On the security of modern Single Sign-On Protocols – OpenID Connect 1.0

Vladislav Mladenov, Christian Mainka, Julian Krautwald,
Florian Feldmann, and Jörg Schwenk

Horst Görtz Institute for IT-Security, Ruhr University Bochum
{vladislav.mladenov, christian.mainka, julian.krautwald,
florian.feldmann, joerg.schwenk}@rub.de

Abstract

OpenID Connect is a new Single Sign-On (SSO) authentication protocol, which is becoming increasingly important since its publication in February 2014. OpenID Connect relies on the OAuth protocol, which currently is the de facto standard for delegated authorization in the modern web and is supported by leading companies like, e.g., Google, Facebook and Twitter.

An important limitation of OAuth is the fact that it was designed for authorization and not for authentication – it introduces a concept that allows a third party, e.g., a mobile App or a web application, to only access a subset of resources belonging to a user. However, it does not provide a secure means to uniquely identify the user. Thus, recent research revealed existing problems in case that OAuth is used for authentication nonetheless. These problems result in severe security vulnerabilities [9,31].

To fill this gap, OpenID Connect was created. It provides federated identity management and authentication by adding authentication capabilities on top of the OAuth protocol. Although OpenID Connect is a very new standard, companies like Google, Microsoft, AOL and PayPal, who were also involved in the development, use it already [22,1,3,2].

In this paper we describe the OpenID Connect protocol and provide the first in-depth analysis of one of the key features of OpenID Connect, the *Discovery* and the *Dynamic Registration* extensions. We show that the usage of these extensions can compromise the security of the entire protocol. We develop a new attack called *Malicious Endpoints* attack, evaluate it against an existing implementation, and propose countermeasures to fix the presented issues.

Keywords: Single Sign-On, OpenID Connect, Authentication flaws

1 Introduction

Password-based authentication is still the most commonly used method to authenticate on the web. Almost every website on the Internet requiring authentication supports at least password-based authentication. The biggest problem

with password-based authentication is its usability. Users tend to have at least 25 accounts on the web on average, but use only 10 different passwords [13]. One possibility to address this issue is the usage of *password managers*. However, the usage of classical desktop password managers is associated with problems with the portability to other platforms, for example mobile devices, and their synchronization. Even though web-based password managers aim to solve those problems, a recent security study revealed these managers are vulnerable to a plethora of attacks [17].

SSO. Another approach is the usage of SSO, which applies delegated authentication by including trusted third parties (called Identity Providers (IdPs)) into the authentication procedure. An IdP manages identities of multiple users, provides specific authentication mechanisms (e.g., username/password or 2-factor), and creates authentication tokens for authenticated users. These authentication tokens are consumed by a party called Service Provider (SP), which then grants access to corresponding restricted resources to the *End-User*.

The main advantage of SSO is the fact that an End-User has to manage only the credentials for his IdP. Thus, he does not have to remember multiple username/password combinations, but only one. Additionally, once the End-User is authenticated on the IdP, he gains access to multiple SPs without being prompted with another login dialog again. As a result the usability regarding authentication procedures increases.

OpenID. One widely used SSO protocol is OpenID [7]. One of the main OpenID features is its *openness*. *Open* in the context of SSO means that users can be logged into an SP even if the user's IdP has not been known to the SP beforehand. The *trust relationship* between the user's IdP and the SP is established dynamically and on the fly.

OpenID Connect. Google ceased to support OpenID in April 2015. Thus, all SPs previously using OpenID had to switch to another SSO protocol. Google advises its customers to switch to the successor of OpenID – OpenID Connect [4]. The OpenID Connect protocol is based on OAuth, but uses several ideas from OpenID. The key feature of OpenID, the dynamic and fully automatic trust establishment between IdP and SP, is also present in the OpenID Connect protocol by means of the *Discovery* and *Dynamic Registration* extensions.

Since OpenID Connect is based on OAuth, all the attacks evaluated by previous security researches regarding OAuth are relevant for OpenID Connect, too [24,30,9]. Thus, in our work we concentrate on the OpenID Connect features that are not part of the OAuth protocol and thus have not been considered until now.

Contribution. In this paper, we analyze the *Discovery* and *Dynamic Registration* extensions and their impact on the security of the whole OpenID Connect protocol. We present a new attack – *Malicious Endpoints attack* – which can lead to a theft of sensitive data like authentication tokens or secret keys. As a result, an attacker can use these parameters to access restricted resources of the victim. The attack can be applied on any OpenID Connect implementation

using *Discovery* and *Dynamic Registration*, since it uses a logical flaw in the specification.

The contribution of this paper can be summarized as follows:

- We are the first to analyze OpenID Connect's extensions *Discovery* and *Dynamic Registration* in depth.
- We develop a novel attack called *Malicious Endpoints*, which exploits a lack in the OpenID Connect specification and compromises the security of the OpenID Connect protocol.
- We introduce three countermeasures preventing the discovered authentication flaws, and discuss their respective advantages and disadvantages.

Our results show that protocol extensions must be designed with extreme care, and their security implications have to be discussed thoroughly. Otherwise, they can lead to novel attacks with a critical impact, even in secure standards.

