Security_4_Plugfest – Cheat Sheet

Author: Oliver Pfaff, Siemens AG, CT ITS RTC

Introduction

This document assumes that you will (or consider to) participate in the Plugfest during the Nice F2F of the W3C IG WoT and you are developing any of the following components:

- A **resource server** (short: RS) responding to requests via **HTTP** or **CoAP**
- A **client** (short: C) sending requests via **HTTP** or **CoAP**

This document provides a cheat-sheet for the security-enabling of these interactions that underpins the accompanying HowTo document.

Client

The client engages in 3 security-related interactions:

- Registration via HTTP (RFC 7591)
- Token acquisition via HTTP (RFC 6749)
  - Minimal token (default, described herein) or
  - Normal token
- Token supply (with a business request) via HTTP (RFC 6750) or CoAP (draft-tschofenig-ace-oauth-bt-01)
  - Minimal token (default, described herein) or
  - Normal token

Registration and Token Acquisition

Sample requests for client registration and token acquisition are provided as Postman[1] collection at https://www.w3.org/WoT/IG/wiki/images/7/7b/Nice_F2F_Plugfest.json.zip

The request samples can be directly executed (in Postman)[2]. If other tools such as cURL are preferred: Postman is also able to generate representations for other tools. Postman also provides code snippets for common programming languages such as Java.

The handling of the registration and token responses requires a JSON library. See json.org for JSON libraries in various programming languages. The following code snippets use Jackson (Java) and (Https)URLConnection (Java) for illustration[3].

Extracting Client_Id/Secret from the Registration Response

```
// Check if the HTTP response is 401 Created


// Get the HTTP response body

byte[] regResponseBody = getResponsePayload(httpsURLConnection);


// Extract the client credentials

ObjectMapper mapper = new ObjectMapper();

JsonFactory jsonFactory = mapper.getFactory();

JsonParser jsonParser = jsonFactory.createParser(regResponseBody);

JsonNode regResponseObj = mapper.readTree(jsonParser);

JsonNode clientId = regResponseObj.get("client_id");

JsonNode clientSecret = regResponseObj.get("client_secret");

String client_id = clientId.textValue();

String client_secret = clientSecret.textValue();
```

[1] Postman (https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop) provides a UI-based test client for RESTful Web services

[2] For the token request: replace the dummy values for client_id/secret with the values obtained from the client registration response

[3] The snippets aim at simplicity. See the documentation of the corresponding toolkits for more details/elaborate processing

Extracting AS Token from the Token Response

```
// Check if the HTTP response is 200 Ok


// Get the HTTP response body

byte[] tokenResponseBody = getResponsePayload(httpsURLConnection);


// Extract the AM token

ObjectMapper mapper = new ObjectMapper();

JsonFactory jsonFactory = mapper.getFactory();

JsonParser jsonParser = jsonFactory.createParser(tokenResponseBody);

JsonNode tokenResponseObj = mapper.readTree(jsonParser);

JsonNode asToken = tokenResponseObj.get("access_token");


// Extract the AS token: AS token is in the AM token payload, and this part is
// encoded by Base64 (AM token checks skipped here, see below for a snippet)
String[] asTokenSplits= asToken.textValue().split("\\.");

String asTokenDecoded= new String(Base64.getDecoder().decode(asTokenSplits[1]));

jsonParser = jsonFactory.createParser(asTokenDecoded);

JsonNode amTokenPayload = mapper.readTree(jsonParser);

JsonNode asToken = amTokenPayload.get("as_token");

String as_token = asToken.textValue();
```

For more details see: Security4NicePlugfest.java

Attaching AS Token to RS Request

```
// Prepare as usual

// Add Authorization header
httpURLConnection.setRequestProperty("Authorization", “Bearer: “ + as_token);
```

Resource Server

The resource server engages in 1 security-related interaction:

- Endpoint protection for HTTP (RFC 6750) and/or CoAP (draft-tschofenig-ace-oauth-bt-01) enforcing the presence of a
    - Minimal token (default, described herein) or

- Normal token

Endpoint Protection

The processing of security tokens requires a JWT library. See jwt.io for JWT libraries in various programming languages. The Web page jwt.io also provides a UI-based debugging utility. The following code snippets use Jose4Java (Java) and HttpServletRequest (Java) for illustration[4].

Enforce Valid Token in the Resource Request

Note that this description considers the AS token in its *minimal* layout. Additional checks are needed for the *normal* layout

```
String authorization = httpServletRequest.getHeader(“Authorization”);

// Enforce presence of Authorization header

If (isNullOrEmpty(authorization)) {

   httpServletResponse.sendError(401);

}

// Enforce presence of Bearer token in Authorization header

if (!authorization.contains(“Bearer: ”))  {

   httpServletResponse.sendError(401);

}

// Get the bearer token value

String[] splits = authorization.split(“Bearer: ”);

if (splits.lenght != 2)  {

   httpServletResponse.sendError(401);

}

String jwt =splits[1];

// Process the bearer token value

if (isValid(jwt, httpServletRequest)) {

  // Expose the business functionality evtl. subject to information found in the JWT

  // e.g. “sub”

} else {
```

[4] The snippets aim at simplicity. See the documentation of the corresponding toolkits for more details/elaborate processing

```
  httpServletResponse.sendError(401);
}


// Utility methods
Private Boolean isValid(String jwt, HttpServletRequest httpServletRequest) {
  // See https://bitbucket.org/b_c/jose4j/wiki/JWT%20Examples (JWT-intrinsic checks)
  // If the normal token is utilized do check against the HttpServletRequest instance
  If (okay) {
    // Instead of returning a Boolean one can also return JWT contents esp. the value
    // of “sub”
    return true;
  } else {
    return false;
  }
}
```