**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 thefoundation for the new SSO standard OpenID Connect. Despite thepopularity of OAuth, so far analysis efforts were mostly targeted atfinding bugs in specific implementations and were based on formalmodels which abstract from many web features or did not providea formal treatment at all.In this paper, we carry out the first extensive formal analysis ofthe OAuth 2.0 standard in an expressive web model. Our analysis aims at establishing strong authorization, authentication, andsession 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. Theymay even run simultaneously in the same and different relying parties and identity providers, where malicious relying parties, identityproviders, and browsers are considered as well. Our modeling andanalysis of the OAuth 2.0 standard assumes that security recommendations and best practices are followed in order to avoid obviousand known attacks.When proving the security of OAuth in our model, we discoveredfour attacks which break the security of OAuth. The vulnerabilitiescan be exploited in practice and are present also in OpenID Connect.We propose fixes for the identified vulnerabilities, and then, forthe first time, actually prove the security of OAuth in an expressiveweb 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-basedprotocol that allows a user to grant web sites access to her resources(data or services) at other web sites (*authorization*). The formerweb 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 heridentity 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 themost 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, StackExchange, and Dropbox. This enables billions of users to log in atmillions of RPs or share their data with these [35], making OAuthone of the most used single sign-on systems on the web.OAuth is also the foundation for the new single sign-on protocolOpenID Connect, which is already in use and actively supportedby PayPal (“Log In with PayPal”), Google, and Microsoft, amongothers. Considering the broad industry support for OpenID Connect,a widespread adoption of OpenID Connect in the next years seemslikely. OpenID Connect builds upon OAuth and provides clearlydefined interfaces for user authentication and additional (optional)features, such as dynamic identity provider discovery and relyingparty registration, signing and encryption of messages, and logout.In OAuth, the interactions between the user and her browser,the RP, and the IdP can be performed in four different flows, or*grant types*: authorization code grant, implicit grant, resource ownerpassword credentials grant, and the client credentials grant (we referto these as *modes* in the following). In addition, all of these modesprovide further options.The goal of this work is to provide an in-depth security analysisof OAuth. Analyzing the security of OAuth is a challenging task,on the one hand due to the various modes and options that OAuthprovides, and on the other hand due to the inherent complexity ofthe web.So far, most analysis efforts regarding the security of OAuth weretargeted towards finding errors in specific implementations [6, 10,25, 33, 34, 36, 38], rather than the comprehensive analysis of thestandard itself. Probably the most detailed formal analysis carriedout on OAuth so far is the one in [6]. However, none of the existinganalysis efforts of OAuth account for all modes of OAuth runningsimultaneously, which may potentially introduce new security risks.In fact, many existing approaches analyze only the authorizationcode mode and the implicit mode of OAuth. Also, importantly,there are no analysis efforts that are based on a comprehensiveformal web model (see below), which, however, is essential to rule1Following the OAuth 2.0 terminology, IdPs are called *authorization servers* and *resource servers*, RPs are called *clients*, andusers are called *resource owners*. Here, however, we stick to themore common terms mentioned above.2Note that in this document, we consider only OAuth 2.0, whichis very different to its predecessor, OAuth 1.0(a).out security risks that arise when running the protocol in the contextof common web technologies (see Section 6 for a more detaileddiscussion of related work).

