参考文献原文

A Comprehensive Formal Security Analysis of OAuth 2.0

Daniel Fett
University of Trier, Germany
fett@uni-trier.de
Ralf Küsters
University of Trier, Germany
kuesters@uni-trier.de
Guido Schmitz
University of Trier, Germany
schmitzg@uni-trier.de

ABSTRACT

The OAuth 2.0 protocol is one of the most widely deployed authorization/single sign-on (SSO) protocols and also serves as the foundation for the new SSO standard OpenID Connect. Despite the popularity of OAuth, so far analysis efforts were mostly targeted at finding bugs in specific implementations and were based on formal models which abstract from many web features or did not provide a formal treatment at all. In this paper, we carry out the first extensive formal analysis of the OAuth 2.0 standard in an expressive web model. Our analysis aims at establishing strong authorization, authentication, and session integrity guarantees, for which we provide formal definitions. In our formal analysis, all four OAuth grant types (authorization code grant, implicit grant, resource owner password credentials grant, and the client credentials grant) are covered. They may even run simultaneously in the same and different relying parties and identity providers, where malicious relying parties, identity providers, and browsers are considered as well. Our modeling and analysis of the OAuth 2.0 standard assumes that security recommendations and best practices are followed in order to avoid obvious and known attacks. When proving the security of OAuth in our model, we discovered four attacks which break the security of OAuth. The vulnerabilities can be exploited in practice and are present also in OpenID Connect. We propose fixes for the identified vulnerabilities, and then, for the first time, actually prove the security of OAuth in an expressive web model. In particular, we show that the fixed version of OAuth(with security recommendations and best practices in place) provides the authorization, authentication, and session integrity properties we specify.

1. INTRODUCTION

The OAuth 2.0 authorization framework [20] defines a web-based protocol that allows a user to grant web sites access to her resources(data or services) at other web sites (authorization). The former web sites are called relying parties (RP) and the latter are called identity providers (IdP).1 In practice, OAuth 2.0 is often used for authentication as well. That is, a user can log in at an RP using her identity managed by an IdP (single sign-on, SSO).Authorization and SSO solutions have found widespread adoption in the web over the last years, with OAuth 2.0 being one of the most popular frameworks. OAuth 2.0, in the following often simply called OAuth,2 is used by identity providers such as Amazon, Facebook, Google, Microsoft, Yahoo, GitHub, LinkedIn, Stack Exchange, and Dropbox. This enables billions of users to log in at millions of RPs or share their data with these [35], making Outdone of the most used single sign-on systems on the webfoot is also the foundation for the new single sign-on protocol OpenID Connect, which is already in use and actively supported by PayPal (“Log In with PayPal”), Google, and Microsoft, among others. Considering the broad industry support for OpenID Connect, a widespread adoption of OpenID Connect in the next years seems likely. OpenID Connect builds upon OAuth and provides clearly defined interfaces for user authentication and additional (optional)features, such as dynamic identity provider discovery and relying party registration, signing and encryption of messages, and logout. In OAuth, the interactions between the user and her browser, the RP, and the IPDU can be performed in four different flows, or grant types: authorization code grant, implicit grant, resource owner password credentials grant, and the client credentials grant (we refer to these as modes in the following). In addition, all of these modes provide further options. The goal of this work is to provide an in-depth security analysis of OAuth. Analyzing the security of OAuth is a challenging task, on the one hand due to the various modes and options that Autoploids, and on the other hand due to the inherent complexity of the web. So far, most analysis efforts regarding the security of OAuth were targeted towards finding errors in specific implementations [6, 10,25, 33, 34, 36, 38], rather than the comprehensive analysis of the standard itself. Probably the most detailed formal analysis carried out on OAuth so far is the one in [6]. However, none of the existing analysis efforts of OAuth account for all modes of OAuth running simultaneously, which may potentially introduce new security risks. In fact, many existing approaches analyze only the authorization code mode and the implicit mode of OAuth. Also, importantly, there are no analysis efforts that

are based on a comprehensive formal web model (see below), which, however, is essential to rule1Following the OAuth 2.0 terminology, Dip's are called authorization servers and resource servers, RPs are called clients, and users are called resource owners. Here, however, we stick to the more common terms mentioned above.2Note that in this document, we consider only OAuth 2.0, which is very different to its predecessor, OAuth 1.0(a).out security risks that arise when running the protocol in the context of common web technologies (see Section 6 for a more detailed discussion of related work).

