OAuth H. Tschofenig
Internet-Draft Nokia Siemens Networks
Intended status: Informational P. Hunt
Expires: March 10, 2013 Oracle Corporation
September 6, 2012

OAuth 2.0 Security: Going Beyond Bearer Tokens
draft-tschofenig-oauth-security-00.txt

Abstract

The OAuth working group has finished work on the OAuth 2.0 core protocol as well as the Bearer Token specification. The Bearer Token is a TLS-based solution for ensuring that neither the interaction with the Authorization Server (when requesting a token) nor the interaction with the Resource Server (for accessing a protected resource) leads to token leakage. There has, however, always been the desire to develop a security solution that is "better" than Bearer Tokens (or at least different) where the Client needs to show possession of some keying material when accessing a Resource Server. This document tries to capture the discussion and to come up with requirements to process the work on solutions.

This document aims to discuss threats, security requirements and desired design properties of an enhanced OAuth security mechanism.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction 3
2. Terminology 4
3. Security and Privacy Threats 5
4. Threat Mitigation 7
4.1. Confidentiality Protection 7
4.2. Sender Constraint 8
4.3. Key Confirmation 8
4.4. Summary 9
5. Requirements 11
6. Security Considerations 16
7. Next Steps 17
8. IANA Considerations 18
9. Acknowledgements 19
10. References 20
10.1. Normative References 20
10.2. Informative References 20
Authors' Addresses 22

1. Introduction

OAuth 1.0 [RFC5849] included a mechanism for putting a digital signature (when using asymmetric keys) and a keyed message digest (when using symmetric keys) to a resource request when presenting the OAuth access token. OAuth 2.0 [I-D.ietf-oauth-v2] generalized the protocol and the Bearer Token security specification [I-D.ietf-oauth-v2-bearer] is close to publication as an RFC.

Figure 1 shows the OAuth 2.0 exchange at an abstract level and illustrates the main entities. For most parts of this document the focus is on the interaction between the Client and the Authorization Server and between the Client and the Resource Server.

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

Figure 1: OAuth: Abstract Protocol Flow

From a security point of view the following aspects of the OAuth 2.0 specification are worth mentioning:

- Standardization of a JSON-based format and the content of the access token are still work in progress [I-D.ietf-oauth-json-web-token]. The same is true for the JSON-based security mechanisms.

- The interaction to obtain an access token in step #1 mandates to implement and to use TLS with server-side authentication to protect the confidentiality of the transmitted information.

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 RFC 2119 [RFC2119].

This document uses the terminology defined in RFC 4949 [RFC4949]. The terms 'keyed hash' and 'keyed message digest' are used interchangable. For privacy related matters we utilize the terminology defined in [I-D.iab-privacy-considerations].

This document uses OAuth 2.0 terminology [I-D.ietf-oauth-v2]. In particular, the terms Client, Resource Server, Authorization Server, and Access Token are used.

3. Security and Privacy Threats

The following list presents several common threats against protocols utilizing some form of tokens. This list of threats is based on NIST Special Publication 800-63 [NIST800-63]. We exclude a discussion of threats related to any form of identity proofing and authentication of the Resource Owner to the Authorization Server since these procedures are not part of the OAuth 2.0 protocol specificaiton itself.

Token manufacture/modification:

An attacker may generate a bogus tokens or modify the token content (such as authentication or attribute statements) of an existing token, causing Resource Server to grant inappropriate access to the Client. For example, an attacker may modify the token to extend the validity period. A Client may modify the token to have access to information that they should not be able to view.

Token disclosure: Tokens may contain personal data, such as real name, age or birthday, payment information, etc.

Token redirect:

An attacker uses the token generated for consumption by the Resource Server to obtain access to another Resource Server.

Token reuse:

An attacker attempts to use a token that has already been used once with a Resource Server. The attacker may be an eavesdropper who observes the communication exchange or, worse, one of the communication end points. A Client may, for example, leak access tokens because it cannot keep secrets confidential. A Client may also re-use access tokens for some other Resource Servers. Finally, a Resource Server may use a token it had obtained from a Client and use it with another Resource Server that the Client interacts with. A Resource Server, offering relatively unimportant application services, may attempt to use an access token obtained from a Client to access a high-value service, such as a payment service, on behalf of the Client using the same access token.

