Open Authentication Protocol	T. Lodderstedt, Ed.
Internet-Draft	Deutsche Telekom AG
Intended status: Standards Track	M. McGloin
Expires: October 09, 2011	IBM
	P. Hunt
	Oracle Corporation
	A. Nadalin
	Microsoft Corporation
	April 07, 2011

OAuth 2.0 Security Considerations
draft-lodderstedt-oauth-securityconsiderations-02

Abstract

This document gives security considerations for the OAuth 2.0 protocol. The proposed text is intended to be included into [I-D.ietf-oauth-v2].

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]*.

Status of this Memo

Copyright Notice

described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Definitions

This document considers the following clients categories:

Web Application Such an application is installed on a server. End-users access it via a HTML user interface rendered in the user agent on the end-user's device. All application data relevant to the OAuth protocol are stored on the server and is not accessible by the user.

User Agent-based Application Such an application is downloaded from a web site and runs within the user agent on the end-user's device. All application data relevant to the OAuth protocol is accessible by the user. Since such applications directly reside within the user agent, they can seamlessly make use of it capabilities in the end-user authorization process.

Native Application Such an app is installed and runs on an end-user's device. All application data relevant to the OAuth protocol is accessible by the user. It is assumed that such applications can protect dynamically issued secrets, such as refresh tokens, from eavesdropping by other applications residing on the same device.

2. Security Considerations

Note: This section focuses on the security guidelines implementors of the protocol MUST consider. We encourage readers to consult the more detailed analysis with additional background information in [I-D.lodderstedt-oauth-security].

2.1. Client Authentication

Authorization servers MAY issue client secrets to web applications for the purpose of authenticating them. Authorization servers are encouraged to consider stronger client authentication means.
Application developers MUST ensure confidentiality of client secrets and other credentials.
Authorization server MUST NOT issue client secrets to native or user agent-based applications in general. An authorization server MAY issue a client secret for an installation of a native application on a specific device. Alternatively, authorization servers MUST utilize other means than client authentication to achieve their security objectives.

2.2. Malicious Client Obtains Authorization

A malicious client could impersonate a valid client and obtain access to a protected resource.
Assumption: It is not the task of the authorization server to protect the end-user's device from malicious software. This is the responsibility of the platform running on the particular device probably in cooperation with other components of the respective ecosystem (e.g. an application management infrastructure). The sole responsibility of the authorization server is to control access to the end-user's resources living in resource servers and to prevent unauthorized access to them. Based on this assumption, the following countermeasures are recommended.
Where the client can be authenticate, the authorization server MUST authenticate it. If the authorization server cannot authenticate the

particular impersonated client, the authorization server MUST utilize other means to achieve its security objectives. The authorization server MAY enforce explicit user authentication or ask the end-user for consent. In this context, the user SHALL be explained the purpose, scope, and duration of the authorization. The authorization server MUST make the meta-data it associates with the particular client (e.g. the name) available to the end-user. It is up to the user to validate the binding of this data to the particular application and to approve the authorization request.
Authorization servers MUST NOT automatically process (without user interaction) repeated authorizations without authenticating the client.
The authorization server SHOULD require clients to pre-register their redirect_uri's and validate the actual redirect_uri against the pre-registered value.
It is higly RECOMMENDED that the authorization server limits the scope of tokens.

2.3. Access Tokens

Access tokens MUST only be accessible to the authorization server, the resource servers this token is valid for and the client to whom they have been issued. The only exception is the implicit grant where the user agent gets access to the access token that is transmitted in the URI fragment.
Authorization server as well as application developers MUST ensure confidentiality of access tokens, on transit and in storage.
Application developers MUST NOT store access tokens in non-transient memory.
Authorization servers MUST ensure that access tokens cannot be manufactured, modified, or guessed.

2.4. Refresh Tokens

Authorization servers MAY issue refresh tokens to web and native applications.
Refresh tokens MUST only be accessible to the authorization server and the client to whom they have been issued. The authorization server MUST maintain the link between a refresh token and the client to whom it has been issued.
Where the client can be authenticated, this relation between client and refresh token MUST be validated on every token refreshment request. If this is not possible, it is RECOMMENDED for authorization servers to implement other means to detect abuse of refresh tokens.
Authorization server as well as application developers MUST ensure confidentiality of refresh tokens, on transit and in storage.
Authorization servers MUST ensure that refresh tokens cannot be manufactured, modified, or guessed.

2.5. Token Scope

It is strongly RECOMMENDED that application developers only acquire access tokens with the minimal scope they need in order to implement the respective application function.
When obtaining end user authorization and where the client requests scope, the authorization server MAY want to consider whether to honour that scope based on who/what the client is and the type of access grant the client asked for. The resource owner MAY also further restrict the scope of the access tokens.

