Does the IdP Mix-Up attack really work?

Wanpeng Li and Chris J. Mitchell
Information Security Group,
Royal Holloway, University of London
TW20 0EX

June 3, 2016

Abstract

This document briefly describes and analyses the IdP Mix-Up attack on OAuth 2.0, as described by Fett, Küsters and Schmitz. We suggest that a key assumption on which the attack depends is incorrect, and as a result the attack does not appear to be a genuine threat to the security of OAuth 2.0.

1 Introduction

The IETF OAuth 2.0 Working Group has recently released a mitigation[1] for the IdP Mix-Up attack, first described by Fett, Küsters and Schmitz, [1]. Given the serious potential impact of the attack, it is clearly vitally important to understand in precisely which circumstances the attack could be performed. This is the main focus of this brief report.

2 The IdP Mix-UP attack

2.1 Operation of the attack

The IdP Mix-Up attack applies to the *Implicit Mode* flow of OAuth 2.0 [2]. The original description of the attack, taken from section 3.2 of Fett, Küsters and Schmitz, [1], is given in Fig. 2.1.

In this description, HIdP refers to the 'honest' IdP, i.e. the one which the user chooses to use, and AIdP refers to the 'attacker' IdP, i.e. an authorised but dishonest IdP.

2.2 RP behaviour after redirection

In the underlined text in Fig. 2.1, it is asserted that when the HIdP redirects the user back to the RP, the RP then assumes that the authorization

[1] `http://self-issued.info/?p=1524`

Attack. We now describe the attack on the OAuth implicit mode in detail. As mentioned, a very similar attack also applies to the OAuth authorization code mode and both attacks even work if IdP supports just one of these two modes, rather than both or all four OAuth modes. Note that these two modes are the most common modes in practice.

Attack on Implicit Mode. The IdP mix-up attack for the implicit mode is depicted in Figure 6. Just as in the implicit mode, the attack starts when the user selects that she wants to log in using HIdP (Step 1 in Figure 6). Now, the attacker intercepts the request intended for the RP and modifies the content of this request by replacing HIdP by AIdP. The response of the RP 3 (containing a redirect to AIdP) is then again intercepted and modified by the attacker such that it redirects the user to HIdP 4. The attacker also replaces the OAuth client id of the RP at AIdP with the client id of the RP at HIdP.[12] (Note that we assume that from this point on, in accordance with the OAuth security recommendations, the communication between the user's browser and HIdP and the RP is encrypted by using HTTPS, and thus, cannot be inspected or altered by the attacker.) The user then authenticates to HIdP and is redirected back to the RP 8. The RP, however, still assumes that the access token contained in this redirect is an access token issued by AIdP, rather than HIdP. The RP therefore now uses this access token to retrieve protected resources of the user (or the user id) at AIdP 12, rather than HIdP. This leaks the access token to the attacker who can now access protected resources of the user at IdP. This breaks the authorization property (see Section 5.2 below). (We note that at this point, the attacker might even provide false information about the user or her protected resources to the RP.)

Figure 2.1: IdP Mix-Up attack on Implicit Mode

response is coming from the AIdP and not the HIdP; as a result the RP subsequently submits the *access_token* to the AIdP. This is a key assumption underlying the IdP Mix-Up attack.

However, we believe that this assumption is incorrect. As specified in the OAuth 2.0 Authorization Framework, [2], the *redirect_uri* defines where the authorization response should be redirected. The *redirect_uri* defines the RP's behaviour after the redirection, in contrast to the implicit but we believe incorrect assumption in the attack description that the RP's behaviour is bound to the first request. In the IdP Mix-Up attack scenario, the UA will actually be redirected to the *redirect_uri* the RP registered with the HIdP, as the user is authenticated by the HIdP, and not the *redirect_uri* the RP registered with the AIdP; hence the RP will submit the access_token to the HIdP not the AIdP. Thus we believe that the IdP Mix-UP attack will not work, at least as described in Fig. 2.1.

One might argue that the RP might register the same *redirect_uri* with the AIdP and the HIdP. However, as HTTP is a stateless protocol, the RP must implement logic to distinguish authorization responses originating from multiple IdPs. That is, the RP must be able to distinguish authorization responses received from different IdPs in order to be able to submit the access_token to the correct IdP. Otherwise, the RP will not be able to determine where the authorization response comes from.

Even if the *redirect_uri* is the same for the two IdPs, as discussed above, the RP is able to distinguish which IdP has generated the authorization response, and will submit the access_token to the correct IdP to retrieve the user's information. As a result, the IdP Mix-Up attack will not work.

In a small experiment, we checked whether or not RPs register different *redirect_uri*s with different IdPs. In all the cases we examined we were unable to find an RP that registered the same redirect_uri for different IdPs.

2.3 A simple example of the IdP Mix-UP attack

To help explain our concern with the attack, we now describe its operation in a simple scenario; in this example we use BBC as the RP, Facebook as the HIdP, and EvilCo as the AIdP (where we suppose that EvilCo is a legitimate but ill-intentioned IdP). The attack operates as follows.

1. Suppose the user wishes to use Facebook to log in to the BBC web site. The user clicks the Facebook login button. This will generate a request to the BBC which indicates that the user wants to use Facebook to login. The attacker intercepts the request and modifies it to make the BBC believe that the user wants to use EvilCo to sign in to the BBC.

2. The BBC generates the authorization request for EvilCo, and the attacker intercepts it and changes the *client_id* and *redirect_uri* to the values the BBC registered with Facebook. This will make the authorization request look as if it was intended for Facebook.

3. The user authenticates to Facebook and clicks the authorization button. An authorization response is generated by Facebook and sent to the BBC.

4. The authors assume that at this point the BBC will process this response as if it was generated by EvilCo; as a result it will send the received *access_token* to EvilCo to retrieve the required user information. As a result, the attacker (EvilCo) will discover the user's *access_token* for Facebook, and can then use this *access_token* to retrieve sensitive user information from Facebook.

However, Step 4 will not proceed as the above description suggests, since the authorization response generated by Facebook in Step 3 will be redirected to the *redirect_uri* that the BBC registered with Facebook, not the *redirect_uri* that the BBC registered with EvilCo. Thus the BBC will know where the authorization response comes from and will only submit the *access_token* to Facebook.

3 Conclusion

Our main conclusion is that the IdP Mix-Up attack cannot work on RPs that support two or more IdPs. As a result, it is far from clear whether any mitigations need to be added to the OAuth specifications.

Interestingly, the issue raised by the workability of this attack sheds light on the interactions between formal models of security protocols and the protocols themselves. The attack we have discussed above was discovered by Fett, Küsters and Schmitz, [1], through their formal modelling of the OAuth protocol. It appears that the attack *does* work in their model of the protocol, whereas we believe the attack will not work because certain assumptions underlying the model do not apply in practice. That is, because the model does not perfectly correspond to the real-life protocol, as is very often the case for such mathematical models, then an attack that is valid in the model is not a genuine threat in practice.

References

[1] Daniel Fett, Ralf Küsters, and Guido Schmitz. A Comprehensive Formal Security Analysis of OAuth 2.0. Technical Report arXiv:1601.01229, arXiv, 2016. Available at `http://arxiv.org/abs/1601.01229`.

[2] D. Hardt (editor). *RFC 6749: The OAuth 2.0 Authorization Framework*. Internet Engineering Task Force (IETF), October 2012.