The *Discovery* and *Dynamic Registration* are optional extensions and are currently supported by few libraries. Only two of these libraries are officially certified to specific conformance profiles and interoperability among the implementations – MITREidConnect and phpOIDC [14]. Thus, for testing purposes, we verified our attacks using one of the certified libraries – MITREid Connect. The extensions are not currently used by any publicly available systems.

However, it should be considered that our Malicious Endpoints attack is targeted at the OpenID Connect specification itself and not at specific implementations. Thus, any implementation using the *Discovery* and *Dynamic Registration* extensions is vulnerable against the Malicious Endpoints attack.

2 Modern SSO Protocols

Since their establishment in the early 1980s, protocols like Kerberos (first officially published in 1987 as Version 4 [18]) and the corresponding concepts of delegated authentication and authorization using Trusted Third Partys (TTPs) have been constantly developed and refined into modern SSO protocols. These protocols aim at being compliant with the requirements of the modern and flexible Internet infrastructure. Mainly, modern SSO protocols strive to achieve the following goals:

(1) **Decentralization** – SSO appears to be centralized embracing only a very small set of fixed TTPs. The most widely known of these TTPs are Facebook and Google. However, exporting data and outsourcing infrastructure to companies like Facebook and Google can include certain security risks and trust issues.

Luckily, modern protocols like OAuth 2.0, SAML, BrowserID, OpenID and OpenID Connect are designed and specified to set up custom TTPs, which act independently from each other. This enables companies to set up their own TTPs and use these for authentication purposes instead of having to rely on external providers.

(2) **Trust Establishment** – Every SP has to establish a trust relationship with the TTP. In order to do this, key material has to be exchanged. An im-

portant requirement for modern SSO protocols is that this process occurs with minimal configuration, implementation, and installation effort. In the best case, the trust establishment should work automatically.

2.1 OpenID Connect

The OpenID Connect protocol efficiently addresses the goals stated above – it is decentralized and allows for automated trust establishment without any configuration effort or user interaction.

OpenID Connect was designed on basis of the OAuth 2.0 family of specifications [23] in order to enable the authentication of *End-Users* without changing the OAuth 2.0 protocol flow. Thus, OAuth 2.0 capabilities are integrated with the protocol itself [25], providing OpenID Connect with the capability of delegated authorization.

Additionally, OpenID Connect also incorporates concepts utilized by another SSO protocol – OpenID [28]. Such concepts are the Discovery and Dynamic Registration of OpenID Providers (OPs). The Discovery process allows a SP to automatically discover new OPs without any configuration and user interaction. The Dynamic Registration enables the on-the-fly registration and trust establishment between a Client and OP, also without any user interaction.

A major advantage of OpenID Connect is its integration into existing applications: OpenID Connect was designed to be easily integrated into current OAuth 2.0 compliant systems, with only minimal extensions to the already available OAuth 2.0 APIs.

2.2 Roles

Within the OpenID Connect protocol, three different parties each assuming a different role can be found. The relationship between the different roles can be seen in Figure 1.

The End-User, represented by his user agent (UA), wants to access selected services of a Client. Therefore, he needs to prove his identity to the Client. Additionally, the End-User has the possibility to authorize the Client to access a specific set of his resources stored on the OP.

The Client is an application providing a certain service, which requires authentication of an End-User. This authentication process is delegated to the corresponding OP. Therefore, the Client requests an authentication token signed by OP, which proves the identity of the End-User. Optionally, the Client can also request authorization to access certain protected resources of the End-User stored on OP, e.g. photos.

Please note that the term "Client" according to OpenID Connect terminology denotes an SP according to the common SSO terminology – a service, which can be accessed by the End-User. In order to be compliant to the terminology within the OAuth and OpenID Connect specification, we will use the term "Client" from now on.

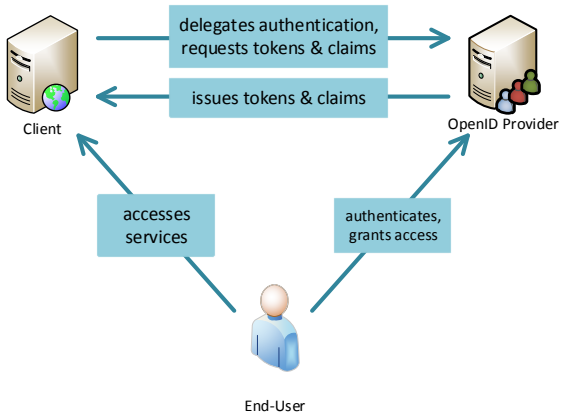

Fig. 1: Role Relationship within the OpenID Connect protocol [25, Section 1.3]

The OpenID Provider (OP) acts as a TTP/IdP towards Client and End-User, handles End-User authentication and issues an authentication token containing a specific set of claims proving the identity of the End-User. Additionally, an authorization token can be issued, in order to authorize the Client to access End-User's resources.

