Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

H. Tschofenig
ARM Limited
E. Maler
Forgerock
E. Wahlstroem
S. Erdtman
Nexus Technology

March 9, 2015

Authentication and Authorization for Constrained Environments Using OAuth and UMA
draft-maler-ace-oauth-uma-00.txt

Abstract

Authentication and authorization are fundamental security features used in Internet and Web applications. Providing the same level of security functionality to the Internet of Things (IoT) environment as well is a logical enhancement and reduces the risk of unauthorized access to personal data.

IoT devices, however, have limitations in terms of processing power, memory, user interface, Internet connectivity, etc. Since many use cases span Web and IoT environments and the question of "Web" vs. "IoT" can in some cases be considered a continuum, it is required to find security solutions that can accommodate the capabilities and constraints of both environments without significant compromises.

Thus, an approach of adapting already standardized and deployed authentication and authorization technologies is worth examining. This document describes how the Web Authorization Protocol (OAuth) in combination with User-Managed Access (UMA) can be used for an IoT environment to bring Web-scale authorization services to the IoT world.

Status of This Memo

Table of Contents

1. Introduction

Deciding when a certain use case falls under the category of IoT and when it is not turns out to be a difficult task. For this reason, [RFC7228] made an attempt to describe characteristics of constrained-node networks and highlights some of the challenges. Companies often

have some degree of freedom to make trade-off decisions, for example, in terms of cost vs. physically available resources to push the boundaries of what can be done with IoT devices.

Manufacturers must take not only hardware costs into account, but also software development costs; reusing existing software, standards, practices, and expertise can help to lower the total cost of a product. Hence, the use cases combine the already existing identity and access management infrastructure with access control to objects in the physical world.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

This document leverages terminology from [RFC6749] and [I-D.hardjono-oauth-umacore] . Especially pertinent definitions are paraphrased below.

Resource Owner: An entity capable of granting access to a protected resource.

Resource Server: The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

Authorization Server: The server issuing access tokens to the client after successfully authorizing it.

Requesting Party: An entity (which may or may not be the same as the resource owner) that uses a client to seek access to a protected resource.

Client: An application making protected resource requests with the resource owner's authorization and on the requesting party's behalf.

3. Use Cases

The sub-sections below illustrate some use cases that start with classic OAuth functionality and then extend it to functionality only available with UMA-based environments. The scenarios involve Web, smart phone app, and IoT devices. Unlike the scenarios described in [I-D.ietf-ace-usecases] this write-up is not solution agnostic but

instead aims to take the OAuth/UMA solutions into account. In a stepwise refinement we then add even more details in Section 5.

3.1. Using OAuth with Scales

In a classic OAuth flow, an end-user (the resource owner) can enable a client application to call an API (at the resource server) on his or her behalf securely and with authorized consent, without having to reveal his or her credentials, such as a username and password, to the client. An app-specific access token (issued by the authorization server at which the resource owner is able to authenticate), whose operation may be scoped to some subset of the API's capabilities, is substituted for the long-term credentials instead.

The basic OAuth architecture is shown in Figure 1 and the corresponding message exchange in Figure 2.

```
                               +-------------+
                               |Authorization|
                               |Server (AS)  |\
                               +-------------+ \
                               ^        /   ^    \
                    Request   /        /     \     \     *Token
                    Access   / Access /       \     \  Introspection
                    Token   /  Token /          \     \
                           /        /            \     \
                          /        /              \      \
                         /        /                 \      \
                        /        /                   \      \
         O             /        v                     \       v
        /|\          +-----------+                  +-----------+
         |    -----> |           |   Access Token   | Resource  |
        / \   <----- |  Client   |----------------->|  Server   |
      Resource       |           |<================>|   (RS)    |
       Owner         +-----------+ Application Data +-----------+

         *: indicates optional exchange.
```

Figure 1: OAuth Architecture.

```
    +--------+                               +---------------+
    |        |--(A)- Authorization Request ->|   Resource    |
    |        |                               |     Owner     |
    |        |<-(B)-- Authorization Grant ---|               |
    |        |                               +---------------+
    |        |
    |        |                               +---------------+
    |        |--(C)-- Authorization Grant -->| Authorization |
    | Client |                               |     Server    |
    |        |<-(D)----- Access Token -------|               |
    |        |                               +---------------+
    |        |                                  ^ |    *Token
    |        |                               (F)| |(G) Introspection
    |        |                                  | v
    |        |                               +---------------+
    |        |--(E)----- Access Token ------>|    Resource   |
    |        |                               |     Server    |
    |        |<-(H)--- Protected Resource ---|               |
    +--------+                               +---------------+
```

Figure 2: OAuth 2.0 Message Exchange.

