Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 3, 2016

S. Baer
B. Haberstumpf
Elektrobit

August 31, 2015

Lightweight Token authentication (LTA) 1.0
draft-baer-lightweight-token-authentication-01

Abstract

This document contains a specification for a token authentication mechanism that is sufficiently resource-friendly to use it on small embedded or mobile devices. The three involved parties are a service consumer (CON) a service provider (SP) and an authentication provider (AP). The authentication provider decouples authentication and the actual service that is consumed. It also serves as anonymizer. The consumer authenticates at the authentication provider, requests one or more authorization tokens and redeems those tokens when accessing the service provider's offered services.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 3, 2016.

Table of Contents

1. Introduction

The Lightweight Token Authentication is motivated by the need for a simple and resource friendly authentication mechanism mainly targeted (but not limited to) the use in embedded devices. This document specifies the LTA authentication protocol.

1.1. Requirements Language

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 RFC 2119 [RFC2119] .

1.2. Terminology

Authentication provider (AP): Network component that accepts authentication requests and issues authentication tokens to a consumer.

Consumer (CON): Client application that wants to use a service offered by a service provider.

Service provider (SP): Network component that offers a service to the consumer.

Service identification URI (SIU): URI of the service used to identify a service in a token. Not necessarily the URI under which the service can be reached.

Service permission URI (SPU): URI that represents a permission on the service.

Time to use (TTU): Recommended time span for which a token should be used by a consumer before the consumer requests a new token.

1.3. Design Goals

The design of the Lightweight Token Authentication (LTA) aims to reach the goals described in the following sub-sections.

1.3.1. Low Resource Usage and Simple Implementation in the Consumer

It is the most important goal of the LTA to be able to run on small embedded computers (like electronic control units in a car or motorcycle) and mobile devices. Therefore the design keeps the software dependencies and resource requirements low on the consumer side. Calculation and traffic intensive parts of the LTA are shifted to the authentication provider and the service provider.

The aim is to keep the following points down in order of importance:

1. Network traffic

2. Memory consumption in the consumer

3. CPU consumption in the consumer

4. Dependencies on 3rd-party software modules in the consumer

1.3.2. Works with Standard HTTP + TLS

HTTP software libraries can be considered a commodity - just like the availability of network connections that can transport HTTP. LTA therefore uses HTTP as its transport protocol and TLS as transport security.

1.3.3. Small Network Overhead

LTA aims to keep the additional network overhead low. Token requests and the token headers are both designed to stay as small as reasonable while still being in alignment with the philosophy of HTTP. Another goal is to keep the total number of request for authentication down.

Typical sizes for LTA tokens including signature are below 500 bytes.

The authors are clear on the fact that the biggest part of the network overhead comes from transport security, namely the use of TLS (see Section 2.1) but they consider the advantages of using such widely accepted standards to be worth choosing them over proprietary protocol stacks that might reduce network overhead further.

1.3.4. Robustness

LTA works without a permanent connection between the authentication provider and the service provider in order to remove one possible cause for a service outage.

LTA allows consumers to repeat requests. This is important for mobile or embedded devices with an unstable network connection. Service providers that are based on LTA are encouraged to design their services so that they also accept request repetition.

1.3.5. Support a Stateless Service Provider

The LTA does not force the service provider to manage state. Many services are intentionally designed stateless, especially to allow for efficient scaling.

1.3.6. Anonymization

The separation into authentication provider and service provider aims to anonymize the consumer towards the service provider. This protects the consumer's privacy while still allowing the provider to offer services with access restrictions.

While the authentication provider knows the identity of the consumers - or even the users behind the consumer applications - it does not know, what data the consumers send to a service.

From the service provider's perspective the consumer is anonymous. The token that the user offers to the service provider does not allow the service provider to identify a consumer.

To ensure anonymization, service provider and authentication provider must be separate entities which should also be clearly separated on an organizational level. Especially the authentication provider must not disclose a token-to-consumer mapping to the service provider.

See Section 17.2 for limitations on the achieved level of anonymity.

2. Preconditions

2.1. Transport security

LTA relies on transport encryption (as opposed to message encryption) between the three involved parties.

Consumer, authentication provider and service provider MUST all use Transport Layer Security (TLS) in version 1.2 or newer. See [RFC5246] for details.

