# SoK: Single Sign-On Security – An Evaluation of OpenID Connect

Christian Mainka,
Vladislav Mladenov and
Jörg Schwenk
*Horst Görtz Institute for IT Security*
*Ruhr University Bochum*

Tobias Wich
*ecsec GmbH*

## Abstract

OpenID Connect is the OAuth 2.0-based replacement for OpenID 2.0 (OpenID) and one of the most important Single Sign-On (SSO) protocols used for delegated authentication. It is used by companies like Amazon, Google, Microsoft, and PayPal. In this paper, we systematically analyze well-known attacks on SSO protocols and adapt these on OpenID Connect. We additionally introduce two novel attacks on OpenID Connect, *Identity Provider Confusion* and *Malicious Endpoints Attack* abusing lacks in the current specification and breaking the security goals of the protocol. We communicated these attacks in 2014 with the authors of the OpenID Connect specification and helped to develop a fix (currently an RFC Draft).

We categorize the described attacks in two classes: Single-Phase Attacks abusing a lack of a single security check and Cross-Phase Attacks requiring a complex attack setup and manipulating multiple messages distributed across the whole protocol workflow. We provide an evaluation of officially referenced OpenID Connect libraries and find 75% of them vulnerable to at least one Single-Phase Attack. All libraries are susceptible Cross-Phase Attacks which is not surprising since the attacks abuse a logic flaw in the protocol and not an implementation error. We reported the found vulnerabilities to the developers and helped them to fix the issues. We address the existing problems in a Practical Offensive Evaluation of Single Sign-On Services (PrOfESSOS). PrOfESSOS is our open source implementation for *fully automated* Evaluation-as-a-Service for SSO. PrOfESSOS introduces a generic approach to improve the security of OpenID Connect implementations by systematically detecting vulnerabilities. In collaboration with the IETF OAuth and OpenID Connect working group, we integrate PrOfESSOS into the OpenID Connect certification process.

PrOfESSOS is available at https://openid.sso-security.de

*The research was supported by the German Ministry of research and Education (BMBF) as part of the* VERTRAG *research project and the European Commission through the FutureTrust project (grant 700542-Future-Trust-H2020-DS-2015-1).*

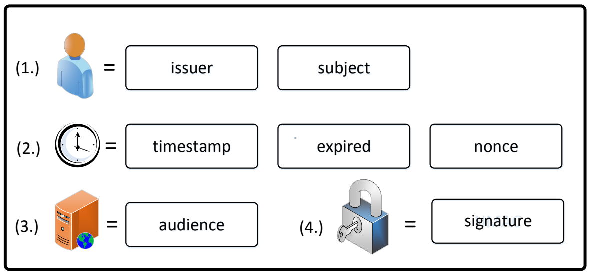


Figure 1: Abstract overview of an SSO authentication token.

## 1. Introduction

Single Sign-On (SSO) is a concept to delegate the authentication of an End-User on a Service Provider (SP) to a third party: the so-called Identity Provider (IdP). Standardized in 2014, *OpenID Connect* is the latest SSO protocol and supported by large companies like Google, Microsoft and PayPal. In 2015, Google announced to abandon the preceding protocol OpenID 2.0 (OpenID) and recommended switching to its OAuth 2.0 (OAuth) based successor OpenID Connect. The OpenID Connect specification itself offers a list of available libraries supporting OpenID Connect [25], and an additional list of certified libraries [24]. On the one hand, using such a library makes the integration of OpenID Connect into a web application quite easy since the entire authentication (including all security-related operations) can be delegated to it. On the other hand, the security of the web application then depends on the used library.

**SSO Attacks.** We investigated attacks on well-known SSO protocols including SAML [2, 29], BrowserId [9, 10], and OpenID [19, 33] to get a systematization for the different OpenID Connect protocol. The essence of each SSO protocol is the authentication token containing statements regarding the End-User who is going to be authenticated.

These statements can be grouped into four classes: (1.) identity, (2.) recipient, (3.) freshness, and (4.) signature. The authentication token is similar across all SSO protocols (Figure 1 shows the case of OpenID Connect) and form the basis of our identified Single-Phase Attacks: by changing one or more of the values in the token different attacks can be conducted. For example, the XML Signature Exclusion attack on SAML [29] targets the signature statements, while the identity spoofing attack on OpenID [19] and on BrowserId [9] targets the identity statements.

More complex attacks are Cross-Phase Attacks. They target not only the token statements, but multiple protocol messages in different phases of the protocol execution. Due to the high complexity of such attacks, they are very hard to identify. In this paper, we describe three different Cross-Phase Attacks: one implementation flaw (Issuer Confusion), and two specification flaws (IdP Confusion, Malicious Endpoints) that have not been previously discovered and could only be revealed through a deep protocol understanding of the relation between all SSO messages.

The specification flaws have been reported to the authors of the OpenID Connect specification (together with another independent research group which has identified similar results on OAuth). In collaboration with the OAuth and OpenID Connect working groups, we have created a fix for them that is currently available as a draft [17].

**Research Challenges.** Correctly implementing SSO libraries is a challenging task and previous research on different protocols has revealed serious vulnerabilities in implementations [19, 29, 32, 41]. We focus on OpenID Connect, because it is the latest SSO protocol, and includes most features from previous SSO protocols, which opens doors for all potential vulnerabilities. OpenID Connect is by far more complex than any previous SSO protocol. It supports different protocol variants (called *flows*), diverse SP types (Websites, mobile apps and native applications), and extensively uses server-to-server communication. In summary, this leads to new challenges and insights during its investigation. This paper answers three general research questions:

(Q1) OpenID Connect – as the latest SSO protocol – should be aware of previous and known attacks. Which existing attacks are addressed by the specification and which are not?
(Q2) How *secure* are the *officially* referenced implementations and do they follow the specification hints regarding attacks?
(Q3) How can the implementation of SSO libraries be improved regarding state-of-the-art security?

**Evaluation-as-a-Service.** In this paper, we show how attacks, partially known from other SSO systems, can be adapted to the new SSO system OpenID Connect. The adaption ranges from simple format changes (e.g. replay attacks) to complex Cross-Phase Attacks.

Our results highlight that even simple and well-studied attacks still exist and remain an open problem. As depicted in Table 2 (p. 14), 75% of the evaluated libraries were susceptible to at least one critical issue resulting into *broken End-User authentication*. Even if the specification addresses such issues (Q1), implementations are vulnerable to them (Q2). The gap between *specification* and *implementation* has several reasons ranging from too complex specifications to standard developer mistakes and forgotten checks.

To address this problem, and to bring an implementation *closer* to the state-of-the-art (Q3), we propose a Practical Offensive Evaluation of Single Sign-On Services (PrOfESSOS). PrOfESSOS is a security Evaluation-as-a-Service (EaaS) for SSO, applicable to all existing libraries, independent of the used programming language. To the best of our knowledge, this is the most beneficial and reliable approach to provide a practical evaluation of SSO security and it can be integrated into the development cycle of new as well as existing OpenID Connect libraries. To execute all attacks PrOfESSOS applies a barely known penetration testing concept – it simulates both, an honest IdP and an attacker IdP [19]. The concept of using attacker IdP means that the IdP behaves maliciously with respect to the protocol flow, for example, by removing or manipulating parameters, or by sending messages in a wrong order. Thus, PrOfESSOS controls more SSO related messages and supports more (and especially more complex) attacks than other SSO evaluation tools.

In addition to finding security issues, PrOfESSOS offers information on the vulnerabilities and advices how to fix them. A demo is available at https://openid.sso-security.de.

**Open Source.** In recent years, many research papers implemented automated validation tools. These tools were then left unpublished or even protected by patents. Published tools are not documented or used. We believe that proper open source tools help the scientific community to proceed with the research and produce verifiable and comparable results. In addition, this may help other researchers to quickly solve annoying but repeating challenges (e.g., the automatism to login into a website). PrOfESSOS is open source and available on Github.[1]

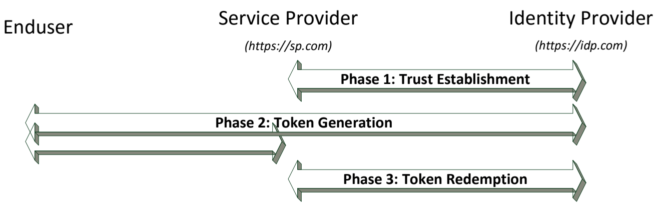
**Our Contribution.**

- We show how to adapt attack patterns known from other SSO protocols to OpenID Connect and categorize them into *Single-Phase Attacks* and *Cross-Phase Attacks*.
- We present two unpublished Cross-Phase Attacks vulnerabilities on OpenID Connect (IdP Confusion, Malicious Endpoints Attack), which abuse a logical flaw in the OpenID Connect specification.
- We provide a security evaluation of all officially listed OpenID Connect libraries. We responsibly disclosed all security issues to the according developers and helped to fix them.
- We provide PrOfESSOS, our comprehensive open source Evaluation-as-a-Service platform. PrOfESSOS is the first tool automatically analyzing OpenID Connect implementations and capable to evaluate high complex Cross-Phase Attacks.
- We are cooperating with the OAuth and OpenID Connect working group to integrate PrOfESSOS into the OpenID Connect certification process to support the development and security of SSO.

1. https://github.com/RUB-NDS/PrOfESSOS

## 2. Single Sign-On in Three Phases

Every SSO protocol consists of three phases (cf. Figure 2): (1.) Phase 1: registration and trust establishment between Service Provider (SP) and Identity Provider (IdP), (2.) Phase 2: End-User authentication on the IdP, (3.) Phase 3: End-User authentication on the SP via the authentication token.



Enduser
Service Provider
(https://sp.com)
Identity Provider
(https://idp.com)
Phase 1: Trust Establishment
Phase 2: Token Generation
Phase 3: Token Redemption

Figure 2: SSO consists of three phases.

**Phase 1: Trust Establishment.** In the first phase the trust establishment phase between SP and IdP is provided. In *classical* SSO systems, *trust* is established by an administrator *manually registering* a specific IdP on the SP. A typical example is SAML: The administrator visits the IdP and downloads the IdP's metadata, for instance its certificate. Next, he uploads it on the SP and configures further parameters like important URLs of the IdP. We call this *full trust establishment*, since only authorized people (the administrator) can invoke this manual trust establishment.