We can apply a similar pattern to IoT devices as well. For example, envision an end-user Alice and her new purchase of an Internet-connected scale designed for "quantified self" scenarios. In our example, the scale has a micro-controller that was pre-provisioned with a certificate during manufacturing enabling the device to authenticate itself to the vendor-authorized software update server as well as to other parties. The identifier used for authentication of a scale is something as benign as an EUI-64 serial number.

Once the identifier used by the scale and Alice's account information have been provisioned into an online repository, and if Alice can demonstrate appropriate control of the device -- for example, by entering a confirmable PIN code or serial number that was packaged with the shipped device into her online account record, whether through a Web or mobile app -- it is possible to treat the device as an OAuth client and issue it an OAuth token so that it can act on Alice's behalf.

The value of this association is that any API calls made by the scale, for example to report Alice's weight, body mass index (BMI), or progress against health goals into her online account, will be associated with her alone. If other household members use the scale as well, their unique associations will ensure that their data will go to the right place (assuming there is a mechanism at the scale

that allows family members to be differentiated). Further, each token can be revoked and expired exactly like any other OAuth token.

3.2. Using UMA with Scales

UMA builds on top of OAuth (and optionally OpenID Connect [OIDC]) to let an end-user achieve three main goals:

1. authorize other parties to access APIs under his or her control using client applications;

2. set conditions for access so that those other parties may have to provide "claims" and do step-up authentication to get access (in a so-called claims gathering process); and

3. centralize management of all these conditions for access in one cloud service.

The basic architecture and flow is shown in Figure 3. A protection API token (PAT) is an OAuth token with a scope that gives the resource server access to the UMA-standardized protection API at the authorization server; an authorization API token (AAT) is an OAuth token with a scope that gives the client access to the UMA-standardized authorization API; and a requesting party token (RPT) is the main access token issued to a requesting party, which does not rely on resource owner presence for issuance.

```
                                              +--------------+
                                              |   resource   |
             +---------manage (A)------------ |     owner    |
             |                                +--------------+
             |         Phase 1:                      |
             |         protect a                control (C)
             |         resource                      |
             v                                       v
      +------------+               +----------+--------------+
      |            |               |protection|              |
      |  resource  |               |   API    | authorization|
      |   server   |<-protect (B)--|  (needs  |    server    |
      |            |               |   PAT)   |              |
      +------------+               +----------+--------------+
      | protected  |                          | authorization|
      | resource   |                          |     API      |
      |(needs RPT) |                          |  (needs AAT) |
      +------------+                          +--------------+
             ^                                       |
             |         Phases 2 and 3:         authorize (D)
             |         get authorization,            |
             |         access a resource             v
             |                                +--------------+
             +---------access (E)-------------|    client    |
                                              +--------------+

                                              requesting party
```

                  Figure 3: OAuth++: The UMA Architecture.

UMA can be thought of as "OAuth++", in that it adds two major elements: a formal protection API presented by the authorization server, so that resource servers running in different domains can be "authorization relying parties" to it, and the "requesting party" concept distinct from the resource owner (as discussed in Section 2).

The requesting party may be required to interact with the authorization server when the client asks for permission to access a resource. However, if this interaction requires authentication, this authentication step may be outsourced to a variety of different identity providers, including the client (which may be allowed to "push" identity claims to the authorization server), the authorization server itself, or any other identity provider, with the authorization server functioning as a relying party in this case.

Similarly to the previous use case in Section 3.1, there is value in extending the Web world to the world of devices because the data

originating in a device often travels to the cloud. Alice may want to share her scale data with friends, with her doctor, or in anonymized form with a public health service.