2.3 OP Endpoints

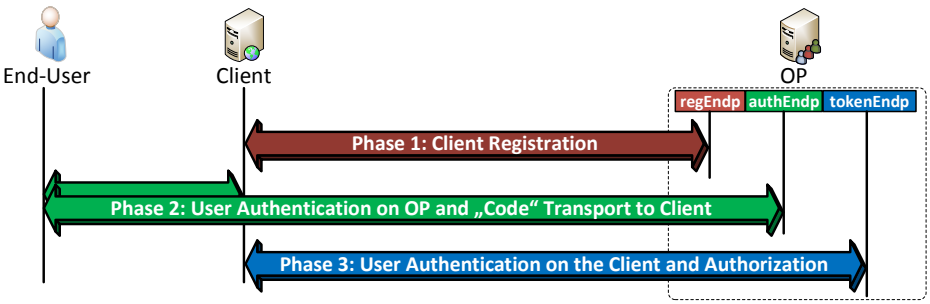

Fig. 2: The general information flow in OpenID Connect contains three phases.

Within the OpenID Connect Core specification [25] the following endpoints on the OP are defined:

(1.) *Registration Endpoint (regEndp):* In order to use OpenID Connect services for authentication, a Client has to register on the OP. For this registration, the Client accesses this URL *regEndp*, e.g., `https://google.com/register`.

(2.) *Authorization Endpoint (authEndp):* In order to execute the Authentication Request of the Client, the End-User has to be redirected to the *authEndp* of the OP, e.g., `https://login.google.com/`. Here, the End-User has to authenticate to the OP via a corresponding authentication process and authorize the Client to access the requested resources.

(3.) *Token Endpoint (tokenEndp):* The Client communicates with the *tokenEndp*, e.g., `https://google.com/consume-token`, in order to obtain the `id_token` described in Section 2.5 and authenticate the End-User. In addition, an `access_token` can be sent to the Client in order to authorize the access to restricted resources. This communication is done directly between Client and OP (without involving the End-User).

Figure 2 depicts the relation between the endpoints and the phases of the OpenID Connect protocol.

2.4 Information Flow

OpenID Connect contains three phases, as shown in Figure 2. In this section we explain the information flow during the different phases.

Phase 1: Client Registration. The Client initially communicates with the OP's registration endpoint (*regEndp*). It submits the *domain(s)*, where the Client is deployed, for example `https://clientA.com`. The OP then generates a random `client_id`/ `client_secret` pair, stores them both together with the domain as a triplet, and sends the credentials to the Client. The Client stores the same information in order to use it during phases 2 and 3.

The Client's registration is mostly processed only once and is usually done via the web interface of the OP, for example by the domain administrator or the developer of the Client. Thus, the registration needs user interaction, causes management overhead, and cannot be executed automatically.

Phase 2: User Authentication on the OP. In the context of delegated authentication the Client redirects unauthenticated End-Users to the Authentication service endpoint on the OP (*authEndp*). Subsequently, the End-User authenticates to the OP using his credentials. Then, the OP generates an authorization `code` and sends it to the End-User. The `code` an intermediary between the End-User and the Client. By sending the `code` to the Client, the End-User authorizes the Client to access restricted resources.

Once received, the Client uses the `code` to retrieve the authentication token (`id_token`), containing user's identity, and optionally the `access_token` granting access to restricted resources.

Phase 3: User Authentication on the OP – ID and Access Token. Once the Client receives the `code`, it sends it to the Token Service endpoint on the OP (*tokenEndp*). In the same message, the Client sends its credentials (`client_id` and `client_secret`, cf. Phase 1) and authenticates to the OP.

The OP responds with the `id_token` and possibly the (optional) `access_token`. Once the `id_token` is received, the Client verifies it and subsequently authenticates the End-User.

The optional `access_token` is part of the OAuth protocol flow and it authorizes the Client to access restricted resources of the End-User on the OP.

2.5 ID Token

The OpenID Connect protocol is basically an extension of OAuth 2.0 by adding an `id_token`. The `id_token` is a security token containing claims about the identity of an End-User by an OP, proving the End-User's identity to a Client. Its data structure is represented as a JSON Web Token (JWT) [16]. In order to provide authenticity as well as integrity of the token, the OP is responsible for signing it using JSON Web Signature (JWS) [15].

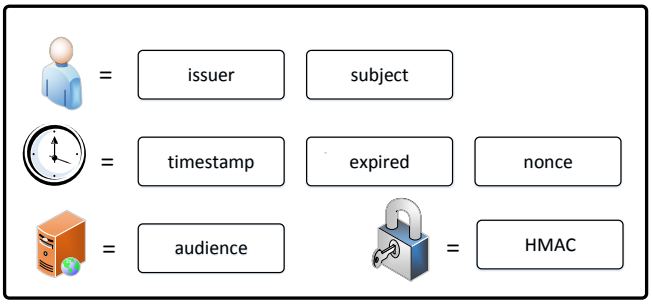

Fig. 3: Abstract overview of the `id_token` used in OpenID Connect for End-User authentication.

Identity of the End-User. The identity of the user consists of two parts: (1.) *issuer* and (2.) *subject*.