2.2. Authentication of the communication partners

The consumer MUST verify the authenticity of the communication partners - i.e. authentication provider and service provider.

The recommended way is using TLS server certificates. See [RFC5246] section 7.4.2 for details.

The service provider indirectly authenticates the authentication provider via the token signature. See Section 9 for more information.

2.3. Syntax Definitions

The syntax definitions in this document use the "Augmented Backus Naur Form" (ABNF) as defined in [RFC5234] .

2.4. Base Elements of the Syntax Definitions

The following recurring syntax elements are used throughout the document and therefore introduced here.

LTA's protocol version (and implicitly the token format):

lta-version = 1*DIGIT "." 1*DIGIT

In this revision of the LTA specification the LTA version is 1.0.

The URI that identifies a service:

service-identification-uri

3. Authentication Overview

The authentication provider is a service that provides consumer authentication and authorization. It also anonymizes the consumers towards the service provider. Instead of authenticating themselves directly at the service provider the consumers use the authentication provider as authentication authority.

In short a successful authentication follows these steps:

1. Consumer uses the authentication offers discovery to find out where to request tokens.

2. Consumer requests a token at the authentication provider.

3. Authentication provider authenticates the consumer and issues a token.

4. Consumer redeems the token at the 3rd party service.

5. Service provider verifies the authenticity of the token.

4. Consumer Authentication at the Authentication Provider

The consumer MUST provide credentials with all requests to the authentication header. This rule applies to the authentication offer list request and all other (subsequent) requests to the authentication provider.

An authentication provider SHOULD at least support the "Basic" authentication scheme as defined in [RFC2617] section 2. Other authentication schemes can be used between the consumer and the authentication provider also but implementers should keep in mind that complicated authentication schemes are in conflict with the design goal to keep the effort low for the consumer.

A consumer using basic authentication would set the following header:

authorization = "Authorization:" 1*SP "Basic" 1*SP credentials

The credentials are encoded with the "Base64 Content-Transfer-Encoding" described in [RFC2045] section 6.8.

Example:

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Since basic authentication can be considered a commodity of the widely available web servers the implementation is not in the scope of this document.

5. Authentication Offers Discovery

The authentication provider has a single entry point. The consumer accesses this entry point and gets a list of URIs that it uses to find out for which services the authentication provider offers tokens.

5.1. Consumer Request for Authentication Offer List

The consumer requests the offer list with the following request:

```
authentication-offer-uri = "GET" SP ap-entry-uri "/"
            lta-version-number
```

Where "ap-entry-uri" is the entry URI of the authentication provider that the consumer must know.

The mandatory protocol version number allows to easily route consumers that use different protocol versions to the right authentication provider. If the authentication provider supports multiple versions it MUST offer one offer discovery URI in the form of "authentication-offer-uri" for each supported protocol version.

5.1.1. Authentication Offer List Response

The authentication provider first checks the credentials of the consumer. If the user has a valid account, the authentication provider creates a list of all services the consumer is registered for.

5.1.2. Access to Offer List Granted

If the consumer's credentials are valid, the authentication provider MUST answer with 200 (OK).

The offer response body MUST contain a map of URIs with the Mime type "application/vnd.uri-map".

Each line of the map has the following format:

```
offer-list-entry = service-identification-uri ">"
              token-request-uri CR LF
```

So for each service the map tells the consumer under which URI it must request the authentication token.

Example URI map:

```
https://www.example.org/wiki>https://example.com/
  ap_node234/1.0/https%3A%2F%2Fexample.org%2Fwiki
blog.example.org>https://example.com/ap/1.0/blog.example.org
```

Note that in the example above there is an extra newline an indentation due to line length restrictions. Both are not present in the actual URI map.

The authentication provider MUST only list those services for which it actually offers a token to the individual consumer. That means a consumer can expect that it has the necessary rights to request a token under each of the listed URIs.

If the authentication provider does not offer authentication for any services to the consumer, the offer list MUST be empty.

5.1.3. Offer List Request Denied Because of Missing Credentials

The authentication provider MUST answer 401 (Unauthorized) if the consumer does not provide the required authentication header.

5.1.4. Offer List Request Denied Because of Invalid Credentials

The authentication provider MUST answer 401 (Unauthorized) if the credentials the consumer used are not valid.