Modern SSO systems offer a more flexible alternative to this approach: *dynamic registration* that is executed automatically on the fly during an End-User's login procedure. It is supported by protocols like OpenID, BrowserId, OAuth and OpenID Connect and basically works as follows: (1.) The End-User starts a login process on the SP by submitting his identifier (e.g., his email `bob@idp.com`). (2.) The SP extracts the domain of it (the part after the @-sign) to identify the IdP. (3.) Then, the SP can dynamically register on the IdP, for example, by sending a POST request to a specified URL on the IdP. We call this *conditional trust establishment*, since every End-User can invoke this dynamic trust establishment on a custom IdP.

SSO systems based on a *conditional trust establishment* require additional verification steps by processing an SSO token. If these steps are not implemented correctly, attacks such as ID Spoofing (IDS) (see Section 5.1) are applicable.

**Phase 2: Token Generation.** In the second phase, the SP typically forwards the End-User to the IdP. This is usually an HTTP redirect to a pre-registered URL on the IdP with additional parameters (e.g., the identity of the SP). The End-User then logs in at the IdP, which then generates an SSO token. The token is then submitted to the SP.

**Phase 3: Token Redemption.** In the final phase, the SP receives the SSO token in order to authenticate the End-User. This is a security critical process, since the token contains multiple parameters which must be verified.

**SSO Token Structure.** Independent of the concrete SSO protocol implementation, every SSO token contains information that can be categorized as follows:

- ► **Class I: *Identity*.** The SSO token contains information about a subject (e.g. an End-User) authenticating at the SP. In some protocols, this is an email address. Others like OpenID, OpenID Connect and BrowserId use unique URLs or usernames. An important fact about this category is that multiple parameters (and not only one) represent the unique identity of the End-User.
- ► **Class II: *Recipient*.** An SSO token contains information on the intended recipient, for example, the URL or a unique ID of the SP.
- ► **Class III: *Freshness*.** Parameters like timestamps and nonces belong to this category.
- ► **Class IV: *Signature*.** The SSO token (or a subset of it) can be signed. The signature value as well as parameters specifying meta information like key references or used algorithms belong to this category.

## 3. Threat Model

The Single-Phase and Cross-Phase Attacks presented in the following sections have the goal to login with an identity of some victim.[2] We distinguish between two categories describing the behavior of the victim:

**Category $\mathcal{A}$ (Cat $\mathcal{A}$).** Attacks belonging to this category need certain interaction of the victim. For example, the victim has to click on a link, or he has posted his (expired) token somewhere on the web (e.g., in a support forum).

**Category $\mathcal{B}$ (Cat $\mathcal{B}$).** This category is stealthy for the victim: it does not need any interaction. This means that the attacker can log in with an arbitrary identity, for example, an @google.com identity, on the SP simply by using his attacker IdP. Cat $\mathcal{B}$ attacks are more powerful than Cat $\mathcal{A}$ since no user interaction is necessary.

**Attacker Capabilities.** The attacks introduced in this paper have been strictly verified in the web attacker model. Thus, the attacker does not control the network and is not able to eavesdrop or manipulate network communications. He can to set up a web service accessible on the Internet. The attacker can use links (e.g. by posting them in web-blogs) to lure the victim into opening a URL.

We assume that TLS channels are secure, the End-User does not use a compromised/malicious software, and the End-User detects Phishing attacks.

## 4. Attacker IdPs in SSO

One approach to analyze SSO is by using attacker IdPs [9, 19]. At first glance this seems trivial: an IdP is considered as a Trusted Third Party (TTP), allowing it to compromise the security of the entire SP. But in protocols such as OpenID, BrowserId, OAuth and OpenID

2. We also present Malicious Endpoints Attack with different goals, e.g., Denial-of-Service or Server Side Request Forgery.

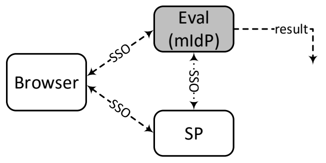


Figure 3: Attacker IdPs.

Connect, the IdPs are conditionally trusted. In other words: the protocols themselves provide mechanisms detecting a maliciously acting attacker IdP. It depends on the SP if these mechanisms are implemented correctly.

**Advantages.** By using the concept of attacker IdPs more messages can be analyzed since the used IdP takes part in every phase and step of the protocol, controls the content of every message, and can provide tests with valid or invalid tokens. Note that the attacker IdP does not control the communication between the browser and the SP, but since it controls the messages that are forwarded to the SP (e.g. by using HTTP redirects), he can even manipulate these messages. Thus, this approach gives the full flexibility to analyze an SSO protocol implementation.

**Disadvantages.** An important limitation is that such attacker IdPs are bound to one protocol. An implementation of every SSO protocol has to be created, but this is an acceptable downside, since even the attacks themselves have to be adapted to the specific protocol.

**Applicability.** There exist multiple SSO libraries providing IdP functionality. Thus, every attacker can download, install and configure such a library and deploy his own IdP – a *custom* IdP. An attacker IdP is a custom IdP acting maliciously by creating invalid or malicious tokens or exchanging the order of sent messages [9, 19]. There are two ways enforcing an SP to use the attacker IdP.

*Manual configuration of the SP*. First, an SP has to establish a trust relationship with the IdP. For this purpose, key material and important URLs (e.g., the URL of the IdP and the callback URL of the SP) are exchanged manually. As a result, the SP is able to verify the signature contained in the received authentication tokens.

*Dynamic configuration.* In OpenID Connect, the features Discovery [35] and Dynamic Registration [36] can be used to establish on-the-fly a trust relationship between an SP and the custom IdP (conditional trust establishment). In other words, a user can enforce the usage of the attacker IdP by entering its URL on the SP. The SP discovers all necessary information about the IdP, such as supported cryptographic algorithms, important URLs and exchanges the key material. Similar approach is described for other SSO protocols like SAML [5], OpenID [30], and OAuth [15].

## 4.1. Comparison with other Approaches

By systematically studying previous work on SSO analysis, we identify five other approaches to analyze SSO: manual testing, formal analysis, static code analysis, dynamic code analysis, and message invariants. We compare these with our attacker IdP approach and describe the results.

**Manual testing.** Even though manual testing is often considered as out-of-scope in research papers and even not described directly, it is an important part during every security evaluation. Manual testing offers great flexibility, which is needed at least at the beginning. Most research ideas start by manual testing, so does this paper: we first applied our attacks manually on multiple libraries to prove our ideas and methodology. Since manual testing does not scale for a larger number of targets, manual testing cannot be the answer to research question Q3.

**Formal Analysis.** This approach provides a security evaluation based on a formal protocol description, see Figure 4a. The main goal of such analysis is to find *generic* vulnerabilities. An important limitation however is that implementation flaws and specific vulnerabilities are not covered, since the *Evaluation* module (Eval) is not able to analyze real network traffic and to see, how an implementation reacts to manipulated messages.

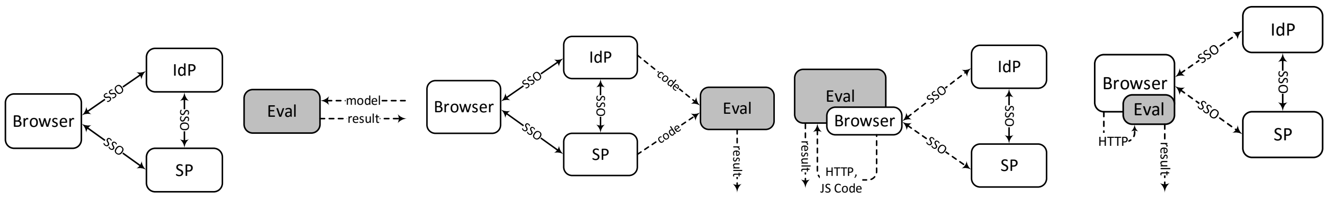
This approach does not answer research question Q3 (and even Q2), because specifications precisely address several attacks. For example, the OpenID Connect specification clearly states how to prevent Replay attacks, but our observation via manual testing was, that implementations are not aware of this. The real advantage of formal analysis relies in finding generic issues in specifications like in SAML [1], BrowserId [9] and OAuth [12]. Thus, in a perfect world, this step should be included during the creation of a specification to proof its correctness, which was also stated by Bai et al. [3].

**Static Code Analysis.** During a static code analysis, the *Eval* module has full access to the code of the targeted system. Figure 4b illustrates this approach.

Analyzing the program code is a reliable approach to track down implementation flaws, for example, in the verification logic. However, static code analysis is hardly used for SSO protocols. We see two reasons for this: (1.) such analysis requires full (or at least partial) access to the code of the running implementation (see dashed lines). This is possible for a library evaluation (especially for open-source libraries), but it is not applicable to online websites (e.g., Amazon). (2.) The biggest downside of static code analysis is, that it has to be applied for every programming language and in addition, for every SSO protocol.

**Dynamic Code Analysis.** Bai et al. [3] showed a more convenient approach combining program code analysis with formal analysis (cf. Figure 4c). In this case the *Eval* module expects HTTP traces plus initial knowledge, for example, the credentials of at least two different users. The *Eval* module then creates formal model based on this information, generates test cases, and simulates the SSO authentication. The output of the tests can then be further inspected to identify vulnerabilities.

Although dynamic code analysis is very flexible, it has significant downsides: (1.) signed messages are considered as unmodifiable and thus not evaluated. This is an enormous



(a) Formal Analysis. (b) Static Code Analysis. (c) Dynamic Code Analysis. (d) Message invariants.

Figure 4: The *Eval* module has access to different messages and information in different approaches. *Dashed lines* are accessible, *solid* lines are not accessible for *Eval*.

limitation regarding *conditionally trusted* SSO protocols since attacks like IDS are not considered. (2.) the direct communication between IdP and SP cannot be evaluated. By this means, our Cross-Phase Attacks cannot be evaluated.

**Message invariants.** The *message invariants* approach [37, 41] is depicted in Figure 4d. The *Eval* module again expects HTTP traces and analyzes the traffic to define messages and parameters that are modifiable. Instead of creating large subset of tests by executing all possibilities, it examines relations between different parameters. Such relations can be used to influence parameters in one message, which automatically reflects in another one. In this manner points of attacks can be identified and evaluated. The limitations of this approach are identical with the previous one.

