Master’s Thesis
submitted in partial fulfilment of the
requirements for the course “Applied Computer Science”

Extending Single Sign-On Service in Federated
Identity Environments deploying OpenAM

Pranay Tare

Institute of Computer Science

Bachelor’s and Master’s Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
✉ +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌐 www.informatik.uni-goettingen.de

First Supervisor: Prof. Dr. Ramin Yahyapour
Second Supervisor: Dr. Sven Bingert
I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Acknowledgment

Firstly, I would like to express my sincere gratitude to my advisory Prof. Dr. Ramin Yahyapour for allowing me to write my final masters thesis in GWDG. Secondly, I would like to thank Dr. Sven Bingert, for helping and motivating me while writing my thesis. I am also thankful his patience, support and immense knowledge that helped me finish this on time. I could not have imagined having a better advisor and mentor for my masters study.

Besides my advisors, I would like to thank the rest of my thesis committee: Christoph Pohl for motivating and boosting me while problems, Christopher Menke for his advice in writing this thesis and for insightful comments.

I would also thank my friends for commenting on several areas, providing suggestions and for taking their time reviewing my thesis. And Last but not the least, I would like to thank my family: my parents and to my brothers for supporting me throughout writing this thesis.
Abstract

Single Sign-On has became a vital part of every web based system. Today service providers, organizations etc. internally or externally facilitate end-users to browse each service or content without using their credentials again. This thesis attempts to authenticate and authorize users using Single Sign-On facility with OpenAM access management system. It also aims to provide federated identity management solutions using OpenAM to utilize multiple protocols such as OAuth, SAML etc. In this study we develop a communication channel to communicate between several protocols and exploit them using Single Sign-On solution.

Keywords: Single Sign-On, Identity, Credentials, Authentication, Uni-Factor Authentication, Multi-Factor Authentication, Identity Provider, Service Provider, OpenAM, OpenIDM, OpenIG, OpenSSO, SAML, SAML2, Shibboleth IDP, Shibboleth SP, Authentication Authority, Authentication Server.
Contents

1 Introduction 2
  1.1 Motivation ................................................................. 2
  1.2 Thesis Outline ............................................................ 5

2 Foundations 6
  2.1 Single Sign-On (SSO) ..................................................... 6
  2.2 OpenAM ................................................................. 7
    2.2.1 Working of OpenAM ............................................. 9
  2.3 OpenDJ .............................................................. 11
    2.3.1 Use Cases ......................................................... 11
    2.3.2 OpenDJ Architecture ......................................... 14
  2.4 Shibboleth ............................................................ 16
    2.4.1 Shibboleth Identity Provider ................................. 17
    2.4.2 Shibboleth Service Provider ................................ 17
  2.5 OAuth ................................................................. 18
    2.5.1 OAuth Roles ...................................................... 19
    2.5.2 OAuth Protocol Flow ........................................ 20
  2.6 SAML ................................................................. 21
    2.6.1 SAML Architecture ........................................... 23
  2.7 kerberos .............................................................. 24
    2.7.1 Working of kerberos ......................................... 25

3 Requirement Analysis 27
  3.1 Comparison ........................................................... 27
    3.1.1 License ........................................................... 28
    3.1.2 Directory Service ............................................... 28
    3.1.3 Identity Intelligence .......................................... 28
    3.1.4 Space requirements ........................................... 30
    3.1.5 Web Access Management ...................................... 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.6</td>
<td>Mobile Access Management</td>
<td>30</td>
</tr>
<tr>
<td>3.1.7</td>
<td>Third Party Application Support</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>Why OpenAM?</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>Concept</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Multi-Protocol Federations and Single Sign-On</td>
<td>34</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Achieving OpenAM SSO with SAML 2.0</td>
<td>34</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Achieving OpenAM SSO with OAuth 2.0</td>
<td>42</td>
</tr>
<tr>
<td>4.1.3</td>
<td>SSO using WindowsDesktopSSO module in OpenAM</td>
<td>48</td>
</tr>
<tr>
<td>4.1.4</td>
<td>SSO decisions in OpenAM using Policy Agents</td>
<td>51</td>
</tr>
<tr>
<td>5</td>
<td>Configuration and Implementation</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Connection Bridge / Gateway</td>
<td>53</td>
</tr>
<tr>
<td>5.1.1</td>
<td>System Configuration</td>
<td>53</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Configuring Communication Bridge</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>Result</td>
<td>65</td>
</tr>
<tr>
<td>6.1</td>
<td>Outcome of bridging between OpenAM IdP and Shibboleth SP</td>
<td>65</td>
</tr>
<tr>
<td>6.2</td>
<td>Outcome of bridging between OpenAM IdP and OpenAM SP</td>
<td>67</td>
</tr>
<tr>
<td>6.3</td>
<td>OAuth 2.0 authorization with OpenAM</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion and Future Work</td>
<td>72</td>
</tr>
<tr>
<td>A</td>
<td>Installation Guide</td>
<td>73</td>
</tr>
<tr>
<td>A.1</td>
<td>OpenAM and OpenDJ Installation</td>
<td>73</td>
</tr>
<tr>
<td>A.1.1</td>
<td>Basic installation</td>
<td>73</td>
</tr>
<tr>
<td>A.1.2</td>
<td>OpenDJ Installation</td>
<td>73</td>
</tr>
<tr>
<td>A.1.3</td>
<td>Install OpenAM</td>
<td>74</td>
</tr>
<tr>
<td>A.2</td>
<td>Shibboleth IdP and SP</td>
<td>74</td>
</tr>
<tr>
<td>A.2.1</td>
<td>Basic Configuration</td>
<td>74</td>
</tr>
<tr>
<td>A.2.2</td>
<td>IDP Installation</td>
<td>75</td>
</tr>
<tr>
<td>A.2.3</td>
<td>Installing SP</td>
<td>78</td>
</tr>
<tr>
<td>A.2.4</td>
<td>SP Installation</td>
<td>78</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Single Sign-On using OpenAM ................................................. 3
1.2 Management system for multi-protocol federations ................. 4

2.1 High-Level Architecture of OpenSSO (OpenAM) ......................... 7
2.2 Functional Architecture of OpenAM ........................................ 8
2.3 OpenAM Basic working flow .............................................. 10
2.4 Use Case: OpenDJ Replication ............................................ 12
2.5 OpenDJ Architecture ....................................................... 14
2.6 Shibboleth IdP and SP Architecture ..................................... 16
2.7 OAuth working ............................................................... 19
2.8 OAuth protocol flow ....................................................... 20
2.9 SAML based Web Browser Single Sign-On (SSO) Use-Case ............ 22
2.10 General process of Kerberos Ticket Exchange ......................... 25

3.1 Forgerock Products .......................................................... 33

4.1 SSO using OpenAM as IDP and Shibboleth as SP ......................... 35
4.2 SSO using OpenAM as IDP and Shibboleth as SP ......................... 37
4.3 SSO using OpenAM as IDP and Shibboleth as SP ......................... 38
4.4 SSO using OpenAM as IDP and Shibboleth as SP ......................... 39
4.5 OpenSSO (OpenAM) Load Balancer in a Single Site with Session Failover .............................. 40
4.6 Session Failover with CTSI .................................................. 41
4.7 Authorization code flow when OpenAM act as OAuth 2.0 provider ............ 42
4.8 Code flow when OpenAM act as service request handler ............... 44
4.9 Code flow when OpenAM act as service request handler ............... 45
4.10 Code flow when OpenAM act as service request handler .......... 46
4.11 Code flow when OpenAM act as service request handler .......... 47
4.12 Code flow when OpenAM act as service request handler .......... 48
4.13 Code flow when OpenAM act as service request handler .......... 49
4.14 Working of Policy agents with OpenAM ................................ 52
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Working of Policy agents with OpenAM</td>
<td>54</td>
</tr>
<tr>
<td>5.2</td>
<td>OpenAM create Hosted IdP console</td>
<td>58</td>
</tr>
<tr>
<td>5.3</td>
<td>OpenAM create Hosted IdP console</td>
<td>60</td>
</tr>
<tr>
<td>5.4</td>
<td>krb5 configuration</td>
<td>62</td>
</tr>
<tr>
<td>5.5</td>
<td>kdc configuration</td>
<td>63</td>
</tr>
<tr>
<td>5.6</td>
<td>kdc database creation</td>
<td>63</td>
</tr>
<tr>
<td>5.7</td>
<td>creating service principal creation</td>
<td>63</td>
</tr>
<tr>
<td>5.8</td>
<td>keytab creation</td>
<td>64</td>
</tr>
<tr>
<td>6.1</td>
<td>Shibd log: INFO about resource detection</td>
<td>65</td>
</tr>
<tr>
<td>6.2</td>
<td>Tomcat log: HTTP GET request, /SSORedirect from SP to IdP</td>
<td>66</td>
</tr>
<tr>
<td>6.3</td>
<td>Tomcat log: HTTP GET request to fetch data from IdP login</td>
<td>66</td>
</tr>
<tr>
<td>6.4</td>
<td>Tomcat log: HTTP POST response from user browser to IdP</td>
<td>66</td>
</tr>
<tr>
<td>6.5</td>
<td>Tomcat log: GET Credential check IdP datastore</td>
<td>67</td>
</tr>
<tr>
<td>6.6</td>
<td>Tomcat log: HTTP POST request to fetch values from IdP</td>
<td>67</td>
</tr>
<tr>
<td>6.7</td>
<td>Tomcat log: HTTP GET/POST OpenAM attempts to create session and validates it</td>
<td>67</td>
</tr>
<tr>
<td>6.8</td>
<td>Tomcat log: HTTP GET resource information</td>
<td>68</td>
</tr>
<tr>
<td>6.9</td>
<td>Tomcat log: HTTP GET requests information from IdP</td>
<td>68</td>
</tr>
<tr>
<td>6.10</td>
<td>Tomcat log: HTTP GET request to fetch data from IdP login</td>
<td>68</td>
</tr>
<tr>
<td>6.11</td>
<td>Tomcat log: HTTP POST response from user browser to IdP</td>
<td>69</td>
</tr>
<tr>
<td>6.12</td>
<td>Tomcat log: GET: Credential checking in OpenAM IdP datastore</td>
<td>69</td>
</tr>
<tr>
<td>6.13</td>
<td>Tomcat log: HTTP POST request to fetch values from IdP</td>
<td>69</td>
</tr>
<tr>
<td>6.14</td>
<td>Tomcat log: HTTP GET/POST OpenAM attempts to create session and validates it</td>
<td>70</td>
</tr>
<tr>
<td>6.15</td>
<td>Tomcat log: HTTP GET/POST OpenAM create SSO session</td>
<td>70</td>
</tr>
<tr>
<td>6.16</td>
<td>Tomcat log: Login request via Facebook</td>
<td>70</td>
</tr>
<tr>
<td>6.17</td>
<td>Tomcat log: Redirection to Facebook page requesting Login</td>
<td>71</td>
</tr>
<tr>
<td>6.18</td>
<td>Tomcat log: Redirection from Facebook after authentication</td>
<td>71</td>
</tr>
<tr>
<td>6.19</td>
<td>Tomcat log: Redirection from Facebook after authentication</td>
<td>71</td>
</tr>
</tbody>
</table>
List of Tables