Contributions of this Paper. We perform the first extensive formal analysis of the OAuth 2.0 standard for all four modes, which can even run simultaneously within the same and different RPs and Disabused on a comprehensive web model which covers large parts of how browsers and servers interact in real-world setups. Our analysis also covers the case of malicious Dip's, RPs, and browsers/users. Formal model of OAuth. Our formal analysis of OAuth uses inexpressive Dole-Yao style model of the web infrastructure proposed by Fett, Kusters, and Schmitz (FKS). The FKS model has already been used to analyze the security of the Browse rid single sign-on system [14, 15] as well as the security and privacy of espresso single sign-on system [16]. This web model is designed independently of a specific web application and closely mimics published (de-facto) standards and specifications for the web, for instance, the HTTP/1.1 and HTML5 standards and associated (proposed) standards. It is the most comprehensive web model to date. Among others, HTTP(S) requests and responses, including several headers, such as cookie, location, strict transport security (STS),and origin headers, are modeled. The model of web browsers captures the concepts of windows, documents, and iframes, including the complex navigation rules, as well as new technologies, such as web storage and web messaging (via post Message). JavaScript is modeled in an abstract way by so-called scripts which can be sent around and, among others, can create frames and initiate XML HTTP Requests (XHRs). Browsers may be corrupted dynamically by the adversary. Using the generic FKS model, we build a formal model of OAuth, closely following the OAuth 2.0 standard (RFC6749 [20]). Since this RFC does not fix all aspects of the protocol and in order to avoid known implementation attacks, we use the OAuth 2.0 security recommendations (RFC6819 [26]), additional RFCs and Outworking Group drafts (e.g., RFC7662 [30], [8]) and current web best practices (e.g., regarding session handling) to obtain a model of OAuth with state-of-the-art security features in place, while making as few

assumptions as possible. Moreover, as mentioned above, our model includes RPs and Dip's that (simultaneously) support all four modes and can be dynamically corrupted by the adversary. Also, we model all configuration options of OAuth (see Section 2).Formalization of security properties. Based on this model of OAuth, we provide three central security properties of OAuth: authorization, authentication, and session integrity, where session integrity in turnips concerned with both authorization and authentication. Attacks on OAuth 2.0 and fixes. While trying to prove these properties, we discovered four attacks on OAuth. In the first attack, which breaks the authorization and authentication properties, Dip's inadvertently forward user credentials (i.e., username and password) tithe RP or the attacker. In the second attack (IPDU mix-up), a network attacker playing the role of an IPDU can impersonate any victim. This severe attack, which again breaks the authorization and authentication properties, is caused by a logical flaw in the OAuth 2.0 protocol. Two further attacks allow an attacker to force a browser to be logged in under the attacker's name at an RP or force an RP to use a resource of the attacker instead of a resource of the user, breaking the session integrity property. We have verified all four attacks on actual implementations of OAuth and OpenID Connect. We present our attacks on OAuth in detail in Section 3. In our technical report [17],we show how the attacks can be exploited in OpenID Connect. Wealso show how the attacks can be fixed by changes that are easy to implement in new and existing deployments of OAuth and OpenID Connect. We notified the respective working groups, who confirmed the attacks and that changes to the standards/recommendations are needed. The IdP mix-up attack already resulted in a draft of a new RFC [22].Formal analysis of OAuth 2.0. Using our model of OAuth with the fixes in place, we then were able to prove that OAuth satisfies the mentioned security properties. This is the first proof which establishes central security properties of OAuth in a comprehensive and expressive web model (see also Section 6).We emphasize that, as mentioned before, we model OAuth with security recommendations and best practices in place. As discussed in Section 5, implementations not following these recommendations and best practices may be vulnerable to attacks. In fact, many such attacks on specific implementations have been pointed out in the literature (e.g., [6, 10, 20, 25, 26, 36, 37]). Hence, our results also provide guidelines for secure OAuth implementations. We moreover note that, while these results provide strong security guarantees for OAuth, they do not directly imply security of OpenID Connect because OpenID Connect adds specific details on top of OAuth. We

leave a formal analysis of OpenID Connect to future work. The results obtained here can serve as a good foundation for such an analysis.