We excluded one threat from the list, namely 'token repudiation'. Token repudiation refers to a property whereby a Resource Server is given an assurance that the Authorization Server cannot deny to have created a token for the Client. We believe that such a property is

interesting but most deployments prefer to deal with the violation of this security property through business actions rather than by using cryptography.

4. Threat Mitigation

The purpose of this section is to discuss ways to mitigate the threats without taking the current working group status into consideration.

A large range of threats can be mitigated by protecting the content of the token, using a digital signature or a keyed message digest. Alternatively, the content of the token could be passed by reference rather than by value (requiring a separate message exchange to resolve the reference to the token content). To simplify the subsequent description we assume that the token itself is digitally signed by the Authorization Server and therefore cannot be modified.

To deal with token redirect it is important for the Authorization Server to include the identifier of the intended recipient - the Resource Server. A Resource Server must not be allowed to accept access tokens that are not meant for its consumption.

To provide protection against token disclosure two approaches are possible, namely (a) not to include sensitive information inside the token or (b) to ensure confidentiality protection. The latter approach requires at least the communication interaction between the Client and the Authorization Server as well as the interaction between the Client and the Resource Server to experience confidentiality protection. As an example, Transport Layer Security with a ciphersuite that offers confidentiality protection has to be applied. Encrypting the token content itself is another alternative. In our scenario the Authorization Server would, for example, encrypt the token content with a symmetric key shared with the Resource Server.

To deal with token reuse more choices are available.

4.1. Confidentiality Protection

In this approach confidentiality protection of the exchange is provided on the communication interfaces between the Client and the Resource Server, and between the Client and the Authorization Server. No eavesdropper on the wire is able to observe the token exchange. Consequently, a replay by a third party is not possible. An Authorization Server wants to ensure that it only hands out tokens to Clients it has authenticated first and who are authorized. For this purpose, authentication of the Client to the Authorization Server will be a requirement to ensure adequate protection against a range of attacks. This is, however, true for the description in Section 4.2 and Section 4.3 as well. Furthermore, the Client has to make sure it does not distribute the access token to entities other

than the intended the Resource Server. For that purpose the Client will have to authenticate the Resource Server before transmitting the access token.

4.2. Sender Constraint

Instead of providing confidentiality protection the Authorization Server could also put the identifier of the Client into the protected token with the following semantic: 'This token is only valid when presented by a Client with the following identifer.' When the access token is then presented to the Resource Server how does it know that it was provided by the Client? It has to authenticate the Client! There are many choices for authenticating the Client to the Resource Server, for example by using client certificates in TLS [RFC5246], or pre-shared secrets within TLS [RFC4279]. The choice of the preferred authentication mechanism and credential type may depend on a number of factors, including

- security properties
- available infrastructure
- library support
- credential cost (financial)
- performance
- integration into the existing IT infrastructure
- operational overhead for configuration and distribution of credentials

This long list hints to the challenge of selecting at least one mandatory-to-implement Client authentication mechanism.

4.3. Key Confirmation

A variation of the mechanism of sender authentication described in Section 4.2 is to replace authentication with the proof-of-possession of a specific (session) key, i.e. key confirmation. In this model the Resource Server would not authenticate the Client itself but would rather verify whether the Client knows the session key associated with a specific access token. Examples of this approach can be found with the OAuth 1.0 MAC token [RFC5849], Kerberos [RFC4120] when utilizing the AP_REQ/AP_REP exchange (see also [I-D.hardjono-oauth-kerberos] for a comparison between Kerberos and OAuth), the OAuth 2.0 MAC token [I-D.ietf-oauth-v2-http-mac], and the

Holder-of-the-Key approach [I-D.tschofenig-oauth-hotk].

To illustrate key confirmation the first examples borrow from Kerberos and use symmetric key cryptography. Assume that the Authorization Server shares a long-term secret with the Resource Server, called K(Authorization Server-Resource Server). This secret would be established between them in an initial registration phase. When the Client requests an access token the Authorization Server creates a fresh and unique session key Ks and places it into the token encrypted with the long term key K(Authorization Server-Resource Server). Additionally, the Authorization Server attaches Ks to the response message to the Client (in addition to the access token itself) over a confidentiality protected channel. When the Client sends a request to the Resource Server it has to use Ks to compute a keyed message digest for the request (in whatever form or whatever layer). The Resource Server, when receiving the message, retrieves the access token, verifies it and extracts K(Authorization Server-Resource Server) to obtain Ks. This key Ks is then used to verify the keyed message digest of the request message.

