RUHR-UNIVERSITÄT BOCHUM

RUB

RUHR-UNIVERSITÄT BOCHUM

Single Sign-On Security: Security Analysis of real-life OpenID Connect Implementations

Lauritz Holtmann

Master's Thesis – September 30, 2020.
Chair for Network and Data Security.

Supervisor: Dr.-Ing. Christian Mainka
Advisor: Prof. Dr. Jörg Schwenk
Advisor: Dr.-Ing. Vladislav Mladenov

hgi Lehrstuhl für Netz- und Datensicherheit

Abstract

OpenID Connect 1.0 is an authentication protocol that extends the OAuth 2.0 Authorization Framework. A typical OpenID Connect 1.0 setup involves three parties: an End-User who wants to sign-in at a service, the OpenID Provider that authenticates the End-User and a Relying Party that provides a service to the End-User. Implementing Single Sign-On protocols like OpenID Connect enables Service Providers to delegate authorization and authentication tasks to a dedicated third party. This decentralized scenario comes with flexibility for implementing entities and usability benefits for End-Users but also introduces new challenges regarding secure and reliable authentication mechanisms. In this thesis, three novel variants of attacks on OpenID Connect implementations and two attacks on the OpenID Connect specification are presented. Besides these novel attacks, four Identity Provider and five Service Provider implementations are evaluated against a set of previously known attacks and requirements resulting from the specification and current security best practices. During the execution of the analysis, NodeJS implementations of the Identity Provider and Service Provider parts of the OpenID Connect specification were created, which are also introduced in this thesis. Finally, common vulnerability patterns observed within the set of OpenID Connect implementations are derived and recommendations for additions to the OpenID Connect security considerations are given.

Official Declaration

Hereby I declare, that I have not submitted this thesis in this or similar form to any other examination at the Ruhr-Universität Bochum or any other institution or university.

I officially ensure, that this paper has been written solely on my own. I herewith officially ensure, that I have not used any other sources but those stated by me. Any and every parts of the text which constitute quotes in original wording or in its essence have been explicitly referred by me by using official marking and proper quotation. This is also valid for used drafts, pictures and similar formats.

I also officially ensure that the printed version as submitted by me fully confirms with my digital version. I agree that the digital version will be used to subject the paper to plagiarism examination.

Not this English translation, but only the official version in German is legally binding.

Eidesstattliche Erklärung

Ich erkläre, dass ich keine Arbeit in gleicher oder ähnlicher Fassung bereits für eine andere Prüfung an der Ruhr-Universität Bochum oder einer anderen Hochschule eingereicht habe.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen, die anderen Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen kenntlich gemacht. Dies gilt sinngemäß auch für verwendete Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Ich erkläre mich damit einverstanden, dass die digitale Version dieser Arbeit zwecks Plagiatsprüfung verwendet wird.

Date	Lauritz Holtmann

Erklärung

Ich erkläre mich damit einverstanden, dass meine Masterarbeit am Lehrstuhl NDS dauerhaft in elektronischer und gedruckter Form aufbewahrt wird und dass die Ergebnisse aus dieser Arbeit unter Einhaltung guter wissenschaftlicher Praxis in der Forschung weiter verwendet werden dürfen.

Date	Lauritz Holtmann

Contents

Glossary

Carriage Return and Line Feed The CRLF sequence is a series of meta characters which is used to represent a newline.

Content Security Policy The Content Security Policy is a web security measure against common attacks like data or script injection attacks. The CSP is specified using the `Content-Security-Policy` HTTP header.

Cross-Site-Request-Forgery Cross-Site-Request-Forgery is a web application vulnerability. To exploit such a vulnerability, a malicious actor tricks the user's browser to perform authenticated actions using the victim's session on behalf of the attacker.

Cross-Site-Scripting Cross-Site-Scripting is a web application vulnerability. To exploit such a vulnerability, a malicious actor injects JavaScript into the context of a website that is then executed in the victim's browser.

Hypertext Transfer Protocol The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for data exchange. It is in use since 1990 and is the foundation of the World-Wide Web.

Insecure Direct Object Reference Insecure Direct Object Reference (IDOR) is a web application vulnerability. If an application does not implement proper access control and performs actions based on user-controlled references, a malicious actor could trick the application to perform actions on objects the attacker has no permissions for.

Referrer-Policy The Referrer-Policy is a web security measure that defines how much information should be included within the HTTP `Referer` header.

Same-Origin Policy The Same-Origin Policy is a web security measure for documents and client-side scripting languages like JavaScript. An origin is determined by an URI's *host*, *scheme* and *port*. The Same-Origin Policy restricts a script's access to resources that are cross-origin.

Server-Side Request Forgery Server-Side Request Forgery is a network-level vulnerability. To exploit such a vulnerability, a malicious actor forces a server - e.g. using inputs within a web application - to perform requests on behalf of the malicious actor.

SQL Injection SQL Injection is a web application vulnerability. If an application does not filter metacharacters, a malicious actor can utilize these characters to modify database queries.

Transmission Control Protocol The Transmission Control Protocol (TCP) is part of the Internet protocol suite and was developed to provide reliable data transmission between applications that communicate over the network.

Transport Layer Security Transport Layer Security (TLS) is a cryptographic protocol to transfer data in an encrypted manner. Its deprecated predecessor is SSL.

Acronyms

CRLF Carriage Return and Line Feed.

CSP Content Security Policy.

CSRF Cross-Site-Request-Forgery.

CVE Common Vulnerabilities and Exposures.

GUI Graphical User Interface.

HTTP Hypertext Transfer Protocol.

IDOR Insecure Direct Object Reference.

IdP Identity Provider.

OIDC OpenID Connect.

PKCE Proof Key for Code Exchange.

SP Service Provider.

SQLi SQL Injection.

SSO Single Sign-On.

SSRF Server-Side Request Forgery.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UI User Interface.

XSS Cross-Site-Scripting.

1 Introduction

This chapter gives a short introduction to the master's thesis "*Single Sign-On Security: Security Analysis of real-life OpenID Connect Implementations*".
At first, motivation for the general research topic is given. Afterward, the topic is set into context by discussing related work. Subsequently, the research question and the contribution of this thesis are outlined. Finally, the organization of this thesis is described.

1.1 Motivation

With the increasing number of online accounts per user, End-Users tend to utilize password strategies. According to a study performed by `Statista` in 2017, more than half of the German users reuse their passwords for multiple accounts [43]. An additional study published by `Statista` in 2019 concludes that more than 20% of the users include personal information like their date of birth in their passwords [44]. One possible concept to address this potentially dangerous behavior is to integrate Single Sign-On protocols into services. In this case, the user has to create an account at one central instance, the *Identity Provider*. In the following, the user may register on other services and websites without having to perform a full registration, and instead, use Single Sign-On and his identity created at the Identity Provider. As a result, the user only has to maintain and protect one central account by choosing one strong password.
In the real world, the initial registration at an Identity Provider is often done implicitly, as services and social networks like Facebook and Google implemented so-called "social logins". As a result, users may log in using their Google or Facebook account on many websites without having to create an explicit account at an Identity Provider. According to `builtwith`, who monitor "*over a quarter of a billion websites*" [6] as stated in their official sources, over 185,000 websites use the Facebook login button or used it in the past [5].

Single Sign-On is not only relevant for individuals, but also for large enterprises who want their employees to use one account on multiple different internal services or let their employees use their corporate login on external services. The Deutsche Telekom for instance adopted OpenID Connect in 2013 (OpenID Connect Core 1.0 was finalized in 2014) and switched its largest service to OpenID Connect in mid of

2014 [10].
One solution to implement OpenID Connect in a corporate environment is using Keycloak, an "*Open Source Identity and Access Management*" [17] tool. Recently there was a penetration test on Keycloak that focused on common web security findings [8]. Keycloak supports multiple Single Sign-On protocols, but this thesis will focus on its *OpenID Connect* implementation and Single Sign-On specific security considerations in the first of two parts.

The second part will cover more implementations of the OpenID Connect protocol but will focus on Client implementations. We will analyze four products: Bitbucket, GitLab, Salesforce and Amazon Cognito. For each product, we will at first attempt to set up a benign Identity Provider based on Keycloak. After verification that everything works as intended, we will proceed and change the benign Identity Provider against a Malicious Identity Provider and analyze the behavior of the products.

1.2 Related Work

Previous research developed techniques to analyze Single Sign-On protocols, attacker models and classifications of attacks. In addition, these techniques were applied to the major Single Sign-On protocols. There are few academic publications on concrete implementations.
In 2016, Fett et al. published their paper "*A Comprehensive Formal Security Analysis of OAuth 2.0*" [14], in which they analyze OAuth 2.0, which is the foundation of OpenID Connect. In the following year, the authors published their follow-up paper "*The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines*" [15], in which they present an in-depth security analysis of OpenID Connect.
The security of Google's OpenID Connect implementation was analyzed by Li and Mitchell in their paper "*Analysing the Security of Google's Implementation of OpenID Connect*" in 2016 [20].
In 2017 Christian Mainka defended his dissertation "*On Message-Level Security*" [25], in which he discusses Web Service Security and Single Sign-On Security, elaborating deep insights in attacks on Single Sign-On protocols. In his dissertation he furthermore analyzes the usage of malicious Identity Providers and Single- and Cross-Phase Attacks on different Single Sign-On protocols including OpenID Connect.
Vladislav Mladenov defended his Dissertation "*On the Security of Single Sign-On*" [31] likewise in 2017. He defines generic Attack Concepts on Single Sign-On protocols and attacker models, in order to apply them to OpenID Connect, OpenID and SAML.
In their paper "*Do not trust me: Using malicious IdPs for analyzing and attacking*

Single Sign-On" [27] Mainka, Mladenov and Schwenk introduce their new approach of using malicious Identity Providers in order to analyze and finally attack Single Sign-On Protocols. This research finally leaded to two papers concerning OpenID Connect Security: "*SoK: Single Sign-On Security – An Evaluation of OpenID Connect*" [28] and "*OpenID Connect Security Considerations*" [26].
Steinegger et al. published their paper "*Migration von OpenID Connect in eine bestehende Anwendungslandschaft*" [45], in which they provide concrete case studies of an implementation of OpenID Connect in a complex environment in 2017.
In the last years, more and more non-academic research was published about OAuth 2.0 and OpenID Connect 1.0. In 2017, Arne Swinnen published a blog post on an authentication bypass he achieved at *Airbnb* by stealing OAuth 2.0 `access_token`s [47]. Likewise in 2017, Max Moroz published a blog post on his OAuth 2.0 research [30]. During his research, Moroz encountered common race conditions and `access_token`/`code` invalidation issues.
Nowadays, Bug Bounty programs encourage researchers to analyze public services and software, so does *Twitter* with its program. In November 2018, Terence Eden reported an issue regarding incorrect permissions that were displayed on Twitter's consent screen [50], resulting in access to direct messages without explicit consent given by the End-User.
Li et al. published their research on "*Mitigating CSRF attacks on OAuth 2.0 and OpenID Connect*" [21] in 2018. In their paper, they outline the root causes for real-world Cross-Site-Request-Forgery issues in the context of OpenID Connect and OAuth 2.0 and propose a novel approach to mitigate CSRF in the wild.
In 2019 Fett et al. published their paper "*An Extensive Formal Security Analysis of the OpenID Financial-Grade API*" [13]. They analyze the OpenID Financial-Grade API, a profile of OAuth 2.0 that is developed in an open process by the OpenID foundation.
In August 2020, Saito et al. published their paper "*Comparison of OAuth/OpenID Connect Security in America and Japan*", in which they analyze 500 American electronic commerce websites regarding their OAuth 2.0 and OpenID Connect 1.0 implementations. Additionally, the results of this evaluation are compared with similar evaluation of Japanese websites [39].

1.3 Research Question

The overall research question of this master's thesis can be summed up to: "*How secure are real-life OpenID Connect implementations in regard to previous research, the specification's security considerations and the OAuth 2.0 Security Best Current Practices?*". Since its specification in 2014, many researchers have analyzed the OpenID Connect 1.0 protocol and outlined security risks. This thesis aims to give an impression on the adoption of previously proposed security mitigations by real-life

implementations and the general awareness of the specification's security considerations.

1.4 Contribution

In this master's thesis, a basic set of previously known attacks from academia and non-formal sources is outlined and complemented with novel attack scenarios. The resulting evaluation catalog is then applied to real-life OpenID Connect implementations. The most relevant findings are described in detail. Further, the analyzed products and services are thoroughly evaluated regarding the evaluation catalog. Finally, common patterns and resulting proposed additions to the OpenID Connect security considerations are derived and a conclusion is drawn.

1.5 Organization of this Thesis

A brief and motivating introduction to the topic of Single Sign-On, in general, and OAuth 2.0 and OpenID Connect 1.0 security, in particular, is given in the first chapter.
The second chapter focuses on the foundations needed for this master's thesis, including technical background regarding JSON and JWT, Single Sign-On and an overview of Single Sign-On security as well as Identity and Service Provider flaws and attacks.
The third chapter introduces the attacker models that are used throughout the thesis. In the fourth chapter, the selection of OpenID Connect 1.0 implementations as well as the setup of the test environment is outlined.
The fifth chapter gives a short overview on the overall evaluation results regarding Service Provider and Identity Provider implementations. Furthermore, the most significant findings among the different products and services that were observed are explained in detail.
The lessons learned including common issue patterns, resulting proposals for new OpenID Connect security considerations and general information about the Responsible Disclosure processes are presented in the sixth chapter.
Finally, in the seventh chapter, a conclusion on this master's thesis is drawn and future work is proposed.

2 Foundations

This chapter focuses on the foundations that are needed to understand the observations and findings in the following chapters.

2.1 JSON

The JavaScript Object Notation (JSON) is a serialization format for data. It was initially developed for ECMAScript (third edition, 1999) but was later derived and standardized as an independent standard in RFC4627 [9].
As per RFC8259, "*JSON can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays)*" [12].
A JSON object consists of key-value pairs. Among these zero or more pairs, the key is a string and the value is either "*string, number, boolean, null, object, or array*" [12]. A non-normative example of a JSON object including possible value types is shown below:

```
1  {
2      "a" :"b",
3      "c" : 1,
4      "d" : null,
5      "e" : {
6          "f" : "g"
7      },
8      "h" : [1, 2, 3],
9      "i" : true
10 }
```

Listing 2.1: An example JSON object including *string*, *number*, *null*, *object*, *array* and *boolean* as values.

In contrast, an array is an ordered set of values in which the values can be from different types:

```
["a", "b", 3, {"d":"e"}]
```

Listing 2.2: An example JSON array including *string*, *number* and *object* as values.

2.1.1 JWT, JWS and JWE

JSON Web Token (JWT) is a format to transfer data in an url-safe manner that was standardized in RFC7519. It defines a JSON structure that is the payload of a JSON Web Signature (JWS) or JSON Web Encryption (JWE) structure [24]. Therefore, a JWT can be cryptographically secured using encryption and/or signatures.

The JOSE header of the JWT structure holds general information about the encoding of the structure, including the algorithms used ("alg" claim) for integrity and confidentiality protection as well as key references.

```
{
    "typ" : "JWT",
    "alg" : "HS256"
}
```

Listing 2.3: An example JOSE header declaring that it belongs to a JWT structure that is a JWS using the HMAC SHA-256 algorithm for MAC-generation.

Specific and standardized JWT contents are part of the payload, including the following claims:

- **iss**: The issuer claim holds information on the entity that issued the JSON Web Token. The "iss" claim is a case sensitive string.
- **sub**: The subject claim holds information on the entity that is accountable for the JSON Web Token. The "sub" claim is a case sensitive string.
- **aud**: The audience claim holds information on the entity that should receive the JSON Web Token. The "aud" claim is a case sensitive string.
- **exp**, **iat**, **nbf**: These claims hold information on the freshness of the JSON Web Token. A JWT is neither valid before its "nbf" claim nor after its "exp" claim and was issued at the time provided via "iat". The claims are encoded as numbers, containing an UNIX Time Stamp value.

```
{
    "iss" : "Joe",
    "sub" : "Alice",
    "aud" : "Bob",
    "iat" : 821404800
}
```

Listing 2.4: An example JWT structure (only decoded payload) that was issued by Joe for Alice on 12th January 1996 that is intended for Bob.

2.2 Single Sign-On (SSO)

This section covers the general concept of Single Sign-On mechanisms. After this high-level introduction, specific protocols are introduced and their relationships are explained.

2.2.1 General Concept

Users frequently have to authenticate at a growing amount of services and websites. In a traditional setup, this could be implemented using dedicated accounts per service.

"Single Sign-On" protocols address this by adding a trusted third party to this setup, which is responsible for user management. This third party is often called "Identity Provider", in contrast to the "Service Provider" that offers functionality for End-Users. Identity and Service Providers can either communicate using a direct server-to-server connection that is called the "back-channel" or using an indirect channel through the user's User Agent that is called "front-channel", as shown in Figure 2.1.

The front-channel is in most cases implemented by redirecting the user's User Agent and passing arguments along, as shown in Figure 2.2.

The trusted third party can provide different information to a service. One key difference lies in the distinction between *authentication* and *authorization*. In an *authentication* scenario the Service Provider tries to determine the identity of an entity, e.g. to create an account that is linked to this identity or to log-in an existing user. In contrast in an *authorization* scenario, the Service Provider does not necessarily get information about the identity of an entity but is authorized to perform privileged actions on behalf of that entity.

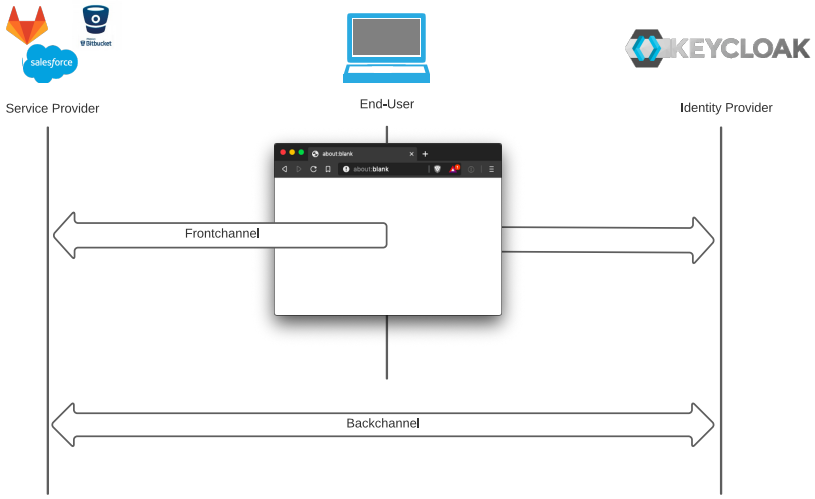

Figure 2.1: Idealized Single Sign-On scenario with front- and back-channel.

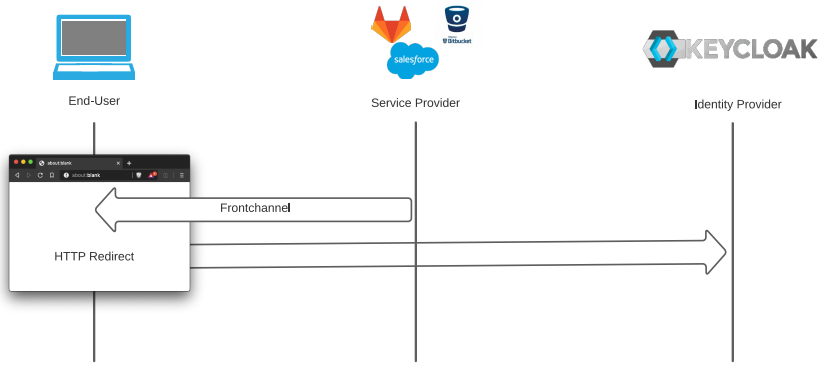

Figure 2.2: Single Sign-On front-channel communication using redirect.

2.2.2 OAuth 2.0

The OAuth 2.0 Authorization Framework was standardized in RFC 6749 in 2012 [11]. It defines the following four roles:

- **Resource Owner.** An entity or person who can grant access to a resource.
- **Resource Server.** The server that hosts the resource. It implements access control and accepts `access_token`s for authorization.
- **Client.** The application that is granted access to the protected resource. The client performs authorized requests on behalf of the Resource Owner.
- **Authorization Server.** The server that authenticates the Resource Owner and issues `access_token`s.

Within the protocol the following two Authorization Server endpoints are defined:

- **Authorization Endpoint.** This endpoint is used by the client to obtain authorization. The Resource Owner is redirected to this endpoint via his User Agent. The Authorization Server then authenticates the Resource Owner, the exact implementation (e.g. authentication using credentials or an active session) is not specified. After successful authentication, the Resource Owner is redirected to the Client's *Redirection Endpoint*.
- **Token Endpoint.** This endpoint is used by the Client to exchange a previously granted authorization grant (e.g. `code`) for an `access_token`. The bearer token can then be used by "*Any party in possession of a bearer token*" to "*get access to the associated resources*" [23]. To obtain the token, *Confidential Clients* authenticate themselves using Client Credentials, *Public Clients* do not possess a `client_secret`.

Additionally, one Client endpoint is defined:

- **Redirection Endpoint.** After successful authentication, the Authorization Server returns the authorization credentials to this endpoint via the Resource Owner's User Agent.

2.2.2.1 Code Flow

The Authorization Code Grant is a redirect-based flow that was initially "*optimized for confidential clients*" [11, Section 4.1.].

In Figure 2.3, an idealized protocol flow is given. The flow consists of the following steps:

1. **Request Resource.** The initial step is non-normative. The End-User requests a resource for which the Client needs further authorization.
2. **Authorization Request.** To start the OAuth 2.0 authorization flow, the Client redirects the End-User to the *Authorization Endpoint* of the Authorization Server. In doing so, it must include `response_type=code` and the `client_id` as parameters. It is recommended to include an additional `state` parameter to prevent Cross-Site-Request-Forgery issues. Optionally a `scope` and a `redirect_uri` can be submitted.
3. **Authorization Response.** After the non-normative authentication of the Resource Owner at the Authorization Server, the End-User is redirected to the *Redirection Endpoint* of the Client. In this redirect, the Authorization Server includes the `code` parameter. If the Client includes a `state` within its `Authentication Request`, this values is also submitted as CSRF protection.

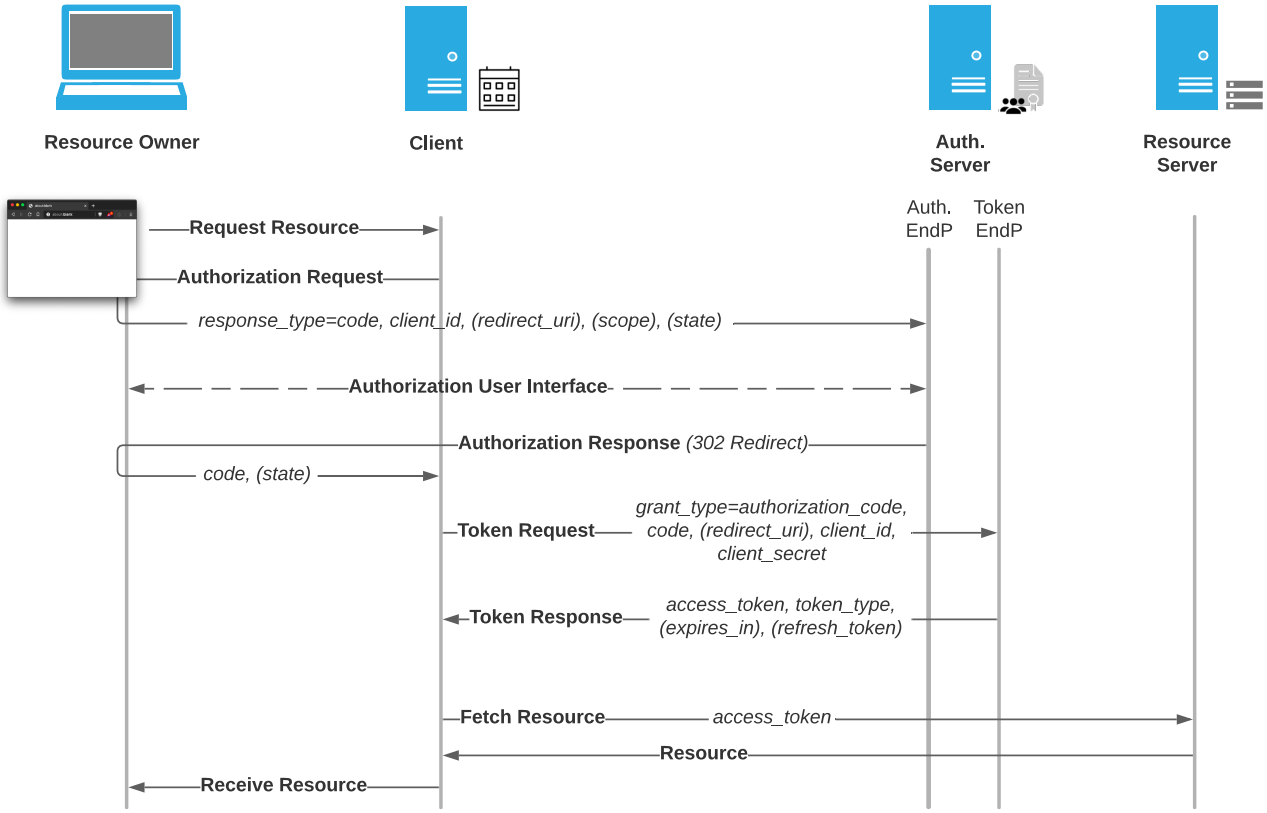

Figure 2.3: Protocol Flow: OAuth 2.0 Authorization Code Flow including most significant parameters.

4. **Token Request.** After receiving the `code` and validating the `state`, the Client redeems the `code` at the *Token Endpoint* of the Authorization Server. It includes `grant_type=authorization_code`, the `code`, the `client_id` and the `redirect_uri` (if included within the `Authorization Request`) in this request.

5. **Token Response.** Within the `Token Response`, the Authorization Server transfers the `access_token`, the `token_type` and optionally a `refresh_token` and an `expires_in` parameter to the Client.

6. **Resource Request.** After receiving the bearer `access_token` that grants the privileges to request resources on behalf of the Resource Owner, the Client requests the resource from the Resource Server.

7. **Resource Response.** If a valid and authorized `access_token` was provided, the Resource Server responds with the requested resource.

8. **Receive Resource.** Finally, the Client responds with the initially requested resource.

2.2.2.2 Implicit Flow

The Implicit Grant is a redirect-based flow that was initially intended for public clients that can not store Client Credentials confidently. As within the flow `access_token`s are transferred through the front-channel, "*clients SHOULD NOT use the implicit grant (response type "token") or other response types issuing access tokens in the authorization response, unless access token injection in the authorization response is prevented and the aforementioned token leakage vectors are mitigated*" [48, Section 2.1.2.].

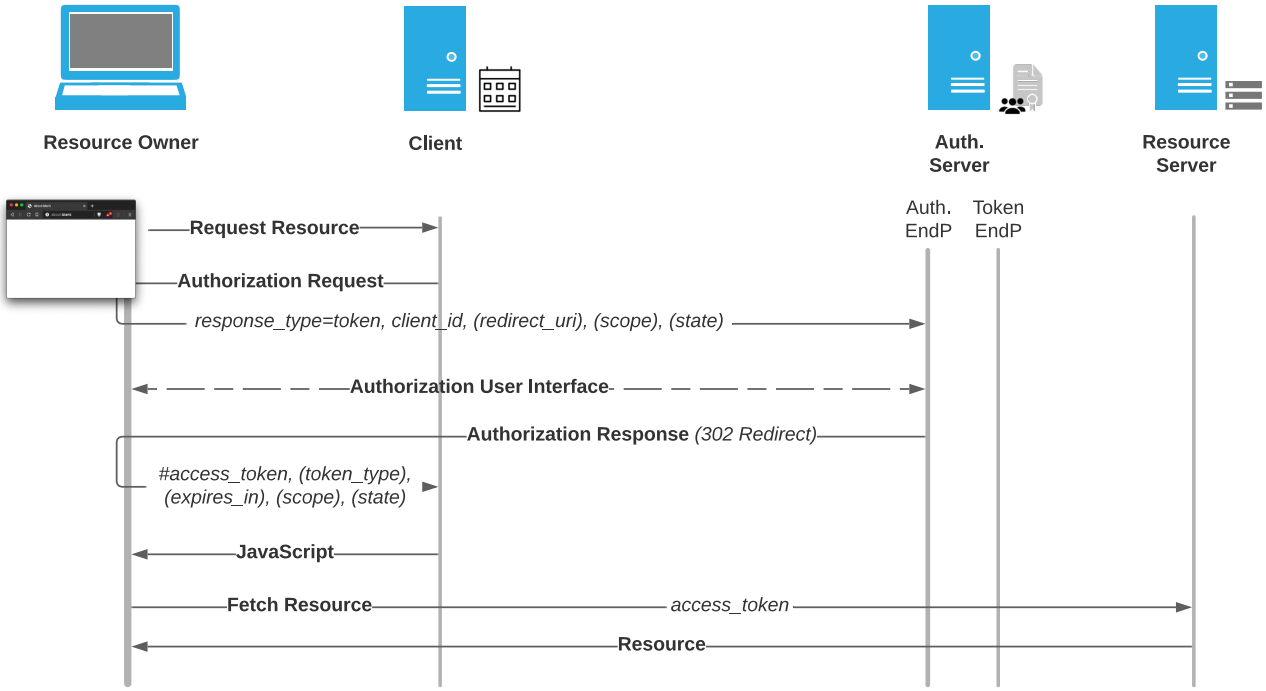

Figure 2.4: Protocol Flow: OAuth 2.0 Authorization Implicit Flow including most significant parameters.

The Implicit Flow differs from the Authorization Code Flow in the following requests:

1. **Authorization Response.** After the non-normative authentication of the Resource Owner at the Authorization Server, the End-User is redirected to the *Redirection Endpoint* of the Client. In this redirect, the Authorization Server includes within the fragment (`#`) the `access_token`, the `token_type` and the `state` (if used in `Authorization Request`). Optionally, the `scope` parameter can be included. Specifying the `expires_in` value is recommended.
   User Agents omit the fragment of URLs during redirection, so that the request without the fragment is received at the Client's *Redirection Endpoint*.

2. **Script.** In response to the request received at the *Redirection Endpoint*, the Client sends JavaScript to the Resource Owner's User Agent. The JavaScript has access to the fragment and extracts the `access_token`.

3. **Fetch Resource.** After receiving the bearer `access_token` that grants the privileges to request resources on behalf of the Resource Owner, the Client requests the resource from the Resource Server.

4. **Receive Resource.** Finally, the Resource Server responds with the initially requested resource.

2.2.2.3 Refresh Flow

An Authorization Server can grant `refresh_token`s which are normally long-living tokens in contrast to the rather short-living `access_token`s. A `refresh_token` can be redeemed against a fresh `access_token` within the Refresh Flow. To obtain the `access_token`, a Client sends a `Refresh Request` to the Authorization Server's *Token Endpoint* including `grant_type=refresh_token` and the `refresh_token` value. Optionally, a `scope` parameter can be included within the `Refresh Request` [11, Section 1.5.].

2.2.2.4 Resource Owner Password Credentials and Client Credentials

If the Resource Owner Password Credentials Grant is used, the Client obtains the Resource Owner's credentials in order to request an `access_token`. According to the specification, the grant type should only be used "*in cases where the Resource Owner has a trust relationship with the client*" [11, Section 4.3.]. As of 2020, this has been updated to "*The Resource Owner password credentials grant MUST NOT be used*" within the OAuth 2.0 Security Best Current Practices [48, Section 2.4.].

The Client Credentials Grant is used, if the Client requests authorization for resources under its control. Therefore, it is sufficient to send an authenticated `Access Token Request` using Client Credentials and `grant_type=client_credentials` to obtain an `access_token`.

2.2.3 OpenID Connect 1.0

OpenID Connect 1.0 was specified 2014 as "*a simple identity layer on top of the OAuth 2.0 protocol*" [33]. The OpenID Connect protocol consists of the following three entities:

- **End-User.** A person that wants to authenticate at a Relying Party. In terms of OAuth 2.0, this is the Resource Owner.
- **OpenID Provider.** The Identity Provider that authenticates the End-User for a Relying Party. In terms of OAuth 2.0, this entity unites the Authorization Server and the Resource Server.
- **Relying Party.** The Service Provider that wants to authenticate an End-User. In terms of OAuth 2.0, this is the Client application.