**Contributions of this Paper.** We perform the first extensive formalanalysis of the OAuth 2.0 standard for all four modes, which caneven run simultaneously within the same and different RPs and IdPs,based on a comprehensive web model which covers large parts ofhow browsers and servers interact in real-world setups. Our analysisalso covers the case of malicious IdPs, RPs, and browsers/users.*Formal model of OAuth.* Our formal analysis of OAuth uses anexpressive Dolev-Yao style model of the web infrastructure [14]proposed by Fett, Küsters, and Schmitz (FKS). The FKS model hasalready been used to analyze the security of the BrowserID singlesign-on system [14, 15] as well as the security and privacy of theSPRESSO single sign-on system [16]. This web model is designedindependently of a specific web application and closely mimicspublished (de-facto) standards and specifications for the web, forinstance, 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 severalheaders, 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, includingthe complex navigation rules, as well as new technologies, such asweb storage and web messaging (via postMessage). JavaScript ismodeled in an abstract way by so-called *scripts* which can be sentaround and, among others, can create iframes and initiate XMLHTTPRequests (XHRs). Browsers may be corrupted dynamicallyby the adversary.Using the generic FKS model, we build a formal model of OAuth,closely following the OAuth 2.0 standard (RFC6749 [20]). Sincethis RFC does not fix all aspects of the protocol and in order toavoid known implementation attacks, we use the OAuth 2.0 securityrecommendations (RFC6819 [26]), additional RFCs and OAuthWorking Group drafts (e.g., RFC7662 [30], [8]) and current webbest practices (e.g., regarding session handling) to obtain a model ofOAuth with state-of-the-art security features in place, while makingas few assumptions as possible. Moreover, as mentioned above, ourmodel includes RPs and IdPs that (simultaneously) support all fourmodes 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 turnis 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, whichbreaks the authorization and authentication properties, IdPs inadvertently forward user credentials (i.e., username and password) tothe RP or the attacker. In the second attack (IdP mix-up), a networkattacker playing the role of an IdP can impersonate any victim. Thissevere 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 loggedin 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 thesession integrity property. We have verified all four attacks on actualimplementations of OAuth and OpenID Connect. We present ourattacks 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 toimplement in new and existing deployments of OAuth and OpenIDConnect.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 withthe fixes in place, we then were able to prove that OAuth satisfiesthe mentioned security properties. This is the first proof whichestablishes central security properties of OAuth in a comprehensiveand expressive web model (see also Section 6).We emphasize that, as mentioned before, we model OAuth withsecurity recommendations and best practices in place. As discussedin Section 5, implementations not following these recommendationsand best practices may be vulnerable to attacks. In fact, many suchattacks on specific implementations have been pointed out in theliterature (e.g., [6, 10, 20, 25, 26, 36, 37]). Hence, our results alsoprovide guidelines for secure OAuth implementations.We moreover note that, while these results provide strong securityguarantees for OAuth, they do not directly imply security of OpenIDConnect because OpenID Connect adds specific details on top ofOAuth. We leave a formal analysis of OpenID Connect to futurework. The results obtained here can serve as a good foundation forsuch an analysis.

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

1. **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 theremaining three modes (grant types).OAuth was first intended for *authorization*, i.e., users authorizeRPs to access user data (called *protected resources*) at IdPs. Forexample, a user can use OAuth to authorize services such as IFTTT3to access her (private) timeline on Facebook. In this case, IFTTT isthe RP and Facebook the IdP.Roughly speaking, in the most common modes, OAuth worksas follows: If a user wants to authorize an RP to access some ofthe 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 grantthe RP access to some of her user data at the IdP. Then, along withsome token (an *authorization code* or an *access token*) issued by theIdP, the user is redirected back to the RP. The RP can then use thetoken as a credential at the IdP to access the user’s data at the IdP.OAuth is also commonly used for *authentication*, although it wasnot designed with authentication in mind. A user can, for example,use her Facebook account, with Facebook being the IdP, to log in atthe social network Pinterest (the RP). Typically, in order to log in,the user authorizes the RP to access a unique user identifier at theIdP. The RP then retrieves this identifier and considers this user tobe logged in. 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 amanual 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 theterm “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 OAuthmodes, the IdP redirects the user’s browser to one of these URIs.Note that (depending on the implementation of an IdP) an RP mayalso register a pattern as a redirect URI and then specify the exactredirect 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 ouranalysis), in the following descriptions, we consider only a specificset of options. For example, we assume that an RP always providesa redirect URI and shares an OAuth client secret with the IdP.