6. Authentication Step by Step

The following diagram depicts the sequence for a successful authentication:

6.1. Consumer Requests Token

The consumer MUST initiate the authentication by requesting a token from the authentication provider with the following request:

token-request = "GET" SP token-request-uri

Where the "token-request-uri" is a URI that the consumer previously discovered via an offer list request to the authentication provider. See Section 5 for details.

Example:

GET https://example.com/ap/1.0/https%3A%2F%2Fexample.org%2Fwiki

6.2. Consumer Sets "Accept" Headers

The consumer MUST NOT set the "Accept" header since the token format is dictated by the common denominator of authentication provider and service provider.

The consumer CAN set an "Accept-Charset" header that has one of the following values:

- UTF-8
- US-ASCII

Tokens only contain 7bit ASCII characters. See Section 6.4 for details.

The consumer CAN set an "Accept-Language"" header. See [RFC7231] section 5.3.5. for more information. This only has an influence on the language of error messages issued by the authentication provider. Those error messages are used in response to illegal requests or other abnormal situations. Note that authentication providers are allowed to ignore this header and only provide English error messages. See Section 10.10 for details.

6.3. Authentication Provider Authenticates the Consumer

The authentication provider checks the credentials provided by the consumer.

6.4. Creating a Token

An authentication token is a credential issued by the authentication provider and used by the consumer to gain access to a service provider's services.

The authentication token MUST only contain characters from the 7bit ASCII character set. This way maximum compatibility across different character sets is ensured. For example UTF-8 is compatible to 7bit ASCII.

An authentication token created by the authentication provider MUST have the following format:

```
token = token-payload " " signature
```

The token payload is defined as follows:

```
token-payload = lta-version " " service-specification " " expiration
            " " time-to-use
```

The service specification identifies service and permissions:

```
service-specification = service-identification-uri ( *( "|"
            service-permission-uri ) / "*" )
```

The signature is defined as follows:

signature = signature-info "|" signature-container

signature-info = (hash-algorithm "|" encryption) / container-format

Note that the payload of the token and the signature meta-data are not encrypted since the consumer must be able to read the contents of the token but not to modify them. Refer to Section 6.5 for details. This is important because the consumer has to take the token expiration into account before trying to redeem a token at a service provider. It would cause unnecessary traffic if the consumer tried to redeem an expired token.

The list of service permission URIs can be replaced by an asterisk "*" symbol in case no access restrictions below the service level are necessary.

Examples for service specifications:

https://example.org/blog|*

https://example.org/blog|get|head

blog.example.org editor|admin

6.4.1. The Token Format Explained

There are restrictions that lead to the design of the token format.

1. The consumer MUST be able to copy the received token directly into an HTTP header without modifying it. Therefore only ASCII characters that are allowed as header fields values can be used.

2. The parts the token consists of MUST be separable by the most simple string manipulation. It is designed to be split at separator characters.

    1. Since parts of the token are URIs, only characters that are not allowed in a URI are suitable as separators

    2. The building blocks are split at the " " character

    3. The permissions and signature parts are separated by "|"

The reason why there are two kinds of separators is that we want to be able to vary the number of sub-items.

6.4.2. Token Expiration

The authentication provider MUST calculate an expiration date and time for each token.

The expiration delay of a token MUST be configurable in the authentication provider per service provider URI.

The expiration time in the token MUST be encoded as date and time according to the following pattern as specified in [RFC3339] section 5.6.

```
expiration = year "-" month "-" day "T" hours ":" minutes ":"
              seconds "Z"

year = 4DIGIT month = ( "0" DIGIT ) / 11 / 12

day = ( "0" / "1" / "2") DIGIT / 30 / 31

hours = ( ( "0" / "1") DIGIT ) / 21 / 22 / 23

minutes = ( "0" / "1" / "2" / "3" / "4" / "5") DIGIT

seconds = ( "0" / "1" / "2" / "3" / "4" / "5") DIGIT
```

All expiration timestamps MUST be encoded in Coordinated Universal Time (UTC). If the server is in a different time zone , the authentication provider MUST convert the time accordingly.

```
time-to-use = 1*DIGIT
```

The time to use (TTU) is the number of seconds the token should be used after it was issued.

In simple implementations the time to use is equal to the time to live. In this case the TTU is redundant at first glance, but it is a concession to consumers that do not have proper clock synchronization. See Section 8.3 for details.