Structure of this Paper. In Section 2, we provide a detailed description of OAuth 2.0 using the authorization code mode as an example. In Section 3, we present the attacks that we found during our analysis. An overview of the FKS model we build upon in our analysis is provided in Section 4, with the formal analysis of OAuth presented in Section 5. Related work is discussed in Section 6. We conclude in Section 7. Full details, including how the attacks can be applied to OpenID Connect, further details on our model of OAuth, and our security proof, can be found in our technical report [17].

2. OAUTH 2.0

In this section, we provide a description of the OAuth authorization code mode, with the other three modes explained only briefly. In our technical report [17], we provide a detailed description of the remaining three modes (grant types).OAuth was first intended for authorization, i.e., users authorizers to access user data (called protected resources) at IdPs. For example, a user can use OAuth to authorize services such as IFTTT3to access her (private) timeline on Facebook. In this case, IFTTT is the RP and Facebook the IdP. Roughly speaking, in the most common modes, OAuth works as follows: If a user wants to authorize an RP to access some of the user's data at an IdP, the RP redirects the user (i.e., the user'sbrowser) to the IdP, where the user authenticates and agrees to grant the RP access to some of her user data at the IdP. Then, along with some token (an authorization code or an access token) issued by the IdP, the user is redirected back to the RP. The RP can then use the token as a credential at the IdP to access the user's data at the IdP. OAuth is also commonly used for authentication, although it was not designed with authentication in mind. A user can, for example, use her Facebook account, with Facebook being the IdP, to log in at the social network Pinterest (the RP). Typically, in order to log in,the user authorizes the RP to access a unique user identifier at the IdP. The RP then retrieves this identifier and considers this user to be logged in.

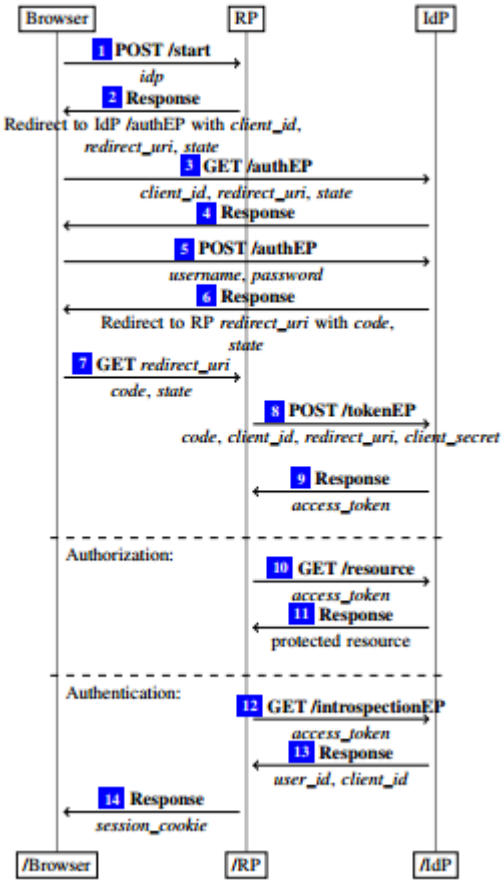

Figure 1: OAuth 2.0 authorization code mode. Note that data depicted below the arrows is either transferred in URI parameters, HTTP headers, or POST bodies.

Before an RP can interact with an IdP, the RP needs to be registered at the IdP. The details of the registration process are out of thescope of the OAuth protocol. In practice, this process is usually am anual task. During the registration process, the IdP assigns credentials to the RP: a public OAuth client id and (optionally) a clientsecret. (Recall that in the terminology of the OAuth standard the term “client” stands for RP.) The RP may later use the client secret(if issued) to authenticate to the IdP. Also, an RP registers one or more redirection endpoint URIs(located at the RP) at an IdP. As we will see below, in some OAuth modes, the IdP redirects the user’s browser to one of these URIs. Note that (depending on the implementation of an IdP) an RP may also register a pattern as a redirect URI and then specify the exact redirect URI during the OAuth run.In all modes, OAuth provides several options, such as those mentioned above. For brevity of presentation (and in contrast to our analysis), in the following descriptions, we consider only a

specific set of options. For example, we assume that an RP always provides redirect URI and shares an OAuth client secret with the IPDU.

