# Frameworks For Centralized Authentication And Authorization

by

Ken Erikson

Thesis of 30 ECTS credits submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
**Master of Science (M.Sc.) in Computer Science**

June 2020

# Frameworks For Centralized Authentication And Authorization

Ken Erikson

Thesis of 30 ECTS credits submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
**Master of Science (M.Sc.) in Computer Science**

June 2020

Student:

Ken Erikson

Examining Committee:

Sébastien Lafond

Dragos Truscan

Marcel Kyas

........................................................................
date

.....................................................................................................................................................................
Ken Erikson
Master of Science

Ken Erikson

# ABSTRACT

A commonly used method of authentication on the Internet is to provide a combination of a username and a password. One way to make this method more secure is to have long passwords. Single sign-on solutions have led to users having to remember fewer passwords and the passwords can, therefore, be longer than previously. Different standardized frameworks can be used when implementing these types of solutions, each with various advantages and disadvantages.

This thesis examines different authentication and authorization frameworks available to integrate a centralized single sign-on solution with an existing system. The studied frameworks are OAuth 2.0, OpenID Connect, SAML, LDAP, Kerberos and RADIUS. These frameworks are reviewed and compared with each other on a high level with an emphasis on security and features. A subset of them, namely OAuth 2.0, OpenID Connect and SAML, is chosen to be more extensively evaluated through integration with Hibox Systems's internal solution. The reliability of the system is tested through simulating heavy load conditions, while successful throughput and system resource usage are used as metrics to determine the efficiency of the system.

The integration of both OpenID Connect, combined with OAuth 2.0, and SAML had a smooth user experience during normal authentication flows, but the combination of OAuth 2.0 and OpenID Connect had an advantage over SAML during heavy load conditions. The user experience was kept on a reasonable level for a larger number of simultaneous users with the OpenID Connect and OAuth 2.0 integration and the frameworks are therefore deemed to be better suited for the given use case.

**Keywords**: Authentication, Authorization, OAuth 2.0, OpenID Connect, SAML

# CONTENTS

**Abstract** **v**

**Contents** **vi**

**List of figures** **viii**

**Glossary** **ix**

**Abbreviations** **x**

**1 Introduction** **1**
1.1 Context . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2 Purpose of this thesis . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Methodology . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.4 Thesis structure . . . . . . . . . . . . . . . . . . . . . . . . . . 3

**2 Background** **5**
2.1 Authentication . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.2 Authorization . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.3 Related work . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**3 Frameworks** **8**
3.1 OAuth 2.0 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
3.2 OpenID Connect . . . . . . . . . . . . . . . . . . . . . . . . . . 13
3.3 SAML . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
3.4 LDAP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
3.5 Kerberos . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
3.6 RADIUS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

**4 Evaluation setups** **26**
4.1 Criteria . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
4.2 Test setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
4.2.1 Network setup . . . . . . . . . . . . . . . . . . . . . . . . 29
4.2.2 Identity provider server . . . . . . . . . . . . . . . . . . . 30

4.2.3 Hibox server . . . . . . . . . . . . . . . . . . . . . . . . 31
4.2.4 Load testing tool . . . . . . . . . . . . . . . . . . . . . . 32
4.3 Test scenarios . . . . . . . . . . . . . . . . . . . . . . . . . . 33
4.4 Performance metrics collection . . . . . . . . . . . . . . . . . 34
4.4.1 Request metrics . . . . . . . . . . . . . . . . . . . . . . . 34
4.4.2 Resource monitoring . . . . . . . . . . . . . . . . . . . . 34
4.5 Evaluation of the experimental setups . . . . . . . . . . . . . . . . . 35

**5 Implementation** **37**
5.1 Platform . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
5.2 Integration overview . . . . . . . . . . . . . . . . . . . . . . . 38
5.3 Authentication flows . . . . . . . . . . . . . . . . . . . . . . . 40
5.3.1 OpenID Connect flow . . . . . . . . . . . . . . . . . . . . 40
5.3.2 SAML flow . . . . . . . . . . . . . . . . . . . . . . . . . 42
5.4 Challenges . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

**6 Results analysis** **47**
6.1 Implementation overview . . . . . . . . . . . . . . . . . . . . . 47
6.2 Evaluation overview . . . . . . . . . . . . . . . . . . . . . . . 48
6.2.1 User throughput . . . . . . . . . . . . . . . . . . . . . . . 49
6.2.2 Response time . . . . . . . . . . . . . . . . . . . . . . . . 50
6.2.3 Error rate . . . . . . . . . . . . . . . . . . . . . . . . . . 53
6.2.4 Data rate . . . . . . . . . . . . . . . . . . . . . . . . . . 53
6.2.5 RAM increase . . . . . . . . . . . . . . . . . . . . . . . . 55
6.2.6 CPU utilization . . . . . . . . . . . . . . . . . . . . . . . 57
6.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59

**7 Future work and conclusions** **60**
7.1 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
7.2 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

**Bibliography** **62**

**Swedish summary** **67**

# LIST OF FIGURES

3.1 OAuth 2.0 Authorization Code flow. . . . . . . . . . . . . . . . . . . 10
3.2 OAuth 2.0 network diagram. . . . . . . . . . . . . . . . . . . . . . . . 12
3.3 SAML authentication flow. . . . . . . . . . . . . . . . . . . . . . . . . 16
3.4 LDAP flow. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
3.5 The entities and shared keys in a Kerberos system. . . . . . . . . . . . 21
3.6 The RADIUS request and possible responses. . . . . . . . . . . . . . . 24

4.1 Network diagram of the use case. . . . . . . . . . . . . . . . . . . . . 27
4.2 Network diagram of the test setup. . . . . . . . . . . . . . . . . . . . 30

5.1 OpenID Connect authentication flow. . . . . . . . . . . . . . . . . . . 41
5.2 SAML authentication flow. . . . . . . . . . . . . . . . . . . . . . . . . 43

6.1 Throughput of users. . . . . . . . . . . . . . . . . . . . . . . . . . . 49
6.2 Average response times. . . . . . . . . . . . . . . . . . . . . . . . . 50
6.3 Average response times for up to 100 users. . . . . . . . . . . . . . . 51
6.4 Error rate of authentication requests. . . . . . . . . . . . . . . . . . 52
6.5 Bytes transmitted per second for authentication requests. . . . . . . 54
6.6 RAM increase on the Gluu identity server relative to a baseline. . . 55
6.7 RAM increase on the Hibox server relative to a baseline. . . . . . . 56
6.8 CPU utilization of the Gluu identity server. . . . . . . . . . . . . . 57
6.9 CPU utilization of the Hibox server. . . . . . . . . . . . . . . . . . . 58

# GLOSSARY

*Standard*
An agreed upon and approved, by general consent, model of how something ought to work on a high level.

*Protocol*
Detailed description of how a standard works and how it should be implemented.

*Framework*
Implementation of a well-defined protocol.

*Principal*
An entity that seeks to be authenticated by a system.

*Supplicant*
An entity at the edge of a point-to-point LAN that seeks to be authenticated.

*Single sign-on*
A system where one trusted entity can forwards proof of authentication accepted by other independent systems.

*Request for Comments*
A formal document drafted by the Internet Engineering Task Force (IETF) that has been put up for public review and revision.

*Identity provider*
A system component that is able to provide proof of authentication of an end user.

# ABBREVIATIONS

| | |
|---|---|
| **SAML** | Security Assertion Markup Language |
| **LDAP** | Lightweight Directory Access Protocol |
| **RADIUS** | Remote Authentication Dial-In User Service |
| **XML** | Extensible Markup Language |
| **PKCE** | Proof Key for Code Exchange |
| **RFC** | Request for Comments |
| **ACL** | Access-control list |
| **SQL** | Structured Query Language |
| **IETF** | Internet Engineering Task Force |
| **SSO** | Single sign-on |
| **GUI** | Graphical user interface |
| **CSS** | Cascading Style Sheets |

# 1 INTRODUCTION

The Internet, as we know it today, has existed for close to four decades and has evolved much since its invention [1]. The original use case of the Internet and, later, the world wide web was mainly simple communication and efficient information sharing. The need for making the Internet more secure became more important as the popularity of it started rising and the general public started using the world wide web on a daily basis [2]. The Internet was no longer mostly used for searching information, but also for tasks where sensitive information was shared over a medium that had put a focus on being openly available to everyone. As the online community grew, so did the attack surface and many new security threats capitalized on this. At the same time as new security threats were recognized, it had also become clear that the widespread use of the Internet had led to the fact that a complete overhaul of the system had become very difficult, if not completely impossible, to achieve [2]. The bigger stakeholders and organizations have now come together to make the security of the Internet as good as possible with the given circumstances and the Internet's many inherent limitations [2].

One of the most crucial security problems on the Internet is that two entities need to be able to verify the identity of each other. Website-specific certificates and trusted certificate authorities have become the standard way of confirming the authenticity of the website one is visiting [3]. How a website can verify the identity of a user is a completely different problem. Every user having his own certificate would not have the required usability and would lead to a single point of failure. The most used method currently is to have a username and a password to authenticate oneself to a service [4]. The problem then becomes that the user needs to keep track of one username and password combination per service, as well as go through a cumbersome registration process every time a new one is opened. One push to reduce the inconvenience is to use a trusted source as a central identity provider [5].

There are many ways an identity provider can function behind the scenes, but in its most basic form a user logs in to the identity provider and the identity provider then verifies the identity of the user to the service [5]. Security is crucial in any publicly available system, and this has led to the availability of a multitude of different authentication frameworks and protocols that an identity provider can follow and implement. By conforming to a certain protocol, services can work together through an agreed-upon interface and the services are not tightly coupled together [6]. Every protocol has its own strengths and weaknesses, and knowing which one to choose for any certain project can be a difficult task.

Confirming the identity of the user is only the first step. The system also has to verify that any authenticated user is authorized to do what the user is trying to achieve. A centralized solution that can handle both authentication and authorization can be split up into separate parts, and it is not always clear how these work together.

## 1.1 Context

The implementation part of this thesis is done for, and in collaboration with, Hibox Systems.

Hibox Systems have their own in-house extensive entertainment and management system deployed on many hospitality, healthcare and Internet service provider customers' servers. To expedite the support process provided to the customer, while also providing a more secure service, Hibox Systems have investigated the possibility of establishing a centralized single sign-on server for its support and developer personnel. The centralized service would need to be integrated with their own system, as well as be easily configurable for any possible change in personnel or updated level of customer security requirements.

This thesis will investigate the possible choices of frameworks, protocols and services available that fulfill the security and functional requirements. Furthermore, a complete functional single sign-on server will be integrated with Hibox Systems's own service. The solution will be heavily scrutinized by the Hibox Systems internal development team and only be taken past the proof-of-concept stage if it is deemed secure enough for systems deployed in production.

## 1.2 Purpose of this thesis

The purpose of this thesis is to give an insight into available authentication and authorization frameworks and protocols to show how these can be used in practice, and to present differences between available ones. Different state-of-the-art authentication and authorization frameworks and protocols will be examined to obtain a clear high-level overview. The focus will be exclusively on frameworks that have been proven to work securely and have been deployed widely, as well as extensively studied and reviewed.

## 1.3 Methodology

There is a multitude of frameworks available in literature to screen through. The initial selection of available state-of-the-art authentication and authorization frameworks is a critical step in this thesis, and the selection of frameworks will be motivated. The selected frameworks will then be compared with each other and tested.

The frameworks will initially be compared by analyzing features, security and complexity. Later, a subset of the initial frameworks will be integrated with Hibox Systems's solution and tested in an offline environment. The tests will consist of factors such as response times, throughput, and usability. The best performing solution will be more thoroughly implemented.

## 1.4 Thesis structure

This thesis is divided into three main parts. The first part outlines the current state of the field. The second part focuses on the implementation of a system that uses a chosen subset of the initially selected authentication and authorization frameworks, and by which criteria these frameworks may be evaluated. Finally, the evaluation results are discussed, the implementation is reviewed and prospects are considered.

The second chapter presents a general overview of the field of study, while the third chapter focuses on a high-level examination of the chosen frameworks. The fourth chapter describes the different evaluation criteria and test setups. Only a subset of viable frameworks will be used in the actual implementation, and the motivation behind excluding some frameworks will therefore be discussed in this chapter. In the fifth chapter, the implementation of a centralized authentication and authorization integration is presented. The sixth chapter will focus on the result analysis of the evaluation of frameworks and the implementation as a whole. In the seventh and final chapter, the conclusions will be discussed and future prospects will be deliberated.

# 2 Background

This chapter will present a general description of the field of user authentication and authorization. Authentication and authorization will be discussed separately to clarify the differences between the topics. The current state-of-the-art in the field will be examined. Existing works that discuss topics similar to the one in this thesis will be considered in the related work section, while specific frameworks will be comprehensively presented in the next chapter.

## 2.1 Authentication

Authentication is the process of verifying the identity of a user. Throughout the relatively short history of the Internet, there have been many different proposals for how authentication should be done online. Some proposals have been tried successfully to some extent, but the most popular one has always been a username and password combination [4]. Additions, such as 2-factor authentication and biometric scanners have seen some use, even though the added security is not always worth the extra inconvenience for the user [7].

Different solutions for making users use strong unique passwords for different services have been tried and the use of password managers is at least moderately widespread. Lately, there has been a push for using bigger corporations in the field, such as Facebook and Google, as identity providers for many services around the Internet [5]. This has led to fewer passwords needing to be remembered and managed, but it has also led to a single point of failure. This method, which provides simplicity and ease of use, has undoubtedly gained popularity among average users, and the reduction of logins per day has led to extra layers of security, e.g. 2-factor authentication, not being as intrusive as in the technologies from earlier days.

## 2.2 Authorization

Authorization involves specifying the rights to access certain resources. In any system that can be accessed by multiple users, what information can be accessed by whom is a crucial factor in guaranteeing the security of the system. The user generally has to be reliably authenticated before the system gives access to the proper resources, though an anonymous user may be given certain basic access, for example access to read particular resources.

Managing what access rights a specific user or a group of users have can be done through an access-control list, more commonly known as an ACL [8]. An ACL can be updated to add or revoke access previously given. There are many different kinds of ACLs available, but the main difference is the context in which they are used. Some common contexts where ACLs are widely used are in file systems, databases, networks, and active directories [8]. Some sort of ACL is available in almost every modern operating system, for example in Windows, macOS and Linux, all of which make extensive use out of ACLs. ACL is also deeply integrated into Structured Query Language (SQL), the most widespread database query language.

There are other ways to store authorization information, such as role-based access-control (RBAC) or not following any standard at all, but ACL is the most widespread access-control model available [9].

## 2.3 Related work

Authentication and authorization are two very broad and widely studied topics in the field of security. Most works are focused on describing one specific framework or solution. This thesis emphasizes comparing solutions and seeing when one framework might be better suited than another.

One of the available works that take a similar approach is "Securing digital identities in the cloud by selecting an apposite Federated Identity Management from

SAML, OAuth and OpenID Connect" [10], but in this article they have only examined a subset of the frameworks selected in this thesis. The field is also ever-changing, as updates on recommended practices are released and new problems are recognized.

Another similar one is "On the Substantiative Experiment Study of Proxing Assurance between OpenID and SAML: Technical Perspective for Private Information Box Project of Japanese e-Government" [11], which looks at only two of the selected frameworks. The use case that they investigate is also very different from the problem presented in this thesis in Chapter 4.

# 3 FRAMEWORKS

There are many different frameworks and protocols available for authentication and authorization, both proprietary and openly available standards. This thesis will focus on a subset of popular protocols, namely OAuth 2.0, OpenID Connect, SAML, LDAP, Kerberos and RADIUS. The protocols listed were chosen due to their recurring prevalence in papers and their general widespread use [12][13][14][15][16][17]. All of the chosen protocols had the possibility to fulfill the requirements of the initial use case presented by Hibox Systems, where an identity provider should provide authentication and authorization to a service provider.

Different frameworks have different use cases, advantages and disadvantages. Frameworks such as Kerberos and RADIUS have been built mostly for an internal company security solution with a focus on high security and some specialized features and logging. The OAuth2 framework is by design only meant to be used for authorization, and has been extended to have a standardized method of providing authentication by the OpenID Connect identity layer. LDAP is a protocol to authenticate and authorize access to available resources. SAML is an advanced open standard for transferring authentication and authorization data between different entities.

In this chapter, these frameworks will be reviewed and described concisely to present an overview of in which environment each framework is intended to be used in, as well as what features and limitations should be taken into account when employing them.