### 4.2. Lessons Learned

Since we need a scalable solution that can be applied to numerous implementations, we need an approach which can evaluate an implementation automatically. Therefore, the *manual* approach is omitted. A *formal* approach unfortunately does not help to find implementation issues, so we cannot apply it, too. *Static* and *dynamic code* analysis have the disadvantage that they depend on the used programming language. Thus, this approach is difficult to apply, since we have to implement this approach in dependence of each language. In addition, the attack surface is limited by the inaccessible communication between the SP and IdP. Thus, this approach cannot bring an implementation *closer* to a specification. The *message invariant* approach seems to be promising, but it has the huge disadvantage that the communication between the SP and the IdP is not part of the evaluation. In addition, not all messages sent through the browser can be manipulated. This disables the evaluation of, for example, Issuer Confusion attack.

To the best of our knowledge, the *attacker IdP approach* is the best way to analyze implementation flaws in SSO since it offers great flexibility, full control over all phases and messages and can be used as a basis for an automated analysis.

**Q3: How can an implementation get closer to a specification?** Before we thought about applying an automatic testing approach, we did a manual analysis with only a few libraries. One surprising result was, even very *simple* (not to say stupid) implementations issues were made. The best example for this are Replay attacks, which are clearly addressed by the OpenID Connect specification, but despite this fact, we could find them in a variety of implementations.

If we want to bring an implementation closer to a specification, we need to validate if the implementation behaves correctly in exceptional scenarios. This is commonly known as *compliance testing* and is a part of every serious software development. However, when it comes to security, these tests are missing. This may have different reasons, for example, that developers are not aware of attacks, or they do not understand the importance of a verification step in a specification (*"Why should one check the recipient of a token?"*).

## 5. Single-Phase Attacks

We executed an extensive research on different SSO protocols and known attacks. By this means, we categorize attacks on SSO into Single-Phase Attacks and Cross-Phase Attacks (cf. Section 6). Most known attacks on SSO abuse an insufficient or missing verification step on the SSO token or one of the security relevant parameters. If this step appears at one single point, for example, at the SP receiving the token in Phase 3, we talk about Single-Phase Attacks. The identification of such vulnerabilities is relatively easy, because a Single-Phase Attack can be conducted by only manipulating at most one message in one phase of the SSO protocol.

In the following, we systematically analyze problems that can occur if such a verification step is not implemented properly and map them precisely on the OpenID Connect protocol.

### 5.1. Class I Attack: ID Spoofing

The IDS attack targets the **Identity** related information (Class I) in the SSO token and belongs to Cat $\mathcal{B}$. The idea of IDS is that the attacker starts a login attempt on the SP using his attacker IdP. The attacker IdP then generates an SSO token for an identity managed by another honest IdP, for example, by Google, but signs it with its own key. If this SSO token is accepted by the SP, the attacker is logged in and has access to accounts managed by other honest IdPs.

In OpenID Connect, the identity of an End-User is represented by the combination of two parameters: (1.) *sub* defining the identity of the authenticated End-User on the IdP. (2.) *iss* defining the issuer of the token. Usually this is the URL of the IdP. Here, the attacker can try two different attacks. First, he can change the *sub* value to the victim's one. Second, he can change both, *sub* and *iss*, to the victims' ones using his attacker IdP. If applicable the impact of the attack is devastating since the attacker can login on every account on the SP (Cat $\mathcal{B}$).

As a countermeasure for IDS, the SP must verify that the attacker IdP is not allowed to issue tokens managed by another IdP. This can be implemented by checking the `iss` value and verifying the signature with the key material corresponding to this `iss`.

The concept of this attack has already been described in 2016 on OpenID [19, `claimed_id`, `identity`, `email` parameter], and BrowserId [9].

## 5.2. Class II Attack: Wrong Recipient

The idea of the Wrong Recipient attack is that an attacker acting as a malicious SP receives SSO tokens from different users. Behind the scenes he tries to redeem these tokens on other SPs and thus get unauthorized access on different accounts.

This attack is feasible since an IdP is used by multiple SPs and issues tokens for all of them. Every issued SSO token must only be consumed by a specified SP. Therefore, the SP must verify the **Recipient** information (Class II). The Wrong Recipient attack belongs to Cat $\mathcal{A}$.

In OpenID Connect the parameter specifying the recipient of the token is the *aud* parameter. It contains the unique identifier of the SP (`client_id`). Once the SP receives an ID Token (`id_token`) it must verify that the *aud* parameter corresponds to its own `client_id` to counter the Wrong Recipient attack.

The concept of this attack is well-known and has already been applied to SAML [13, 18, `Audience/ Recipient` element], to OpenID [19, `return_to` parameter], and to OAuth [6, 32, `redirect_uri` parameter].

## 5.3. Class III Attack: Replay

Replay attacks circumvent the one-time use requirements and the time restrictions of an SSO token. In the most devastating scenario an End-User, for example, a former employee, has access to an SP for infinite amount of time.

Replay attacks are possible if **Freshness** parameters (Class III) are not checked properly and belong to Cat $\mathcal{A}$. Such limitation can be based on timestamps defining a time-slot for the validity of the SSO token. A more restrictive option of single use are *nonces*. In OpenID Connect the relevant parameters are *iat* (issued at), *exp* (expired) and *nonce*.

Replay attacks have been applied to various SSO protocols, for example, OpenID [33], Facebook Connect [41] and SAML [18].

## 5.4. Class IV Attack: Signature Bypass

SSO protocols use cryptographic operations to protect the integrity of the SSO token or at least parts of it.[3]

Signature Bypass attacks evade this integrity protection and enable the modification of any content within the SSO token. Generally spoken, if a Signature Bypass is possible *all* previous described Single-Phase Attacks are applicable since (1.) any identity of an End-User can be inserted, (2.) any audience can be stated, (3.) all timestamps and nonces can be adjusted.

The signature value itself, but also all relevant information to verify the signature[4] belong to the **Signature** information data (Class IV). Signature Bypasses are Cat $\mathcal{B}$ attacks and target these parameters.

Basically, there are three different types of Signature Bypass attacks: (1.) Disabling signature verification by removing all signature information completely [29]. (2.) Enforcing the usage of wrong keys. This was described on SAML [18] and applied on OpenID [19]. (3.) Changing content without invalidating the signature. A typical example for this is XML Signature Wrapping (XSW) [29, 18], but other scenarios have been successfully applied on Facebook Connect [41] and OpenID [37].

An SSO token in OpenID Connect has several parameters belonging to Class IV: The signature value itself is transferred in the JSON Web Token (JWT). The JWT header contains a parameter *alg*. By setting its value to *none*, the signature verification could be disabled [20]. According to the OpenID Connect specification, *none* is not allowed if the token is transferred via the browser of the End-User, but multiple implementations missed this check. The JWT header can optionally contain further parameters such as the key id (*kid*), which is used to identify the key to be used for the signature. If an attacker can point to its own key, the wrong (untrusted) key is used and he can sign arbitrary tokens containing arbitrary identities.

To prevent such attacks, an SP must (1.) blacklist all weak, broken, and thus insecure cryptographic algorithms, for example, the *none* algorithm. (2.) ensure that the correct key material is loaded. This can be done by verifying the signature with the key material corresponding to the `iss` in the `id_token`. (3.) verify that all parameters used for the authentication within the SSO token are protected by the signature.

## 5.5. Q1: Does the specification address the existing threats?

The short answer to this question is *yes*. Single-Phase Attacks are well studied and our investigation revealed that the OpenID Connect specification addresses all attacks described in this section. Additionally, it presents examples to clarify the verification steps, which have to be implemented.

3. E.g. in SAML. The XML-Signature protects the `Assertion` element in most cases, but not the XML root element `Response`.

4. E.g., the used algorithm, certificate information, etc.

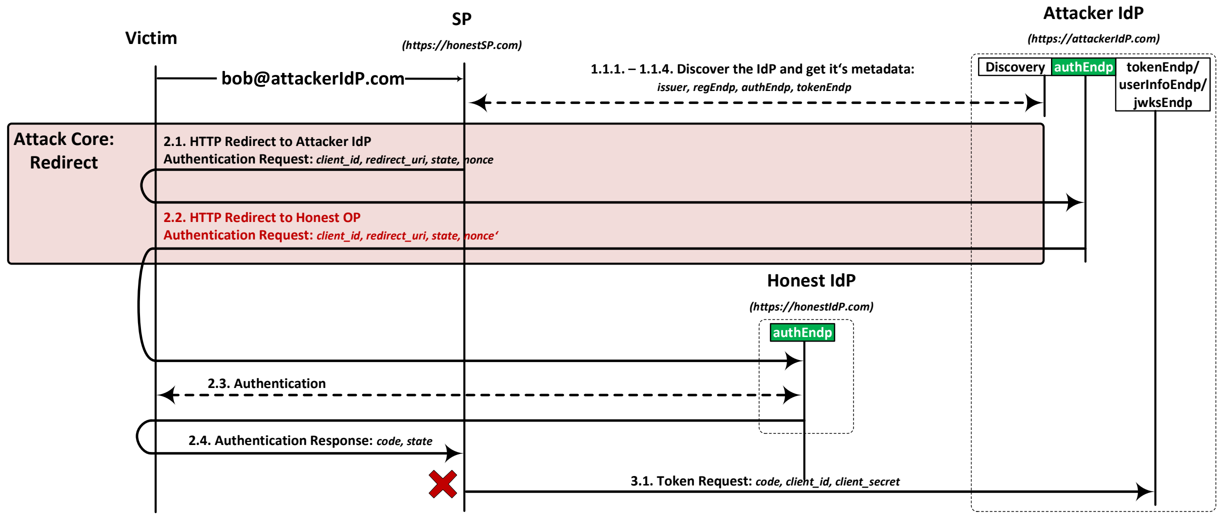

Victim
SP
(https://honestSP.com)
Attacker IdP
(https://attackerIdP.com)
Discovery
authEndp
tokenEndp/
userInfoEndp/
jwksEndp
bob@attackerIdP.com
1.1.1. – 1.1.4. Discover the IdP and get it's metadata:
issuer, regEndp, authEndp, tokenEndp
Attack Core:
Redirect
2.1. HTTP Redirect to Attacker IdP
Authentication Request: client_id, redirect_uri, state, nonce
2.2. HTTP Redirect to Honest OP
Authentication Request: client_id, redirect_uri, state, nonce'
Honest IdP
(https://honestIdP.com)
authEndp
2.3. Authentication
2.4. Authentication Response: code, state
3.1. Token Request: code, client_id, client_secret

Figure 5: IdP Confusion Cross-Phase Attack: Logical flaw in the OpenID Connect specification.