1 Acronyms ......................................................... 1

2.1 Example attributes in OpenDJ ................................ 12
2.2 OpenDJ attribute uniqueness parameter .................. 13

3.1 Comparison of different IAM solutions .................... 29

5.1 System configuration ....................................... 54
5.2 Code snippet for native.logger ............................. 55
5.3 Code snippet for Shibboleth SP <RequestMapper> plugin and ApplicationDefaults section .................. 56
5.4 Code snippet defines SPNativeApplication handler .......... 56
5.5 SPSessionInitiator handler snippet ........................ 56
5.6 SAML 2.0 Embedded AuthnRequest snippet ................ 57
5.7 ApplicationOverride settings ............................... 57
5.8 Settings to store Metadata locally ......................... 57
5.9 Single lined mapping attribute ............................ 58
5.10 Assertion Processing value in remote Federation tab .... 59
5.11 Metadata format for remote identity provider ............ 59
5.12 Components used for configuring WindowsDesktopSSO .... 61

A.1 tomcat configuration .................................... 76
A.2 debug mode configuration ................................ 76
A.3 Configuration login.config file ............................. 76
A.4 Create idp.xml file .................................... 77
A.5 Modify relaying-party.xml file ............................. 77
A.6 Modify shibboleth2.xml file ............................... 78
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSO</td>
<td>Single Sign-On</td>
</tr>
<tr>
<td>OAuth</td>
<td>Authorization protocol</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>AD</td>
<td>Active Directory</td>
</tr>
<tr>
<td>ssoadm</td>
<td>SSO Admin</td>
</tr>
<tr>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ELK</td>
<td>Elasticsearch, Logstash and Kibana</td>
</tr>
<tr>
<td>IAM</td>
<td>Identity and Access Management</td>
</tr>
<tr>
<td>IDM</td>
<td>Identity Management</td>
</tr>
<tr>
<td>IdP</td>
<td>Identity Provider</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Access Object Protocol</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
</tbody>
</table>

Table 1: Acronyms
Chapter 1

Introduction

In this chapter we discuss the notion behind this dissertation. We also provide an outline define each chapter in brief.

1.1 Motivation

There is a growing trend towards simplifying web based access mechanisms in organizations, universities and even in personal businesses for improving efficacy and productivity of end-users. Web services we use today are simplified enough to provide hazel free scrolling and functioning of tasks. Almost all companies, internally or externally use services such as Single Sign-On (SSO) for signing into their systems. As Single Sign-On functionality provides a seamless mechanism to access all configured services by signing with one single credential. Although, this has already been in the industry for several years. The Göttingen Society for Scientific Data Processing (GWDG)[2] still lacks in providing such useful functionality to its users. Due to which, users have to signin every time for every service they use. Meanwhile when a user attempts to log in GWDG portal, he is unable to switch between services such as GWDG SharePoint or GWDG ownCloud without using his credentials everytime. This reduces the efficiency of end-user and increases time wastage.

To resolve this problem, we use OpenAM access management solution in this study. This not only implements Single Sign-On solution but provide a central and federated environment to easily manage all configurations. OpenAM is also beneficial for implementing multiple protocols as it provides a straightforward way to configure Identity providers and Service providers, multiple protocols such as SAML 2.0, OAuth 2.0 etc. This powerful tool would provide a better way to

1.1. MOTIVATION

manage almost everything in GWDG data center.

Figure 1.1 depicts OpenAM as a federated system through which Single Sign-On facility could be easily achieved. When an end-user U attempts to access GWDG service (portal.gwdg.de), he is redirected to OpenAM portal for authentication and after successful authentication, user gains access to requested resource. Although for initiating Single Sign-On services, user can easily switch between services that are configured by GWDG such as GWDG ownCloud without using his credentials. Moreover for confidential resources, higher security mechanisms can be established by implementing policies and policy agents for each resource.

Secondly, GWDG do not have a single management system that could be configured for number of multi-protocol federation solutions such as Shibboleth, OAuth, SAML, CoSign single sign on, ClearLogin etc. To overcome this problem OpenAM can be used as a backbone for configuring and supervising these solutions. It would also enable users from different universities to connect and collaborate in different research activities with GWDG, University of Goettingen and Max-Planck Institutes.

Figure 1.1: Single Sign-On using OpenAM

\(2\)http://openam.forgerock.org, Accessed March
CHAPTER 1. INTRODUCTION

Figure 1.2 shows, whenever a user from Institute B tries to access services, he is redirected to the service provider. Which then authenticates by gaining credential information from identity provider. In this test case, OpenAM can act as a proxy or as an identity provider which asserts user to the service. OpenAM does not store user credentials while acting as a proxy, it only helps authenticating the end user. This is beneficial for institute B, as it not liable to share user credentials with other institute. Hence the aim of this thesis is to implement an access management system that would allow GWDG to manage all identity providers and service providers with single portal. It would also help to create a communication channel between different federated solutions i.e. OpenAM and Shibboleth (in this case) and implement Single Sign-On facility using OpenAM access management system which will facilitate end-users to access all the services using one single log in.

Figure 1.2: Management system for multi-protocol federations
1.2 Thesis Outline

The dissertation is systematically organized in seven chapters: introduction, foundation, analysis, concept, implementation, result and conclusion.

Chapter 1 introduces the motivation and the benefits behind the thesis.
Chapter 2 attempts to provide a basic understanding about technologies used in this work.
Chapter 3 provides a brief analysis of these technologies and why we used OpenAM?
Chapter 4 shows the related work in this area and current solutions.
Chapter 5 implements the systems that were described in Chapter 4.
Chapter 6 The result section, describes the logs and process that undergoes while communication process occurs.
Chapter 7 This chapter provides conclusion and future work of this thesis.
Chapter 2

Foundations

This chapter covers the basic concepts needed to understand the system. It creates a backbone for all implemented systems such as OpenAM, OpenDJ, Shibboleth IDP and SP and protocols like Kerberos, OAuth2, SAML that are used widely for authentication and authorization. Here we also discuss about the characteristics, system architecture and also flow diagrams for every process.

2.1 Single Sign-On (SSO)

Single Sign-On (SSO)\(^1\) was suggested as a single sign-in solution, as it enables users to authenticate themselves only once for all the applications. The user than gets access to the protected services without the need of re-authenticate. The use of SSO, has provided a much easier way to deal with several digital identities, it provides a single identity to each user, while enabling to access easily. A user provides his credentials once and the SSO maintains its digital identity until the system is logged off or the session is expired. The maintenance of authentication data and enforcement of authentication policies become much easier due to centralized mechanism of SSO provider. Single Sign-On can be achieved using several ways:

- It can be achieved using Security Assertion Markup Language (SAML)\(^2\), which is an XML-based framework used for authentication and authorization of services.

- It can also be implemented by using Delegate authentication techniques such as OAuth2 protocols. As OAuth2 issues access tokens to access protected resources. Another way to carry SSO is by using Lightweight Directory Access Protocol (LDAP)\(^3\).

---

\(^1\)https://en.wikipedia.org/wiki/Single_sign-on

...and stored LDAP on systems
2.2 OpenAM

OpenAM is an access management solution that is used to secure web, cloud and mobile accessible resources. It forms "all-in-one" solution consisting of SSO, adaptive authentication, strong authentication, federation, web services security, and fine-grained entitlements. It is easy to use and configure services like multi-protocol federation enterprise access control and SSO.

The main aim of OpenAM is to deliver comprehensive and rich service platform to Administrators and empower end-users with easy sign in solutions. Its major task also include checking access conditions i.e. identity of user and whether user has access rights for the protected application or not? If both policies are fulfilled then user is allowed to access the resource. Figure 2.1 describes the main functions of OpenAM high level architecture.

1. Centralized Configuration: It enables OpenAM to reduce duplication among all the applications, this means the configuration settings are homogeneous to all resources. Although it can also be configured as per application. OpenAM manages and grabs all the data from centralized configuration data.

---

4https://en.wikipedia.org/wiki/OpenAM
5https://www.gartner.com/doc/1476017/multiprotocol-federation-interoperability-demonstration
2. Identity Repository: User and group credentials are stored at one place. The root user is allowed to modify the credentials. It is managed centrally as OpenAM has its own active directory.

3. Administration Console and Core Components: Both are main components of the complete OpenAM system. As administration console (ssoadm) enables administrator to setup services, configure service providers, install policies and even manage active directory servers. And core components are directly attached to the administrative console, it helps in configuring services etc. It includes load balancing, policy agents, protocols etc.

The end user can access all the services via internet connection. The above mentioned functions permits authentication and authorization. For detailed explanation about components, we can look at service level architecture in figure 2.2.

![Figure 2.2: Functional Architecture of OpenAM](http://docs.oracle.com/cd/E19681-01/820-3740/ggesa/index.html)

The above figure describes several service layers that form the complete OpenAM solution. These are discussed in detail below.

1. UI Layer and Protected Layer: The highest layer which consists of several policy agents such as java ee agent, web agent, self contained java applications etc. and service components for management. The UI layer is used for managing end users for administrative purposes.

---

2.2. OPENAM

and as a session providing authority for end user. When an authorized user access protected application, OpenAM generates a session token for the user. It is present in the browser cookie and helps user to authenticate himself for all the services.

2. Access Layer: This helps end users to connect OpenAM services and administrators to configure service providers via OAuth2, SAML protocols. OpenAM provides REST API's for integration with different applications. This layer helps in establishing connection with these services.

3. Service Layer: The layer describes different components of OpenAM architecture and it also provides an overview about services that OpenAM supports. This is designed by integrating several technologies such as Single Sign On, Federations, Auth protocols, Entitlements, Load balancing etc. The administrator configures and manages all the services via this layer.

4. Data Persistent Layer: It is used for storing and retrieving data from active directories, databases etc. OpenAM is often configured with OpenDJ active directory server, which is an easy to deploy java based application. The data persistent layer consists of several databases, active directories etc. which can be easily embedded into OpenAM.

5. External Layer: OpenAM can also be used with analytic tools like Elasticsearch, Logstash, Kibana (ELK). Administrators and users can use OpenAM for analyzing trends and generating reports. This layer facilitates with all the external software packages.

2.2.1 Working of OpenAM

The basic working model of OpenAM explains how requests are generated, executed and processed for a protected resource.

Figure 2.3 explains the basic communication occurring between two parties i.e. browser at user-end and the protected resource at the service providers-end uses HTTP protocols. The user sends an HTTP request for gaining access to protected resource, the policy agents inspects whether the request has a valid authentication token or not? If token is not present the request is forwarded to OpenAM authentication portal, which then authenticates and authorizes the request. After successful authentication a session token is generated, stored and sent back via browser cookie. The session token is present in browser until it expires or browser cookies are cleared.

After getting the cookie from browser, the Policy agent communicates directly with OpenAM to check if the user really has a valid token and is authorized for accessing the resource? After which it grants access to the protected application.