## 3.1 OAuth 2.0

The OAuth 2.0 authorization framework is a framework that enables a third-party application to obtain limited access to an HTTP service [18]. OAuth 2.0 has since its official release in 2012 quickly become the industry-leading framework for limiting access to exposed services. Companies started to experiment with their already available OAuth 2.0 solution for ways to use it for authentication. The service-specific hacks led to the need for OpenID Connect, which extended OAuth 2.0 with a standardized way of authentication [19].

In normal client-server authentication flows, a third-party application needs to have direct access to the client's credentials in order to access a restricted resource [18]. This leads to third-party applications having unnecessarily extensive access, and a single compromised third party can lead to a resource owner's credentials becoming completely compromised. OAuth 2.0 adds an authorization layer and separates the client from the resource owner and limits the access of the third-party to only what is required. A set of credentials other than those of the resource owner is shared with the third party, and if the third party is compromised, those credentials can be invalidated without affecting the resource owner [18].

OAuth 2.0 is constantly being updated and revised by the IETF OAuth Working Group and the general community, who all try to reach a common consensus. The goal of any revision is to increase the security of OAuth 2.0 in general and to make sure the framework reflects actual use cases. Recommendations for how the framework should be used are presented by the most recently published OAuth 2.0 Security Best Current Practice [20]. New recommendations have historically been created when browser technologies have added security features that the framework can utilize. There have been people voicing ideas of moving to version 2.1 of the OAuth framework to simplify all the current additions and recommended practices into one single Request for Comments (RFC) [21].

The OAuth 2.0 framework defines a number of grant types. A grant type refers to how an application can acquire an access token. There are four different OAuth 2.0 authorization grant types available in the original OAuth 2.0 framework specific-

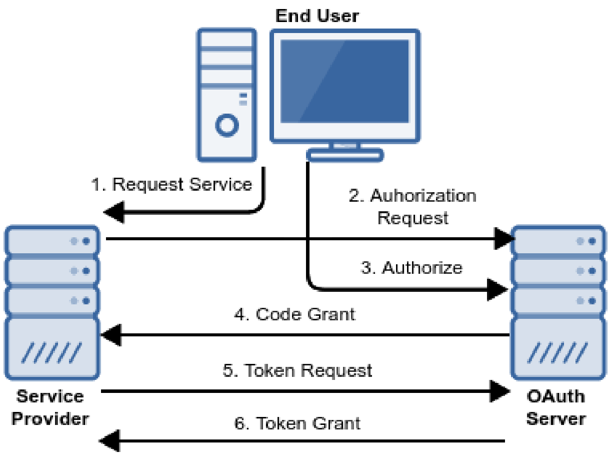

Figure 3.1: OAuth 2.0 Authorization Code flow.

ation, as well as the separate Refresh token grant type. There are also extensions available and completely new grant types proposed [18].

The grant types present in the current specifications are Authorization Code, Client Credentials, Resource Owner Password Credentials, Implicit, Refresh Token and Device Code [18]. All of these grant types represent different supported ways for an application to gain an authorization token, which in turn can be used to gain access to restricted resources of the service provider.

The Authorization Code grant type is the most common grant type and can be used by both web apps and native apps [22] and is visualized in Figure 3.1. The Authorization Code flow starts when the application sends the user to the OAuth 2.0 server and the user is required to approve the application's authorization request for the flow to continue [18]. The user is then sent back to the application with an Authorization Code. The application uses the Authorization Code together with its own configured shared secret to procure the actual access token over a secure back channel. The Authorization Code flow only exposes the Authorization Code, not the token, to the inherently insecure client. The Authorization Code cannot be used for anything unless you also have the shared secret of the application. The Authorization Code flow has been extended with Proof Key for Code Exchange (PKCE) intended mainly for applications without the ability to securely store a shared secret, but it is recommended that all clients use PKCE for the added verification [20].

The Client Credentials grant type is used by applications to gain access to their own resources [18]. The application sends a request containing a client identifier and a secret to the OAuth 2.0 provider through a secure back channel. A response containing an authorization token is returned if the client credentials are successfully validated.

The Resource Owner Password Credentials grant type is used when the application sends the user credentials of the resource owner to the OAuth 2.0 provider in exchange for an access token [18]. This flow is not recommended, because the application requires full knowledge of the user's credentials and other grant types provide a more secure flow.

The Implicit grant type starts like the Authorization Code grant type with a redirect from the application to the OAuth 2.0 provider for approval of the authorization request [18]. The user is then sent back to the application with the access token included in the redirection. The Implicit authorization flow was previously recommended for clients without a secure way to store a secret, but has been superseded by using the Authorization Code grant with Proof Key for Code Exchange as an added security measure [20]. The insecurities were known when OAuth 2.0 was proposed, but no easy solution was possible with the available browser technologies and their limitations.

The Refresh Token grant type is used to obtain a new valid access token [18]. A refresh token is sent alongside the access token during the successful authorization using another grant type. The refresh token has a longer valid period than an access token. The application is therefore forced to regularly retrieve a new access token by sending a request containing the refresh token together with possible client credentials. No additional authentication is required, because a valid refresh token implies that the user has already successfully authenticated with the identity provider.

The Device Code authorization grant type was added as an extension to the original OAuth 2.0 framework specification in order to add an official way of authenticating devices where inputting a password is impractical, such as televisions

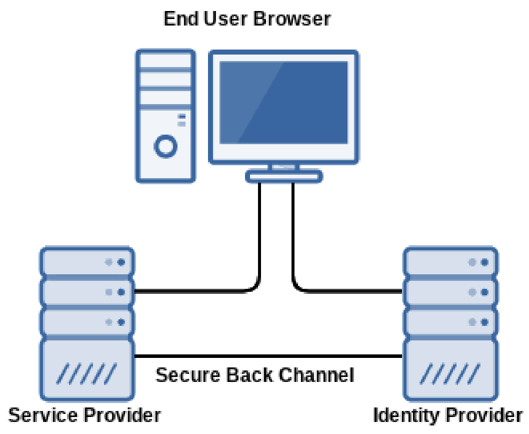

Figure 3.2: OAuth 2.0 network diagram.

and printers [23]. The device executes an authorization request containing a device code and a client identifier. The device then begins polling an endpoint until the user has successfully authorized the request by entering the code visible on the device through some other medium more suited for inputting user credentials, e.g. a phone or a computer.

There have been security issues with certain grant types in the past and the current best recommendations have been updated to reflect these [20]. The Client Credentials and Resource Owner Password Credentials grant types should never be used in a context so that the shared secrets and passwords are revealed to a third-party application. The implicit grant type should be disabled if possible and instead, the Authorization Code grant type should be used with the PKCE extension [20]. The PKCE extension has made the Authorization Code grant type applicable in most common authorization situations and is the preferred grant type, due to its inherent security. The security of the Authorization Code grant type is mainly due to the fact that the access token is only transmitted through the secure back channel, which can be seen in Figure 3.2. The secure back channel describes how requests are sent directly from the service provider to the identity provider and is used to exchange authorization grants for tokens. Sending Requests through the back channel is deemed to be more secure than sending requests through the browser of the end user, because of the higher risk of the access token being shared or stolen while passing through an untrusted user agent [24].

The security of the framework is not exclusively based on selecting the most secure usable grant type. The framework implementation is critical for the level of security of the system [25]. It is therefore recommended to use one of the many publicly available libraries, to protect the users and service providers by employing code examined and approved by others.

In conclusion, OAuth 2.0 is an ever-evolving framework that describes how authorization can be done efficiently and with a high level of security, as long as recommended grant types are used and the framework has been correctly implemented according to the protocol specifications.

## 3.2 OpenID Connect

OpenID Connect is a standardized identity layer to be used with the OAuth 2.0 protocol [26]. It extends the authorization capabilities of the OAuth 2.0 protocol by allowing clients to verify the identity of an end user based on the authentication performed by the Authorization Server [26]. Additionally, the OpenID Connect identity layer also exposes an endpoint, from which basic profile information about the end user can be retrieved [26].

The need for OpenID Connect came from when multiple large companies with a centralized login solution started building their own service-specific hack on top of OAuth 2.0 to enable the authentication flow to authenticate users [19]. The solution of the companies resembled the OpenID Connect flow, but using separate non-standardized specifications led to much confusion and complications when trying to integrate a specific login flow to one's own service [19].

OpenID Connect standardized all the messages to be sent during a login transaction [26]. Consequently, if a service has been integrated with one OpenID identity provider, then the service could be integrated against any number of other OpenID identity providers without much additional configuration.

The OpenID Connect authentication flow follows the flow of the configured

OAuth 2.0 authorization. The identity provider may readily decide what OAuth 2.0 flows are accepted by the authorization server. The choice of OAuth 2.0 flow is an important determiner for the security of the OpenID Connect solution. The scope "openid" is included in the initial request by the client, which informs the server that the client attempts to initiate an OpenID Connect exchange [26]. After the identity provider receives successful authorization by the resource owner, an ID token is sent to the client. The ID token, which is encoded as a JSON Web Token (JWT), contains basic profile information of the resource owner, as well as information about the token itself. In addition, there is also a possibility for the JWT to be signed and encrypted for additional verification and safety [26].

Due to the multiple supported OpenID Connect flows, the security of OpenID Connect rests mostly on the implementation of the framework and how well it conforms to the most recent recommended specifications. The security is also limited by the allowed OAuth 2.0 grant types that the underlying server accepts. For normal resource owner login flows, the Authorization Code grant type with the PKCE extension should be used when possible for ensuring maximum possible security [20]. There are optional security measures, for example, additional encryption and certificate signatures, defined in the framework specifications for when security is paramount [26].

The OpenID Connect protocol was designed with the sole purpose of creating an identity protocol that was simple for developers to integrate with existing systems [27]. This has led to the protocol being heavily used by large identity providers, such as Google, which in turn has led to extensive integration with a multitude of both bigger and smaller services [5]. The fast adaption rate of OpenID Connect can also be explained by the support of single-page web applications and mobile applications, where there is no secure way to store sensitive information.

The OpenID Connect protocol standardizes some features that are optional in the OAuth 2.0 specifications [28]. For example, scopes are always to be included in an OpenID Connect exchange, and identity providers are required to support dynamic registration of clients and endpoint discovery. Dynamic registration and endpoint discovery are both extremely helpful in automating and simplifying the integration with existing systems. The endpoint discovery means that the OpenID Connect provider exposes metadata of the available endpoints of the server at a well-known URL [26].

To conclude, OpenID Connect is an authentication protocol that is simple to integrate with existing services and secure even without its optional extra security features. The broad use of OpenID Connect means that the integration to additional services after the first one is very simple.

## 3.3 SAML

The Security Assertion Markup Language (SAML) is a standard that defines a framework used for exchanging security information between entities through a common XML framework [29]. The current version of the standard is SAML 2.0, which was released in its final revision in 2005.

For a long time, SAML has been one of the most popular frameworks for centralized login among corporations and organizations, such as universities [30]. Because SAML has been integrated with a variety of available enterprise software, it is often required by larger entities. This, in turn, has led to that software aimed towards such entities come with SAML integration. The adaption of newer technologies in these fields is slower than in many other areas, due to the lack of clear benefits of transitioning from a functioning system. The migration cost of large deployed systems has to be weighed against the improvement in other areas, such as security, ease of maintenance and integrations with new systems.

A SAML single sign-on login flow consists of an identity provider and a service provider as illustrated in Figure 3.3. The identity provider has knowledge of the login credentials of all users in the system, as well as an internal list of all the available service providers in the system. The user is, in the most basic flow, redirected to the identity provider when trying to access a service provider. The identity provider issues an authentication assertion to the user who successfully manages to authenticate, and the assertion can then be used to gain access to the requested

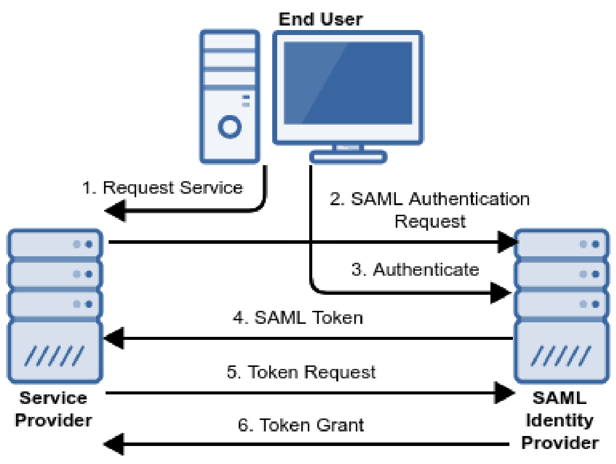

Figure 3.3: SAML authentication flow.

service [29].

The issued authentication assertion is trusted by the service provider because of the trust relationship that has been established between the identity and service providers during the initial configuration of the system. The trust relationship is established by sharing a metadata description of the providers. The metadata describes, for instance, what the authentication information is expected to look like, and what certificates will be used to sign the data to guarantee the validity of the messages sent [29].

SAML can be used by multiple organizations to form a login federation, where users of all member organizations can use a common login flow to access a shared service [30]. Federations are mainly used in very large companies and universities.

Troubleshooting SAML systems is often complicated, due to the huge number of available configuration options [31]. Different services are often configured so that the identity provider needs to send SAML assertions in a very specific format. The SAML identity provider may be following all the requirements in the SAML protocol, but if the service provider does not follow them, or is otherwise erroneously configured, there is nothing the identity provider can reconfigure to make it succeed.

SAML is a clearly standardized framework and much of its popularity comes from the fact that it is clear how to integrate with it, especially compared to an unstandardized approach [29]. Any service that follows the protocol specifications can be configured to authenticate to a SAML identity provider with minimal configuration. Both the SAML service provider and identity provider can provide each other with an XML metadata file, which describes what type of messages both of the systems are expecting.

The widespread use of the framework has also led to much support being available for both developers and administrators. There are pre-made libraries available for most popular programming languages, meaning that developers very seldom need to implement their own solution. Administrators have, apart from the official specifications, over a decade of documented tutorials and troubleshooting help available since the release of the current version of the standard in 2005 [29].

The SAML protocol has over the years been heavily audited and scrutinized, due to its extensive use in fields where wrongfully granted access could leak private and sensitive information to unauthorized users and entities [32].

The user experience for the user, as long as the system is properly configured, is very simple and, as with the OpenID Connect, very minor user interaction is needed apart from providing credentials [31]. The user does not need to be aware of the security assertions and exchanges that happen behind the scenes.

All in all, SAML is a popular protocol that is widely used in corporations and educational establishments. Its extensive use has led to a heavily scrutinized standard, where many of its possible angles of attack have been identified. The user of the resulting system only provides his login credentials, and is not required to have any knowledge of the complicated security that happens under the hood.

## 3.4 LDAP

The Lightweight Directory Access Protocol (LDAP) is a way to provide access to directory services distributed over a network [33]. It is an efficient way to quickly find seldom updated data. LDAP was originally designed as a way to simplify access to a complex enterprise directory system called the X.500, but it can also be used to provide single sign-on where a user needs to gain access to multiple services with the same credentials. LDAP originally had no authentication and security integrated, as this was only added in LDAP version 3 as the Simple Authentication and Security Layer (SASL) [34]. SASL includes a variety of authentication types, including possibilities for making use of two-factor authentication [35].

In LDAP, four standard information models are defined to fully describe what kind of information can be stored in the directories and what it is capable of doing with the stored information. The four models are the Information Model, Naming Model, Functional Model and Security Model [36].

The Information Model defines what type of information an LDAP directory can store. The Naming Model specifies how information in an LDAP directory is named, organized and referenced. The Functional Model defines how information in an LDAP directory can be accessed, updated and what can be done with it. The Security Model determines what privileges are needed to access specific information in an LDAP directory and how information can be secured [36].

The Lightweight part of the LDAP name comes from its strive to be as efficient as possible, while including as many crucial features as possible from the X.500 specification [33]. Many features that were deemed unnecessary were left out of the protocol completely. LDAP can also make use of long-lasting connections, which can be kept alive for days. This may be more efficient than HTTP-based protocols, which often use relatively short-lasting connections, due to being able to resume active connections rather than having to negotiate sessions and request a TLS connection [35].

LDAP is a popular framework to store user credentials in large and small corporations alike. It has been integrated with various popular software that average employees and administrators interact with sometimes multiple times per day, for example OpenVPN, Kubernetes and Docker. As such, their protocol has been heavily scrutinized and tested. LDAP is a crucial part of companies' IT infrastructure, and this has led to a constant strive for increased performance and scalability for the protocol that had its most recent version released in 1997 [35].