The `issuer` is a mandatory identity claim identifying the originator (the OpenID Provider) of the `id_token`, for example `https://www.myOpenIDProvider.com`.

The `subject` is a mandatory identity claim that specifies the End-User's identifier and is consumed by the Client. Issued by the OpenID Provider (`issuer`), it has to be locally unique and never reassigned (e.g. `alice@myOpenIDProvider.com`).

It is essential to note that *both* values – `issuer` and `subject` – must be used to uniquely identify the End-User. As an example, the following two identities are different and must not point to the same End-User.

```
ID Token 1{
  issuer: https://www.myOpenIDProvider.com
  subject: Alice
}
```

```
ID Token 2{
  issuer: https://www.otherOpenIDProvider.org
  subject: Alice
}
```

Timestamps and Freshness. The claims `timestamp` and `expired` define the creation and expiration times of the token. The `nonce` claim is a randomly chosen String value, sent by the Client within the Authentication Request and passed through unmodified to the `id_token`, used to mitigate replay attacks.

Audience Restrictions. The `audience` is a mandatory claim specifying the audience(s) that this `id_token` is intended to be used for (e.g. `https://clientA.com`). It must, at least, contain the `client_id` of the Client which requests the token.

Integrity Protection. OpenID Connect relies on JWT to create the `id_token`. JWT uses JSON Signatures for integrity protection, so does OpenID Connect.

3 Approaches for Attacks

When attacking the OpenID Connect protocol, the eventual goal of an attacker is to get unauthorized access to the victim's protected resources. There are three main approaches how an attacker can achieve this goal:

Compromise Client Credentials. In Phase 1 – during the registration – the Client receives the `client_id` and the `client_secret`. Both parameters are used for the Client's authentication to the OP. If these credentials are compromised, the attacker can use them to impersonate the Client.

Token Theft. In OpenID Connect, there are three different tokens: The `code`, the `id_token` and the `access_token`. Leakage of any of them can allow an attacker to get unauthorized access to restricted resources.

ID Token Manipulation. The `id_token` contains sensitive information about a user's identity. If the attacker can somehow manipulate the information (fully or partly) contained in it, he may be able to impersonate every user on the OP. Note that the `id_token` usually is integrity protected (e.g., by a digital signature), thus, to manipulate it, the attacker has to find a way to circumvent this integrity protection.

4 Gap in Security Evaluations

By considering previous work regarding the security of SSO protocols, we observed that its analysis concentrates on Phase 2 and Phase 3 [24,31,9]. Concentrating on those two phases seems plausible, because the End-User authenticates in Phase 2 and the authentication tokens are transmitted in Phase 3. An attacker targeting Phase 2 or Phase 3 can achieve one or more of the goals defined in

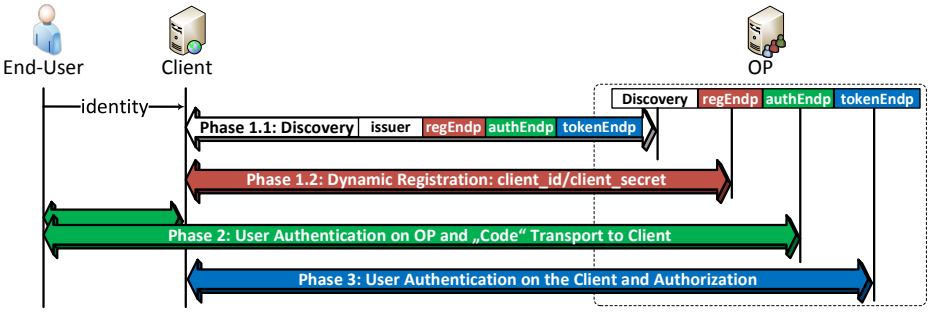

Fig. 4: OpenID Connect Dynamic Registration

Section 3. As a result, previous work revealed security vulnerabilities in Phase 2 and Phase 3, which were fixed and even the specification was changed [9].

Nevertheless, the entirety of Phase 1 has not been considered so far. This is reasoned by the fact that the Client Registration and Key Transport between Client and OP are usually executed manually. For instance, the developer of a Facebook App has to visit his Facebook developer website, and click on *create new App*. Facebook will then generate a `client_id` and a `client_secret` that can be copied by the developer and then pasted into his App configuration.

In contrast to this manual execution, protocols like OpenID and OpenID Connect can also execute the Client Registration automatically. Especially for OpenID Connect, this issue is addressed by introducing a new approach for the Client registration: The so called *Dynamic Registration* [27] allows registration to be automatic, transparent and without any user interaction. However, an important security question raised about this development is: *How does this feature affect the security of the protocol?*

In the following section we describe the *Dynamic Registration* in detail and show a deficiency in its standardization that can be abused to start attacks.

4.1 Phase 1 – Fully Automated Dynamic Registration

The information flow during the automated client registration is shown in Figure 4. Initially, the End-User submits his *identity*, for example *alice@google.com*, to the Client. In order to initiate the SSO authentication, the Client needs to discover the corresponding OP controlling the identity of Alice. Thus, in Phase 1.1, the Client uses the provided *identity* and extracts the domain name of the OP [26, Section 2.1]. In our example, Alice's identity is controlled by the domain *google.com*. The domain name uniquely identifies the corresponding Discovery endpoint[1].