**Authorization Code Mode.** When the user tries to authorize anRP to access her data at an IdP or to log in at an RP, the RP firstredirects the user’s browser to the IdP. The user then authenticatesto the IdP, e.g., by providing her user name and password, andfinally is redirected back to the RP along with an *authorizationcode* generated by the IdP. The RP can now contact the IdP withthis authorization code (along with the client id and client secret)and receive an *access token*, which the RP in turn can use as acredential to access the user’s protected resources at the IdP.*Step-by-Step Protocol Flow.* In what follows, we describe the protocol flow of the authorization code mode step-by-step (see alsoFigure 1). First, the user starts the OAuth flow, e.g., by clickingon a button to select an IdP, resulting in request 1 being sent to theRP. 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 asa token to prevent CSRF attacks). The RP then redirects the browserto the so-called *authorization endpoint* URI at the IdP in 2 and 3with its *client*\_*id*, *redirect*\_*uri*, and *state* appended as parameters tothe URI. The IdP then prompts the user to provide her usernameand password in 4 . The user’s browser sends this information tothe IdP in 5 . If the credentials are correct, the IdP creates a nonce*code* (the authorization code) and redirects the user’s browser toRP’s redirection endpoint URI *redirect*\_*uri* in 6 and 7 with *code*and *state* appended as parameters to the URI. If *state* is the sameas above, the RP contacts the IdP in 8 and provides *code*, *client*\_*id*,*client*\_*secret*, and *redirect*\_*uri*. Then the IdP checks whether thisinformation is correct, i.e., it checks that *code* was issued for the RPidentified by *client*\_*id*, that *client*\_*secret* is the secret for *client*\_*id*,that *redirect*\_*uri* coincides with the one in Step 2 , and that *code*has not been redeemed before. If these checks are successful, theIdP issues an access token *access*\_*token* in 9 . Now, the RP canuse *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 accesstoken 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 uniquelyidentifies 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 shownin 14 .4*Tracking User Intention.* Note that in order for an RP which supports multiple IdPs to process Step 7 , the RP must know whichIdP a user wanted to use for authorization. There are two differentapproaches to this used in practice: First, the RP can use differentredirection URIs to distinguish different IdPs. We call this *naïveuser intention tracking*. Second, the RP can store the user intentionin a session after Step 1 and use this information later. We call this*explicit user intention tracking*. The same applies to the implicitmode of OAuth presented below.

**Implicit Mode.** This mode is similar to the authorization codemode, 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 creatingan authorization code, the IdP issues an access token right away andredirects the user’s browser to RP’s redirection endpoint with theaccess token contained in the fragment of the URI. (Recall that afragment is a special part of a URI indicated by the ‘#’ symbol.)As fragments are not sent in HTTP requests, the access tokenis not immediately transferred when the browser contacts the RP.Instead, the RP needs to use a JavaScript to retrieve the contents ofthe fragment. Typically, such a JavaScript is sent in RP’s answer atthe 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, theuser gives her credentials for an IdP directly to an RP. The RP canthen authenticate to the IdP on the user’s behalf and retrieve an access token. This mode is intended for highly-trusted RPs, such asthe 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 startedby an RP in order to fetch an access token to access the resourcesof RP at an IdP. For example, Facebook allows RPs to use the clientcredentials mode to obtain an access token to access reports of theiradvertisements’ performance.

1. **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 integrityattack*, respectively. In this section, we provide detailed descriptionsof these attacks along with easily implementable fixes. Our formalanalysis of OAuth (see Section 5) then shows that these fixes areindeed sufficient to establish the security of OAuth. The attacks alsoapply to OpenID Connect (see Section 3.5). Figure 2 provides anoverview of where the attacks apply. We have verified our attackson actual implementations of OAuth and OpenID Connect and reported the attacks to the respective working groups who confirmedthe attacks (see Section 3.6).