Note that in this example one could imagine that the mechanism to protect the token itself is based on a symmetric key based mechanism to avoid any form of public key infrastructure but this aspect is not further elaborated in the scenario.

A similar mechanism can also be designed using asymmetric cryptography. When the Client requests an access token the Authorization Server creates an ephemeral public / privacy key pair (PK/SK) and places the public key PK into the protected token. When the Authorization Server returns the access token to the Client it also provides the PK/SK key pair over a confidentiality protected channel. When the Client sends a request to the Resource Server it has to use the privacy key SK to sign the request. The Resource Server, when receiving the message, retrieves the access token, verifies it and extracts the public key PK. It uses this ephemeral public key to verify the attached signature.

4.4. Summary

As a high level message, there are various ways how the threats can be mitigated and while the details of each solution is somewhat different they all ultimately accomplish the goal.

The three approaches are:

Confidentiality Protection:

The weak point with this approach, which is briefly described in Section 4.1, is that the Client has to be careful to whom it discloses the access token. What can be done with the token entirely depends on what rights the token entitles the presenter and what constraints it contains. A token could encode the identifier of the Client but there are scenarios where the Client is not authenticated to the Resource Server or where the identifier of the Client rather represents an application class rather than a single application instance. As such, it is possible that certain deployments choose a rather liberal approach to security and that everyone who is in possession of the access token is granted access to the data.

Sender Constraint:

The weak point with this approach, which is briefly described in Section 4.2, is to setup the authentication infrastructure such that Clients can be authenticated towards Resource Servers. Additionally, Authorization Server must encode the identifier of the Client in the token for later verification by the Resource Server. Depending on the chosen layer for providing Client-side authentication there may be additional challenges due Web server load balancing, lack of API access to identity information, etc.

Key Confirmation:

The weak point with this approach, see Section 4.3, is the increased complexity: a complete key distribution protocol has to be defined.

In all cases above it has to be ensured that the Client is able to keep the credentials secret.

5. Requirements

In an attempt to address the threats described in Section 3 the Bearer Token, which corresponds to the description in Section 4.1, was standardized and the work on a JSON-based token format has been started [I-D.ietf-oauth-json-web-token]. The required capability to protected the content of a JSON token using integrity and confidentiality mechanisms is currently work in progress in the IETF JOSE working group.

Consequently, the purpose of the remaining document is to provide security that goes beyond the Bearer Token offered security protection.

Luckily this is not the first security protocol that has been designed. In trying to seek guidance the authors found RFC 4962 [RFC4962], which gives useful guidelines for designers of authentication and key management protocols. While RFC 4962 was written with the AAA framework used for network access authentication in mind the offered suggestions are useful for the design of other key management systems as well. The following requirements list applies OAuth 2.0 terminology to the requirements outlined in RFC 4962.

These requirements include

Cryptographic Algorithm Independent:

The key management protocol MUST be cryptographic algorithm independent.

Strong, fresh session keys:

Session keys MUST be strong and fresh. Each session deserves an independent session key, i.e., one that is generated specifically for the intended use. In context of OAuth this means that keying material is created in such a way that can only be used by the combination of a Client instance, protected resource, and authorization scope.

Limit Key Scope:

Following the principle of least privilege, parties MUST NOT have access to keying material that is not needed to perform their role. Any protocol that is used to establish session keys MUST specify the scope for session keys, clearly identifying the parties to whom the session key is available.

Replay Detection Mechanism:

The key management protocol exchanges MUST be replay protected. Replay protection allows a protocol message recipient to discard any message that was recorded during a previous legitimate dialogue and presented as though it belonged to the current dialogue.

Authenticate All Parties:

Each party in the key management protocol MUST be authenticated to the other parties with whom they communicate. Authentication mechanisms MUST maintain the confidentiality of any secret values used in the authentication process. Secrets MUST NOT be sent to another party without confidentiality protection.