The Client sends an HTTP request to this Discovery endpoint and subsequently retrieves the OP's configuration information including its endpoint

[1] This is usually realized by applying a modifier to the domain, e.g., `http://google.com/`*discovery*.

locations: The (Dynamic) Registration Endpoint (*regEndp*), the Authentication Endpoint (*authEndp*) and the Token Endpoint (*tokenEndp*) (c.f., Section 2.3).

In Phase 1.2 (Dynamic Registration) the Client can automatically register on the OP. For that purpose, the Client sends its own URL, for example `http://client.com`, to the *regEndp* URL. The OP responds with a `client_id`/ `client_secret` pair. Finally, the Client and the OP store the credentials in their respective databases and use them during the next phases.

Observations. By analyzing the Dynamic Registration we make the following observations:

- The usage of custom OPs is supported by the OpenID Connect protocol without any pre-configuration, installation or manual interaction (neither on the Client nor on the User-Agent). The End-User has to enter his identity on the Client, e.g. `bob@customOP.com`, in order to start the authentication with his custom OP.

- All three discovered endpoints are URLs. There are no limitations specified that restrict these URLs to domains, subdomains, or URL contexts.

4.2 Security considerations

During the Dynamic Registration Phase the Client retrieves the OP's configuration information including the endpoints *regEndp, authEndp, tokenEndp*.

However, a malicious Discovery service can freely choose all these parameters (c.f., Phase 1.1 in Figure 4). By this means, the malicious Discovery service can influence on which URL the Client registers (*regEndp*), which URL is used by the End-User to authenticate (*authEndp*) and to which URL the token will be finally sent (*tokenEndp*).

Section 6 will show an applicable attack achieving the goal stated in Section 3.

5 Security Model

This section will give a detailed description of the security model used in the analysis of the protocol.

5.1 Assumptions

We make the following assumptions for the analyzed systems:

- *Secure TLS channels*: A huge proportion of the security of OpenID Connect is based on the assumption that Transport Layer Security (TLS) is used to secure the communication between the involved parties. Naturally, we follow this approach and assume the corresponding secure channels created by TLS to be secure.

- *No compromised software*: All software used by the End-User is assumed to *not* be compromised. This especially holds for user agent and operating

system – we assume no malicious web browser plugins, no keyloggers etc., are active on the End-User's system.

- *No impersonation towards the End-User*: Even though we allow – and even require – the attacker to operate his own web servers and services, we do not assume the attacker can impersonate web applications. We thus assume that the End-User can neither be tricked into accepting attacker generated TLS certificates as valid certificates for genuine Clients, nor will he react to Phishing mails claiming to originate from the legitimate Client. In short, we assume the attacker is not able to impersonate a legitimate Client towards the End-User in any meaningful way.

5.2 Capabilities of the Attacker

The attacks to-be-introduced in this work have been strictly verified in the "web attacker" model [6]. In contrast to the network-based ("cryptographic") attacker model, the web attacker does not need full control over the network and thus is not able to eavesdrop on or manipulate network connections.

He is, however, able to use a UA or a custom Hypertext Transfer Protocol (HTTP) client to send arbitrary HTTP requests to every publicly available web application in the web and subsequently receive its responses. Corresponding HTTP parameters (to-be-sent within a request) as well as headers of these requests can be freely chosen or manipulated. Requests of the attacker can also be delayed or aborted and responses do not need to be handled in a standard-compliant way – for example, HTTP redirect responses issued by a legitimate web application do not need to be followed by the attacker. For tests within live implementations the attacker is able to register as many accounts on a specific Client or OP as he wishes.

Furthermore, the attacker can use links (e.g., sent via email) or web-blog commentaries to lure the victim into opening a (manipulated) Uniform Resource Identifier (URI) to, for example, conduct Cross-Site Request Forgery attacks.

Other attacks on the web application part not directly handling OpenID Connect – like SQL Injections or Cross-Site Scripting attacks – are considered out of scope.

5.3 Behavior of the Victim

Within our security model, the victim is assumed to visit every publicly available web application he wants to or is directed to (e.g. by sending the victim a link to a web application controlled by the attacker) [5]. By this means, the attacker can provide a malicious link to the victim and the victim will follow this link. HTTP parameters within requests generated by the victim by following such malicious links are not checked for sanity or validity.

Phishing attempts, like reconstructing a known website for example, however, are discovered by the victim and thus do not lead to the exposure of sensitive information.