Beside the *Token Endpoint* and the *Authentication Endpoint* that use previously specified endpoints within the OAuth 2.0 Authorization Framework, the *UserInfo Endpoint* that utilizes the `access_token` for authorization is introduced with OpenID Connect 1.0. In terms of OAuth 2.0, this is the mechanism used for authorization at the Resource Server.

2.2.3.1 ID Token

The most significant extension to the OAuth 2.0 Authorization Framework that is introduced in OpenID Connect 1.0 is the `id_token` [33, Section 2.]. The token is represented as JSON Web Token (JWT) and contains claims about the End-User that authenticates at a Relying Party using an external OpenID Provider. It is issued by the OpenID Provider. The `id_token` must include the following claims:

- **iss.** The *issuer* of the Authentication Response. This claim must be a case sensitive URL using *https* and including host, scheme, path and optionally port. It must not include query or fragment components.
- **sub.** The *subject* of the JWT. This claim must hold a locally unique identifier for the End-User.
- **aud.** The *audience(s)* for which the JWT is intended to. This claim must contain the OAuth 2.0 `client_id` of the Relying Party.
- **exp.** The *expiration time* after which the token must not be accepted.
- **iat.** The time at which the JWT was issued.
- **nonce.** If a `nonce` was included within the `Authentication Request`, this value must be included as "nonce" claim within the `id_token`.

The following claims are optional and may be included:

- **auth_time.** This claim holds the time when the End-User was authenticated.

- **acr.** This value holds the *Authentication Context Class Reference*, which indicates what requirements of ISO 29115 were met during End-User authentication.
- **amr.** The *Authentication Methods References* JSON array indicates which authentication methods were used during authentication, for example *pwd* (password), *face* (facial recognition), *fpt* (fingerprint) or *mfa* (multi-factor authentication).
- **azp.** The *Authorized Party* to which the `id_token` was issued. If present, it must contain the `client_id` of this entity.

2.2.3.2 Code Flow

The Code Flow is based on the OAuth 2.0 Authorization Code Flow. In the following, differences and additions within the requests being performed in an OpenID Connect Code Flow are outlined.

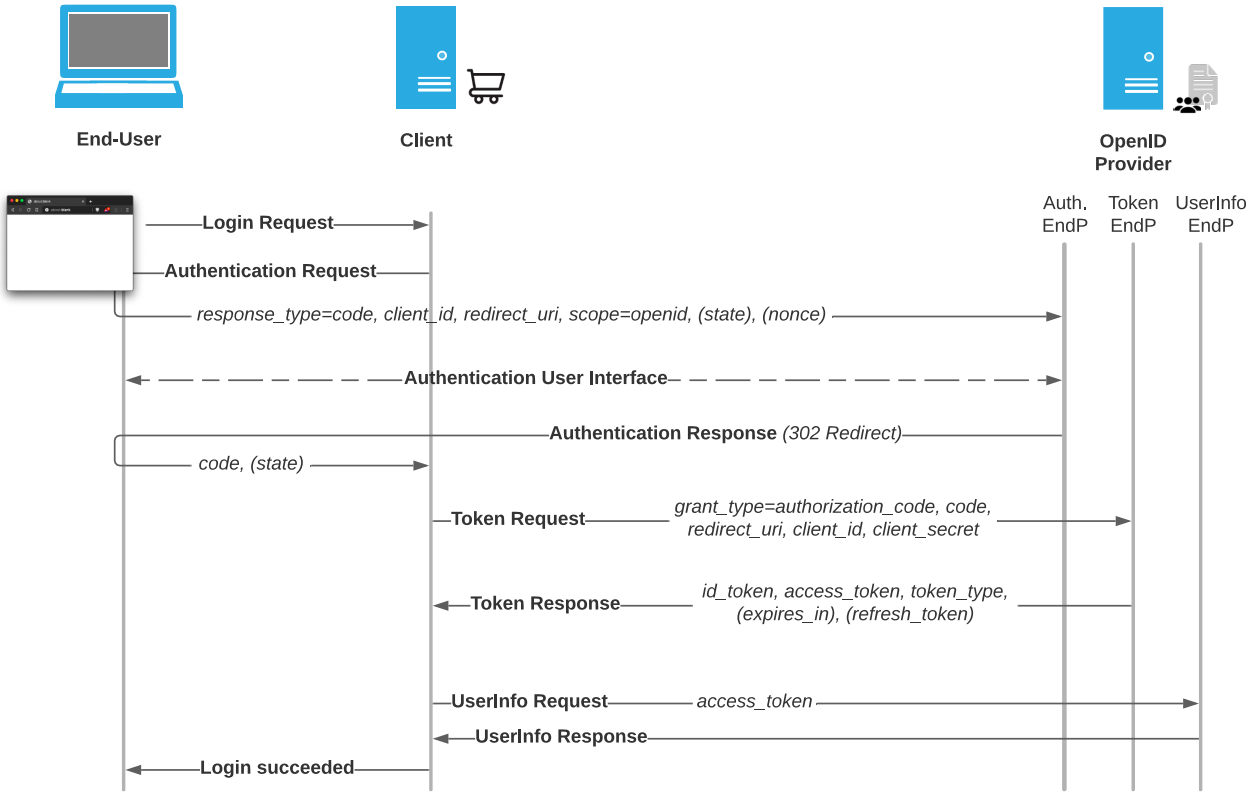

Figure 2.5: Protocol Flow: OpenID Connect Authorization Code Flow including most significant parameters.

1. **Authentication Request.** To start the OpenID Connect authentication flow, the client redirects the End-User to the *Authentication Endpoint* of the OpenID Provider. It must include `response_type=code`, the `client_id`, `scope=openid` and the `redirect_uri`. It is recommended to include an additional `state` parameter to prevent Cross-Site-Request-Forgery (CSRF) issues. There are further optional parameters like `nonce`, `display`, `prompt`, `max_age`, `ui_locales`, `id_token_hint`, `login_hint` and `acr_values`.
2. **Authentication Response.** After the non-normative authentication of the End-User at the OpenID Provider, the End-User is redirected to the *Redirection Endpoint* of the Relying Party. In this redirect, the OpenID Provider includes the `code` parameter. If the Relying Party included a `state` within its `Authentication Request`, this values is also submitted as CSRF protection.
3. **Token Request.** After receiving the `code` and validating the `state`, the Client redeems the `code` at the *Token Endpoint* of the OpenID Provider. It includes `grant_type=authorization_code`, the `code`, the `client_id` and the `redirect_uri` in this request.
4. **Token Response.** Within the `Token Response`, the OpenID Provider transfers the `id_token`, the `access_token`, the `token_type` and optionally a `refresh_token` and an `expires_in` parameter to the Relying Party.
5. **UserInfo Request.** After receiving the bearer `access_token` that grants the privileges to request additional UserInfo for the End-User, the Relying Party sends a request to the *UserInfo Endpoint* of the OpenID Provider.
6. **UserInfo Response.** If a valid and authorized `access_token` was provided, the OpenID Provider responds with the `UserInfo Response`.

2.2.3.3 Implicit and Hybrid Flow

Beside the Code Flow, OpenID Connect offers an Implicit Flow that is based on the OAuth 2.0 Implicit Authorization Grant and a Hybrid Flow combining the Implicit and Code Flow.

For OpenID Connect, flows that transfer `access_token`s through the front-channel are considered deprecated just like for OAuth 2.0. This applies to the Implicit Flow as well as to Hybrid Flow variants which use `response_type=code token` or `response_type=code id_token token`.
As a result, only the Hybrid Flow with `code` and `id_token` within the front-channel is not considered deprecated.

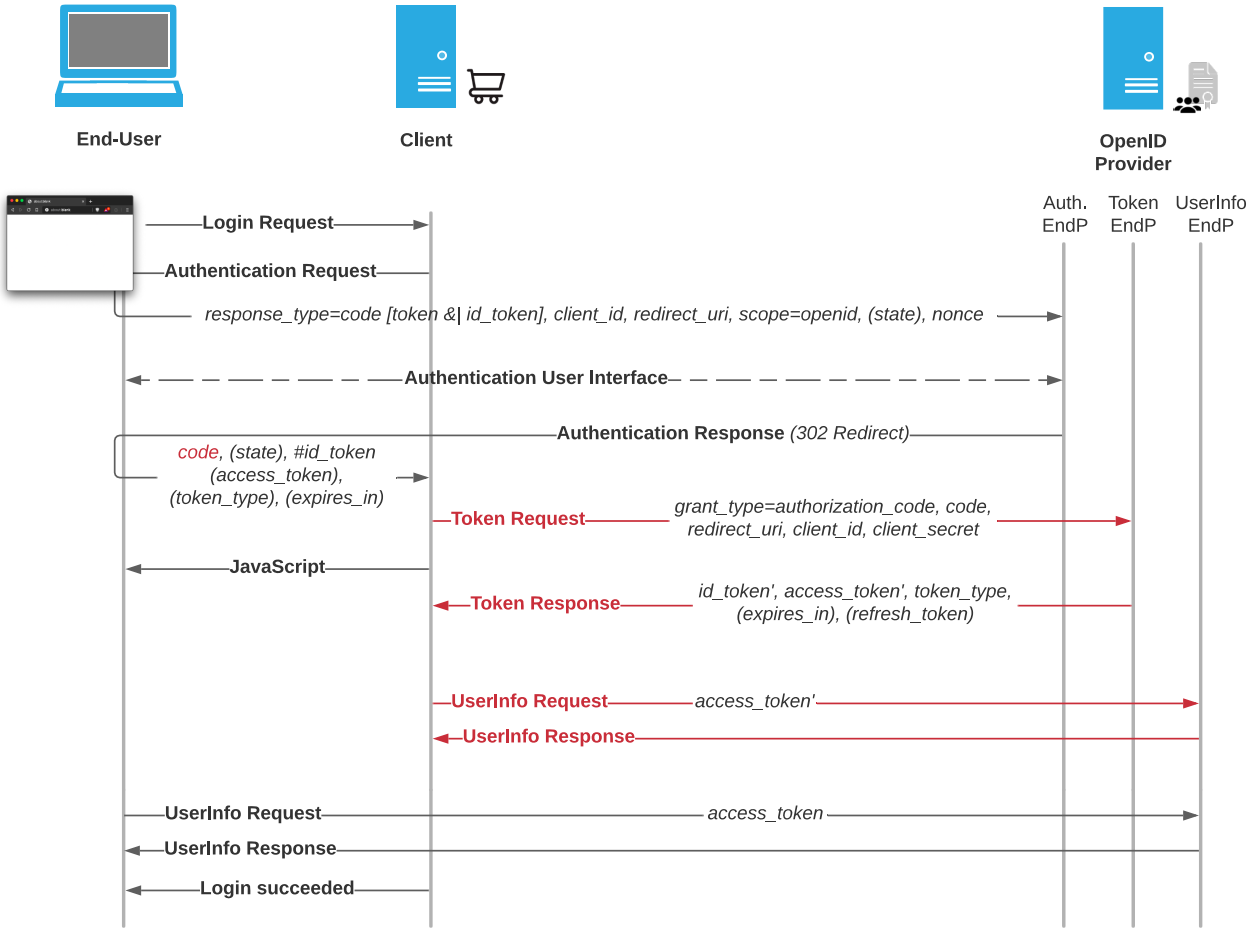

Figure 2.6: Protocol Flow: OpenID Connect Implicit and Hybrid Flow including most significant parameters.

The protocol flow in Figure 2.6 includes the OpenID Connect Implicit Flow and additional requests that are only performed if the Hybrid Flow is used, which are marked in red. In the following, differences to the OpenID Connect Code Flow are outlined:

1. **Authorization Response.** After the non-normative authentication of the End-User at the OpenID Provider, the End-User is redirected to the *Redirection Endpoint* of the Relying Party. In this redirect, the OpenID Provider includes depending on the `response_type` within the `Authentication Request` a fragment (`#`) including the `id_token`, the `access_token`, the `token_type` and the `state` (if used in `Authentication Request`), as well as a `code` value as query parameter if the Hybrid Flow is used. Optionally, the `scope` parameter can be included, specifying the `expires_in` values is recommended.
   User Agents omit the fragment of URLs during redirection, so that the request

without the fragment but including the `code` (Hybrid Flow only) is received at the Client's *Redirection Endpoint*.

2. **Script.** In response to the request received at the `Redirection Endpoint`, the Client sends JavaScript to the End-User's User Agent. The JavaScript has access to the fragment and extracts the `access_token` and the `id_token` (depending on the `response_type`).

3. **Token Request.** Hybrid Flow only: After receiving the `code` and validating the `state`, the Relying Party redeems the `code` at the *Token Endpoint* of the OpenID Provider. It includes `grant_type=authorization_code`, the `code`, the `client_id`, the `client_secret` and the `redirect_uri` in this request.

4. **Token Response.** Hybrid Flow only: Within the `Token Response`, the OpenID Provider transfers the `id_token`, `access_token'`, the `token_type` and optionally a `refresh_token` and an `expires_in` parameter to the Relying Party.

5. **UserInfo Request.** If the Relying Party received the bearer `access_token` that grants the privileges to request resources on behalf of the Resource Owner within the front-channel, it requests additional information about the End-User from the *UserInfo Endpoint*. In Hybrid Flow, the `access_token'` is used to query the *UserInfo Endpoint* through the back-channel.

2.2.3.4 OpenID Connect Discovery 1.0

With the OpenID Connect Discovery 1.0 specification, the OpenID Foundation specifies a "*mechanism for an OpenID Connect Relying Party to discover the End-User's OpenID Provider and obtain information needed to interact with it*" [34].
If a Service Provider performs the discovery, it obtains the `OpenID Provider Metadata` by sending an `OpenID Provider Configuration Request` to the Identity Provider's *Configuration Endpoint* at `<issuer>/.well-known/openid-configuration`. In response, the Identity Provider serves its configuration as JSON array via the `OpenID Provider Configuration Response`. A non-normative `Configuration Response` is given in Listing 2.5.

2.2.3.5 OpenID Connect Dynamic Client Registration 1.0

With the OpenID Connect Dynamic Client Registration 1.0 specification, the OpenID Foundation specifies, "*how an OpenID Connect Relying Party can dynamically register with the End-User's OpenID Provider*" [35].
There are two additional endpoints to the core specification defined within this specification, the *Client Registration Endpoint* and the *Client Configuration Endpoint*. Using a `Client Registration Request`, a Client can register itself at the Identity

```
{
    "issuer":"https://example.com/",
    "authorization_endpoint":"https://example.com/auth",
    "token_endpoint":"https://example.com/token",
    "userinfo_endpoint":"https://example.com/userinfo",
    "jwks_uri":"https://example.com/jwks",
    "registration_endpoint":"https://example.com/register",
    "response_types_supported":["code","token id_token"],
    "subject_types_supported":["public","pairwise"],
    "id_token_signing_alg_values_supported":["RS256"]
}
```

Listing 2.5: OpenID Connect Discovery: Non-Normative `Configuration Response`.

Provider's *Registration Endpoint*.
An already registered Client can determine its configuration at the Identity Provider by sending a `Client Read Request` to the *Configuration Endpoint*.

2.3 Single Sign-On Security

In the following, based on the OAuth 2.0 and OpenID Connect 1.0 specification [11][33], the OAuth 2.0 Security Best Current Practices [48] and previous research, assertions for secure OpenID Connect Implementations are defined. References for the assertions are given and possible attacks that can be performed if the assertions are not met are sketched.

2.3.1 Identity Provider Security

The Identity Provider (IdP), OpenID Provider (OP), or in the context of OAuth 2.0 Authorization Server (AS) has to take care of the following aspects to reliably authenticate End-Users to Relying Parties.

2.3.1.1 Redirect URI Protection

The `redirect_uri` validation is crucial for the security of an OpenID Connect Identity Provider implementation. The OpenID Connect specification requires, that the

provided value within the `Authentication Request` "*MUST exactly match one of the Redirection URI values for the Client pre-registered at the OpenID Provider*" [33, Section 3.1.2.1.]. The comparison must be performed using "Simple String Comparison".
On the other hand, the specification is quite vague regarding the allowed schemes for `redirect_uri`s. *Https* should be used, but "*The Redirection URI MAY use an alternate scheme, such as one that is intended to identify a callback into a native application*" [33, Section 3.1.2.1.]. This allows potentially dangerous schemes like `javascript`, `vbscript` and `data`.

IdP-1-1. The `redirect_uri` within the `Authentication Request` must be compared against the pre-registered URIs using "Simple String Comparison".
IdP-1-2. The `redirect_uri` must not allow dangerous schemes like `javascript`, `vbscript` or `data`.

2.3.1.2 HTTP

According to previous research, OpenID Connect "*explicitly allows for any redirection method to be used for the redirection*" [15, Section III; A. 5)]. This would allow to use the HTTP 307 redirect, which only slightly differs from a 302 redirect but "*guarantees that the method and the body will not be changed when the redirected request is made*" [32].
As a result, if the Identity Provider receives the POST request to its *Login Endpoint* including the End-User's credentials and immediately responds with a status code 307, the User Agent performs the redirect and includes the End-User's Request Body containing the credentials within the request to the Service Provider.

IdP-2-1. The `Authentication Response` redirect to the `redirect_uri` must not utilize the HTTP 307 status code.

2.3.1.3 Request Object, request_uri and Registration Object

The Request Object is a possibility to pass OpenID Connect requests "*in a single, self-contained parameter [...] optionally signed and/or encrypted*" [33, Section 6.].
As this value is a JWT, a secure signing algorithm should be used. In addition, "*response_type and client_id parameters MUST be included using the OAuth 2.0 request syntax, since they are REQUIRED by OAuth 2.0*" [33, Section 6.1.].
The Request Object can be passed along the `Authentication Request` by value or by reference using the `request_uri` parameter. If this parameter is supported, the Identity Provider fetches the external resource to obtain the Request Object [33, Section 6.2.].

The specification defines the `registration` parameter that is intended for Self-Issued OpenID Providers. The parameter "*SHOULD NOT be used when the OP is*

not a Self-Issued OP" [33, Section 7.2.1.]. If the parameter is considered in regular `Authentication Requests`, pre-configured values could be overwritten.

IdP-3-1. The `request_uri` parameter should be disabled by default. If it is enabled, the Identity Provider must enforce the `request_uris` whitelist for `request_uri` values.
IdP-3-2. The signature validation of the `Request Object` must enforce a secure signing algorithm, therefore the "none" algorithm must not be accepted.
IdP-3-3. The `Request Object` must include all required parameters.
IdP-3-4. The `registration` parameter within the `Authentication Request` must only be considered if the OP is self-issued.

2.3.1.4 Access-Token, Refresh-Token and Client Credentials Protection

Token and Client Credentials are valuable secrets within an OpenID Connect setup, as they are normally rather long-living and inherit privileges. Therefore, secure storage of these secrets is crucial and "*credential storage protection best practices*" [49, Section 5.1.4.1.] need to be applied.
Client Credentials need to be strong, "*for instance, for HS256, the client_secret value MUST contain at least 32 octets (and almost certainly SHOULD contain more*" [33, Section 16.19.]. Otherwise, the *Token Endpoint* could act as an oracle for Client Credentials, if no rate limiting is applied and weak client secrets can be chosen [11, Section 2.3.1.].
The `access_token` should expire after a reasonable timeout and have a limited scope to reduce the impact of leakage. We consider a lifespan of 60 minutes as reasonable for our evaluation. The `refresh_token` values normally have a longer lifespan, but should expire in case of security events (for instance if the password is changed) [48, Section 4.12.]. Additionally, the `refresh_token` must be bound to a `client_id` and must be either rotated or sender constrained [48, Section 4.12.].
A `Cache-Control` header misconfiguration can lead to access-control restricted resources like the `UserInfo Response` to be cached within HTTP proxies or the browser cache. Therefore, responses to requests that use the `access_token` for authorization "*should send a Cache-Control header containing the "no-store" option*" [49, Section 4.6.6.].
Finally, End-Users should have control over given permissions to Service Providers and should be able to revoke `access_token`s and `refresh_token`s at the Identity Provider.

IdP-4-1. The storage of sensitive OpenID Connect parameters must follow current credential storage best practices.
IdP-4-2. The Identity Provider must enforce that Clients use sufficiently strong `client_secret`s and should implement rate limiting at the `Token Endpoint`.

IdP-4-3. The `access_token` should not have a life-span longer than 60 minutes and have a limited scope. The `refresh_token` should be invalidated in case of security events.
IdP-4-4. The responses to requests that used the `access_token` for authorization must set the `Cache-Control` HTTP header to "no-store".
IdP-4-5. The End-User should be able to revoke tokens at the Identity Provider.

2.3.1.5 Code Protection

Within the `Token Request`, Client Authentication using a `client_secret` is optional, according to the specification, because in the context of *public* Clients this value can not be kept secret. For this reason, some implementations allow to distinguish between *public* and *confidential* Clients. Wherever applicable, Client Authentication must be used and enforced by the Identity Provider when `code` values are redeemed.
The `code` is a short-living (we consider a lifespan of 10 minutes as reasonable for our evaluation) and one-time-usable sufficiently randomly chosen value. It must be bound to the `client_id` and the exact `redirect_uri` it was issued for. Furthermore, the `code` must be bound to the `nonce` that is returned within the `id_token` that is sent in response to the `Token Request` containing the `code` [48, Section 2.1.2.].
Finally, the `code` value must be immediately invalidated on all potentially distributed hosts if redeemed once. Otherwise, race conditions can occur that lead to multiple `access_token` values being returned for one `code` [30]. If the End-User revokes the permissions previously granted to a Service Provider, all tokens need to be revoked, including `code` values that could be redeemed for fresh `access_token`s otherwise [30].

IdP-5-1. Client Authentication using Client Credentials must be enforced.
IdP-5-2. The `code` must be one-time-usable, short-living (max 10 minutes lifespan) and sufficiently random.
IdP-5-3. The `code` must be bound to the `client_id` and the exact `redirect_uri` it was issued for.
IdP-5-4. The `code` must be bound to the `nonce` that is included within the `Authentication Request`, the `nonce` included within the `id_token` must be equal to the previously chosen value.
IdP-5-5. The `code` must be immediately invalidated after it was used once to prevent race conditions.
IdP-5-6. If the End-User revokes the permissions granted to a Client, valid `code` values that were previously obtained by a Client must be invalidated.

2.3.1.6 Proof Key for Code Exchange

Although the *Proof Key for Code Exchange* (PKCE) was initially designed to protect native applications, the OAuth 2.0 Security Best Current Practice Draft states that "*Authorization servers MUST support PKCE*". Furthermore, "*Authorization servers MUST provide a way to detect their support for PKCE*" [48, Section 2.1.].

IdP-6-1. PKCE must be supported.
IdP-6-2. The OpenID Provider must provide a way to detect its PKCE support.

2.3.1.7 Audience Confusion

In order to prevent confusion, "*Authorization servers SHOULD NOT allow clients to influence their "client_id" or "sub" value or any other claim if that can cause confusion with a genuine Resource Owner*" [48, Section 2.6.].

IdP-7-1. Identity Providers should not allow Clients to influence their `client_id` or "sub" claim.

2.3.1.8 Consent Screen

IdP-8-1. The requested permissions that are presented on the Consent Screen must match the permission-set that is actually applied to the `access_token` that is returned to the Service Provider.

2.3.1.9 Dynamic Client Registration

If OpenID Connect Dynamic Client Registration is implemented, some kind of access-control should be applied before clients could register, for instance using Initial Access Tokens or Host/IP restrictions [35, Section 3.]. Alternatively, any client that registers without further authentication may require manual review to be enabled.
If it is implemented, the `sector_identifier_uri` must only support *https* [35, Section 5.].
The Client Metadata can contain various URIs. The Identity Provider must prevent SSRF vulnerabilities as well as injection issues that could for instance occur if external resources specified during registration are embedded on the Login or Consent Screen.

IdP-9-1. Dynamic Client Registration should be restricted by some kind of access-control.

IdP-9-2. The `sector_identifier_uri` must only support *https*.
IdP-9-3. The URIs present within the Client Metadata must be handled carefully to prevent injection and SSRF issues.

2.3.1.10 Client Authentication

There are multiple different Client Authentication mechanisms that can be used to authenticate Service Providers at the *Token Endpoint*, including the two JWT based mechanisms "client_secret_jwt" and "private_key_jwt".

IdP-10-1. If JWT based Client Authentication mechanisms are used, the JWT must be validated regarding required claims specified within the OpenID Connect 1.0 specification [33, Section 3.1.3.7.].
IdP-10-2. If JWT based Client Authentication mechanisms are used, the signature of the JWT must be validated. In doing so, the "none" algorithm must be explicitly forbidden. The key that is used for the signature validation must be chosen using the `client_id` of the Client.
IdP-10-3. If JWT based Client Authentication mechanisms are used, the freshness of the JWT must be validated. Therefore, the "iat", "exp", "nbf" and "jti" claims need to be checked.

2.3.2 Identity Provider Attacks and Flaws

In the following, Identity Provider related attacks and flaws are described in detail.

2.3.2.1 Injection

Injection flaws are a common issue in web applications. There are various different kinds of injections.

Insufficient Filtering. If an Identity Provider fails to sanitize attacker-controlled contents sufficiently, but this flaw can not be exploited, e.g. for Cross-Site-Scripting (XSS), we still consider this as *Insufficient Filtering*.
If an Identity Provider does not enforce **IdP-1-2** (subsubsection 2.3.1.1), a malicious Service Provider (section 3.3) can choose `redirect_uri` values with an arbitrary scheme. This allows potentially dangerous schemes like `javascript`, `vbscript` and `data`. If an implementation allows registering `redirect_uri`s that use dangerous schemes, we consider this a security-relevant finding. Depending on the User Agent's behavior, this can lead to Cross-Site-Scripting (XSS) vulnerabilities. Even if JavaScript is executed using a `data` URI, which results in script execution within the *null origin* in recent User Agents, an attacker may be able to launch a Cross-Origin

State Inference (COSI) attack like Sudhodanan et al. pointed out in their paper "*Cross-Origin State Inference (COSI) Attacks: Leaking Web Site States through XS-Leaks*" [46].

Cross-Site-Scripting. If an Identity Provider does not sanitize End-User and Service Provider controlled contents, a malicious Service Provider (section 3.2) or a web attacker (section 3.1) can be able to inject JavaScript that is executed within the context of the Identity Provider in the End-User's User Agent. An example of such a value is the `login_hint` that can be passed along the `Authentication Request`. If this value is not sanitized before it is reflected on the Login Screen, this could lead to reflected XSS. Using XSS, a malicious actor could for instance try to steal a victim's session at the Identity Provider or to bypass Cross-Site-Request-Forgery (CSRF) protections and perform actions on the Identity Povider as victim user.

2.3.2.2 HTTP 307 redirect

If an Identity Provider violates **IdP-2-1** (subsubsection 2.3.1.2) and utilizes an HTTP 307 status code for the redirect to the `redirect_uri` in response to the `Login Request`, the User Agent performs the redirect to the Service Provider and includes the End-User's request body containing the credentials within the request. After obtaining the End-User's credentials that are disclosed within the forwarded POST body, the Service Provider is then able to impersonate the End-User at the Identity Provider, as shown in Figure 2.7.

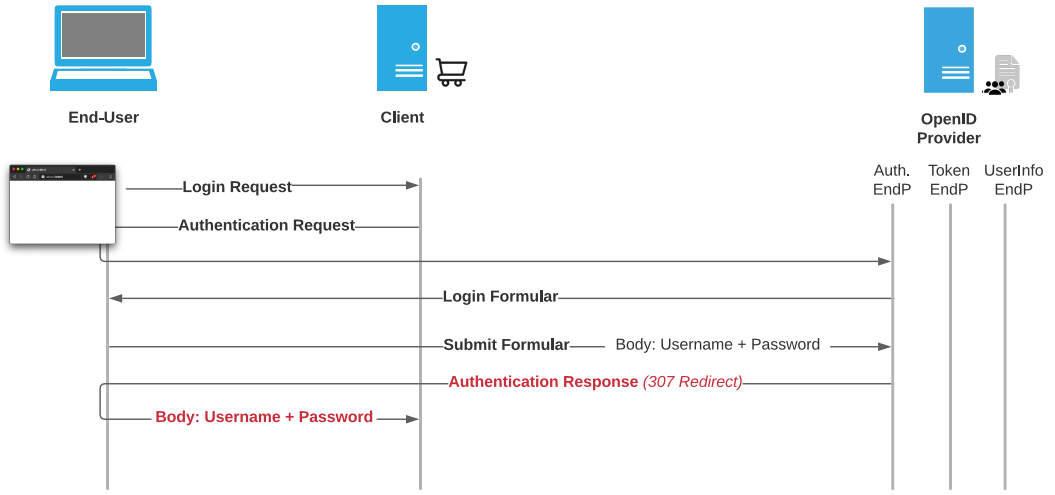

Figure 2.7: HTTP status code 307 Redirect in a Single Sign-On scenario disclosing End-User's credentials.

2.3.2.3 Open Redirect and Token Disclosure

If an Identity Provider violates **IdP-1-1**, (subsubsection 2.3.1.1) and does not check the `redirect_uri` using "Simple String Comparison" against the pre-configured values, this enables an Open Redirect vulnerability [48, Section 4.9.2.][11, Section 4.1.2.1.]. A web attacker (section 3.1) could redirect the victim End-User to the *Authentication Endpoint* including a `redirect_uri` pointing to an attacker-controlled domain. If the victim is authenticated at the Identity Provider, the Identity Provider immediately responds with the `Authentication Response` redirect to the `redirect_uri` including a valid `code` (or `access_token` depending on the OpenID Connect flow that is used) that is sent to the attacker-controlled domain. Besides the risk for phishing that is caused by Open Redirect issues, sensitive OpenID Connect secrets are disclosed to the attacker.

2.3.2.4 Brute Force

An Identity Provider implements multiple endpoints that are used for authentication. If there is no rate limiting implemented on these endpoints and additionally weak secrets can be chosen, this yields the risk of brute force attacks [11, Section 2.3.1.].

User Credentials. If there is no rate limiting at the `Login Endpoint` and End-Users are allowed to choose weak passwords, a web attacker (section 3.1) can launch an online brute force attack against the Identity Provider to guess the End-User's credentials.

Client Credentials. If there is no rate limiting at the `Token Endpoint` and Clients may choose weak `client_secret`s (**IdP-4-2**, subsubsection 2.3.1.4), a malicious web attacker (section 3.1) can launch an online brute force against the Identity Provider to guess the Client's Credentials. Therefore, RFC6749 requires that "*the authorization server MUST protect any endpoint utilizing it against brute force attacks*" [11, Section 2.3.1.].

2.3.2.5 Broken Authentication and Access Control

Code Invalidation. As pointed out in [30], the `code` needs to be invalidated immediately after it was redeemed the first time (**IdP-5-5**, subsubsection 2.3.1.5). Otherwise, race conditions can occur in a decentralized system, resulting in multiple `access_token`s being issued for one `code`.
Likewise pointed out in [30], a common flaw in OAuth 2.0 and OpenID Connect implementations is a missing `code` invalidation after an End-User revokes permissions for a Client (**IdP-5-6**, subsubsection 2.3.1.5). Depending on the actual implementation, if the `code` is not invalidated, a malicious Service Provider (section 3.3)

can at least obtain one fresh `access_token` which holds privileges as long as it is valid. In some implementations, redeeming a valid `code` after the permissions for the Client were revoked even leads to restoring the complete permission-set persistently.

2.3.2.6 Denial-of-Service

Issuing new sufficiently random `code`, `access_token` and `refresh_token` values in Authorization or Refresh Flows can be an exhaustive operation for Identity Providers. Therefore, a reasonable rate limiting should be considered and a sufficient amount of token values should be available in storage to prevent exhaustive generation of new random tokens. If no rate limiting is in place, a web attacker (section 3.1) can launch a Denial-of-Service attack against the Identity Provider [49, Section 4.4.1.11.].

2.3.2.7 Audience Confusion

If an Identity Provider violates **IdP-7-1** (subsubsection 2.3.1.7), a malicious Service Provider (section 3.3) can influence its `client_id` so that it may confuse genuine Service Providers.