The benefit of using an UMA authorization server, requesting party tokens, and so on to manage Alice's control of her doctor's and others' access to the data her scale generates is that she:

1. does not have to be present when they request access, crafting policies prior to access attempts or handling access approval requests after attempts;

2. can demand that requesting parties present proof of their suitability (such as current valid hospital credentials);

3. can change the length permission validity, including revoking sharing relationships;

4. can set policies governing clients used by requesting parties as well; and

5. can do this from a centralizable authorization point, crossing multiple resource servers (and thus devices feeding into them).

3.3. Using OAuth and UMA with Cars

A connected car example illustrates other desirable aspects of IoT authentication and authorization.

Alice buys a new car. At manufacture time, the car was registered at the manufacturer's authorization server. When buying the car, Alice can create an account at the manufacturer's website and reuse the already configured authorization server. Alice installs a car managing mobile app on her phone to manage her car. Alice authorizes the app to act on her behalf as OAuth client to perform actions, such as open car door, which would be similar to authorizing an app to send tweets on my behalf to the twitter API but in this case the resource server is the car and the API is accessed over Bluetooth Smart.

Since the operation of opening the car is security sensitive, it is desirable to require more than a long term access token to open the door and to start the car. So instead of just accepting the access token the authorization server may require Alice to supply more information and a UMA claims gathering process is started, such as requiring a multi-factor authentication using a fingerprint or a PIN code on her phone.

Furthermore, Alice wants to share driving rights with her husband Ted. Alice is owner of the car and is authorized to add new drivers to the car. To do this Alice can setup the policies at the authorization service governing who can do what with the car at what time. Alice configures a rule that allows Ted to request a token for the scope of driving the car, but just as Alice, Ted is required to download the app, authorize it and go through a claims gathering flow to actually get the token to start the car using his smart phone app.

With this delegation of rights to the car Ted could potentially even create a valet key with geo fenced driving range and no access to trunk when he leaves the car in a parking garage and thereby create a valet key for the physical world.

The use of standardized protocols allows Alice to use her own authorization server. Alice could choose to unregister the car at the manufacturer authorization server and register the car to an authorization server of her liking. The car would register available resources and scopes and Alice could configure policies as above using her own authorization server.

Since cars are not always located in areas with Internet connectivity it is envisioned that cars need to be able to verify access tokens locally (without the need to consult an authorization server in real-time). Once the car is online again it could check whether any new revocation information is available and upload information about earlier authorization decisions to the audit log.

A similiar situation may occur when Alice asks her friend Trudy to get the groceries from the trunk of her car (which she forgot there earlier) while they are at their remote summer cottage. Without Internet connectivity Alice cannot delegate access to her car to Trudy using the authorization server located in the cloud. Instead, she transfers an access token to Trudy using Bluetooth. This access token entitles Trudy to open the trunk but not to drive it and grants those permissions only for a limited period. To ensure that the car can actually verify the content of the access token the client app of Alice again uses the capabilities of the proof-of-possession tokens.

3.4. Using OAuth and UMA with Door Locks

Alice, the owner of a small enterprise, buys a door lock system for her office. She would expect to be able to provision policies for access herself, in effect acting as "system administrator" for herself and for her five employees. She may also want to choose her own authorization server, since she wants to integrate the physical access control system with the rest of the resources in her company and the enterprise identity management system she already owns. She

wants to control the cloud-based file system, financial and health data, as well as the version control and issue tracking software.

4. Protocol Designs for the Web and Beyond

The design of OAuth was intentionally kept flexible to accommodate different deployment situations. For example, authentication of the resource owner to the authorization server before granting access is not standardized and different authentication technologies can be used for that purpose. The user interface shown to the resource owner when asking for access to the protected resource is not standardized either.

Over the years various extensions have been standardized to the core OAuth protocol to reduce the need for proprietary extensions that offer token revocation, an access token format called JSON Web Token, or proof-of-possession tokens that offer an alternative security model for bearer tokens [RFC6750].