Authorization:

Client and Resource Server authorization MUST be performed. These entities MUST demonstrate possession of the appropriate keying material, without disclosing it. Authorization is REQUIRED whenever a Client interacts with an Authorization Server. The authorization checking prevents an elevation of privilege attack, and it ensures that an unauthorized authorized is detected.

Keying Material Confidentiality and Integrity:

While preserving algorithm independence, confidentiality and integrity of all keying material MUST be maintained.

Confirm Cryptographic Algorithm Selection:

The selection of the "best" cryptographic algorithms SHOULD be securely confirmed. The mechanism SHOULD detect attempted roll-back attacks.

Uniquely Named Keys:

Key management proposals require a robust key naming scheme, particularly where key caching is supported. The key name provides a way to refer to a key in a protocol so that it is clear to all parties which key is being referenced. Objects that cannot be named cannot be managed. All keys MUST be uniquely named, and the key name MUST NOT directly or indirectly disclose the keying material.

Prevent the Domino Effect:

Compromise of a single Client MUST NOT compromise keying material held by any other Client within the system, including session keys and long-term keys. Likewise, compromise of a single Resource Server MUST NOT compromise keying material held by any other Resource Server within the system. In the context of a key hierarchy, this means that the compromise of one node in the key hierarchy must not disclose the information necessary to compromise other branches in the key hierarchy. Obviously, the compromise of the root of the key hierarchy will compromise all of the keys; however, a compromise in one branch MUST NOT result in the compromise of other branches. There are many implications of this requirement; however, two implications deserve highlighting. First, the scope of the keying material must be defined and understood by all parties that communicate with a party that holds that keying material. Second, a party that holds keying material in a key hierarchy must not share that keying material with parties that are associated with other branches in the key hierarchy.

Bind Key to its Context:

Keying material MUST be bound to the appropriate context. The context includes the following.

* The manner in which the keying material is expected to be used.

* The other parties that are expected to have access to the keying material.

* The expected lifetime of the keying material. Lifetime of a child key SHOULD NOT be greater than the lifetime of its parent in the key hierarchy.

Any party with legitimate access to keying material can determine its context. In addition, the protocol MUST ensure that all parties with legitimate access to keying material have the same context for the keying material. This requires that the parties are properly identified and authenticated, so that all of the parties that have access to the keying material can be determined. The context will include the Client and the Resource Server identities in more than one form.

Authorization Restriction:

If Client authorization is restricted, then the Client SHOULD be made aware of the restriction.

Client Identity Confidentiality:

A Client has identity confidentiality when any party other than the Resource Server and the Authorization Server cannot sufficiently identify the Client within the anonymity set. In comparison to anonymity and pseudonymity, identity confidentiality is concerned with eavesdroppers and intermediaries. A key management protocol SHOULD provide this property.

Resource Owner Identity Confidentiality:

Resource servers SHOULD be prevented from knowing the real or pseudonymous identity of the Resource Owner, since the Authorization Server is the only entity involved in verifying the Resource Owner's identity.

Collusion:

Resource Servers that collude can be prevented from using information related to the Resource Owner to track the individual. That is, two different Resource Servers can be prevented from determining that the same Resource Owner has authenticated to both of them. This requires that each Authorization Server obtains different keying material as well as different access tokens with content that does not allow identification of the Resource Owner.

AS-to-RS Relationship Anonymity:

The Authorization Server can be prevented from knowing which Resource Servers a Resource Owner interacts with. This requires avoiding direct communication between the Authorization Server and the Resource Server at the time when access to a protected resource by the Client is made. Additionally, the Client must not provide information about the Resource Server in the access token request. [QUESTION: Is this a desirable property given that it has other implications for security?]

As an additional requirement a solution MUST enable support for channel bindings. The concept of channel binding, as defined in [RFC5056], allows applications to establish that the two end-points of a secure channel at one network layer are the same as at a higher layer by binding authentication at the higher layer to the channel at the lower layer.

Furthermore, there are performance concerns specifically with the usage of asymmetric cryptography. As such, the requirement can be phrases as 'faster is better'. [QUESTION: How are we trading the benefits of asymmetric cryptography against the performance impact?]