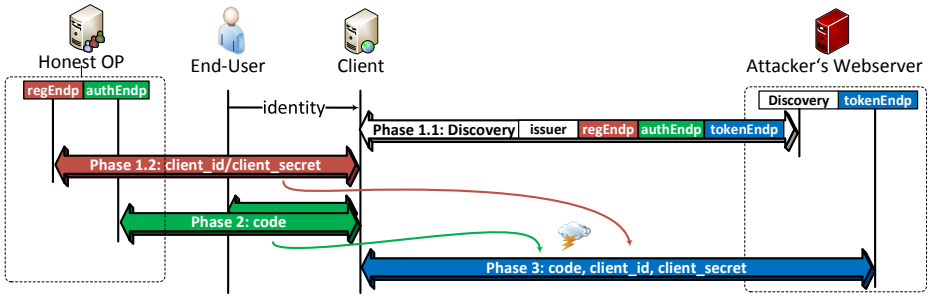

Fig. 5: Malicious Endpoints attack: Attacker's Discovery service sets the endpoint variables in a specific way, such that the secret tokens sent in the third phase are seamlessly distributed to the attacker's server.

6 Malicious Endpoints Attack

This section describes the Malicious Endpoints attack. The idea behind the attack is to influence the information flow in the Discovery and Dynamic Registration Phase in such a way that the attacker gains access to sensitive information.

Setup. The basic setup for the attack is as follows:

- The End-User (*victim*) has an active account on the genuine *honest Client*. We assume that the End-User trusts this Client and the Client follows the OpenID Connect protocol rules.
- The End-User is registered at the *honest OP* on the domain `https://honestOP.com`. The End-User trusts this OP and the OP also follows the OpenID Connect protocol rules.
- To perform the attack, the attacker has to set up his own Discovery service running on the domain `http://malicious.com`. This Discovery Service acts maliciously in that it deviates from the OpenID Connect protocol flow as described in Section 6.1. Note, however, that there is no need to disguise `http://malicious.com` as the regular Discovery service belonging to the honest OP in any way.
- According to the attacker model, the attacker does not hold any control over the honest Client, the End-User, the honest OP or the network traffic between these instances. The attacker is, however, able to lure the victim into activating an attacker provided malicious link.

Main Goal. The attacker pursues the theft of the credentials between the honest OP and the honest Client. Additionally, he steals a valid `code` authorizing the Client to access End-User's resources on the honest OP.

As a proof-of-concept, we evaluated the attack against *MITREid Connect* [19], the most advanced library implementing OpenID Connect. This specific library was chosen because of the fact that *MITREid Connect* was one of the first open-

source libraries supporting all features of the OpenID Connect protocol including both extensions – Discovery and Dynamic Registration.

Responsible Disclosure. According to the Responsible Disclosure principle, we promptly reported our findings to the corresponding security team. However, we have not received any reaction to our report yet.

6.1 Attack Description

In the following, we describe the attack protocol flow. The flow is depicted in Figure 5.

Phase 1.1 - Injecting malicious endpoints. The attacker's intention in the first phase is to force a valid Client to use the attacker's malicious Discovery service. For this purpose, he constructs a malicious link and sends it to the End-User. For example, this can be a link to the valid Client containing an identity *alice@malicious.com*.

If the victim clicks on the constructed link and thus visits the Client, the Client starts a discovery phase with the malicious Discovery service `http://malicious.com`. The Client sends a request to determine the corresponding endpoints. The attacker's Discovery service responds with the following values, initiating the actual attack:

```
issuer:     http://malicious.com
regEndp:    https://honestOP.com/register
authEndp:   https://login.honestOP.com/
tokenEndp:  http://malicious.com
```

Listing 1.1: Endpoints returned by the malicious Discovery Service

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

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

Phase 2 – End-User Authentication and Authorization. In the next phase, the Client redirects the End-User to *authEndp*, `https://login.honestOP.com/`, where the End-User has to authenticate himself and authorize the Client. 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 OP and authorize the honest Client, both of which he knows and trusts. Thus, the End-User authorizes the Client and the OP generates the `code`, which is sent to the Client.

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 OP and the Client. Thus, the attack started at the beginning of the protocol flow can be neither detected nor prevented by any of the participants at this point.

Phase 3 – The Theft. In the final Phase 3, the Client redeems the received `code` from the previous phase: It sends the `code` together with the corresponding Client's credentials received during the Dynamic Registration (*client_id/ client_secret*) to the *tokenEndp* originally specified by the malicious Discovery service – in this example `http://malicious.com`, see Listing 1.1.

6.2 Attack – Impact and Limitations

The attack leads to the theft of the Client's credentials as well as of a valid `code`. The attacker can redeem the stolen `code` at the honest OP to receive a valid ID Token and *access token*.

The impact of the attack is limited due to the fact that the Discovery and Dynamic Registration are optional according to the protocol specification. Thus, Clients, which do not support these features, are not affected. Additionally, the Implicit Flow of the protocol is not affected, since the *tokenEndp* is not used during.

Another limitation is the injection of the malicious identity in the first step of the attack, see Figure 5. A Client could provide protection mechanisms against such HTTP requests, like Cross-site-request-forgery (CSRF) tokens.

The stolen access token, however, is not bound to any Client and can be used by the attacker to get access to the resources of the End-User.

6.3 Countermeasure

During our search for applicable countermeasures, we tried to find a solution requiring minimal changes to the protocol and to the existing implementations. As already explained, the participants of the protocol have no means to detect or prevent this attack once the Dynamic Registration has begun. Furthermore, the malicious authentication request sent by the attacker's Discovery service in Phase 1.1 absolutely adheres to the protocol specification: The attack makes use of OpenID Connect's intended full flexibility to assign independent endpoints during Discovery. Thus, no malicious behavior can be detected at this point either.