This process not only allows the user to securely communicate and access the resource. But, it also
allows service providers to protect their resources using different authentication methods and only allow the users that have valid credentials. Several other protocols such as SAML, OAuth etc. are also used for this purpose.

Figure 2.3: OpenAM Basic working flow
The above figure 2.3 shows step by step procedure that is followed during authentication and authorization process. It also generalizes about different service layers that are accessible via application programming interfaces (API's). These layers coordinate to form a complete unified system.

2.3 OpenDJ

OpenDJ[^8] is a Lightweight Directory Access Protocol (LDAPv3) and Directory Service Markup Language (DSMLv2) directory service, which is highly suitable for managing and securing identities. It is easy to deploy, manage and track record changes. It also consists of embedded big data platform that can be used for real-time implementations in mobile devices and cloud environments etc. OpenDJ provides high-performance and availability in securing and storing information on and off-premises. It is widely used in the industry for fast deployment and management of applications. Some use-cases can be seen in below sub sections.

2.3.1 Use Cases

This section provides a better understanding of OpenDJ in real-life scenarios. Test cases are introduced for key features of OpenDJ and how they are useful in everyday scenario.

I. OpenDJ Replication

Replication of data and configuration services is very useful as it provides a different source to execute services when any calamity occurs. OpenDJ provides N-tier multi-master replication which is scale to many servers and is persistent towards any disaster.

Many organizations might only want to store some values or specific attributes. OpenDJ provides functional replication capability, that allows administrators to replicate only certain parameters from the server. This is highly beneficial in industries like data centers i.e. GWDG, where power is an important factor and power outage can cause hindrance to real-time services.

II. Attribute Uniqueness

In Identity management systems, attribute uniqueness[^9] is obligatory due to authentication and authorization processes. These attributes are often stored in non-efficient active directories that are unable to trace parameter uniqueness. The above table 2.1 shows two different entries in OpenDJ.

[^9]: https://en.wikipedia.org/wiki/OpenDJ
CHAPTER 2. FOUNDATIONS

Figure 2.4: Use Case: OpenDJ Replication

Table 2.1: Example attributes in OpenDJ

<table>
<thead>
<tr>
<th>dn:</th>
<th>uid=pranay1,ou=People dc=example,dc=com</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>uid: pranay1</td>
</tr>
<tr>
<td></td>
<td>email: <a href="mailto:pranay@test.com">pranay@test.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dn:</th>
<th>uid=pran,ou=People dc=example,dc=com</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>uid: pran</td>
</tr>
<tr>
<td></td>
<td>email: <a href="mailto:pranay@test.com">pranay@test.com</a></td>
</tr>
</tbody>
</table>
2.3. **OPENDJ**

The email parameter is same for both. To prevent this OpenDJ provides an efficient mechanism to ensure that all the values in a given active directory are unique. It supplies a plug-in that could be configured easily for attribute uniqueness. This plugin is "disabled by default" i.e. false, but can be modified in main configuration (config.ldif) file of OpenDJ i.e. set to true.

<table>
<thead>
<tr>
<th>dn: cn=Access Control Handler,cn=config</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds-cfg-enabled: false</td>
</tr>
<tr>
<td>ds-cfg-java-class: org.opends.server.authorization.dseecompat.AciHandler</td>
</tr>
<tr>
<td>cn: Access Control Handler</td>
</tr>
</tbody>
</table>

Table 2.2: OpenDJ attribute uniqueness parameter

Everyday many user from different universities sign up for a new account in GWDG, which might have used GWDG services and have forgotten their credentials or changed departments or universities. Having OpenDJ in backend would not only help end-users to reset their accounts, but will help GWDG to have less user identification problems.

### III. Security

Security has always been an important factor in any organization. Protecting data leakage, employee identity and critical information is always main targets of hackers. OpenDJ can help with securing information as it contains wide range of security algorithms which includes SSL, StartTLS, and certificate-based etc. It also gives the ability to encrypt information using policies before storing into LDAP server. It also allows to audit and archive the data in-case rollback is required. Currently, Virtual machines are regularly hacked in GWDG, as a result the threat of loosing critical information is always at top priority. GWDG can use these exceptional features to secure user information, by encrypting data before it is saved.

### IV. Monitoring

In earlier section we discussed about different security features. But, to regularly notice and inform problems monitoring is required. It helps to customize alerts such as password expiration, access controls disablement, backend database corruption detection, user lockouts, credential reset etc. to administrator. These issues can be easily tracked and resolved using several monitoring tools with OpenDJ as backend. GWDG can easily track problems associated with user credentials, account creation etc. This will be helpful and hazel free for administrators.
2.3.2 OpenDJ Architecture

The architecture of OpenDJ consists of two main components: Administration and Directory service:

Administration: OpenDJ provides administrator the flexibility to configure its services either by using graphical environment or by using command line interface. Both methods provide the same set of functions that can be used with ease and convenience.

![OpenDJ Architecture Diagram](https://idmdude.com/2012/07/23/the-opendj-architecture/)

Figure 2.5: OpenDJ Architecture

---

1) idmdude.com/2012/07/23/the-opendj-architecture/
2.3. OPENDJ

Directory Server: It is basically responsible for the backend of OpenDJ server. It consists of several components that are discussed below.

1. Front End: It is the user facing part of OpenDJ server. It is used to provide various services such as:
   - Connection Handling: Connection handler is basically responsible to start, stop and connect OpenDJ services to LDAP clients. It is one of the central components of OpenDJ server, as it manages all the requests to and from the clients towards OpenDJ server. This includes accepting new connections, responding to client requests etc. It also performs protocol translation and encryption services from client to LDAP and vice versa. OpenDJ includes several communication forms such as JMX, SSL, LDAP, LDAPS, LDIF, SNMP etc. used to connect the services. Administrators have the ability to enable or disable these connection handlers to support their client environment.
   - Monitoring and Alerting: It is used to monitor all the information about users, credentials etc. and when a problem is detected the administrator is automatically alerted with the message.
   - Logging: OpenDJ has a robust logging capability, which allow all the information to be stored in log files. Some of the loggers include: Access Logger (stores server operations like binds, searches, modifications, etc.), Error Logger (stores warnings, error messages) and Debug Logger (records debug information when the server is running and Java assertions are active). All the logging information is stored in files and administrator can trace all the information about a particular event.

2. Core Processing: This component contains several modules such as Access Control, policy enforcement, password policy enforcement, Cache management etc. These modules are used with policy agents to enforce rules on particular resource. Access Control, policy enforcement, password policy enforcement are used to manage users requesting the services and allowing them to use them if authorized. Whereas cache management is used to configure and efficiently improve cache memory.

3. Backend Databases: The backend connecting OpenDJ are highly efficient to handle very large work queues in little amount of time. Backends in OpenDJ are of two types: Local and Alternate backends.
   - Local backend databases can include ldif files, java db etc. The local backend databases are highly efficient in indexing, searching, retrieving, and storing data in database. Although, Alternate backend databases can be connected to OpenDJ server in case of big data etc. OpenDJ currently supports Oracle Databases which are highly efficient in processing. There can be multiple backend databases at any given time and each of which handles exclusively to other databases. An administrator can interact with these databases using GUI or command line interfaces which provides tools for enabling, disabling, creating, removing, backing up, and restoring the databases without impacting other backends.
2.4 Shibboleth

Shibboleth[^8][^9] is a Single Sign On solution, which facilitates end-users with signing-in (Login). It uses Security Assertion Markup Language (SAML) protocols for communication and was developed by internet2 for federated identity management i.e. authentication and authorization in web architecture.

Shibboleth also implements Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML), Simple Object Access Protocol (SOAP) in its architecture to interact between Identity Provider (IdP) and Service provider (SP). These two i.e. identity provider, service provider including "Where are you from?" (WAYF) service (optionally) three functional components, which interacts with other subcomponents.

[^9]: https://en.wikipedia.org/wiki/Shibboleth_(Internet2)
2.4. SHIBBOLETH

2.4.1 Shibboleth Identity Provider

Identity Provider[^14] is also known as Identity Assertion Provider, which generates security tokens for authenticating users or system processes. These authenticating statements are sent to relaying parties in SAML formation, which then authorizes and authenticates the user respectively. The Shibboleth identity provider also supplies user information to service providers and is comprised of several components which are helpful for authentication and authorization of users and services. These components are as follows:

1. Authentication Authority: It is an authentication issuing authority, which is obligated to corroborate principles and components to relaying parties (Service Providers). It is based on SAML protocol used in accordance with Browser/POST and Browser/Artifact profiles.

2. Attribute Authority: Similar to Authentication Authority, Attribute Authority is also a SAML based service that process attribute requests i.e. authorizes and authenticates. These attribute requests and principals are governed using access control mechanisms and rely on Secured Socket Layer (SSL/TLS)[RFC 2246] or SAML signatures for validation exchanges.

3. Single Sign-On Service: The single sign on service is the main interaction point of identity provider. When a service provider forwards authentication process to identity provider, the SSO service at IdP’s end initiates authentication mechanism redirecting the browser to Inter-Site Transfer.

4. Inter-Site Transfer Service: Inter-Site Transfer responds to the requests via HTTP upholding Browser/POST and Browser/Artifact profiles.

5. Artifact Resolution Service: Artifact Resolution Service is governed by identity provider using SAML protocol binding services. The service resolves SAML artifacts in agreement with Browser/Artifact profile. The service provider sends artifact to the artifact resolution service at the identity provider via a back-channel and IdP responds with authentication assertion to service provider.

2.4.2 Shibboleth Service Provider

A service provider[^16] is an entity that is liable for authorization of secured resources i.e. web-based service or applications. The connection is formed using SAML browser assertions sent by identity provider to service provider. Although, due to the presence of access control the connection requests are filtered using security context.

[^16]: https://shibboleth.net/products/service-provider.html
CHAPTER 2. FOUNDATIONS

1. Assertion Consumer Service: Assertion Consumer Service also known as SHIRE is an HTTP resource controlled by service provider which process information exchange. SAML GET/POST services are used via Browser profiles to initiate and respond towards requests. After successful authentication the service is redirected to the protected resource.

2. Attribute Requester: The sub component is used for attribute exchange as it combines the service provider and identity provider via a backend-channel for communication. It is executed using SAML protocol binding, which send a SAML <samlp:Request> containing Attribute query <samlp:AttributeQuery> element to process assertions. After which the connection is established and Service provider and identity provider can interact directly, detouring the browser.

3. WAYF Service: WAYF, or "Where are you from?", is an optional service which is used to determine preferred identity provider. This means WAYF acts as a proxy for determining the identity provider to which authentication requests can be relayed. Although, it requires a user agent to cache user’s selection.

2.5 OAuth

OAuth\(^\text{17}\) or Open Standard for Authorization\(^\text{18}\) is a secured way of allowing third-parties to access user resources without sharing credentials. It is often associated with a "valet key", which grant limited access to resources (duration, scope). Just as a it provides restricted access to car, allowing valet to drive the car for certain duration or preventing him to open the trunk. OAuth permits client application to access secured data using tokens distributed by authorization server in response to authorization request. The above figure can be described for photo sharing service (example, Picasa) as a resource server and a printing server (example, print.uni-goettingen.de) as a client. The main goal is to authorize and provide printing service read-only access to our photos for a limited amount of time. After which authorization gets invalid or tokens gets revoked.

