

JSON Web Token (JWT)

Dominick Baier

<http://leastprivilege.com>

@leastprivilege



Outline

- **Overview**
- **History & purpose of security tokens**
- **Properties & structure of JWTs**
- **Producing a token**
- **Consuming a token**

Purpose of a security token

- **Security tokens are (protected) data structures**
 - contain information about issuer and subject (claims)
 - signed (tamper proof & authenticity)
 - typically contain an expiration time
- **A client requests a token**
- **An issuer issues a token**
- **A resource consumes a token**
 - has a trust relationship with the issuer

History

- **SAML 1.1/2.0**

- XML based
- many encryption & signature options
- very expressive

- **Simple Web Token (SWT)**

- Form/URL encoded
- symmetric signatures only

- **JSON Web Token (JWT)**

- JSON encoded
- symmetric and asymmetric signatures (HMACSHA256-384, ECDSA, RSA)
- symmetric and asymmetric encryption (RSA, AES/CGM)
- (the new standard)

JSON Web Token

- **On its way to official standardization**
 - <http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html>
- **Header**
 - metadata
 - algorithms & keys used
- **Claims**
 - Issuer (iss)
 - Audience (aud)
 - IssuedAt (iat)
 - Expiration (exp)
 - Subject (sub)
 - ...and application defined claims

Structure

Header

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

Claims

```
{  
  "iss": "http://myIssuer",  
  "exp": "1340819380",  
  "aud": "http://myResource",  
  "sub": "alice",  
  
  "client": "xyz",  
  "scope": ["read", "search"]  
}
```

eyJhbGciOiJIub251In0.eyJpc3MiOiJqb2UiLA0KICJleHAiOiJlZzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt

Header

Claims

Signature

Producing a token

- **Microsoft library on Nuget**

- <http://nuget.org/packages/Microsoft.IdentityModel.Tokens.JWT/>

```
var token = new JWTSecurityToken(  
    issuer: "http://myIssuer",  
    audience: "http://myResource",  
    claims: GetClaims(),  
    signingCredentials: GetKey(),  
    validFrom: DateTime.UtcNow,  
    validTo: DateTime.UtcNow.AddHours(1));  
  
// serialize  
var tokenString =  
    new JWTSecurityTokenHandler().WriteToken(token);
```

Consuming a token

- **Retrieve serialized token**
 - from HTTP header, query string etc...
- **Validate token**
 - and turn into claims

```
var token = new JWTSecurityToken(tokenString);
var validationParams = new TokenValidationParameters
{
    ValidIssuer = "http://myIssuer",
    AllowedAudience = "http://myResource",
    SigningToken = GetSigningKey()
};

var handler = new JWTSecurityTokenHandler();
var principal = handler.ValidateToken(token, validationParams);
```


Summary

- **JWT is easy to**
 - create
 - transmit
 - parse
 - validate
- **Quickly becomes the standard for web based tokens**
- **Mandatory in OpenID Connect**