Unfortunately, our evaluation revealed multiple gaps in the existing implementations ignoring all or many of the required verification steps leading to *broken authentication*. The main reason for ignoring security relevant checks seems to be that functionality (features) comes before security. Thus, developers concentrate on working implementations and often forget to implement all needed verification steps. Developers are also not aware how critical one check could be and what its impact is regarding the security of the system if this verification step is skipped.

## 6. Cross-Phase Attacks

A Single-Phase Attack can be conducted by only manipulating at most one message in one phase of the SSO protocol. In contrast to this, Cross-Phase Attacks manipulate multiple messages in different phases. This concept introduces more complex attacks, which are barely studied.

In general, Cross-Phase Attacks abuse the lack of a binding between two or more protocol phases. By this means, the attacker can skip or bypass an important verification step leading to broken End-User authentication.

While the concept of Cross-Phase Attacks can be applied to all SSO protocols (and already was, e.g., to OpenID in [19]), we only concentrate on OpenID Connect in this section, because this kind of attacks highly depend on the protocol structure and the according messages.

In this section, we first show how the Discovery phase influences the OpenID Connect protocol flow. Based on this knowledge we developed two attacks – *IdP Confusion* and *Malicious Endpoints* – abusing a flaw in the current OpenID Connect specification. Our third attack – *Issuer Confusion* – makes use of an implementation flaw. As a result of these Cross-Phase Attacks, the attacker is logged in on an SP in the victim's account.

**Discovery phase in OpenID Connect.** The attacks described in this section depend on the Discovery phase. Thus, we briefly introduce the relevant messages. Figure 6

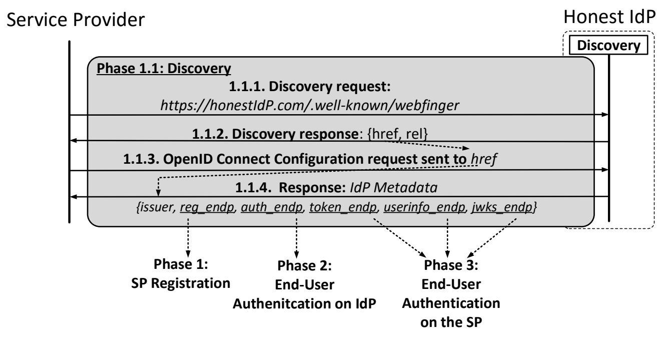

Service Provider
Honest IdP
Discovery
Phase 1.1: Discovery
1.1.1. Discovery request:
https://honestIdP.com/.well-known/webfinger
1.1.2. Discovery response: {href, rel}
1.1.3. OpenID Connect Configuration request sent to href
1.1.4. Response: IdP Metadata
{issuer, reg_endp, auth_endp, token_endp, userinfo_endp, jwks_endp}
Phase 1:
SP Registration
Phase 2:
End-User
Authenitcation on IdP
Phase 3:
End-User
Authentication
on the SP

Figure 6: The OpenID Connect Discovery phase influences all other phases.

shows how the information retrieved during the Discovery phase influences the OpenID Connect phases. We here do not go into detail, but for this paper, message 1.1.4 is most important. This message returns different so-called endpoints. Each endpoint is a URL called by the SP during the different phases. For example, the Dynamic Registration Endpoint (`regEndp`) defines where an SP starts the Dynamic Registration. 75% of the evaluated libraries support this feature.

### 6.1. Specification Flaw: IdP Confusion

IdP Confusion is a novel Cat $\mathcal{A}$ attack on OpenID Connect, which abuses a lack in the current specification: the connection between Phase 2 (the End-User authentication on the IdP) and Phase 3 (the redemption of the received *code*) is missing. More precisely, the SP receives a `code` from the victim by the end of Phase 2 but, due to the flaw in the specification, it is unable to determine to which IdP the `code` belongs to. Because the attacker has modified an important step in Phase 2, the SP sends wrongly the `code` (plus `client_id` and `client_secret`) to the attacker IdP, which can use it for its own purposes.

**Requirements.** We assume that the SP allows the usage of custom IdPs. Additionally, we assume that during the registration the SP receives the same *client_id* from the attacker IdP as on the honest IdP. In other words, the SP has the same *client_id* on two different IdPs, which is allowed according to the specification.

**Execution.** Figure 5 shows the attack:

- ▶ In the first step the victim clicks on a malicious link or visits an attacker controlled website. This click automatically manages to start a login attempt on the SP with *bob@attackerIdP.com*.
- ▶ If the SP supports Discovery, metadata is retrieved.
- ▶ In Step 2.1 the SP redirects the End-User to the authorization endpoint of the attacker IdP.
- ▶ In Step 2.2 the attacker IdP redirects the End-User to the honest IdP. Additionally, it replaces the *nonce* parameter. This manipulation is necessary to successfully impersonate the victim on the SP.

Please note that all steps until now do not require any interaction of the End-User and are transparent for him. Thus, he is not able to detect the attack.

- ▶ In Step 2.3 the End-User must authenticate to the IdP. In case that he is already authenticated, this step will be skipped. This step is the only one, in which it is possible for the victim to detect the attack, for instance, it might seem suspicious for the victim to get an authentication pop-up. If the user is already authenticated on the honest IdP, this step is usually transparent for the End-User.
- ▶ The IdP generates a valid *code* and returns it together with the *state* parameter back to the SP.
- ▶ The SP still believes that it is communicating with the attacker IdP due to Step 1.2. For this reason, it redeems the received *code* on the attacker IdP and additionally sends its *client_id* and *client_secret*.
- ▶ As a result, the attacker has a valid *code*, which he can then redeem through his browser on the SP.

The attacker now starts his authentication on the SP and sends the stolen *code* to it. The SP redeems the *code* on the honest IdP and logs the attacker into the victim's account. We describe the countermeasure for this attack in Section 6.4.

## 6.2. Malicious Endpoints Attacks

The Malicious Endpoints Attack is a novel attack on OpenID Connect, which abuses a lack in the current specification: the connection between Phase 1 (the Discovery) and Phase 3 (the redemption of the received *code*) is missing. Similar to IdP Confusion the idea is to *confuse* and enforce the SP to send a valid *code* together with the SPs *client_id* and *client_secret* to a URL controlled by the attacker. More precisely, the SP downloads in Phase 1 maliciously crafted metadata from the attacker IdP, enforcing the SP to use the honest IdP during Phase 2 for the End-User authentication, but to redeem the rceived tokens on the attacker IdP in Phase 3.

In addition, we extended Malicious Endpoints Attack to Server Side Request Forgery (SSRF) and Denial-of-Service (DoS) attacks.

### 6.2.1. Broken End-User Authentication.

This Cat $\mathcal{A}$ attack manipulates the information in the Discovery and Dynamic Registration Phases in such a way that the attacker gains access to sensitive information. The attacker (1.) pursues the theft of the credentials between the honest IdP and the honest SP and (2.) steals a valid `code` authorizing the SP to access End-User's resources on the honest IdP.

**Requirements.** We assume that the End-User (*victim*) has an active account on the SP and he additionally has an account on the honest IdP. In addition, the SP supports the Discovery.

**Execution.** In the following, we describe the attack protocol flow, which we depicted in Figure 7.

*Phase 1.1 - Injecting malicious endpoints* Similarly to the IdP Confusion attack, the victim is lured to login with *bob@attackerIdP.com* on the SP Consequentially, the SP starts a discovery phase with the attacker IdP, which responds with the values shown in Listing 1 to initiate the actual attack. This differs to the IdP Confusion attack, where the attacker IdP does not manipulate this message.

```
issuer:        https://attackerIdP.com
regEndp:       https://honestIdP.com/register
authEndp:      https://login.honestIdP.com/
tokenEndp:     https://attackerIdP.com
userInfoEndp:  https://attackerIdP.com
```

Listing 1: Endpoints returned by the attacker IdP

*Phase 1.2 – Dynamic Registration.* In the next step, the SP accesses `regEndp` for the Dynamic Registration. It sends a registration request to https://honestIdP.com/register and receives a `client_id` and `client_secret`.

*Note:* The SP automatically starts the Dynamic Registration, even if it is already registered on the honest IdP. The reason for this behavior is that the SP believes that http://attackerIdP.com is the responsible IdP, since it is not known from previous authentication procedures. Thus, http://attackerIdP.com is a new IdP for the SP and it starts the registration procedure.

*Phase 2 – End-User Authentication and Authorization.* In the next phase, the SP redirects the End-User to Authorization Endpoint (`authEndp`), https://login.honestIdP.com/, where the End-User must authenticate himself and authorize the SP. The End-User is not able to detect any abnormalities in the protocol flow: Phase 1.1 and Phase 1.2 cannot be observed by the End-User, and in Phase 2 the End-User will be prompted to authenticate to the honest IdP and authorize the honest SP (he trusts both). Thus, the End-User authorizes the SP and the IdP generates the `code`, which is sent to the SP.