The figure explains, user requests client to access the data and provides its credentials (username and password). After which client requests access to controlled resources that are restricted by resource owner (User). Meanwhile authorization server issues different set of credentials i.e. access tokens (which is a string containing scope, lifetime, and privileges etc.) in accordance with resource owner (User). This grants printing service access to protected application (Picasa) without sharing credentials.

\(^{17}\)https://en.wikipedia.org/wiki/OAuth  
\(^{18}\)http://oauth.net/2/
2.5. OAuth

2.5.1 OAuth Roles

OAuth defines four different roles for its working:

1. **Resource Owner**: When the client generates requests to access the protected resource. Resource Owner helps the client to grant access to protected resource.

2. **Resource Server**: The data is stored in this server and can be accessed after successful authentication and authorization. It responds to authorization requests using tokens.

3. **Authorization Server**: Authorization server basically generates access tokens which contain scope of access, time limit to access the protected application etc. It does this after client is authorized to resource owner.

4. **Client**: It is an application that accesses protected application on end-users behalf.

---

19 [tutorials.jenkov.com/oauth2/roles.html](http://tutorials.jenkov.com/oauth2/roles.html)
20 [www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2](https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2)
2.5.2 OAuth Protocol Flow

The subsection provides a brief overview of how OAuth interacts and manages between all four OAuth Roles.

![Figure 2.8: OAuth protocol flow](image)

1. A: The client generates authorization request from user. The authorization request can be directly handled by the User or it can be handled by authorization server.

2. B: The client gets authorization grant from user. The grants can be of four types (Authorization Code, Implicit, Resource owner password credentials or client credentials. These grants also depend on how the client is requesting the protected application.

3. C: The authorization server authenticates the client and provides an authorization grant.

4. D: The authorization server verifies if the client and authorization grant is valid? After which an access token is generated.

5. E: The client authenticates itself to resource server by presenting access tokens.

6. F: If access tokens are valid, permission is granted to access the private/secured resource.
2.6 SAML

Security Assertion Markup Language (SAML) not only possesses a prominent position in worldwide industry acceptance for federated identity management systems, but is also one of the most secure protocols used for Single Sign-On solutions. It is an XML-based open-standard that allows secure exchange of authentication and authorization data between SAML assertion party and SAML relaying party. This flexibility to exchange data allow different federations to choose whatever data attributes they want to share.


In this scenario, when an end-user attempts to access web application from a web browser that is protected by SAML service provider. The service provider supplies an authentication request to SAML identity provider via user agent(Web browser). The identity provider generates an XHTML form containing the response. If user is identified, the service provider grants access to the protected application.

1. Request to access target resource: SAML is highly reliable on HTTP protocols, which are used when User agent(Web browser) generates a request to access target protected application "https://sp.example.com/myresource". The service provider validates the request by performing security check.

2. After the security check, the service provider checks for valid and preferred identity provider that was configured earlier by service provider and redirects the web browser to Single Sign-On service of identity provider "https://idp.example.org/SAML2/SSO/Redirect?SAMLRequest=request".

3. The browser when redirected to identity provider, performs GET operations using SAML requests and performs authentication while verifying user data.

4. If the user has invalid credentials, the request is canceled. But, if user has a valid permission an XHTML document is generated containing both request and response parameters.

5. The browser generates POST request at service providers end.

6. Which then redirects the browser to service provider while authenticating the user to the protected resource.

7. The web browser again generates the request at service provider to access the application.

8. The service provider grants access to the authenticated application.

Figure 2.9: SAML based Web Browser Single Sign-On (SSO) Use-Case
2.6. SAML

2.6.1 SAML Architecture

SAML is deeply embedded in XML. It provides a basic way of communication between SAML assertion parties and SAML relaying parties and format in which XML is defined in pair of XML Schema. Moreover communication between SAML service provider and SAML identity provider i.e. requests and responses are established and transmitted using Simple Object Access protocol (SOAP) via HTTP interface.

SAML architecture consists of four major components:

1. Assertions : SAML authority from one party generates assertion statements containing security information. These statements are of three types:
   
   (a) Authentication Statements : Authentication statements are created by the authentication authority that is responsible for user authentication. Moreover, it also defines a particular way by which authentication should be carried out and the time at which it takes place.
   
   (b) Attribute Statements : Attribute Statements are responsible for verifying and identifying all the information about the user. Whenever a user requests for authentication, all the attributes like name, email, credentials etc. are checked using this function.
   
   (c) Authorization decision statements : These provide the permissions and checks for the rights of user i.e. it checks whether the user has been granted the permission to access and use the resource? All these decisions are governed by this function.

2. Protocol : Several request/ response protocols are defined by SAML. These are used for all purposes of communication i.e. Single Sign-On/ Single Sign-Off, authentication etc. and are described below.

   (a) Authentication Request Protocol : This protocol is used to request assertion tokens having several assertion statements. It is basically used to establish security context when the user is redirected from Service Provider to Identity Provider.

   (b) Single Logout Protocol : The protocol is often used to logout all the active sessions with one user, which means if a user has several active sessions on computer, mobile device etc. After he initiates logout, all the sessions will be terminated. Altough, it can also be terminated by Identity Provider due to session timeout etc.

   (c) Assertion Query and Request Protocol : The request assertion asks the asserting parties for all the existing assertions whereas query define the conditions how new or existing assertions can be inquired.

   (d) Artifact Resolution Protocol : It provides a means to pass SAML protocol messages using artifacts(fixed-length messages).
(e) Name Identifier Management and Mapping Protocol: The management protocol is used to modify name identifiers of user and mapping protocol verifies if policy matches or not? After which it maps one SAML identifier to another.

3. Binding: Binding basically defines how SAML messages can be mapped to other protocols. In SAML 2.0 several binding methods are defined such as HTTP Redirect Binding, HTTP POST Binding, HTTP Artifact Binding, SAML SOAP Binding, SAML URI Binding and Reverse SOAP (PAOS) Binding. These help to transport messages and connect using HTML.

4. Profiles: Profiles are combination of Assertions, Protocols and Binding that are used to provide better interoperability. Policies that are used by SAML for Single Sign-On are defined below:

(a) Web Browser SSO Profile: The profile defines a way of achieving Single Sign-On using web browsers. It uses HTTP redirect, HTTP post and different HTTP binding methods to connect and transport messages.

(b) Enhanced Client and Proxy (ECP) Profile: Reverse-SOAP (PAOS) and SOAP bindings are used in this profile for gateway proxies or clients.

(c) Identity Provider Discovery Profile: It suggests a way by which any Service Provider can search whether a user has visited an Identity Provider. It tracks all the information about end-user.

(d) Single Logout Profile: It provides a way SOAP can be used with different HTTP protocols.

SAML provides a full control to companies. Google, Microsoft etc. can act as service providers while OpenAM can act as a Identity Provider hosted by GWDG.

2.7 kerberos

kerberos\(^{22}\) is a network authentication protocol which was developed in Massachusetts Institute of Technology (MIT). Its purpose is to provide strong authentication for users and their services and allow them to communicate on a non-secured channel. It is based on client/server model which provides mutual authentication using symmetric key cryptography. The motivation to create such a protocol is the insecurity of communication protocols on Internet. Which are not secure enough to prevent attacks like eavesdropping, replay attacks, sniffing etc. These standard protocols use authentication by assertions, which are unsecure and provide a low level of security. As an example GWDG login account "gwdglogin" has a user gwdgloginPr. The user logs in on remote system with his user credentials. gwdglogin deamon will assert users

\(^{22}\)https://en.wikipedia.org/wiki/Kerberos_(protocol)
identity to remote system and daemon will not require a password at all for logging in. If a hacker is able to persuade the login by proving his authenticity, the machine will be hacked and data might be stolen. Alternative, several web based applications by default use firewalls for securing themselves. But firewalls usually secure the network from outside attacks and not from internal attacks. Moreover they restrict users to from using several web service over the network. To encounter these network security problems kerberos was designed. It solves these problems using cryptographic techniques i.e. it encrypt and decrypt short messages by which user can accurately be identified by authenticating and verifying authority! kerberos can also encrypt complete client/ server communication and assure data integrity.

2.7.1 Working of kerberos

Kerberos uses Ticket-Granting Server (TGS) for connecting to users. Whenever a client want to access the service a ticket is provided with limited time for accessing the server. The figure 2.10 defines each step related to Kerberos message and network authentication. It also describes a way Single Sign-On takes place.

Figure 2.10: General process of Kerberos Ticket Exchange

http://www.zeroshell.org/kerberos/Kerberos-operation/
1. When a user requests to access a resource on a host server. The client machine requests a ticket granting ticket (TGT) for providing access to server.

2. If the user has valid credentials, the authorization server generates a ticket granting ticket and a session key. All these are encrypted and sent back to the requesting authority while requesting user his credentials. If the credentials are matched, user is authenticated to use TGT.

3. After TGT is authenticated, the user sends the request containing client name, realm name and timestamp to TGT asking to use the resource. The user also authenticates himself to TGT using password.

4. The TGT then verifies the request by decrypting ticket and authentication elements. It also and creates a tickets having client name, client IP, session time and realm. This ticket is sent back after generating two copies one encrypted using client password and other using service password.

5. The service is now authenticated using keys and after verification access is granted to the service.

6. It is also possible that mutual authentication is required, in this case the server replies with authentication message.

Kerberos server manages all the encrypted passwords and consequently configures each service to client and server. Also timestamps play a vital role in eliminating replay attacks. But, the clocks at both server and clients end should be synchronized.
Chapter 3

Requirement Analysis

Contemporary web based systems not only require federation, authentication and authorization services, but, are heavily dependent on Identity management and access control solutions for utilizing third party Identity Providers and Services Providers like Google or Microsoft. Today business requirements rapidly change and organizations always need to be flexible and adaptive to transform as per needs. In a typical business scenario, the main goal of an enterprise is to identify the obstructions involved while simplifying the processes in identity and access management solutions. These solutions endorse variety of service profiles that require exhaustive investigation before implementing them in real world environment. Although, the question to investigate different services of a particular Identity management and access control solution is not simple as it seems! For instance, a particular solution might provide Single Sign-On for Web based applications, despite another solution could provide similar facility not only for Web based applications, but also for mobile devices. These facilities could potentially impact the probe.

3.1 Comparison

In this section we provide a systematic comparison between four different prominent Identity management systems naming OpenAM from forgerock, Oracle Identity Management, NetIQ/Novell Identity Management and IBM Tivoli. It contains different characteristics of an Identity and access management system such as Free or not?, Space requirements, Directory Services, Web and mobile management etc. Table 3.1 provides an extended view of this correlation. It also explains which one is suitable for enterprises in the long run.
3.1.1 License

Using open source or free software’s definitely have its own benefits when compared to commercial or proprietary software’s. A huge benefit comes while extending the application base such as extending application with several third party applications, implementing security features, modifying source code to meet companies requirements, finding bugs etc and this is a huge benefit with OpenAM as compared to others. Although, IBM Tivoli is Proprietary, its directory server and security access manager are commercial products. Whereas OpenAM is free to use for development (Common Development and Distribution License (CDDL)i.e. free) and its paid for production use. It is highly scalable and can extend to several instances.