2.3.2.8 Server-Side Request Forgery

Request URI. Fett et al. outlined in 2017, that the `request_uri` allows intentional unauthenticated Server-Side Request Forgery (SSRF) [15, Section III; A. 8)]. If this parameter is implemented, the Identity Provider is supposed to fetch the resource to obtain the `Request Object`. As a result, if the Identity Provider violates **IdP-3-1** (subsubsection 2.3.1.3), an unauthenticated web attacker (section 3.1) can launch a SSRF attack against the victim Identity Provider.

2.3.2.9 Sensitive Information Disclosure

Referer Leaks. If an Identity Provider embeds external resources on sensitive endpoints like the *Login Endpoint* and there is no `Referrer-Policy` defined, the `Referer` HTTP header may disclose sensitive OpenID Connect parameters like `state` and `nonce` to third parties.

Unencrypted Communication. If an Identity Provider violates requirements regarding the TLS-enforcement, e.g. **IdP-7-1** (subsubsection 2.3.1.7), sensitive information is transferred in plain text over the network. A man-in-the-middle attacker (section 3.5) can eavesdrop, modify or drop this communication.

2.3.2.10 Signature Manipulation

If the signature validation for the Client Authentication mechanisms using JWTs is not sufficiently implemented and the Identity Provider violates **IdP-10-1** (subsubsection 2.3.1.10), **IdP-10-2** (subsubsection 2.3.1.10) or **IdP-10-3** (subsubsection 2.3.1.10), a web attacker (section 3.1) that obtained a valid `code` can bypass the Client Authentication and can impersonate any Client at the *Token Endpoint* in order to redeem `code` values.

2.3.3 Service Provider Security

The Service Provider (SP) or Relying Party (RP) has a different set of assertions to fulfill than the Identity Provider. It potentially has to keep track of multiple Identity Providers, internal and external user accounts, replayed messages and many more threats covered in the following sections.

2.3.3.1 Replay Protection

In order to prevent `id_token` replay, the freshness of the JWT must be checked. In doing so, the Service Provider must validate the JWT regarding its "exp", "iat", "nbf" and "jti" claims. Additionally, the `nonce` value has to be checked against the value that was set within the `Authentication Request` [33, Section 16.].

SP-1-1. The freshness of the `id_token` must be validated regarding the "exp", "iat", "nbf" and "jti" claims.
SP-1-2. The "nonce" claim within the `id_token` must be checked against the `nonce` passed along the `Authentication Request`.

2.3.3.2 Signature Validation

If the Code Flow is used, validation of the `id_token`'s JWT received through back-channel communication is not mandatory, according to the specification. Nevertheless, the Service Provider should validate the JWT's signature. In doing so, the "none" algorithm should be declined.
As the `id_token` is a JWT, its `JOSE` header can contain references to the key that is used for signature validation. This information could be specified using the following headers: "kid", "jku", "jwk", "x5u" or "x5c". These parameters should not be interpreted by Service Providers to prevent Key Confusion [26, Section 3.1.5].
If the Service Provider supports the OpenID Connect Discovery, a malicious Identity Provider (section 3.2) can try to overwrite configurations of existing benign Identity

Providers, e.g. by claiming the same `client_id` [26, Section 3.1.4].
Finally, the Service Provider needs to keep track of the signature and MAC algorithms for which each key is intended for. For instance, asymmetric keys must not be used for symmetric algorithms [26].

SP-2-1. The Service Provider should validate the signature of `id_token`s it receives via back-channel communication.
SP-2-2. The Service Provider must not accept the "none" algorithm as signature algorithm for `id_token`s.
SP-2-3. The Service Provider must not interpret external references to keys within the JWT's JOSE header.
SP-2-4. The Service Provider must strictly restrict the usage of keys to their intended algorithm, for instance, RSA keys must not be used for symmetric algorithms like HMAC SHA-256.

2.3.3.3 Token Recipient Validation

SP-3-1. The "aud" claim within the `id_token` must contain the `client_id` of the Relying Party.
SP-3-2. The "azp" claim within the `id_token` must contain the `client_id` of the party the `id_token` was issued to.

2.3.3.4 ID Validation

During the `id_token` validation, the Service Provider needs to pay additional attention to the "iss" claim. If this claim is not sufficiently checked, a malicious Identity Provider (section 3.2) could confuse the benign Service Provider [26, Section 3.1.4].
If the Service Provider uses other claims than the "sub" claim to map the external user to an internal account, these non-normative claims need to be handled carefully. Especially, the "email" claim that could be added to a JWT "*MUST NOT*" be used "*for anything related to identify the End-user*" [26, Section 3.1.4].
Finally, if OpenID Connect Discovery is supported, the issuer within the discovery "*MUST be identical to the issuer value returned by WebFinger. This also MUST be identical to the iss Claim value in ID Tokens issued from this Issuer*" [34, Section 4.3.].

SP-4-1. The Service Provider needs to validate the URL that is provided as "iss" claim carefully to prevent confusion between multiple Identity Providers.
SP-4-2. The Service Provider must not use the "email" claim for authentication purposes. If non-normative claims are used for authentication, these need to be handled carefully.
SP-4-3. If the Service Provider implements OpenID Discovery, the issuer within

the discovery must be identical to the issuer values present during WebFinger and within `id_token`s.

2.3.3.5 Sub Claim Validation

Because there is no guarantee that the `UserInfo Response` is about the End-User identified within the `id_token`, "*The sub Claim in the UserInfo Response MUST be verified to exactly match the sub Claim in the ID Token; if they do not match, the UserInfo Response values MUST NOT be used*" [33, Section 5.3.2].

SP-5-1. The Service Provider needs to make sure, that the "sub" claim within `id_token` and `UserInfo Response` are equal.

2.3.3.6 State Validation

According to the OpenID Connect 1.0 specification, the `state` value is only recommended but not mandatory. Without the `state` parameter, there is no binding between the `Authentication Request` and the `Authentication Response` containing the `code` that is further used to identify the End-User (by redeeming it for the `id_token` and `access_token`) [15, Section III; A. 2)].
Some applications use the `state` to deliver further application state. In doing so, they must make sure to integrity protect the `state` value to prevent injection issues [48, Section 4.7.1.].
Finally, if the `state` is reusable, this enables multiple attack vectors. The OpenID Connect Core Specification states that the Service Provider "*MAY record the state of the use of the token and check the status for each request*" [33, Section 16.9.] in order to prevent token reuse. The OAuth 2.0 Security Best Current Practices advise to invalidate the `state` after each use, to limit the impact in case the parameter is leaked through Referer headers [48, Section 4.2.].

SP-6-1. The Service Provider must use the `state` parameter or an alternative CSRF protection like PKCE or the `nonce` within the `id_token`.
SP-6-2. If the Service Provider delivers application state within the `state` parameter, the parameter must be integrity protected.
SP-6-3. The `state` parameter must be an one-time-usable value that is bound to the End-User's session.

2.3.3.7 Leakage Protection of OpenID Connect Parameters

SP-7-1. The Service Provider must not embed external resources or anchors to external websites on sensitive OpenID Connect endpoints like the `redirect_uri` endpoint, unless there are additional countermeasures against `Referer` leaks in place, for instance, a sufficient `Referrer-Policy` [48, Section 4.2.1.].

SP-7-2. Sensitive OpenID Connect parameters must not be disclosed in the Service Provider's file system, e.g. in access logs or cache files [48, Section 4.8.2.].
SP-7-3. The `access_token` should not be transferred using the front-channel (within the `Authentication Response`) [48, Section 2.1].
SP-7-4. For communication with the OpenID Connect endpoints, the Service Provider must enforce TLS [11, Section 3.1.2.1.].

2.3.3.8 Cross-Site-Request-Forgery Protection

The specification defines a third party *Login Initiation Endpoint* that serves as intentional Cross-Site-Request-Forgery (CSRF) protection bypass as pointed out in [14]. Fett et al. advise not to implement this endpoint as it is optional. Furthermore, it has been shown that some Identity Providers implement different internally used non-normative endpoints that start an OpenID Connect flow.

SP-8-1. If not explicitly needed for a reasonable use-case, Service Providers should not implement the specified *Login Initiation Endpoint*.
SP-8-2. If the Service Provider internally utilizes a non-normative endpoint that starts the OpenID Connect login flow, this endpoint must be protected against CSRF.

2.3.3.9 Server-Side Request Forgery Protection

In an OpenID Connect scenario, especially the back-channel communication is prone to SSRF by design. Depending on the actual setup, reasonable hardening should be applied to limit the possible impact of SSRF. Otherwise, a malicious Identity Provider (section 3.2) could specify localhost or internal IP addresses as its endpoints using OpenID Connect Discovery and target internal hosts [26, Section 2.1.2].

SP-9-1. To limit the risk resulting from Malicious Endpoint attacks that misuse the OpenID Connect flow by specifying fake-endpoints, the Service Provider should restrict endpoints from using localhost or internal IP addresses as OpenID Connect endpoint.
SP-9-2. Error messages for back-channel communication must not contain sensitive connection related information.

2.3.3.10 Covert Redirect Protection

A common pattern for Service Provider implementations is a non-normative parameter (e.g. `next`, `start`, `redirect` or `nextURL`) that indicates where a user should be redirected after authentication. Some applications redirect the End-User to the

URL present as `Referer` header when the OpenID Connect flow is started, resulting in comparable but less obvious behavior. Additionally, if the Service Provider transfers application state using the `state` parameter, the `state` parameter can also include the redirection target.

SP-10-1. The Service Provider must validate the redirection target after login (specified using `next` URL parameter, last `Referer` header, `state` or any other non-normative method) against a whitelist of trusted hosts.
SP-10-2. The Service Provider must not allow the redirection target after login to refer to local endpoints that have session relevant effects, such as *Logout Endpoint* or *Initiate Login Endpoint*.

2.3.3.11 Deprecated Grants

According to the OAuth 2.0 Security Best Current Practices, "*clients SHOULD NOT use the implicit grant (response type "token") or other response types issuing access tokens in the authorization response, unless access token injection in the authorization response is prevented and the aforementioned token leakage vectors are mitigated*" [48, Section 2.1.2.].
Furthermore, the Resource Owner Password Credentials Grant must not be used, because it "*insecurely exposes the credentials of the Resource Owner to the client*" [48, Section 2.4.].

SP-11-1. The Service Provider should not use the Implicit Flow and Hybrid Flow variants that transfer the `access_token` via front-channel.
SP-11-2. The Service Provider must not use the Resource Owner Password Credentials Grant.

2.3.3.12 Mix-Up Protection

In order to prevent Mix-Up attacks, in which a malicious Identity Provider (section 3.2) tries to obtain an `access_token` or `code` by confusing a Relying Party to send these tokens to the wrong entity, a Relying Party could either use dedicated endpoints per Identity Provider, add an "iss" parameter to the `Authentication Response` or use the Hybrid flow with an `id_token` in the front-channel containing `client_id` and issuer, according to the OAuth 2.0 Security Best Current Practices [48, Section 4.4.].

SP-12-1. If a Service Provider does not implement dedicated endpoints per Identity Provider, it must implement further Mix-Up mitigations.

2.3.3.13 Discovery

The OpenID Connect Discovery or in particular the `Configuration Request` and `Response` have crucial effect on the OpenID Connect Setup. If this communication is not performed over a trusted channel, no trustworthy authentication and authorization can be performed. Therefore, according to the specification, "*TLS certificate checking MUST be performed by the RP [...] when making an OpenID Provider Configuration Request*" [34, Section 7.2].

SP-13-1. The Service Provider must enforce TLS for the communication with the `Configuration Endpoint`.
SP-13-2. The Service Provider must enforce TLS for OpenID Connect endpoints that are provided using OpenID Connect Discovery.

2.3.4 Service Provider Attacks and Flaws

In the following, Service Provider flaws and attacks that are possible if the previously described checks are not fulfilled are sketched. In doing so, the checks that are violated as well as references to previous research are given.

2.3.4.1 Replay

If a Service Provider violates **SP-1-1** (subsubsection 2.3.3.1) and **SP-1-2** (subsubsection 2.3.3.1), it can not determine if a provided `id_token` is fresh or an old `id_token` is replayed by a malicious Identity Provider (section 3.2) or a web attacker (section 3.1) (depending on the context in which the `id_token` is used).

2.3.4.2 Signature Manipulation

Signature Bypass. If a Service Provider violates **SP-2-1** (subsubsection 2.3.3.2) or **SP-2-2** (subsubsection 2.3.3.2) and effectively does not enforce the JWT to be signed, a malicious Identity Provider (section 3.2) could authenticate as an arbitrary End-User.

Key Confusion (I). If a Service Provider violates **SP-2-3** (subsubsection 2.3.3.2), a malicious actor could reference a controlled secret key and therefore provide valid self-signed JWTs with controlled contents. As a result, a malicious Identity Provider could authenticate as an arbitrary End-User [26, Section 3.1.5].

Key Confusion (II). Finally, if a Service Provider violates **SP-2-4** (subsubsection 2.3.3.2), a malicious Identity Provider can confuse the Service Provider regarding the keys that are used in accordance to the specified algorithm and the key

identifier ("kid") claim within the JOSE header. In doing so, the attacker specifies a "kid" of a RSA key pair (asymmetric) but specifies the algorithm as `HS256` (symmetric). If the Service Provider does not restrict the RSA key to be used for asymmetric algorithms and as for signature verification always the public key is used, the attacker could sign the JWT using the known public key (in doing so, the attacker could choose the whole public key in PEM or DER format or parameters of the public key like n or e, depending on the implementation's behavior) [26, Section 3.1.5].

2.3.4.3 Token Recipient Confusion

If the Service Provider accepts `id_token`s that violate **SP-3-1** (subsubsection 2.3.3.3) or **SP-3-2** (subsubsection 2.3.3.3), it may accept `id_token`s that have a valid signature but were issued for another entity.
A malicious Identity Provider (section 3.2) can utilize this to authenticate as victim End-User at the benign Service Provider using an `id_token` that was issued for the victim End-User at another service.

2.3.4.4 ID Spoofing

If the Service Provider violates **SP-4-1** (subsubsection 2.3.3.4), a malicious Identity Provider (section 3.2) could authenticate as any End-User that previously registered using a benign Identity Provider [26, Section 3.1.4].
If the Service Provider uses other claims than the "sub" claim to map the external user to an internal account and therefore violates **SP-4-2** (subsubsection 2.3.3.4) or the "email" claim, a malicious Identity Provider could for instance spoof an End-Users identity by setting the claim to the victim's email address.
Finally, if OpenID Connect Discovery is supported and the Service Provider violates **SP-4-3** (subsubsection 2.3.3.4), a malicious Identity Provider can confuse the Service Provider by specifying the issuer within the WebFinger protocol as the value of a benign Identity Provider. If the Service Provider does not enforce that the issuer present in discovery "*MUST be identical to the issuer value returned by WebFinger. This also MUST be identical to the iss Claim value in ID Tokens issued from this Issuer*" [34, Section 4.3.], the malicious Identity Provider can then issue `id_token`s for arbitrary End-Users who used the benign Identity Provider to authenticate at the Service Provider before.

2.3.4.5 Cross-Site-Request-Forgery

OpenID Connect 1.0 and OAuth 2.0 are protocols that use multiple redirects with GET parameters being passed around, which leads to a higher risk for Cross-Site-Request-Forgery vulnerabilities.

If the Service Provider does not utilize the `state` parameter (and no additional mitigation like PKCE is implemented), "Session Donation" via Cross-Site-Request-Forgery is possible (**SP-6-1**, subsubsection 2.3.3.6). Without the `state` parameter, there is no binding between the `Authentication Request` and the `Authentication Response` containing the `code` that is further used to identify the End-User (by redeeming it for the `id_token` and `access_token`). As a result, a web attacker (section 3.1) could launch a Cross-Site-Request-Forgery attack against the SP by sending a request to the `redirect_uri` endpoint including a `code` that was issued for an attacker-controlled account [15, Section III; A. 2)]. In doing so, the web attacker "donates" his session to the victim and the victim is logged into the attacker's account.

In addition, if the Service Provider implements endpoints that start the OpenID Connect flow without CSRF mitigation and therefore violates **SP-8-1** (subsubsection 2.3.3.8) and **SP-8-2** (subsubsection 2.3.3.8), a web attacker (section 3.1) can utilize these endpoints to launch a Login CSRF against the End-User. If the End-User has an active session at the Identity Provider and the SSO account is linked to the Service Provider, the malicious actor only has to redirect the End-User to one of the mentioned endpoints to log in the victim into the victim's SSO account at the Service Provider.

2.3.4.6 Server-Side Request Forgery

If the Service Provider violates **SP-9-1** (subsubsection 2.3.3.9) and allows malicious Identity Providers (section 3.2) or malicious administrative users (section 3.4) to specify malicious endpoints, this enables SSRF by design.
If the Service Provider is self-hosted, administrative users can be considered to have high privileges within the software. Nevertheless, using SSRF mechanisms a malicious actor could potentially escalate privileges further if local or internal services, that are intentionally not exposed to the internet, are accessible by launching a SSRF attack. SSRF vulnerabilities occurring in hosted services have higher impact, because if an attacker can obtain information about the internal network or localhost in a hosted environment, this opens a new attack surface that is normally hidden behind a firewall.
Thus, at minimum SSRF to localhost and private IPs should be restricted by default. If there is a legitimate use-case for access to internal IPs, the software is most likely self-hosted and could implement an optional switch to enable an insecure configuration. Only super-admins that already have access to the underlying infrastructure should be able to enable requests to localhost or private IP addresses.

2.3.4.7 Sensitive Information Disclosure

The `state` and `code` parameters are carried within the query parameters of the `Authentication Response`. If the Service Provider violates **SP-7-1** (subsubsection 2.3.3.7) and external resources are embedded on the `redirect_uri` endpoint or error endpoints that also receive these parameters, the `Referer` header may disclose OpenID Connect parameters to third parties [48, Section 4.2.].
Sensitive OpenID Connect parameters could additionally be disclosed in the file system of the Service Provider, for instance in access logs or cache files [49, Section 4.6.7.], if the Service Provider violates **SP-7-2** (subsubsection 2.3.3.7).
Depending on the flow that is used, `access_token`s may be transferred within the front-channel as query parameters. This behavior is deprecated, no `access_token`s should be present in front-channel communication according to the OAuth 2.0 Security Best Current Practices [48, Section 2.1.] (**SP-7-3**, subsubsection 2.3.3.7).
Finally, if a Service Provider does not enforce TLS and communicates over unencrypted connections (**SP-7-4** , subsubsection 2.3.3.7), sensitive information and OpenID Connect parameters are disclosed to man-in-the-middle attackers (section 3.5).

2.3.4.8 Sub Claim Spoofing

If the Service Provider violates **SP-5-1** (subsubsection 2.3.3.5) and does not make sure the `UserInfo Response` belongs to the same End-User that the `id_token` was issued for, a malicious Identity Provider (section 3.2) could spoof arbitrary End-Users using the `UserInfo Response`'s "sub" claim.

2.3.4.9 Denial-of-Service

A trivial Denial-of-Service attack against a Service Provider can be launched if the implementation does not restrict the download size of contents received from OpenID Connect endpoints and does not set a sufficient connection timeout. If these measures are not taken, a malicious Identity Provider (section 3.2) or malicious administrative user (section 3.4) could exhaust storage and connections in connection pools (if used) by specifying download links to large files as OpenID Connect endpoints, known as "*Malicious Endpoint Attack*" [26, Section 2.1.4].

If the Service Provider violates **SP-6-3** (subsubsection 2.3.3.6) and allows to reuse the `state` parameter and does not implement additional rate limiting on the `redirect_uri` endpoint, it can be exploited as "amplifying proxy" for Denial-of-Service attacks.
It has been observed that the outgoing `Token Request` (back-channel) to the configured *Token Endpoint* can be multiple times larger than the `Authentication`

`Request` (front-channel). A malicious actor could utilize this to launch a Denial-of-Service attack spending limited bandwidth using the Service Provider as an amplifying proxy, as shown in Figure 2.8. A web attacker (section 3.1) could target the configured Identity Provider. A malicious Identity Provider (section 3.2) can additionally specify an arbitrary web server as *Token Endpoint* with a long URL path and issue long client credentials on registration (either manual or using OpenID Connect Dynamic Client Registration), resulting in an even larger outgoing request. If there is no rate limiting at the `redirect_uri` endpoint, the attacker could continuously replay the `Authentication Response` including the valid `state`, resulting in large outgoing `Token Requests` on each try.

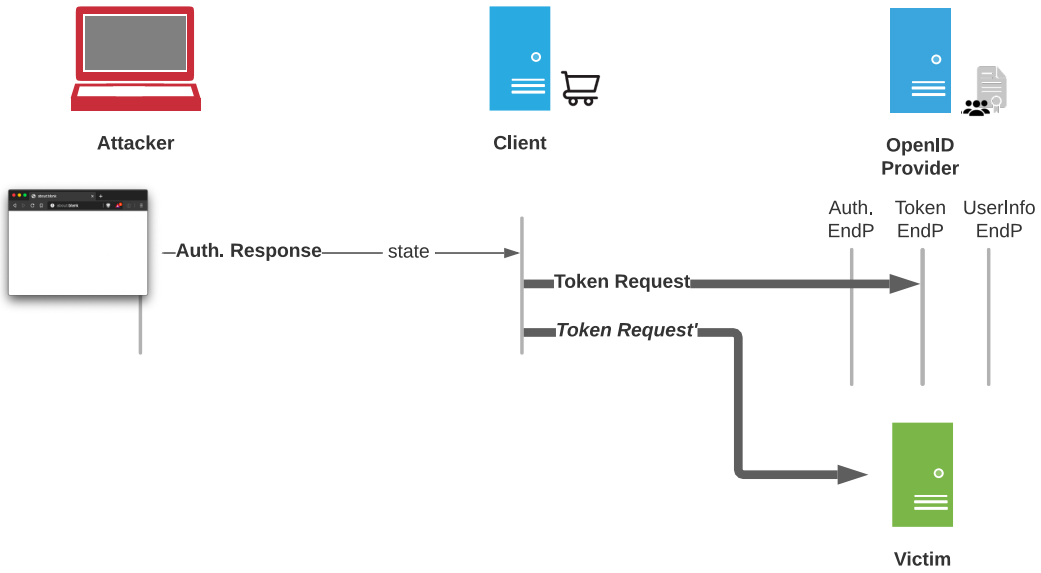

Figure 2.8: Reusable `state`: Denial-of-Service Amplification using `Token Request`.

2.3.4.10 Covert Redirect

If the redirection target is not sufficiently validated against a list of trusted hosts (and therefore violates **SP-10-1** (subsubsection 2.3.3.10)), a *Covert Redirect* occurs. That in doing so, parsing URLs is not an easy task, was for instance outlined by Wang et al. at Blackhat Asia '19 [51]. If not sufficiently validated, a web attacker (section 3.1) can redirect a victim End-User to an arbitrary redirection target after login using the non-normative redirection target parameter.
If session relevant endpoints are allowed as redirection target and **SP-10-2** (subsubsection 2.3.3.10) is violated, the web attacker can additionally influence the victim's session at the Service Provider after successful authentication.

2.3.4.11 Login Confusion

Login Confusion is a new attack that was developed during this thesis. It has the following prerequisites:

- The Service Provider needs to be vulnerable to Login Cross-Site-Request-Forgery (violate **SP-8-1** (subsubsection 2.3.3.8) or **SP-8-2** (subsubsection 2.3.3.8)), i.e. there is an endpoint that launches an OpenID Connect flow with a pre-configured Identity Provider.
- The Service Provider needs to implement a non-normative parameter that holds the redirection target after login. Additionally, session relevant relative paths like the *Initiate Login Endpoints* can be specified as destination (**SP-10-2**, subsubsection 2.3.3.10).
- The victim has two accounts at the Service Provider: One "native" account to which she authenticates using credentials and one "SSO" account which is linked to an account on an external Identity Provider. Furthermore, the victim has an active session at the Identity Provider.

If these prerequisites are met, the following attack could be launched:

1. The victim requests the Login UI: `https://sp.com/?next=/oauth/start`. As one can observe, the `next` parameter holds the relative path of the *Login Initiation Endpoint*.
2. The victim authenticates through the Login UI with credentials for **user A**.
3. After successful authentication, the victim is redirected to the `next` URL (*Login Initiation Endpoint*): `https://sp.com/oauth/start`.
4. Without further interaction, the *Login Initiation Endpoint* starts a new OpenID Connect flow with the pre-configured Identity Provider including a valid `state` that is issued by the Service Provider.
5. As the victim has an active session for **user B** at the Identity Provider, there is an immediate redirect to the `redirect_uri` including the `code` and the `state`.
6. The Service Provider completes the OpenID Connect flow with redeeming the `code` at the *Token Endpoint*. As it receives an `id_token` for **user B** from the Identity Provider, **user B** is logged in at the Service Provider.

As a result, our victim is now authenticated as **user B**, even though she entered her credentials for **user A** and did not perform any additional user interactions.

2.3.4.12 Mix-Up

If a Service Provider violates **SP-12-1** (subsubsection 2.3.3.12) a malicious Identity Provider (section 3.2) can launch a Mix-Up attack, in which it confuses a genuine Service Provider to redeem a valid `code` or `access_token` at the malicious Identity Provider's `Token Endpoint`, enabling the malicious Identity Provider to perform actions on behalf of the End-User and obtain personal information, as pointed out within the OAuth 2.0 Security Best Current Practices [48, Section 4.4.].

2.3.4.13 Injection

State parameter. If the Service Provider delivers application state within the `state` parameter and thus violates **SP-6-2** (subsubsection 2.3.3.6) by not applying integrity protection, this behavior can result in injection issues like SQL Injection (SQLi) and XSS.

IdP provided values: Generic. If Identity Provider provided values are not sufficiently sanitized, this could cause common injection flaws like XSS or SQLi [26, Section 2.1.3].

IdP provided values: CRLF. In addition, a novel variant of injection flaws in OpenID Connect has been observed. Identity Provider provided values that are used in sensitive contexts like HTTP headers need additional attention. If they are sanitized for usage in HTML context, dangerous *Carriage Return and Line Feed (CRLF)* sequences could be missed and end up as bearer token value within the `UserInfo Request`, resulting in potential HTTP header injections. An idealized flow for this injection vulnerability is shown in Figure 2.9.

Beside new HTTP headers, an attacker could smuggle new requests within the open TCP connection, by setting the `Connection: Keep-Alive` header and injecting multiple *CRLF* sequences to indicate that a new requests starts.
If OpenID Connect Discovery is implemented and the configuration of the Identity Provider is regularly updated, the malicious Identity Provider (section 3.2) could additionally alter its configuration and specify an arbitrary web server as `UserInfo Endpoint`, resulting in SSRF with control over Host, Path, HTTP Verb, HTTP Headers and HTTP body. Depending on the actual implementation, a malicious actor could even smuggle in other protocols than HTTP that can be used on top of TCP.
RFC6749 claims, that "*Access tokens can have different formats, structures, and methods of utilization*" and "*Access token attributes and the methods used to access protected resources are beyond the scope of this specification and are defined by companion specifications such as [RFC6750]*" [11, Section 1.4.]. In fact, RFC6750 defines in section 2.1. the format of `access_token` values that are allowed to be used

in the context of HTTP headers correctly, excluding *CRLF* sequences, as shown in Listing 2.6 [23, Section 2.1.].

```
The syntax for Bearer credentials is as follows:
     b64token    = 1*( ALPHA / DIGIT / "-" / "." / "_" / "~" /
     ↪  "+" / "/" ) *"="
     credentials = "Bearer" 1*SP b64token
```

Listing 2.6: RFC6750: Allowed characters for bearer token values.

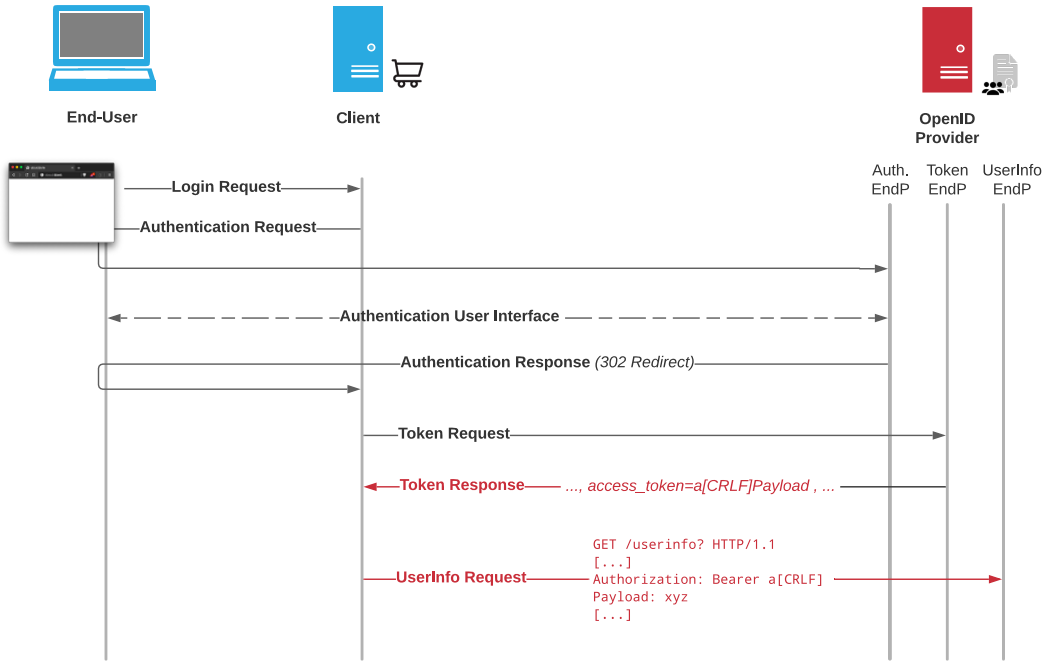

Figure 2.9: CRLF injection within HTTP header leads to arbitrary header injection.

3 Attacker Models

In this thesis, multiple different attacker models are considered. Each individual observation will be categorized to an attacker model that is described hereinafter.

3.1 Web Attacker

The web attacker model was introduced by Barth et al. in 2008 [4].

Victim. The victim in this attacker model is either an End-User, a benign Service Provider or a benign OpenID Provider.

Objectives of the Attacker. Depending on the victim, the attacker's objectives are different. If a web attacker targets an End-User, the ultimate goal is to break the authentication and access restricted resources. Beside this, a malicious actor's objective can be to influence the End-User's session, e.g. log in or log out a victim without indication to the user.
If a web attacker targets a Service Provider or Identity Provider, the primary objective is also an authentication bypass in order to access restricted resources. Additionally, causing notable harm to the application or the underlying infrastructure of the Identity Provider or Service Provider can be the objective of a malicious actor.

Capabilities of the Attacker. The web attacker is the least privileged attacker that is considered in this thesis. A web attacker is not able to intercept or eavesdrop communication that is not intended for him. If the victim is an End-User, we assume that the victim visits an attacker controlled website. Additionally, the web attacker may send and receive HTTP requests and provide a trusted Transport Layer Security (TLS) certificate for a controlled domain, e.g. `attacker.com`, to enable encrypted communication with Service Providers and Identity Providers. If explicitly outlined, the web attacker is in charge of a user account at the Identity Provider or the Service Provider.

3.2 Malicious Identity Provider

The malicious Identity Provider attacker model was introduced in 2016 by Mainka, Mladenov and Schwenk [27]. They highlight that the Identity Provider in traditional Single Sign-On setups like Kerberos is a trusted party. If customers can configure own Identity Providers or Identity Providers are dynamically discovered, these entities must be considered as third parties and must be treated accordingly.

Victim. A malicious Identity Provider could either target an End-User or a benign Service Provider as victim.

Objectives of the Attacker. If the malicious Identity Provider targets an End-User, the attacker's objective is to bypass the authentication of the End-User that used an honest Identity Provider to register at a Service Provider, in order to authenticate as the victim user. If a malicious Identity Provider directly targets a benign Service Provider, beside the authentication bypass, causing harm to the underlying application or infrastructure can be the objective of a malicious actor.