In more sophisticated implementations the authentication provider chooses a TTU that is shorter than the difference between token issuing time and expiration time. See Section 11.1.2 for more information.

6.5. Creating the token signature

The authentication provider MUST sign the token payload using a hash algorithm and its private key.

LTA supports exchanging the signing mechanism. It needs an asymmetric encryption in order to create a token signature.

The authentication provider MUST state either the hash algorithm and encryption it uses to create in the token signature or a signature container format. This tells the service provider how to validate the signature.

Refer to Section 10.8 for a list of supported algorithms.

7. Authentication Provider Answers Token Request

Depending on whether or not the authentication provider was able to identify the consumer, the authentication provider either issues a token or tells the consumer that it could not be authenticated.

7.1. Token Request Granted

The authentication provider must answer 200 (OK) if the consumer was authenticated successfully and if the user is authorized to use the requested service or services.

In this case the response body contains the token.

token-response-body = token

The authentication provider MUST mark responses that contain an LTA token with MIME type "application/lta" in the "Content-Type" header.

Example:

```
HTTP/1.1 200 OK Date: Mon, 01 Jan 2015 14:21:16 GMT
Content-Type: application/lta
Content-Length: 451

1.0 org-example-wiki * 2015-01-01T14:21:46Z 25
sha-1|rsa|iQEcBA[...]c+s=
```

The example above is shortened (marked by "[...]") for better readability.

7.2. Token Request Denied Because of Missing Credentials

The authentication provider MUST answer 401 (Unauthorized) if the consumer does not provide the required authentication header.

7.3. Token Request Denied Because of Invalid Credentials

The authentication provider MUST answer 401 (Unauthorized) if the credentials the consumer provided are invalid.

8. Consumer Redeems Token

The consumer sends a requests to the service provider and uses the token as credential. For this it MUST set the following HTTP header in the service request:

auth-header = "Authorization:" SP "Token" token

Note that the consumer does not necessarily need to understand the signature. It can rely on the authenticity of the sender when using HTTPS instead. This allows the consumer to work without needing to decode the signature.

8.1. Consumers Should Not Try to Use Expired Tokens

Consumers SHOULD NOT try to use expired tokens. There are two alternative ways for the consumer to determine if a token is expired:

1. via the absolute expiration time - for a consumer with reliable time synchronization

2. via the time to use (TTU) - works without time synchronization

8.2. Expiration Time

Properly time synchronized consumers should use the absolute expiration time from the token to determine if the token can still be used or if it already expired.

8.3. Time to Use

The time to use (TTU) is a time span between issuing of the token and the time when it is recommended the consumer requests a new token. The TTU may be lower than the difference between issuing and expiration of a token See Section 11.1.2 for more information.

Also consumers like mobile devices and embedded computers might not have proper time synchronization or might use the wrong time zone.

In those cases using the TTU instead of the absolute expiration time is recommended on the consumer side.

The downside of this mechanism is that a part of the TTU given in the token is already used up in the transmission from the authentication provider to the consumer. Since the consumer has no simple means to tell how long this delay is, it will consider the token valid longer than it actually is. The authentication provider CAN mitigate that problem by choosing reasonable safety margin between the expiration of a token and the TTU specified in the token.

9. Service Provider Validates and Authenticates Token

Before the service provider offers its services, it validates and authenticates the token sent by the consumer.

9.1. Validating a token

The service provider MUST validate the token format. If the token format does not conform to this document, the service provider MUST deny the request.

9.2. Checking if the Token Is Addressed to the Right Service Provider

Since consumers might accidently or maliciously try to use an authentication token for the wrong service provider, a service provider MUST always check if a token is really addressed to it. The service provider MUST check if the service identification URI in the token matches the requested service.

Note that the service identification URI is not necessarily the URI under which the service can be reached in a network. Authentication provider and service provider just have to agree on a common URI.

9.3. Checking the Signature

The service provider checks the signature of the token to verify that it has been issued by the authentication provider and has not been modified.

If the signature does not match the token payload, the service provider MUST deny the request.

9.4. Checking if the Token Expired

The service provider compares the expiration date and time from the token to the current date and time. If the token's expiration time lies in the past, the service provider MUST deny the request.