Authorization Code Mode. When the user tries to authorize amp to access her data at an IPDU or to log in at an RP, the RP first redirects the user's browser to the IPDU. The user then authenticates to the IPDU, e.g., by providing her user name and password, and finally is redirected back to the RP along with an authorization code generated by the IPDU. The RP can now contact the IPDU with this authorization code (along with the client id and client secret)and receive an access token, which the RP in turn can use as a credential to access the user's protected resources at the Diptote-by-Step Protocol Flow. In what follows, we describe the protocol flow of the authorization code mode step-by-step (see also Figure 1). First, the user starts the OAuth flow, e.g., by clicking on a button to select an IPDU, resulting in request 1 being sent to Tharp. The RP selects one of its redirection endpoint URIs redirect Uri(which will be used later in 7) and a value state (which will serve as a token to prevent CSRF attacks). The RP then redirects the browser to the so-called authorization endpoint URI at the IPDU in 2 and 3with its clionitid, redirect Uri, and state appended as parameters tithe URI. The IPDU then prompts the user to provide her username and password in 4 . The user's browser sends this information tithe IPDU in 5 . If the credentials are correct, the IPDU creates a nonce code (the authorization code) and redirects the user's browser to RP's redirection endpoint URI redirect Uri in 6 and 7 with code and state appended as parameters to the URI. If state is the same as above, the RP contacts the IPDU in 8 and provides code, client_id, client_secret, and redirect Uri. Then the IdP checks whether this information is correct, i.e., it checks that code was issued for the RP identified by client_id, that client_secret is the secret for client_id, that redirect_uri coincides with the one in Step 2 , and that cod ehas not been redeemed before. If these checks are successful, the IdP issues an access token access_token in 9 . Now, the RP can use access_token to access the user's protected resources at the IdP(authorization) or log in the user (authentication), as described next. When OAuth is used for authorization, the RP uses the access token to view or manipulate the protected resource at the IdP (illustrated in Steps 10 and 11).For authentication, the RP fetches a user id (which uniquely identifies the user at the IdP) using the access token, Steps 12 and 13 .The RP then issues a session cookie to the user's browser as show nin 14 .4Tracking User Intention. Note that in order for an RP which supports multiple IdPs to process Step 7 , the RP must know which IdP a user wanted to

use for authorization. There are two different approaches to this used in practice: First, the RP can use different redirection URIs to distinguish different IdPs. We call this naïve user intention tracking. Second, the RP can store the user intentio nin a session after Step 1 and use this information later. We call this explicit user intention tracking. The same applies to the implicit mode of OAuth presented below.

Implicit Mode. This mode is similar to the authorization code mode, but instead of providing an authorization code, the IdP directly delivers an access token to the RP via the user's browser. More specifically, in the implicit mode, Steps 1 – 5 (see Figure 1)are the same as in the authorization code mode. Instead of create ingan authorization code, the IdP issues an access token right away and redirects the user's browser to RP's redirection endpoint with the access token contained in the fragment of the URI. (Recall that a fragment is a special part of a URI indicated by the '#' symbol.)As fragments are not sent in HTTP requests, the access token is not immediately transferred when the browser contacts the RP. Instead, the RP needs to use a JavaScript to retrieve the contents of the fragment. Typically, such a JavaScript is sent in RP's answer at the redirection endpoint. Just as in the authorization code mode, the RP can now use the access token for authorization or authentication(analogously to Steps 10 – 14 of Figure 1).

Resource Owner Password Credentials Mode. In this mode, the user gives her credentials for an IdP directly to an RP. The RP can then authenticate to the IdP on the user's behalf and retrieve an access token. This mode is intended for highly-trusted RPs, such as the operating system of the user's device or highly-privileged applications, or if the previous two modes are not possible to perform(e.g., for applications without a web browser).

