OAuth working group	B. Leiba
Internet-Draft	Huawei Technologies
Intended status: Informational	March 28, 2011
Expires: September 29, 2011	

OAuth Additional Security Considerations
draft-leiba-oauth-additionalsecurityconsiderations-00

Abstract

The Open Authentication Protocol (OAuth) specifies a security protocol that involves significant end-user interaction -- the model is based on having the end-user approve the authorization that is being requested. That aspect makes the user interaction a part of the security model, and raises additional security considerations beyond those that are typical for client/server protocols. This document describes those considerations.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The Open Authentication Protocol [I-D.ietf-oauth-v2] specifies a security protocol used to authorize one service to act on behalf of a user to a second service (to give a somewhat oversimplified description). For example:

*A photo-printing service needs authorization to retrieve photos from a user's private photo albums, stored with a photo-sharing service.

*A social-network service needs authorization to read a user's personal address book, stored in the email service she uses.

In most use cases, part of the protocol involves prompting the user to log in to the second service and accept the authorization request. That aspect makes the user interaction a part of the security model, and raises additional security considerations beyond those that are typical for client/server protocols.

2. Laying Out the Issues

The OAuth version 2 spec [I-D.ietf-oauth-v2] describes, in its introduction, a typical example of the use of OAuth:

*For example, a web user (resource owner) can grant a printing service (client) access to her protected photos stored at a photo sharing service (resource server), without sharing her username and password with the printing service. Instead, she authenticates directly with a server trusted by the photo sharing service (authorization server) which issues the printing service delegation-specific credentials (access token).

In such a typical case:

1. The user (resource owner) visits the client's web site (the printing service) and makes a web request ("Print a photo from my photo-sharing site.")

2. The client, behind the scenes and not apparent to the user, requests authorization from the resource server, and is given a request token.

3. The client sends a response to the user's web request. The response redirects the user's web browser to the authorization server (of the photo-sharing service) and passes the server the request token.

4. The authorization server prompts the user to accept the requested authorization. The user sees this authorization

prompt in the browser, and it appears to be the first response to the original request.

5. The user accepts the authorization request. She might have to log in first, or an existing login might be used. Typically, she will click a visual "button" that says "Accept", "Authorize", "OK", or the like.

6. The authorization server sends a response to that action. The response redirects the user's web browser to the client web server (the printing service) and passes the server an access token.

7. The client (the printing service) uses the access token behind the scenes to contact the resource server (the photo-sharing service) and to perform the service for the user. The user is given a response that indicates that the request is in the works.

From the user's point of view, the interaction has seemed simple; this is a great benefit of OAuth. But there was actually a reasonable amount of complexity "behind the scenes" that the user did not understand, and that leaves us with a conflicted situation:

1. the user shouldn't have to understand all of the behind-the-scenes detail, but

2. the user needs to understand what she's being asked to authorize, in order that she might make the right security decision about the request.

2.1. On Granting Access

Some of the things a user might need to understand to make the decision include

*Who is requesting the access. This can be a tricky point. The authorization server might likely know only the domain name, or even just the IP address of the client that's making the request. The user *might* be able to make some sense of the domain name, but really would do better with a real, human-readable name that matches what she calls the service. And the authorization server has no cause to trust any human-readable string the requestor gives.

A DKIM-like digital signature [I-D.crocker-dkim-doseta] along with a mapping of known signing identities to readable names can be a great help here. But be aware that any name provided by the client is open to abuse by a bad actor, and should not be trusted.

*Who will be granting the access. It's easy to leave this out, with the idea that it should be self-evident. The problem is that if a client is requesting access beyond what it should be asking for, it might be asking the wrong entity as well. If a user asks for a photo to be printed, the requested authorization should not be to the user's email account.

*The specific access that is being requested. This might not always be obvious, depending upon the request, and depending upon how the prompt is worded. "Print a photo for me," will likely translate into read access to the photo. For "Auto-adjust the contrast before printing, and save the adjusted version," the service will need access to update the photo or to save a new copy.

"Send e-cards to my family on their birthdays," might translate into authority to send email on her behalf, plus read access to her address book. The address book access is somewhat less evident, and leaves a different avenue for abuse.

*The scope of the access. If she wants to print a single photo, then read access to just that photo will do. If she wants to print all photos in an album, she'll need to grant read access to the whole album.

*The duration of the access. If she just wants to print a photo, then a single read access to the photo should be enough. If she wants the service to automatically print all her new photos every week, persistent, long-term read access to her "new uploads" photo album might work.

Because end-users are accepting or rejecting the authorization that the client service is requesting, their understanding of what they're being asked is important to the overall security of the system. And because end-users often know nothing about computer security, the way these various points are presented to them is a critical piece of the security design. That is, the prompts and the user's understanding of them and response to them have to be considered part of the security model.

This is especially important because the user is thinking in terms of a task, while the authorization system is working in terms of what accesses are needed for the task. The mapping between the two is often not clear to the user, and the user's trust of the service requesting the access might be tenuous.

We need to avoid asking users questions they're not prepared or qualified to answer. Unfortunately, most security-related questions fall into that category. The more we can put the request into plain language, and the better we can explain, in clear, simple terms what's being asked and what the ramifications of it are, the more likely it is that we'll be working with informed consent and will have a chance at fending off attacks on the system.