The login flow for LDAP is very straightforward both for the user and behind

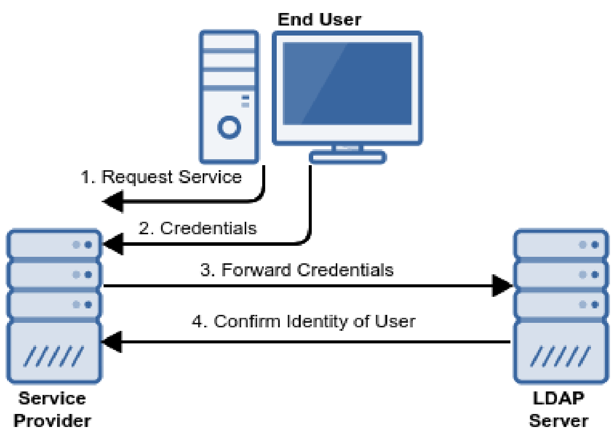

Figure 3.4: LDAP flow.

the scenes as visualized in Figure 3.4. The user supplies his user credentials and then the client sends them to the LDAP server for verification [33]. There are no additional checks or sessions. If the credentials are successfully verified, then the user is deemed to be properly authenticated.

The LDAP protocol was originally created without any authentication in the specifications, this was only added later [34]. The problem with using LDAP as a makeshift single sign-on service is that it is not built for securely transmitting login credentials over the public internet. It is generally not recommended to use as an identity provider outside a secure network or with untrusted third-party applications, due to the fact that the user has to provide his complete credentials. These credentials can then be used by anyone to impersonate the original user using a valid form of authentication.

Applications that are used over the open Internet are often connected to an identity provider using another framework [37]. That identity provider can then be connected securely to an LDAP server and use that as the identity source of truth.

To sum up, LDAP is a lightweight protocol that can be used to access distributed directory services that follow the X.500 specification [33]. A common use case is to store usernames and passwords in an LDAP server, which then applications can use as an identity provider [37]. The application and LDAP server should be on the same secure network.

## 3.5 Kerberos

The Kerberos network authentication service is a protocol based on the model presented by Needham and Schroeder [38] which was developed in the 1980s as a part of MIT's Project Athena [39], and is currently on its fifth version [40]. The protocol enables secure verification of principals, which are an entities that seek to be authenticated by a system, over open networks. Everything transmitted over the network is assumed to be openly readable and modifiable.

Kerberos uses strong cryptography to achieve this [41]. When new clients are registered to the Kerberos system, a private key is shared between the server and the client. The private key can be any long string, for example an encrypted password [38].

The authentication process starts by the principal authenticating itself to the key distribution center using the pre-shared private key, as well as requesting access to a specific application server. Upon being successfully authenticated, the principal receives a session key and a ticket-granting ticket which can, during a limited time, be used to access the agreed-upon application server [39]. The integrity of the ticket is verified by confirming that the shared encryption key can be used to decrypt it. The ticket-granting ticket, together with an authenticator, is sent to the application server for verification. The most important fields in authenticator are the current time, a checksum, and an optional encryption key, all of which are encrypted with the session key also included in the ticket-granting ticket [39]. The ticket-granting ticket and session key are saved by the system, so that the password does not have to be saved in memory for the duration of the session. The principal can then access multiple resources without constantly having to reinsert the password [40].

The way the secret keys are distributed in a Kerberos system can be seen in Figure 3.5. All communication between the different entities in the system is encrypted by the shared keys [40]. This ensures that the entities always can trust that all messages only can be read by the intended party. The keys shared between the client and the authentication server is based on the username and password of the supplicant.

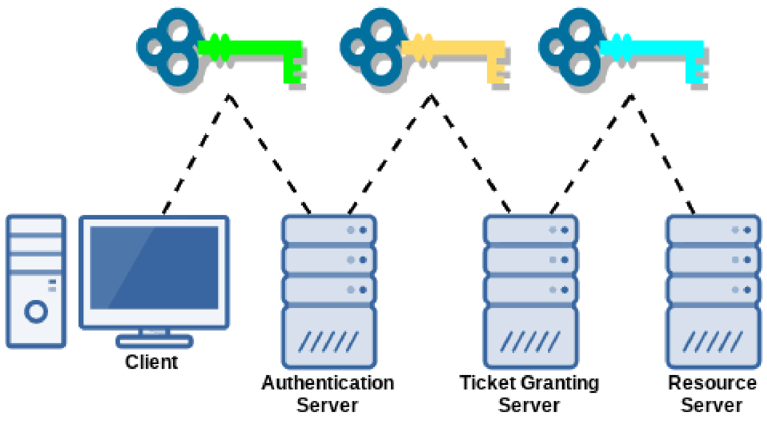

Figure 3.5: The entities and shared keys in a Kerberos system.

The other shared keys are configured during the initial configuration. The resource server containing the resources and services requested is, for instance, only aware of the secret key shared with the ticket-granting server and only trusts messages from that server. This chain of encryption ensures that only users passing through the whole authentication chain is provided with access to the requested resources.

The Kerberos protocol contains security safeguards against replay attacks, where successful authentication requests are recorded and then replayed with the intent of tricking the system into authenticating the entity sending the copied request. This is done by checking the timestamp of every request and rejecting it if the timestamp is the same as a previous authenticated request [40]. The timestamp is also verified to be within, by default, five minutes of the local authenticator server time in Coordinated Universal Time (UTC). This requirement leads to that all servers that have to be accessed through the Kerberos system have to be within 5 minutes of each other, which can be a challenge for the network administrators. Different time zones can make this more complicated for the administrators, because the time is always converted to UTC. There are different time services available to automate this after some configuration, but no service is infallible and authentication errors due to skewed clocks can become very complex [42].

The main security advantage of Kerberos is that no passwords are transmitted in plain text over any parts of the network. All services that are sending plain text passwords over the network need to either be integrated to be used with Kerberos or entirely removed from the network to get any real increase in security from using Kerberos [43]. Integrating Kerberos with applications that do not support it may require major programming efforts, which could be infeasible, especially for closed-source software [43].

As with any authentication system, the system is only secure if the trusted authenticator stays uncompromised. In the case of Kerberos, the system is only secure as long as the ticket issuer is kept secure [43]. The system builds on that the ticket issuer is trusted by all other parts of the system and tickets that are issued can be used to gain access to services and impersonate users.

To conclude, Kerberos is a secure authentication and authorization protocol with strict safety measures that ensure that a supplicant can safely authenticate with requests going over both secure and insecure networks, as long as the trusted entities are uncompromised. These strict safety measures also mean that the framework can become very complicated to configure for the network administrators, as well as difficult to successfully troubleshoot. The strict requirement of the server having the same time, within a small margin of error, is one part of the protocol that may end up causing a troublesome user experience. Kerberos is, however, a very secure system with a fluid user experience if everything is implemented and configured correctly.

## 3.6 RADIUS

RADIUS stands for "Remote Authentication Dial-In User Service" and is a protocol for carrying authentication, authorization and configuration information between a Network Access Server (NAS), to which a supplicant has initiated an authentication request, and an authentication server [44]. RADIUS is an all-in-one AAA (Authentication, Authorization and Accounting) protocol and a standard proposed by IETF [44]. The authentication part establishes how the identity of a supplicant can be confirmed. What an authenticated supplicant is allowed to do and access in the system is described by the authorization part. The Accounting part of this protocol defines how to log and keep track of any interaction with the RADIUS service and is out of the scope of this thesis.

RADIUS is a very extensible network [44] and one of the reasons it is still being used by many enterprises. RADIUS is, for example, not reliant on how the users of the system are stored, and if the identity source is locally integrated with the RADIUS server, in a directory service or in a cloud solution, which is becoming more popular [45]. The administrators then only have to configure the RADIUS server to point to the user managing system being used currently, if one already exists.

The RADIUS system consists of a centralized RADIUS server and any number of RADIUS clients. The server and clients communicate with each other through standard IP protocols, either TCP or UDP, operating on the port 1812, which is specifically reserved for RADIUS exchanges [44]. The RADIUS clients do not have to be on the same network as the RADIUS server, but extra security measures should be considered if RADIUS messages are sent over an insecure network.

The default security of RADIUS uses a shared secret along with the MD5 hashing algorithm to encrypt parts of the messages. The hashing is only performed on the password and other data that can be seen as sensitive, such as the username, is sent as part of the RADIUS message payload in plain text [46]. MD5 has also proven to be insufficiently secure and is not recommended to be used in applications where security is important [46].

Other more secure standards for encrypting the RADIUS messages are available, especially for when the RADIUS server sends messages over open networks. The added security can be by wrapping the exchanges in a Transport Layer Security (TLS) encrypted stream [47], as is the standard for most secure websites. Most RADIUS implementations also support RADIUS client-specific certificates, that can be used to encrypt all communication. Each client then must have a certificate installed and then the client must be configured with the certificate from the RADIUS server.

RADIUS was originally made for authenticating and authorizing access to network services, but there is no limitation in the protocol to authenticate users to any

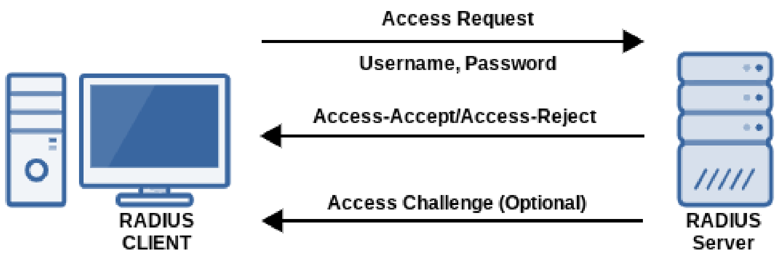

Figure 3.6: The RADIUS request and possible responses.

other type of services [44]. There are, however, features that are available in other single sign-on solutions that are not included in RADIUS, such as limited-time authentication and extra authorization options.

A normal use case for the RADIUS systems is to have the entry points to the network behind RADIUS authentication [44]. In these cases, the access points are RADIUS clients. One of the largest examples of this is the "eduroam" network, which provides internet access around education establishments in over a thousand countries [48]. The authentication to this network is provided through a unified RADIUS network.

The requests executed in a normal RADIUS exchange are visualized in Figure 3.6. The exchange starts with the user trying to authenticate with the RADIUS client, which then sends an Access-Request, containing the provided username and password, to the RADIUS server [44]. The RADIUS server then verifies that the client knows a previously configured shared secret and that the user credentials are correct. The RADIUS server then responds with either Access-Accept or Access-Rejected, with an optional Access-Challenge response as an extra security measure. If the authentication is successful it can also be used to access certain configured services on the network, using RADIUS as a single sign-on service [44].

As an added security measure, the RADIUS server can on a successful authentication return an Access-Challenge response [44]. The Access-Challenge response contains a number, which then has to be put through a series of calculations, through e.g. some software, to generate the correct expected response. The correct response should only be able to be calculated by trusted users. The RADIUS server then sends an Access-Accept or Access-Rejected based on if the response to the challenge was correct. For additional security, another Access-Challenge response can be sent on a success [44].

In conclusion, RADIUS is a protocol designed to be an all-in-one AAA solution and has seen widespread use in providing access to network services. A RADIUS network builds on RADIUS servers and clients having a trust relationship with each other, and RADIUS clients will have to be trusted with the credentials of the user. The RADIUS framework was not made explicitly for authentication of users on the web, and as such there are certain limitations of what the framework can do, compared to other solutions.

# 4 Evaluation setups

The evaluation and comparison of the frameworks focus on such aspects that are important in the field of security, namely reliability, scalability and safety.

To provide a clear and focused evaluation of the most promising frameworks, only a few of the initially chosen ones are presented in the latter parts of this thesis. The frameworks chosen to be more extensively tested are SAML and OAuth 2.0 with OpenID Connect as a security layer. The rest of the originally chosen frameworks are only discussed on a purely theoretical level using available previous works and framework specifications.

The use case, for which the chosen frameworks should function, is to be able to sign in to a service provider by authentication with a separate identity provider. The service provider and identity provider may be configured to form a trust relationship between the two entities. The end user, service provider and identity provider may be on three separate networks and any communication must be transmitted securely over the open Internet as illustrated in Figure 4.1. The authentication must be time-limited, so that one login does not give access to the service provider for an indefinite amount of time. There must be a possibility for the identity provider to give a closer specification of what the authenticated user is authorized to do while signed into the service. The user experience must be seamless, as not to make the workflow of the users more complicated than necessary, while still being secure.

The requirements have been evolving during the course of this thesis and some frameworks that were an option during the beginning of the thesis are no longer a very suitable option. The main change in requirements is that the identity provider and service must be able to be on separate networks, and not within the same secure

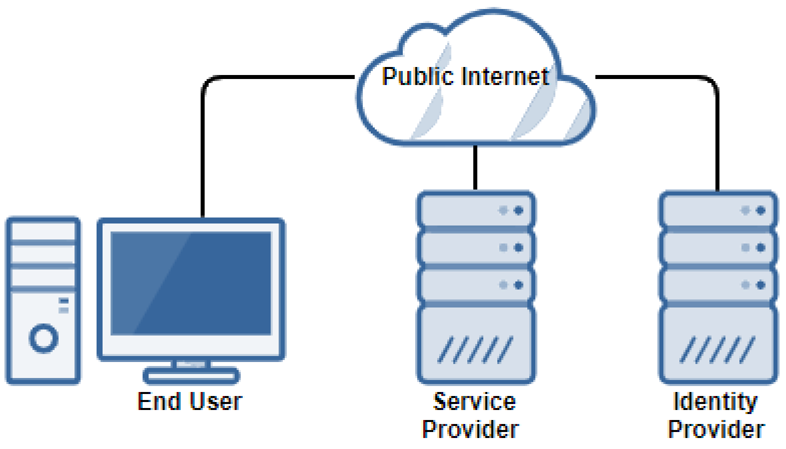

Figure 4.1: Network diagram of the use case.

network as it was specified when work began on this thesis.

OAuth 2.0 and OpenID Connect were chosen to be tested together to provide a complete authentication and authorization solution. The combination of the two frameworks has been heavily utilized by highly regarded corporations in the field of security, for instance Google and Microsoft [15]. The user experience is very streamlined with many unnecessary steps omitted. The requirements for the given use case is fully handled by the combination of the two frameworks.

SAML was chosen to be tested more thoroughly due to it fitting all of the requirements specified for the use case, as well as being a generally secure and heavily analyzed framework. SAML has seen much use in large corporations and universities for similar use cases, and has therefore been laboriously scrutinized and tested in public [32].

LDAP was not chosen as one of the frameworks to be integrated with Hibox Systems's solution because of the incompatibility with being used securely over the open Internet. Opening up an LDAP server directly to the public internet is not recommended, even if some encryption is used [49]. Other requirements, such as lack of standardized time-limited authentication, made LDAP a mediocre choice for the given use case.

Kerberos was deemed to be too complicated during the configuration phase and the user experience is not the most streamlined, at least if any errors are encountered.

Also, the maintenance required to keep a Kerberos system functional is not an easy task for an administrator. Kerberos was therefore regarded as a poor choice and more promising frameworks were chosen in the continued testing instead of Kerberos.

RADIUS is a framework that also was chosen to be omitted. RADIUS is built more for on-premise network authentication and authorization. To adapt it to an over the Internet single sign-on solution would not be following the use case which RADIUS was initially designed for and the security would be questionable. This is not to say that a secure solution would be impossible to achieve using this framework, but there are other frameworks, which have been more heavily scrutinized in these environments.

## 4.1 Criteria

The chosen frameworks are evaluated based on how reliable and efficient the frameworks are in practice. The frameworks are compared with each other based on theory and how well the integration of each of the frameworks functions.

The safety levels of each framework are compared with each other based on previous works and protocol descriptions. Most of the protocols describe ways to increase security by enabling extra security features, but this thesis focuses on recommended normal practices to ensure compatibility with as many implementations of the protocols as possible.

The reliability of the system is based on stability during an extended stress test. The stability during the stress test was measured by keeping track of the rate of erroneous responses from both the Hibox server and the identity provider server. The servers were monitored closely for any crashes or prolonged unresponsiveness, both on the identity provider and the Hibox server. The stress test featured different amounts of simultaneous emulated user agents from one to a thousand.

The efficiency of the solution is mainly measured by the actual throughput of users during emulated normal login flows. The successful throughput of users per second were recorded. The bytes sent and received per second by the user agents of the end users were also be documented and calculated. The ratio between bytes and throughput is expected to give some indication of how efficient the login flow is for the framework in question.

The CPU and RAM were monitored on both the identity provider server and the Hibox server during the time the simulated login flows are executed. The focus was on the difference between idle and testing scenarios. The relative increase during different loads is used to determine what framework is most suitable for what number of expected users.

## 4.2 Test setup

The test is conducted by taking SAML and OAuth 2.0 with OpenID Connect as an identity layer and integrating them with Hibox Systems's solution and running stress tests on the integrations. The stress tests involve emulating heavy load conditions by sending up to a thousand login requests within a short time frame. The successful throughput of test users, request timings and system resource usage is used to determine the efficiency of the system.