2.6. Request Confidentiality

The following security sensitive data elements MUST NOT be transmitted in clear: access tokens, refresh tokens, resource owner passwords, authorization codes, and client secrets.

2.7. Endpoints Authenticity

In order to prevent men-in-the-middle and phishing attacks, HTTPS with server-side authentication MUST be implemented and used by authorization servers in all exchanges.
For the same purpose, HTTPS with server-side authentication SHOULD/MUST [note: this is still subject to a WG discussion] be implemented and used by web application clients at least on their redirect_uri.
Application developers MUST provide mechanisms to validate the authorization server endpoint's authenticity and ensure proper handling of CA certificates as well as certificate chain validation.

2.8. Online Guessing Attacks

Authorization servers MUST prevent guessing attacks on the following credentials: authorization codes, refresh tokens, resource owner passwords, and client secrets.
When creating token handles or other secrets not intended for usage by human users, the authorization server MUST include a reasonable level of entropy in order to mitigate the risk of guessing attacks. When creating secrets intended for usage by human users, the authorization server MUST utilize other means to protect those secrets.

2.9. Phishing Attacks

It is strongly RECOMMENDED that native application developers use external browsers instead of browsers embedded in the application for performing the end-user authorization process. External browsers offer a familiar usage experience and a trusted environment, in which users can confirm the authentictity of the site.
To reduce the risk of phishing attacks, authorization servers MUST support the authentication of their endpoint. For example, they can utilize HTTPS server authentication for that purpose. Moreover, service

providers should attempt to educate users about the risks phishing attacks pose, and should provide mechanisms that make it easy for users to confirm the authenticity of their sites. e.g. extended validation certificates.

2.10. Authorization Code Disclosure

Confidentiality of authorization codes MUST be ensured on transport.
Note: Since the code is transmitted via browser redirects, it could also be disclosed through browser histories and HTTP referers.
The authorization server and the client MUST ensure that the authorization code transmission is protected by using channel security, such as TLS, and that the authorization code is short lived.
Where the client can be authenticated, the authorization servers MUST authenticate the client and validate that the authorization code had been issued to the same client. If the client cannot be authenticated, authorization servers MUST enforce one time usage of the authorization code. Moreover, if an authorization server observes multiple attempts to redeem an authorization code, the authorization server MAY want to revoke all tokens granted based on this authorization code.

2.11. Session Fixation

The session fixation attack leverages the authorization code flow in an attempt to get another user to log-in and authorize access on behalf of the attacker. The victim, seeing only a normal request from an expected application, approves the request. The attacker then uses the victim's authorization to gain access to the information unknowingly authorized by the victim.
In order to prevent such an attack, authorization servers MUST ensure that the redirect_uri used in the authorization flow is the same as the redirect_uri used to exchange the respective authorization code into tokens. The authorization server operators SHOULD require client application developers to pre-register their redirect_uri's and validate the actual redirect_uri against the pre-registered value.

2.12. Resource Owner Password Credentials

The “Resource Owner Password Credentials” grant type is often used for legacy/migration reasons. It reduces the overall risk of storing username and password in the client.
It has higher risk than the other OAuth grant types because it maintains the password anti-pattern. The client could abuse the password or the password could unintentionally be disclosed on the client site e.g. through log files. Additionally, because the user does not have control over the authorization process, clients could acquire tokens with much broader scope and longer livetime than desired by the user.

The authorization server MUST ensure the resource owner's control and transparency with respect to all authorizations issued to clients. Authorization servers and application developers SHOULD minimize use of this grant type. Other grant types which facilitate user control and transparency should be used instead.
The authorization server SHOULD generally restrict the scope of access tokens issued by this grant type.

3. Acknowledgements

4. References

4.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[I-D.ietf-oauth-v2]	Hammer-Lahav, E, Recordon, D and D Hardt, "The OAuth 2.0 Authorization Protocol", Internet-Draft draft-ietf-oauth-v2-22, September 2011.

4.2. Informative References

[I-D.lodderstedt-oauth-security]	Lodderstedt, T, McGloin, M and P Hunt, "OAuth 2.0 Threat Model and Security Considerations", Internet-Draft draft-lodderstedt-oauth-security-01, March 2011.

Authors' Addresses

Dr.-Ing. Torsten Lodderstedt editor Lodderstedt Deutsche Telekom AG
EMail: torsten@lodderstedt.net

Mark McGloin McGloin IBM EMail: mark.mcgloin@ie.ibm.com

Phil Hunt Hunt Oracle Corporation EMail: phil.hunt@yahoo.com

Anthony Nadalin Nadalin Microsoft Corporation EMail: tonynad@microsoft.com