Finally, there are threats that relate to the experience of the software developer as well as operational policies. For example, a frequently raised concern is the absent of verifying that the server's presented identity matches its reference identity so it can authenticate the communication endpoint and authorize it. Verifying the server identity in TLS is discussed at length in [RFC6125]. There are also various guesses about what application developers are able to implement correctly and easily and to what degree they can rely on third party libraries.[QUESTION: How do we reflect these requirements in the design?]

6. Security Considerations

The main focus of this document is on security.

7. Next Steps

From this description so far a few observations and next steps can be derived:

1. Bearer Tokens are a viable solution for protecting against the threats described in Section 3. Further standardization work on OAuth security mechanisms needs to provide additional security benefits on top of those provided by the bearer token solution.

2. The requirements listed in Section 5 aim to provide a starting point for a discussion on a security solution that provides additional security and privacy benefits for OAuth 2.0.

3. It is likely that implementers will find security solutions hard to implement and hard to configure right. Additional guidance and the availability to libraries may help to improve security on the Internet for OAuth-based implementations. Fundamentally, there is the question about a design that is based on symmetric vs. asymmetric cryptography. Ideally, only a single solution should be developed (or a very small number) since the differences between different variations of such as protocol are minor.

4. A standardized solution for the token format is needed to mitigate a number of attacks and this work is already ongoing under the name of JWT [I-D.ietf-oauth-json-web-token].

To make progress with the above-mentioned items before the next IETF meeting in Atlanta I therefore suggest to (a) solicit for document reviews regarding the JWT document, and (b) progress the work on the extended OAuth security mechanism. Regarding the latter aspect consider the following questions:

Threats:

Section 3 lists a few security threats. Are these the threats you care about? Which threats missing?

Requirements:

The working group has expressed interest to work on an extended OAuth security mechanism. Assuming that the group wants to develop a key distribution protocol (as described in Section 4.3) are the requirements listed in Section 5 complete? Who is interested to develop early prototypes of support the standards development?

8. IANA Considerations

This document does not require actions by IANA.

9. Acknowledgments

The authors would like to thank the OAuth working group for their discussion input. A group of regular OAuth participants met at the IETF #82 meeting in Vancouver to discuss this topic in preparation for the face-to-face meeting. The participants were:

- John Bradley
- Brian Campbell
- Phil Hunt
- Leif Johansson
- Mike Jones
- Lucy Lynch
- Tony Nadalin
- Klaas Wierenga

This document reuses content from [RFC4962] and the author would like thank Russ Housely and Bernard Aboba for their work on that document.

Finally, I would like to thank Blaine Cook. This document was derived from an earlier draft that Blaine and I wrote.

10. References

10.1. Normative References

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

[I-D.ietf-oauth-v2]
Hardt, D., "The OAuth 2.0 Authorization Framework", draft-ietf-oauth-v2-31 (work in progress), August 2012.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[I-D.ietf-oauth-v2-bearer]
Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", draft-ietf-oauth-v2-bearer-23 (work in progress), August 2012.

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-03 (work in progress), July 2012.

10.2. Informative References

[RFC4962] Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, July 2007.

[I-D.iab-privacy-considerations]
Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", draft-iab-privacy-considerations-03 (work in progress), July 2012.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[I-D.hardjono-oauth-kerberos]
Hardjono, T., "OAuth 2.0 support for the Kerberos V5 Authentication Protocol", draft-hardjono-oauth-kerberos-01 (work in progress), December 2010.

[RFC5849] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

[I-D.ietf-oauth-v2-http-mac]
Hammer-Lahav, E., "HTTP Authentication: MAC Access Authentication", draft-ietf-oauth-v2-http-mac-01 (work in progress), February 2012.

[I-D.tschofenig-oauth-hotk]
Bradley, J., Hunt, P., Nadalin, A., and H. Tschofenig, "The OAuth 2.0 Authorization Framework: Holder-of-the-Key Token Usage", draft-tschofenig-oauth-hotk-01 (work in progress), July 2012.

[NIST800-63]
Burr, W., Dodson, D., Perlner, R., Polk, T., Gupta, S., and E. Nabbus, "NIST Special Publication 800-63-1, INFORMATION SECURITY", December 2008.

Authors' Addresses

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Phil Hunt
Oracle Corporation

Email: phil.hunt@yahoo.com