*Note:* Phase 2 exactly follows the original OpenID Connect protocol flow – there are no parameter manipulations, no redirects to malicious websites and no observation of the network traffic between the End-User, the honest IdP and the SP. Thus, the attack started at the beginning of the

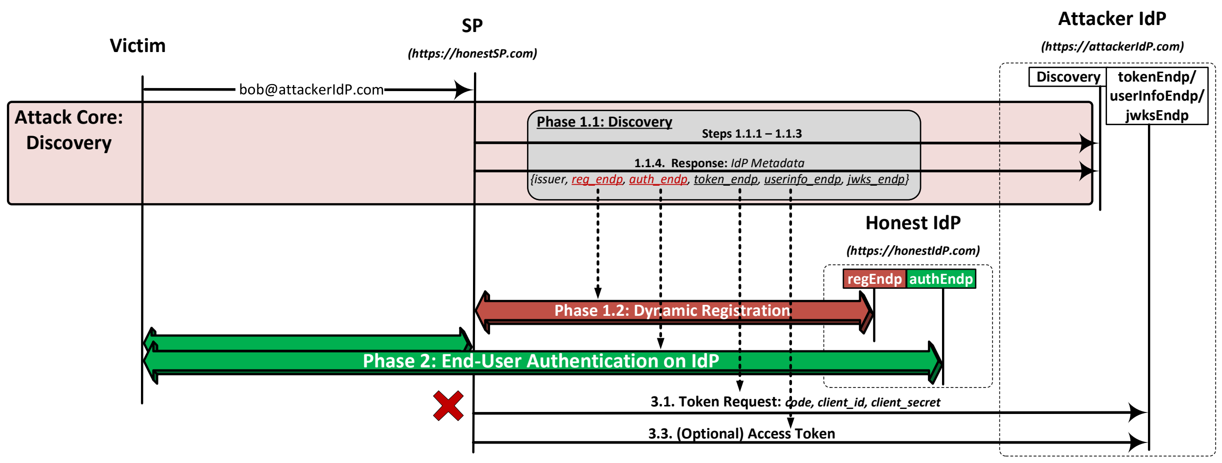

Victim
SP
(https://honestSP.com)
Attacker IdP
(https://attackerIdP.com)
Discovery
tokenEndp/
userInfoEndp/
jwksEndp
bob@attackerIdP.com
Attack Core: Discovery
Phase 1.1: Discovery
Steps 1.1.1 – 1.1.3
1.1.4. Response: IdP Metadata
{issuer, reg_endp, auth_endp, token_endp, userinfo_endp, jwks_endp}
Honest IdP
(https://honestIdP.com)
regEndp
authEndp
Phase 1.2: Dynamic Registration
Phase 2: End-User Authentication on IdP
3.1. Token Request: code, client_id, client_secret
3.3. (Optional) Access Token

Figure 7: Malicious Endpoints attack: by manipulating the Discovery, the attacker steals the `code`.

protocol flow can be neither detected nor prevented by any of the participants at this point.

*Phase 3 – The Theft.* Now, the SP redeems the received `code` from the previous phase: It sends the `code` together with the corresponding SP's credentials received during the Dynamic Registration (`client_id`/`client_secret`) to the *real* Token Endpoint (`tokenEndp`).

**Attack Summary.** The attacker receives a valid `code` and `client_id`/`client_secret`. Thus, he is authorized to access End-User's resources on the IdP, retrieve a valid `id_token`, and can impersonate the SP.

**6.2.2. Server Side Request Forgery.** A Server Side Request Forgery (SSRF) attack describes the ability of an attacker to create requests from a vulnerable web application to the application's Intranet and the Internet. Usually, SSRF is used to attack internal services placed behind a firewall and not accessible from Internet.

SSRF attacks on OpenID Connect can be conducted by configuring URLs pointing to other services in a local area network (LAN, e.g. http://192.168.0.1/shutdown) in the Discovery information (cf. Listing 1).

**6.2.3. Denial-of-Service Attacks.** By applying Denial-of-Service (DoS) attacks the attacker allocates resources on a SP and negatively affects its workflow. Such resources are CPU usage, network traffic or memory. The attack can target one or multiple of these resources during the execution of DoS attack.

DoS attacks in OpenID Connect can be easily executed if the attacker IdP uses endpoints containing large files (e.g., public Linux DVD images or video files). The SP then starts a GET request on these endpoints and downloads the file, which takes lot of time and consumes memory

## 6.3. Malicious Endpoints Attack, IdP Confusion, and IdP Mix-Up

On a quick peek, IdP Confusion and IdP Mix-Up look very similar, but the following differences must be mentioned:

- ▶ The IdP confusion attacker works in the web attacker model. The initial version of the IdP Mix-Up requires a network attacker [11, v2], the SP and the IdP. Consequentially and after reporting both attacks, the authors of the IdP Mix-Up attack optimized their attack to the web attacker model [11, v3].
- ▶ The main goal of the IdP Mix-Up is to compromise the OAuth flow by revealing a valid `code` or `access_token`. These can be used to get unauthorized access to restricted resources.
  The main goal of the IdP Confusion attack is the impersonation of the victim on the SP. It requires the manipulation of more parameters due to the validation of the `id_token`, for example, `state`, `nonce` and `client_id`.
- ▶ IdP Mix-Up is not able to get a valid access token in code flow, but the IdP Confusion does, [11, Page 35].

## 6.4. Fixing the OpenID Connect Specification

After discovering the IdP Confusion and Malicious Endpoints Attacks, we promptly contacted the *OpenID Connect working group* in October 2014. Unfortunately, they did not respond and even a second and third attempt failed. In November 2015, we were surprisingly contacted by the *IETF OAuth* working group, because our attacks could be applied to the OAuth protocol as well (which was a parallel work of a different research group [11]).

**Result: Specification Change.** We were invited to a special security meeting in which we discussed different mitigation techniques and helped them to create an update for the OAuth as well as the OpenID Connect specification [17].

In general, the reason for the authentication issues is the missing connection across all phases leading to confusion on the SP side. To mitigate the attacks a parameter linking the phases was added. In Phase 2, by adding the *issuer* parameter in addition to the *code* parameter (see Step 2.4 in Figure 5) the SP knows to which IdP the code must be sent in Phase 3.

Summarized, the *issuer* parameter must be provided in (1.) Phase 1 during the Discovery, (2.) Phase 2 as a mitigation against the reported Cross-Phase Attacks, and (3.) Phase 3 within the `id_token`. Thus, the Cross-Phase Attack can be detected and the leackage of valid SSO tokens can be prevented on the SP side.

## 6.5. Implementation Flaw: Issuer Confusion

*Issuer Confusion* is a Cat $\mathcal{B}$ attack and depicted in Figure 8. The idea of *Issuer Confusion* is to *confuse* the SP to successfully accept an `id_token` issued by the attacker IdP, while believing that it was issued by an honest IdP. The main idea is related to the IDS attack – an `id_token` issued and signed by attacker IdP in the name of the honest IdP is successfully verified on the SP.

But, in comparison to IDS the Issuer Confusion abuses a missing verification step in Phase 1 to break the entire security even if the verification of the SSO token in Phase 3 is implemented correctly. In Section 5.1, we showed the IDS attack abusing the skipped verification of the *iss* parameter. Consequentially, the question arises how this verification step can be implemented correctly. In other words, how can the SP verify that the *iss* within the SSO token really belongs to the IdP that has generated the token. According to the specification: "The Issuer Identifier for the OpenID Provider (which is typically obtained during Discovery) MUST exactly match the value of the iss (issuer) Claim." [34]. This means, the correct verification of the SSO token in Phase 3 depends on parameters exchanged in Phase 1.

**Requirements.** The attack assumes an SP allowing the usage of custom IdPs: the attacker IdP.

**Execution.** The attack proceeds as depicted in Figure 8:

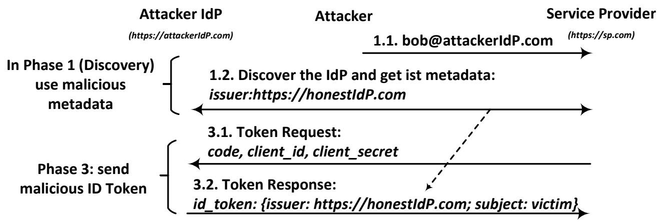

Attacker IdP
(https://attackerIdP.com)
Attacker
Service Provider
(https://sp.com)
1.1. bob@attackerIdP.com
In Phase 1 (Discovery) use malicious metadata
1.2. Discover the IdP and get ist metadata:
issuer:https://honestIdP.com
3.1. Token Request:
code, client_id, client_secret
Phase 3: send malicious ID Token
3.2. Token Response:
id_token: {issuer: https://honestIdP.com; subject: victim}

Figure 8: Issuer Confusion Cross-Phase Attack.

- ► The attacker starts the login on the SP.
- ► In Step 1.2 the SP discovers the IdP – in our example https://attackerIdP.com – and retrieves the identity of the IdP, public keys and important URLs. Here, the attacker IdP returns the parameter `issuer`*: https://honestIdP.com* (instead of https://attackerIdP.com).
- ► Phase 2 is not depicted in Figure 8 since no manipulations or any attack takes place here.
- ► In Phase 3, Step 3.2, the SP receives an SSO token (called `id_token` in OpenID Connect) and verifies it. The `id_token` contains once more the value *issuer: https://honestIdP.com*, which matches the value of Step 1.2. If the validation is successful, the attacker gets access to the victim's account on the SP.

In summary, the attacker has to manipulate the *issuer* parameter in two different protocol phases – we have a Cross-Phase Attack. This attack abuses an implementation flaw: the SP must verify that the returned `issuer` parameter matches to the IdP. To fix it, the SP must verify that the `issuer` returned in Step 1.2 matches the called URL (according to the specification).

A similar attack was applied to OpenID [19] by sending manipulated XRDS/HTML discovery files.

## 6.6. Q1 - Does the Specification Address the Existing Threats?

In comparison to Single-Phase Attacks, Q1 cannot be answered clearly for Cross-Phase Attacks. The specification addresses the *Issuer Confusion* attack, but all necessary verification steps are distributed over the whole specification and not documented at one central place. It is thus hard for a developer to see the relation between all verification steps.

Another issue is that Cross-Phase Attacks are barely studied and could up to now only be found in, for example, OpenID [19] and OAuth [11].

## 7. PrOfESSOS: Analyzing OpenID Connect with Attacker IdPs

To the best of our knowledge, PrOfESSOS is the first automated EaaS security tool for practically analyzing OpenID Connect implementations. It is the first tool capable to evaluate the high complex Cross-Phase Attacks, instead of only detecting, for example, simple Replay attacks (Single-Phase Attacks). In this section we describe the main challenges implementing PrOfESSOS. We then describe its design, architecture, and automated workflow.

### 7.1. Challenges in Analyzing OpenID Connect

Analyzing OpenID Connect by using an attacker IdP has many novel challenges to be addressed in comparison to previous work.