The test setup featured four entities, namely a router, an identity provider server, a server running Hibox Systems's solution and a computer that simulated all end-user browsers and their requests. Each of the entities is presented separately in this section of this thesis.

### 4.2.1 Network setup

The router functioned as a central hub which all requests had to pass through as can be seen in Figure 4.2. All connections to the router were through wired Ethernet connections to avoid variable wireless network delays and needless package losses due to wireless interference. Except for obligatory operating system requests that could not be disabled with reasonable efforts, no unnecessary network traffic was present at the time of the testing. The connection to the public Internet was disabled

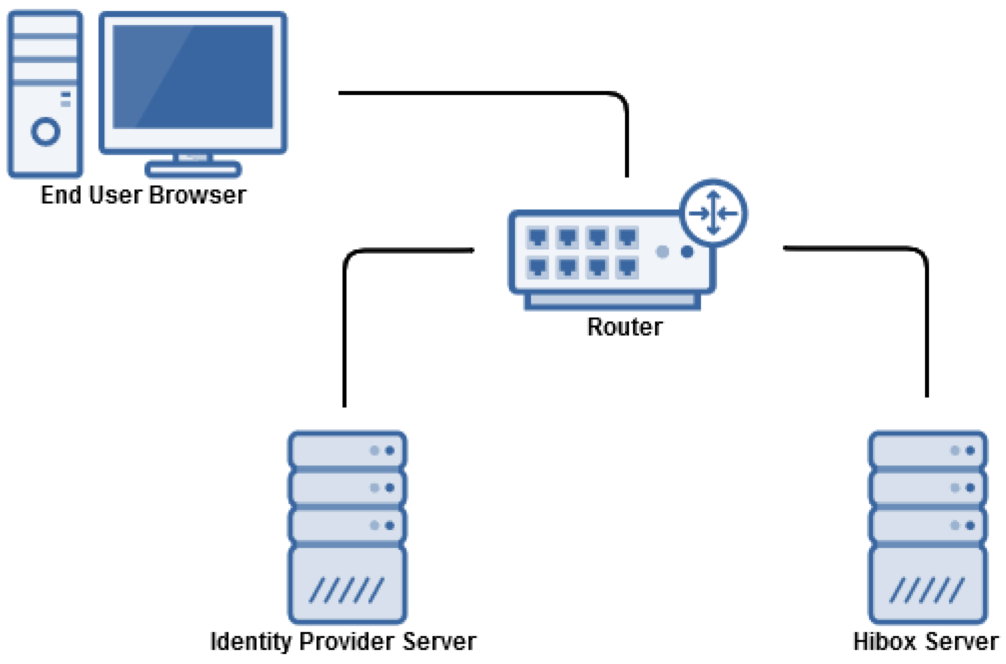

Figure 4.2: Network diagram of the test setup.

during testing, as it was not necessary for testing purposes.

### 4.2.2 Identity provider server

The identity provider server is a Gluu identity and access management server [50]. The server was chosen from all the available solutions because of its support for both SAML and OpenID Connect with only minor configuration needed.

The Gluu identity server was required to be run on a UNIX based system. The chosen distribution was Ubuntu 18.04 with GUI disabled as to have an as efficient system as possible. All unnecessary background services were disabled to free up as much of the system resources for the identity server.

There was some initial effort expended to assess how complicated it would be to implement an identity provider without using already openly available solutions. Different libraries were tested and implementations completely without using any libraries were considered. The advantages of these approaches would be to have full control of the login flows and possibilities to optimize them for the given use case.

The main disadvantage, that led to the using of an open-source available solution, was that the implementations would be of different quality levels. There is then an increased risk of comparing specific implementations, rather than the frameworks in question. The risk is not eradicated by using an open-source solution, but the chance that they are of similar quality is higher if they are bundled together in the same solution. The difference between the two frameworks when collecting system resource usage is also reduced if both can be run on the same system simultaneously and be connected to the same back end. The whole system then has the same idling memory usage and has the same storage for the registered users, which removes some variable factors from the test setup.

The identity provider server was configured to accept both SAML and OpenID Connect as valid login flows from the Hibox server. The Hibox server was added to a SAML trusted relationship by exchanging XML files containing metadata information about the SAML login flow configuration of both the identity provider server and Hibox server. The Hibox server was also added as an OpenID Connect client to the Gluu identity server. The configuration consisted of adding the OpenID Connect endpoints available on the Hibox server, as well as generating a client identifier and a password. The Authorization Code grant type was the only OAuth 2.0 grant type to be configured to verify that only secure Authorization Code requests are accepted by the system during the authentication flow following from OpenID Connect exchanges.

### 4.2.3 Hibox server

Hibox Systems's solution is represented with a separate server, which is called the Hibox server, connected to the test network as can be seen in Figure 4.2. The relevant implementation details of how the frameworks have been integrated with the Hibox System's solution is provided in Chapter 5.

The Hibox server was run with Ubuntu 18.04 [51] as the operating system due to the requirements of the underlying Hibox System's solution. The GUI, as well as unnecessary background services, was disabled to keep as much as possible of the system resources available for the authentication flow of the simulated users.

The Hibox server provided separate endpoints for both SAML and OpenID Connect login flows. Both endpoints were active during the whole testing process, as to not change the idle resource usage of the system when changing which of the frameworks were tested. Both of the frameworks provided flows which led to an authenticated session by the session handler of the underlying Hibox Systems's solution.

### 4.2.4 Load testing tool

The user agents of the end users are simulated with Apache JMeter, which is an open-source load testing tool that can be used to test and measure the performance of, mainly, web applications [52].

The JMeter application was run on a Windows 10 system. The application is built on Java, and is therefore completely portable to most operating systems. Windows 10 was chosen due to the availability of a computer with Windows 10 installed that was deemed to be powerful enough to prevent it from being a bottleneck in the different testing scenarios. The added convenience of having the application running on the same system that the data processing applications were run on led to a faster iteration process, and more tests could be run in a shorter time frame than would otherwise be possible.

JMeter was run without a GUI to make sure all available resources could be used by the sending and receiving of simulated authentication-related requests. The results were stored and then processed afterwards with the JMeter application itself, so relevant timing averages, error rates and throughput could be extracted from the saved request data. The data was then processed and organized to be visualized in Chapter 6.

Both the SAML and OpenID Connect login flows was recorded by sending all browser requests through a proxy provided by the JMeter application while using the browser to complete authentication flows for both frameworks. The requests are then replayed to simulate as many web browsers as needed to mimic a high load scenario.

Each simulated user agent has its headers and cache handled separately. Every new login exchange has an empty cache and has an unauthenticated session.

The simulated authentication flow does not execute website JavaScript and CSS styling, which could make the test differ from a real-world scenario, by changing the timing between requests and not fetching extra resources from the servers. There is also no delay between the user being able to input the login credentials and the providing of them, as this was concluded to not change the results other than offsetting the response times of the results.

## 4.3 Test scenarios

The chosen test scenarios to be executed are simulations of a number of users. Each simulation was to be simulated over a certain period of time to simulate a possible real-world scenario, where it is unlikely that too many users try to access the service at the exact same time. In addition, to consolidate the validity of the results, all of the tests would be run multiple times.

The JMeter application would mimic real-world scenarios by simulating a varied number of user agents. The number of user agents to simulate were chosen to be 1, 2, 5, 10, 20, 50, 100, 200, 500 and 1000. These numbers were chosen to provide various test scenarios where the systems would be tested thoroughly. The wide range of number of users also led to the logarithmic scale having to be used when visualizing the results, and roughly doubling every number each iteration leads to clearly explanatory graphs.

The requests simulating the user behaviours spread out over a set time span to emulate real-world scenarios. The time over which the requests were spread out is ten seconds. This was chosen to make sure that the integration would stay responsive during really short periods of intense usage and still not be unrealistic. It should be noted that, while the initial requests were sent over ten seconds, the responses were not time-limited and could continue execution until they were either successful or failed.

The test scenarios were all repeated to obtain more trustworthy results that have been calculated from the average of all test runs. All tests were run ten times, making a total of two hundred test runs, due to the ten different number of users to simulate and two different frameworks.

## 4.4 Performance metrics collection

The collection of the performance metrics of the different parts of the complete solution, as well as the monitoring of the made requests, are critical to determining the overall performance of the frameworks. The identity provider server and Hibox server had their system performance metrics gathered separately from each other, while the machine running the simulated end users compiled an overview, containing e.g. the request timings and error rate, of all generated requests.

### 4.4.1 Request metrics

The metrics of the request resulting from the authentication flows are recorded on the machine simulating the user agents. The raw requests were recorded by the JMeter application, which also then composed an overview.

The JMeter application compiled overview contained the throughput per second, average bytes sent and received, average request response time and request error rate. These were stored away after every test scenario to be examined and visualized.

### 4.4.2 Resource monitoring

The system resources on both the identity provider server and the Hibox server was monitored. Different available collection tools were tested, but in the end, the command line application "top" was used to record the state of the system at a set interval during the execution of the tests. This application was chosen because it was lightweight enough not to interfere unnecessarily with the results, as well as the ability to be able to visualize what resources were used by which program, including itself.

The system-wide memory usage of the servers was monitored during the tests. The RAM used by an idle system was recorded to see the increase that comes from different scenarios. The idle system RAM was recorded after the Java virtual machine on both the identity provider server and Hibox server had been running for a while, this to make sure the idle level of RAM is a good baseline to compare against when calculating the increase during the tests.

The CPU usage of the servers was monitored and stored to later be more closely examined. The CPU utilization of the identity provider server and Hibox server were recorded separately. The processor usage of the identity provider server was gathered by looking at the CPU usage of the system, and how much was used by the processes of the Gluu identity server. The Hibox server also had its CPU utilization recorded and examined. The processor usage of Hibox Systems's solution, along with its underlying database, was monitored using the same approach.

## 4.5 Evaluation of the experimental setups

One possible threat to the validity of the results in this thesis is the different implementations of the frameworks. This thesis intends to evaluate the frameworks in general and not caring about any specific implementation. The most simple implementations were the ones chosen to be tested, but the degree of efficiency of the chosen ones if they were to be compared to each other is unclear. Different implementations could be compared to each other to minimize the effect this may have on the results, but this would be an unreasonable effort within the scope of this thesis.

The collection of CPU utilization on the identity provider server and the Hibox server may also pose a threat. The CPU utilization is collected at certain intervals during the executions of the tests. The servers included in these tests had all their unnecessary background services shut down and only the CPU utilization of relevant processes is recorded, but there is always an inherent unpredictability in any multi-tasking operating system [53]. These reasons may all lead to inaccuracies in the results of the tests. This inaccuracy in CPU measurements could be minimized by running the tests repeatedly over a longer time span. All tests are run ten times to gather enough data for the results of the experiments to be accurate, while still being feasible to examine considering the extent of this thesis.

The measurement of the RAM utilization of the system may pose a threat. The RAM is collected at a set interval, which introduces the same possible lack of accuracy as for the CPU collection. In addition, both the identity provider server and the Hibox server make use of java, with only some relevant processes outside the Java virtual machine (JVM). The JVM garbage collector tries to predict what the memory footprint of the program will be in the future. The JVM garbage collector algorithm could be configured to reduce the effect it has on the results, but this is not feasible within the scope of this thesis [54].

Finally, networking introduces variable delays and occasional random errors that also may be a threat. The impact this has on the results by keeping a minimal number of entities connected to the network as illustrated in Figure 4.2. The delays present in a real-world scenario, where the requests are going over the Internet, and how that is shown on the resource usage on the entities may be very different from this local test scenario. In addition, the load put on any specific entity depends on how many requests the other entities manages to handle and forward. One way to minimize this threat and identify possible bottlenecks is to make sure only one of the servers are running at a time, with both the user agent and other server having their requests simulated. This thesis focuses on the performance of the system as a whole, so these threats are considered when the results of the tests are examined, but the impact on the results is, considering the scope of this thesis, not deemed to be large enough to warrant multiple tests where all parts except for one would be simulated.

# 5 Implementation

In this chapter, the integration is examined and the used platforms, solutions and libraries are presented. Finally, the authentication flow of the implemented solution is analyzed thoroughly.

The implementation described here is the integration of a centralized authentication and authorization solution with Hibox Systems's administration tool. The tool is almost exclusively written in Java, therefore, all integrations make use of Java libraries. The integrations are with SAML and OAuth 2.0, with OpenID Connect as an identity layer. The reasoning for choosing only these frameworks for the implementation is presented in the previous chapter. The identity provider server is, during the testing phase, the Gluu identity and access management server [50]. There are many possible solutions for the identity provider server, and the Gluu Server is chosen for the testing phase because of its support for both SAML and OpenID Connect. This should reduce the inherent differences present during the testing of the resulting implementations. Potential identity provider solutions will be discussed extensively with Hibox Systems before enabling the system on a customer server.

## 5.1 Platform

Most parts of the resulting system are written in java and can, therefore, in theory, be run on any operating system that can run the Java virtual machine (JVM). There are however some limiting factors of both the identity provider server and Hibox Systems's solution that led to a Unix based operating system.

The identity provider server is running the Gluu identity and access management server [50], which is mainly built of different Java parts, with Jetty as the Java HTTP server and Java servlet container. There are however many separate parts included in the Gluu Server, for instance the users are stored in a separate LDAP directory and the web view is separate from the API exposed by the system. All of these parts work together, and many assumptions are made by the system to make them run smoothly together, namely where to find different parts and how to communicate with them if they are running. These assumptions bind the Gluu Server to only run on Linux based operating systems. The Gluu Server was during testing, therefore, deployed on a system running Ubuntu 18.04.

Hibox System's solution is almost exclusively written in Java, with Tomcat as the Java web server environment. This means that it would certainly be possible to make it run on any operating system, but certain assumptions about the underlying system are made by the solution. These assumptions, such as the system paths to resources, lead to that the solution runs only on Unix based operating system, if one does not want to do extra configurations and troubleshooting. For the purpose of this thesis, Hibox System's solution will be deployed on a system with Ubuntu 18.04 as its operating system.

## 5.2 Integration overview

The implementation is done by using publicly available Java libraries to enable SAML and OpenID Connect, through OAuth 2.0, as valid methods of authentication. During the initial configuration, a trust relationship is established between the centralized identity provider server and Hibox Systems's solution.

The OAuth 2.0 Authorization Code authorization grant type was used for the OAuth 2.0 and OpenID Connect integration. The SAML solution was based on a Shibboleth server integrated with the Gluu authentication server solution. The frameworks are linked with the authentication system already present in the Hibox Systems's solution. The details of Hibox System's authentication are omitted for security reasons.

The OpenID Connect client was in the end implemented without using any previously available libraries due to the lack of solutions that were able to integrate with Hibox System's platform without requiring long dependency chains to be added to the system. One of the OpenID Connect implementations that were successfully integrated with Hibox System's authentication system, and that was most efficient during the authentication of one user, broke down if more simultaneous users were added, due to the fact that the same client was used to send all requests. This was the "Google OAuth Client Library for Java" [55], which was re-implemented without using the library, as it gave a completely offset perspective of the capabilities of the framework.

The integration had the OpenID Connect identity provider hardcoded, along with the shared key used during the final token exchange. The OpenID Connect client had the Authorization Code grant type as the only enabled flow to ensure that the secure code exchange flow was used. The same flow was also the only enabled one on the identity provider server.

The SAML client of the system is based on "Dead Simple SAML 2.0 Client" [56]. The library was integrated with Hibox System's solution by adding the necessary endpoints and handling the requests and responses with the functions included in the library. The SAML HTTP POST binding [57], one of the most common SAML bindings, was used during the execution of the flow.

The SAML client had the path to the metadata XML on the identity provider server hardcoded during the integration and test. The trust relationship was established between the SAML client and SAML identity provider by sharing the metadata files and registering the client on the identity provider server.

The most vital part of the authentication, for both of the frameworks, happens on the identity provider server. As long as the service provider can verify if the identity provider is the trusted party, the authentication can be relied on. The actual logic on the client-side is relatively simple, which makes the integration rather straightforward.

## 5.3 Authentication flows

The exact authentication flows used in the implementation have a massive impact on the results collected from the executions of the tests. Both SAML and OpenID Connect support a multitude of different possible flows. The most commonly recommended authentication flows were chosen for both of the different frameworks. The chosen flows were verified to be among the most secure available, while still being supported by most libraries and other solution. This to verify that the tests are representative of real-world scenarios.