Client Credentials Mode. In contrast to the modes shown above, this mode works without the user's interaction. Instead, it is started by an RP in order to fetch an access token to access the resources of RP at an IdP. For example, Facebook allows RPs to use the client credentials mode to obtain an access token to access reports of their advertisements' performance.

3. ATTACKS

As mentioned in the introduction, while trying to prove the security of OAuth based on the FKS web model and our OAuth model, we found four attacks on OAuth, which we call 307 redirect attack, IdP mix-up attack, state leak attack, and naïve RP session integrity attack, respectively. In this section, we provide detailed descriptions of these attacks along with easily implementable

fixes. Our formal analysis of OAuth (see Section 5) then shows that these fixes are indeed sufficient to establish the security of OAuth. The attacks also apply to OpenID Connect (see Section 3.5). Figure 2 provides an overview of where the attacks apply. We have verified our attacks on actual implementations of OAuth and OpenID Connect and reported the attacks to the respective working groups who confirmed the attacks (see Section 3.6).

3.1 307 Redirect Attack

In this attack, which breaks our authorization and authentication properties (see Section 5.2), the attacker (running a malicious RP)learns the user's credentials when the user logs in at an IdP that uses the wrong HTTP redirection status code. While the attack itself is based on a simple error, to the best of our knowledge, this is the first description of an attack of this kind.

Assumptions. The main assumptions are that (1) the IdP that issued for the login chooses the 307 HTTP status code when redirecting the user's browser back to the RP (Step 6 in Figure 1), and(2) the IdP redirects the user immediately after the user entered her credentials (i.e., in the response to the HTTP POST request that contains the form data sent by the user's browser).Assumption (1). This assumption is reasonable because neither the OAuth standard [20] nor the OAuth security considerations [26](nor the OpenID Connect standard [31]) specify the exact method of how to redirect. The OAuth standard rather explicitly permits any HTTP redirect: While the examples in this specification show the use of the HTTP 302 status code, any other method available via the user-agent to accomplish this redirections allowed and is considered to be an implementationdetail.5The response from the IdP in Step 13 includes the RP's OAuth client id, which is checked by the RP when authenticating a user(cf. RFC7662 [30]). This check prevents re-use of access tokens across RPs in the OAuth implicit mode, as explained in [37]. This check is not needed for authorization. Assumption (2). This assumption is reasonable as many examples for redirects immediately after entering the user credentials can be found in practice, for example at github.com (where, however, assumption (1) is not satisfied.)

Attack. When a user uses the authorization code or implicit mode of OAuth to log in at a malicious RP, then she is redirected to the IdP and prompted to enter her credentials. The IdP then receives these credentials from the user's browser in a POST request. It checks the credentials and redirects the user's browser to the RP's redirection endpoint in the response to the POST request. Since the 307 status code is used for this redirection, the user's browser will send a POST request

to RP that contains all form data from the previous request, including the user credentials. Since the RP is run by the attacker, he can use these credentials to impersonate the user. Fix. Contrary to the current wording in the OAuth standard, the exact method of the redirect is not an implementation detail but essential for the security of OAuth. In the HTTP standard [18],only the 303 redirect is defined unambiguously to drop the body of an HTTP POST request. Therefore, the OAuth standard should require 303 redirects for the steps mentioned above in order to fix this problem.

Attack on Authorization Code Mode.

We now describe the IdPMix-Up attack on the OAuth authorization code mode. As mentioned, a very similar attack also applies to the implicit mode. Both attacks also work if IdP supports just one of these two modes. The IdP mix-up attack for the authorization code mode is depicted in Figure 3. Just as in a regular flow, the attack starts when the user selects that she wants to log in using HIdP (Step 1 in Figure 3). Now, the attacker intercepts the request intended for theRP and modifies the content of this request by replacing HIdP byAIdP.8 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 OAuthclient id of the RP at AIdP with the client id of the RP at HIdP(which is public information). (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 theRP is encrypted by using HTTPS, and thus, cannot be inspected oraltered by the attacker.) The user then authenticates to HIdP and is redirected back to the RP 8 . The RP thinks, due to Step 2 of the attack, that the nonce code contained in this redirect was issued by AIdP, rather than HIdP. The RP therefore now tries to redeem thisnonce for an access token at AIdP 10 , rather than HIdP. This leaks code to the attacker. Breaking Authorization. If HIdP has not issued an OAuth client secret to RP during registration, the attacker can now redeem code for an access token at HIdP (in 11 and 12).9 This access token allows the attacker to access protected resources of the user at HIdP. This breaks the authorization property (see Section 5.2). We note that at this point, the attacker might even provide false information8At this point, the attacker could also read the session id for the user's session at RP. Our attack, however, is not based on this possibility and works even if the RP changes this session id as soonas the user is logged in and the connection is protected by HTTPS(a best practice for session