**Complexity.** OpenID Connect is by far more complex than, for example, OpenID and BrowserId. This is reasoned by the following properties:

(1.) OpenID Connect supports different *protocol flows* leading to significant differences in the messages exchanged between the entities. In OpenID Connect, there are three different main flows: *code*, *implicit* and *hybrid*. All flows expect different parameters and messages. This increases the necessarily depth of analysis. In the implicit flow, the `id_token` is validated by the End-User itself. The SP therefore sends, for example, JavaScript code to the End-User's browser. In the other flows, the `id_token` is validated on the SP using server code (PHP, Java, . . . ). Thus, there are different implementations on a single SP, which have to be separately validated.

(2.) OpenID Connect defines different SP types: web applications, mobile apps and native applications. Each category requires different flows and messages. Even more, for each SP capability different security considerations have to be made. For example, by using OpenID Connect on a web application an attacker cannot see the entire communication: the communication between the web application and the IdP is hidden. In contrast to that, the attacker has full access to a mobile device: the communication between an installed app and the IdP can be easily observed and even manipulated, for example, by using an HTTP-Proxy.[5] In OpenID and BrowserId, only web applications are supported and evaluated with respect to the security.

(3.) OpenID Connect defines a more powerful Discovery phase which influences all other phases, especially the token validation phase. A Discovery is also supported in OpenID and BrowserId, but therein, it simply defines the URL of the IdP. This reduces the potential for attacks.

**Back-channel Communication.** In the code and hybrid flow, the SP and the IdP exchange multiple messages including the SP authentication on the IdP and the `id_token` and `access_token`. In other words, multiple security related messages will be sent. A huge restriction for an attacker is that it is not possible to observe or manipulate this communication. Thus, he is not able to access the `id_token` and manipulate it, which automatically limits the attack surface. To bypass this restriction, we found novel attacks (IdP Confusion, Malicious Endpoints) leaking the SP credentials and a valid token to an attacker. In comparison to other SSO protocols there is no such back-channel communication and thus such an analysis does not exist.

**Using Attacker IdPs.** Some SPs provide authentication possibilities only with a limited range of IdPs, for example Google, Facebook, Twitter or PayPal. Thus, a security evaluation using an attacker IdP is possible if the administrator manually configures the attacker IdP. The application of an attacker IdP can be additionally enforced in some scenarios by the HTTPoxy [7] vulnerability, or by setting specific HTTP GET parameters in a request (e.g., changing `idp=google.com` parameter). In OpenID and BrowserId this problem is less complex since the Discovery and Dynamic Registration are part of the core specification. Thus, by implementing these protocols, the SP automatically supports the usage of attacker IdPs.

## 7.2. Implementation Challenges

In addition to general, protocol related challenges, we had to cover different implementation challenges. The first challenge is to cover all flows and all SPs like web, mobile and native applications. To be platform independent, we decided to implement PrOfESSOS as a web service. Thus, no installation is required and it easily allows continues integration. The second challenge is to establish a flexible meta language for configuring the security tests. The flexibility is needed since each attack differs from others by the following properties:

**Success Condition.** Attacks can have different goals. For example, the win condition in a Replay attack is to be successfully logged in. For IDS, this is not enough: it is only successful if the attacker is logged in with a specific, that means the victim's, identity.

**Flows.** Some attacks are only applicable to specific OpenID Connect flows. This problem does not exist in OpenID and BrowserId where only one flow exists.

**Login.** PrOfESSOS must be able to login on the SP. It therefore has to find the input form, which is complicated to detect [41]. Because of this, we first try to do so automatically. If it fails, the user can provide a Selenium script.[6]

## 7.3. Architecture

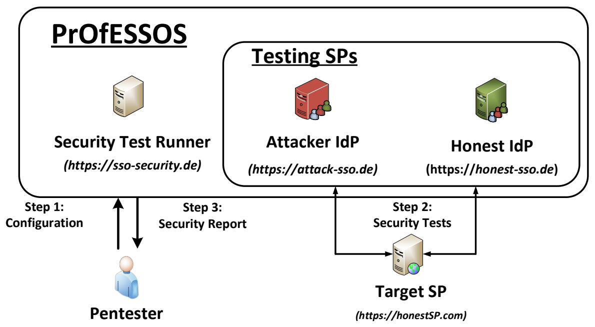


Figure 9: Test scenario when using PrOfESSOS.

A user of PrOfESSOS – the penetration tester – can access it via its web interface.[7] The penetration tester there enters the URL of the target SP and selects the to be tested attacks. In Figure 9 we depict the architecture and components of PrOfESSOS.

**Security Test Runner (STR).** PrOfESSOS is an EaaS. As such, it is available for multiple penetration testers simultaneously. The Security Test Runner (STR) component is basically the executing processor of PrOfESSOS so that multiple, parallel, running tests do not interfere with each other. The STR navigates the attacker IdP during testing, gets the HTTP and HTML results back, and evaluates the attack's success condition.

**Attacker IdP.** The attacker IdP is a highly configurable IdP capable to act honestly and maliciously. The IdP gets information from the STR about the target SP, for example, important URLs needed to start the authentication flow (e.g. login page URL). It needs information to verify whether the login attempt was successful.

**Honest IdP.** Using only one IdP for testing implementations is quite limited. During some attacks, the attacker IdP confuses the SP by making it to believe that it communicates with the honest IdP (mainly for Cross-Phase Attacks). Unfortunately, the attacker IdP *cannot* observe this communication. Thus, it is not possible to analyze the

5. Certificate pinning can be used for protection, but it can also be circumvented: https://eaton-works.com/2016/07/31/reverse-engineering-and-removing-pokemon-gos-certificate-pinning/

6. http://www.seleniumhq.org/
7. Demo: https://openid.sso-security.de.

| Attacks | Targeted Flows | Discovery Required | Modification by the Attacker IdP | | Success Condition(s) |
|---|---|---|---|---|---|
| | | | Phase | Message | |
| IDS (v1) | code, hybrid, implicit | – | `Phase 3` | `id_token.sub = honestIdP.sub` | (1) Login Successful?<br>(2) User = honestIdP.sub? |
| IDS (v2) | code, hybrid, implicit | – | `Phase 3` | `id_token.iss = honestIdP.iss`<br>`id_token.sub = honestIdP.sub` | (1) Login Successful?<br>(2) User = honestIdP.sub? |
| Wrong Recipient | code, hybrid, implicit | – | `Phase 3` | `id_token.aud = honestIdP.aud` | (1) Login Successful? |
| Replay (v1) | code, hybrid, implicit | – | `Phase 3` | `id_token.nbf = 01.01.2070` | (1) Login Successful? |
| Replay (v2) | code, hybrid, implicit | – | `Phase 3` | `id_token.exp = 01.01.1970` | (1) Login Successful? |
| Replay (v3) | code, hybrid, implicit | – | `Phase 3` | `id_token.nonce = old.nonce` | (1) Login Successful? |
| Signature Bypass (v1) | code, hybrid, implicit | – | `Phase 3` | `id_token.alg = "none"` | (1) Login Successful? |
| Signature Bypass (v2) | code, hybrid, implicit | – | `Phase 3` | `id_token.signature = "Invalid Signature"` | (1) Login Successful? |
| Issuer Confusion | code, hybrid, implicit | yes | `Phase 1`<br>`Phase 3` | `discovery.issuer = honestIdP.iss`<br>`id_token.sub = honestIdP.sub`<br>`id_token.iss = honestIdP.iss` | (1) Login Successful?<br>(2) User = honestIdP.sub? |
| Spec. Flaw: IdP Confusion | code, hybrid | – | `Phase 2` | `authRequest.nonce = nonce'`<br>`authRequest.redirectURL = honestIdP.authEndp` | (1) Attacker IdP receives `code, client_id, client_secret`? |
| Spec. Flaw: Malicious Endpoints | code, hybrid | yes | `Phase 1` | `discovery.regEndp = honestIdP.regEndp`<br>`discovery.authEndp = honestIdP.authEndp` | (1) Attacker IdP receives `code, client_id, client_secret`? |

TABLE 1: Configuration summary of Single-Phase and Cross-Phase Attacks supported by PrOfESSOS.

*exact* behavior of the SP during the attacks and evaluate the results. To solve this limitation, PrOfESSOS introduces a second honest IdP which does not perform any attacks. Instead, it is only used to simulate the victim's IdP and to observe the communication between the SP and any honest IdP.

## 7.4. Automated Analysis Workflow

PrOfESSOS evaluates the target SP in three stages: setup, configuration evaluation and attacks.

**Stage 1: Setup.** The penetration tester starts the SSO evaluation by visiting the homepage of PrOfESSOS. In the background, the STR creates one attacker IdP and another honest IdP instance on the fly (cf. Figure 9).

**Stage 2: Configuration Evaluation.** The penetration tester enters the URL of the SP that he wants to test. PrOfESSOS then fetches the HTML document at the URL and tries to detect the login form. If it can automatically detect the form, it enters the URL of the attacker IdP to start the authentication. Otherwise, the penetration tester must provide additional information, so that PrOfESSOS is able to start a login attempt automatically.

Once PrOfESSOS is able to login successfully to the SP, the next challenge has to be solved: PrOfESSOS must detect the name of the currently logged in user. In some cases, the username is simply printed on the website. On more complex SPs, the penetration tester can provide a link to a URL where this information can be found. PrOfESSOS needs this kind of information for attacks related to the *Identity* information (Class I) of the SSO token. If an attack sends, for example, to different usernames in one token, PrOfESSOS must decide which value is used.

Finally, PrOfESSOS starts a new login attempt, but this time, the attacker IdP creates an *invalid* token (containing a wrong signature). By this means, it tests whether a false login state can be distinguished from a correct login. If this test fails, PrOfESSOS aborts the SP evaluation.

**Stage 3: Attacks.** In this stage, the penetration tester selects different attacks that will be executed against the target SP. In Table 1 we summarize the configuration of *all* attacks described in this paper.