3.1.2 Directory Service

Directory services are the foundations of any Identity Access Management solution. They not only help in successful authentication and authorization but also with better management of identities. There are several directory service managers in market, although the best once are highly scalable, reliable and support data migration. IBM Tivoli is highly scalable, but does not support data migration from UNIX environments to Windows environments or from Windows environments to UNIX environments respectively. Whereas Oracle IAM and NetIQ does support migration and are highly reliable, but, comes with a price tag. OpenAM/ OpenDJ on the other hand support live data migration, OpenDJ is highly scalable, cross-platform directory service and can provide a better service in case of high load.

3.1.3 Identity Intelligence

Identity intelligence is comprised of mining the information for automated compliance reporting, role optimization, role management, SLA conformance, reporting and analyzing user identity. Oracle has its built in identity warehouse, which contains tools for report generation, performance analysis etc. It can also be used for importing identity data, entitlement data. But, it has huge space requirements which makes it inefficient for Small Scale Industries (SME’s). NetIQ on the other hand does not have its own role manager, it is usually deployed with third party applications for analysis. Although, this also provides a limited support to extend and modify roles, role based report generation. IBM Tivoli also does not provide an inbuilt solution for identity intelligence, but can be extended with applications such as SailPoint Compliance IQ. This might help in identifying which user has the access to what resource. OpenAM contains inbuilt modules that can be used to

\[1\text{https://en.wikipedia.org/wiki/Common_Development_and_Distribution_License}\]
### Table 3.1: Comparison of different IAM solutions

<table>
<thead>
<tr>
<th>Service</th>
<th>OpenAM</th>
<th>Oracle IAM</th>
<th>NetIQ/Novell</th>
<th>IBM Tivoli</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>License</strong></td>
<td>CDDL</td>
<td>Commercial</td>
<td>Proprietary</td>
<td>Proprietary</td>
</tr>
<tr>
<td><strong>Directory Service</strong></td>
<td>OpenDJ Active Directory,</td>
<td>Oracle Internet Directory,</td>
<td>Novell eDirectory</td>
<td>Tivoli Directory Server</td>
</tr>
<tr>
<td></td>
<td>MS Active Directory</td>
<td>Oracle DSEE</td>
<td></td>
<td>(Trial Download)</td>
</tr>
<tr>
<td><strong>Web Access Management</strong></td>
<td>OpenAM</td>
<td>Oracle Access Manager</td>
<td>NetIQ Access Manager</td>
<td>IBM Security Access Manager</td>
</tr>
<tr>
<td><strong>Federation</strong></td>
<td>OpenAM SAML based Federation</td>
<td>Identity Federation</td>
<td>NetIQ Access Manager</td>
<td>Federated Identity Manager</td>
</tr>
<tr>
<td><strong>Enterprise SSO</strong></td>
<td>OpenAM SSO Solutions</td>
<td>Oracle Enterprise SSO</td>
<td>NetIQ Secure Login(Licensed Active identity)</td>
<td>Security Access Manager for Enterprise SSO</td>
</tr>
<tr>
<td><strong>Authentication and Authorization Application Service Platform</strong></td>
<td>Web based SSO, Mobile based SSO, Application based SSO</td>
<td>Web based SSO Security Application</td>
<td>Web based SSO</td>
<td>Web based SSO</td>
</tr>
<tr>
<td><strong>Third Party Application SSO</strong></td>
<td>Yes (LifeRay, ELK analysis etc.)</td>
<td>Limited Support</td>
<td>Limited Support</td>
<td>Limited Support</td>
</tr>
<tr>
<td><strong>Memory required for proper working</strong></td>
<td>1-4 GB complete installation</td>
<td>8-16 GB Minimum</td>
<td>10 GB complete installation</td>
<td>2-5 GB minimum</td>
</tr>
<tr>
<td><strong>Identity Intelligence</strong></td>
<td>Support Identity and Contextual intelligence</td>
<td>Support Identity Intelligence</td>
<td>Does not support either</td>
<td>Does not support either</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of different IAM solutions
analyze and identify the roles and their rights. Other than Identity Intelligence, OpenAM can also be extended with Contextual Intelligence for consultation during login processes.

### 3.1.4 Space requirements

Memory has always been a critical aspect for any software distribution. If requirements are unmet, softwares could not be installed or deployed. Contemporary software distributions not only have huge space requirements but consistently require RAM for proper executions. In the current scenario, we consider different Identity access management systems and their space requirements. Oracle IAM has a huge software distribution package and consist of several SDK’s, API’s etc. The Minimum Physical Memory Required is 8GB’s and Minimum Available Memory Requirement is 16GB’s. Whereas NetIQ takes about 10GB space for its complete installation and about 7-8GB for partial installation. IBM Tivoli takes only 2-5GB’s for its complete installation, but can be extended with security manager or directory server application. OpenAM reduces the space constraint at the lowest possible level. It takes only 1-4 GB’s for complete installation including OpenDJ Directory Server.

### 3.1.5 Web Access Management

Web access management provides an efficient way to secure web resources. Various companies tailor their products according to business requirements. When implemented correctly, these web access management solutions provide an extensible adeptness to manage secured services. Oracle, IBM, NetIQ and OpenAM all offer prominent web access management services, which helps in reducing implementation cost, increase control over resources. Although, the difference arise while selecting the best product with lowest implementation. IBM, Oracle and NetIQ are very good products but come with a huge price tag. While OpenAM comes free for ForgeRock non-subscribers and can be implemented easily.

### 3.1.6 Mobile Access Management

Securing mobile applications and mobile resources are the top notch priority for any enterprise. As the world is shifting towards mobile computing more and more security is required to protect critical data. Mobile access management system can help achieve this goal. OpenAM provides mobile support using rigorous and adaptive authentication techniques such as OATH/Soft Token Generator, One Time Password (HOTP), fingerprinting etc. to enable multifactor mobile authentication and Single Sign-On. It also extends mobile profiles using OpenID connect ensuring greater interoperability and consistency to users and developers. On the other hand, Oracle IAM, NetIQ provides powerful security services for securing sensitive information and controlling
risk on mobile devices using four-digit MobileAccess pin. IBM Tivoli does not include mobile management although IBM provides a different suit i.e. IBM Endpoint Manager for Mobile device management.

3.1.7 Third Party Application Support

All the Identity Access Management applications support third party applications either in limited or unbounded manner. OpenAM diversifies its application support base with third party applications in an unrestricted manner. It supports major distributions for companies such as Microsoft SharePoint, Microsoft Drive, Google services like Drive, Docs, Calendar, Applications such as Liferay, Analytical tools such as Elasticsearch, Logstash, Kibana etc. This provides users with several options best suitable for business. Whereas Oracle IAM, IBM Tivoli and NetIQ provides limited support to third party applications. The application base could be extended using add-ons that needs further purchasing.
CHAPTER 3. REQUIREMENT ANALYSIS

3.2 Why OpenAM?

In the case study conducted by Zalando, provide insightful facts about Forgerock OpenAM and OpenDJ. Zalando implemented Single Sign-On and user management services for its internal employees using OAuth2.0 protocol. The main motive was connect Amazon Web Services (AWS) to a single management portal that could serve all its applications (Single Sign-On) without modification. OpenAM worked as a charm and helped them to achieve this goal. They also extended it to unify user identities in OpenDJ server.

In a broader scenario, all the current internal services running in Zalando such as Authorization, Single Sign-On, internal Web Services Security etc. is governed using OpenAM "All-In-One" access management portal. This also comprehends the case study stating about the firmness, performance and security that OpenAM and OpenDJ can provide.

Moreover OpenAM has modular architecture, it is cross platform or Operating System (OS) Independent and can be used with every major operating system supporting Java. It is easy to install, configure, test and deploy over production environment. If ran into problems, ForgeRock Community provides number of developers that could assist in resolving deployment and runtime issues.

Additionally as an open source product by ForgeRock, OpenAM can additionally extend its domain by implementing OpenIG, OpenIDM with it. This could also result in additional functionality while configuring identity and access management.

---

2http://www.slideshare.net/ForgeRock/zalando-case-study-the-big-switch-rewiring-zalandos-digital-trade-routes
3https://www.identitymethods.co.uk/solution-partners/forgerock
3.2. WHY OPENAM?

Figure 3.1: Forgerock Products
Chapter 4

Concept

This chapter scrutinize majority of use-cases that are possible using OpenAM Access Management solution. It contributes in comprehending the intention of this dissertation by establishing a notion. The chapter also focuses on showcasing OpenAM with Shibboleth IdP and Shibboleth SP, OpenAM with Windows kerberos Single Sign-On, OpenAM as an IdP and SP etc. This foregrounds OpenAM as an important solution that could benefit GWDG in longer run.

4.1 Multi-Protocol Federations and Single Sign-On

Collaboration between multiple protocol federation technologies such as Shibboleth, SAML, OAuth etc. has always been strenuous in organizations. Due to which linking person’s electronic identity, authenticating users across organizations, centralization of identity management, cross-company and cross-domain access etc. using single management portal was a challenge. OpenAM resolves all these challenges by providing an “all-in-one” access management portal that bridges several heterogeneous systems. It can be used for Single Sign-On solutions across federations, implementing policies, agents etc.

In this section we consider diversified approach in which OpenAM could be configured and operate.

4.1.1 Achieving OpenAM SSO with SAML 2.0

Based on the requirement section, here we discuss several use cases that are required for multi-protocol authentication and SSO.
SSO with OpenAM as IdP and Shibboleth as SP

To link different federated solutions, OpenAM relies heavily on Security Assertion Markup Language (SAML) 2.0. Which illustrates how messages are relayed and information is passed in cross-domain and cross-company environment. It is configured in OpenAM by indicating circle of trust, which provides a way to connect identity provider and service provider.

1. Identity Provider store user information i.e. credentials, profiles etc. and is used for authentication purposes.

2. Service Provider is used for protecting applications, data or resources. When service is requested service provider handles authentication by redirecting it to identity provider for user verification.

3. Circle of Trust (COT) is a group in which providers (IdP and SP) are configured and metadata is shared using assertions. This facilitates identity provider and service provider to share information.

4. SAML 2.0 achieves SSO by mapping user identity attributes from identity provider to service provider. After verification service provider gains authority to grant resource access.

The figure 4.1 below describes a test case when OpenAM acts as an Identity Provider and Shibboleth as a Service Provider.

![Figure 4.1: SSO using OpenAM as IDP and Shibboleth as SP](image-url)
GWDG as a data center can use OpenAM for configuring multiple service providers in single management portal. It will prescribe an easier way to implement Single Sign-On solution where GWDG users can directly access all secured resources from different service providers without having to login individually.