Due to the nature of the Web, OAuth protocol interactions have used HTTPS as a transport; however, other transports have been investigated as well, such as OAuth for use over SASL (for use with email) and more recently OAuth over the Constrained Application Protocol (CoAP).

This document provides the reader with information about which OAuth extensions will be useful for the IoT context. In its structure it is very similar to the DTLS/TLS IoT profile document that explains what TLS extensions and ciphersuites to use for different IoT deployment environments. Interestingly, very little standardization effort is necessary to make OAuth and UMA fit for IoT. To a large extend the work is centered around using alternative transports (such as CoAP and DTLS instead of HTTP over TLS) to minimize the on-the-wire overhead and to lower code-size and to define profiles for highly demanded use cases.

The UMA group, benefiting from observing the OAuth experience and from the era in which UMA itself has been developed, has built extension points into the protocol, already anticipating a need for flexibility in transport bindings. Thus, UMA has three "extensibility profiles" that enable alternate bindings (such as CoAP) to be defined for communications between an authorization server and resource server, a resource server and client, and an authorization server and client respectively. It also, similarly to OAuth, as other extensibility options, such as token profiling and the ability to extend JSON formats to suit a variety of deployment needs.

5. Instantiations

In this section we provide additional details about the use of OAuth and UMA for solving the use cases outlined in Section 3. In general, the following specifications are utilized:

- OAuth 2.0 [RFC6749] for interacting with the authorization server. The use of the CoAP-OAuth profile [I-D.tschofenig-ace-oauth-iot] maybe used but is not essential for the examples in this section since the client is less constrained.

- Bearer tokens and proof-of-possession tokens as two different security models for obtaining and presenting access tokens. Bearer tokens are defined in [RFC6750] and the architecture for proof-of-possession (PoP) tokens can be found at [I-D.ietf-oauth-pop-architecture]. PoP tokens introduce the ability to bind credentials, such as an ephemeral public key, to the access token.

- UMA [I-D.hardjono-oauth-umacore] for registering the resource server with the authorization server provided by Alice and for management of policy.

- Dynamic Client Registration [I-D.ietf-oauth-dyn-reg] for the client app to register at the authorization server.

- Token introspection [I-D.ietf-oauth-introspection] for optionally allowing the resource server to verify the validity of the access token (if this step is not done locally at the resource server). The use of token introspection over CoAP [I-D.wahlstroem-ace-oauth-introspection] reduces overhead.

- JSON Web Token (JWT) [I-D.ietf-oauth-json-web-token] for the format of the access token. JSON Web Signatures (JWT) [I-D.ietf-jose-json-web-signature] are used for creating a signature over the JWT. The use of a CBOR encoding of various JSON-based security specifications is under discussion to reduce the size of JSON-based tokens.

- A new Bluetooth Smart service and profile for conveying access tokens securely from the client to the resource server. If CoAP runs between the client and a constrained resouce server then [I-D.tschofenig-ace-oauth-bt] provides additional overhead reduction.

5.1. Car Use Case

In the car use case, as described in Section 3.3, the car acts as the resource server and an application on the smart phone plays the role of the client. Alice is first a delegated administrator then becomes a resource owner of the car.

Alice creates an account, downloads and authorizes the mobile app:

1. Alice creates an account on manufacturer's website.

2. Alice selects that two factor authentication must be used to be able to start controlling car from an app.

3. Alice downloads app and starts it.

4. App has never been provisioned so a browser is started, user selects manufacturer's authorization server from a list.

5. Alice authenticates using two factors and authorizes the application.

6. Access and refresh tokens are provisioned to the app.

Alice configures policies to add Tim as new driver:

1. Alice opens the car-settings page within the app.

2. Alice selects to add a new driver by supplying Tims email address.

3. Alice checks the checkboxes that also makes Tim a delegated administrator.

4. Alice saves the new policies.

Alice opens car door over Bluetooth Smart:

1. The smartphone detects the advertising packets of the door lock and asks Alice whether she wants to open the car door.

2. Alice confirms and a request is sent to the authorization server together with an ephemeral public key created by the phone. The request indicates information about the car Alice is seeking access to.