In an effort to simulate normal end user authentication flows as closely as possible, the user is assumed to have authorized the identity server to let the service provider access necessary information prior to the initiation of the flow. In a real-world application, this happens only once during the first interaction with the service or through automatic authorization due to provider wide policy. In addition, the authentication flow assumes that the session of the end user is not already authenticated with the service provider or identity provider.

### 5.3.1 OpenID Connect flow

The implemented OpenID Connect authentication flow is visualized in Figure 5.1. The illustrated flow assumes that the end user's user agent does not have an already authenticated session with any of the servers in the system. The authentication flow follows the Authorization Code grant type flow of the OAuth 2.0 specification. Only the Authorization Code grant type was implemented during the integration of Hibox Systems's solution. The Authentication Code authentication flow was also the only enabled flow on the identity provider server to confirm that only that flow is accepted. This was done to guarantee that the secure back channel was used for the sensitive OAuth 2.0 access token.

The flow is initiated with an initial request to the desired service that has been configured as an OpenID Connect client with the identity provider. The example in Figure 5.1 shows an initial request to an admin page of the service provider. The

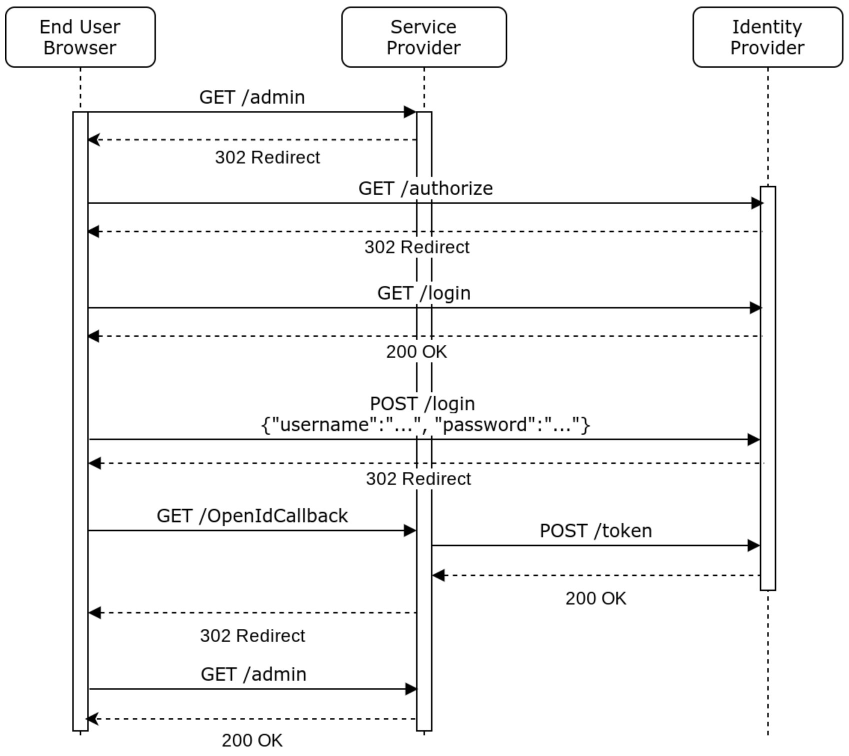

Figure 5.1: OpenID Connect authentication flow.

response on the initial request is a redirect to the identity provider, as long as the session of the user agent of the end user does not already have an authenticated session with the service provider.

The user agent is redirected to a previously configured authorization endpoint on the identity provider server. The request contains a set of parameters included in the query string appended to the request URL. These parameters are used to define the client id, callback URL, scopes, expected response type and a string representing the current state of the session. The client id is required to be configured with the identity provider and specifies which client will be allowed to request the access token if the authentication flow continues. The OpenID Connect callback URL is the URL to which the user will be redirected upon successful authentication with the identity provider. The callback URL is expected to be configured with the client with the given client id on the identity provider server. The provided scopes are required to contain "openid" to specify that the OpenID Connect flow is used. Other scopes may be included to expand what information is shared with the service provider. The response type is required to be "code" to initiate an Authorization Code flow, which is the only enabled flow by both the identity provider and service provider. The string representing the state can be used by the service provider to bring back the past state of the application when the user is returned, as well as to prevent cross-site request forgery attacks [58].

The end user would be redirected from the authorization endpoint to the OpenID Connect callback endpoint on the service provider if the user would have an active authenticated session on the identity provider. If the end user is not already authenticated with the identity provider, as it is assumed in the flow in Figure 5.1, then the user is redirected to the identity provider login page. The end user then has to provide a valid combination of username and password to verify his identity with the identity provider.

After the identity provider successfully authenticates the end user, the user is redirected to the OpenID Connect callback endpoint provided by the service provider, as long as the callback URL is verified with both the identity provider and service provider for the client in question. An authorization code is included in the query string of the redirect URL. The callback endpoint parses the authorization code and sends it to the identity provider token endpoint along with a shared secret, to verify the identity of the service provider. If the identity provider can verify the request, then it supplies the service provider with an access token and an id token.

The current session of the user agent of the end user is authenticated with the service provider if the id token can be successfully verified. The user can then access the service on behalf of the user identified in the id token. Finally, the end user is redirected to the initially requested resource.

### 5.3.2 SAML flow

The SAML authentication flow used in the implementation and tests is illustrated in Figure 5.2. The demonstrated authentication flow is a service provider-initiated

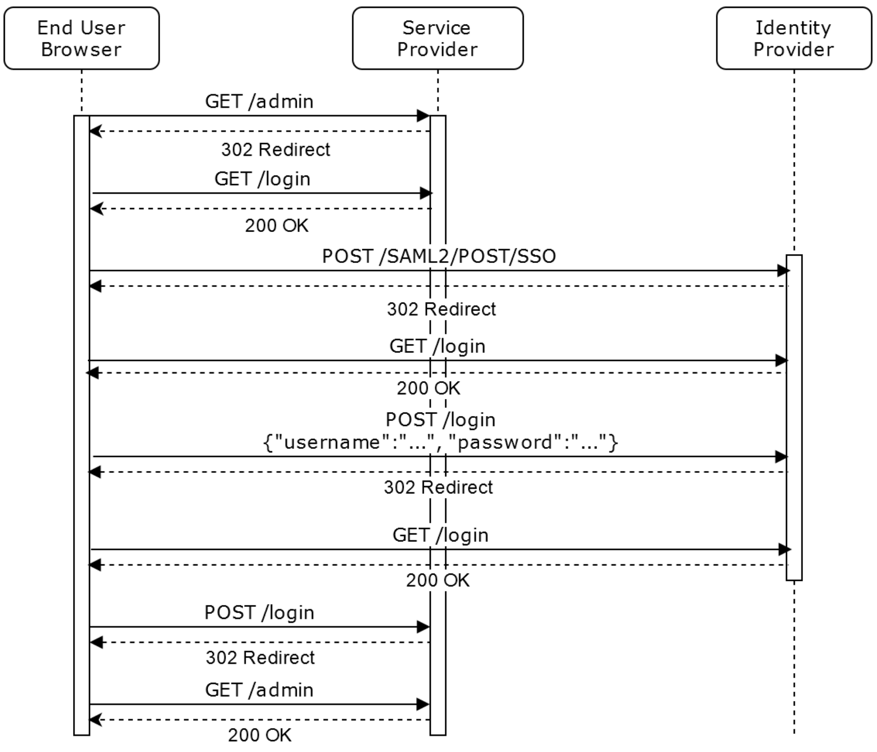

Figure 5.2: SAML authentication flow.

flow, which is triggered by the user agent of the end user accessing the service provider without an authenticated session [59]. The SAML metadata XML files are assumed to be shared between the service provider and identity provider prior to the initiation of the flow. Both the service provider and identity provider make use of a SAML HTTP POST binding when executing the SAML authentication flow.

The initial request is sent to the desired service, that has configured a SAML identity provider. The response contains a redirect to the login page on the service provider, as long as the user agent of the end user does not have an authenticated session with the service provider.

The SAML login page initiates a SAML request to the configured SAML single sign-on endpoint. The SAML authentication request sent to the identity provider contains information about the issuer, name policy and signature all unified and represented by a base64 encoded XML request. The issuer is described with a unique identifier specific to the identity provider to make sure the request is only handled by its intended target. The name policy describes in what format the SAML client expects the name identifier of the end user if the authentication succeeds. The signature and certificate included in the request can be used to verify the integrity of the entire request. The signature of the request will not be the same as the expected value if the request has been tampered with in any way. The certificates have been shared between the SAML identity provider and SAML client beforehand to establish a trust relationship.

Assuming the user agent is not already authenticated with the identity provider, the response redirects the end user to the login page of the identity provider as long as the SAML authentication request had its integrity successfully verified. The end user then has to provide valid credentials to the identity provider to then be redirected back to the service provider.

The identity provider sends back a SAML response, including a SAML assertion. The response is described with XML and base64 encoded. The information is sent in XML format to follow the SAML specification. Base64 encoding is used to ensure the correct characters are transmitted and the transported data can be received without modification [60].

The SAML assertion, included in the SAML response, contains the conditions and attributes of the successful authentication. The conditions of the authentication include restrictions such as the time window within which the authenticated session is valid, and the authentication will be invalid before and after the specified times. The assertion may also limit the audience for which the authentication is valid, meaning that only the specified service providers should accept the authentication. The attributes included in the assertion are the attributes that were requested in the authentication request. Only attributes that the user can access and the SAML identity provider has configured to be available are included. The attributes are in the name format specified in the initial request to the identity provider. The username or some other unique identifier of the authenticated user is a required attribute for the service provider to map the external user to an internal user on their system.

Apart from the SAML assertion, the SAML response also contains information about the SAML protocol, issuer, signature and status of authentication. The version of the SAML protocol that was used is specified at the beginning of the response. The identity provider includes an identifier of the issuer of the response to inform the service provider of which entity handled the request. The response, including the assertion, should not be trusted unless its signature can be validated. Information on the method used to sign the response, as well as the certificate of the identity provider may be included in the response. This information can then be used to verify the integrity of the entire response. The status of the authentication is included in the response. The response contains a "Success" status code if the credentials and the request could be verified. If the request was invalid in any way, a status code of "Requester" would be returned. In case of an error on the identity provider server, a response code of "Responder" would be returned. If the SAML entities are using different versions a "VersionMismatch" status code is returned. In addition to these general status codes, there might be a more descriptive status code included, such as "UnknownPrincipal" and "InvalidNameIDPolicy" [61].

Finally, the user agent is redirected back to the initially requested service. As long as the included assertion can be successfully verified by the service provider, the end user then has completed the SAML authentication flow and has a valid authenticated session with the service provider.

## 5.4 Challenges

There were different challenges during the implementation of the system. The challenges were due to the limitations following the requirements from Hibox Systems, as well as the large number of available frameworks.

Hibox Systems's solution used Gradle [62] as a build and dependency management tool, which caused some problems mostly due to unfamiliarity with the more advanced features of the tool. Gradle was configured in such a way that all dependencies had to be available either locally or on Hibox Systems's own artifact server. There was no success in adding the Maven repository as a fallback without having the build fail, and therefore all libraries, including all dependency chains, had to be downloaded for each protocol implementation that would be evaluated.

Included in the requirements of the implementation was to keep external library usage to a minimum to limit the number of artifacts that Hibox Systems are required to keep on their artifact server. There were some challenges during the finding of viable libraries. Simple libraries are often small because they depend on a multitude of other libraries, which made them not viable for the given requirements. Many possible libraries would also drag in different versions of dependencies already present in the system, which led to much confusion in Gradle.

One challenge was that is was difficult to find the most efficiently implemented versions of the protocols. The different implementations first had their code inspected in their respective repository to ensure that their implementation would suit the use case and requirements. The few most promising frameworks were then integrated with Hibox Systems's authentication system. Some were not successfully integrated and were left out at this stage. In the end, the best performing implementation was chosen for each of the frameworks chosen to be further investigated. Finally, a challenge was to ensure that the chosen frameworks were comparable during testing, so that the features of the frameworks were tested and not the specific implementation. There was no easy way to ensure this, but the most straightforward and efficient implementations were chosen. The different frameworks should not give a false view of the throughput and error rate just because of the design choices of the specific implementation.

# 6 RESULTS ANALYSIS

The results of the tests, simulations and implementations that are presented in this thesis are discussed and analyzed in this chapter. Conclusions are inferred from the collected data with a focus on how the results differ depending on which framework was used in the implementation.

Firstly, the implementation and the arisen problems are briefly examined. Later, all data collected during the tests are presented in graphs, which are then thoroughly analyzed. The frameworks are compared with each other based on the different defined metrics. The metrics that would impact the user experience directly, such as erroneous response rate and throughput, are weighted when drawing conclusions from the data.

## 6.1 Implementation overview

The implementation of the centralized authentication system was successful and a complete solution, with all necessary parts, was available in an offline network. Many open-source solutions and libraries were used to construct a working system that could be reliably tested. The choice of using available solutions as much as possible was based on trying to keep the quality of the implementations as close to those used in a real-world scenario as possible. Only the custom parts necessary to integrate SAML and OpenID Connect with the existing Hibox Systems's solution was implemented.

The OAuth 2.0 and OpenID Connect client/side and integration with Hibox Systems's solution was in the end implemented without using any OAuth 2.0 specific libraries. This due to many of the simple libraries available were either unsuited for the intended use case or not possible to configure without rewriting much of the logic that was deeply integrated into the libraries. One library that was simple to integrate worked well for a small number of users, but locked the process when more than one request had to be handled at the same time by separate threads. The limited amount of logic needed to implement an OpenID Connect with only the Authorization Code grant type functioning was deemed to be simple enough to be implemented, while still being able to attribute the results of the tests to the capabilities of the framework and not the specific implementation. The configuration of the Gluu identity provider server worked without any problems. The identity provider, following the clearly defined OpenID Connect protocol, exposed an endpoint containing all information needed to configure the client.

The SAML integration used available libraries to establish a working SAML client. The chosen libraries worked well with the Hibox System's solution. The configuration of the Gluu identity provider server was more complicated with the SAML solution than with the OpenID Connect one, though. The client XML metadata to be provided to the identity provider server had to be manually created. The identity provider error logs were not descriptive of what problems had occurred, which led to that many different versions of the metadata had to be provided until one was accepted by the SAML back end.

When comparing the two authentication flows with each other, it is clear that there are more SAML requests, and that the number of bytes transmitted by the requests is much larger than for the OpenID Connect flow. The OpenID Connect is clearly more lightweight in that sense. It should be noted during the evaluation that all of the SAML requests are executed by the user agent, while some requests are passed directly between the service provider and identity provider during the OpenID Connect flow.

## 6.2 Evaluation overview

The performance of the system in general and the metrics collected during the test are discussed here. All tests were repeated ten times with 1,2,5,10,20,50,100,200,500

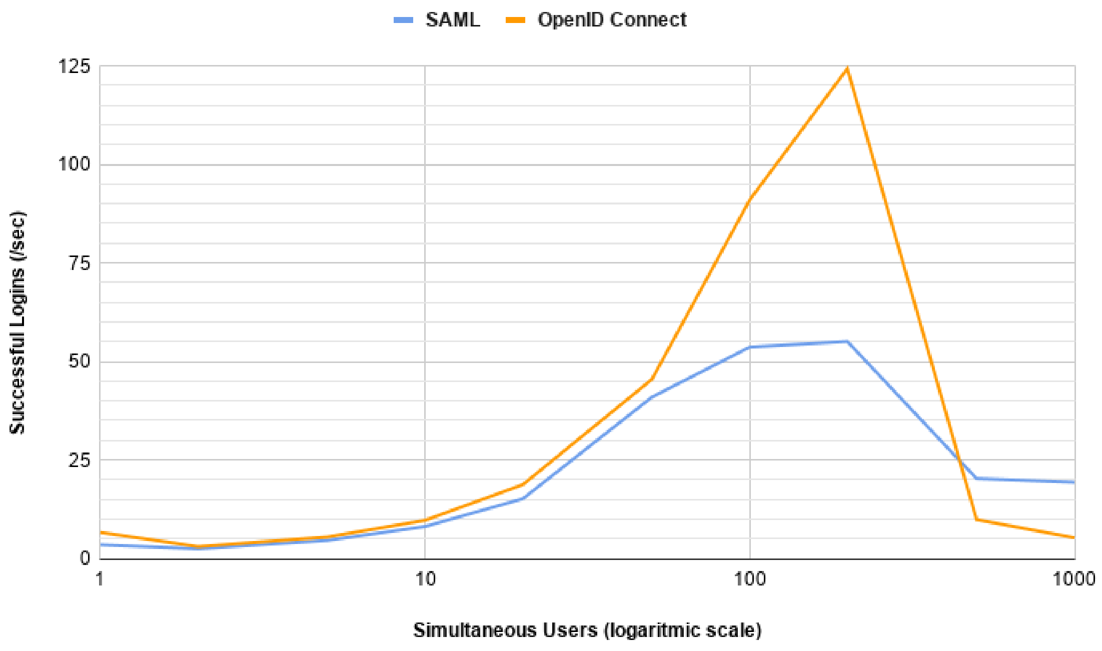

Figure 6.1: Throughput of users.

and 1000 simulated users for both SAML and OpenID Connect authentication flows. The initial login requests were sent over ten seconds, and run until all requests had been handled, successfully or not.