In the following, we present three possible countermeasures to our attacks. Unfortunately, it seems to be impossible to prevent this type of attack without restricting protocol flexibility.

OPs Whitelisting. A suitable option to prevent the Malicious Endpoints attack is to whitelist the allowed OPs on Client side. By this means, an attacker will not be able to start the authentication process with his Discovery service. However,

this countermeasure limits the flexibility of the Client and reduces the support of custom OPs, which, depending on the according Client, could cause problems.

Endpoint Restrictions. A different means to prevent the attack would be to restrict the possible contents of the *tokenEndp* according to the contents of the *authEndp*. For example, it could be required that the *tokenEndp* MUST be located on the same domain as the *authEndp* and may only differ in subdomain and/or path. This way, an honest Client receiving a Discovery response as described in Section 6.1 could detect this attack and abort the protocol.

However, even though this countermeasure restricts the introduced attack, it does not mitigate it completely. In case the attacker runs his malicious Discovery service on the same Infrastructure-as-a-Service cloud environment as the honest OP, he could bypass this proposed countermeasure by using the same domain or subdomain.

Remove Token Endpoint. The third countermeasure lies in the strongest possible restriction of the endpoints and requires changes in the specification: The main problem shown in Figure 5 and described in Section 6 is that the End-User's authentication and authorization occurs on a different endpoint (*authEndp*) than the generated *code* is transported to (*tokenEndp*).

In order to solve this problem, we propose to entirely remove the *tokenEndp* from the specification and include its functionality into the *authEndp*. Thus, the `code` has to be sent to the (honest) *authEndp* and the attacker will not receive the generated `code` or the Client's credentials in Phase 3.

7 Related Work

In 2012, Wang et al. [29] concentrated on real-life SSO and the analysis of SSO protocols and implementation flaws via Browser related messages (BRM). The authors have well demonstrated the problems related to token verification with different attacks. Additionally, they introduced a tool named BRM-Analyzer, which analyses the traffic passing through a user's browser and detects abnormalities in the protocol flow, attempting to notice attacks. However, since the Malicious Endpoints attack described in this paper follows the protocol specification, no abnormalities can be detected by tools like BRM-Analyzer. Moreover, the BRM-Analyzer cannot observe the communication during the Discovery and Dynamic Registration phase, since the communication between Client and OP occurs directly and not via the user's browser.

In 2012, Sun et al. [24] analyzed nearly 100 OAuth implementations, and found serious security flaws in many of them. The research focused on the impact of classical web attacks like XSS, CSRF and TLS misconfiguration on the OAuth implementations. In [12,10,11,21,20,31], further attacks on OAuth implementations were discovered and reported. However, all these works concentrated on individual attacks and especially implementation misconfiguration.

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 2.0 specification [30]. Chen et al. revealed in 2014 serious vulnerabilities in OAuth applications on mobile devices caused by the developer's misinterpretation of the OAuth protocol [9].

In 2014 Cao et. al. studied vulnerabilities in existing SSO protocols that allow impersonation attacks and analyzed the main reasons leading to these flaws [8]. The authors concentrated only on phase 2 and phase 3 of the SSO protocol flow and did not consider phase 1. Interestingly, the authors of the paper recognized that one main problem in SSO is the lack of authentication between the OP and Client, which is part of the Malicious Endpoints attack.

Please note that none of the previous papers considers the OpenID Connect protocol flow and the Discovery and Dynamic Registration phases.

8 Future Work

In this paper, we analyzed the OpenID Connect protocol flow and found a serious vulnerability in the Discovery and Dynamic Registration.

As proof-of-concept, we evaluated the attack against the library *MITREid Connect*, which is one of the most advanced libraries supporting OpenID Connect and its extensions. To further analyze the impact of the attack, we plan to extend our evaluation to all available libraries implementing OpenID Connect. A starting point for this will be the list published on the official website of OpenID Connect[2]. In addition to analyzing the available libraries, we also plan to examine live websites which use OpenID Connect for authentication and support the Discovery and Dynamic Registration.

Finally, further security analyses are planned on the phases of OpenID Connect and the idea of the usage of arbitrary OPs. With respect to the security, this aspect could be very interesting and bring novel ideas regarding the security of SSO systems.