3. The authorization server evaluates the request to open the car door on the specific car and verifies it against the access

control policy. Note that the app authenticated itself to the authorization server.

4. The authorization server prompts Alice for a PIN code using claims gathering.

5. Alice enters pin and the application communicates it to the authorization server.

6. It turns out that the system administrator has granted her access to that specific car and she is given access by returning an access token.

7. The smart phone app then uses the obtained access token to create a request (which includes the access token) over Bluetooth Smart using on the (not yet existing) Physical Access Control Profile, which is a security protocol that utilizes public key cryptography where the app demonstrates that it knows the private key corresponding to the finger of the public key found in the token.

8. The car receives the request and verifies it.

9. To check whether the permissions are still valid the car sends the access token to the introspection endpoint.

10. The authorization server validates the access token and returns information about the validity of the token to the car. In this case it's a valid token.

11. The request is logged.

12. The car gets a response and opens the car door.

Alice changes authorization server:

1. Alice wants to connect the car to her own authorization server instead of the manufacturers default authorization server.

2. Alice makes a request to the current authorization server to unbind the device from the authorization server.

3. The authorization server validates Alice request to remove the authorization server.

4. Alice configures a new authorization server in the apps UI.

5. The app starts an authorization code grant flow with the private authorization server of Alice. Alice logs on and authorizes the app to act on her behalf.

6. The app sends information about the new authorization server to the car using Bluetooth Smart.

7. The car registers the resource it offers with the new authorization server.

8. Alice configures herself as the car owner in the new authorization server.

9. The car unbinds itself from the old authorization server by invalidating the access tokens using the revocation endpoint.

5.2. Door Lock Use Case

In the constrained server use case, as described in Section 3.4, the door lock acts as the resource server and an application on the smart phone plays the role of the client.

Since the client runs on a powerful smartphone standard OAuth according to OAuth Core can be used. To avoid leakage of the access token the use of a proof-of-possession token is utilized instead of a bearer token. This allows the client to demonstrate the possession of the private key to the client. Both symmetric as well as asymmetric cryptography can be used. The use of asymmetric cryptography is beneficial since it allows the client to create a public / private key pair and to never expose the private key to other parties.

As a setup-step the following steps are taken as part of the enterprise IT

1. Alice, as the enterprise network administrator and compay owner, enables the physical access control rights at the identity management server.

2. Alice downloads the enterprise physical access control system app on her phone. By downloading the app she agrees to the terms of use and she accepts the permissions being asked for by the app.

3. Alice associates her smart phone app with her account by login into the enterprise management software, which uses OAuth 2.0 for delegating access to the app.

4. Alice, as the enterprise administrator, configures policies at the authorization server to give her employees access to the office building as well.

5. In this use case each door lock is provisioned with an asymmetric key pair and the public key of the authorization server. The public key of each door lock is registered with the authorization server. Door locks use these keys when interacting with the authorization server (for authentication in case of token introspection), for authenticating towards the client, and for verifying the signature computed over the access token.

When Alice uses her smartphone for the first time to access the office building the following steps take place:

1. The smartphone detects the advertising packets of the door lock and asks Alice whether she wants access.

2. Alice confirms and a request is sent to the authorization server together with an ephemeral public key created by the phone. The request indicates information about the door Alice is seeking access to. The request is protected using TLS.

3. The authorization server evaluates the request and verifies it against the access control policy. Since Alice has added herself to access control policies already she is given access by returning an access token. This access token includes the fingerprint of the public key provided in the request. The access token is digitally signed to avoid any modification of the content.

4. The smart phone app then uses the obtained information to create a request (which includes the access token) over Bluetooth Smart using the (not yet existing) Physical Access Control Profile, which is a security protocol that utilizes public key cryptography where the app demonstrates that it knows the private key corresponding to the finger of the public key found in the token.

5. The door lock software receives the request and verifies the digital signature, inspects the content (such as expiry date, and scope), and determines whether the fingerprint of the public key corresponds to the private key used by the client. Once successfully verified the door is unlocked, and Alice is allowed to enter.