management).9In the case that RP has to provide a client secret, this would not work in this mode (see also Figure 2). Recall that in this mode, client secrets are optional.about the user or her protected resources to the RP: he could issue a self-created access token which RP would then use to access such information at the attacker.Breaking Authentication. To break the authentication property (see Section 5.2) and impersonate the honest user, the attacker, after obtaining code in Step 10 , starts a new login process (using his own browser) at the RP. He selects HIdP as the IdP for this login process and receives a redirect to HIdP, which he ignores. This redirect contains a cookie for a new login session and a fresh state parameter. The attacker now sends code to the RP imitating a real login (using the cookie and fresh state value from the previous response). The RP then retrieves an access token at HIdP using code and uses this access token to fetch the (honest) user's id. Being convinced that the attacker owns the honest user's account, the RP issues a session cookie for this account to the attacker. As a result, the attacker is logged in at the RP under the honest user's id. (Note that the attacker does not learn an access token in this case.)

4. CONCLUSION

In this paper, we carried out the first extensive formal analysis of OAuth 2.0 based on a comprehensive and expressive web model. Our analysis, which aimed at the standard itself, rather than specific OAuth implementations and deployments, comprises all modes (grant types) of OAuth and available options and also takes malicious RPs and IdPs as well as corrupted browsers/users into account. The generic web model underlying our model of OAuth and its analysis is the most comprehensive web model to date. Our in-depth analysis revealed four attacks on OAuth as well as OpenID connect, which builds on OAuth. We verified the attacks, proposed fixes, and reported the attacks and our fixes to the working groups for OAuth and OpenID Connect. The working groups confirmed the attacks. Fixes to the standard and recommendations are currently under discussion or already incorporated in a draft fora new RFC [22].With the fixes applied, we were able to prove strong authorization, authentication, and session integrity properties for OAuth 2.0. Our security analysis assumes that OAuth security recommendations and certain best practices are followed. We show that otherwise the security of OAuth cannot be guaranteed. By this, we also provide clear guidelines for implementations. The fact that OAuth is one of the most widely deployed authorization and authentication systems in the web and the basis for other protocols makes our analysis particularly relevant. As for future work, our formal

analysis of OAuth offers a good starting point for the formal analysis of OpenID Connect, and hence, such an analysis is an obvious next step for our research.

7. REFERENCES

[1] M. Abadi and C. Fournet. Mobile Values, New Names, andSecure Communication. In POPL 2001, pages 104–115.ACM Press, 2001.

[2] D. Akhawe, A. Barth, P. E. Lam, J. Mitchell, and D. Song.Towards a Formal Foundation of Web Security. In CSF 2010,pages 290–304. IEEE Computer Society, 2010.

[3] A. Armando, R. Carbone, L. Compagna, J. Cu ́ellar,G. Pellegrino, and A. Sorniotti. An authentication flaw in browser-based Single Sign-On protocols: Impact and remediations. Computers & Security, 33:41–58, 2013.Elsevier, 2013.

[4] A. Armando, R. Carbone, L. Compagna, J. Cu ́ellar, and M. L.Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-on: Breaking the SAML-based Single Sign-on for Google Apps. In FMSE 2008, pages 1–10. ACM, 2008.

[5] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis.Keys to the Cloud: Formal Analysis and Concrete Attacks on Encrypted Web Storage. In POST 2013, volume 7796 of LNCS, pages 126–146. Springer, 2013.

[6] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis.Discovering Concrete Attacks on Website Authorization by Formal Analysis. Journal of Computer Security,22(4):601–657, 2014. IOS Press, 2014.

[7] A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In CCS 2008, pages 75–88. ACM,2008.