The service provider MUST always use the absolute expiration time from the token. The time-to-use is reserved for consumer use only and MUST be ignored by the service provider. The service provider MUST use clock synchronization (e.g. the "Network Time Protocol", see [RFC5905]) that grants a synchronization quality of typically below one second. See Section 17.1.2 for security considerations of time synchronization.

The service provider MUST take the time differences into account that result from the timezone the service provider is located in. Token expiration times inside the token are always given in UTC (see [RFC3339]). Service providers MUST convert their local time to UTC before evaluating token expiry.

Service providers MUST NOT accept tokens where the expiration time is more than two hours in the future. This is a safety measure in order to avoid long-term valid tokens issued by accident by the authentication provider.

10. Service Provider Answers the Request

10.1. Service access granted

The service provider MUST answer 200 (OK) if the token was successfully validated and authenticated. The response body is the normal service response.

10.2. Access Denied due to Missing Authentication Token

The service provider MUST answer 401 (Unauthorized) if the consumer does not provide the token in the authentication header.

In addition the service provider must set the "WWW-Authenticate" header (see [RFC2617] section 1.2) as follows:

```
challenge = "Token" 1*SP "realm" "=" DQOUTE
            service-identification-uri DQUOTE
```

Example:

```
WWW-Authenticate: Token realm="https://example.org/blog"
```

10.3. Access Denied Because of Illegal Token Format

The service provider MUST answer 400 (Bad Request) if the token or parts of the token do not conform to this document.

10.4. Access Denied Because of Signature Mismatch

The service provider MUST answer 401 (Unauthorized) if the token signature and does not match the token payload.

10.5. Access Denied Because of Unsupported Signing Mechanism

If the service provider does not understand the encryption algorithm or hash function used for signing the token it MUST answer 400 (Bad Request).

The response MUST contain a header field listing the supported hash algorithms:

```
accept-hash-header = "Accept-Token-Hashes:" SP hash *( "," SP hash)
```

The response MUST contain a header field listing the supported ciphers:

```
accept-cipher-header = "Accept-Token-Ciphers:" SP cipher *( "," SP
            cipher)
```

Example:

```
Accept-Token-Hashes: sha-1, sha-256
Accept-Token-Ciphers: rsa
```

10.6. Access Denied Because of Expired Token

The service provider MUST answer 401 (Unauthorized) if the token expired.

10.7. Access Denied Because of Insufficient Permissions

The service provider MUST answer 403 (Forbidden) if the rights specified in the token are not sufficient to execute the service request.

10.8. Supported Signing Mechanisms

LTA is designed to support different token representations in order to be able to replace the signing mechanism when a more efficient or more secure algorithm is available.

This document therefore contains only a very short list of supported hashes and cyphers. While requiring mandatory implementations is good for interoperability, it would be unwise to use mechanisms that are not cryptographically secure anymore.

The hash function textual names are as defined in in the IANA registry. [IANA1]

Examples:

- sha-1
- sha-256

The textual names for encryptions follow the same style.

Examples

- rsa
- ecc

As long as AP and SP agree, they can use implementations not listed here.

10.9. Using a Signature container

Signature container formats contain the signature plus the meta information on how to verify it and can be used as an alternative to encoding the hash function an encryption algorithm. While in principle any container format could be used, some technical restrictions apply:

1. The signature containers encoding must only contain characters allowed in an HTTP header field.
2. The container should be small because it otherwise conflicts with the goal of low network overhead.

The benefit for using standard containers is that they often come with a full infrastructure for key distribution and revoking (both mechanisms outside of the scope of this document).

10.10. Error reporting in the response body

When connecting to web services, it is useful to have clear error messages in case something went wrong. The following recommendations apply to both, the authentication provider and the service provider.

The response body of a response with an "4xx" error code should contain a clear text description of the error cause.

The error message SHOULD respect the language requested if an Accept-Language header was set in the request. At least English error messages SHOULD be supported.

Unless specified explicitly, the error messages do not have to be machine readable.

Error messages MUST NOT disclose confidential information to the consumer. Especially error messages MUST NOT contain information that helps an attacker to guess credentials.

11. Optimizations

11.1. Authentication Provider Optimizations

11.1.1. Authentication Provider Sets Cache Header of a Token

The authentication provider MAY set the following cache header:

```
cache-control = "cache-control:" SP "private," SP "max-age="
              time-to-use
```

The cache directive "private" tells all involved parties that caching is only supposed to happen in the consumer.