### 6.2.1 User throughput

The successful throughput of simulated users per second is visible in Figure 6.1. The throughput is very similar between the different frameworks for a smaller number of simulated users. Up until fifty simultaneous users, OpenID Connect has only a slight advantage over SAML. The advantage then rises from less than five users per second to almost 70 users per second at 200 users.

Both frameworks, with the given hardware, are struggling to handle more than 500 simultaneous users. This can be seen in Figure 6.1 where both solutions, especially the OpenID Connect one, have a drastic fall in the number of successfully authenticated users per second. For SAML, the throughput falls from 55 users per

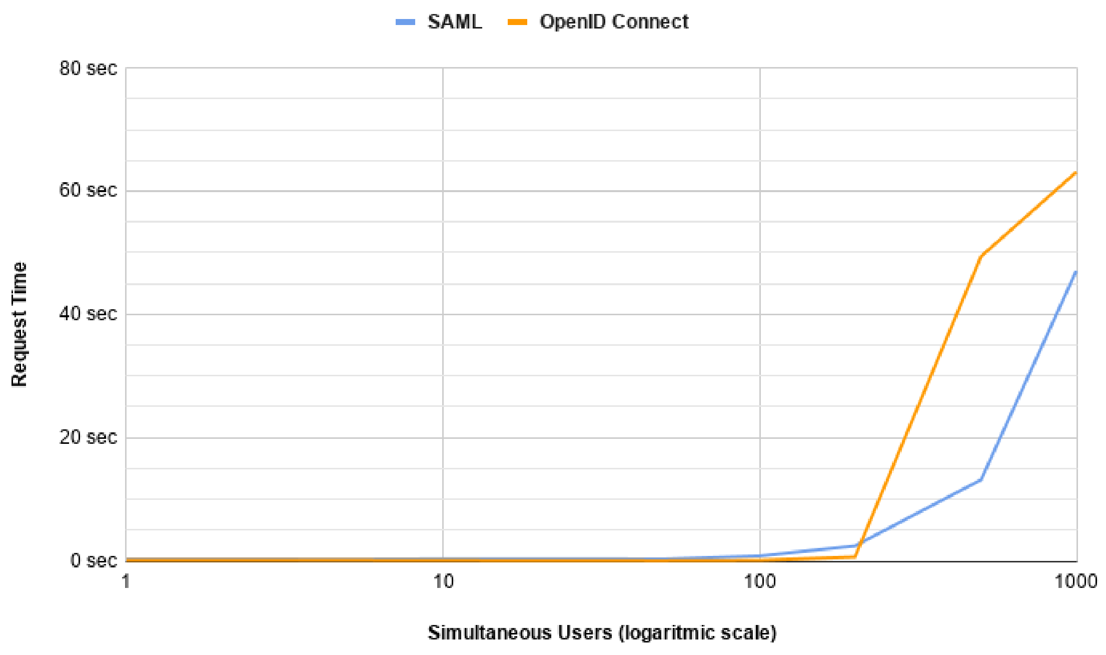

Figure 6.2: Average response times.

second to around 20 users per second. For OpenID Connect, the throughput falls from 124 users per second all the way down to less than 10 users per second.

The reason why the throughput is not at its highest with one simulated user may be because the servers in the system are most efficient when they get to process a bulk of users, as long as the server is not overburdened by the load of the users. This means that the initialization done by the first request becomes an insignificant part of the average throughput time when the number of users rises enough.

### 6.2.2 Response time

The response time of the different frameworks are visualized in Figure 6.2. The time to get a response on a request is kept on reasonable levels until around 100 users for SAML and 200 users for OpenID Connect. After that, there is a huge spike in average response time. The timings for simulated users authenticating using OpenID Connect spikes up higher than for users with SAML. OpenID Connect

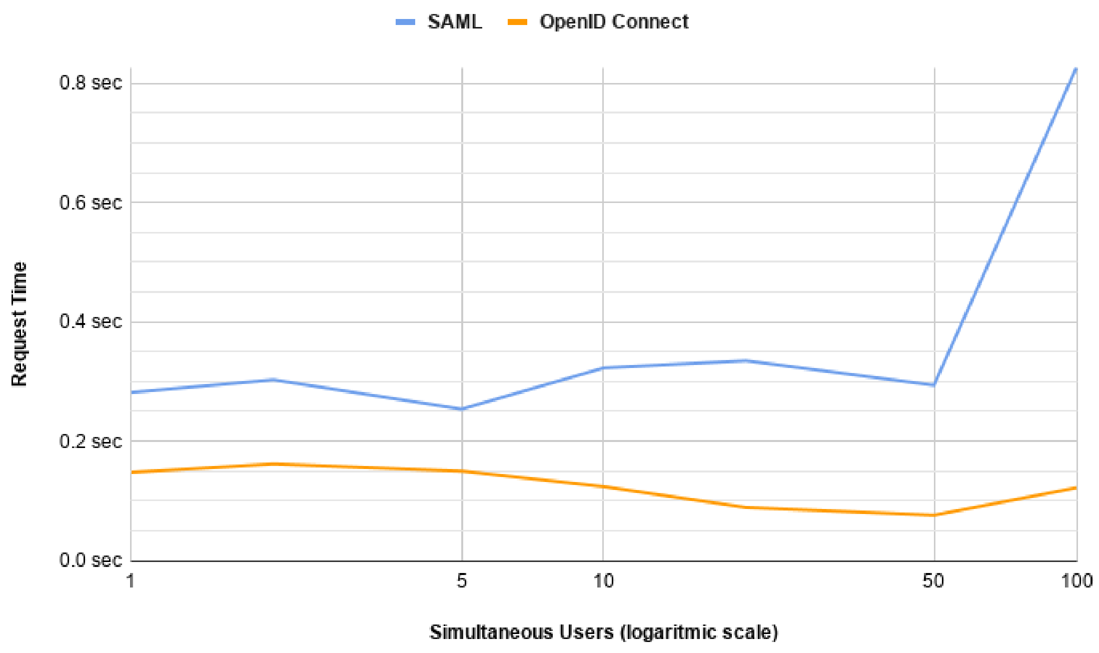

Figure 6.3: Average response times for up to 100 users.

reaches 49 seconds at 500 users and 63 seconds at 1000 users. Simulated users using SAML only reach 13 seconds at 500 users and 47 seconds at 1000 users.

The notable difference in the average response time for OpenID Connect compared to SAML may be due to the difference in implementations and where the bottleneck is in the system. The part of the system that seems to be the limiting factor in the case of the OpenID Connect flow is the Gluu Server, which has its CPU utilization rate rising to its maximum already at 100 simultaneous users as visualized in Figure 6.8. The SAML solution looks to be limited mostly by the Hibox server, as can be seen by its large CPU utilization in Figure 6.9, especially for 100 users or more.

In Figure 6.3 the response times are shown again, but only for simultaneous users up to 100. The responses times during this part of the test were not clearly visible in Figure 6.2 due to the sharp rise in the timings when the system got overwhelmed with users. In addition, the timings for less than 100 simultaneous users is the most representative of a generic use case, and must, therefore, be unambigu-

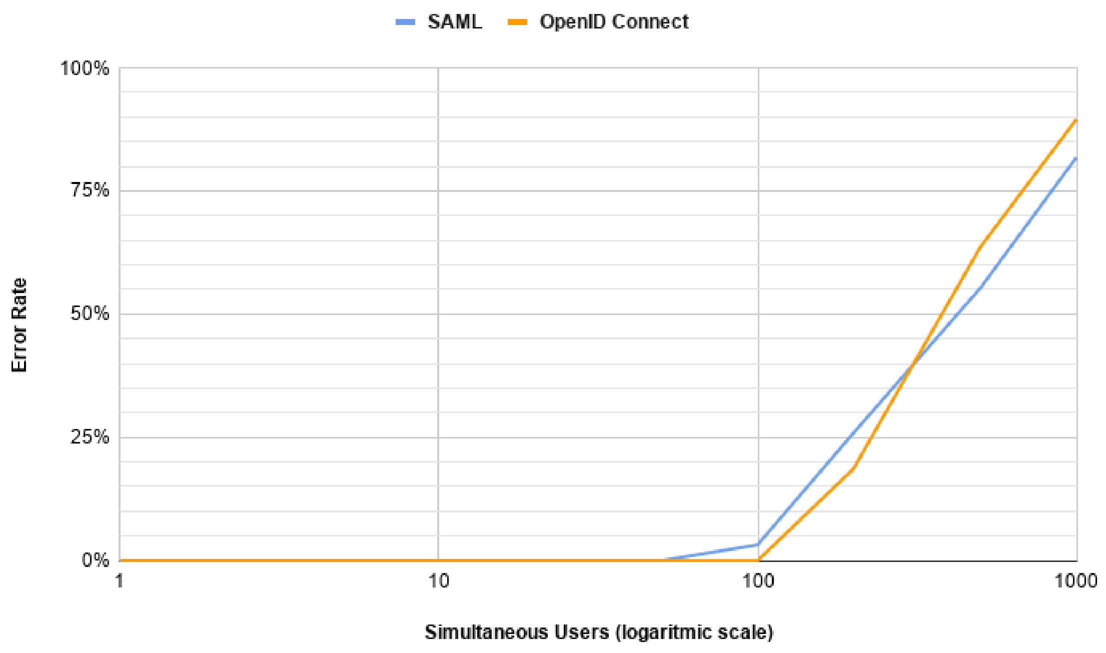

Figure 6.4: Error rate of authentication requests.

ously shown to be able to help draw accurate conclusions from the data.

The OpenID Connect had noticeably lower response times in the whole interval from one to 100 simulated users compared to SAML. The average response times for users using the OpenID Connect flow started at 150 ms at one user, which fell to 76 ms for 50 users and up to 120 ms for 100 users. The response times for the users using the SAML flow started at a higher 280 ms for one user and stayed around the same until it suddenly jumped from 290 ms at 50 users to 830 ms at 100 users.

The reason for the sudden increase response times in the case of SAML may be due to a bottleneck in the system. The bottleneck is most likely to be in the Hibox server, which has an almost 100% CPU utilization as can be seen in Figure 6.9 at 100 simultaneous simulated users.

### 6.2.3 Error rate

The error rate of authentication requests can be a good indicator for when a system is under too much load and the recorded error rate for the different number of simulated users is shown in Figure 6.4. The error rate should preferably stay at 0% and a huge number of simultaneous users should only experience a rise in the response time from the system. In reality, any system needs to handle errors, but keeping the error rate low stops the system from having to handle retry requests. The implementation during the test did not send any retry requests on failure.

The error rate is kept at 0% for up to 50 simulated users authenticating with SAML and for up to 100 users using the OpenID Connect flow. The error rate goes up to 3% for SAML at 100 simulated users. At 200 users the rate was 19% for OpenID Connect and 26% for SAML. The framework with the higher error rate then swaps to OpenID Connect. For 500 to 1000 users the error rate of OpenID Connect rises from 64% to 90% and the error rate of SAML goes from 55% to 82%.

### 6.2.4 Data rate

The average rate of sent and received data by the simulated users can be seen in Figure 6.5. The rate of transmitted bytes is an indicator of the efficiency of the system and can then be compared to the total throughput of users. It can be noted that some of the requests are directly between the service provider and identity provider during the OpenID Connect flow, as opposed to the SAML flow where all requests are passed through the user agent.

The data rates increase with the same slope for both frameworks up until 100 users, with SAML having a small edge. For OpenID Connect the rate continues to rise until it reaches 200 users. The rate for SAML goes down slightly in the interval between 100 and 200 users, but both the rate at which bytes are sent and received is still higher than for OpenID Connect.

The transmission rate falls sharper for OpenID Connect compared to SAML

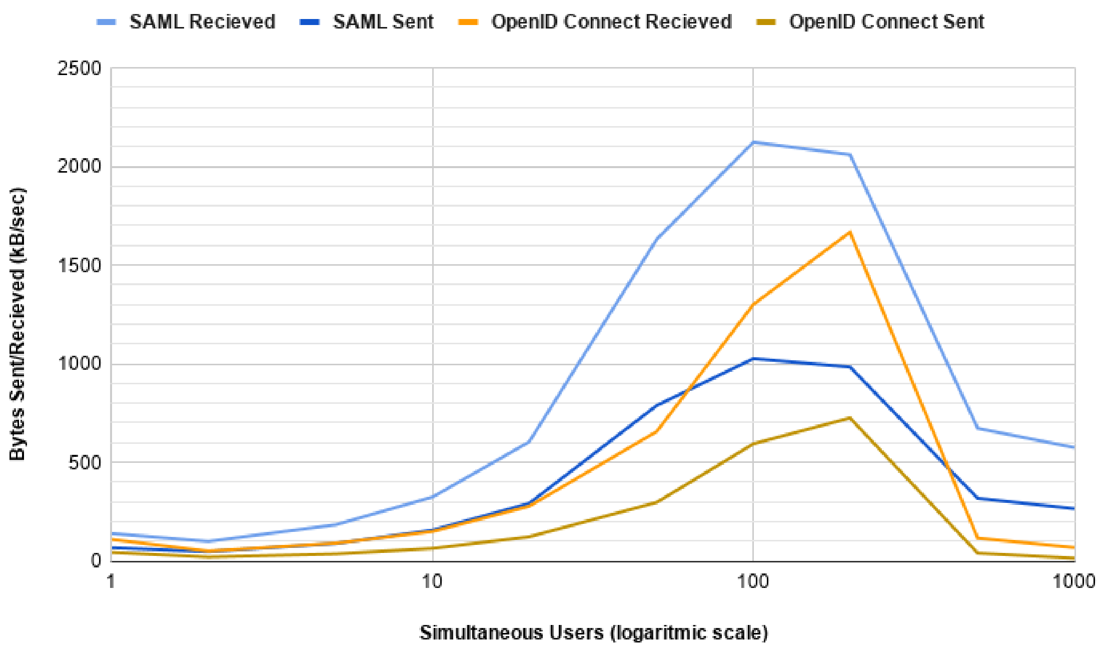

Figure 6.5: Bytes transmitted per second for authentication requests.

when testing 200 users compared to 500 users. The rate OpenID Connect users receive bytes falls from 1670 KB per second to 116 KB per second and the rate bytes are sent falls from 725 KB per second to 40 KB per second. The rate SAML users receive bytes falls from 2060 KB per second to 670KB per second and the rate bytes are sent falls from 980 KB per second to 320 KB per second. The fall of the transmission rate indicates how severely the systems are congested by a rising number of simulated users depending on the framework used.

The average data transmission rate during the OpenID Connect authentication flow is lower on average than during the SAML flow. This could mean that the system is more efficient when handling the SAML flow, but it has to be compared to the relative throughput of users. When comparing the average data rate, as visualized in Figure 6.5, to the rate of successfully authenticated users shown in Figure 6.1, it can be inferred that the data rate is higher for SAML, while OpenID Connect has a higher rate of successfully authenticated users. This means that SAML requires more bytes per successfully authenticated user relative to OpenID Connect.