Capabilities of the Attacker. A malicious Identity Provider is able to perform a genuine OpenID Connect flow and to authenticate users for a Relying Party. In doing so, it is not bound to restrictions being made by the specification and could also act maliciously. For instance, the malicious Identity Provider may send malformed requests, set `id_token` claims in a misleading manner or may tamper with the key references that are used for signature validation. Prerequisite for this is that either a victim Client supports Dynamic OpenID Provider Discovery e.g. using the *WebFinger* protocol [36] or a manually configured and initially trusted Identity Provider turns rogue at some point in time. In general, as per the overall setup of Single Sign-On, the Identity Provider is a third party from a Client's perspective.

3.3 Malicious Service Provider

Victim. A malicious Service Provider could either target an End-User or a benign Identity Provider as victim.

Objectives of the Attacker. If a malicious Service Provider targets an End-User, his main objective is to gather sensitive information about the End-User and to gain privileges to perform actions on behalf of the End-User. In contrast, if a benign Identity Provider is targeted, beside authentication bypasses the underlying application and infrastructure of the Identity Provider can be target of the malicious Service Provider.

Capabilities of the Attacker. A malicious Service Provider is able to start a genuine OpenID Connect flow and retrieve a `code`, an `id_token` and an `access_token` from an Identity Provider. Just like the malicious Identity Provider, it may act maliciously when communicating with a victim Identity Provider or victim End-User. In contrast to the regular web attacker, it is in charge of valid client credentials. Thus, a malicious Service Provider needs to initially register at the benign Identity Provider in order to obtain these client credentials. This could either be done manually or using the OpenID Connect Dynamic Client Registration [35].

3.4 Malicious Administrative User

The malicious administrative user attacker model was introduced by Matsumoto et al. in 2014 [29, Section 3.1]. The impact of attacks under this attacker model is highly conditional, as described in the following:

Victim. A malicious administrative user can either target a benign Service Provider or a benign Identity Provider as victim. Additionally, a malicious administrative user could target End-Users of the Provider he has administrative privileges on.

Objectives of the Attacker. Main objective of a malicious administrative user is to target the application or the underlying infrastructure of a Service Provider or Identity Provider. In some cases, a malicious administrative user could additionally target End-Users of the Service Provider or Identity Provider and try to obtain sensitive OpenID Connect related parameters that should not be accessible to administrative users.

Capabilities of the Attacker. A malicious administrative user is a powerful attacker. As he is privileged, he may change OpenID Connect related configurations either at the Identity Provider *or* at the Service Provider. Thus, exploitability and severity of security issues that occur under this attacker model are highly conditional. Considering a hosted service, a customer that is able to exploit vulnerabilities to target the service directly could in fact cause notable harm.
In contrast, if the software is self-hosted, the administrative user is highly trusted in most cases and might have even further access to the system than through a web service. Nevertheless, if the environment enforces the concept of least privileges, there may still be the possibility for malicious administrative users on one host to gain access to restricted resources by exploiting vulnerabilities in the self-hosted software and escalate privileges.

3.5 Man-in-the-Middle

The most powerful attacker that is considered in this thesis is the man-in-the-middle attacker. Man-in-the-middle attacks are "*well-known in security environments, and have drawn significant attention*" [42].

Victim. In an OpenID Connect Flow, every entity can be target of a man-in-the-middle attacker, as the attacker operates on the requests and responses that are sent over the network.

Objectives of the Attacker. A malicious entity's main goal is to bypass authentication and authenticate as an End-User at a Service Provider. Additionally, a man-in-the-middle attacker can aim to learn sensitive information and personal data that is transferred over the network.

Capabilities of the Attacker. In the following, we consider that if TLS is used and enforced, all entities use TLS in a secure manner. Thus, the man-in-the-middle attacker is not able to tamper with HTTPS communication, but he is able to obtain, modify or drop unencrypted communication like plain HTTP without TLS.

4 Selection and Test Environment

This chapter deals with the methodology of this thesis. After outlining the selection of OpenID Connect implementations the setup of the test environments are described.

4.1 Selection of OpenID Connect Implementations

In this master's thesis, we focus on products and services that implement OpenID Connect in contrast to libraries that implement OpenID Connect primitives and capabilities. As libraries are often designed to be flexible and universal, depending on their actual configuration, the security level of the resulting OpenID Connect implementation is highly conditional. Therefore, we will analyze products and services that actually implement and utilize OpenID Connect to evaluate *real-life* OpenID Connect implementations.

As of April 2020 many products and services use OpenID Connect 1.0 and OAuth 2.0. The analysis of each product and service would have its own benefits and limitations, for instance regarding setup of the analysis environment (self-hosted versus hosted service), availability of the source code, relevance in terms of usage and the possibility to configure a malicious Identity Provider in order to perform Service Provider related tests. Therefore, the following sections discuss the selection criteria for OpenID Connect implementations applied in this master's thesis.

4.1.1 Identity Provider Selection

Keycloak is a broadly used open source Identity and Access Management (IAM) software. Written in Java, it comes with support for OpenID Connect and SAML as Identity Provider. When acting as *Identity Broker*, Keycloak additionally can act as Relying Party against other OpenID Connect Identity Providers. We analyze v10.0.0 of Keycloak.
As it is open-source, beside focusing on network analysis, a greybox testing approach can be used to prove observations and to support the analysis process. We consider this approach to be "greybox" instead of "whitebox", because we do not perform thorough code analysis. The public Keycloak Docker Image available at `https://hub.docker.com/r/jboss/keycloak/` was downloaded more than 50 million times

(September 2020) and allows a standardized and developer-friendly setup of the analysis environment.

In addition to Keycloak, some of the products and services that were chosen as test subjects for the Service Provider evaluation implement Identity Provider capabilities. All of them do not implement the complete set of features required to apply the entire evaluation catalog, so that only a subset of the Identity Provider evaluation catalog could be applied to these implementations.

4.1.2 Service Provider Selection

The requirements for Service Providers that could be covered in this master's thesis were quite specific. At first, all implementations that could be considered have to support the setup of a custom Identity Provider. This is the main reason, why other popular self-hosted products and popular services could not be considered, as some services only support a fixed set of commonly used Identity Providers.
Additionally, the environment of a Service Provider can have huge effect on the impact of potential vulnerabilities. Therefore, self-hosted software *and* remote services are chosen as analysis subjects.
Beside the focus on the ability to specify a custom Identity Provider, most implementations that are mentioned below also implement the Identity Provider part of the OpenID Connect specification themselves. Thus, the Identity Provider specific parts of the evaluation catalog are also applied to these implementations, resulting in a broader test-set for Identity Provider implementations.

Bitbucket Server is a commercial git and code management software developed by *Atlassian*. The server version of Bitbucket can be self-hosted, a Docker Image that was pulled more than 10 million times is available at `https://hub.docker.com/r/atlassian/bitbucket-server/`. Additionally, Bitbucket's OpenID Connect implementation was just recently announced and published, resulting in a vivid development and the potential to nudge the ongoing processes to take a secure and specification compliant direction [3]. We analyze v7.2 of Bitbucket Server.

In contrast, GitLab's OAuth 2.0 and OpenID Connect implementation was introduced with v7.7 in January 2015 [18]. Just like Keycloak, GitLab supports the Identity Provider and the Service Provider part of the OpenID Connect specification. There is a public Docker Image for the Community Edition that was solely pulled more than 100 million times: `https://hub.docker.com/r/gitlab/gitlab-ce/` (September 2020). We analyze v13.2 of GitLab's Community Edition.

Salesforce is a Customer-Relationship-Management service that has more than 150.000 users, according to official sources [41]. The service implements the OpenID Connect Service and Identity Provider part of the specification.

Amazon Web Services (AWS) is an umbrella term for many cloud services provided by Amazon. According to official sources, AWS has "*millions of customers*" [1]. Among these services, there is Amazon Cognito, a cloud-based Identity and Access Management (IAM) service. Serving as Identity Broker, Identity Provider and Service Provider parts of the OpenID Connect specification are implemented and can be analyzed.

4.2 Setup

The setup for our test environment is mainly based on Docker. If possible, local instances of the test subjects are setup on our analysis machine. Only Salesforce and Amazon Cognito as web services require remotely accessible Identity and Service Providers hosted within the NDS-Cloud. To monitor and modify front- and back-channel communication, we configure the user's User Agent and all self-hosted Identity and Service Providers to use an intercepting proxy running on our analysis machine. Furthermore, custom minimal OpenID Connect Service and Identity Provider Implementations were created.

4.2.1 Custom Implementations

While most of the local tests were performed using Burp Suite, "*the world's most widely used web application security testing software*" [38], as intercepting Proxy to modify requests on the fly, during the execution of the analysis more flexible OpenID Connect implementations turned out to be very helpful tools. This applies especially to cross-phase attacks and cryptography attacks that require the generation of signatures based on *nonces* which are provided by the Relying Party. Therefore custom NodeJS-based Service and Identity Provider implementations were created. Among other features, they are very flexible regarding JWT signature generation, can be configured to use TLS and allow to control the entire protocol flow from either the Service or the Identity Provider's perspective. The source code can be found on Github[1][2].
Thus, most of the remote tests and the more advanced local tests were performed using these custom implementations.

4.2.2 Local Test Environment

The local test environment uses Burp Suite as an intercepting proxy, Docker containers for our test subjects and NodeJS webservers for our custom OpenID Connect implementations, all running on the same machine.

[1] Custom OpenID Connect Service Provider: `https://github.com/lauritzh/oidc-custom-sp`
[2] Custom OpenID Connect Identity Provider: `https://github.com/lauritzh/oidc-custom-idp`

During the analysis, the 8GB of RAM available at the analysis machine turned out to be barely sufficient. The migration to a more powerful machine showed that 16GB to 20GB RAM should be available to sufficiently run multiple containers running heavy Java and Ruby applications, Burp Suite as a Java application and multiple Chrome/Chromium instances at the same time.

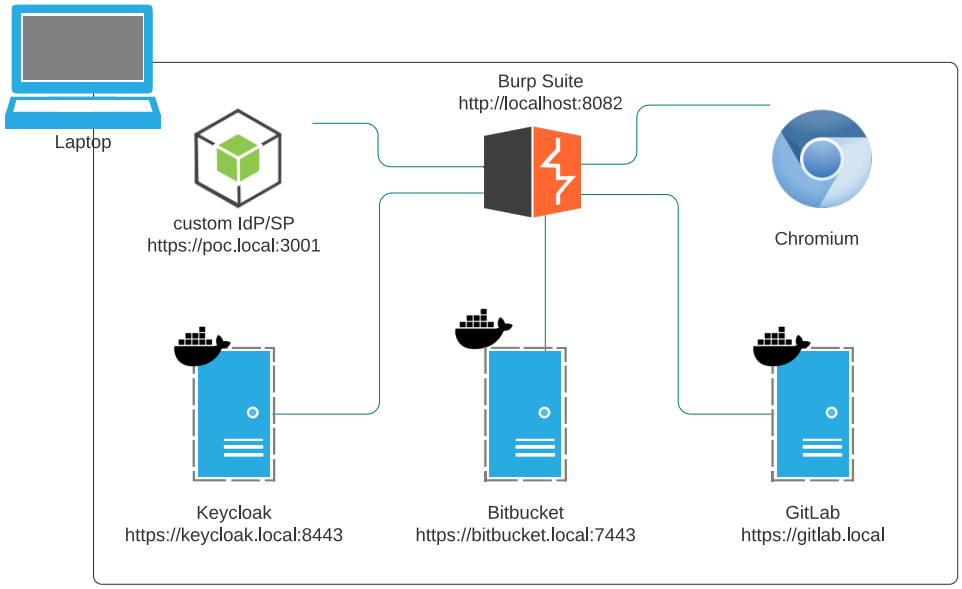

Figure 4.1: High Level overview of the local docker-based test environment.

4.2.2.1 Keycloak

In our setup, we use the official *JBoss* Keycloak Docker image available at: `https://hub.docker.com/r/jboss/keycloak/`. As we would like to use Burp Suite as intercepting proxy and Keycloak as a Java application performs certificate validation, we need to add Burp Suite's certificate authority to Keycloak's trust store. Afterwards Keycloak can be configured to use the proxy for outgoing HTTP requests. We use the built-in H2 database bundled with Keycloak.

4.2.2.2 Bitbucket

Just like for Keycloak, *Atlassian* publishes an official Docker image for Bitbucket available at: `https://hub.docker.com/r/atlassian/bitbucket-server/`. Bitbucket is a commercial software, so that we need to obtain an evaluation license that expires after 30 days. Additionally, we need to use a "Data Center" license, as the relevant "SSO 2.0" functionality is not available for "Server" instances. Bitbucket is also a Java application, so that the Burp Suite certificate authority once again needs to

be added to the application's trust store. To observe the outgoing requests from Bitbucket, the Burp Suite is configured as outgoing proxy.

4.2.2.3 GitLab

For GitLab's Community Edition there are official Docker-Compose files and Docker images available at: `https://hub.docker.com/r/gitlab/gitlab-ce/`. This version could be installed without further licenses, in contrast to the Enterprise Edition. GitLab is a Ruby-based software, so that there is no global trust store just like Java applications have. In contrast, GitLab requires certificates that should be added to the trust store to have a specific name and be mounted to a specific path on the file system, so that they are imported during startup. The configuration of the Burp Suite as an outgoing proxy is comparable to Bitbucket.

4.2.3 Remote Test Environment

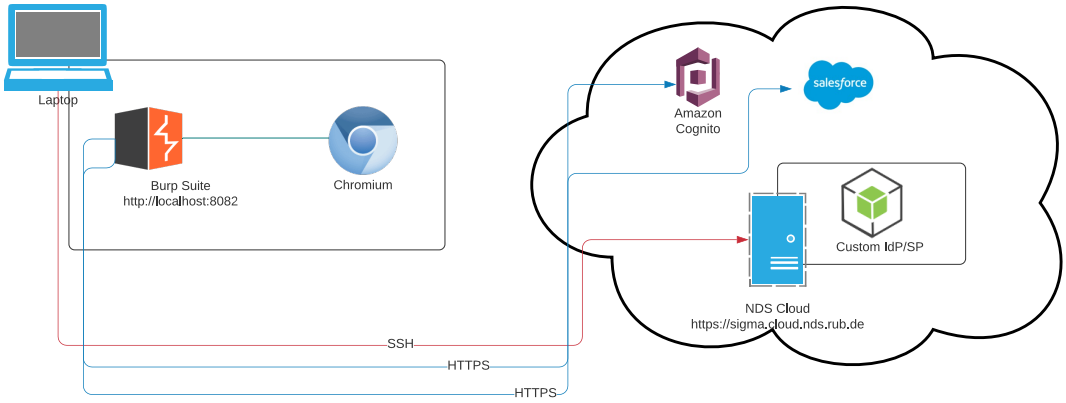

Figure 4.2: High Level overview of the remote test environment.

Core of the remote test environment is a virtual machine hosted at the chair for network and data security. This machine has 8GB of RAM which is sufficient, as we would like to run at most one Docker container with Keycloak. Most of the time we will use our minimal and custom OpenID Connect implementations.

4.2.3.1 Salesforce

Salesforce allows users of an organization to authenticate using an external provider. All we have to do is launching the custom Identity Provider at the publicly accessible NDS machine and configure Salesforce to use our custom Identity Provider for external authentication. As *https* is required and we need a valid certificate, we set an additional DNS entry for a controlled website (e.g. `https://openid.lauritz-`

holtmann.de/) to the NDS IP and claim a Let's Encrypt certificate for this subdomain. Alternatively, the service *ngrok* could be used to make a locally running instance of the NodeJS application publicly accessible having a valid certificate.

4.2.3.2 Amazon Cognito

Amazon SSO only supports SAML for Logins to an organisation using Single Sing-On. But there is a service that is part of the AWS family that can act as Service and Identity Provider, namely "Amazon Cognito" . Thus, we configure our publicly available Service and Identity Providers hosted within the NDS cloud at Amazon's Cognito service. As Amazon Cognito should enforce TLS for communication with OpenID Connect endpoints and restricts the depth of subdomains (only one subdomain is allowed), we use `https://openid.lauritz-holtmann.de/` with our previously obtained Let's Encrypt certificate. Alternatively, *ngrok* could be used.

5 Security Evaluation

In the following, the evaluation results of this thesis are presented. At first, the evaluation table for Identity Providers is explained. Afterwards, the most significant Identity Provider observations are outlined.
Thereafter, the results of the Service Provider evaluation are outlined. Finally, the most significant observations regarding the Service Provider implementations are described in detail.

5.1 Identity Provider Evaluation

Category	**Keycloak**	**GitLab**	**Salesforce**	**Amazon Cognito**
Redirect URI Protection	●	✔	✔	●
HTTP	✔	✔	✔	✔
Request Object, request_uri and Registration Object	●	N/A	N/A	N/A
Access/Refresh Token and Client Credentials Protection	●	✔	●	●
Code Protection	○	●	○	✔
Denial-of-Service	✔	✔	N/A	N/A
PKCE	✔	●	✔	✔
Audience Confusion	●	✔	✔	✔
Consent Screen	✔	✔	✔	✔
Dynamic Client Registration	●	N/A	○	N/A
Client Authentication	○	N/A	N/A	N/A

✔: Secure / No attack found in this category
●: Insecure / One or more attacks found in this category
○: Theoretical issues identified or limited attack success

Table 5.1: Evaluation of Identity Provider related Test and Attack Categories.

Table 5.1 presents the evaluation results of the analyzed Identity Provider implementations. Only Keycloak implements the complete subset of the specification

including optional aspects as well as Dynamic Client Registration and Discovery that allows performing tests for all outlined categories.

Redirect URI Protection Two out of four analyzed Identity Provider implementations show issues regarding the `redirect_uri`. The comparison against pre-registered values is implemented correctly among the test set, but Keycloak and Amazon Cognito support `redirect_uri` values with dangerous schemes like `javascript` or `data`.

HTTP All analyzed implementations use HTTP 302 redirects as expected.

Request Object, request_uri and Registration Object Only Keycloak supports the `request_uri` and does not implement counter measures against the unauthenticated SSRF previously described by Fett et al. in 2017 [15, Section III; A. 8)].

Access/Refresh Token and Client Credentials Protection Three out of four implementations have issues regarding their Token or Client Credentials protection. Keycloak allows Clients to specify weak `client_secret` values and does not prevent brute force attacks on the *Token Endpoint*. Additionally, if the built-in H2 database is used, default credentials are used. Salesforce does not issue `refresh_token`s at all and uses `access_token`s that are longer valid than 60 minutes. Amazon Cognito does not rotate `refresh_token`s and does not allow End-Users to revoke granted permissions on the Identity Provider.

Code Protection Most of the `code` protection assertions were fulfilled among the test set. Keycloak (default 60 minutes) and Salesforce chose a slightly longer expiry time for `code` values than the expected 10 minutes. Striking is the `code` revocation flaw observed in GitLab that allows a malicious Client to restore complete permissions after they were revoked, using a previously issued `code`.

Denial-of-Service None of the self-hosted implementations show longer response times if a huge amount of fresh `code` or `access_token` values is requested. Assertions that could potentially cause harm to the systems (e.g. Denial-of-Service) were not tested for hosted services.

PKCE Among the test-set, only GitLab does not support PKCE at all.

Audience Confusion. All tested implementations issue the `client_id` and `client_secret` values to clients. But for Keycloak, there is a non-normative registration API (beside the OpenID Connect Dynamic Client Registration) that additionally allows authenticated clients to manually specify `client_id` and `client_secret`. In doing so, even registration of clients that only differ in capitalization is possible, additionally increasing the risk of Audience Confusion.

Consent Screen The information presented on the Consent Screen was not found to be incorrect among the test-set.

Dynamic Client Registration Only Keycloak and Salesforce support Dynamic Client Registration. Salesforce issues Initial Access Tokens that can be used to register an arbitrary amount of clients without possibility to limit the validity. Keycloak in contrast allows to issue tokens that may be used to register n previously defined clients. Keycloak supports the `sector_identifier_uri` and in doing so supports HTTP without TLS.

Client Authentication Finally, only Keycloak supports JWT based Client Authentication mechanisms. The signature validation itself was not found to be erroneous, nevertheless, the implementation uses the fact that "iss" and "sub" claim are expected to equally hold the `client_id` and therefore only utilizes the "sub" claim. As a result, the "iss" claim could be omitted from the JWT, but the JWT would have to be signed with the correct key nevertheless.

5.2 Identity Provider Analysis Details

In the following, the most significant observations regarding Identity Provider implementations made during the practical analysis phase of this thesis are described in detail. The observations are ordered based on their severity, which is represented as scale from *None* to *Critical* (None, Low, Medium, High, Critical). The title of each finding indicates if it is considered as `[BUG]` with significant impact to the OpenID Connect implementation, as an issue regarding the `[SPEC]` with no immediate resulting attack in our considered attacker models or `[VULNERABILITY-TYPE]` (see section 2.3) that is mapped to an attacker model. For each finding a recommendation to mitigate the issue is given.

5.2.1 Analysis of Keycloak

As Identity and Access Management (IAM), the OpenID Connect Identity Provider implementation is part of Keycloak's core functionality. Keycloak implements broad parts of the specification including optional aspects.

Section	Vulnerability Type	Attacker Model	Severity
5.2.1.1	SSRF	Web Attacker	High
5.2.1.2	Insufficient Filtering	Malicious Administrative User	Low
5.2.1.3	Sensitive Information Disclosure	Malicious Administrative User	Low
5.2.1.4	Brute Force	Web Attacker	Low
5.2.1.5	Insufficient Filtering	Malicious Service Provider or Malicious Administrative User	Low
5.2.1.6	Parser Error	N/A	None
5.2.1.7	Unencrypted Communication	N/A	None
5.2.1.8	Incomplete Validation	N/A	None

Table 5.2: Overview of the most relevant observations and findings in Keycloak's Identity Provider implementation.

5.2.1.1 [SSRF] "request_uri" in Authentication Request (CVE-2020-10770)

Attacker Model. The attacker model that is applied for the following attack is an unauthenticated web attacker (section 3.1).

Description. The `request_uri` is an optional parameter within the OpenID Connect `Authentication Request` that allows specifying an external URI where the `Request Object` may be found. Fett et al. discovered in 2017 [15, Section III; A. 8)] that, as the Identity Provider is supposed to request the external `Request`

`Object`, this parameter can be easily used to launch a SSRF attack against the Identity Provider.

The OpenID Connect Core specification defines the `request_uri` as an `Authentication Request` parameter that "*enables OpenID Connect requests to be passed by reference, rather than by value*" [33, Section 6.]. If a Service Provider uses this parameter, the Identity Provider retrieves the `Request Object` "*from the resource at the specified URL*" [33, Section 6.2.]. The Identity Provider could restrict allowed URLs by allowing the Service Provider to specify the `request_uris` parameter that is an "*array of request_uri values*", at Registration [35, Section 2.]. If OpenID Connect Dynamic Client Registration is used, the Identity Provider can require the Service Provider to "*pre-register request_uri values using the request_uris parameter*" [33, Section 6.2.]. Thus, only if OpenID Connect is implemented with the Registration Extension, there is a mitigation hint given by the specification.

Keycloak supports using a `Request Object` and referencing it externally. The "Fine Grained OpenID Connect Configuration" for a specific client allows specifying the following values for the option "Request Object Required" :

- Not required (default)
- request or request_uri
- request only
- request_uri only

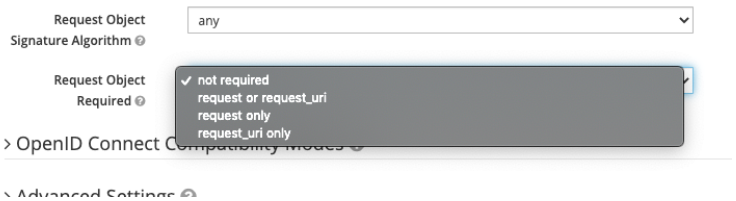

Figure 5.3: Keycloak: Configuration options for the *Request Object*.

As one can see, if an OpenID Connect Client is set-up with default settings, the default value for this option is "Not required". There is no option to do not support a `Request Object` at all, as "not required" means submitting the request object is just optional but if you submit one, it is used by Keycloak. Additionally, there is no option to manually define the `request_uris` parameter during manual client registration. Finally, using the default configuration, Keycloak does not require a `Request Object`, but supports passing the `request_uri` parameter with an `Authentication Request`. As a result, Keycloak is in its default configuration for clients vulnerable to the SSRF attack Fett et al. discovered in 2017 [15, Section III; A. 8)].

Recommendation. To mitigate the described issue, the `request_uri` should be disabled in Keycloak's default Client configuration. Furthermore, localhost and private IPs should be forbidden as `request_uri`, unless explicitly enabled by an administrative user with high privileges. Finally, the `request_uris` whitelist should be used to further restrict the `request_uri` parameter.

5.2.1.2 [Insufficient Filtering] Base URL Allows "data" URIs (CVE-2020-10748)

Attacker Model. The administrative user attacker model (section 3.4) is considered in the following, as privileges are required to modify Keycloak's configuration.

Background. In November 2019 there was a penetration test on Keycloak performed by Cure53 [8]. In this pentest, Heiderich et al. found a XSS vulnerability that was acknowledged as `CVE-2020-1697`.

Keycloak allows specifying a "baseUrl" for each client. This URL was reflected on different pages, in two cases without applying sanitization before setting it as `href` attribute of an anchor. As a result, there was a stored XSS vulnerability present if a Client registered a "baseUrl" with `javascript` scheme. If a user clicked a link on a website that used the `javascript` scheme, the JavaScript was executed in the same context as the website the anchor was embedded.

Description. To mitigate this behavior, Keycloak introduced a filter for the `javascript` scheme. This filter can be bypassed using another potential dangerous scheme, namely the `data` scheme. The `data` scheme is less dangerous, as most popular Browsers changed their behavior in 2017 and do not navigate to these URIs anymore [7]. Nevertheless, Safari 13.5 for macOS and iOS executes JavaScript in a different context than the embedding site (the *null origin*) if a user clicks an anchor with a `data` URI.

An example "baseUrl" could be data:text/html;base64,PHNjcmlwdD5jb25maXJtKG-RvY3VtZW50LmRvbWFpbik7PC9zY3JpcHQ+ resulting in the markup snippets given in Listing 5.1 and Listing 5.2.

Recommendation. To mitigate this issue, any user supplied contents and the "baseURL" scheme in special should be sufficiently sanitized before further usage.

```
<p><a id="backToApplication"
↪ href="data:text/html;base64,PHNjcmlwdD5jb25maXJtKGRv
↪ Y3VtZW50LmRvbWFpbik7PC9zY3JpcHQ+">Back to Application</a></p>
```

Listing 5.1: Keycloak: Missing sanitization on `https://keycloak.local/auth/realms/master/protocol/openid-connect/auth?scope=openid&response_type=code&redirect_uri=wrong&state=something&nonce=something&client_id=test.local` (can be accessed without authentication).

```
<td>
    <a href="data:text/html;base64,PHNjcmlwdD5jb25maXJtKGRv
    ↪ Y3VtZW50LmRvbWFpbik7PC9zY3JpcHQ+">
    test.local
    </a>
</td>
```

Listing 5.2: Keycloak: Missing sanitization on `https://keycloak.local/auth/realms/master/account/applications` (authentication required).

5.2.1.3 [Sensitive Information Disclosure] Missing Referrer Policy and Image Injection on Login Screen

Attacker Model. The following attack utilizes the malicious administrative user attacker model (section 3.4), as the configuration that needs to be set is only accessible for administrative users that have access to the realm configuration.

Description. Administrative users can customize Keycloak's Login Screen for each realm using the "HTML display name". Keycloak uses an HTML sanitizer that aims to only allow harmless markup, for instance `<script>`-tags are stripped out of the rendered realm name. The sanitizer assumes images without event handlers as harmless.

Furthermore, Keycloak does not use an HTTP Referrer-Policy or a Content Security Policy (CSP) that prevents the User Agent from disclosing the path of the embedding page in HTTP `Referer` headers. As a result, an embedded image pointing to a third party resource discloses OpenID Connect parameters to these parties, including the `state` and `nonce`.

Considering a basic image tag like
`<img src="https://lauritz-holtmann.de/image.png">`, the resulting HTTP request including a sensitive `Referer` header is presented in Listing 5.3.

```
1 GET /image.png HTTP/1.1
2 Host: lauritz-holtmann.de
3 Referer: https://keycloak.local:8443/auth/realms/master/protocol/
  ↪ openid-connect/ auth?client_id=test.client
  ↪ &redirect_uri=https%3A%2F%2Fkeycloak.local%3A8443%2Fauth%2F
  ↪ &state=10e42242-d14a-4674-87b0-a09a71770f6b&response_type=code
  ↪ &scope=openid&nonce=212af5ad-d957-4354-84db-d7c16afd26d3
4 [...]
```

Listing 5.3: Keycloak: Third Party request that discloses OpenID Connect `state` and `nonce`.

Recommendation. To mitigate this issue, a `Referrer-Policy` that prevents the leakage of sensitive date through `Referer` headers should be introduced.

5.2.1.4 [Brute Force] Weak "client_secret" and Missing Rate Limiting on Token Endpoint Enable Client Credential Brute Force

Attacker Model. In the following the web attacker model (section 3.1) is considered. If a client is manually configured using the web interface, the `client_secret` that is chosen by Keycloak is sufficiently strong to mitigate the risk of brute force attacks. Nevertheless, using Keycloak's registration API, weak `client_secret` values could be chosen. Thus, a more secure configuration than outlined in the following is possible but not consistently enforced.

Description. When querying Keycloak's *Token Endpoint*, Clients can provide their credentials using Basic Authorization. When doing so, Keycloak responds with different error codes depending on the validity of the provided client credentials. Additionally, even if a Client repeatedly provides wrong credentials over and over again, there is no rate limiting in place.

A sufficiently strong `client_secret` mitigates the risk of brute force, so that Clients that are registered using the web UI and receive a secret from Keycloak are not at risk. But Keycloak additionally allows Clients to register using a non-normative registration API that allows the client to choose its `client_id` and `client_secret` on its own, without applying a policy regarding the strength of chosen secrets. As a result, Clients can register themselves using a single character secret, as shown in

Listing 5.4 and Listing 5.5. That the weak secret is accepted could be additionally observed using the web UI presented in Figure 5.4.

Recommendation. Keycloak should introduce a rate limiting on the *Token Endpoint*. If a client repeats to provide erroneous credentials there should be a configurable rate limiting in place – just like the existing brute force detection for the user login. Furthermore, a policy should enforce sufficiently strong `client_secret` values if a Client registers itself using the API.

```
1 POST /auth/realms/master/clients-registrations/default HTTP/1.1
2 Host: keycloak.local
3 Connection: close
4 Authorization: Bearer [REDACTED]
5 Content-Type: application/json
6 Content-Length: 40
7
8 {"clientId" : "test1234", "secret": "a"}
```

Listing 5.4: Keycloak: The Client Registration allows using weak secrets, an example request is shown above.

```
1 HTTP/1.1 201 Created
2 Location: https://keycloak.local/auth/realms/master/
  ↪ clients-registrations/default/test1234
3 Content-Type: application/json
4 Content-Length: 1259
5  [...]
6
7 {"id":"64872404-a71d-4e44-a0f3-6889ddae1ea8","clientId":"test1234",
  ↪ "surrogateAuthRequired":false,"enabled":true,
  ↪ "alwaysDisplayInConsole":false,
  ↪ "clientAuthenticatorType":"client-secret","secret":"a",
  ↪ [...]}
```

Listing 5.5: Keycloak: The Client Registration allows using weak secrets, an example response is shown above (only most significant headers are presented).