Figure 4.1 can be described as, a GWDG user attempts to log into a protected application. Service Provider responds the request by redirecting to identity provider (OpenAM) for authentication. Identity Provider (OpenAM) provides the credentials stored in its active directory to service provider, which than maps the attributes (credentials, profiles) and grants authentication. After successful authentication, an authorization request is generated by service provider to determine whether to grant or to deny user access. OpenAM acts as a policy administrator which helps in determining access rights. After successful resolution, user is granted or denied access respectively. It can also be described by an example:

1. User wants to access services (say webconf.dfn-aai) protected by Shibboleth Service Provider and GWDG Cloud Share on gwdg server without using credentials for both services.
2. GWDG can create Hosted Identity provider and Hosted Service Provider in OpenAM console. Hosted Identity Provider implements OpenAM as IdP and Hosted Service Provider as Shibboleth SP.
3. Establishing these in one circle of trust allows both service provider’s and identity provider to exchange attributes and map them to authenticate and authorize the user.

**SSO with OpenAM and Shibboleth as IDP and SP**

Another suitable test case, where OpenAM acts as a proxy while Shibboleth IdP and SP interacts to authenticate the user. Usually Shibboleth IdP and SP are connected to provide better secured services. Although the complexity arises when configuring multiple IdP’s with multiple SP’s as metadata of SP’s configuration file is different in each case. Using OpenAM this problem can be easily resolved as service provider needs to change its configuration file according to OpenAM requirements. It automatically resolves which IdP is associated to the user service and redirects it to IDP.

The complete flow for Figure 4.2 is illustrated below.

1. A data center
2. User wants to access the resource secured by Shibboleth service provider.
3. User wants to access services (say webconf.dfn-aai) protected by Shibboleth Service Provider and GWDG Cloud Share on gwdg server without using credentials for both services.
4. GWDG can create Hosted Identity provider and Hosted Service Provider in OpenAM console. Hosted Identity Provider implements OpenAM as IdP and Hosted Service Provider as
Shibboleth SP.

5. Establishing these in one circle of trust allows both service provider's and identity provider to exchange attributes and map them to authenticate and authorize the user.

Figure 4.2: SSO using OpenAM as IDP and Shibboleth as SP
**SSO with OpenAM as IdP and OpenAM as SP**

In this scenario, let's assume one instance of OpenAM (at port 8080) acting as an identity provider and is used to protect GWDG dashboard (example, portal.gwdg.com). It uses certain credentials (username and password) to log into the application. Whereas another instance on OpenAM (at port 8081) protects issuer application (example, issue.gwdg.com) and acts as service provider. It is assumed IdP has configured a SP issuer link in its dashboard.

The complete flow is as follows:

1. When a user logs into GWDG dashboard using credentials.
2. A link for SP application i.e. issue.gwdg.de is already available in GWDG dashboard. After clicking the link user is redirected to the SP application login page using SAML and HTTP protocols.
3. Provided valid credentials, SP application with user pranay is linked to GWDG dashboard.

4. The SP application redirects the user to protected resource.

Steps 2, 3 and 4 occurs only one time to link user account. After which the application can be accessed easily without authenticating in issuer app every time for same user. This is highly beneficial in GWDG, as several OpenAM instances could be established to protect resources. Moreover, these can be secured from unauthorized access by creating policies individually for each resource.

**Load Balancing with OpenAM as IdP and OpenAM/Shibboleth as SP**

To provide better service availability during heavy traffic, a load balancer is installed behind firewall to disperse traffic at peak time. This increases overall throughput, maximize response time, optimize resources and avoid overloading while preventing server failure. While configuring OpenAM for session failure, users information is stored in database for hardware or software failure. OpenAM during session failure interacts with message broker which manage session state in database. During failover the complete state is saved by OpenAM i.e. users data is unharmed ensuring complete session recovery with user customized settings. Although, one load balancer could be responsible for single point of failure.
Figure 4.5: OpenSSO (OpenAM) Load Balancer in a Single Site with Session Failover

The above figure represents multiple OpenSSO (OpenAM) instances on a single site and connected by load balancer. When service request comes, load balancer checks for least utilized server and redirects the request to particular OpenAM instance, which is then processed respectively.

As a data center, GWDG should provide 24*7 uninterrupted services and implementing this method with multiple OpenAM instances and load balancers on several locations could help in achieving this goal in future.
OpenAM Session Failover

OpenAM provides High availability sessions to recover sessions during failovers caused due to calamities, mishaps etc. To manage session failover OpenAM configures OpenAM site where multiple OpenAM instances could be configured and accessed using load balancers. After fulfilling requirements OpenAM site is configured for session failovers. These failovers are heavily dependent shared Core Token Service (CTS) to store user sessions and settings. OpenAM share all the information with other instances and when a failure occurs other running instances can easily access all the information using CTS.

Figure below explains connection between different OpenAM instances and transaction between them. It also describes OpenDJ interaction to exchange data and replicate itself instantaneously with other instances. In session failure client application will be redirected to active OpenAM instance which will server all requests until other OpenAM server becomes available.

After implementing OpenAM at GWDG, session failover can be configured for providing higher availability and reliability. User requests can be managed by OpenAM access management portal automatically for number of services.
4.1.2 Achieving OpenAM SSO with OAuth 2.0

Single Sign-On using OAuth 2.0 in OpenAM can be achieved by several ways. In the first scenario, figure 4.2 shows OpenAM as an OAuth 2.0 provider and request handler. Whereas in second scenario, figure 4.3 depicts OpenAM as a request handler and authorization corroborator.

OpenAM as OAuth 2.0 provider and request handler

Figure 4.2 explains how OAuth authorization process takes place in OpenAM. Where it acts as a management portal and an OAuth 2.0 provider. When an access request flows from client-end towards protected resource, the protected application requests authorization token from Authorization Server i.e. OpenAM. The detailed explanation for the process is as follows:

1. The client tries to access the protected resource saved on resource server. The protected resource verifies if the client is authorized for access? It responds asking Client for a valid authorization token provided by OAuth 2.0 authorization server.
2. The clients sends request via browser to authorization server. Which validates clients request and verifies the protected resource.
3. Authorization server inquires protected resource about the request and queries resource owner to grant permission for the resource.

Figure 4.7: Authorization code flow when OpenAM act as OAuth 2.0 provider
4.1. MULTI-PROTOCOL FEDERATIONS AND SINGLE SIGN-ON

4. After getting the permission from user, the OpenAM OAuth 2.0 authorization server sends a code informing client that it has been authorized to access the resource.

5. The client responds the information by asking authorization server for an access token to the code user authorized. The client provides credentials and interacts directly with authorization server. The OAuth provider verifies the credentials of client and the user.

6. After verifying credentials an access token is generated and sent to the client.

7. The client using access token gains access to protected resource.

OpenAM when acting as OAuth 2.0 provider authenticates resource owners and obtains authorization credentials in order to provide access tokens to clients. All these clients can be registered in OpenAM console with policy agents.

Use Case I In a typical web based scenario, user is first authenticated by OpenAM and then authorization request is sent back to OpenAM, which validates by confirming governed policies and access rights. If user has valid permissions, OpenAM responds with session token and user is authorized for access.

After implementing OpenAM, GWDG can configure OAuth 2.0 provider service through which GWDG can be an OAuth service provider ————————
### OpenAM as service request handler

**Figure 4.8: Code flow when OpenAM act as service request handler**

1. The user requests to access the secured application by passing authentication request through OAuth 2.0 module.

2. OpenAM redirects the request via browser to Authorization service at OAuth 2.0 provider. The request contains several parameters that were configured while defining module i.e client_id etc.

3. After receiving the request from OpenAM, OAuth provider verifies if user has already granted permission for access? and if the user is authenticated with authorization provider?

4. After OpenAM gets permission to access the resource, OAuth provider sends authorization code to OpenAM using browser redirection.

5. As authorization code is received, OpenAM requests authorization provider to provide access token by providing client_id, client_secret etc. The provider returns the request with access token.

6. OpenAM accesses user information such as email_id, educational information etc. and map it to local system (if configured).

7. The authentication and authorization is successful and SSO is achieved, as several service providers can be connected using this method.
**4.1. MULTI-PROTOCOL FEDERATIONS AND SINGLE SIGN-ON**

**Use Case 1 - Mapping user info. with OpenAM local profile**  
User or identity mapping in OpenAM is either configured using default plugins or implemented by creating new plugins. In this use case, assume GWDG configures OAuth 2.0 authentication service in OpenAM portal and a Google service user having username and password, attempts to login to GWDG provided service (example owncloud.gwdg.com).

The complete flow is illustrated below:

1. A Google service user is already authenticated in google services.
2. When user attempts to login GWDG service, OpenAM acquires authorization code or access token.
3. Using access token and valid permission from user, it extracts attributes from Google.
4. These attributes are then mapped to local data store in OpenAM.
5. All these attributes i.e. from Google and from Local data store are pushed towards protected application.
6. If the mapping and authorization is valid. OpenAM grants access to protected application.

Identity mapping would validate user information in GWDG’s own system. If user is not found, access is not granted to protected application.
Use Case II - Provisioning OpenAM SSO  Another use case enables user to be created automatically by gaining information from OAuth 2.0 provider. This can be implemented using two techniques. First, to authenticate user by sending an activation code in email account and secondly, by not informing user and directly creating an account.

1. User is already authenticated in google services.
2. When he/she attempts to login GWDG service, OpenAM acquires authorization code or access token
3. Using access token and valid permission from user, it extracts attributes from Google
4. OpenAM verifies if the user already has an account or not?
5. If account is not available, a user account is created using user information from OAuth 2.0 provider.
6. After account is created, user is granted access to protected resource

If GWDG wants to extend its services to third party users or users that are not directly linked to GWDG, this use case could enable them to do so. Users could use services like GWDG OwnCloud or GWDG CloudShare etc. to share data.
Use Case III - Accessing user information without storing in local Data store  
OpenAM also facilitates administrators to temporarily use user information pulled from OAuth 2.0 provider. These attributes are present only in browser cookies or session tokens and are destroyed after the time has expired.

1. User authenticates in Google services.
2. After signing in GWDG service, OpenAM acquires authorization code or access token to pull attributes
3. Using access token and valid permission from user, it extracts attributes from Google
4. The extracted information is stored in browser session and is used when user access the services.

This use case attempts to solve an issue related to authorizing users indirectly connected to GWDG. As GWDG can provide temporary access for users with valid credentials to protected information.
4.1.3 SSO using WindowsDesktopSSO module in OpenAM

OpenAM could be configured with WindowsDesktopSSO using Kerberos Distribution Center (KDC) or Kerberos Active Directory (KAD). It can be deployed for internal as well as external users with an extension of Single Sign-On facility. Although for security reason external users are provisioned with limited access. Figure 4.12 elucidates the working of WindowsDesktopSSO with OpenAM. When a user attempts to access the web application protected by policies. He is redirected to OpenAM that is attached with LDAP/ KDC for authentication and authorization. Using Kerberos tickets, OpenAM authenticates the client with KDC and responds back with successful authentication. The entire authentication process is discussed as follows:
4.1. MULTI-PROTOCOL FEDERATIONS AND SINGLE SIGN-ON

1. Let's assume a Windows user, which is already authenticated using Windows Active Directory attempts to retrieve an application protected with OpenAM policy agents.