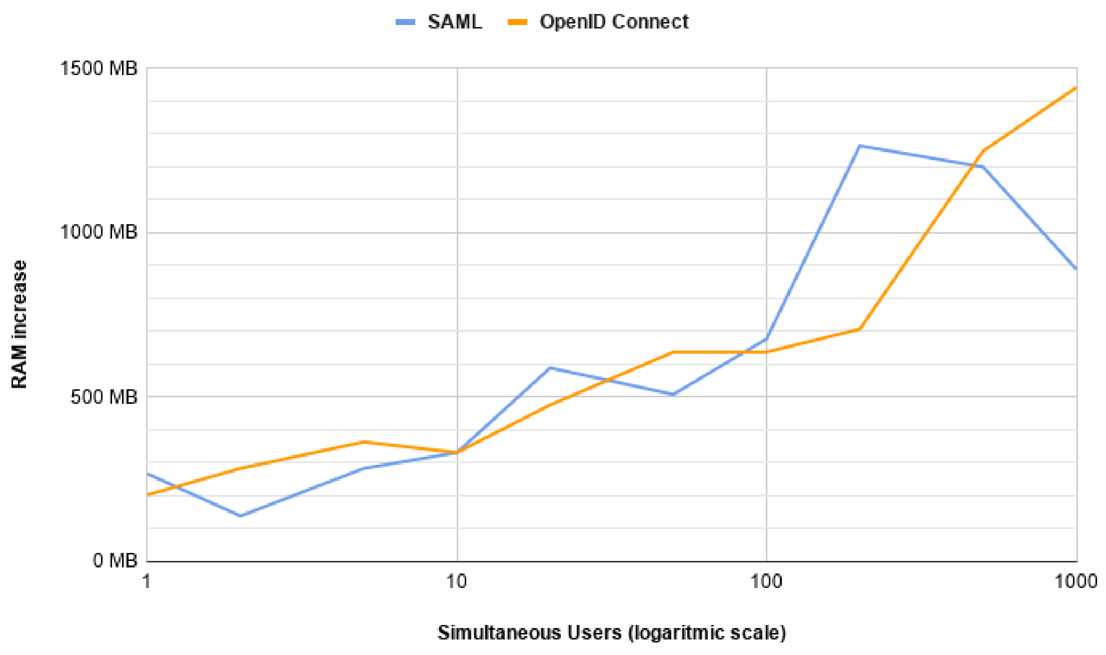

Figure 6.6: RAM increase on the Gluu identity server relative to a baseline.

For instance, at 200 simultaneous users, 1670 kB per second is received by OpenID Connect users while 124 users are authenticated per second. For SAML users 55 users per second are authenticated and 2060 kB is received per second at the same number of users.

### 6.2.5 RAM increase

The Random-access memory (RAM) usage increase on the Gluu identity server relative to a baseline is visualized in Figure 6.6. The baseline is from a system that has been idle after successfully handling 100 logins successfully to get the JVM ready and optimized for handling the upcoming requests, as it would be in a normal real-world use scenario.

The RAM increase is rising at about the same rate over the whole span of different numbers of users for both frameworks. There is an increase of around 300 MB for one to ten simulated users, with the SAML framework being more efficient

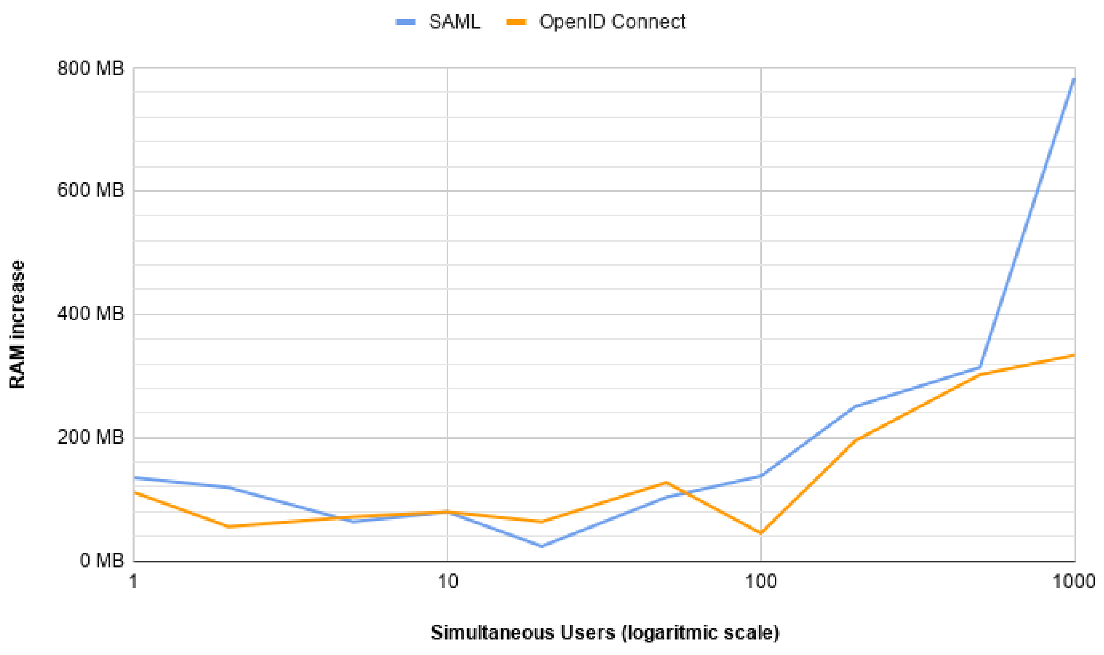

Figure 6.7: RAM increase on the Hibox server relative to a baseline.

by a small, but not insignificant, amount. From ten to 100 users, the increase in RAM rises at very similar rates for both frameworks. SAML reaches the peak of 1260 MB RAM usage already at 200 simulated users, and then starts to drop until it is down to 890 MB at 1000 users. The RAM increase during the OpenID Connect flows is rising at a constant rate until it reaches 710 MB at 200 users, and then it rises sharply to 1440 MB at 1000 users.

The RAM usage increase of the Hibox server is visible in Figure 6.7. The baseline to which the increase is compared to was decided by using the same method as with the Gluu Server. The system was allowed to idle and stabilize after 200 users had been handled.

The increase is minimal for both frameworks with the RAM increase lying at around 100 MB until the number of simultaneous users rises to 100. The average increase of the OpenID Connect in the interval from one to 100 users was 79 MB, while the SAML framework had an average of 95 MB in that same interval of users. Both SAML and OpenID Connect see a similar rise from 100 to 500 simulated users

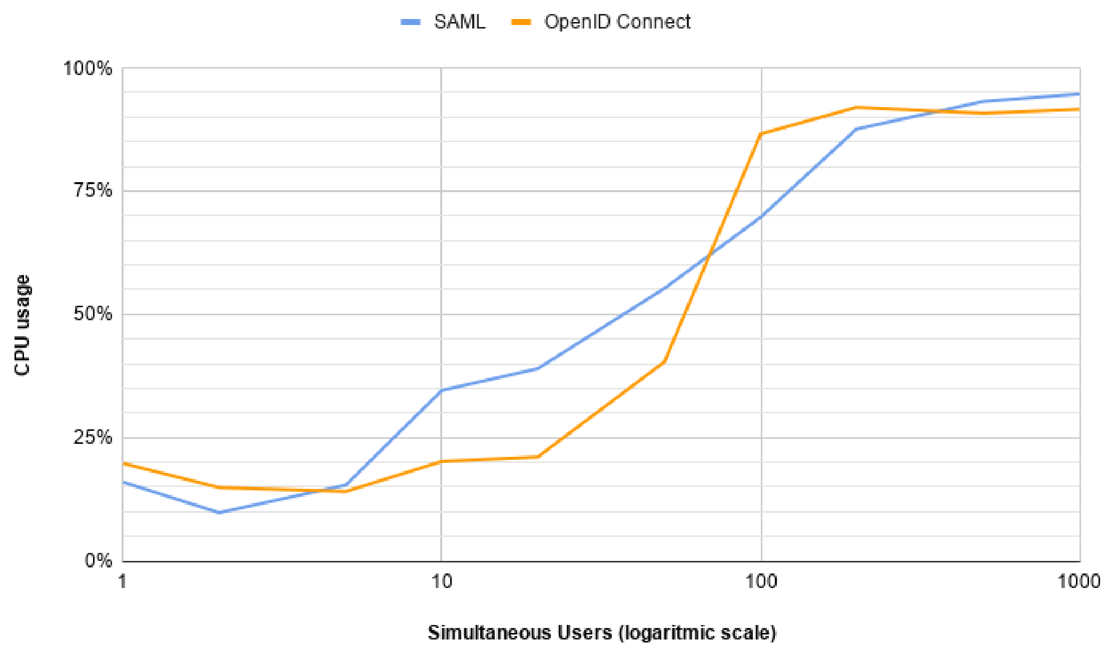

Figure 6.8: CPU utilization of the Gluu identity server.

to around 300 MB. The RAM usage increase of the OpenID Connect clients then rises to only 330 MB at 1000 users, while the SAML framework sees an increase all the way to 780 MB.

It can be noted that for users following the SAML flow, in Figure 6.6 the RAM usage decreases between 500 to 1000 users, perhaps due to patterns detected by the JVM garbage collector. The RAM usage increases in Figure 6.7 in the same interval.

### 6.2.6 CPU utilization

The average CPU usage of the Gluu identity server during the test execution is shown in Figure 6.8. The CPU usage on the identity provider during the testing consisted mostly of the Java virtual machine, as well as some use by the Apache server which provided some requested resources.

The identity provider CPU usage difference between simulated SAML and OpenID

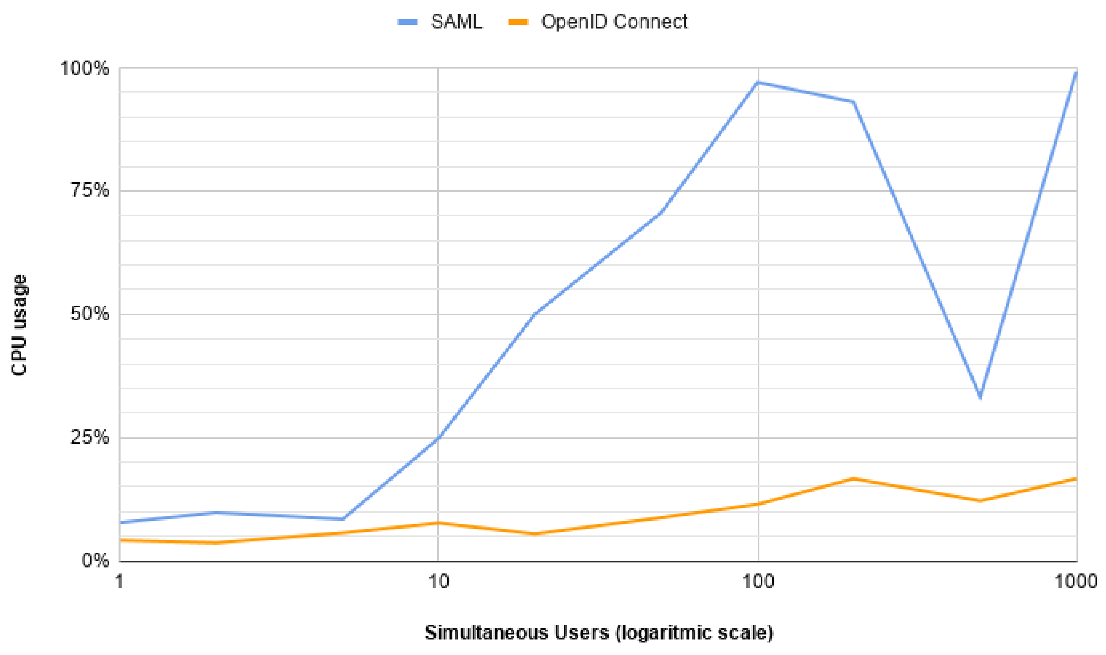

Figure 6.9: CPU utilization of the Hibox server.

Connect is minimal from one to five users, with SAML being slightly better. From five to fifty simultaneously simulated users OpenID Connect is noticeably lighter on resources, with around 15% less total CPU usage compared to SAML. Both frameworks are struggling to handle more than 200 simultaneous users and stay around 90-95% CPU utilization, but OpenID Connect reaches this level already at around 100 users. The percentage not quite reaching 100% may be due to the implementation of the Gluu Server not being able to handle the users in an optimal multi-threaded fashion, meaning that one crucial handler is fully utilizing its available core, and can not successfully start new threads faster than requests are coming in.

The processing power used on the Hibox server was split between the complete Java tomcat system and the back end database. The database was MariaDB, which is a MySQL community fork [63].

The average CPU utilization of the Hibox server during the testing phase can be seen in Figure 6.9. What is clear is that the CPU usage, in general, is less for the server when it is handling OpenID Connect users, rather than SAML users. The difference is not too large between the two frameworks for under ten users, but the resource usage then rises to close to 100% utilization for when handling more than 100 SAML users. This, while the CPU usage is only slowly rising from 5% to 15% when going from one to 1000 simulated OpenID Connect users.

The sharp reduction in CPU utilization for when 500 SAML users were processed at the same time can be attributed to congestion on the identity server after the initial request is sent to the identity server, so that the Hibox server has more time to process each user. Why the percentage goes back to 100% when reaching 1000 almost simultaneous users may be because the Hibox server now fails to send the initial request to the identity provider server due to the Hibox server now being under too much load to successfully redirect all of the initial login requests.

## 6.3 Summary

Because the intended use case of the Hibox System's solution is for a small number of users, realistically under fifty, the focus will be given to the results of the framework in the interval from one to fifty simultaneous users. In practice, often no more than five users will have to be handled simultaneously, but crucial systems should be able to handle the realistic worst-case scenario.

The OpenID Connect seems to give an overall smoother experience for the end user at a small number of users, compared to SAML. The most notable metrics of this are the results from the response timing and user throughput tests, where OpenID Connect was clearly more efficient than SAML. Error rate and overall resource usage were overall very similar for both frameworks, at least for a smaller number of simultaneous users.

# 7 FUTURE WORK AND CONCLUSIONS

In this thesis, different authentication and authorization protocols and frameworks have been compared with each other. They have been evaluated on the basis of what framework or protocol would be most suited for a single sign-on server with a limited number of users and fairly strict security requirements. The frameworks chosen to be examined in this thesis were OAuth 2.0, OpenID Connect, SAML, LDAP, Kerberos and RADIUS. These frameworks were selected based on their heavily scrutinized security features and general widespread use.

The frameworks were initially evaluated solely on a theoretical level with a focus on accessibility and user-friendliness. Furthermore, the complexity of the configuration and integration with another system was investigated.

## 7.1 Future work

This thesis has focused on a small subset of available frameworks and, while these have been chosen with great consideration, some other viable omitted frameworks could be compared to the chosen ones. Also, the testing of prototypes with end users has been very limited and could certainly be expanded upon.

The implementations and integration done as a part of this thesis are not ready to be deployed in a real production environment, but following how the system functions under such circumstances would provide more reliable data for the comparison between the different frameworks. The lack of such data led to a focus on theory and simulations to gain an understanding of how the frameworks would most likely function in such an environment.

The threats to the validity of the conclusions presented in this thesis could be minimized by running the test more times and with a variety of implementations to ensure that the capabilities of the frameworks are being tested and measured instead of specific implementations. Also, different methods of metrics collection could be evaluated to determine what method would be the most accurate, as well as the precision of the measurements.

## 7.2 Conclusions

Out of the chosen frameworks, OAuth 2.0 with OpenID Connect as an identity layer was the most suited for the problem at hand. Both Kerberos and RADIUS proved to be too complex for both the users and administrators, and are relatively heavy on resources due to the many included advanced features. SAML and LDAP were quite straightforward for the user, but the setup and configuration were quite cumbersome for the administrator. SAML also included all necessary identity provider features. OAuth 2.0 was very user-friendly, as well as easy to integrate with available solutions. The OpenID Connect identity layer was simple to add on top of OAuth 2.0 to make it a standardized authentication solution and not only usable for authorization. The lightweight resulting system had great usability, features and security that were meeting all of the requirements.

Many advantages come with using a single sign-on server. It leads to a more secure and manageable system for both the users of the system and the administrators, mostly because of fewer passwords for the user to memorize and a smaller number of accounts for the administrators to manage. There are many different ways to implement such a solution, but for Hibox Systems's use case, OAuth 2.0 and OpenID Connect are together the best-suited frameworks out of the chosen ones.

# BIBLIOGRAPHY

[1] W. Stewart. Internet history – one page summary, Jan 2000. URL: https://www.livinginternet.com/i/ii_summary.htm.

[2] Barry M Leiner, Vinton G Cerf, David D Clark, Robert E Kahn, Leonard Kleinrock, Daniel C Lynch, Jon Postel, Lawrence G Roberts, and Stephen S Wolff. The past and future history of the internet. Communications of the ACM, 40(2):102–108, 1997.

[3] Nevena Vratonjic, Julien Freudiger, Vincent Bindschaedler, and Jean Hubaux. The inconvenient truth about web certificates. In Economics of information security and privacy iii, pages 79–117. 2013.

[4] Richard Duncan. An overview of different authentication methods and protocols. SANS Institute, 2001.

[5] G. P. Koslovski M. A. Pillon N. M. Gonzalez G. C. Batista, C. C. Miers and M. A. Simplicio. Using externals idps on openstack: A security analysis of openid connect, facebook connect, and openstack authentication. In 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA), pages 920–927, 2018.

[6] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. Systematic design of two-party authentication protocols. In Annual International Cryptology Conference, pages 44–61. Springer, 1991.

[7] Bruce Schneier. Two-factor authentication: too little, too late. Communications of the ACM, 48(4):136, 2005.

