordAuthenticationFilter.class);
```