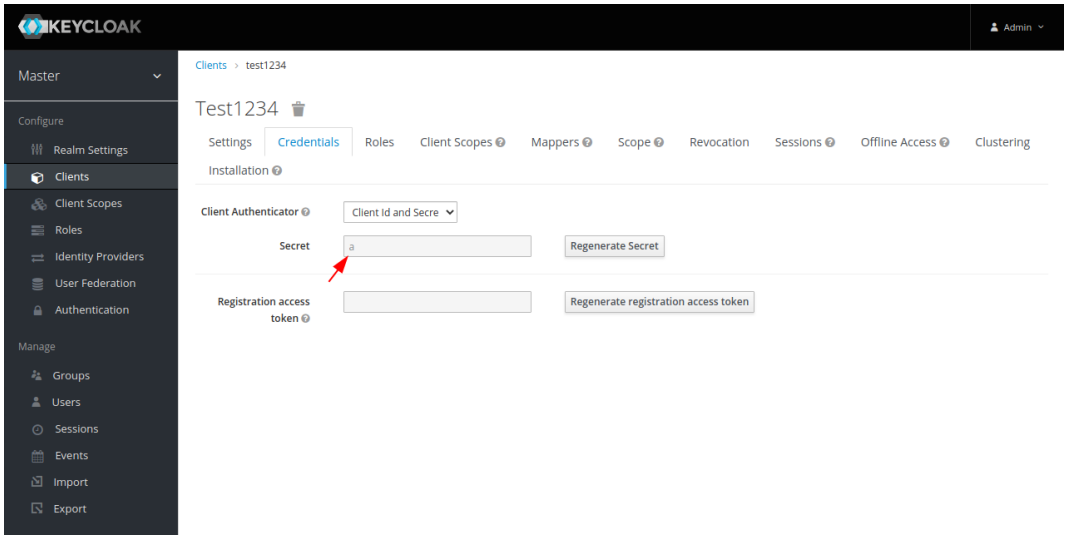

Figure 5.4: Keycloak: The Client Registration allows using weak secrets. The weak `client_secret` is also displayed on the configuration page.

5.2.1.5 [Insufficient Filtering] OIDC "redirect_uri" Allows Dangerous Schemes Resulting in Potential XSS (CVE-2020-10776)

Attacker Model. A malicious administrative user (section 3.4) or a malicious Service Provider (section 3.3) that registers using OpenID Connect Dynamic Client Registration are considered as attacker models in the following.

Description. According to the OpenID Connect specification, alternative schemes may be used as `redirect_uri` beside *https*. If the Client Type is configured as confidential, *http* is allowed as scheme. For native applications – e.g. the Twitter Application – even custom schemes like *twitter://* may be used [33, Section 3.1.2.1.].

Following the specification consequently, Keycloak does not restrict the `redirect_uri` regarding its scheme at all. This enables dangerous behavior, as the `Authentication Response`'s 302 redirect is sent to this URI with an arbitrary scheme. As a result, depending on the Browser's behavior and treatment of the `Location` HTTP header, this may cause several security issues, potentially even leading to Cross-Site-Scripting (XSS).

Browser behavior regarding dangerous schemes in `Location` headers highly differs and changed in the last decade. Among a test-set of current and popular Browser versions, only Safari 13.5 (iOS and macOS) executes JavaScript provided as `data`-URI, as shown in Figure 5.5.

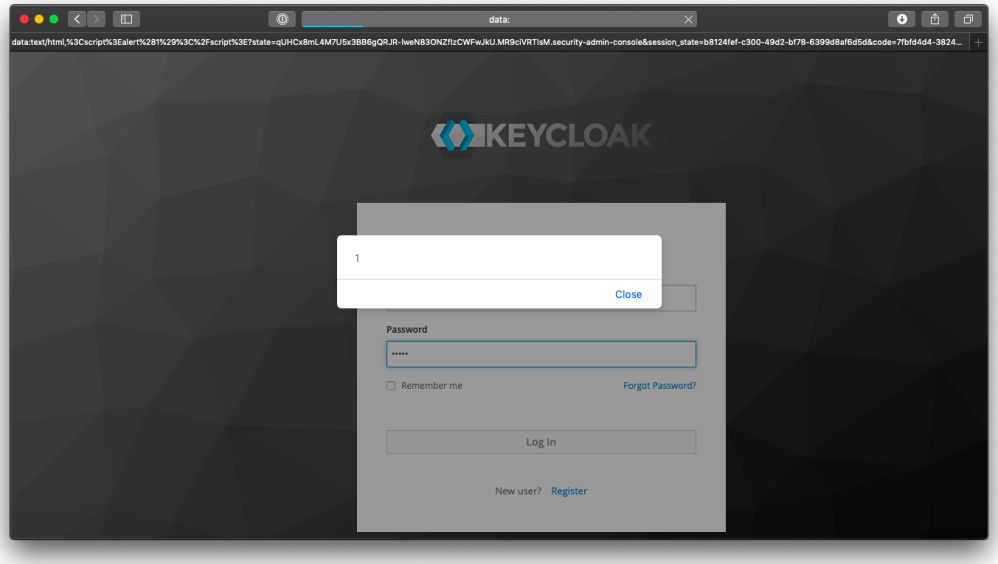

Figure 5.5: Keycloak: JavaScript execution using `data` URI as `Location` header.

The JavaScript execution is limited, as the script is executed in a different origin than the Identity Provider that sets the `Location` header. Thus, cross-origin access is restricted by the Same-Origin Policy. Nevertheless, an attacker may be able to launch a Cross-Origin State Inference (COSI) attack like Sudhodanan et al. pointed out in their paper "*Cross-Origin State Inference (COSI) Attacks: Leaking Web Site States through XS-Leaks*" [46] using a `data` URL as `redirect_uri`.

Steps to reproduce:

1. Register a new client and configure the following `recirect_uri`: `data:text/html,%3Cscript%3Ealert%281%29%3C%2Fscript%3E`

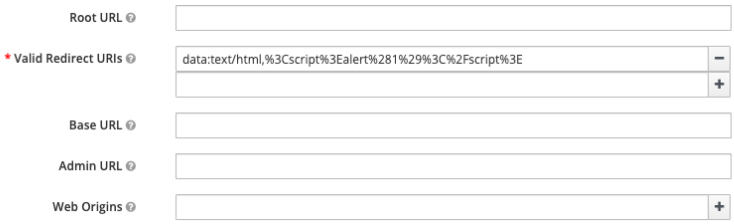

Figure 5.6: Keycloak: Client configuration with `data`-URI as `redirect_uri`.

2. Visit the *Authentication Endpoint* for the created Client: `https://keycloak.local:8443/auth/realms/master/protocol/openid-connect/auth?scope=openid&state=`

```
a&response_type=code&client_id=test&redirect_uri=data%3Atext%2Fhtml%
2C%253Cscript%253Ealert%25281%2529%253C%252Fscript%253E&nonce=b
```

3. After successful authentication there is a redirect to the registered `redirect_uri`:

```
1 HTTP/1.1 302 Found
2 Location:
  ↪ data:text/html,%3Cscript%3Ealert%281%29%3C%2Fscript%3E?state=a
  ↪ &session_state=b&code=c
3 [...]
```

Listing 5.6: Keycloak: 302 Redirect to `data`-URI (parameters shortened).

Depending on the Browser's behavior, the `data`-URI is parsed and the embedded JavaScript is executed. Safari 13.5 for instance evaluates the provided location header, see Figure 5.5.

Recommendation. To mitigate the above described issue, potential dangerous schemes should be forbidden for `redirect_uri` values, as they open an unnecessary attack surface and there is, to our knowledge, no reasonable use-case for these schemes.

5.2.1.6 [Bug] Internal Server Error When Processing "client_id" in Authentication Request

Description. When processing OpenID Connect `Authentication Requests`, Keycloak parses the `client_id` parameter. If the `client_id` is valid and includes curly braces ({ or }), Keycloak responds with an internal server error.

Steps to reproduce:

1. Set-up a new OpenID Connect client with a `client_id` including curly braces.

2. Send a valid `Authentication Request` to the `/auth/realms/<realm-name>/protocol/openid-connect/auth` endpoint:

```
1 GET /auth/realms/<realm-name>/protocol/openid-connect/auth?scope
  ↪ =openid&response_type=code&redirect_uri=[REDACTED]&state=[RE
  ↪ DACTED]&nonce=[REDACTED]&client_id=bitbucket.local%7b3-1%7d
  ↪ HTTP/1.1
2 Host: keycloak.local
3  [...]
```

Listing 5.7: Kecloak: `Authentication Response` including valid (pre-registered) `client_id` containing curly braces.

3. Keycloak responds with an internal server error with HTTP status code 500. The server log file reflects `3-1` which was included in our crafted `client_id`:

```
1 10:21:54,750 ERROR
  ↪ [org.keycloak.services.error.KeycloakErrorHandler] (default
  ↪ task-7) Uncaught server error:
  ↪ java.lang.IllegalArgumentException: RESTEASY003720: path
  ↪ param 3-1 has not been provided by the parameter map
2 [...]
```

Listing 5.8: Keycloak: The Internal Server Error leads to the above error being written to the application log, including the `3-1` that was part of the pre-registered `client_id`.

An authenticated user with administrative privileges may register a Client that uses an OpenID Connect-wise valid `client_id` that leads to a limited Denial-of-Service of Keycloak, as on each login flow using this `client_id` an internal server error occurs.

5.2.1.7 [Bug] "sector_identifier_uri" Supports HTTP Without TLS

Description. If OpenID Connect Dynamic Client Registration is used, the Client can specify multiple valid `redirect_uri` values by providing a `sector_identifier_uri` that "*references a JSON file containing an array of redirect_uri values*" [35, Section 5.]. The OpenID Connect Dynamic Client Registration specification additionally states that the "*value of the sector_identifier_uri MUST be a URL using the https scheme*" [35, Section 5.]. Keycloak instead accepts `sector_identifier_uri` with *HTTP* without TLS. As a result, the information Keycloak uses to determine

legitimate `redirect_uri` values is therefore potentially not integrity and confidentiality protected.

Steps to reproduce. The following steps need to be performed in order to repoduce the issue.

1. Launch a simple web server at `http://poc.local`

2. Send an authenticated request to dynamic client registration endpoint at `https://keycloak.local:8443/auth/realms/master/clients-registrations/openid-connect`

```
POST /auth/realms/master/clients-registrations/openid-connect
↪ HTTP/1.1
Host: keycloak.local:8443
Content-Type: application/json
Authorization: Bearer [REDACTED]
Content-Length: 162

{
   "redirect_uris":
     ["https://client.example.org/callback"],
   "sector_identifier_uri":
   "http://poc.local/sector_identifier.json"
}
```

Listing 5.9: Keycloak: Dynamic Client Registration with `sector_identifier_uri` using *http* (HTTP headers stripped).

3. The incoming request can be observed within the access log of the web server. Keycloak fetches the resource using plain *http*:

```
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:1337/) ...
127.0.0.1 - - [22/May/2020 17:22:48] "GET /sector_identifier.json
↪ HTTP/1.1" 200 -
```

Listing 5.10: Keycloak: Dynamic Client Registration with `sector_identifier_uri` using *http* – the listener receives an incoming request using plain *http*.

5.2.1.8 [Bug] Missing Check for "iss" Claim within JWT Validation on Client Authentication

Description. The OpenID Connect Core specification specifies a set of possible Client Authentication mechanisms. Among this set `client_secret_jwt` and `private_key_jwt` utilize JSON Web Tokens. According to the specification, "iss" and "sub" claim of the JWT must contain equally "*"client_id" of the OAuth Client*" if a JWT is used for Client Authentication [33, Section 9.].

An excerpt of Keycloak's Client Authentication Code in `/org/keycloak/authentication/authenticators/client/JWTClientAuthenticator.java` is shown below:

```
1 RealmModel realm = context.getRealm();
2 String clientId = token.getSubject();
3 if (clientId == null) {
4     throw new RuntimeException("Can't identify client. Issuer
      ↪  missing on JWT token");
5 }
```

Listing 5.11: Keycloak: The JWT validation for the Client Authentication determines the `clientId` solely based on the "sub" claim.

Keycloak solely determines the `client_id` based on the "sub" claim of the JWT and chooses the public key that is used to verify the token's signature based on this. In doing so the "iss" claim is completely ignored.

As there is no check for the "iss" value, one could assume a potential key confusion, as normally the public key to verify the token's signature is based on the "iss" claim. But as Keycloak uses the implicit equality of "sub" and "iss" claim defined within the specification, a malicious actor would still have to forge a signature for the victim-client's secret key in order to bypass the Client Authentication. Thus, this is only an issue regarding compliance with the specification, but does not result in an immediate security issue.

5.2.2 Analysis of GitLab

GitLab implements Service Provider and Identity Provider parts of the OpenID Connect specification. As Identity Provider it additionally distinguishes between user and system OAuth applications.

Section	Vulnerability Type	Attacker Model	Severity
5.2.2.1	Broken Authentication	Malicious Service Provider	Medium
5.2.2.2	IDOR	Malicious Administrative User	Low

Table 5.7: Overview of the most relevant observations and findings in GitLab's Identity Provider implementation.

5.2.2.1 [Broken Authentication] Missing "code" Parameter Invalidation After Permissions Are Revoked (CVE-2020-13294)

Attacker Model. The attacker model applied in the following is a malicious Service Provider (section 3.3).

Description. GitLab can be configured to act as OpenID Connect Identity Provider. If this is the case, in a Single Sign-On scenario a Relying Party allows GitLab users to sign in using their existing account. As defined at the OpenID Connect 1.0 specification, the user has to give explicit consent which information is shared with the relying party: "*Once the End-User is authenticated, the Authorization Server MUST obtain an authorization decision before releasing information to the Relying Party*" [33, Section 3.1.2.4.].

Additionally, GitLab allows users to revoke their consent for already granted accesses. This causes previously issued tokens to be invalidated. But if the Service Provider holds a valid `code` value in the moment the access is revoked, it can be still redeemed for a fresh `id_token` and `access_token`/`refresh_token` at GitLab's *Token Endpoint*. As a result, the revoked application reappears at the "Authorized applications" section having restored all previously granted permissions permanently.

Thus, a maliciously acting Service Provider could obtain sensitive data (`id_token`) and bearer tokens (`access_token`/`refresh_token`) without user's consent (as the consent was explicitly revoked beforehand). Even worse, the malicious actor may restore the previously given and then revoked consent persistently, resulting in future access to resources and information.

Recommendation. To mitigate this issue, beside `access_token` and `refresh_token`, issued `code` values must be invalidated if a user revokes the permissions of a Service Provider. In addition, the `code` must be one-time-usable and short-living.

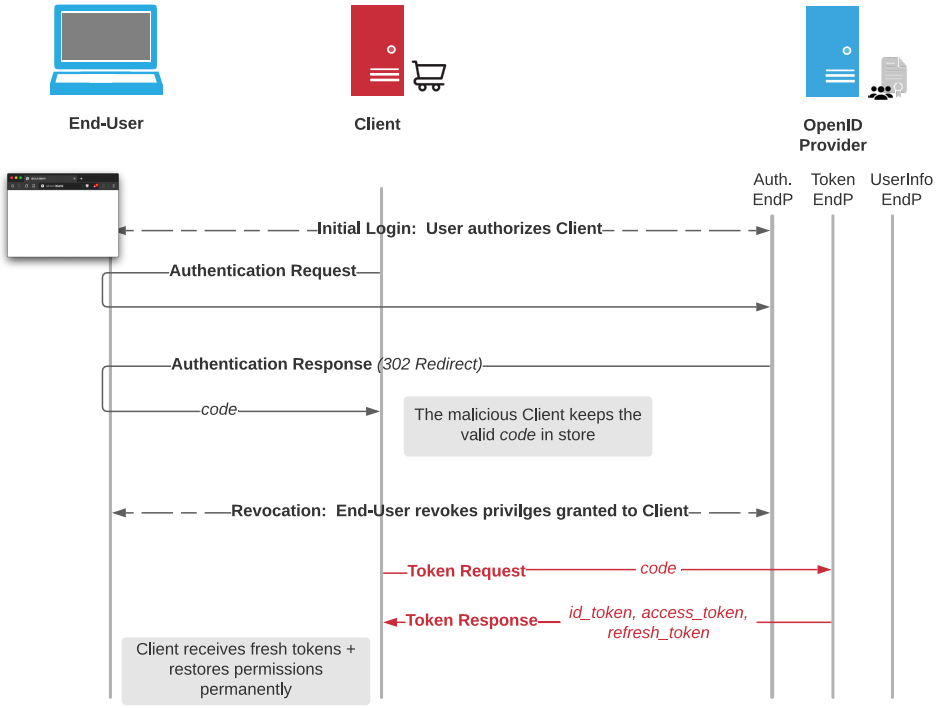

Figure 5.8: GitLab: Missing `code` revocation.

5.2.2.2 [IDOR] System OAuth Application Configuration Allows IDOR to User OAuth Applications

Attacker Model. The following issue could be exploited as an administrative user (section 3.4) that has access to the administrative configuration GUI of GitLab.

Description. In GitLab, users can register custom OAuth 2.0 Clients. Normally, only the user that created a Client has access to its configuration. By directly accessing the configuration page of an OAuth 2.0 application, administrative users can access and change the configuration of other user's OAuth 2.0 applications, even though the applications do not appear using the administrative GUI (as they are owned by another user) and should be access-restricted. This is caused by an Insecure Direct Object Reference (IDOR) flaw. Additionally, `client_id` and `client_secret` for these applications created by users can be obtained that should normally only be accessible for the owner of the application.

Steps to reproduce. To reproduce the behavior, the following steps can be performed:

1. As normal user, create a new application at `http://gitlab.local/oauth/applications`

2. As administrative user, observe that the new application is not visible at the administrative configuration pane.
3. As administrative user, observe that the new application is not configurable using `http://gitlab.local/oauth/applications/8` (adjust ID of Application). This is the direct link to the configuration of the application that the owner uses to configure the application.
4. As administrative user, access the configuration pane for the new application at `http://gitlab.local/admin/applications/8` and `http://gitlab.local/admin/applications/8/edit`. Note that the `/admin/applications/` path is normally used for system OAuth applications exclusively.

Recommendation. The actual impact of this missing access restriction appears to be limited, as administrative users may "impersonate" low privileged users and intentionally access their OAuth 2.0 Application's configuration panes. Nevertheless, the applications are not owned by the administrative user and `client_id` and `client_secret` are considered as sensitive data. Thus, a working access control must be implemented.

5.2.3 Analysis of Amazon Cognito (AWS)

As *Identity Broker*, Amazon Cognito also implements the Identity Provider part of the OpenID Connect specification.

Section	Vulnerability Type	Attacker Model	Severity
5.2.3.1	Insufficient Filtering	Malicious Service Provider or Malicious Administrative User	Low

Table 5.9: Overview of the most relevant observations and findings in Amazon Cognito's Identity Provider implementation.

5.2.3.1 [Insufficient Filtering] "redirect_uri" Is Allowed To Use Potentially Dangerous Schemes

Attacker Model. A malicious administrative user (section 3.4) or a malicious Service Provider (section 3.3) are considered as attacker models in the following.

Description. Amazon Cognito supports, just like previously described for Keycloak in subsubsection 5.2.1.5, dangerous schemes for registered `redirect_uris`.

In case of Amazon Cognito, multiple callback URIs are configured using one text input field being separated using commas. As a result, due to the comma-limitation no `data` URIs may be specified using this UI (otherwise the last part of a `data` URI is misunderstood as new URI without scheme), but other potential dangerous schemes like `javascript` are still allowed.

An example error response being sent to the `redirect_uri` endpoint using a `javascript` URI is shown in the following:

```
1 HTTP/1.1 302 Found
2 Date: Wed, 12 Aug 2020 16:57:50 GMT
3 Location: javascript:alert(1)//
  ↪ ?error_description=id_token+expired+at+1597250052
  ↪ &state=state_static&error=invalid_request
4  [...]
```

Listing 5.12: Amazon Cognito: Redirect to `javascript` URI.

Recommendation. To mitigate the above described issue, potential dangerous schemes should be forbidden for `redirect_uris`, as they open an unnecessary attack surface and there is to our knowledge no reasonable use-case for these schemes.

5.3 Service Provider Evaluation

Category	Keycloak	Bitbucket	GitLab	Salesforce	Amazon Cognito
Replay Attacks	✔	✔	○	○	○
Signature Manipulation	✔	○	✔	○	✔
Token Recipient Confusion	✔	✔	✔	○	✔
ID Spoofing	✔	●	✔	○	✔
Key Confusion	✔	○	✔	○	✔
Sub Claim Spoofing	✔	✔	✔	✔	✔
State Parameter	●	✔	✔	●	●
Leakage	●	●	○	✔	✔
CSRF	✔	●	✔	●	●
SSRF	○	○	●	✔	●
Open Redirect	○	○	✔	○	✔
Login Confusion	✔	●	✔	●	✔
Mix-Up Attacks	✔	N/A	N/A	✔	●
Deprecated Grants	✔	✔	✔	✔	○
Discovery	●	●	○	○	○
Injection	✔	●	●	○	✔

✔: Secure / No attack found in this category
●: Insecure / One or more attacks found in this category
○: Theoretical issues identified or limited attack success

Table 5.10: Evaluation of Service Provider related Test and Attack Categories.

Table 5.10 presents the evaluation results of the analyzed Service Provider implementations.

Replay Attacks Keycloak and Bitbucket do not show any issues regarding Replay Attacks, as both validate “exp”, “iat” and “nonce” claim of the JWT. GitLab introduced JWT validation in v13.2. Previous versions did not verify the contents of the `id_token` at all. From v13.2 GitLab verifies “exp” and `nonce` but ignores the “iat” claim. Salesforce does not use the provided `id_token` at all, but solely relies on the `UserInfo Response`. Amazon Cognito does not use a `nonce` to prevent replay and only validates the “exp” claim.

Signature Manipulation If it is performed, the signature validation is sufficient and does not accept the “none” algorithm. But Bitbucket and Salesforce do not verify the `id_token`’s signature at all. As both use the Code Flow, the validation of the signature is not mandatory because the `id_token` is exchanged using the trusted back-channel, according to specification.

ID Spoofing Bitbucket's OpenID Connect implementation allows ID spoofing by design. Bitbucket only supports one external login provider. This Identity Provider can authenticate any existing Bitbucket user by setting the public username as "sub" claim (or alternate username field that could be specified). Thus, a configured Identity Provider could "spoof" any user account including super-admin due to missing prior account linking. According to *Atlassian*, this is intended behavior. GitLab has a bug within its configuration parser, resulting in the theoretical possibility for "Issuer Confusion": If two OpenID Connect Identity Providers are configured and both use OpenID Connect Discovery, the frontend uses the description and labels of the first Identity Provider but the issuer and provided endpoints that are internally used are determined from Discovery with the second Identity Provider.

Key Confusion No Key Confusion issues were found, but as mentioned earlier, Bitbucket and Salesforce do not validate the `id_token`'s signature at all.

Subclaim Spoofing If parsed, the `id_token`'s "sub" claim is checked to be equal to the `UserInfo Response`'s "sub" claim within all implementations.

State Parameter Keycloak, Salesforce and Amazon Cognito did not invalidate the `state` parameter after it was redeemed once. As a result, the Denial-of-Service amplification attack introduced in this thesis could be launched using these Service Providers as amplifying proxies.

Leakage `Referer` Leaks were observed in Keycloak and Bitbucket. In both cases, no sufficient `Referrer-Policy` was defined. For Bitbucket there are references to external pages on the error page that include OpenID Connect parameters in its query parameters. Keycloak allows specifying harmless HTML that is rendered within the Login form. In doing so, external images are allowed, disclosing the `Authentication Request` as `Referer` header. GitLab has a CLI tool that does not require further authentication. Using `gitlab-psql`, all `access_token` values stored within the database are accessible.

CSRF Three out of five implementations implement an endpoint that starts the OpenID Connect flow without CSRF protection, resulting in Login CSRF into the user's own account if the user has an active session at the Identity Provider.

SSRF SSRF is a general problem in OpenID Connect setups if a malicious administrative user (section 3.4) is considered as part of the threat model. Keycloak and Bitbucket have blind SSRF issues that allow to send HTTP requests to localhost or internal IPs. As these products are self-hosted, we classify this as limited attack success. Additional to blind SSRF, malicious actors could inject arbitrary headers or split requests in GitLab. These Service Provider implementations are self-hosted so that the impact is generally considered less severe, as the blind SSRF issue Amazon Cognito has that allows sending requests to localhost in Amazon's hosted environment. Furthermore, Amazon

Cognito provides error messages resulting in the possibility to perform port scans and observe HTTP error codes.

Open Redirect If implemented, the non-normative parameter that specifies where the user should be redirected to after successful authentication is correctly validated to only redirect to allowed hosts. Bitbucket and Salesforce allow specifying any path of the trusted host as the redirection target, including session relevant and OpenID Connect flow relevant endpoints. Keycloak allows to pass a parameter along with the logout that redirects the End-User to any path at the Keycloak instance.

Login Confusion Bitbucket and Salesforce implement the necessary requirements for login confusion: An Endpoint that starts an OpenID Connect flow that is not CSRF protected and an Open Redirect or Covert redirect that allows specifying the *Login Initiation Endpoint* as the redirection target.

Mix-Up Attacks Most of the implementations that support to set up multiple Identity Providers do not suffer from Mix-Up issues, as Keycloak and Salesforce use a distinct endpoint per Identity Provider. Amazon Cognito uses one *Redirection Endpoint* for all configured Identity Providers and does not implement additional countermeasures, so that it is vulnerable for Mix-Up attacks.

Deprecated Grants Most Service-Provider implementations only support the Code Flow. Among the tested Service Providers, only Amazon Cognito supports the Implicit Flow in which `access_token` values are passed through the front-channel, but default is the Code Flow.

Discovery Bitbucket and Keycloak allow performing the OpenID Connect Discovery over HTTP without TLS. GitLab and Amazon Cognito enforce TLS for the *Configuration Endpoint*, all other OpenID Connect endpoints may use plain HTTP. In case of Amazon Cognito, the frontend validates the OpenID Connect endpoints regarding their scheme, but endpoints that are configured using OpenID Connect Discovery are allowed to use HTTP without TLS. Salesforce does not implement OpenID Connect Discovery but allows to manually specify endpoints using plain HTTP.

Injection Bitbucket throws an error in case the `access_token` includes a *CRLF* sequence when it tries to use this token within the `Token Request`. The stack trace is written to the application logs, including sensitive data like the `access_token`s. Salesforce is also a Java implementation and responds with an internal server error, but without access to application logs, it could not be proven that an equal error message is written to the logs. The most severe issue regarding *CRLF* sequences was observed in GitLab, as it uses the Identity Provider provided `access_token` within the context of HTTP headers and does not strip the *CRLF* sequence. Therefore, the injection of a *CRLF* sequence leads to an HTTP header and HTTP request injection within the `UserInfo Request`.

5.4 Service Provider Analysis Details

In the following, the most significant observations regarding Service Provider implementations are described in detail.

5.4.1 Analysis of Keycloak

Keycloak can be configured as *Identity Broker*, serving as proxy-like service for multiple external Identity Provider. In this scenario Keycloak acts as Relying Party to foreign Identity Providers, therefore Keycloak also implements the Service Provider specific parts of the OpenID Connect specification.

Section	Vulnerability Type	Attacker Model	Severity
5.4.1.1	Request Amplification	Web Attacker or Malicious Identity Provider	Low
5.4.1.2	Unhandled Exception	N/A	None

Table 5.11: Overview of the most relevant observations and findings in Keycloak's Service Provider implementation.

5.4.1.1 [Request Amplification] Reusable "state" Parameter at "redirect_uri" Endpoint Enables Multiple Attack Vectors (CVE-2020-14302)

Attacker Model For the following attack, two attacker models could be considered (three if you additionally consider the high privileged malicious administrative user (section 3.4)). A web attacker (section 3.1) having no influence on the target of the `Token Request` could only target the already configured Identity Provider. A malicious Identity Provider (section 3.2) using the OpenID Connect Discovery could additionally target arbitrary web servers accessible for the Service Provider.

Description Keycloak as Service Provider provides dedicated `redirect_uri`s for each Identity Provider: `https://keycloak.local/auth/realms/{realm}/broker/{alias}/endpoint`.

The `redirect_uri` endpoint does not invalidate `state` values if they are redeemed once. As a result, multiple requests to this endpoint including the valid `state` value result in requests to the configured *Token Endpoint* being initiated by Keycloak.

A malicious administrative user (section 3.4) or malicious Identity Provider (section 3.2) could additionally increase the outgoing `Token Request` by choosing a

long path for the *Token Endpoint* URL and setting imaginary long Client Credentials. As a result, one request to the Keycloak instance could be amplified multiple times compared to the request that is received at the victim server's end.

Steps to reproduce The steps to launch a Denial-of-Service attack using many large requests through Keycloak as amplifying proxy are the following:

1. Configure the victim server as *Token Endpoint* with a long path and long Client Credentials (if the attacker model is a regular web attacker (section 3.1), this step is skipped).

2. Start an OpenID Connect flow and obtain a valid `state` value (i.e. save a valid `Authentication Response` including the `state`).

3. After the `Authentication Response` is received, Keycloak tries to redeem the received `code` value at the victim web server. This request can be multiple times larger than the attacker's request (`Authentication Response`).

4. Replay the obtained `Authentication Response`, on each try Keycloak sends a new HTTP request to the victim server.

Recommendation To mitigate this issue, the `state` should not only be bound to the End-User's session, but additionally be a one-time-usable value.

5.4.1.2 [Bug] Unhandled NullPointerException If "state" is Missing in Authentication Response

Description. If Keycloak receives an `Authentication Response` after successful login at the Identity Provider that does not include an OpenID Connect `state` parameter, an unhandled *NullPointerException* occurs.

Steps to reproduce:

1. Send a request to Keycloak's Redirect URI Endpoint without *state* parameter:

```
1 GET /auth/realms/master/broker/midp/endpoint?code=aabbccddeeff
  ↪ HTTP/1.1
2 Host: keycloak-sp.local:9443
```

Listing 5.13: Keycloak: Missing *state* at Authentication Response (Request).

2. Observe the HTTP 502 error code of the response:

```
HTTP/1.1 502 Bad Gateway
[...]

[...]
<header class="login-pf-header">
        <h1 id="kc-page-title"> We are sorry...</h1>
</header>
<div id="kc-content">
<div id="kc-content-wrapper">
<div id="kc-error-message">
    <p class="instruction">Unexpected error when authenticating
    ↪  with identity provider</p>
</div>
[...]
```

Listing 5.14: Keycloak: Missing *state* at Authentication Response (Response).

And the corresponding log entries:

```
11:10:11,370 ERROR
↪  [org.keycloak.broker.oidc.AbstractOAuth2IdentityProvider]
↪  (default task-11) Failed to make identity provider oauth
↪  callback: java.lang.NullPointerException
```

Listing 5.15: Keycloak: Missing *state* at Authentication Response (application log).

Recommendation. Keycloak always utilizes the `state` value. Nevertheless, the error case if an Identity Provider omits the `state` within the `Authentication Response` should be handled and an adequate error message should be displayed.

5.4.2 Analysis of Bitbucket

Bitbucket only implements the Service Provider part of the OpenID Connect specification. It distinguishes between two authentication modes:

a) An external Identity Provider is used as secondary authentication provider.

b) An external Identity Provider is used as primary authentication provider disabling access using credentials.

Section	Vulnerability Type	Attacker Model	Severity
5.4.2.1	SSRF	Malicious Administrative User	Medium
5.4.2.2	Login CSRF	Web Attacker	Medium
5.4.2.3	Login Confusion	Web Attacker	Medium
5.4.2.4	Sensitive Information Disclosure	N/A	Low
5.4.2.5	Missing Best Practice	Malicious Administrative User or Web Attacker with Code Execution	Low
5.4.2.6	Log Injection	Malicious Identity Provider	Low
5.4.2.7	Unhandled Exception	N/A	None

Table 5.12: Overview of the most relevant observations and findings in Bitbucket's Service Provider implementation.

5.4.2.1 [SSRF] Filter Bypassed: OpenID Connect Configuration Allows Authenticated SSRF to Localhost

Attacker Model. The attacker model that is applied for this attack is a malicious administrative user (section 3.4) that is allowed to configure the OpenID Connect and "SSO 2.0" settings of the Bitbucket instance.

Description. Bitbucket does not validate the "Issuer URL" in accordance to the specification that requires an URL that is "*using the https scheme with no query or fragment component that the OP asserts as its Issuer Identifier*" [34, Section 3.]. Additionally, Bitbucket does not remove the questionmark character "?" from the "Issuer URL". As a result, it is possible to omit the last part of the URL that is appended by Bitbucket on `Configuration Request` and therefore send requests to arbitrary URL paths.

The OpenID Connect Discovery can be considered as intentional SSRF for authenticated users with administrative privileges. To mitigate the risk that is caused by this design, there is a blacklist in place that prevents administrative users from setting the host of the issuer to `localhost` or `127.0.0.1`, because there could be services running on localhost that must not be accessible to external entities. This

filter can be bypassed using IPv4-mapped IPv6 addresses [22, Section 2.2] as host name, additionally arbitrary ports can be specified.