[8] Jiang Qian, Susan Hinrichs, and Klara Nahrstedt. Acla: A framework for access control list (acl) analysis and optimization. In Communications and Multimedia Security Issues of the New Century, pages 197–211. 2001.

[9] John Barkley. Comparing simple role based access control models and access control lists. In Proceedings of the second ACM workshop on Role, pages 127–132, 1997.

[10] Nitin Naik and Paul Jenkins. Securing digital identities in the cloud by selecting an apposite federated identity management from saml, oauth and openid connect. In 2017 11th International Conference on Research Challenges in Information Science (RCIS), pages 163–174. IEEE, 2017.

[11] Masakazu Ohashi, Nat Sakimura, Tatsuki Sakushima, and Mayumi Hori. On the substantiative experiment study of proxing assurance between openid and saml: Technical perspective for private information box project of japanese e-government. In International Conference on ENTERprise Information Systems, pages 381–390. Springer, 2010.

[12] Richard Duncan. An overview of different authentication methods and protocols, Oct 2001. URL: https://www.sans.org/reading-room/whitepapers/authentication/paper/118.

[13] Deb Shinder. Understanding and selecting authentication methods, Jul 2015. URL: https://www.techrepublic.com/article/understanding-and-selecting-authentication-methods/.

[14] Whitson Gordon. Understanding oauth: What happens when you log into a site with google, twitter, or facebook, Jun 2013. URL: https://lifehacker.com/understanding-oauth-what-happens-when-you-log-into-a-s-5918086.

[15] Eric Eldon. Single sign-on service openid getting more usage, Dec 2019. URL: https://venturebeat.com/2009/04/14/single-sign-on-service-openid-getting-more-usage/.

[16] Ryan Squires, Alex Moiseev, Charlie J. Powers, Michael Vizard, Richi Jennings, and Ajay Kumar. When to use ldap, Mar 2019. URL: https://securityboulevard.com/2019/03/when-to-use-ldap/.

[17] Josh Fruhlinger. What is saml? how it works and how it enables sso, Oct 2017. URL: https://www.csoonline.com/article/3232355/what-is-saml-how-it-works-and-how-it-enables-single-sign-on.html.

[18] D. Hardt. The oauth 2.0 authorization framework. RFC 6749, RFC Editor, Oct 2012. URL: http://www.rfc-editor.org/rfc/rfc6749.txt.

[19] A. Parecki. End user authentication with oauth 2.0, 2020. URL: https://oauth.net/articles/authentication/.

[20] Torsten Lodderstedt, John Bradley, Andrey Labunets, and Daniel Fett. Oauth 2.0 security best current practice, May 2018.

[21] A. Parecki D. Hardt and T. Lodderstedt. The oauth 2.1 authorization framework draft. Technical report, Mar 2020. URL: https://tools.ietf.org/html/draft-parecki-oauth-v2-1-01.

[22] R. Davis. Oauth 2.0 grant types, Mar 2020. URL: https://docs.pivotal.io/p-identity/1-11/grant-types.html.

[23] W. Denniss, J. Bradley, M. Jones, and H. Tschofenig. Oauth 2.0 device authorization grant. RFC 8628, RFC Editor, Aug 2019.

[24] Damian Rusinek. What is going on with oauth 2.0? and why you should not use it for authentication., Jan 2019. URL: https://medium.com/securing/what-is-going-on-with-oauth-2-0-and-why-you-should-not-use-it-for-authentication-5f475

[25] Using oauth 2.0 for web server applications, Apr 2020. URL: https://developers.google.com/identity/protocols/oauth2/web-server.

[26] Nat Sakimura, John Bradley, Mike Jones, Breno De Medeiros, and Chuck Mortimore. Openid connect core 1.0 incorporating errata set 1. The OpenID Foundation, specification, 335, 2014.

[27] The OpenID Foundation. Openid connect faq and q&as, 2020. URL: https://openid.net/connect/faq/.

[28] What is openid connect?, 2020. URL: https://www.okta.com/openid-connect/.

[29] John Hughes and Eve Maler. Security assertion markup language (saml) v2. 0 technical overview. OASIS SSTC Working Draft sstc, pages 29–38, 2005.

[30] The Shibboleth Consortium. Members - the shibboleth consortium, 2020. URL: https://www.shibboleth.net/consortium/.

[31] P. Otemuyiwa. How saml authentication works, Dec 2016. URL: https://auth0.com/blog/how-saml-authentication-works/.

[32] Gro. Security analysis of the saml single sign-on browser/artifact profile. In 19th Annual Computer Security Applications Conference, 2003. Proceedings., pages 298–307. IEEE, 2003.

[33] J. Sermersheim. Lightweight directory access protocol (ldap): The protocol. RFC 4511, RFC Editor, Jun 2006. URL: http://www.rfc-editor.org/rfc/rfc4511.txt.

[34] A. Melnikov and K. Zeilenga. Simple authentication and security layer (sasl). RFC 4422, RFC Editor, Jun 2006. URL: http://www.rfc-editor.org/rfc/rfc4422.txt.

[35] Why choose ldap?, May 2018. URL: https://ldap.com/why-choose-ldap/.

[36] Directory services (ldap), Aug 2000. URL: https://docs.oracle.com/cd/A87860_01/doc/ois.817/a83729/adois09.htm.

[37] Z. DeMeyer. The difference between ldap and saml sso, Apr 2019. URL: https://jumpcloud.com/blog/difference-ldap-saml-sso.

[38] Jennifer G Steiner, B Clifford Neuman, and Jeffrey I Schiller. Kerberos: An authentication service for open network systems. In Usenix Winter, pages 191–202. Citeseer, Mar 1988.

[39] B Clifford Neuman and Ts. Kerberos: An authentication service for computer networks. IEEE Communications magazine, 32(9):33–38, 1994.

[40] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The kerberos network authentication service (v5). RFC 4120, RFC Editor, Jul 2005. URL: http://www.rfc-editor.org/rfc/rfc4120.txt.

[41] Kerberos: The network authentication protocol, Apr 2020. URL: https://web.mit.edu/kerberos/.

[42] O. Geir. Kerberos authentication 101: Understanding the essentials of the kerberos security protocol, Jan 2012. URL: https://redmondmag.com/articles/2012/02/01/understanding-the-essentials-of-the-kerberos-protocol.aspx.

[43] Red Hat Linux. Why not use kerberos?, 2000. URL: https://www.linux.co.cr/distributions/review/2000/red-hat-7.0/rhl-rg/s1-kerberos-whynot.html.

[44] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote authentication dial in user service (radius). RFC 2865, RFC Editor, Jun 2000. URL: http://www.rfc-editor.org/rfc/rfc2865.txt.

[45] Freeradius documentation. URL: https://networkradius.com/doc/3.0.10/concepts/introduction/databases.html.

[46] Joshua Hill. An analysis of the radius authentication protocol. InfoGard Laboratories, Nov 2001. URL: http://lms.uni-mb.si/.

[47] S. Winter, M. McCauley, S. Venaas, and K. Wierenga. Transport layer security (tls) encryption for radius. RFC 6614, RFC Editor, May 2012. URL: https://tools.ietf.org/html/rfc6614.

[48] eduroam. How does eduroam work?, 2020. URL: https://www.eduroam.org/how/.

[49] Andrew Findlay. Best practices in ldap security, 2011.

[50] GluuFederation. Gluu server 4.1 documentation, 2020. URL: https://gluu.org/docs/gluu-server/.

[51] Canonical. The leading operating system for pcs, iot devices, servers and the cloud, 2020. URL: https://ubuntu.com/.

[52] Apache. Apache jmeter, 2020. URL: https://jmeter.apache.org/.

[53] Mitchell S Fletcher and Richard P Semma. Operating system for a multi-tasking operating environment, 1991.

[54] Guangyu Chen, R Shetty, Mahmut Kandemir, Narayanan Vijaykrishnan, Mary Jane Irwin, and Mario Wolczko. Tuning garbage collection in an embedded java environment. In Proceedings Eighth International Symposium on High Performance Computer Architecture, pages 92–103. IEEE, 2002.

[55] Googleapis. Google oauth client library for java, Apr 2020. URL: https://github.com/googleapis/google-oauth-java-client.

[56] Coveooss. Dead simple saml 2.0 client, Dec 2019. URL: https://github.com/coveooss/saml-client.

[57] Rob Philpott, Nick Ragouzis, Thomas Wisniewski, Entrust Greg Whitehead, HP Heather Hinton, Connor P Cahill, John Bradley, Individual Jeff Hodges, Individual Joni Brennan, Liberty Alliance, et al. Bindings for the oasis security assertion markup language (saml) v2. 0–errata composite. Sep 2015. URL: https://www.oasis-open.org/committees/download.php/56779/sstc-saml-bindings-errata-2.0-wd-06.pdf.

[58] Auth0. State parameter, 2020. URL: https://auth0.com/docs/protocols/oauth2/oauth-state.

[59] Understanding saml, 2020. URL: https://developer.okta.com/docs/concepts/saml/.

[60] Base64 - mdn web docs glossary, Feb 2020. URL: https://developer.mozilla.org/en-US/docs/Glossary/Base64.

[61] Microsoft Open Specifications. Saml - statuscode, Feb 2019. URL: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-samlpr/96b92662-9bf7-4910-ab16-e1c28bce962b.

[62] Gradle. Community home, 2020. URL: https://docs.gradle.org/6.3/userguide/userguide.html.

[63] About mariadb server, 2020. URL: https://mariadb.org/about/.

# SAMMANFATTNING

# CENTRALISERAD AUTENTISERING OCH AUKTORISERING

## Introduktion

För enheter på nätet har det alltid varit viktigt att kunna verifiera identiteten sinsemellan. Det finns många olika lösningar och förslag på hur detta bör gå till, men det som ofta används i praktiken är fortfarande en kombination av användarnamn och lösenord. Denna avhandling ämnar utforska olika sätt där samlad inloggning kan användas för att öka säkerheten genom att minska antalet inloggningssystem och på samma gång minska antalet lösenord som varje användare måste hålla koll på.

Avhandlingen görs som en del av ett projekt med Hibox Systems, ett företag som tillhandahåller omfattande tv- och underhållningslösningar för hotell, internetleverantörer och sjukhus. Företaget har undersökt olika möjligheter att på ett säkert sätt försnabba underhåll och utvecklingsåtgärder genom att ge sina anställda tillgång till en centraliserad inloggningsserver. Denna server skulle ge rätt personal tillgång till andra konfigurerade externa servrar enligt behov, både enklare och mera flexibelt än tidigare.

Till projektet hör att utvärdera olika tillgängliga ramverk och tjänster som uppfyller både Hibox Systems funktionella krav samt deras säkerhetskrav. Till dessa krav hör att endast personal med tillräckliga rättigheter har tillgång till olika kunders servrar och att eventuella personalbyten kan leda till snabba uppdateringar av nämnda rättigheter. Utöver detta är kravet också att man effektivt kan reagera på förändringar i säkerhetskraven.

## Ramverk

Det finns en mängd olika ramverk tillgängliga för autentisering och auktorisering. På grund av den mängd ramverk som finns beskrivna i litteratur och tillgängliga på nätet kommer i denna avhandling endast ett litet urval av alla ramverk att behandlas. I denna avhandling har fokus lagts på att hitta ett varierande sortiment av ramverk som har blivit noggrant granskade av säkerhetsexperter samt visat sig fungera bra i praktiken.

De utvalda ramverken är OAuth 2.0, OpenID Connect, SAML, LDAP, Kerberos och RADIUS.

OAuth 2.0 och OpenID Connect är båda beroende av varandra på grund av att OpenID Connect är ett säkerhetslager som kan läggas till OAuth 2.0. OAuth 2.0 är ett industriledande protokoll för auktorisering och har med OpenID Connect en fullskalig autentiserings- och auktoriseringslösning. Utan OpenID Connect är OAuth 2.0 endast ett auktoriseringsramverk, vilket innebär att det inte innehåller ett standardiserat flöde för hur inloggningen till systemet bör genomföras.

SAML är en standard som definierar ett ramverk för hur entiteter sinsemellan kan utbyta säkerhetsinformation i ett XML-format. XML är en standard för hur man kan skriva text som enkelt kan skickas och förstås mellan olika entiteter.

LDAP är ett protokoll som beskriver kommunikation med en katalogtjänst. En katalogtjänst är ett system som innehåller och hanterar till exempel namn, adress och användares rättigheter.

Kerberos definierar ett protokoll som bygger på ömsesidig autentisering, vilket innebär att både användaren och servicen kan vara säkra på att den andra parten är den som den utger sig för att vara.

RADIUS bygger även den på ömsesidig autentisering och har historiskt sett använts främst av större företag för att begränsa åtkomst till och inom privata nätverk.

Olika ramverk har olika styrkor och svagheter, vilka tas upp och analyseras i avhandlingen.

## Testning och implementation

De utvalda ramverken jämförs först enbart på en teoretisk nivå. De ramverk som visar sig vara bäst sinsemellan i teoridelen testas och implementeras vidare. Detta görs för att kunna lägga större fokus på de ramverk som passar för det uppsatta målet för denna avhandling. De mest lämpliga ramverken integreras med Hibox Systems interna system, med fokus på att få fungerande och testbara prototyper. Prototyperna testas med betoning på genomströmning, användarvänlighet och säkerhet.

## Resultat och analys

Utgående från undersökningen av ramverken och testningen av integrationerna kan man se hur ramverken lämpar sig bättre och sämre i olika situationer, beroende på vilka säkerhets- och funktionella krav som följs.

Kerberos och RADIUS visade sig båda vara ytterst komplexa för både användare och administratörer. Problem som inkorrekta klockor och oklara felmeddelanden kan med dessa ramverk leda till onödigt komplicerad felsökning. Den extra säkerhet som dessa erbjuder uppväger inte de komplikationer som de medför.

LDAP och SAML medförde liknande komplexitet, men inte på en samma skala som de tidigare nämnda ramverken. SAML är lämpligare än LDAP för det givna användningsfallet på grund av de funktionaliteter som SAML erbjuder. Detta betyder också att det i vissa omständigheter kan vara ett onödigt omfattande och tungt ramverk.

OAuth 2.0 med OpenID Connect som säkerhetslager visade sig både i teorin och i integreringstester vara det bästa av de utvalda ramverken för att bygga ett effektivt och användarvänligt inloggningsflöde. Integreringen lyckades utan större svårigheter tack vare de väldefinierade protokollen. Det kräver en del administrativt arbete för att få systemet att fungera, men det relativt säkra inloggningsflödet kan följas väldigt smidigt ur användarens synvinkel.

SAML och OAuth 2.0, med säkerhetslagret OpenID Connect, integrerades båda med Hibox Systems interna lösningar. SAML var väldigt komplicerat att få konfigurerat ihop med det nuvarande systemet. OpenID Connect var överlag lättare att få att fungera, men vissa komplikationer uppstod även där.

En större genomströmming av användare kunde på samma hårdvara åstadkommas med hjälp av OpenID Connect jämfört med SAML. Resursanvändingen på servrarna som körde integrationen var lägre för OpenID Connect än för SAML. Processoranvändningen på dessa servrar var betydligt mindre för OpenID Connect, medan minnesanvändningen var väldigt lika för de bägge ramverken. OpenID Connect visade sig vara bättre än SAML på grund av lättare konfigurering och underhåll samt genomgående effektivare system.

## Slutsats

Baserat på resultaten från denna avhandling kan det konstateras att det ramverk som lämpar sig bäst för det syfte som presenterades i introduktionen är OAuth 2.0 med OpenID Connect som säkerhetslager liggande ovanpå.

RADIUS och Kerberos valdes bort ur de ursprungligt valda ramverken i ett relativt tidigt skede av testningen på grund av den oproportionerligt stora komplexitet dessa ramverk inför utan att tillföra tillräckligt med kompenserande positiva funktionaliteter och säkerhetsrelaterade fördelar.

LDAP och SAML visade sig fungera relativt bra ut säkerhetsmässig synvinkel. Dessa två ramverk var dock inte lika användarvänliga och lätta att konfigurera som OAuth 2.0. Av LDAP och SAML var det dock SAML som visade sig vara mest lämpad.

OAuth 2.0 med OpenID Connect som säkerhetslager visade sig kräva relativt lite resurser och var enkel att integrera med ett nuvarande system. Genomströmningen var även bättre än de andra testade ramverken, och även konfigureringen var väldigt okomplicerad.

Det finns många fördelar med att erbjuda endast en inloggningsserver, både för systemadministratörer och användare. Den kan även implementeras på många olika sätt, men för Hibox Systems ändamål kan man se att OAuth 2.0 och OpenID Connect lämpar sig bäst bland de utvalda ramverken.