* 1. **307 Redirect Attack**

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

**Assumptions.** The main assumptions are that (1) the IdP that isused 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 hercredentials (i.e., in the response to the HTTP POST request thatcontains the form data sent by the user’s browser).*Assumption (1).* This assumption is reasonable because neither theOAuth standard [20] nor the OAuth security considerations [26](nor the OpenID Connect standard [31]) specify the exact methodof how to redirect. The OAuth standard rather explicitly permitsany HTTP redirect:While the examples in this specification show the useof the HTTP 302 status code, any other method available via the user-agent to accomplish this redirectionis allowed and is considered to be an implementationdetail.5The response from the IdP in Step 13 includes the RP’s OAuthclient id, which is checked by the RP when *authenticating* a user(cf. RFC7662 [30]). This check prevents re-use of access tokensacross RPs in the OAuth implicit mode, as explained in [37]. Thischeck is not needed for authorization.*Assumption (2).* This assumption is reasonable as many examplesfor redirects immediately after entering the user credentials canbe 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 modeof OAuth to log in at a *malicious* RP, then she is redirected to theIdP and prompted to enter her credentials. The IdP then receivesthese credentials from the user’s browser in a POST request. Itchecks the credentials and redirects the user’s browser to the RP’sredirection endpoint in the response to the POST request. Sincethe 307 status code is used for this redirection, the user’s browserwill send a POST request to RP that contains all form data from theprevious request, including the user credentials. Since the RP is runby the attacker, he can use these credentials to impersonate the user.**Fix.** Contrary to the current wording in the OAuth standard, theexact method of the redirect is not an implementation detail butessential for the security of OAuth. In the HTTP standard [18],only the 303 redirect is defined unambiguously to drop the bodyof an HTTP POST request. Therefore, the OAuth standard shouldrequire 303 redirects for the steps mentioned above in order to fixthis 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. Bothattacks 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 whenthe 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 itredirects 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 thispoint 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 isredirected back to the RP 8 . The RP thinks, due to Step 2 of theattack, that the nonce *code* contained in this redirect was issued byAIdP, 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 clientsecret to RP during registration, the attacker can now redeem *code*for an access token at HIdP (in 11 and 12 ).9 This access tokenallows the attacker to access protected resources of the user at HIdP.This breaks the authorization property (see Section 5.2). We notethat at this point, the attacker might even provide false information8At this point, the attacker could also read the session id forthe user’s session at RP. Our attack, however, is not based on thispossibility 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 wouldnot 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 issuea self-created access token which RP would then use to access suchinformation at the attacker.*Breaking Authentication.* To break the authentication property (seeSection 5.2) and impersonate the honest user, the attacker, afterobtaining *code* in Step 10 , starts a new login process (using his ownbrowser) at the RP. He selects HIdP as the IdP for this login processand receives a redirect to HIdP, which he ignores. This redirectcontains a cookie for a new login session and a fresh state parameter.The attacker now sends *code* to the RP imitating a real login (usingthe cookie and fresh state value from the previous response). TheRP then retrieves an access token at HIdP using *code* and uses thisaccess token to fetch the (honest) user’s id. Being convinced thatthe attacker owns the honest user’s account, the RP issues a sessioncookie for this account to the attacker. As a result, the attackeris logged in at the RP under the honest user’s id. (Note that theattacker does not learn an access token in this case.)

1. **CONCLUSION**

In this paper, we carried out the first extensive formal analysis of OAuth 2.0 based on a comprehensive and expressive webmodel. Our analysis, which aimed at the standard itself, rather thanspecific OAuth implementations and deployments, comprises allmodes (grant types) of OAuth and available options and also takesmalicious RPs and IdPs as well as corrupted browsers/users intoaccount. The generic web model underlying our model of OAuthand its analysis is the most comprehensive web model to date.Our in-depth analysis revealed four attacks on OAuth as well asOpenID 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 groupsconfirmed the attacks. Fixes to the standard and recommendationsare 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. Oursecurity analysis assumes that OAuth security recommendationsand certain best practices are followed. We show that otherwise thesecurity of OAuth cannot be guaranteed. By this, we also provideclear guidelines for implementations. The fact that OAuth is one ofthe most widely deployed authorization and authentication systemsin the web and the basis for other protocols makes our analysisparticularly relevant.As for future work, our formal analysis of OAuth offers a goodstarting 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. Cuéllar,G. Pellegrino, and A. Sorniotti. An authentication flaw inbrowser-based Single Sign-On protocols: Impact andremediations. *Computers & Security*, 33:41–58, 2013.Elsevier, 2013.

[4] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L.Tobarra. Formal Analysis of SAML 2.0 Web Browser SingleSign-on: Breaking the SAML-based Single Sign-on forGoogle 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 onEncrypted 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 byFormal Analysis. *Journal of Computer Security*,22(4):601–657, 2014. IOS Press, 2014.

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