PrOfESSOS uses two IdPs – an honest IdP and an attacker IdP. The honest IdP strictly follows the OpenID Connect specification. It is used for monitoring the HTTP traffic and does not apply any protocol or message manipulation. On the other side, the attacker IdP can differ from the OpenID Connect specification. Once an attack is started, the attacker IdP takes the attack configuration and manipulates the specified messages. All other messages are untouched. The attack configuration thus only specifies the manipulations deviating from the specified protocol flow. The configuration contains additional information about the applicable protocol flows, required OpenID Connect features (e.g., whether the attack requires Discovery), and success conditions. For example, an IDS attack (see Table 1, row 1), can be executed on all OpenID Connect flows (code, hybrid and implicit). During the attack execution, the attacker IdP acts according to the OpenID Connect specification until Phase 3. Therein, the attacker IdP must manipulate the `id_token` by setting the *sub* value to the victim's identity. More precisely, the attacker IdP first creates a valid `id_token`, then it replaces the *sub* value, and finally signs the `id_token`. Once the `id_token` is sent, PrOfESSOS observes the reaction of the SP and the according results. In case of IDS, PrOfESSOS validates the success conditions (1.) whether the login was successful and (2.) if the name of the logged in user corresponds to the *victim's* identity (belonging to the honest IdP). Table 1 also depicts the more complex Cross-Phase Attacks, such as Issuer Confusion, by specifying manipulations in the Discovery and in the `id_token`. Further attacks can be added by creating a new configuration, which allows to potentially manipulate every message or add new ones according to the attack's requirements.[8]

## 7.5. Limitations

PrOfESSOS currently supports only OpenID Connect. Some tests can be started on OAuth SPs due to the familiarities of both protocols. Protocols such as OpenID and SAML are not yet implemented. We consciously decided to implement the *manual* configuration of PrOfESSOS (see Stage 1). The reason for our decision are the existing challenges documented by Zhou et al. [41]: automatically finding the login button, captchas, invisible identity, atypical HTML elements etc. Such problems make automated configuration and analysis very hard, error-prone, and in case of complex attacks like Cross-Phase Attacks, they can lead to false results. The configuration overhead by filling out six text fields is considered feasible for a penetration tester and developer. PrOfESSOS was not designed for large scale studies, crawling Alexa Top websites and finding security issues. The main goal was to improve the security of SSO by continuously supporting developers during the implementation of new libraries.

## 8. Evaluation

We selected all 12 open source libraries officially referenced on the OpenID Connect website [25].[9] In addition, 11 commercial products like Amazon and MS Azure and 18 tools supporting JSON-based operations (e.g. encryption and signing) are referenced, which we did not consider in our evaluation.

**Setup.** We documented the supported features of the SPs. Such features are: (1.) flows (code, implicit, hybrid), (2.) cryptographic algorithms, (3.) additional features such as Discovery and Dynamic Registration. 4 of 12 libraries do not support a full SP functionality. Such libraries are used as a foundation to implement an SP. Features like an authenticated session after successful login and End-User profile page are not implemented. Without adding this part, it is not possible to verify if an attack is successful or not. Thus, we did not further consider these libraries.

We concentrated on the remaining 8. These libraries support the usage of attacker IdP which allows us to use PrOfESSOS for the security evaluation. Only 2 out of 8 libraries did not support Discovery and Dynamic Registration. Because of that, we manually configured them to use PrOfESSOS. For the remaining 6, no pre-configuration and no manual steps are required.

**Amazon.** PrOfESSOS can also be applied to live websites. We choose Amazon Webservices for our proof-of-concept since Amazon supports the usage of attacker IdP. PrOfESSOS can be started by simply configuring its URL in the configuration area. Thus, we were able to evaluate all known attacks. Amazon is not susceptible against any of them. Interestingly, Amazon already implemented a countermeasure against the IdP Confusion attack similar to the countermeasure proposed by IETF [17].

**Q2: How secure are reference implementations? (Part 2/2).** The results of our security evaluation are presented in Table 2. 75% of the libraries were susceptible to at least one critical implementation issue[10], for instance, one of the Single-Phase Attacks or the Issuer Confusion attack.

Even though Single-Phase Attacks are well-known, security relevant parameters were ignored and not verified at all. We reported all issues to the developers of the libraries and supported them during fixing the vulnerabilities. Even though many of the vulnerabilities exist due to skipped checks, reporting and fixing the issues was a huge outlay since we had to exactly explain the security issues and the impact of the attacks. This showed us that many developers are not aware of the risks by skipping one security check, especially in the complex Cross-Phase Attacks.

In case of attacks abusing lacks in the specification like IdP confusion, *all* implementations are vulnerable. This is an expected result since even a correct implementation strictly following the specification rules is still susceptible.

**Automated Analysis with PrOfESSOS.** Simultaneously to our initial manual analysis, we developed PrOfESSOS and confirmed the results of our evaluation and to verify if the reported issues were fixed by the developers. Once configured, the re-evaluation with PrOfESSOS requires only few minutes.

## 9. Related Work

We separated existing research into two categories.

**SSO protocol security.** `SAML`: Groß [13], Groß and Pfitzmann [14] and Armando et al. [1] analyzed a formal model for the SAML Browser/Artifact profile and identified several generic flaws allowing connection hijacking/replay, Man-in-the-Middle (MitM) and HTTP referrer attacks. In 2012, Somorovsky et al. [29] investigated the XML Signature

8. https://github.com/RUB-NDS/PrOfESSOS/blob/master/src/main/resources/testplan/rp_test.xml

9. Note that the referenced list is growing rapidly.

10. Beyond that, all the libraries were vulnerable to the specification issues.

| SPs Libraries / Attack Category | Custom IdP | Dynamic Trust | Single-Phase Attacks: IDS Cat $\mathcal{B}$ | Wrong Recipient Cat $\mathcal{A}$ | Replay Cat $\mathcal{A}$ | Signature Bypass Cat $\mathcal{B}$ | Cross-Phase Attacks: Issuer Conf. Cat $\mathcal{B}$ | Specification Flaws Cat $\mathcal{A}$ |
|---|---|---|---|---|---|---|---|---|
| mod_auth_openidc | **Yes** | **Yes** | ✓ | ✓ | ✓ | **Vuln.** | ✓ | **Vuln.** |
| MITREid Connect | **Yes** | **Yes** | ✓ | ✓ | ✓ | ✓ | ✓ | **Vuln.** |
| oidc-client | **Yes** | **Yes** | **Vuln.** | **Vuln.** | **Vuln.** | ✓ | **Vuln.** | **Vuln.** |
| phpOIDC | **Yes** | **Yes** | **Vuln.** | **Vuln.** | **Vuln.** | **Vuln.** | **Vuln.** | **Vuln.** |
| DrupalOpenIDConnectd | **Yes** | *No* | **Vuln.** | **Vuln.** | **Vuln.** | **Vuln.** | **Vuln.** | **Vuln.** |
| pyoidc | **Yes** | **Yes** | **Vuln.** | **Vuln.** | **Vuln.** | **Vuln.** | ✓ | **Vuln.** |
| Ruby OpenIDConnect | **Yes** | **Yes** | ✓ | ✓ | ✓ | ✓ | ✓ | **Vuln.** |
| Apache Oltu | **Yes** | *No* | ✓ | ✓ | **Vuln.** | **Vuln.** | ✓ | **Vuln.** |
| Total Successful Attacks | 8/8 | 6/8 | 4/8 | 4/8 | 5/8 | 5/8 | 3/8 | 8/8 |

TABLE 2: Security analysis results of officially referenced SPs libraries. Only 2 of 8 (25%) libraries implemented *all* required verification steps properly. Legend. Secure/Attack fails: ✓; Insecure/Attack successful: **Vuln.**.

validation of several SAML frameworks. In 2014, Mainka et al. [18] evaluated the SAML interfaces of cloud provider and successfully applied different Single-Phase Attacks, for example, Replay attacks, Token Recipient Confusion (TRC) and XSW. None of the previous work considered an evaluation via an attacker IdP.

`BrowserId:` In 2014 Fett et al. [9] built a formal model of the BrowserId protocol. Based on the analysis the authors defined possible points of Single-Phase Attack and found a IDS vulnerability by manual testing. Cross-Phase Attacks were not considered. Since Mozilla will end the BrowserId support in 30th November 2016, further security evaluation is not likely.

`OpenID:` In 2008, Newman and Lingamneni [22] created a model checker for OpenID and identifying a session swapping vulnerability, which enforces the victim to log in into attacker's account on an SP. In 2012 Sun et al. [33] analyzed OpenID Connect in a formal analysis and identified several existing threats such as CSRF, Man-in-the-middle attacks and the SSL support of OpenID implementations. Wang et al. [37] demonstrated the problems related to token verification with different attacks targeting implementation issues. In 2016 a comprehensive evaluation regarding the OpenID security was published by Mainka et al. [19]. The authors considered for the first time an attacker IdP for security evaluation of SSO.

`OAuth:` OAuth has been analyzed in different formal models [4, 28]. Additional threats are also considered in the OAuth Threat Model and Security Considerations [26]. In 2016, Fett et al. [12] formally analyzed OAuth parallels and independent to our work and discovered generic flaws that can be exploited by a network attacker. The IdP Mix-Up attack on OAuth is similar to our IdP Confusion attack on OpenID Connect, but our attack requires only the web attacker model instead of an attacker controlling the network. In 2012 Sun and Beznosov [32] provided a large-scale study regarding the security of OAuth implementations, and found serious security flaws in many of them. The evaluation concentrated on classical web attacks like Cross-Site Scripting (XSS), CSRF and TLS misconfiguration. Further security flaws in OAuth based applications were discovered [8, 23, 27, 41] whereby the authors concentrated on individual attacks. In 2013 Wang et al. introduced a systematic process for identifying critical assumptions in SDKs, which led to the identification of exploits in constructed apps resulting in changes in the OAuth specification [38]. Chen et al. [6] revealed in 2014 serious vulnerabilities in OAuth applications on mobile devices caused by the developer's misinterpretation of the OAuth protocol.

`OpenID Connect:` In 2015, Wanpeng Li [39] analyzed OpenID Connect by evaluating the security of 103 SPs using Google as an IdP and found several vulnerabilities like Replay attacks, Man-in-the-middle, session swapping and XSS. Attacks like IDS and Cross-Phase Attacks were not considered. The authors of this paper summarized their knowledge of all existing attacks on OpenID Connect in a technical report [21]. The document contains the attacks described in this paper and examples contributing to a better understanding.