Where the time to use in seconds is used as maximum cache age of the response.

```
time-to-use = 1*DIGIT
```

Example:

```
cache-control: private, max-age=25
```

This is useful for consumers that do not implement their own consumer side token cache and rely on the caching their HTTP client library offers. Otherwise it is superfluous.

11.1.2. Issuing a new Token Shortly Before the Existing Token Expires

The authentication provider SHOULD introduce a configurable time span (TTU) that is smaller than the difference of token issuing time to expiration time. The TTU is used to determine when a new token must be issued. This helps to avoid situations where the consumer gets a token that is almost expired.

11.1.3. Token Representation Compatibility with Service Provider

The applicable token formats are dictated by the common denominator between authentication provider and service provider. For each service the authentication provider SHOULD store and respect the token representations the service provider understands.

11.2. Service Provider Optimizations

11.2.1. Token Cache on Service Provider Side

Service providers MAY cache the result of a token evaluation until the token expires, especially for services where a series of request can be expected during the life time of a token.

Note that in a clustered service this would have to be centralized potentially introducing a new point of failure and requiring additional network communication.

Therefore in clustered environments a node-local token cache is recommended in combination with consumer affinity (e.g. IP affinity).

11.3. Consumer Optimizations

11.3.1. Caching the Token Request URIs

The Consumer SHOULD cache the token request URIs listed by the authentication provider. This removes unnecessary subsequent discovery traffic.

The recommended way is doing this is on application layer.

If a token request fails, a consumer MUST invalidate its token request URI cache, because it is likely that either the URI or the consumers permissions changed.

For security reasons the consumers MUST NOT cache the token request URIs indefinitely. The recommended caching time is a day.

Example:

An embedded device requests the token offer list and keeps the result cached in RAM until the next power cycle or day.

11.3.2. Using Tokens until They Expire

Although it is possible that consumers request a token before each service request, this introduces unnecessary network traffic if the last token is not yet expired.

The Consumer SHOULD reuse the token until the token expired.

12. Limitations

12.1. No Token Cache on Authentication Provider Side

While returning cached resources on GET requests usually is a desired behaviour, in case of an LTA token this is not the case. The reason is the way the TTU works (see Section 8.3).

Consumers use the TTU to calculate the remaining time the token can be used based on the time they received the token. If a consumer decides to send a subsequent request for a token to the authentication provider and got a cached token, this simple mechanism would break. Therefore authentication providers MUST answer each token request with a fresh token.

13. Permission Handling

13.1. Service identification URIs

A service identification URI (short "SIU") uniquely identifies a service. It MUST be agreed upon by both, the service provider and the authentication provider.

If permissions on a sub-service level are necessary, then the service provider MUST define a service permission URI for each permission.

13.2. Choosing Service Identification URIs and Permission URIs

It is not mandatory that service identification URIs or permission URIs are can be dereferenced.

While it is easier to understand if the service identification URI is identical to the URI the service can be reached, it is not required.

Note that the token - and therefore the SIUs are transmitted in the HTTP header. Although the HTTP specification does not define a size limit on the HTTP headers, in real-world scenarios the web servers use default limits between four and eight kilobytes.

This fact and the goal that LTA should be resource-friendly both suggest that service providers define short URIs to identify services.

Permission URIs should be relative to the service identification URI to save more space.

Examples:

1. Sub-service level permissions for a restful HTTP service could be the request verbs "get", "put" or "delete".

2. For an accounting system permissions could be on artifact level "invoices", "cancellations" or "customers".

3. A service could also use roles instead of permissions "admin", "user" or "guest".

13.3. Permission URI wildcard

If no sub-service level permissions are needed, the authentication provider should use the wildcard "*" instead of listing of all permissions.

Example:

A service provider defines dash-notation URIs for its services like "org-example-wiki". The service provider does not impose restrictions on the use of the service and marks this with the "*" wildcard.

13.4. Parameterized Permissions

Imagine a user payed for a storing ten photos with a total size of not more than 20 MiB on a service. The service provider and the authentication provider agreed on a common scheme for representing this as a permission URI which looks like:

store?max-items=10&max-size=20M

It is a valid URI and it contains parameters. Given that the authentication provider knows how to build this URI and the service provider knows how to interpret it.