2. Policy agents prevent the request and verify whether the user is authenticated in OpenAM?

3. If the user is authenticated, he is given access to the application else the user is redirected to OpenAM via browser and is stipulated to use WindowsDesktopSSO module for authentication purposes. The OpenAM WindowsDesktopSSO Authentication module is a server-side SPNEGO protocol implementation that uses the Java GSS-API to process a Kerberos token sent by a SPNEGO-supported browser.

4. OpenAM later checks for valid SSOToken in the browser verifying the user is authenticated or not. If the user is not authenticated (authenticate: NEGOTIATE) is sent using SPNEGO protocol. If the browser is not configured for handling integrated Windows authentication requests (SPNEGO), 401 ERROR will be displayed in the browser.

5. The user negotiates a Kerberos service ticket using TGT (Ticket granting ticket) with KDC.
6. Browser obtains the service ticket (TGT) generated during authentication process and returns to OpenAM.

7. OpenAM decrypts the TGT and authenticates the user using kerberos decrypted ticket.

8. After successful authentication with KDC, response is sent back with principle name

9. The user is redirected back to policy protected application with SSOToken provided by OpenAM

10. The browser again generates request to access resource

11. Policy agents verifies and validates SSOToken in browser cookie. The results are stored in browser cache.

12. These results are used by Policy agents while authorizing the user. Although, OpenAM evaluates policy requests to validate authorization request of user. The evaluation is again cached in policy agent

13. These results are passed to application

14. Application renders the page and responds to user browser’s request.

Almost all Windows Desktop computers in University of Goettingen, Max Planck Institutes are governed by GWDG. Using WindowsDesktopSSO module in OpenAM can facilitate authenticated users to enable Single Sign-On facility. Users logged into Windows PC’s can easily access all the GWDG services without having to use their credentials for each service. This will reduce the time spent on individual login processes and accelerate the productivity of users. Few use cases are as follows:

Remote Access

Windows kerberos authentication can be used in GWDG to allow user access from outside university departments. Kerberos uses Secured Socket Layer/ Secured Shell (SSL/SSH) for this scenario. Implementing with OpenAM will provide an extra protecting layer to assist Single Sign-On.

Queries by Unauthorized and Authorized users

Unauthorized Searches  Suppose user A isn’t a member of GWDG team, but interested in internal product information. User queries in browser and the browser responds by providing the information. Although before serving the user by opening content, search utility verifies if user has valid credentials after which it validates the kerberos ticket from Kerberos Key Distribution Center (KDC) or network domain controller (NDC). Due to valid authentication in kerberos ticket,
4.1. MULTI-PROTOCOL FEDERATIONS AND SINGLE SIGN-ON

user is not provided with a login page and is authenticated through Kerberos. Although, due to lack of access rights, he is served with "HTTP status 401" (Unauthorized Access page).

**Authorized Searches**  This process is similar to unauthorized search besides in this case User has access rights. The user is not served with login page, as its authenticated using kerberos ticket. This ticket is sent as HTTPS response to search engine and is interpreted as HTTP HEAD request to search and list all documents. User is than server with the document.

**Useful for researchers**

WindowsDesktopSSO with Kerberos is extensively used in research laboratories for authenticating users. Many high energy physics experiments require access to distributed file systems such as AFS. Using OpenAM WindowsDesktopSSO module can facilitate researchers for easier access.

**4.1.4 SSO decisions in OpenAM using Policy Agents**

Policy agents are used to enforce policies, control access to protected resource and authenticate users by redirecting them to OpenAM. These are very small agents that interpret user requests and are installed on application running servers. After installation these agents could be centrally managed and configured to protect web applications.

Policy agents are of several types, but in our scenario we focus of Web agents and J2EE policy agents. Both works almost the same way but are configured on different servers.

**Working of Policy Agent**

When a user tries to use protected resource configured with OpenAM via browser. The web agents checks for valid session token in browser, if token is present user is allowed to access the resource else the request is passed to OpenAM authentication page. After valid authentication, a session token is generated and set in users browser. The response is sent back to web policy agent, which again checks for valid token. After receiving the token, OpenAM verifies if the authenticated user is authorized to access the protected application. The request is forwarded to OpenAM policy management service, which determines if the user have access rights. If user has access rights, OpenAM responds to web agent for allowing user access and web policy agent responds user request with the browser page he requested.
The above figure depicts the complete flow of instructions between OpenAM and Web policy agent.

1. The user (Client) requests to access the protected application.
2. OpenAM determines and manages policy configurations required for proper accessing resource. It is connected to policy agent installed on web server.
3. OpenAM interacts with policy agents instructing to enforce policies on protected resource.
4. After proper authentication and authorization OpenAM and policy agent allows access to protected resource.
5. The web browser responds the request by displaying requested resource

Single Sign-On is used with policy agents by configuring cookie properties under Web > Agent Name > SSO and can be configured for single domain and cross domain platforms.
Chapter 5

Configuration and Implementation

This chapter describes the method to implement each designed system in organization. Some of implemented solutions have some guidelines that were used to configure them. The figure below depicts existing work shown in solid lines, whereas dotted lines show implemented work. It exhibits the connection established between OpenAM and Shibboleth, OpenAM and OpenAM, Windows and OpenAM using connection bridge. The chapter also covers other SSO implementations that are integral part of OpenAM access management portal such as SSO with Policy agents, SSO with desktop applications, SSO with fedlets.

5.1 Connection Bridge/ Gateway

The communication bridge provides an easy and successful path to connect multiple protocol implementations such as Shibboleth and OpenAM, OpenAM and OpenAM etc. To implement this bridge, both OpenAM and Shibboleth are configured and extended to perform interactions. In each scenario, IdP metadata, SP standard and extended metadata are to be configured. This develops a standard way to communicate between different protocols and provide a standard mechanism to exchange identity attributes.

5.1.1 System Configuration

Besides the complexity of communication bridge, the basic system architect used for implementation is provided in Table. The testing was finished using the configuration below and is the minimum recommendation for better performance.
Figure 5.1: Working of Policy agents with OpenAM

<table>
<thead>
<tr>
<th>Tools Used</th>
<th>System</th>
<th>Version</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>VirtualBox</td>
<td>Oracle</td>
<td>4.3.28</td>
<td>Installing Operating systems</td>
</tr>
<tr>
<td>OpenAM</td>
<td>ForgeRock</td>
<td>12.0</td>
<td>Implementing Access management and SSO</td>
</tr>
<tr>
<td>OpenDJ</td>
<td>ForgeRock</td>
<td>2.6.0</td>
<td>Directory Server</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>OpenLDAP</td>
<td>2.4.41</td>
<td>Directory Server</td>
</tr>
<tr>
<td>SAML</td>
<td>OASIS</td>
<td>2.0</td>
<td>protocol used to exchange authentication and authorization</td>
</tr>
<tr>
<td>OAuth</td>
<td>Google API</td>
<td>2.0</td>
<td>Authorization protocol</td>
</tr>
<tr>
<td>Tomcat Server</td>
<td>Apache</td>
<td>7.0</td>
<td>Run OpenAM, Shibboleth IdP and SP, Policy agents</td>
</tr>
<tr>
<td>Web Server</td>
<td>Apache</td>
<td>2.4</td>
<td>HTTP server for hosting services</td>
</tr>
<tr>
<td>Kerberos</td>
<td>MIT</td>
<td>5.1</td>
<td>Authentication Protocol</td>
</tr>
</tbody>
</table>

Table 5.1: System configuration
5.1. CONNECTION BRIDGE/ GATEWAY

5.1.2 Configuring Communication Bridge

The subsection provides a comprehensive overview to develop communication bridge and suggests different modifications that were used for successful communication between multi-protocols federations. This enables exchanging user attributes (username, password etc.) in OpenAM with other protocols without frequently modifying Service providers metadata and Identity providers metadata. The access management system doesn't require any modifications in its configurations as it interacts using metadata files. Although, the service providers and identity providers need to alter their configurations to meet OpenAM configuration demands.

Configuration for bridging between OpenAM IdP and Shibboleth SP

Shibboleth service provider communicates with OpenAM by sharing its metadata file, mapping attributes and customizing Shibboleth2.xml file.

Shibboleth2.xml configuration in Shibboleth SP  The Shibboleth2.xml inside the Shibboleth SP source code is modified to facilitate systematic event logging, service chaining etc. The native.logger maps IIS instance ID to hosts name/ports, which is relevant for mapping exact <Host>.

```xml
<!-- Example of Shibboleth2.xml configuration snippet -->

<Site id="1" name="sp.thesisshib.com"/>
```

Table 5.2: Code snippet for native.logger

In native.logger function, matches IIS sitename or apache servername/port to detect proper <Host> name/port. Similarly, service Provider maps all incoming requests on <Host> using <RequestMapper> plugin to set of configuration options. In our scenario, the plugin is modified to map each request to /secure application. Here, /secure contains secured webpage, which can be accessed after proper authentication and authorization using HTTP or HTTPS protocols.

Secondly, <RequestMapper> relatively maps all incoming resource requests by pointing them to default Application ID in ApplicationDefaults section.

But to assist <RequestMapper> plugin with an exact application URL, each service provider have its own handlerURL, which effectively redirects the application request to respective application. The <Handler> element in handlerURL is used to implement a SSO protocol, it also issues SSO requests, deals with logout, it controls session lifetime, address checks and cookie handling. Moreover, it is defined using HTTP but for secure communications HTTPS can be used and handlerURL could be set to true (handlerSSL="true").

1https://blogs.oracle.com/identityfever/entry/connect_opensso_as_a_shibboleth
CHAPTER 5. CONFIGURATION AND IMPLEMENTATION

Table 5.3: Code snippet for Shibboleth SP <RequestMapper> plugin and ApplicationDefaults section

Table 5.4: Code snippet defines SPNativeApplication handler

To handle URL requests, <SessionInitiator> elements play an important role in initiating authentication process. <SessionInitiator> also establish, handles sessions and relays them to IdP or Discovery page. This process is referred to as "SP-Initiated" flow, where browser redirects to IdP for login and then returning back with login information.

In the table below, Shib1 SessionInitiator and SAML2 SessionInitiator (Protocol Handler) i.e. type="SAML2", which supports SAML 2.0 authentication requests and type="Shib1" provides support for Shibboleth 1.x authentications. SAML2 SessionInitiator help protocol handler to check metadata with an <md:IDPSSODescriptor> role by detecting entityId. Whereas, entityId should be known to protocol handler for checking metadata with <md:IDPSSODescriptor> role as Shib1 SessionInitiator.

Shib1 and SAML2 are required as their absence can cause a logged warnings or denial of requests.