**Automated Penetration Testing Tools.** In 2013, Bai et al. [3] introduced AuthScan, a penetration testing tool extracting the authentication protocol automatically based on HTTP traces and JavaScript code. The authors found security flaws in several SSO systems like MitM attacks, Replay attacks and Guessable tokens. More complex attacks, like IDS or Cross-Phase Attacks were not considered. Xing et al. [40] published *InteGuard* - a tool detecting the invariance in the communication between the browser and the SP. Another tool similar to *InteGuard* is *BLOCK* [16]. Both tools can detect Single-Phase Attacks. However, Cross-Phase Attacks requiring the usage of attacker IdP were not covered by both tools. Yuchen Zhou [41] published a fully automated tool named *SSOScan* for analyzing the security of OAuth implementations and described five attacks, which can be automatically tested by the tool. We used this work as a basis of PrOfESSOS – a website capable to evaluate different implementations. However, *SSOScan* is limited to the analysis of Facebook SSO and cannot consider attacks like IDS and Cross-Phase Attacks. Mainka et al. [19] published in 2016 OpenID Attacker - the full automated tool to analyze OpenID by using an attacker IdP. OpenID Attacker is limited to OpenID and is not available as a service. Thus, it has to be downloaded and then configured for each test. In 2016 Sudhodanan et al. [31] introduced an automated tool for black box testing of Multi-Party Web applications, including Cashier-as-a-Service, and discovered multiple vulnerabilities. They extended the OWASP ZAP tool and implemented seven attack patterns. However, the implementation is not public available and the authors of the paper excluded the usage of attacker IdP.

## 10. Conclusion

We showed that although the OpenID Connect specification addresses (most) attacks or at least provides countermeasures without stating attacks (Q1), implementations often do not follow its guideline to implement it securely (Q2). All investigated OpenID Connect libraries have critical implementation flaws resulting in broken End-User authentication. We have thus to assume that implementation flaws will always exist since developers are not always security experts and do not follow the specification instructions perfectly. To tighten the gap between a specification and an implementation, we propose PrOfESSOS, an EaaS that automatically performs and evaluates attacks (Q3) which we divide into Single-Phase and Cross-Phase Attacks. Since PrOfESSOS is open source and available on Github, it can deal as the basis for many other research projects, not only limited to SSO, but also to other multi-party web applications, for example, Cashier-as-a-Service. In a future work, we plan to extend PrOfESSOS to also validate IdP implementations of OpenID Connect for completeness. Based on our experiences with SSO security with all major protocols (SAML, OpenID, OpenID Connect, OAuth, BrowserId), we are absolutely convinced that an approach like PrOfESSOS is the best solution to reach the required security level, which a critical component, like an SSO library, needs. To reach this goal, we do not plan to start a large scale study which will obviously reveal that implementation issues exist, but instead, we are working with the OAuth and OpenID Connect working group to include PrOfESSOS in their certification process. If big companies such as Google or Microsoft follow this, the overall security of their SPs will be significantly improved.

## Acknowledgement

We would like to thank Julian Krautwald for supporting our research by his master thesis, Vladimir Dzhuvinov for initiating the communication with the IETF, and Hannes Tschofenig from the IETF working group for his insightful discussions and suggestions. We further thank Blase Ur for shepherding this paper and giving valuable input and contributions.

## References

[1] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra. Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps. pages 1–10. ACM, 2008.

[2] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, and M. Llanos Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In Vitaly Shmatikov, editor, *ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 1–10, Alexandria and VA and USA, 2008. ACM.

[3] Guangdong Bai, Jike Lei, Guozhu Meng, Sai Sathyanarayan Venkatraman, Prateek Saxena, Jun Sun, Yang Liu, and Jin Song Dong. Authscan: Automatic extraction of web authentication protocols from implementations. *Network and Distributed System Security Symposium (NDSS)*, 2013.

[4] Chetan Bansal, Karthikeyan Bhargavan, and Sergio Maffeis. Discovering concrete attacks on website authorization by formal analysis. In *IEEE Computer Security Foundations Symposium (CSF)*, Washington, DC, USA, 2012.

[5] Scott Cantor and Rod Widdowson. Identity provider discovery service protocol and profile, March 2008. URL http://www.oasis-open.org/committees/download.php/28049/sstc-saml-idp-discovery-cs-01.pdf.

[6] Eric Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague. Oauth demystied for mobile application developers. In *ACM Conference on Computer and Communications Security (CCS)*, November 2014.

[7] Dominic Scheirlinck and Scott Geary. CVE-2016-5385 CVE-2016-5386 CVE-2016-5387 CVE-2016-5388 CVE-2016-1000109 CVE-2016-1000110, 7 2016. URL https://www.kb.cert.org/vuls/id/797896.

[8] Egor Homakov. How we hacked Facebook with OAuth2 and Chrome bugs, 2 2013. URL http://homakov.blogspot.ca/2013/02/hacking-facebook-with-oauth2-and-chrome.html.

[9] Daniel Fett, Ralf Küsters, and Guido Schmitz. An expressive model for the web infrastructure: Definition and application to the browserid sso system. In *IEEE Symposium on Security and Privacy (S&P)*, 2014.

[10] Daniel Fett, Ralf Küsters, and Guido Schmitz. Spresso: A secure, privacy-respecting single sign-on system for the web. In *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[11] Daniel Fett, Ralf Küsters, and Guido Schmitz. A comprehensive formal security analysis of OAuth 2.0. *arXiv preprint arXiv:1601.01229*, 2016.

[12] Daniel Fett, Ralf Küsters, and Guido Schmitz. A comprehensive formal security analysis of oauth 2.0. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[13] T. Groß. Security analysis of the SAML Single Sign-on Browser/Artifact profile. In *Annual Computer Security Applications Conference (ACSAC)*. IEEE Computer Society, 2003.

[14] Thomas Groß and Birgit Pfitzmann. SAML artifact information flow revisited. Research Report RZ 3643 (99653), IBM Research, 2006.

[15] M. Jones, N. Sakimura, and J. Bradley. Oauth 2.0 authorization server metadata. IETF, August 2016. URL https://tools.ietf.org/html/draft-ietf-oauth-discovery-04.

[16] Xiaowei Li and Yuan Xue. Block: A black-box approach for detection of state violation attacks towards web applications. In *Annual Computer Security Applications Conference (ACSAC)*, 2011.

[17] J. Bradley M. Jones. Oauth 2.0 mix-up mitigation draft. IETF, Internet Draft, January 16 2016. URL https://tools.ietf.org/html/draft-ietf-oauth-mix-up-mitigation-01.

[18] Christian Mainka, Vladislav Mladenov, Florian Feldmann, Julian Krautwald, and Jörg Schwenk. Your software at my service: Security analysis of saas single sign-on solutions in the cloud. In *ACM Cloud Computing Security Workshop (CCSW)*, 2014.

[19] Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. Do not trust me: Using malicious IdPs for analyzing and attacking single sign-on. In *IEEE European Symposium on Security and Privacy (EuroS&P 2016)*, 2016.

[20] Tim McLean. Critical vulnerabilities in json web token libraries, 3 2015. URL https://auth0.com/blog/2015/03/31/critical-vulnerabilities-in-json-web-token-libraries/.

[21] Vladislav Mladenov and Christian Mainka. Openid connect. security considerations, 2016. URL https://www.nds.rub.de/media/ei/veroeffentlichungen/2017/01/13/OIDCSecurity_1.pdf.

[22] Ben Newman and Shivaram Lingamneni. Cs259 final project: Openid (session swapping attack), 2008. URL http://www.stanford.edu/class/cs259/projects/cs259-final-newmanb-slingamn/report.pdf.

[23] Nir Goldshlager. How I Hacked Facebook OAuth To Get Full Permission On Any Facebook Account (Without App "Allow" Interaction), 2 2013. URL http://www.nirgoldshlager.com/2013/02/how-i-hacked-facebook-oauth-to-get-full.html.

[24] openid.net. Openid connect 1.0: Openid certification. Technical report, OpenID Foundation, January 2016. URL http://openid.net/certification/.

[25] openid.net. Openid connect 1.0: Libraries, products, tools. Technical report, OpenID Foundation, January 2016. URL http://openid.net/developers/libraries/.

[26] RFC6819. Oauth 2.0 threat model and security considerations.

[27] Ethan Shernan, Henry Carter, Dave Tian, Patrick Traynor, and Kevin Butler. More guidelines than rules: Csrf vulnerabilities from noncompliant oauth 2.0 implementations. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2015.

[28] Quinn Slack and Roy Frostig. Oauth 2.0 implicit grant flow, 2011. URL http://web.stanford.edu/class/cs259/WWW11/.

[29] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. On breaking saml: Be whoever you want to be. In *USENIX Security Symposium*, 2012.

[30] specs@openid.net. OpenID Authentication 2.0 – Final, December 2007. URL https://openid.net/specs/openid-authentication-2_0.html.

[31] Avinash Sudhodanan, Alessandro Armando, Roberto Carbone, and Luca Compagna. Attack patterns for black-box security testing of multi-party web applications. In *Network and Distributed System Security Symposium (NDSS)*, 2016.

[32] San-Tsai Sun and Konstantin Beznosov. The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[33] San-Tsai Sun, Kirstie Hawkey, and Konstantin Beznosov. Systematically breaking and fixing openid security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. *Computers & Security*, 31(4), 2012.

[34] The OpenID Foundation (OIDF). OpenID Connect Core 1.0, 2 2014. URL http://openid.net/specs/openid-connect-core-1_0.html.

[35] The OpenID Foundation (OIDF). Openid connect discovery 1.0, 2 2014. URL http://openid.net/specs/openid-connect-discovery-1_0.html.

[36] The OpenID Foundation (OIDF). Openid connect dynamic client registration 1.0, 2 2014. URL http://openid.net/specs/openid-connect-registration-1_0.html.

[37] Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[38] Rui Wang, Yuchen Zhou, Shuo Chen, Shaz Qadeer, David Evans, and Yuri Gurevich. Explicating sdks: Uncovering assumptions underlying secure authentication and authorization. In *USENIX Security Symposium*, Berkeley, CA, USA, 2013.

[39] Chris Mitchell Wanpeng Li. Analysing the Security of Google's implementation of OpenID Connect, 2015. URL http://arxiv.org/abs/1508.01707.

[40] Luyi Xing, Yangyi Chen, X Wang, and Shuo Chen. Integuard: Toward automatic protection of third-party web service integrations. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[41] David Evans Yuchen Zhou. Automated testing of web applications for single sign-on vulnerabilities. In *USENIX Security Symposium*, San Diego, CA, August 2014.