Thus, combining the two flaws, an authenticated administrative user (section 3.4) can launch a SSRF attack against Bitbucket targeting services that are running on localhost by setting the "Issuer URL" to `http://[0:0:0:0:0:ffff:127.0.0.1]:1337/testfile?`. This results in a GET request over HTTP to a service that is accessible on localhost port 1337 and path `/testfile`.

The incoming request on an HTTP listener running on localhost as presented within the access log is given in the following:

```
1 Serving HTTP on 0.0.0.0 port 1337 ...
2 127.0.0.1 - - [27/Apr/2020 18:10:51] "GET
  ↪ /testfile?/.well-known/openid-configuration HTTP/1.1" 404 -
```

Listing 5.16: Bitbucket: SSRF to localhost on arbitrary port with arbitrary path.

Recommendation. To mitigate this issue, the existing blacklist should be hardened regarding IPv4-mapped IPv6 addresses. Additionally, the issuer must not include a query or fragment component.

5.4.2.2 [CSRF] Intentional Login CSRF

Attacker Model. In the following the web attacker model (section 3.1) is considered.

Description. The OpenID Connect 1.0 Core specification defines a mechanism how third parties may trigger an OpenID Connect login flow. The specification introduces the "target_link_uri" that instructs the Service Provider where the user should be redirected after successful login [33, Section 4.].
Additionally, Bitbucket implements a non-normative endpoint that starts an OpenID Connect login flow if an Identity Provider is configured as external login provider. This endpoint accepts the *next* GET parameter that instructs Bitbucket where the user should be redirected after successful login.

Both endpoints do intentionally not implement CSRF mitigations, so that with a request to `https://bitbucket.test/plugins/servlet/oidc/initiate-login` or `https://bitbucket.test/plugins/servlet/external-login` an OpenID Connect login flow is started. If a user already has an active session at the Identity Provider and previously consented to share information with Bitbucket as Service

Provider, the Identity Provider immediately responds with the `Authentication Response` when the *Authentication Endpoint* is queried.

As a result, in case a user has an active session at the Identity Provider, a third party can login the user to his Single Sign-On connected Bitbucket account using this login CSRF attack. The attack is shown in Figure 5.13.

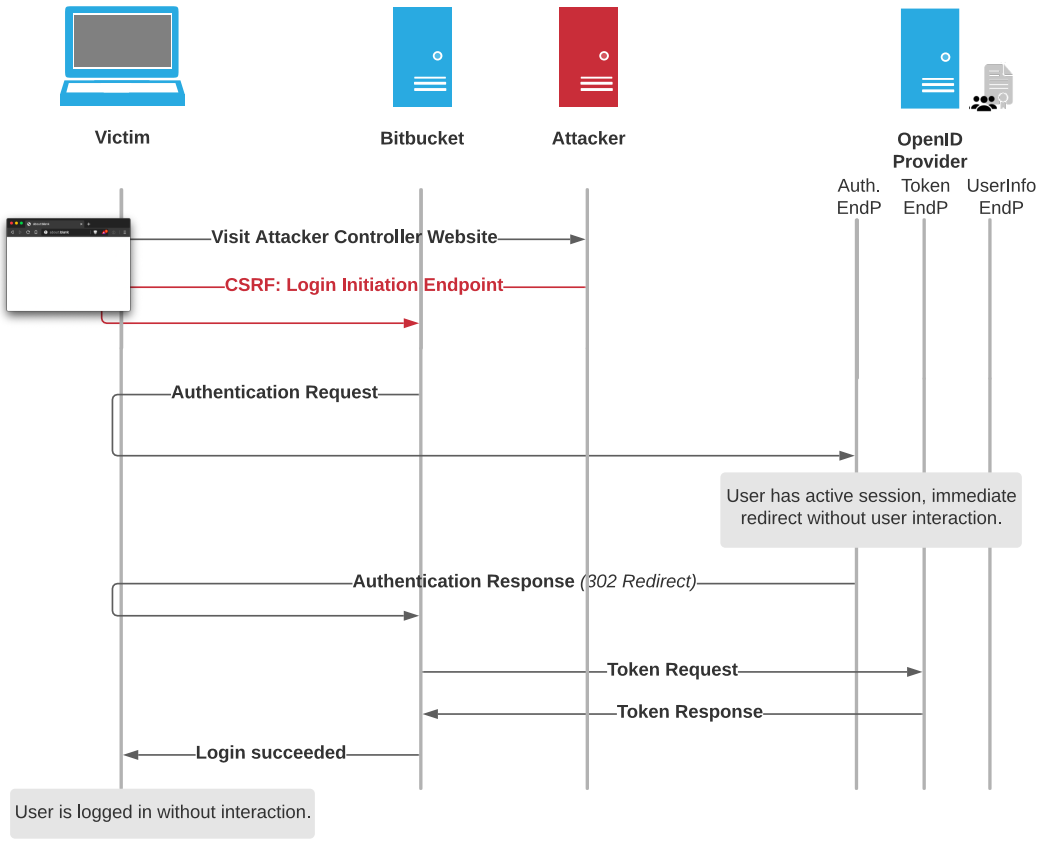

Figure 5.13: Bitbucket: A malicious actor can start an OpenID Connect authentication flow resulting in login CSRF.

As OpenID Connect is a complex protocol with multiple parties that highly depends on redirects being made in the user's browser, the risk resulting from CSRF vulnerabilities significantly increases. The "target_link_uri" and "next" parameters for instance allow to chain an authenticated request after the authentication was successful. In doing so, the `Referer` of this request is a trusted origin, potentially bypassing additional referrer-based CSRF mitigations.

Recommendation. Fett et al. already recommended in 2017 that "*login initiation endpoints [should] not to be implemented (they are not a mandatory feature), or to require explicit confirmation by the user*" [15, Section III; A. 7)].

5.4.2.3 [Login Confusion] Redirect + CSRF Enable Login Confusion

Attacker Model. The attacker model for this attack is an unauthenticated web attacker (section 3.1). We assume that the victim browses an attacker controlled website. Prerequisites for this attack are that the victim has two accounts at Bitbucket. For *user A* our victim uses credentials, *user B* is authenticated using an External Login Provider. Additionally, the victim has an active session at the Identity Provider.

Description. As previously described in subsubsection 5.4.2.2, Bitbucket implements two endpoints that enable intentional login CSRF. Furthermore, both endpoints allow GET parameters which hold information where the user should be redirected to after successful authentication. Whilst this parameter is sanitized to prevent Covert Redirects to external destinations, any path of the Bitbucket instance is allowed as final destination, including session related endpoints like the *Logout Endpoint* (that terminates the user's session) or the Single Sign-On *Login Initiation Endpoint*. This enables the following Login Confusion attack, that is visualized in Figure 5.14:

1. Prerequisite: The victim has an active session at the Identity Provider and previously used this account to authenticate as *user B* at Bitbucket.

2. The victim visits an attacker controlled website and is redirected to `https://bitbucket.test/login?next=/plugins/servlet/external-login`.

3. Bitbucket serves a regular Login Form, the victim authenticates using her credentials for *user A* (1).

4. After successful authentication, Bitbucket redirects the victim to the *Login Initialization Endpoint*, as this was the previously provided destination (2).

5. Bitbucket receives the GET request to the *Login Initiation Endpoint* and launches a new OpenID Connect Login flow accordingly.

6. The victim has, per our prerequisites, an active session at the Identity Provider. As a result, there is a silent redirect with the `Authentication Response` to the `redirect_uri` endpoint including valid `code` and `state` parameters.

7. Bitbucket validates the `state` and redeems the `code` at the Identity Provider's *Token Endpoint*. The Identity Provider responds with `id_token` and `access_token`.

8. After validating the identity of *user B*, the victim is logged in as *user B*, although she entered her credentials for *user A* and did not perform any additional interactions.

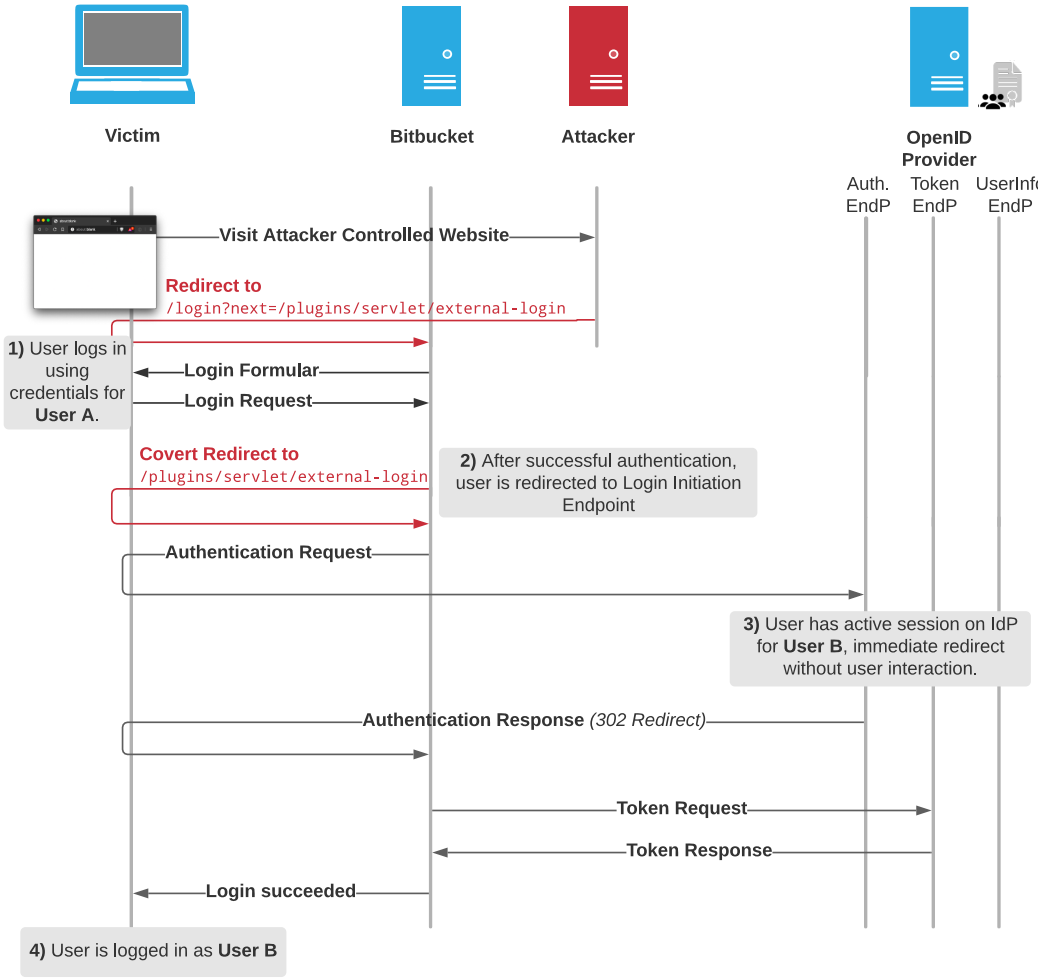

Figure 5.14: Bitbucket: A malicious actor can utilize a login CSRF vulnerability combined with the `next` parameter to launch a Login Confusion attack.

To conclude the previous steps: The victim authenticates using credentials for **user A** but is then logged in as **user B** after multiple intransparent redirects and without further user interaction. This underlines the conclusion Fett et al. [15, Section III; A. 7)] drew in 2017: Endpoints that provide OpenID Connect related functionality require Cross-Site-Request-Forgery protection.

Recommendation. To mitigate this behavior, the *Initiate Login Endpoint* and *External Login Endpoints* should be removed entirely. Alternatively, token- or referrer-based mitigations could be implemented. Furthermore should endpoints that have effects on the session be entirely excluded from "target_link_uri" and "next" parameters, as a silent request to these endpoints has unforeseen effects on the current user session and the application's behavior.

5.4.2.4 [Sensitive Information Disclosure] Referer Leaks OIDC "code" Parameter to Third Parties

Attacker Model. Even though the following section describes a flaw that leads to disclosure of sensitive information to third parties, there is no actual active attacker present. Due to the overall setup as self-hosted service, *Atlassian* normally does not receive any information about ongoing OpenID Connect authentication processes. Nevertheless, the below presented flaw leads to disclosure of sensitive OpenID Connect parameters to *Atlassian*.

Description. During authentication using a custom OpenID Connect Identity Provider, Bitbucket returns an error after the user was sent to the `redirect_uri` in case the user's identifier (i.e. "sub" claim of the OIDC `id_token`) can not be found in Bitbucket's user base. The following error is written to the application log:

```
1 2020-05-04 13:29:44,644 ERROR [http-nio-7990-exec-4]
  ↪ @1GWJER3x809x186x0 16qfs24 172.17.0.1,172.17.0.1 "GET
  ↪ /plugins/servlet/oidc/callback HTTP/1.1"
  ↪ c.a.p.a.i.w.f.ErrorHandlingFilter Received SSO request for
  ↪ user [sub value], but the user doesn't exist in the product
```

Listing 5.17: Bitbucket: User could not be found in user base.

As a result the user receives a generic error message (Figure 5.15, Listing 5.18) that is displayed as direct response to the OpenID Connect `Authentication Response` redirect, so that the URL still includes sensitive OpenID Connect parameters like `code` and `state`: https://bitbucket.local/plugins/servlet/oidc/callback?state=[REDACTED]&session_state=[REDACTED]&code=[REDACTED].
The `code` value is very valuable, as it is used by the Service Provider to obtain the `access_token` and `id_token` from the Identity Provider. Furthermore, the error page includes references to external resources. Bitbucket Server is a self-hosted software, so that *Atlassian* can be considered as third party in this case and should not receive any OpenID Connect parameters. As there is no `Referrer-Policy` present on this error page, the full path of the referring website including OpenID Connect `state` and `code` is disclosed to third parties if a user clicks one of the links to external resources, as shown in Listing 5.19.

Recommendation. To mitigate this issue, a proper `Referrer-Policy` should be specified, so that sensitive OpenID Connect related information is not disclosed to third parties. Additionally, a redirect to a neutral error page like `https://example.com/error` would prevent leakage for outdated browsers that do not understand the `Referrer-Policy` header.

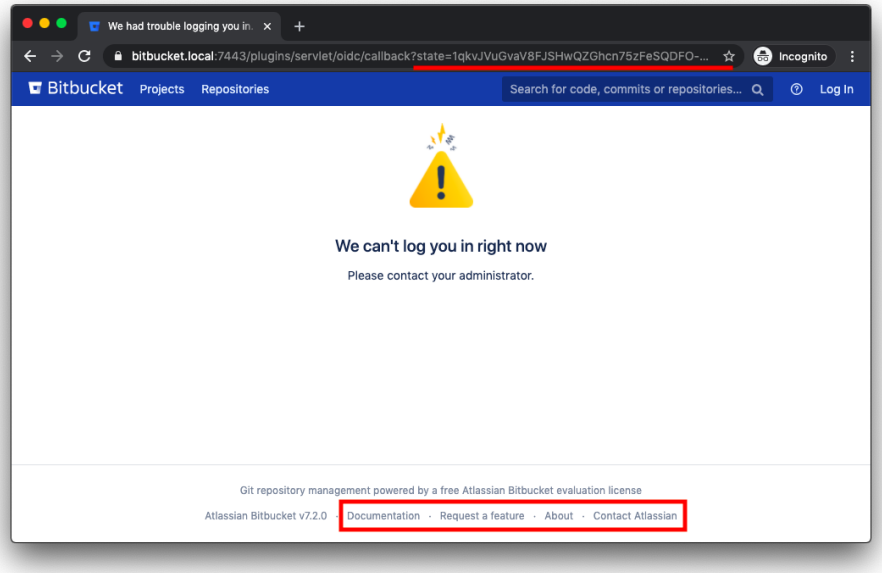

Figure 5.15: Bitbucket: A generic error message is shown if the user does not exist, including links to external resources (Screenshot).

```
1 <ul>
2   <li data-key="footer.license.free.eval"> [...] <a
    ↪  href="http://www.atlassian.com/software/bitbucket/">
    ↪  Atlassian Bitbucket</a> evaluation license</li>
3 </ul><ul> [...]
4   <li data-key="footer.links.documentation"><a
    ↪  href="http://docs.atlassian.com/bitbucketserver/ docs-072/
    ↪  Bitbucket+Server+documentation?utm_campaign=in-app-help
    ↪  &amp;utm_medium=in-app-help&amp;utm_source=stash"
    ↪  target="_blank">Documentation</a></li>
5   <li data-key="footer.links.jac"><a
    ↪  href="https://jira.atlassian.com/browse/BSERV"
    ↪  target="_blank">Request a feature</a></li>
6         [...]
7 </ul>
```

Listing 5.18: Bitbucket: A generic error message is shown if the user does not exist, including links to external resources (Sources).

```
GET /browse/BSERV HTTP/1.1
Host: jira.atlassian.com
Referer:
 ↪ https://bitbucket.local:7443/plugins/servlet/oidc/callback
 ↪ ?state=DJoFR895ZYP59zZm1rwdqjXgfcrwzqf0u1gEXNZPWC8
 ↪ &session_state=ce18a9f7-3979-4d2c-bc45-d989eab882ee
 ↪ &code=6cb99d4a-c1e0-46dd-91e9-2c4aa0b052ed.[...]
 [...]
```

Listing 5.19: Bitbucket: Third Party request including sensitive information within the `Referer` header.

5.4.2.5 [Missing Best Practice] Default Password for Java Keystore

Attacker Model. The following section describes a security misconfiguration. In order to exploit this, either a malicious administrative user (section 3.4) could be considered that already has high privileges on the system leading to near negligible impact. Alternatively, a web attacker (section 3.1) that gains code execution on the Service Provider's host can be considered. In this case, the malicious entity could effectively break the OpenID Connect setup by adding his certificate to the truststore.

Description. Docker installations of Bitbucket Server use the default Java keystore with default password to establish trust on remote connections. This truststore is for instance used for Single Sign-On requests to Identity Providers.

Steps to reproduce. In order to reproduce the issue, the following steps need to be performed:

1. Create new container "bitbucket-test":

```
docker run --name="bitbucket-test" -p 7990:7990 -p 7999:7999
 ↪ atlassian/bitbucket-server
```

Listing 5.20: Bitbucket: Default password for Java keystore (Step 1).

2. Wait until container is up and running. Then spawn shell inside container:

```
# docker exec -it bitbucket-test bash
root@f5dbfd8dd533:/var/atlassian/application-data/bitbucket#
```

Listing 5.21: Bitbucket: Default password for Java keystore (Step 2).

3. Access default keystore with default password *changeit*:

```
root@f5dbfd8dd533:/var/atlassian/application-data/bitbucket# file
↪ /opt/java/openjdk/jre/lib/security/cacerts
/opt/java/openjdk/jre/lib/security/cacerts: Java KeyStore
root@f5dbfd8dd533:/var/atlassian/application-data/bitbucket#
↪ keytool -list -keystore
↪ /opt/java/openjdk/jre/lib/security/cacerts
Enter keystore password:
Keystore type: jks
Keystore provider: SUN
Your keystore contains 93 entries
verisignclass2g2ca [jdk], May 18, 1998, trustedCertEntry,
Certificate fingerprint (SHA1):
↪ B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
[...]
```

Listing 5.22: Bitbucket: Default password for Java keystore (Step 3).

A malicious actor with code execution on the Bitbucket instance can import his own certificate that then would be accepted by Bitbucket Server for OpenID Connect back-channel communication:

```
# keytool -import -file /tmp/malicious.der -keystore
↪ $JAVA_HOME/jre/lib/security/cacerts
```

Listing 5.23: Bitbucket: Default password for Java keystore (Step 4).

For Single Sign-On scenarios, it is crucial that the Service Provider – in this case Bitbucket Server – is able to create a trusted connection to the Identity Provider.

Otherwise, a malicious actor may perform impersonation attacks on the Single Sign-On flow. The OpenID Connect specification elaborates on this in [34, Section 7.2.].

As Bitbucket Server uses the default truststore instead of a custom truststore with unique strong password, modifying the default truststore with known credentials changes the trust behaviour of the Bitbucket Server installation.

Recommendation. To mitigate the risk, a unique strong password should be set for the truststore on creation of the Docker container.

5.4.2.6 [Log Injection] Insufficient filtering of "access_token" in UserInfo Request

Attacker Model. The attacker model for this attack is a malicious Identity Provider (section 3.2) that provides malicious information to a victim Service Provider.

Description. If a custom "Username claim" is specified, Bitbucket queries the *UserInfo Endpoint* using the previously obtained `access_token`. In doing so, Bitbucket fails to properly sanitize the bearer token before setting the HTTP header to its value. As a result, if the token includes a *CRLF* sequence, an unhandled exception occurs in `sun.net.www.protocol.http.HttpURLConnection.checkMessageHeader` caused by the illegal characters. This exception including the `access_token` is written in plain text and without stripping sensitive information or dangerous characters to the application log file situated at `/var/atlassian/-application-data/-bitbucket/log/atlassian-bitbucket.log`. An example is shown in Listing 5.24.

```
1 2020-07-09 21:31:23,098 ERROR [http-nio-7990-exec-10]
  ↪ o.a.c.c.C.[.[.[/].[plugins] Servlet.service() for servlet
  ↪ [plugins] in context with path [] threw exception
2 java.lang.IllegalArgumentException: Illegal character(s) in
  ↪ message header value: Bearer
  ↪ AccessTok[HERE IS THE CRLF SEQUENCE]
3 en
4   at sun.net.www.protocol.http.HttpURLConnection
    ↪ .checkMessageHeader(HttpURLConnection.java:542)
5 [...]
```

Listing 5.24: Bitbucket: Injected *CRLF* sequences are reflected within the application log file (I).

Beside the storage of sensitive information in log files, this yields additional security risks. Bitbucket comes with a GUI log analyzer that parses the application

log. Using the log injection, it is possible to spoof log entries that are rendered and presented to administrative users. As the log analyzer uses regular expressions to search for known issues and only renders a trusted description and markup (no attacker controlled contents), no further injection issues within the frontend representation of injected log entries could be observed. Nevertheless, if the log analyzer is configured to run periodically and sends mails on recognized errors, a malicious actor can trick Bitbucket to send notification mails to administrative users.

Additionally, there are other situations when Bitbucket does not sanitize Identity Provider provided contents and writes these values without any sanitization to the application logs. For instance, if Bitbucket is not able to map the provided user identifier (either from `id_token` or `UserInfo` depending on the "Username claim" configuration) the observed identifier is logged to the application log. In doing so, *CRLF* sequences are not sanitized and are directly written to the application log. An example is shown in Listing 5.25.

```
1 2020-07-10 16:19:03,191 ERROR [http-nio-7990-exec-1]
  ↪ @1PUTJUQx979x631x0 tu2ic5 172.17.0.1,172.17.0.5 "GET
  ↪ /plugins/servlet/oidc/callback HTTP/1.1"
  ↪ c.a.p.a.i.w.f.ErrorHandlingFilter Received SSO request for
  ↪ user Toni Te[HERE IS THE CRLF SEQUENCE]
2 st, but the user doesn't exist in the product
3 com.atlassian.plugins.authentication.impl.web.usercontext
  ↪ .AuthenticationFailedException: Received SSO request for user
  ↪ Toni Te[HERE IS THE CRLF SEQUENCE]
4 st, but the user doesn't exist in the product
5 [...]
```

Listing 5.25: Bitbucket: Injected *CRLF* sequences are reflected within the application log file (II).

Recommendation. To mitigate this issue, Identity Provider provided values need to be carefully sanitized. Special attention needs to be payed to the context in which the values are used.

5.4.2.7 [Bug] Unhandled NullPointerException if "state" Is Missing In Authentication Response

Description. Bitbucket itself issues a `state` on each `Authentication Request`. Thus, an Identity Provider is supposed to include this `state` in the `Authentication`

`Response`, in order to prevent CSRF attacks. If the `state` is not valid, Bitbucket correctly shows a corresponding error message, but if the parameter is entirely missing, there is an internal server error caused by a *NullPointerException*.

The request given in Listing 5.26 could be send to the `redirect_uri` endpoint with arbitrary `code` value.

```
1 GET /plugins/servlet/oidc/callback?code=test HTTP/1.1
2 Host: bitbucket.local:7443
```

Listing 5.26: Bitbucket: `Authentication Response` without `state` parameter.

The stacktrace given in listing 5.27 is logged to Bitbucket's application log file.

```
1 2020-05-15 13:40:41,470 ERROR [http-nio-7990-exec-9]
  ↪ @SNOOPWx820x116x0 172.17.0.1,172.17.0.1 "GET /mvc/error500
  ↪ HTTP/1.1" c.a.s.i.web.ErrorPageController There was an
  ↪ unhandled exception loading [/plugins/servlet/oidc/callback]
2 java.lang.NullPointerException: null
3   at com.atlassian.plugins.authentication.impl
    ↪ .web.oidc.OidcConsumerServlet
    ↪ .doGet(OidcConsumerServlet.java:102)
4   at com.atlassian.applinks.core.rest.context
    ↪ .ContextFilter.doFilter(ContextFilter.java:24)
5   at com.atlassian.applinks.core.rest.context
    ↪ .ContextFilter.doFilter(ContextFilter.java:24)
6   at com.atlassian.applinks.core.rest.context
    ↪ .ContextFilter.doFilter(ContextFilter.java:24)
7 [...]
```

Listing 5.27: Bitbucket: Stacktrace of the uncatched *NullPointerException* that is thrown if no `state` parameter is present within `Authentication Response`.

Reverse Engineering of the closed source application reveals the following code snippet of the *doGet()* method that is responsible for the `state` validation, given in Listing 5.28. The implementation does not consider the case "no `state` is provided" (in line 7 only the case is catched that the state is present but unknown):

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws IOException {
    this.applicationStateValidator.checkCanProcess
        AuthenticationRequest();
    AuthenticationSuccessResponse successResponse =
        parseResponse(request);

    OidcConfig oidcConfig =
        this.ssoConfigService.getOidcConfigOrFail();

    SessionData sessionData =
        (SessionData)this.sessionDataService.getSessionData(request,
        response,
        successResponse.getState().getValue()).orElseThrow(() ->
        new AuthenticationFailedException("Unknown state in
        response"));
[..]
```

Listing 5.28: Bitbucket: Missing handling of *NullPointerException*s if no `state` is present within `Authentication Response`.

Recommendation. To mitigate this issue, the *NullPointerException* needs to be catched and an adequate error message needs to be passed to the user interface.

5.4.3 Analysis of GitLab

GitLab can be configured to use external login providers using various protocols, including LDAP, SAML and OpenID Connect. In the following, observations regarding the OpenID Connect Client implementation are presented.

Section	Vulnerability Type	Attacker Model	Severity
5.4.3.1	CRLF Injection and SSRF	Malicious Identity Provider	High - Critical*
5.4.3.2	Unencrypted Communication	Man-in-the-Middle	Low
5.4.3.3	Parser Error	Malicious Administrative User	Low

**Depending on the actual setup, this can be escalated to Remote Code Execution.*

Table 5.16: Overview of the most relevant observations and findings in GitLab's Service Provider implementation.

5.4.3.1 [CRLF Injection] HTTP Header And HTTP Request Injection in UserInfo Request

Attacker Model. For this attack, the attacker model is a malicious Identity Provider (section 3.2).

Description. If GitLab is configured to use an external OpenID Connect login provider, it obtains the `access_token` from the Identity Provider's *Token Endpoint* and redeems this token as bearer token when requesting the Identity Provider's *UserInfo Endpoint*. In doing so, GitLab does not correctly sanitize the token value for usage in the context of HTTP headers. As a result, a malicious Identity Provider can inject *CRLF* sequences into the `access_token` that are then directly injected into the request that is performed by GitLab, resulting in HTTP header injection and HTTP request splitting.
Additionally, the target of the `UserInfo Request` can be arbitrarily chosen without restrictions regarding localhost or private IPs within the `Configuration Response`, so that the malicious Identity Provider can send arbitrary controlled HTTP requests (verb, headers, body) to arbitrary internal hosts, external hosts or localhost.
An attack flow is presented in Figure 5.17. Note that for illustration purposes, the target of the `UserInfo Request` was not altered, but could point to an arbitrary host. If a Redis instance is present either on localhost or within the internal network, the impact of this attack can even be escalated to Remote Code Execution, as it was previously shown for other SSRF vulnerabilities with the ability to add HTTP headers in the context of GitLab [19].

Recommendation. It is recommended to mitigate this issue by carefully sanitizing Identity Provider provided values. Special attention needs do be paid to the values that are injected into HTTP headers.

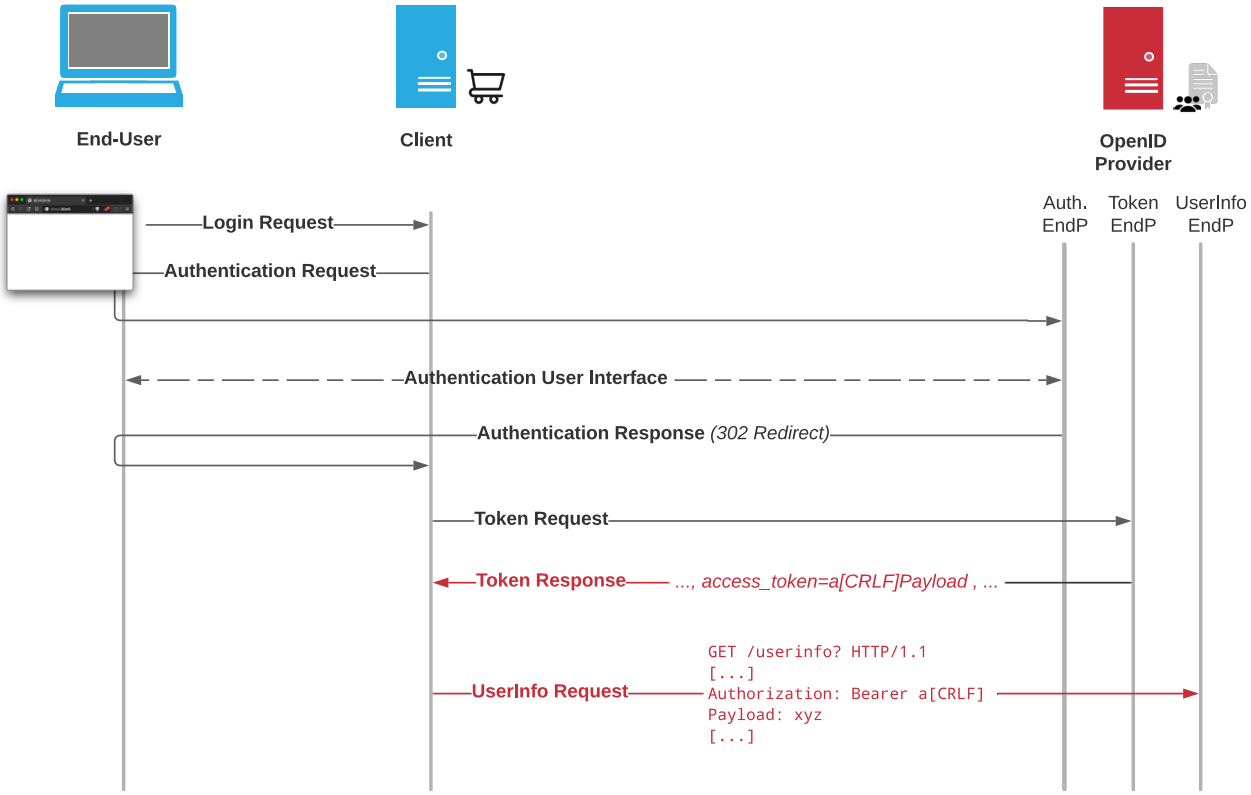

Figure 5.17: GitLab: CRLF injection leads to request splitting.

5.4.3.2 [Unencrypted Communication] "id_token" Not Used And HTTP Without TLS Supported

Attacker Model. The following section outlines a violation of the specification that enables a man-in-the-middle attacker (section 3.5) to influence the authentication process. Notably, GitLab supports TLS but does not enforce it, so that the Service Provider could be configured in a more secure manner than described below. With v13.2 GitLab coincidentally introduced `id_token` validation, just few days after we noticed and reported that this was completely missing. Therefore the `id_token` validation part in the following only applies for versions prior v13.2.

Description. OpenID Connect is an OAuth 2.0 based protocol that adds an identity layer on top of OAuth 2.0 [33]. Hence, in contrast to OAuth 2.0, which is intended to be used for authorization, OpenID Connect is intended for authentication purposes. As extension for OAuth 2.0, OpenID Connect introduces the `id_token` and the *UserInfo Endpoint*.

If the *openid_connect* "omniauth_provider" is used, GitLab behaves not compliant to the OpenID Connect specification, as the `id_token` is completely omitted and only information obtained from the *UserInfo Endpoint* is used. In contrast, the specification requires that "*Clients MUST validate the ID Token in the Token Response*" [33, Section 3.1.3.7.] and that "*The sub Claim in the UserInfo Response MUST be verified to exactly match the sub Claim in the ID Token; if they do not match, the UserInfo Response values MUST NOT be used*" [33, Section 5.3.2.].

Additionally, the `UserInfo Response` can be optionally signed or encrypted according to the specification [33, Section 5.3.2.], which is not enforced by GitLab either. According to specification, "*Communication with the UserInfo Endpoint MUST utilize TLS*" [33, Section 5.3.], but GitLab supports *http* and *https*.

Thus, GitLab relies on the potentially neither integrity nor confidentiality protected `UserInfo Response`. If the GitLab instance is configured to communicate with the Identity Provider using plain HTTP, a man-in-the-middle attacker can arbitrarily modify the `UserInfo Response` and bypass the authentication.

Recommendation. To mitigate this issue, TLS must be enforced for communication with Identity Provider endpoints. In addition, the `id_token` should be parsed and validated in accordance to the validation steps required within the specification [33, Section 3.1.3.7.].

5.4.3.3 [Bug] Erroneous OpenID Connect Configuration Parsing

Attacker Model. The following bug could be theoretically exploited by a malicious administrative user (section 3.4) that has permissions to configure OpenID Connect Identity Providers. As the Identity Provider's Configuration Endpoint is fetched, an Identity Provider that is compromised or becomes rogue at some point in time after initial configuration could additionally benefit from this bug, if the erroneous configuration was not initially observed.

Description. GitLab's documentation does not clearly indicate if one or more external OpenID Connect Identity Providers can be configured. It was observed that if two Identity Providers are configured at a time and the second Identity Provider has OpenID Connect Discovery enabled, the second Identity Provider can overwrite the first Identity Provider's endpoints, even though the frontend shows the "label" of the first Identity Provider.

A syntactical correct excerpt of a *docker-compose.yml* file is given in listing 5.29.