6. The physical access control app caches the access token for future use.

As a variation of the above-described procedure, the door lock might consult the authorization server using token introspection to determine the validity of the access token. This allows the enterprise system software to make real-time access control decisions and to better gain visibility about the number of employees in the building (in case of an emergency).

When Alice approaches the door next time her physical access control app determines that a cached (and still valid) access token is available and no further interaction with the authorization server is needed. Decisions about how long to cache access tokens are a policy decision configurable into the system and impact the performance of the protocol execution.

When Bob, who is employed by Alice, approaches the office building for the first time his downloaded physical access control app also interacts with the door. While Bob still has to consent to the use of app, Alice does not need to authorize access of Bob to the office building in real-time since she has already granted access to her employees earlier already.

6. UMA Use Case Mapping Exercise

An analysis of [I-D.hardjono-oauth-umacore] suggests that its capabilities have a good architectural match with many published ACE use cases. The following are aggregated and paraphrased versions of use cases discussed in [I-D.ietf-ace-usecases]:

Owner grants different resource access rights to different parties (U1.1, U2.3, U.3.2):

> UMA meets this use case because the requesting party is formally distinct from the resource owner and because each requesting party, and each client, is represented distinctly at each authorization server, able to have differential policy applied to it.

Owner grants different access rights for different resources on a device (U1.3, U4.4, U5.2):

> UMA meets this use case because the resource server is able to register each resource set (according to boundaries it unilaterally determines) at the authorization server, so that the resource owner can apply policy to it distinctly.

Owner not always present at time of access (U1.6, U5.5):

UMA meets this use case because it is a profile of OAuth that defines an asynchronous authorization grant, meaning that the client's interactions during a resource access attempt do not require a resource owner's interaction.

Owner grants temporary access permissions to a party (U1.7):

UMA meets this use case because the default, mandatory-to-implement permissions associated with a requesting party token (the "bearer" profile) are able to be time-limited and are in a time-limitable JSON Web Token as well.

Owner applies verifiable context-based conditions to authorizations (U2.4, U4.5, U6.3):

UMA meets this use case because a resource owner can configure an authorization server with policies, or an authorization server can apply system-default policies, to demand "trust elevation" when a client requests authorization data, such that a requesting party or client must satisfy authentication, claims-based, or (through extension) any other criteria prior to being issued authorization data.

Owner preconfigures access rights to specific data (U3.1, U6.3):

UMA meets this use case because it defines an asynchronous authorization grant, as described above. Preconfiguration is a case when a resource owner sets policy prior to an access attempt.

Owner adds a new device under protection (U4.1):

UMA meets this use case because it enables a resource owner to associate a device and its corresponding resource server with an authorization server through consenting to the issuance of a protection API token (PAT), enabling the resource server to outsource protection of its resources to the authorization server.

Owner puts a previously owned device under protection (U4.2):

UMA meets this use case because a previous resource owner can revoke a pre-existing PAT if one existed, revoking the previous consent in place, and the new owner can mint a new PAT.

Owner removes a device from protection (U4.3):

UMA meets this use case because the resource owner can revoke the PAT.

Owner revokes permissions (U4.6):

UMA meets this use case because the resource owner can configure the authorization server to revoke or terminate an existing permission. The default, mandatory-to-implement requesting party token profile ("bearer") requires runtime token introspection, ensuring relatively timely retrieval of a revoked permission (barring authorization server caching policy). Other profiles may have different results.

Owner grants access only to authentic, authorized clients (U7.1, U7.2):

UMA meets this use case because it enables OAuth as well as OpenID Connect authentication of clients, including dynamic authentication, and also enables resource owners to configure authorization servers with policy, such that only desired clients wielded by desired requesting parties are given access to the owner's resources.

7. Security Considerations

This specification re-uses several existing specifications, including OAuth and UMA, and hence the security-related discussion in those documents is applicable to this specification. A reader is encouraged to consult [RFC6819] for a discussion of security threats in OAuth and ways to mitigate them. On a high level, the security guidance provided in [I-D.iab-smart-object-architecture] will help to improve security of Internet of Things devices in general.