14. Examples

14.1. Token Requests

Consumer:

```
GET https://example.com/ap/https%3A%2F%2Fexample.org%2Fblog
Authorization: Basic ZXhhbXBsZV91c2VyOmV4YW1wbGVfcGFzc3dvcmQ=
[...]
```

Authentication provider:

```
HTTP/1.1 200 OK
Date: Mon, 01 Jan 2015 14:21:16 GMT
Content-Type: application/lta
Content-Length: 451

1.0 https://example.org/blog|get|post|delete 2015-01-01T14:21:46Z 25
sha-1|rsa|iQEcBAABAgAGBQJT/yYfAAoJEBYE2BQYko+r47sH/jZNWgVpzoVXwfdFMVd
FkZYG69qDnYNy3rfxw8HtYWa7x1VEngo9x79G+Bk5GhlG62rNpyZAFc63pi9/9eddZEO
BBBwqWu7RA/h24DHfp0ngT0MO+H0zLldzTMSLCVkYYp+O3K5HlIqGhA9Rj32XKYBwjiM
wPBoIYAcTzbUOb9kih0Ru3jGp/7+K/FbQPZHYK98znvKN/r81PqK0LM3KFpBi1SL+gpv
IDm1sz/GjXGlAtBfMViHPcY6Vw6kjhpbuYEqPkOWty5gjhUntrrCyKpA069COR64bLqC
eIM7++3E5rZvH1dIYZ86u629xNna5Tj+TJSkev7Jnlfw0YFoc+s=
```

Note that the line breaks in the example HTTP body have been added for readability and are _not_ present in an actual token.

14.2. Service Request with LTA Token

Consumer:

```
GET https://example.org/blog/2015/01/01/img42.jpg

Authorization: Token 1.0 https://example.org/blog|get|post|delete
2015-01-01T14:21:46Z 25 sha-1|rsa|iQEcBAABAgAGBQJT/yYfAAoJEBYE2BQYko+
r47sH/jZNWgVpzoVXwfdFMVdFkZYG69qDnYNy3rfxw8HtYWa7x1VEngo9x79G+Bk5Ghl
G62rNpyZAFc63pi9/9eddZEOBBBwqWu7RA/h24DHfp0ngT0MO+H0zLldzTMSLCVkYYp+
O3K5HlIqGhA9Rj32XKYBwjiMwPBoIYAcTzbUOb9kih0Ru3jGp/7+K/FbQPZHYK98znvK
N/r81PqK0LM3KFpBi1SL+gpvIDm1sz/GjXGlAtBfMViHPcY6Vw6kjhpbuYEqPkOWty5g
jhUntrrCyKpA069COR64bLqCeIM7++3E5rZvH1dIYZ86u629xNna5Tj+TJSkev7Jnlfw
0YFoc+s=
[...]
```

Again the line breaks are for readability an do not exist in a real token.

Service provider:

```
HTTP/1.1 200 OK
Date: Mon, 01 Jan 2015 14:21:17 GMT
Content-Type: image/jpeg
Content-Length: 513017

[...]
```

15. Acknowledgements

This document's structure is based on a template by Elwyn Davies (initial version by Pekka Savola).

We'd like to thank Peer Sterner for his thorough reviews of the early drafts.

Thomas Fleischmann

16. IANA Considerations

This document uses the existing IANA registry for "Hash Function Textual Names" in Section 10.8 which was introduced in [RFC4572] (see [IANA1] for a list of values) .

This document defines a public key protocol value in Section 10.8 . Figure 1 contains the initial values.

Public Key Algorithm	Reference
"rsa"	RFC2313

Figure 1: IANA Public Key Algorithm Textual Name Registry

The mime type "vnd/uri-map" was registered with IANA on July 21st 2015.

The mime type "application/lta" will be requested at the the IANA at the end of the review process for the LTA in order to be able to react on changes proposed by the reviewers.

17. Security Considerations

The following section lists possible attack vectors and mitigation strategies (if applicable).

17.1. Attack Vectors

17.1.1. Flooding of the Service Provider

Since tokens described in this document are intentionally designed to be reused until they expire, they do not grant any protection against flooding of the service provider

Tokens with a usage limit would either require the service provider to consult the authentication provider for each service request or require holding a connection state at the service providers.

Service providers must implement their own flood protection mechanisms - independently of the LTA.