```
environment:
    GITLAB_OMNIBUS_CONFIG: |
      external_url 'http://gitlab.local'
      gitlab_rails['omniauth_allow_single_sign_on'] = true
      gitlab_rails['omniauth_block_auto_created_users'] = false
      gitlab_rails['omniauth_external_providers'] =
      ↪  ['openid_connect']
      gitlab_rails['omniauth_allow_bypass_two_factor'] =
      ↪  ['openid_connect']
      gitlab_rails['omniauth_providers'] = [
        {
          'name' => 'openid_connect',
          'label' => 'Correct IdP',
          'args' => {
            'name' => 'openid_connect',
            'scope' => ['openid', 'profile'],
            'response_type' => 'code',
            'issuer' => 'https://a.com',
            'discovery' => true,
            'client_auth_method' => 'query',
            'client_options' => {
              'identifier' => 'test.test',
              'secret' => 'password',
              'redirect_uri' => 'http://gitlab.local/users/auth/
              ↪  openid_connect/callback'
            }
          }
        },
        {
          'name' => 'openid_connect',
          'label' => 'Malicious IdP',
          'args' => {
            'name' => 'openid_connect',
            'scope' => ['openid', 'profile'],
            'response_type' => 'code',
            'issuer' => 'https://b.com',
            'discovery' => true,
            'client_auth_method' => 'query',
            'client_options' => {
              'identifier' => 'test.test',
              'secret' => 'password2',
              'redirect_uri' => 'http://gitlab.local/users/auth/
              ↪  openid_connect/callback'
            }
          }
        }
      ]
```

Listing 5.29: GitLab: *docker-compose.yml* including multiple OpenID Connect Identity Providers.

GitLab uses the OpenID *Configuration Endpoint* of the second Identity Provider displaying the "label" of the first Provider on the *Login Endpoint*, so that a malicious Identity Provider could theoretically launch an Identity Spoofing attack. In doing so, the malicious Identity Provider would adapt the benign Identity Provider's configuration with exception of the *Token Endpoint*. As a result, the user would start the OpenID Connect flow with the benign Identity Provider that is indicated in the frontend and also configured as first Identity Provider within the configuration. After successful authentication using the Code Flow, GitLab would receive the `code` and redeem it with its Client Credentials at the *Token Endpoint*. This endpoint is controlled by the malicious Identity Provider, so that GitLab would send the valid *code* to an attacker-controlled server.
This observation is considered a bug rather than a vulnerability, because based on our observations, the configuration is only parsed on startup of the Docker container. Thus, only administrative users on the GitLab host can specify an erroneous configuration that would enable a malicious Identity Provider to launch an Identity Spoofing attack. These administrative users already are in charge of high permissions and can control the complete GitLab instance, so that the only scenario in which a third party would benefit from this behavior is, if an administrative user triggers the behavior by accident and does not notice the Mix-Up between the two Identity Providers.

Recommendation. To mitigate the above described issue, there should be an error if multiple *omniauth_providers* with type *openid_connect* are present within a configuration file. Alternatively, the possibility to register multiple OpenID Connect Identity Providers could be implemented.

5.4.4 Analysis of Salesforce Lightning

Salesforce Lightning allows to configure *Auth. Providers* for organizations. In doing so, it does not support OpenID Connect Discovery but allows administrative users to manually configure OpenID Connect endpoints.

Section	Vulnerability Type	Attacker Model	Severity
5.4.4.1	Login Confusion	Web Attacker	Medium
5.4.4.2	Request Amplification	Web Attacker or Malicious Identity Provider	Low

Table 5.18: Overview of the most relevant observations and findings in Salesforces's Service Provider implementation.

5.4.4.1 [Login Confusion] "startUrl" Parameter Allows "Login Confusion" Due to Intransparent Redirects

Attacker Model. For the following attack scenarios, the web attacker model (section 3.1) is applied.

Description. Salesforce supports a non-normative GET parameter that specifies where the user should be redirected after successful authentication. There is basic validation applied to the `startUrl` parameter, but the following issues were identified within this validation:

- Endpoints that perform session-relevant actions are allowed as redirection target. The user does not receive any indication prior the redirect to these endpoints after login, resulting in unforeseen behavior.
  - *Logout endpoint*: If the `startUrl` parameter is set to `%2Fsecur%2Flogout.jsp`, the user is immediately logged out after succesful authentication.
  - *SSO Account linking*: If the `startUrl` parameter is set to an *Account Linking Endpoint* for a registered Authentication Provider, e.g. https://-login.salesforce.com/?startURL=/services/auth/oauth/identifier/name, the account linking process is triggered immediately after authentication without additional user interaction. As a result, there is an immediate prompt to link the SSO account to the user's account that logged in before, if the user has an active session at the corresponding Identity Provider. Alternatively a malicious Identity Provider could always respond with an `Authentication Response` and try to link an attacker-controlled dummy account into a victim's account. In order to complete the linking, the End-User has to approve the dialogue and give consent.

  - *Login Confusion*: The same Login Confusion patter that was previously outlined for Bitbucket (subsubsection 5.4.2.3) can be applied to Salesforce, if the `startUrl` parameter is set to an "OAuth-Only Initialization URL" for a registered Authentication Provider, after Login with credentials, a second login flow using OpenID Connect can be performed.

- The validation routine for the redirection target allows any subdomain of `*.salesforce.com`. This could potentially be turned into a covert redirect in case an attacker manages to find a forgotten and dangling subdomain. If a DNS entry for a subdomain of `*.salesforce.com` points to a cloud service, e.g. for a temporary campaign, but is not removed after it is not needed anymore, it may be possible for some cloud services to re-claim and therefore hijack this reference to the cloud service. Frans Rosén gave a talk on different variants of "subdomain takeovers" at OWASP AppSec EU 2017 [16].
  During the analysis, `saturn.salesforce.com` was found that apparently fulfills the requirements having a dangling `CNAME` record, but as of 2020, Heroku prevents arbitrary registration of dangling domains using random subdomains of `*.herokudns.com` [37]:

```
1 user@laptop:~$ dig saturn.salesforce.com
2 [...]
3 ;; ANSWER SECTION:
4 saturn.salesforce.com.  268  IN  CNAME  closed-othnielia-
  ↪  faezuoinck1tonw5fxq3r0t3.herokudns.com.
```

Listing 5.30: Salesforce: `CNAME` entry to Heroku that is not used anymore.

- Special characters like `%0A` (= linefeed) lead to mutations of the redirect target. Example: `https://login.salesforce.com/?startURL=javascript://%0aprompt(1)` results in a redirect to:

```
1 https://eu13.salesforce.com/https://login.salesforce.com/https:/⌋
  ↪  /login.salesforce.com/
  ↪  https://login.salesforce.com/https://login.salesforce.com/ja⌋
  ↪  vascript://prompt(1)
```

Listing 5.31: Salesforce: Redirection target after mutation by Salesforce.

Even though no harmful mutation was found, this behavior indicates that the validation routine performs unnecessary and potentially dangerous mutations.

Recommendation. To mitigate these issues, multiple actions should be taken:

- Session relevant endpoints should be explicitly excluded from the `nextUrl` parameter, as redirects to these endpoints have unforeseen effect on the current user session and the application's behavior.
- The dangling DNS entry for `saturn.salesforce.com` should be removed if it is not needed anymore. Additionally, a whitelist for trusted subdomains of `*.salesforce.com` should be introduced. Other subdomains should be handled like `force.com` (End-User has to confirm redirect to page, third party domains like `lauritz-holtmann.de` already result in direct error message instead).
- The redirect target needs to be sanitized regarding dangerous characters, as the layers of the web application stack may treat illegal characters and sequences like *CRLF* differently.

5.4.4.2 [Request Amplification] Reusable "state" Parameter At "redirect_uri" Endpoint Enables Multiple Attack Vectors

Attacker Model. For the following attack, two attacker models could be considered. A web attacker (section 3.1) having no influence on the target of the `Token Request` could only target the already configured Identity Provider. A malicious administrative user (section 3.4) could additionally target arbitrary web servers being accessible for the Service Provider.

Description. Salesforce as Service Provider provides dedicated `redirect_uri`s for each Identity Provider. The *Redirection Endpoint* does not invalidate `state` values if they are redeemed once. As a result, multiple request to this endpoint including the valid `state` value result in requests to the configured *Token Endpoint* being initiated by Salesforce. As the "Token Endpoint URL" may be chosen arbitrarily, this behavior could be described as intentional SSRF and administrative users that may configure "Auth. Providers" may decide where Salesforce will send the `Token Request` to. A malicious administrative user (section 3.4), could additionally increase the outgoing `Token Request` by choosing a long path for the *Token Endpoint* URL and setting imaginary long client credentials. As a result, one request to Salesforce could be amplified multiple compared to the request that is received at the victim server's end.

Steps to reproduce. The steps to launch a Denial-of-Service attack using many large requests through Salesforce as amplifying proxy are the following:

1. Configure the victim server as *Token Endpoint* with long path and long client credentials (if the attacker model is a regular web attacker (section 3.1), this step is skipped).

2. Start OpenID Connect flow and obtain a valid `state` value (i.e. save valid `Authentication Response` including state)
3. After the `Authentication Response` is received, Salesforce tries to redeem received `code` value at victim web server. This request can be multiple times larger than attacker's request.
4. Replay obtained `Authentication Response`. On each try Salesforce sends a new HTTP request to the victim server.

Recommendation. To mitigate this issue, the `state` should be a one-time-usable value.

5.4.5 Analysis of Amazon Cognito (AWS)

Amazon Cognito is an Identity and Access Management (IAM) service. Cognito implements Identity Provider and Service Provider parts of the OpenID Connect specification. As part of the AWS family, Amazon Cognito is a hosted service within the Amazon Cloud.

Section	Vulnerability Type	Attacker Model	Severity
5.4.5.1	SSRF	Malicious Identity Provider or Malicious Administrative User	High
5.4.5.2	Mix-Up Attack	Malicious Identity Provider	Medium
5.4.5.3	Request Amplification	Web Attacker or Malicious Identity Provider	Low
5.4.5.4	Unencrypted Communication	Man-in-the-Middle	Low
5.4.5.5	Token Handling	N/A	None

Table 5.19: Overview of the most relevant observations and findings in Amazon Cognito's Service Provider implementation.

5.4.5.1 [SSRF] SSRF to "localhost" Allows Portscan And Sending HTTP Requests To Internal Hosts

Attacker Model. The attacker model that is applied in the following is a malicious administrative user (section 3.4). Amazon Cognito is a hosted service, administrative users only have access to the configuration interface, the underlying infrastructure in Amazon's hosted environment should not be accessible to external users. Alternatively, a malicious Identity Provider (section 3.2) could be considered as attacker model, as the actual configuration is fetched using OpenID Connect Discovery from the *Registration Endpoint* so that the endpoints are Identity Provider-controlled.

Description. Amazon Cognito allows specifying custom OpenID Connect Identity Providers for user pools. An administrative user can choose to "Run discovery", triggering an OpenID Connect Discovery that requests the *Configuration Endpoint* of the Identity Provider.

Amazon does not restrict the endpoints observed using OpenID Connect Discovery regarding internal IP addresses or localhost, so that SSRF to the local network and localhost is possible. A malicious actor can utilize this to perform port scans or send nearly arbitrary HTTP requests to hosts intentionally not exposed to the internet.

An example `Configuration Response` is presented in Listing 5.32.

```
{
    "issuer":"https://example.com/",
    "authorization_endpoint":"https://example.com/auth",
    "token_endpoint":"http://127.0.0.1:22",
    "userinfo_endpoint":"http://127.0.0.1:22",
    "jwks_uri":"https://example.com/jwks",
    "registration_endpoint":"https://example.com/register",
    "response_types_supported":["code","token id_token"],
    "subject_types_supported":["public","pairwise"],
    "id_token_signing_alg_values_supported":["RS256"]
}
```

Listing 5.32: Amazon Cognito: Configuration Response including localhost as *Token Endpoint* and *UserInfo Endpoint*.

Portscan. The following error messages can be used to determine which ports are internally open on localhost, as the error message sent to the `redirect_uri` (`https://a.com/cb?error_description=[ERROR]&error=invalid_request`) differs, depending on the underlying Transmission Control Protocol (TCP) connection (if it can not be established) or the HTTP error code Amazon receives in response to the `Token Request`:

- If the *Token Endpoint* is specified as `http://localhost:22`, the error message is "Connection reset", so that we assume Port 22 to be open.
- If the *Token Endpoint* is specified as `http://localhost:20`, the error message is "Connect to 127.0.0.1:20 [/127.0.0.1] failed: Connection refused", so that we assume Port 20 to be closed.
- If the *Token Endpoint* is specified as `http://test123.ngrok.io`, the error message is "test Error – 502 error getting token", so that we assume that there was an HTTP request to the specified target but the webserver responded with HTTP Error Code 502.

Arbitrary HTTP requests. Beside the feedback an attacker receives that enables him to determine the status of the connection, there is no information on the actual response to the HTTP request. Thus, this gadget can be considered as blind SSRF to the local network, as having no direct feedback a malicious entity could still control a GET request (`UserInfo Request`) and a POST request (`Token Request`) regarding scheme (*http* or *http*), host (potentially localhost or internal IP), path and query parameters.

Combining port scan and blind SSRF, a malicious administrative user (section 3.4) or malicious Identity Provider (section 3.2) could:

1. Gather information: Which ports are open on localhost? Are there other hosts accessible? If HTTP error codes are reflected, which web service is running at this destination (fingerprinting)?

2. If a service could be identified: Use blind SSRF to directly target internally accessible service.

Recommendation. A blacklist is never perfect. Nevertheless Amazon Cognito should introduce a restriction for internal IPs and localhost as OpenID Connect endpoints, as there is no legitimate use-case in the context of a hosted service.

5.4.5.2 [Spec] Mix-Up Attack

Attacker Model. The attacker model that is applied in the following is a malicious Identity Provider (section 3.2). Prerequisite is that Amazon Cognito has multiple Identity Providers configured, at least one Honest Identity Provider (H-IdP) and one Malicious Identity Provider (M-IdP). Additionally, it is assumed that the attacker can modify the first request-response pair to Amazon Cognito, as it is for instance described in the Attacker Model A2 of the *OAuth 2.0 Security Best Current Practices* [48, Section 3.]. Alternatively and without the possibility to modify the first request-response pair, a Malicious Identity Provider could immediately redirect a victim user to the Honest Identity Provider's *Authentication Endpoint*. In this case, the user could notice the redirect, but if she still authenticates at the Honest Identity Provider, the tokens are disclosed to the Malicious Identity Provider [48, Section 3.].

Description. Amazon Cognito is vulnerable to Mix-Up attacks that were introduced by Fett at al. in 2016 [14]. Amazon Cognito supports multiple Identity Providers that use a single Redirection URI: `https://subdomain.auth.us-east-2.amazoncognito.com/oauth2/idpresponse`. In doing so, the user's choice which Identity Provider should be used is stored within the `state`. An attacker can confuse Amazon Cognito to disclose the OpenID Connect `code` from the Honest Identity Provider to the Malicious Identity Provider. As a result, the Malicious Identity Provider "*can either exchange the code for an access token (for public clients) or perform an authorization code injection attack*" [48, Section 4.4.1.].

Steps to reproduce. To reproduce the behavior, the following steps need to be performed:

1. The victim End-User chooses the Honest Identity Provider at the "Hosted UI", resulting in a request to Amazon Cognito including the `identity_provider` (= "name") and `client_id` parameter of the Honest Identity Provider.

2. The Attacker changes the selection to Malicious Identity Provider and forwards it to Amazon Cognito (adjusts `identity_provider` and `client_id`).

3. Amazon Cognito responds with a redirect to the Malicious Identity Provider including a `state` bound to Malicious Identity Provider. An excerpt of the state, base64 decoded, is given in the following:

```
{
    "userPoolId":"us-east-2_AAAAAAAA",
    "providerName":"Malicious IdP",
    "clientId":"AAAAAAAAAAAAAAAAAAA",
    "redirectURI":"https://sp.com/cb",
    "responseType":"code",
    "providerType":"OIDC",
    "scopes":["openid"],
    [...]
}
```

Listing 5.33: Amazon Cognito: Payload of the OpenID Connect `state` decoded (excerpt).

4. The attacker modifies the redirect, the new redirection target is the *Authentication Endpoint* of the Honest Identity Provider including the valid `state` (bound to the Malicious Identity Provider).

5. After authenticating the End-User, the Honest Identity Provider redirects the End-User to the shared *Redirection Endpoint* including a `code` and the `state` that is bound to the Malicious Identity Provider.

6. Amazon Cognito redeems the `code` at the Malicious Identity Provider's *Token Endpoint*, disclosing the valid `code` that was issued by the Honest Identity Provider.

Alternatively, the following steps to reproduce can be considered, if the attacker can not modify the first request-response pair:

1. The victim End-User chooses the Malicious Identity Provider at the "Hosted UI", resulting in a request to Amazon Cognito including the `identity_provider` and `client_id` parameters of the Malicious Identity Provider.

2. Amazon Cognito responds with a redirect to the Malicious Identity Provider including a `state` bound to the Malicious Identity Provider.

3. The Malicious Identity Provider immediately redirects the victim to the *Authentication Endpoint* of the Honest Identity Provider, including the valid `state`, `redirect_uri` and `client_id`. The following steps are performed as seen before.

Recommendation. It is recommended to use a dedicated *Redirection Endpoint* per Identity Provider. If this is not possible, the recommendations according to the OAuth 2.0 Security Best Current Practices are, to either use a non-normative "iss" parameter within the `Authentication Response` enabling the Service Provider to compare this value with the Identity Provider it will send the `code` to or alternatively include an `id_token` within the front-channel `Authentication Response` and utilize the "iss" claim for this purpose. Note that the OpenID Connect specific mitigation utilizes the Hybrid Flow with only an `id_token` and the `code` within the front-channel [48, Section 4.4.].

5.4.5.3 [Request Amplification] Reusable "state" Parameter At "redirect_uri" Endpoint Enables Multiple Attack Vectors

Attacker Model. For the following attack, multiple attacker models could be considered. A web attacker (section 3.1) having no influence to the target of the Token request could only target the already configured Identity Provider. A malicious administrative user (section 3.4) could additionally target arbitrary web servers being accessible for the Service Provider using manual configuration. If OpenID Connect Discovery is used, a malicious Identity Provider (section 3.2) could also specify an arbitrary web server as *Token Endpoint* and issue large client credentials.

Description Amazon Cognito as Service Provider allows specifying custom OpenID Connect Identity Providers for user pools. The `redirect_uri` endpoint does not invalidate `state` values if they are redeemed once. As a result, multiple requests to this endpoint including the valid `state` value result in requests to the configured *Token Endpoint* being initiated by Amazon.
As the "Token Endpoint URL" may be chosen arbitrarily, this behavior could be described as intentional SSRF and administrative users that are allowed to configure "Auth. Providers" may decide where Amazon will send the `Token Request` to. The same applies to the configuration using OpenID Connect Discovery, so that a malicious Identity Provider also could specify where Amazon sends its `Token Request` to. A malicious administrative user could additionally increase the outgoing `Token Request` by choosing a long path for the *Token Endpoint* URL and setting imaginary long client credentials. The same applies to a malicious Identity Provider that could simply issue large credentials to Amazon Cognito. As a result, one request to Amazon could be amplified multiple times compared to the request that is received at the victim server's end.

Steps to reproduce The steps to reproduce are comparable to the occurrence of this behavior in Keycloak (5.4.1.1) or Salesforce (5.4.4.2).

Recommendation. To mitigate this issue, the `state` should be a one-time-usable value.

5.4.5.4 [Unencrypted Communication] OpenID Connect Discovery Allows Specifying OIDC Endpoints Using HTTP Without TLS

Attacker Model. This attack utilizes the man-in-the-middle attacker model (section 3.5). In the following, it is described that using OpenID Connect Discovery the missing validation of endpoints can lead to insecure configurations. Amazon Cognito supports TLS but does not enforce it for critical OpenID Connect endpoints if they are specified during Discovery.

Description. Amazon Cognito allows specifying OpenID Connect Identity Provider for user pools. When configuring the Identity Provider's endpoints manually, Cognito enforces that all Endpoints must use HTTP with TLS. But in contrast, if Amazon receives an OpenID Connect `Configuration Response` including Endpoints using HTTP without TLS, it accepts and uses these endpoints. An example `Configuration Response` that is accepted (note that the `Token Endpoint` uses HTTP only) is given in Listing 5.34.