Despite all the available guidance it is nevertheless worthwhile to repeat the most important aspects regarding the use of access tokens, which are a core security mechanism in the OAuth / UMA specifications.

Safeguard bearer tokens: Client implementations MUST ensure that bearer tokens are not leaked to unintended parties, as they will be able to use them to gain access to protected resources. This is the primary security consideration when using bearer tokens and underlies all the more specific recommendations that follow. This document also outlines the use of proof-of-possessions, which provide stronger security properties than bearer tokens and their use is RECOMMENDED.

Validate TLS certificates: TLS/DTLS clients MUST validate the certificates received during the handshaking procedure. TLS/DTLS is used heavily in OAuth/UMA between various parties. Failure to verify certificates will enable man-in-the-middle attacks.

Always use TLS/DTLS: The use of TLS/DTLS is mandatory for use with OAuth as a default. Particularly when bearer tokens are exchanged the communication interaction MUST experience communication security protectoin using TLS (or DTLS). Failing to do so exposes bearer tokens to third parties and could consequently give attackers unintended access. Proof-of-possession tokens on the other hand do not necessarily require the use of TLS/DTLS but TLS/ DTLS is RECOMMENDED even in those cases since TLS/DTLS offers many desireable security properties, such as authentication of the server side.

Issue short-lived tokens: Authorization servers SHOULD issue short-lived tokens. Using short-lived bearer tokens reduces the impact of them being leaked and allows easier revocation in scenarios where resource servers are offline.

Issue scoped tokens: Authorization servers MUST issue tokens that restrict tokens for use with a specific resource server and contains appropriate entitlements to control access in a fine-grained fashion.

8. IANA Considerations

This document does not require actions by IANA.

9. Acknowledgements

This is the first version of the document. We appreciate feedback.

10. References

10.1. Normative References

[I-D.hardjono-oauth-umacore]
Hardjono, T., Maler, E., Machulak, M., and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", draft-hardjono-oauth-umacore-12 (work in progress), February 2015.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-41 (work in progress), January 2015.

[I-D.ietf-oauth-dyn-reg]
ietf@justin.richer.org, i., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", draft-ietf-oauth-dyn-reg-24 (work in progress), February 2015.

[I-D.ietf-oauth-introspection]
ietf@justin.richer.org, i., "OAuth 2.0 Token Introspection", draft-ietf-oauth-introspection-05 (work in progress), February 2015.

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-32 (work in progress), December 2014.

[I-D.ietf-oauth-pop-architecture]
Hunt, P., ietf@justin.richer.org, i., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-01 (work in progress), March 2015.

[I-D.tschofenig-ace-oauth-bt]
Tschofenig, H., "The OAuth 2.0 Bearer Token Usage over the Constrained Application Protocol (CoAP)", draft-tschofenig-ace-oauth-bt-01 (work in progress), March 2015.

[I-D.tschofenig-ace-oauth-iot]
Tschofenig, H., "The OAuth 2.0 Internet of Things (IoT) Client Credentials Grant", draft-tschofenig-ace-oauth-iot-01 (work in progress), March 2015.

[I-D.wahlstroem-ace-oauth-introspection]
Wahlstroem, E., "OAuth 2.0 Introspection over the Constrained Application Protocol (CoAP)", draft-wahlstroem-ace-oauth-introspection-00 (work in progress), October 2014.

[OIDC]   Sakimura, N., "OpenID Connect Core 1.0 incorporating Errata Set 1", http://openid.net/specs/openid-connect-core-1_0.html, November 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012.

[RFC6819] Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, January 2013.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

10.2. Informative References

[I-D.iab-smart-object-architecture]
Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", draft-iab-smart-object-architecture-06 (work in progress), October 2014.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-02 (work in progress), February 2015.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

Authors' Addresses

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Eve Maler
Forgerock

Email: eve.maler@forgerock.com

Erik Wahlstroem
Nexus Technology
Sweden

Email: erik.wahlstrom@nexusgroup.com
URI: https://www.nexusgroup.com

Samuel Erdtman
Nexus Technology
Sweden

Email: samuel.erdtman@nexusgroup.com
URI: https://www.nexusgroup.com