17.1.2. Time Synchronization Attack

Depending on the expiry delay a few seconds difference in the token expiration between what the authentication provider specified and the expiry in the service provider are no problem. Whereas minutes or more would open a potential attack vector where outdated tokens could be used.

Therefore proper time synchronization of the AP and SP is crucial.

Attackers could try to fake the time source of either the AP or SP. Therefore both must make sure to use a secure and trusted time source.

17.1.3. Token Hijacking

One of the goals of the LTA is to provide anonymization of the consumer towards the service provider. Since the service provider does not know the sender, it can not verify if the token belongs to the sender or was copied from another consumer.

For this reason it is recommended to use LTA alone only for services that do not disclose personal or confidential data. If service designers plan to use LTA for such a service it is recommended that they use additional user authentication.

17.1.4. Man-in-the-Middle

If an attacker manages to intercept the communication between the SP and the AP, the attacker could try to impose an authentication provider towards the unknowing consumer. The consumer would then either disclose its credentials to the attacker or the attacker would intercept and abuse the issued token.

In a different scenario the attacker could impose a service provider to collect tokens which it could redeem at the real service provider.

Consumers need to verify the identity of both, authentication provider and service provider in order to prevent man-in-the-middle attack.

17.1.5. Replay Attacks

If attackers are able to record a token during transmission, they can try to run a replay attack. Tokens that are not expired can be used in a replay attack.

The first countermeasure is the mandatory use of TLS to prevent eavesdropping. If you are really concerned about replay attacks, the service provider may use the token cache to accept each token only once. The tradeoff is that this contradicts the design goal of keeping the number of requests to the authentication provider down.

17.1.6. Service Permission Information Leaking

Since tokens contain service identification URIs, an attacker could try to get tokens to gather information about the services a consumer is using and the associated permissions.

Consumers are responsible for protecting their token on the local machine and making sure they are addressing them at the authentic service provider.

Losing a token has - at least for the time the token is valid - the same effect as losing other credentials like user name and password.

17.1.7. Addressing a Token to the Wrong Service

Token based authentication without callback to the authentication provider from the service provider carries the risk of a malicious consumer trying to feed a token to the service provider that is a valid token addressed at a different service provider. See also Section 9.2 for more information.

Therefore service providers must check the service identification URI in the token.

17.1.8. Using Token Request URIs That Are Not Valid Anymore

If the consumer cached a token request URI which is not used anymore by an authentication provider and that URI belongs to a different domain there is the chance of an attacker to impose the

authentication provider. For this the attacker would need to gain control over the domain, create a matching certificate and deploy what looks like the token granting part of the authentication provider.

It is an unlikely scenario but theoretically possible. To mitigate the potential damage consumers SHOULD NOT cache token request URIs indefinitely.

17.1.9. Compromized Authentication Provider or Stolen AP Secrets

If attackers successfully take over an authentication provider or copy the authentication providers secret, they can use this to create valid tokens.

It is recommended to use a signing mechanism that supports the revocation of encryption keys so that at least after the attack was discovered, the compromised key can be rejected.

17.2. Anonymization Limitations

A malicious service provider can collect and compare meta-data of a service request in order to break the anonymization. In the simplest case IP addresses already help narrowing down the consumer. More sophisticated methods include fingerprinting (e.g. by using additional HTTP headers or timing)

If anonymity is an important requirement then consumers can only prevent metadata exploitation by adding additional anonymization measures (like using the TOR network).

18. References

18.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <http://www.rfc-editor.org/info/rfc3339>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

18.2. Informative References

[IANA1] IANA, "Textual names for hash functions", 2015, <https://www.iana.org/assignments/hash-function-text-names/hash-function-text-names.xhtml>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, DOI 10.17487/RFC2617, June 1999, <http://www.rfc-editor.org/info/rfc2617>.

[RFC4572] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 4572, DOI 10.17487/RFC4572, July 2006, <http://www.rfc-editor.org/info/rfc4572>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <http://www.rfc-editor.org/info/rfc5905>.

Authors' Addresses

Sebastian Baer
Elektrobit
Am Wolfsmantel 46
Erlangen 91058
Germany

Email: sebastian.baer@elektrobit.com

Bernd Haberstumpf
Elektrobit
Am Wolfsmantel 46
Erlangen 91058
Germany

Email: bernd.haberstumpf@elektrobit.com