Consider the difference, for example, between the following two prompts.

```
    ------------------------------------------------------------------
    Give printpix.example r/o access to
    http://photoshare.example/usr213554/fnxgrptl/250/43/342500134.jpg

         (OK)      (Cancel)
    ------------------------------------------------------------------
```

Example 1a

```
-----------------------------------------------------------------
The Print My Pix service (printpix.example) is asking PhotoShare
for access to your photo titled "Ralph in the park" in album
"New York". Granting access will allow Print My Pix to read, but
not alter nor delete, your photo.  Access will be allowed one
time only.

For a more detailed explanation of what this means, [click here].

Do you want to allow access?

     (Yes)      (No)
-----------------------------------------------------------------
```

Example 1b

The second prompt is wordier, but might be easier for most people to understand. Some of the technical details (such as the URL to the photo in question) are available at a click, for users who want more details. Note how that difference appears to a user when the Print My Pix service is actually abusive, and tries to get broader access than it needs.

```
-----------------------------------------------------------------
Give printpix.example r/w access to
http://photoshare.example/usr213554/

     (OK)      (Cancel)
-----------------------------------------------------------------
```

Example 2a

```
-----------------------------------------------------------------
The Print My Pix service (printpix.example) is asking PhotoShare
for access to all your photo albums. Granting access will allow
Print My Pix to read, alter and delete, your photos.  Access will
be allowed permanently.

For a more detailed explanation of what this means, [click here].

Do you want to allow access?

     (Yes)      (No)
-----------------------------------------------------------------
```

Example 2b

The second prompt makes it clearer that Print My Pix is asking for a lot. Specific warnings might also be added for atypical access requests, or ones that seem to be overstepping.
It is not the intent, here, to give specific user-interface design advice, and the design and wording of these prompts and other user-interface elements will not necessarily come out well if given to some protocol designers. The point, rather, is to highlight the issues and raise awareness of the security implications of how OAuth requests are communicated, so interface designers can take that into account.

2.2. On Discovering and Revoking Access

As we use a great many services that each act as clients to other services we use, we soon get into a very complex and hard-to-manage situation. Suppose our hypothetical user allows Facebook to access Flickr and to import contacts from Gmail; she allows her Blogger account to post photos to Flickr and to read the photos there; she allows Twitter to post her tweets to Facebook....

By the nature of these arrangements, and by the design of OAuth, there's no central place that keeps track of all the authorizations. Each service knows what accesses she has authorized to it, but she has to check each service individually. It's critical that she have an easy way to do that, that she can easily find the way to do that, and that she can understand the results that her queries return -- it won't do to have them in computer-geek gibberish.

Further, it has to be easy for her to suspend or revoke the authorizations she's made -- stale authorizations are entry points for security vulnerabilities. She needs to be able to correct errors she has made in authorizing things, as well as to rescind the authorizations that are no longer appropriate.

3. Recommendations

1. Implementations should consider clarity to end users as a critical part of the security model. Because users who are not experts in security are being asked to make security decisions, it is very important that the questions be clear, and that it be easy to find more information.

2. Implementations should explain exactly what is being requested by whom, and should provide one-click access to a more detailed explanation.

3. Implementations should explain the ramifications of accepting the request. Because the user is thinking of a specific task, other aspects of what the requested access permits are not in the user's mind. Put them there. Explain that this means that they will be able to put all your photos into their photo pool, send email on your behalf forever, and put everyone you know into their marketing database.

4. Implementations should be attuned to the more usual requests, and should highlight unusual requests when they arrive. If most reasonable requests need read access and one asks for read/write, tell the user it's unusual. Similarly, if most requests need one-time access, or access to a single resource, let the user know that a request for permanent access to multiple resources is cause for concern. This particular task might need it, but make sure the user is aware that this isn't what you usually see.

5. This should be a tractable problem, because most authorization servers will be dealing with a limited set of resources, and, hence, a limited set of typical authorization requests. The set of typical requests will often be readily enumerable, and out-of-the-ordinary ones will usually stand out.

6. Implementations should provide an easy way, obvious to the user, to inquire about active authorizations. This should not be hidden behind a sort of "advanced" page, but should be within easy reach of every user, along with a clear explanation of what it means.

7. Implementations should provide an easy way, once the active authorizations are shown, to get a detailed explanation of the authorization -- what it means, what it allows access to, what the ramifications are. Anything that should have been part of the original access prompt should be available here.

8. Implementations should provide an easy way, once the active authorizations are shown, to suspend or revoke each authorization immediately.

9. Client implementations should gracefully re-authorize in the event of a revoked authorization. Failure because an expected authorization has been revoked is considered poor quality of implementation. It is only appropriate to fail after an attempt to re-authorize is denied.

4. Security Considerations

This entire document is about security considerations. Can we have the RFC Editor remove this section prior to publication?

5. IANA Considerations

There are no IANA actions needed for this document, and the RFC Editor may remove this section prior to publication.

6. Acknowledgements

[Thanks will go here.]

7. References

[I-D.ietf-oauth-v2]	Hammer-Lahav, E, Recordon, D and D Hardt, "The OAuth 2.0 Authorization Protocol", Internet-Draft draft-ietf-oauth-v2-13, February 2011.
[I-D.crocker-dkim-doseta]	Crocker, D and M Kucherawy, "DomainKeys Security Tagging (DOSETA)", Internet-Draft draft-crocker-dkim-doseta-00, January 2011.

Author's Address

Barry Leiba Leiba Huawei Technologies Phone: +1 646 827 0648 EMail: barryleiba@computer.org URI: http://internetmessagingtechnology.org/