References

1. Google Identity Platform: OpenID Connect (Mai 2015), `https://developers.google.com/identity/protocols/OpenIDConnect`
2. Integrate Log In with PayPal (Mai 2015), `https://developer.paypal.com/docs/integration/direct/identity/log-in-with-paypal/`
3. MIcrosoft Azure: OpenID Connect 1.0 (Mai 2015), `https://msdn.microsoft.com/de-de/library/azure/dn645541.aspx`
4. Migrating from OpenID 2.0 to OpenID Connect (April 2015), `https://developers.google.com/identity/protocols/OpenID2Migration#shutdown-timetable`
5. Akhawe, D., Barth, A., Lam, P.E., Mitchell, J., Song, D.: Towards a Formal Foundation of Web Security. CSF '10 Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium pp. 290–304 (2010)

[2] `http://openid.net/developers/libraries/`

6. Barth, A., Jackson, C., Mitchell, J.C.: Securing frame communication in browsers. Communications of the ACM - One Laptop Per Child: Vision vs. Reality 52, 83–91 (June 2009), http://seclab.stanford.edu/websec/frames/post-message.pdf
7. BuiltWith: OpenID Usage Statistics (2014), http://trends.builtwith.com/docinfo/OpenID
8. Cao, Y., Shoshitaishvili, Y., Borgolte, K., Kruegel, C., Vigna, G., Chen, Y.: Protecting web-based single sign-on protocols against relying party impersonation attacks through a dedicated bi-directional authenticated secure channel. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) Research in Attacks, Intrusions and Defenses, Lecture Notes in Computer Science, vol. 8688, pp. 276–298. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-11379-1_14
9. Chen, E., Pei, Y., Chen, S., Tian, Y., Kotcher, R., Tague, P.: OAuth Demystied for Mobile Application Developers. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS). ACM – Association for Computing Machinery (November 2014), http://research.microsoft.com/apps/pubs/default.aspx?id=231728
10. Egor Homakov: How we hacked Facebook with OAuth2 and Chrome bugs (Februrary 2013), http://homakov.blogspot.ca/2013/02/hacking-facebook-with-oauth2-and-chrome.html
11. Egor Homakov: OAuth1, OAuth2, OAuth...? (March 2013)
12. Egor Homakov: How I hacked Github again (Februrary 2014)
13. Florencio, D., Herley, C.: A large-scale study of web password habits. In: Proceedings of the 16th International Conference on World Wide Web. pp. 657–666. WWW '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1242572.1242661
14. Foundation, O.: Openid certification (Mai 2015), http://openid.net/certification/
15. Jones, M., Bradley, J., Sakimura, N.: JSON Web Signature (JWS) draft-ietf-jose-json-web-signature-30 (July 2014), http://tools.ietf.org/html/draft-ietf-jose-json-web-signature-30
16. Jones, M., Bradley, J., Sakimura, N.: JSON Web Token (JWT) draft-ietf-oauth-json-web-token-25 (July 2014), http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-25
17. Li, Z., He, W., Akhawe, D., Song, D.: The emperor's new password manager: Security analysis of web-based password managers. In: 23rd USENIX Security Symposium (USENIX Security 14) (2014)
18. Miller, S.P., Neuman, B.C., Schiller, J.I., Saltzer, J.H.: Kerberos authentication and authorization system. In: In Project Athena Technical Plan. Citeseer (1987)
19. MITRE Corporation, MIT Kerberos, Internet Trust (KIT): MITREid Connect (2014)
20. Nir Goldshlager: How I Hacked Any Facebook Account...Again! (March 2013), http://www.nirgoldshlager.com/2013/03/how-i-hacked-any-facebook-accountagain.html
21. Nir Goldshlager: How I Hacked Facebook OAuth To Get Full Permission On Any Facebook Account (Without App "Allow" Interaction) (February 2013), http://www.nirgoldshlager.com/2013/02/how-i-hacked-facebook-oauth-to-get-full.html
22. OpenID Foundation: Openid foundation leadership, http://openid.net/foundation/leadership/
23. RFC6749, I.: The OAuth 2.0 Authorization Framework

24. Sun, S.T., Beznosov, K.: The devil is in the (implementation) details: an empirical analysis of oauth sso systems. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 378–390. ACM (2012)
25. The OpenID Foundation (OIDF): OpenID Connect Core 1.0 (February 2014), http://openid.net/specs/openid-connect-core-1_0.html
26. The OpenID Foundation (OIDF): OpenID Connect Discovery 1.0 (February 2014), http://openid.net/specs/openid-connect-discovery-1_0.html
27. The OpenID Foundation (OIDF): OpenID Connect Dynamic Client Registration 1.0 (February 2014), http://openid.net/specs/openid-connect-registration-1_0.html
28. The OpenID Foundation (OIDF): How is OpenID Connect different from OpenID 2.0 and how does it overcome the problems experienced with OpenID 2.0? (2015), http://openid.net/connect/faq/
29. Wang, R., Chen, S., Wang, X.: Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In: Proceedings of the 2012 IEEE Symposium on Security and Privacy. pp. 365–379. SP '12, IEEE Computer Society, Washington, DC, USA (2012), http://dx.doi.org/10.1109/SP.2012.30
30. Wang, R., Zhou, Y., Chen, S., Qadeer, S., Evans, D., Gurevich, Y.: Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization. In: Proceedings of the 22Nd USENIX Conference on Security. pp. 399–414. SEC'13, USENIX Association, Berkeley, CA, USA (2013), http://dl.acm.org/citation.cfm?id=2534766.2534801
31. Yuchen Zhou, D.E.: Automated Testing of Web Applications for Single Sign-On Vulnerabilities. In: 23rd USENIX Security Symposium (USENIX Security 14). USENIX Association, San Diego, CA (Aug 2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/zhou