```
{
    "issuer":"https://a.ngrok.io/",
    "authorization_endpoint":"https://a.ngrok.io/auth",
    "token_endpoint":"http://b.ngrok.io",
    "userinfo_endpoint":"https://a.ngrok.io/userinfo",
    "jwks_uri":"https://a.ngrok.io/jwks",
    "registration_endpoint":"https://a.ngrok.io/register",
    "response_types_supported":["code","token id_token"],
    "subject_types_supported":["public","pairwise"],
    "id_token_signing_alg_values_supported":["RS256"]
}
```

Listing 5.34: Amazon Cognito: Insecure Configuration due to missing validation of `Configuration Response`.

Recommendation. To mitigate this issue, consistently allow only OpenID Connect Endpoints using HTTP with TLS .

5.4.5.5 [Spec] OpenID Connect Token Handling

Description. Amazon Cognito allows to configure "App clients" that use OpenID Connect to authenticate against an Amazon Cognito "User Pool". In doing so, minor specification related issues were identified.

Recommendation. Amazon Cognito's token handling should be hardened regarding the following aspects:

1. There is no possibility for end users to revoke `access_token`s at the Identity Provider. According to the documentation, the Service Provider could implement an API call ("*An app can use the GlobalSignOut API to allow individual users to sign themselves out from all devices.*" [2]), but the End-Users should be able to manage the permissions themselves at the Identity Provider (via Hosted UI).

2. Redeemed `refresh_token`s are not rotated. According to the OAuth 2.0 Security Best Current Practices, "*Authorization server MUST utilize*" either "*sender-constrained refresh tokens*" or "*refresh token rotation*" [48, Section 4.12.2.].

3. If a `code` is redeemed it is invalidated correctly. Further tries to redeem the `code` yield errors, but do not cause previously issued tokens for this `code` to be invalidated. RFC6749 and the OAuth 2.0 Security Best Current Practices recommend that, "*the AS SHOULD revoke all tokens issued previously based on that code*" [48, Section 4.2.4.].

6 Lessons Learned

The following chapter outlines the lessons learned within this master's thesis. At first, a high level summary of the expectations and evaluation results is given. Afterward, common findings among the test-set are derived and based on these patterns additions to previously known security considerations are proposed. Finally, the Responsible Disclosure processes with the maintainers and vendors are broadly outlined.

6.1 Expectations and Results

In chapter 5, the results and observations based on the initially defined tests and expectations for secure OpenID Connect implementations (section 2.3) were discussed. It has been observed, that most of the previously known attacks were not possible among the test-set. Especially, the basic assertions made within the specification [33, Section 16.] and the security best practices [48] were fulfilled.
Nevertheless, the evaluation also shows that the analyzed products and services have security issues regarding their OpenID Connect implementations. The severity of the individual vulnerabilities that were discovered during execution of this thesis varies from *None* (security relevant bug that still could not be exploited due to outer limitations) to *Critical* (e.g. SSRF that can be escalated to Remote Code Execution). Even previously discussed issues like the unauthenticated SSRF by design that is present if the `request_uri` is implemented [15, Section III; A. 8)], a vulnerability with *High* severity, were present within the test-set.
Some observations and issues were present in multiple implementations among the test-set. In the following, these issues and pitfalls are derived to patterns within the overall analysis results.

6.2 Derived Common Issue Patterns

The evaluation leads to the conclusion that there are some more common issues that could be found in multiple implementations independently.

1. It has been observed for multiple Identity Providers that the `redirect_uri` allows any possible scheme. This is potentially dangerous, as browsers treat

dangerous schemes like `data` or `javascript` very differently. Additionally, the vague specification of the `redirect_uri` was made having native clients in mind, so that there are very few legitimate use-cases for these schemes as `redirect_uri` values in OpenID Connect setups.

2. Multiple Identity and Service Provider implementations were vulnerable to different types of Server-Side Request Forgery vulnerabilities.
   The Identity Provider's `request_uri` parameter is already known to be vulnerable for Server-Side Request Forgery by design since 2017 [15, Section III; A. 8)]. Notably, this could be exploited by any unauthenticated user.
   In 2017, Mladenov and Mainka pointed out [26, Section 2.1.2] that a malicious Discovery Service could be used to launch a SSRF attack and in doing so perform port scans or retrieve data. Additionally, having access to the endpoint configuration, administrative users could launch SSRF attacks on the Service Provider by design. Depending on the actual setup, this yields serious security implications, especially considering hosted services like Amazon Cognito. Nevertheless, SSRF to internal IPs and localhost was a commonly observed issue.

3. The OpenID Connect Core specification already enforces TLS for communication with *Authorization*, *Token* and *UserInfo Endpoint*, so does the OpenID Connect Dynamic Client Registration regarding the *Client Configuration Endpoint*. Nevertheless, the majority of the analyzed implementations allow performing the discovery using unencrypted communication or allow to specify OpenID Connect endpoints using HTTP without TLS.

4. Error messages and pages need extra attention. During the analysis, multiple Service Provider implementations presented error pages as a direct response to erroneous `Authentication Responses`. As a result, the GET parameters of this page include OpenID Connect values like `state` and `code`. If there are external resources like images, scripts, styles or links to external resources on any endpoint that receives sensitive information via query parameters, the `Referer` header could leak sensitive OpenID Connect parameters.

5. The Service Provider's *Initiate Login Endpoint* implements Login Cross-Site-Request-Forgery by design and per specification [33, Section 4.]. This behavior is well known and was previously discussed [15, Section III; A. 7)]. Besides this endpoint, non-normative endpoints to start an external OpenID Connect login without Cross-Site-Request-Forgery protection were observed. If the `state` is correctly validated, these endpoints still allow to log in a victim End-User into her own account.
   If there are additional non-normative parameters that indicate where the user should be redirected to after successful authentication including *Login Initiation Endpoints*, Login Confusion attacks are possible, as described in subsubsection 2.3.4.11.

6. OpenID Connect `state` parameter handling at the Service Provider's *Redirection Endpoint* showed multiple more common issues. Some of the analyzed implementations did not handle the unexpected behavior of not receiving a `state` if it was present within the `Authentication Request`. As a result, unhandled exceptions occur.
   Further, multiple implementations correctly bound the `state` to the user session but failed to make it one-time-usable, increasing the attack surface for token-reuse and Denial-of-Service Amplification attacks.

7. Service Providers often treat Identity Provider provided contents as trustworthy. It has been shown in this master's thesis that this behavior in general yields injection issues (as previously outlined as "Injection Attacks" by Mainka et al. in [26, Section 2.1.3]). Striking among the test-set were *CRLF*-related issues, as missing sanitization leads to HTTP header injections in Service Provider initiated requests and injections to application log files.

6.3 Derived OpenID Connect Security Considerations

In addition to previously known security considerations [26][33, Section 16.][48], it has been shown that the following aspects need extra attention. Some of these considerations have been made earlier but with a less severe indication. In terms of RFCs and specifications this would mean that a "MAY" or "SHOULD" would become a necessary requirement being reflected as "MUST".

1. The specification needs to be tightened regarding the `redirect_uri` definition. The scheme of this URI MUST NOT be `data`, `javascript`, `vbscript` or any other scheme that is not HTTP(S) or related to a dedicated native client.

2. The Identity Provider SHOULD restrict allowed `request_uri` values by allowing the Service Provider to specify the `request_uris` parameter that is an "*array of request_uri values*", at registration [35, Section 2.].
   Currently, only if the OpenID Connect Dynamic Client Registration is used, the Identity Provider can require the Service Provider to "*pre-register request_uri values using the request_uris parameter*" [33, Section 6.2.]. This should be moved from the Registration Extension to the Core Specification.

3. Identity and Service Providers SHOULD globally restrict outgoing HTTP requests to `localhost` and private IPs [52, Section 3.] in the context of OpenID Connect. There MAY be a configurable option to allow these connections. Further, error messages that are returned in case an OpenID Connect request fails MUST NOT include connection-specific details that could give an attacker information on the underlying transport layer (for instance error messages that indicate if a TCP connection was *refused* or *reset*).

4. As the Specification already outlines [34, Section 7.2.], OpenID Connect Service and Identity Providers MUST support and MUST use TLS for communication with the *Configuration*, *Token* and *UserInfo Endpoint*.
5. Error messages SHOULD not be shown as a direct response to erroneous OpenID Connect requests, if the error page includes external resources, without omitting the sensitive GET parameters from `Referer` headers. This could either be done using a prior redirect to a generic path (e.g. `https://example.com/error`) or by defining a restrictive `Referrer-Policy` (e.g. "origin-when-cross-origin" or "strict-origin-when-cross-origin").
6. Service Providers MUST use a `state` value to prevent Cross-Site-Request-Forgery login into arbitrary accounts, if no other CSRF protection is implemented. If the `state` was present within the `Authentication Request`, it MUST be included within the Authentication Response. If this is not the case, the Service Provider needs to fail safely (i.e. not ignore that the `state` is missing and proceed anyway, but handle the occurring exception adequately). In addition, the `state` value MUST be invalidated after it was redeemed once to prevent amplification attacks.
7. As previously advised by Fett et al. [15, Section III; A. 7)], the optional *Initiate Login Endpoint* SHOULD NOT be implemented if it is not explicitly needed. The same applies to non-normative external login endpoints. If these endpoints are implemented, token- or `Referer` header-based Cross-Site-Request-Forgery mitigations MUST be in place. In case there is a non-normative GET parameter that controls the redirection target after successful authentication, session relevant endpoints like *External Login Endpoints* or *Logout Endpoints* SHOULD NOT be allowed as redirection target.
8. Identity Provider provided contents MUST be handled as third party input. Therefore, strict and context-aware sanitization MUST be performed before using these values.

6.4 Responsible Disclosure

All vendors were informed about the observations being presented within this master's thesis. In the following, a short overview of the communication and disclosure processes is given.

During this master's thesis, different depths and extents of Proof-of-Concepts that were attached with the initial report were iterated. In the beginning, we attached a tutorial on how a developer could set-up a Keycloak instance with proxies and TLS so that an issue regarding a Service Provider implementation could be validated. This process turned out to be time-consuming and error-prone, as the tutorial needed to be adjusted for each finding. Thus, for later Responsible Disclosure processes the

custom NodeJS implementations which were introduced within this thesis were preconfigured and sent to the developers alongside the advisories.

6.4.1 Keycloak

Keycloak is an open-source software that is maintained by *Red Hat*. There is a Jira issue tracker for Keycloak issues that allows to flag reported issues as "Security Sensitive Issue": `https://issues.redhat.com/browse/KEYCLOAK`. Using this channel, 13 tickets for individual findings were opened and discussed with the maintainers and developers of Keycloak. The issues were triaged and categorized by Red Hat employees either as security hardening (violation of specification or general best practices without direct exploitability) or security vulnerability that received a CVE.
All reported and fixed issues that are public can be observed using the following link: `https://issues.redhat.com/browse/KEYCLOAK-15246?filter=-2&jql=reporter%20in%20(lauritz)%20order%20by%20created%20DESC`.

6.4.2 Bitbucket

Atlassian as vendor of Bitbucket offers multiple communication channels. We decided to use the Bugcrowd program (`https://bugcrowd.com/atlassian`) for severe security-relevant findings. After the triage that was performed by Bugcrowd, direct contact was held with Atlassian's security staff. We filed six reports, among these reports four reports were accepted and two reports were marked as "not applicable" in regard to their policy. In addition, two non-security-relevant bugs were reported via `https://getsupport.atlassian.com/servicedesk`.

6.4.3 GitLab

For GitLab, there are multiple channels to communicate bugs and security issues. For one less relevant finding, a GitLab issue was created on `https://gitlab.com/gitlab-org/gitlab/-/issues/`.
For the more severe issues `https://hackerone.com/gitlab` was used as a direct communication channel with GitLab's Security Team. After submission of four issues found within the Service Provider implementation, the security team notified us that they do not consider rogue Identity Providers as part of their threat model and closed all issues accordingly. Highlighting the impact and hints for escalation to Remote Code Execution did not lead to reconsideration.
One additional report on an Identity Provider flaw was immediately closed with an unclear explanation. After weeks of discussion, it was reopened and triaged as duplicate, because parts of the issue were covered in a two-year-old unfixed report. As we elaborated further on the impact, there was immediate progress in this case.

As a result the issue has been fixed within weeks in v13.2.3 and was acknowledged as CVE-2020-13294.

6.4.4 Salesforce

Even though Salesforce has a Responsible Disclosure Policy [40], communication with the vendor turned out to be quite challenging. We reported our observations at the End of July. As we did not receive a reply, additional contact attempts were made using different mail addresses, previously known contact persons and the German sales support. The only working communication channel that could be established was through the German sales staff. After repeatedly asking for status updates and offering assistance during triage (e.g. answering further questions or providing test environments for Proof-of-Concepts), the very first direct mail from Salesforce's security team was received 36 days after our initial mail. The security team claimed that our reports were received and were considered as advisories. Additionally, the Salesforce PSIRT informed us that they consider the cases to be closed and will not perform further investigations so that we likewise did not perform further actions.

6.4.5 Amazon Cognito

As Amazon Cognito is part of AWS, we addressed our reports to aws-security@amazon.com. The Security Staff replied and asked for a call, so that we were able to present an overview of our findings to the AWS Security Staff and Amazon Cognito Developers on August 19th, 2020. We presented a demo on some observations and gave further hints regarding the impact and mitigation of the reported issues.

7 Conclusion

In this master's thesis, four real-life OpenID Connect Identity Provider and five real-life Service Provider implementations were analyzed regarding their security.
The evaluation regarding the initially developed test and evaluation catalog as well as the observations that were made parenthetically resulted in new variants of previously known attack classes and particular attacks on OpenID Connect implementations. Furthermore, two specification-wise inaccuracies were discovered that lead to real-life security implications.
These evaluation results and derived patterns should not fall into oblivion with completion of this thesis, so that concrete patterns and hints for security best practices were derived. These aspects will be additionally covered in blog posts to increase awareness.
During the execution phase, custom NodeJS-based OpenID Connect Service Provider and Identity Provider implementations were created that assist researchers during testing and reporting, as they allow to provide lightweight retest environments for complex OpenID Connect scenarios. These implementations are open source and available on Github[1][2].
Finally, we performed more than 35 individual Responsible Disclosure or Bug Report processes and assisted vendors and maintainers during triage and fix. As far as possible, we will encourage vendors to make our reports public after the issues are fixed, so that others can learn from these reports. In the beginning, filing a report for a complex OpenID Connect Setup including multiple parties was challenging. Hopefully, our reports give insights that could be adapted for comparable Responsible Disclosure attempts on complex Single Sign-On scenarios.

7.1 Future Work

During this thesis we mainly focused on Service Provider and Identity Provider implementations separately. We encountered multiple products, for instance Keycloak and Amazon Cognito, which implement both parts of the specification, but for a higher mean: The services can be used for identity brokerage and federation. As in this case, one entity unites different roles within the protocol, there

[1] Custom OpenID Connect Service Provider: `https://github.com/lauritzh/oidc-custom-sp`
[2] Custom OpenID Connect Identity Provider: `https://github.com/lauritzh/oidc-custom-idp`

should be further research regarding Identity and Access Management (IAM) services combining Service Provider and Identity Provider parts of OpenID Connect 1.0 specification.

One of the more common issue patterns were *CRLF* injections in Identity Provider given values. Identity Providers are third parties, nevertheless, some applications treat every Identity Provider provided value as trusted. As Mainka et al. previously showed [26], there is a risk of injection flaws regarding Identity Provider controlled contents. As among our test-set, the *CRLF* injection was such a common issue, the evaluation of this specific pattern (second order vulnerabilities that exploit edge-cases and different character sets that are allowed in different contexts) among a larger set of Service Providers could yield interesting results. Beside *CRLF* injections, there is a broad field of more sophisticated injections like *template injections* that should be also applied to a broader test-set of OpenID Connect implementations.

The specification allows `redirect_uris` with potentially dangerous schemes. To determine the impact if an Identity Provider supports `data`, `javascript` and other reserved schemes, there should be a thorough evaluation of the HTTP `Location` header handling among popular User Agents. In doing so, beside JavaScript execution especially `data` URIs using different MIME types like `application/postscript`, `application/pdf` and other `application/*` types should be evaluated, as some User Agents render contents with these MIME types if they are observed as `data` URI.

Finally, Server-Side Request Forgery played a relevant role during our research. Depending on the threat model and if the software is self-hosted or a cloud service, turning genuine OpenID Connect requests like the *Token Request* in an attack vehicle to target localhost is just one step ahead. Keycloak aims to fix this by adding a "Trusted Hosts" policy that is intended to define a global policy on where the instance may communicate to. This mechanism could possibly become a standardized part of the specification and a thorough analysis regarding up- and downsides of the method should be applied.

List of Figures

List of Tables

Bibliography

[1] Amazon: *AWS customer success*, 2020. https://aws.amazon.com/solutions/case-studies, as of September 30, 2020.

[2] Amazon: *Using tokens with user pools*, 2020. https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-using-tokens-with-identity-providers.html, as of September 30, 2020.

[3] Atlassian: *Bitbucket server 7.0 release notes*, 2020. https://confluence.atlassian.com/bitbucketserver/bitbucket-server-7-0-release-notes-990546638.html, as of September 30, 2020.

[4] A. Barth, C. Jackson, and John C. Mitchell: *Securing Frame Communication in Browsers*, 2008. https://www.adambarth.com/papers/2008/barth-jackson-mitchell.pdf, as of September 30, 2020.

[5] BuiltWith Pty Ltd: *Websites using Facebook Login Button including Historical*. https://trends.builtwith.com/websitelist/Facebook-Login-Button/Historical, as of September 30, 2020.

[6] BuiltWith Pty Ltd: *Find out what websites are Built With*. https://builtwith.com/, Online as of September 30, 2020, 2020. "Build lists of websites from our database of 38,047+ web technologies and over a quarter of a billion websites showing which sites use shopping carts, analytics, hosting and many more. Filter by location, traffic, vertical and more.".

[7] Christoph Kerschbaumer: *Blocking top-level navigations to data urls for firefox 59*, 2014. https://blog.mozilla.org/security/2017/11/27/blocking-top-level-navigations-data-urls-firefox-59/, as of September 30, 2020.

[8] Cure53, Dr.-Ing. M. Heiderich, M. Rupp, Dr. N. Kobeissi, BSc. C. Kean, MSc. S. Moritz, B. Walny, BSc. T.-C. Hong: *Pentest-Report Keycloak 8.0 Audit & Pentest 11.2019*. https://cure53.de/pentest-report_keycloak.pdf, as of September 30, 2020.

[9] D. Crockford: *RFC4627: The application/json media type for javascript object notation (JSON)*, 2006. https://tools.ietf.org/html/rfc4627, as of September 30, 2020.

[10] Dr. Torsten Lodderstedt: *OpenID Connect @ Deutsche Telekom.* https://www.gsma.com/identity/wp-content/uploads/2014/03/OpenID-Connect-at-Deutsche-Telekom-Torsten-Lodderstedt.pdf, 2014. Online, as of September 30, 2020.

[11] Ed. D. Hardt: *RFC6749: The OAuth 2.0 authorization framework*, 2012. https://tools.ietf.org/html/rfc6749, as of September 30, 2020.

[12] Ed. T. Bray: *RFC8259: The JavaScript Object Notation (JSON) data interchange format*, 2017. https://tools.ietf.org/html/rfc8259, as of September 30, 2020.

[13] D. Fett, P. Hosseyni, and R. Kästers: *An extensive formal security analysis of the OpenID financial-grade API.* In *2019 2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1054–1072, Los Alamitos, CA, USA, may 2019. IEEE Computer Society. https://doi.ieeecomputersociety.org/10.1109/SP.2019.00067.

[14] D. Fett, R. Küsters, and G. Schmitz: *A comprehensive formal security analysis of OAuth 2.0.* In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, p. 1204–1215, New York, NY, USA, 2016. Association for Computing Machinery, ISBN 9781450341394. https://doi.org/10.1145/2976749.2978385.

[15] D. Fett, R. Küsters, and G. Schmitz: *The web sso standard OpenID Connect: In-depth formal security analysis and security guidelines.* In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pp. 189–202, Aug 2017.

[16] Frans Rosén: *DNS hijacking using cloud providers – no verification needed*, 2017. https://de.slideshare.net/fransrosen/dns-hijacking-using-cloud-providers-no-verification-needed-76812183, as of September 30, 2020.

[17] JBoss Developers: *Keycloak: Open source identity and access management.* https://www.keycloak.org/, as of September 30, 2020.

[18] Job van der Voort: *GitLab 7.7 and GitLab ci 5.4 with GitHub importer and OAuth authorization*, 2015. https://about.gitlab.com/releases/2015/01/22/gitlab-7-7-and-ci-5-4-released/, as of September 30, 2020.

[19] Jobert Abma: *Evaluating ruby code by injecting rescue job on the system_hook_push queue through web hook*, 2017. https://hackerone.com/reports/299473, as of September 30, 2020.

[20] W. Li and C.J. Mitchell: *Analysing the security of google's implementation of openid connect.* In J. Caballero, U. Zurutuza, and R.J. Rodríguez (eds.): *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 357–376, Cham, 2016. Springer International Publishing, ISBN 978-3-319-40667-1.

[21] W. Li, C.J. Mitchell, and T. Chen: *Mitigating CSRF attacks on OAuth 2.0 and OpenID Connect*. CoRR, abs/1801.07983, 2018. http://arxiv.org/abs/1801.07983.

[22] M. Blanchet: *RFC5156: Special-use IPv6 addresses*, 2008. https://tools.ietf.org/html/rfc5156, as of September 30, 2020.

[23] M. Jones and D. Hardt: *RFC6750: The OAuth 2.0 authorization framework: Bearer token usage*, 2012. https://tools.ietf.org/html/rfc6750, as of September 30, 2020.

[24] M. Jones, J. Bradley, and N. Sakimura: *RFC7519: JSON Web Token (JWT)*, 2015. https://tools.ietf.org/html/rfc7519, as of September 30, 2020.

[25] C. Mainka: *On message-level security*. doctoralthesis, Ruhr-Universität Bochum, Universitätsbibliothek, 2017.

[26] C. Mainka and V. Mladenov: *OpenID Connect Security Considerations*, 2017. https://www.nds.ruhr-uni-bochum.de/media/ei/veroeffentlichungen/2017/01/13/OIDCSecurity_1.pdf, as of September 30, 2020.

[27] C. Mainka, V. Mladenov, and J. Schwenk: *Do not trust me: Using malicious IdPs for analyzing and attacking single sign-on*. CoRR, abs/1412.1623, 2014. http://arxiv.org/abs/1412.1623.

[28] C. Mainka, V. Mladenov, J. Schwenk, and T. Wich: *Sok: Single sign-on security — an evaluation of OpenID Connect*. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pp. 251–266, April 2017.

[29] S. Matsumoto, S. Hitz, and A. Perrig: *Fleet: Defending sdns from malicious administrators*. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, p. 103–108, New York, NY, USA, 2014. Association for Computing Machinery, ISBN 9781450329897. https://doi.org/10.1145/2620728.2620750.

[30] Max Moroz: *A couple more common OAuth 2.0 vulnerabilities*, 2017. https://blog.avuln.com/article/4, as of September 30, 2020.

[31] V. Mladenov: *On the security of single sign-on*. doctoralthesis, Ruhr-Universität Bochum, Universitätsbibliothek, 2017.

[32] Mozilla: *307 temporary redirect*, 2020. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/307, as of September 30, 2020.

[33] OpenID Foundation: *OpenID Connect core 1.0 incorporating errata set 1*, 2014. https://openid.net/specs/openid-connect-core-1_0.html, as of September 30, 2020.

[34] OpenID Foundation: *OpenID Connect discovery 1.0 incorporating errata set 1*, 2014. https://openid.net/specs/openid-connect-discovery-1_0.html, as of September 30, 2020.

[35] OpenID Foundation: *OpenID Connect dynamic client registration 1.0 incorporating errata set 1*, 2014. https://openid.net/specs/openid-connect-registration-1_0.html, as of September 30, 2020.

[36] P. Jones, G. Salgueiro, M. Jones, and J. Smarr: *RFV7033: WebFinger*, 2013. https://tools.ietf.org/html/rfc7033, as of September 30, 2020.

[37] Patrik Hudak: *Heroku proofs*, 2018. https://github.com/EdOverflow/can-i-take-over-xyz/issues/38, as of September 30, 2020.

[38] Portswigger: *Burp Suite*, 2020. https://portswigger.net/burp, as of September 30, 2020.

[39] T. Saito, S. Shibata, and T. Kikuta: *Comparison of OAuth/OpenID Connect security in america and japan.* In L. Barolli, K.F. Li, T. Enokido, and M. Takizawa (eds.): *Advances in Networked-Based Information Systems*, pp. 200–210, Cham, 2021. Springer International Publishing, ISBN 978-3-030-57811-4.

[40] Salesforce: *Responsible disclosure policy*, 2020. https://trust.salesforce.com/en/security/responsible-disclosure-policy/, as of September 30, 2020.

[41] Salesforce: *Thanks to our trailblazing customers, we're #1 in sales applications*, 2020. https://www.salesforce.com/campaign/worlds-number-one-SALES/, as of September 30, 2020.

[42] D. Serpanos and R.J. Lipton: *Defense against man-in-the-middle attack in client-server systems.* Proceedings. Sixth IEEE Symposium on Computers and Communications, pp. 9–14, 2001.

[43] Statista GmbH: *Sicherheit im Netz: Der große Passwort-Stress.* https://de.statista.com/infografik/7705/der-grosse-passwort-stress/, 2017. Online, as of September 30, 2020.

[44] Statista GmbH: *Internetnutzung: So erstellen die Deutschen ihre Passwörter.* https://de.statista.com/infografik/17492/so-erstellen-die-deutschen-ihre-passwoerter/, 2019. Online, as of September 30, 2020.

[45] R. Steinegger, A. Hotz, N. Hintz, and S. Abeck: *Migration von OpenID Connect in eine bestehende Anwendungslandschaft.* Sept. 2017.

[46] A. Sudhodanan, S. Khodayari, and J. Caballero: *Cross-Origin State Inference (COSI) attacks: Leaking web site states through XS-Leaks*, 2019.

[47] A. Swinnen: *Authentication bypass on airbnb via OAuth tokens theft*, 2017. https://www.arneswinnen.net/2017/06/authentication-bypass-on-airbnb-via-oauth-tokens-theft/, as of September 30, 2020.

[48] T. Lodderstedt, J. Bradley, A. Labunets, and D. Fett: *OAuth 2.0 security best current practice*, 2020. https://tools.ietf.org/html/draft-ietf-oauth-security-topics-15, as of September 30, 2020.

[49] T. Lodderstedt, M. McGloin, and P. Hunt: *RFC6819: OAuth 2.0 threat model and security considerations*, 2013. https://tools.ietf.org/html/rfc6819, as of September 30, 2020.

[50] Terence Eden: *Incorrect details on OAuth permissions screen allows DMs to be read without permission*, 2018. https://hackerone.com/reports/434763, as of September 30, 2020.

[51] Xianbo Wang, Wing Cheong Lau, Ronghai Yang, and Shangcheng Shi: *Make redirection evil again: Url parser issues in OAuth*, 2019. https://i.blackhat.com/asia-19/Fri-March-29/bh-asia-Wang-Make-Redirection-Evil-Again-wp.pdf, as of September 30, 2020.

[52] Y. Rekhter, B. Moskowitz, D. Karrenberg, and G. de Groot: *RFC1597: Address allocation for private internets*, 1994. https://tools.ietf.org/html/rfc1597, as of September 30, 2020.

A Evaluation Table for Identity Providers

Table A.1: Applied evaluation table for Identity Providers, only negative findings are explained in detail.

Vulnerability	**Keycloak**	**GitLab**	**Salesforce**	**Amazon Cognito**
"Sub" claim Spoofing				
"Sub" claim differs in UserInfo Response and ID Token [26, Section 3.1.6]	✔	✔	✔	✔
Redirect URI				
"redirect_uri" is not correctly checked against pre-registered value for SP [33, Section 3.1.2.1.]	✔	✔	✔	✔
Redirect URI supports dangerous schemes [new]	E.g. `data`, `javascript` allowed	✔	✔	E.g. `javascript` supported
HTTP				
HTTP status code 307 for redirection [15, Section III, A. 5)]	✔	✔	✔	✔
Request Object, request_uri and Registration Object				

Vulnerability	Keycloak	GitLab	Salesforce	Amazon Cognito
SSRF using "request_uri" [15, Section III, A. 8)]	Default configuration supports "request_uri"	N/A	N/A	N/A
Registration Object allows to overwrite preconfigured values [new]	✔	N/A	N/A	N/A
Mismatch between contents of Request Object and values of OIDC Request (GET parameters) possible [33, Section 6.2.]	✔	N/A	N/A	N/A
Only secure signing methods allowed for JWT signatures	Per default "any" algorithm including "none" is allowed	N/A	N/A	N/A
Access/Refresh Token and Client Credentials protection				
Insecure storage of refresh/access tokens e.g. in accessible DB or on file system [49, Section 5.3.3.]	N/A	access_-tokens are accessible using "gitlab-psql" without further authentication	N/A	N/A

Vulnerability	**Keycloak**	**GitLab**	**Salesforce**	**Amazon Cognito**
Insecure storage of "client_id" and "client_secret" [49, Section 5.3.3.]	If built-in H2 database is used: Default credentials!	N/A	N/A	N/A
Weak Client Credentials [33, Section 16.19]: "*For instance, for HS256, the client_secret value MUST contain at least 32 octets (and almost certainly SHOULD contain more [...])*"	Clients can choose weak "client_secret"	✔	✔	✔
Online Guessing of Client Credentials [49, Section 4.3.5.]	No rate limiting + Oracle at token endpoint	✔	N/A	N/A
"access_token" and "refresh_token" are long-living (and no scope limitation was performed to limit impact) [49, Section 3.2.]	✔	✔	No "refresh_token" at all, test indicates that "access_token" is longer than 60 minutes valid	✔
"Cache-Control" misconfiguration may leak sensitive information [49, Section 4.6.6.]	✔	✔	✔	✔

Vulnerability	**Keycloak**	**GitLab**	**Salesforce**	**Amazon Cognito**
"refresh_token" is not bound to "client_id" [49, Section 3.3.]	✔	✔	N/A	✔
No "refresh_token" rotation: It is advised to set a new "refresh_token" on each refresh request to prevent reuse. [48, Section 4.12.2.]	New "refresh_token" is issued if old one is used, but the old token is not invalidated	✔	N/A	No "refresh_token" rotation
No possibility for End-Users to revoke "access_token" and "refresh_token" [new]	✔	✔	✔	No End-User revocation implemented
Code protection				
Miss of Client authentication: (whereas possible – if client_secret was issued) authentication is enforced when code is redeemed [49, Section 5.2.3.]	✔	✔	✔	✔
"code" has too long expiry time (> 10 minutes) [11, Section 4.1.2.]	Default value 60 minutes	✔	Code is longer than 10 minutes valid	✔
"code" is more than one-time usable [11, Section 4.1.2.]	✔	✔	✔	✔

Vulnerability	**Keycloak**	**GitLab**	**Salesforce**	**Amazon Cognito**
The "code" is not bound to the "client_id" it was issued for [11, Section 4.1.2.]	✔	✔	✔	✔
The "code" is not bound to the "redirect_uri" used as the redirection target of the client in the End-User authentication process [11, Section 4.1.2.]	✔	✔	Allowed to use another valid and registered "redirect_uri" in token request than in user authorization in frontend	✔
"code" injection due to missing binding between "nonce" and "code" in "id_token". Other mitigations do not work if attacker may choose victim's "nonce" value! [48]	✔	✔	✔	✔
Race condition when redeeming "code" [30]	✔	✔	N/A	N/A
Missing invalidation of "code" if access is revoked [30]	✔	Code is not invalidated if user revokes permissions	✔	N/A (Users can not revoke access)
Denial-of-Service				

Vulnerability	**Keycloak**	**GitLab**	**Salesforce**	**Amazon Cognito**
DoS by requesting many tokens for one user [49, Section 4.4.1.11.]	✔	✔	N/A	N/A
PKCE				
IdP does not support PKCE [48, Section 2.1.]	✔	Not implemented	✔	✔
IdP does not indicate if it supports PKCE, "*either by (a) publish the element'code_challenge-_methods_supported' in their AS metadata ([RFC8418]) containing the supported PKCE challenge methods (which can be used by the client to detect PKCE support) or (b) provide a deployment-specific way to ensure or determine PKCE support by the AS*" [48, Section 2.1.]	✔	N/A	✔	✔
Audience Confusion				

Vulnerability	**Keycloak**	**GitLab**	**Salesforce**	**Amazon Cognito**
IdP allows clients "*to influence their "client_id" or "sub" value or any other claim if that can cause confusion with a genuine resource owner*" [48, Section 2.6.]	Clients can choose their clientId on their own. Additionally, Keycloak allows to add colliding clientIds like "Test" and "test" that could cause confusion at a genuine RP.	✔	✔	✔
Consent Screen				
Incorrect information on consent screen, SP receives more rights than specified and consented by resource owner on consent screen [50]	✔	✔	✔	✔
Dynamic Client Registration				
No access control for Dynamic Client Registration, anyone may register clients (instead of using initial access token or trusted hosts policy)	✔	N/A	Initial access_token required to register client, BUT: The token can be reused for multiple clients	N/A

Vulnerability	Keycloak	GitLab	Salesforce	Amazon Cognito
"sector_identifier_uri" supports HTTP without TLS (optional parameter but if it is supported, this is mandatory) [35, Section 5.]	URI is allowed to use HTTP	N/A	N/A	N/A
SSRF using URIs from Client Metadata [new]	N/A	N/A	N/A	N/A
logo_uri, client_uri, policy_uri, tos_uri are vulnerable to client side vulnerabilities like XSS [new]	N/A	N/A	N/A	N/A
Client Authentication				
client_secret_jwt/ private_key_jwt: none-Algorithm is supported ("Signature Exclusion")	✔	N/A	N/A	N/A
client_secret_jwt/ private_key_jwt: SSRF using jku and other claims	✔	N/A	N/A	N/A
client_secret_jwt/ private_key_jwt: Token exp date is not checked	✔	N/A	N/A	N/A
client_secret_jwt/ private_key_jwt: JTI is not checked	✔	N/A	N/A	N/A

Vulnerability	**Keycloak**	**GitLab**	**Salesforce**	**Amazon Cognito**
client_secret_jwt/ private_key_jwt: Required parameter [33] are not enforced	"iss" is not checked at all, instead Keycloak solely relies on "sub"	N/A	N/A	N/A
client_secret_jwt/ private_key_jwt: Key Confusion [26]	✔	N/A	N/A	N/A

B Evaluation Table for Service Providers

Table B.1: Applied evaluation table for Service Providers, only negative findings are explained in detail.

Vulnerability	**Keycloak**	**Bitbucket**	**GitLab**	**Salesforce**	**Amazon Cognito**
Replay Attacks					
ID Token Replay	✔	✔	"exp" and "nonce" checked, "iat" not	Not verified at all, but only Code Flow available	No "nonce", "iat" not verified
Signature Manipulation					
No signature validation at all [26, Section 3.1.2]	✔	No signature verification at all	✔	Not verified at all, but only Code Flow available	✔
Support of "none" algorithm [26, Section 3.1.2]	✔	No signature verification at all	✔	Not verified at all, but only Code Flow available	✔
Token Recipient Confusion					
"aud" claim of "id_token" is not validated [26, Section 3.1.3]	✔	✔	✔	No id_token validation at all	✔
ID spoofing					

Vulnerability	**Keycloak**	**Bitbucket**	**GitLab**	**Salesforce**	**Amazon Cognito**
ID spoofing in the ID token [26, Section 3.1.4]	✔	External Login provider can authenticate any internal user with any privileges without further consent needed from internal user	✔	Information is only retrieved from UserInfo endpoint but checked correctly	✔
ID spoofing in the ID token - email [26, Section 3.1.4]	✔	✔	✔	Information is retrieved from UserInfo endpoint but checked correctly	✔
Issuer Confusion [26, Section 3.1.4]	✔	✔	Parser issue enables Issuer confusion	No discovery implemented, but unsafe endpoints are supported during manual configuration	✔
Key Confusion					
Key Confusion with wrong references [26, Section 3.1.5]	✔	No signature verification at all	✔	Not verified at all, but only Code Flow available	✔

Vulnerability	**Keycloak**	**Bitbucket**	**GitLab**	**Salesforce**	**Amazon Cognito**
Key Confusion with session overwriting [26, Section 3.1.5]	✔	No signature verification at all	✔	Not verified at all, but only Code Flow available	✔
Key confusion using asymmetric key (as string, only n or e)	✔	No signature verification at all	✔	Not verified at all, but only Code Flow available	✔
Sub claim spoofing					
UserInfo Response and id_token are checked regarding their "sub" claim [26, Section 3.1.6]	✔	✔	✔	✔	✔
State parameter					
CSRF Session Donation [15, Section III, A 2)]	✔	✔	✔	✔	✔
"state" is used to carry application state but is not integrity protected [48, Section 4.7.1.]	✔	✔	✔	✔	✔
"state" is reusable enabling DoS Amplification [new]	State is reusable	✔	✔	State is reusable	State is reusable
Leakage					

Vulnerability	Keycloak	Bitbucket	GitLab	Salesforce	Amazon Cognito
state or authorization "code" are leaked through Referer header [15, Section III, A 2)]	Administrative users can inject resources to login screen, resulting in "state" disclosure through Referer	Referer leaks OIDC parameters on error page	✔	✔	✔
Insecure storage of refresh/access tokens e.g. in accessible DB or on file system [49, Section 4.1.2.]	✔	N/A (storage mechanism not identified)	access_tokens are accessible using "gitlab-psql" without further authentication	N/A	N/A
SP passes "access_token" via query parameter [49, Section 4.4.]	✔	✔	✔	✔	✔
CSRF					
Login Initiation Endpoint enabled (allows intentional CSRF) [15, Section III, A 7)]	✔	Intentional CSRF possible	✔	✔	✔
Non-normative External Login endpoint without CSRF protection [new]	✔	CSRF Login into victim account possible	✔	CSRF Login into victim account possible	CSRF Login into victim account possible
SSRF					

Vulnerability	**Keycloak**	**Bitbucket**	**GitLab**	**Salesforce**	**Amazon Cognito**
SSRF to localhost or private IPs (may allow Portscan or Blind SSRF) [26, Section 2.1.2]	Discovery allows blind SSRF to localhost	Discovery allows blind SSRF to localhost (filter bypassed, arbitrary paths and GET parameters, HTTP and HTTPS)	UserInfo Request allows SSRF to localhost + CRLF injection allows RCE if Redis Instance is present	✔	Discovery allows SSRF with Error Message disclosure allowing Portscans, Token/UserInfo Request allow blind SSRF)
Open Redirect					
Covert Redirect after successful authentication [48, Section 4.9.1.]	Redirect to any relative path	Redirect to any relative path	✔	Redirect to any relative path	✔
HTTP Referer header-based open redirect: User is redirected to the referer he came from when starting OAuth/OIDC flow [47]	✔	✔	✔	✔	✔
Login Confusion					
Login CSRF and Covert Redirect to relative path can be chained to “Login Confusion” attack [new]	✔	CSRF + Redirect can be chained to Login confusion	✔	CSRF + Redirect can be chained to Login confusion	✔
Mix-Up Attacks					

Vulnerability	**Keycloak**	**Bitbucket**	**GitLab**	**Salesforce**	**Amazon Cognito**
SP does not use "*either (a) a distinct redirect URI per IdP or (b) check the'iss'and'client_id'in the id_token*" [48, Section 4.4.]	✔	Only one IdP supported at a time	Only one IdP supported at a time	✔	Mix-up possible: No distinct Redirection Endpoint + No additional countermeasures implemented
Deprecated Grants					
SP uses deprecated Implicit Flow (risking token leakage) [48, Section 2.1.2]	✔	✔	✔	✔	Implicit flow can be configured
SP uses deprecated Resource Owner Password Credentials Grant [48, Section 2.4]	✔	✔	✔	✔	✔
Discovery					
Unsafe URLs (HTTP without TLS) are supported [34, Section 7.2]	HTTP without TLS supported	HTTP without TLS supported	Discovery endpoint enforces HTTPS, but observed endpoints are allowed to use HTTP	No discovery implemented, but unsafe endpoints are supported during manual configuration	✔
Denial of service					

Vulnerability	**Keycloak**	**Bitbucket**	**GitLab**	**Salesforce**	**Amazon Cognito**
DoS using malicious Endpoint Attack [26, Section 2.1.4]	Higher Memory Consumption observed, but no actual DoS	✔	Timeout "*Rack::Timeout::RequestTimeoutException (Request ran for longer than 60000ms)*"	N/A	N/A
Injection					
XSS using IdP provided values [26, Section 2.1.3]	✔	✔	✔	✔	✔

Vulnerability	Keycloak	Bitbucket	GitLab	Salesforce	Amazon Cognito
CRLF injection using IdP provided values [new]	Log Injection: Sensitive Data in log files	HTTP Request splitting in UserInfo request: “access_token” is not sanitized, resulting in SSRF to chosen endpoint (set using discovery) with arbitrary injected headers. As GitLab is often configured with a Redis instance, if the instance is present RCE may be a consequence.	✔	Internal Server Error: We can not prove a log injection, but as Salesforce uses Java like Bitbucket and shows comparable behavior, potentially there is a log injection, too.	✔

C Source Code: Malicious Identity and Service Providers

The most recent version of each implementation could be found on Github:

- Malicious Identity Provider: `https://github.com/lauritzh/oidc-custom-idp`.
- Malicious Service Provider: `https://github.com/lauritzh/oidc-custom-sp`.