Table 5.5: SPSessionInitiator handler snippet

Another typical embedded authentication request can be defined by example below.
Besides chaining the application, application consists of certain rules that cannot be used as defined by defaults. For this purpose, `<applicationoverride>` is used to override default settings.

```
<ApplicationOverride id="sp.thesisshib.com" entityID="http://sp.thesisshib.com/shibboleth"/>
```

Table 5.7: ApplicationOverride settings

Shibboleth2.xml file locally maintains metadata for SAML2.0 connection. The metadata file contains EntityDescriptor, IDPSSODescriptor, KeyDescriptor, certificates etc. It also uses attribute-map.xml file to extract attributes from SAML assertions.

```
<MetadataProvider type="XML" file="idp.xml"/>
```

Table 5.8: Settings to store Metadata locally

**attribute-map.xml configuration in Shibboleth SP**  
Service provider by default ignores users information provided by identity provider until it is specifically specified by administrator what and how to process. This can be configured in service provider in order to recognize new application attribute mappings. 
Service provider uses “attribute extractor” to receive attributes in realtime from SAML assertions via attribute-map.xml file. This file contains certain rules to create new mapping functions and refer them for "on the wire" connections.
CHAPTER 5. CONFIGURATION AND IMPLEMENTATION

Table 5.9: Single lined mapping attribute

OpenAM configuration for forming communication gateway  First, we start configuring Hosted Identity Provider in OpenAM admin dashboard.

1. Open browser and navigate to http://sp.thesisshib.com:8080/OpenAM-12.0.0.
2. Login as an administrator (amAdmin - by default) in OpenAM web console.
3. The main screen consists of several tabs. Click on "Common tasks" and choose "Create Hosted Identity Provider".
4. Following screen appears on screen.

(a) Realm : Select default "test".
(b) Name : Identity provider’s unique name/URL (http://sp.thesisshib.com:8080/OpenAM-12.0.0).
(c) Circle of trust (COT) : It is important to provide a name, as identity provider and service provider communicates with each other within same circle of trust.
(d) Attribute mapping : Both identity provider and service provider might have different
schema but similar information to store in active directory or data store. In this section, we can define which attributes are to be mapped example in IdP mobile information is saved as mobile_no, whereas in SP it is saved as cellphone_no. These can be easily configured for mapping.

5. After finishing, click on Configure and then Finish.

Similarly, Configure Remote Service Provider in amAdmin console.

1. We can either provide a URL to metadata file (https://sp.thesisshib.com/Shibboleth.sso/Metadata) or provide the location to file.
2. Attribute mapping are set to empty in this case.
3. Click configure and go to Federation tab and then Entity Providers.
4. In Assertion Processing add the configuration.

```
urn:oasis:names:tc:SAML:2.0:attrname-format:uri
urn:oid:0.9.2342.19200300.100.1.1=uid
```

Table 5.10: Assertion Processing value in remote Federation tab

Configuration for bridging between OpenAM IdP and OpenAM SP

To achieve OpenAM IdP and OpenAM SP connection, we configure both instances to coordinate with each other. The configuration for Host Identity Provider is same for OpenAM IdP as in Table above and for OpenAM SP with some minor changes. After Hosted IdP and Remote IdP are configured. We configure Hosted Service Provider and Remote Service Provider.

OpenAM configuration for forming communication gateway

1. Open browser and navigate to http://sp.thesisshib.com:80811/OpenAM-12.0.0.
2. Login as an administrator (amAdmin - by default) in OpenAM web console.
3. The main screen consists of several tabs. Click on "Common tasks" and choose "Create Hosted Service Provider" from "Create SAMLv2 Providers".

Table 5.11: Metadata format for remote identity provider

SP URL : http://sp.thesisshib.com:80811/OpenAM-12.0.0
URL metadata format : https://sp.thesisshib.com/saml2/jsp/exportmetadata.jsp

CHAPTER 5. CONFIGURATION AND IMPLEMENTATION

(a) Name: Provide OpenAM SP instance URL.
(b) COT: Select same as for Identity provider.
(c) Use default mapping attribute checked.

4. Click Configure.

Configuration for Register Remote Service Provider:

1. Open browser and navigate to http://sp.thesisshib.com:8080/OpenAM-12.0.0.
2. Login as an administrator (amAdmin - by default) in OpenAM web console.
3. The main screen consists of several tabs. Click on "Common tasks" and choose "Register Remote Service Provider" from "Create SAMLv2 Providers".
   (a) Realm: Provide OpenAM SP instance URL.
   (b) Metadata URL: Select same as for Identity provider.
   (c) COT : Use default mapping attribute checked.
4. Click Configure.

Multiple service providers can be added in similar manner and to edit configuration.

1. Open browser and navigate to SP or IdP console.
2. Login as an administrator (amAdmin - by default) in OpenAM web console.
3. Goto Federations tab
4. You can see in the figure below, how COT, IdP and SP is defined.

![Figure 5.3: OpenAM create Hosted IdP console](image)

5. Click entity for modification.
Commutionican between Windows kerberos and OpenAM

OpenAM provides WindowsDesktopSSO authentication module, which is based on Kerberos plug-in to achieve Single Sign-On. It enables windows users that are already authenticate with Kerberos Distribution Center (KDC) to authenticate in OpenAM, without using their credentials. This subsubsection provides details to configure OpenAM WindowsDesktopSSO Authentication module, the Kerberos domain controller, and Windows Active Directory using GSS-API Negotiation (SPNEGO) protocol.

Components WindowsDesktopSSO Authentication  The following shows the basic system configuration needed to achieve WindowsDesktopSSO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Product Name</th>
<th>Platform</th>
<th>Hostname</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenAM Server</td>
<td>ForgeRock</td>
<td>ubuntu</td>
<td>sp.thesisshib.com</td>
</tr>
<tr>
<td>Kerberos Domain Controller (KDC)</td>
<td>NA</td>
<td>Ubuntu</td>
<td>kerberos.thesis.com</td>
</tr>
</tbody>
</table>

Table 5.12: Components used for configuring WindowsDesktopSSO

Configuring WindowsDesktopSSO Authentication in Ubuntu  After installing Kerberos server in ubuntu, we need to change some configuration in order to connect kerberos with OpenAM and communicate with Windows Server.

For authentication, Kerberos key distribution center (KDC) uses tickets, which assists kerberos domain controller to recognize requests. Although, to recognize KDC requests Kerberos Domain Controller should be running on Linux/Unix based system or could run on Windows machines. Microsoft Windows Active Directory and Windows Domain Controller are used on Windows machine to support Kerberos Domain Controller.

1. To accept requests, we modify Kerberos Domain Controller in our krb5.conf file. Kerberos krb5.conf file contains configuration information such as admin servers for realms, KDC’s location, default config for Kerberos applications, mapping of hostnames onto Kerberos realms etc. A sample file is shown in the figure below.

---

4https://wikis.forgerock.org/confluence/display/openam/How+does+OpenAM+work+with+Windows+Desktop+SSO
CHAPTER 5. CONFIGURATION AND IMPLEMENTATION

[logging]
default = FILE:/var/log/krb5libs.log
dc = FILE:/var/log/krb5kdc.log
admin_server = FILE:/var/log/kadmind.log
[libdefaults]
dns_lookup_realm = false
dns_lookup_kdc = false
default_keytab_name = /etc/krb5/krb5.keytab
default_realm = DEMO.IDENTITY.COM
default_tkt_enctypes = des-cbc-md5
default_tgs_enctypes = des-cbc-md5
default_checksum = rsa-md5
kdc_timesync = 0
kdc_default_options = 0x40000010
clockskew = 300
check_delegation = 0
ccache_type = 3
kdc_timeout = 60000
[realms]
DEMO.IDENTITY.COM = {
dc = demo1.identity.com:88
admin_server = demo1.identity.com:749
default_domain = identity.com
}
[domain_realm]
.identity.com = DEMO.IDENTITY.COM
identity.com = DEMO.IDENTITY.COM
[appdefaults]
pam = {
debug = true
ticket_lifetime = 36000
renew_lifetime = 36000d
forwardable = true
krb4_convert = false
}

Figure 5.4: krb5 configuration

2. After modifying krb5.conf, we configure kdc.conf file which contains KDC configuration information and the defaults that are used while issuing Kerberos tickets. These are later used for authentication purposes inside companies intranet.
5.1. CONNECTION BRIDGE/ GATEWAY

3. Configuring krb and kdc conf files helps to connect with Kerberos Domain Controller. As a first step, we generate a Kerberos Domain Controller database using kdb5_util command.

```
# /etc/kd5/N kdb5_util create -s
Initializing database '/var/krb5/principal' for realm 'DEMO.IDENTITY.COM',
master_key name 'K/AMSSD_IDENTITY.COM'.
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:
Re-enter KDC database master key to verify:
```

4. At last we create service principals and keytab

```
# kadmin.local: addprinc -randkey HTTP/unserver.identity.com
WARNING: no policy specified for HTTP/unserver.identity.com#DEMO.IDENTITY.COM;
defaulting to no policy Principal
"HTTP/unserver.identity.com# DEMO.IDENTITY.COM" created.
```

Implementing Policy Agents in OpenAM

This section provide basic details about configuring and implementation Policy Agents in OpenAM.

1. Global properties can be easily configured in OpenAM console under Access Control > Realm Name > Agents > Web > Agent Name > Global tab
   (a) Profile properties are as follows
   (b) Group : We can easily assign policies for groups i.e. assigning group for already configured web agent.

5http://docs.forgerock.org/en/openam/12.0.0/admin-guide/index/chap-agents.html
(c) Password Agent password is used for accessing policies

(d) Agent Configuration Change Notification: If OpenAM configuration changes, it detects and notifies it to administrator.
   Property: com.sun.identity.agents.config.change.notification.enable

(e) Agent Notification URL: Notification listeners are used to detect URL’s of user

(f) Agent Root URL for CDSSO: It is the root URL for single sign on. It uses protocols such as http and https for secured communication.

(a) General properties are as follows

(b) SSO Mode Only: The web agent enforces authentication (Single Sign On), without any policies.
   Property: com.sun.identity.agents.config.sso.only

(c) Resources Access Denied URL: A URL is defined for forbidden communications.
   Property: com.sun.identity.agents.config.access.denied.url

The policy agents have several properties such as policy agent filter, Java Authentication and Authorization Server (JAAS) realm, agent application war file, agent installer, and client SDK.

Using these properties Policy agents could be easily configured.

---

6http://docs.forgerock.org/en/openam/12.0.0/admin-guide/index/chap-agents.html
Chapter 6

Result

The chapter focuses on discussing result of chapter 7 (Configuration and Implementation).

6.1 Outcome of bridging between OpenAM IdP and Shibboleth SP

Here we discuss about the backend process that undergoes when the communication between OpenAM IdP and Shibboleth occurs. The complete process is follows:

1. When a user tries to access the secured resource protected by Shibboleth SP (sp.thesisshib.com/secure).

2. The browser attempts to connect the resource request to service provider.

3. Service provider redirects the request using ( /SSORedirect ) via browser to identity provider.

The request contains certificate, relay request, hash value etc.
After successful redirection, the browser fetches login page information example locales, serverinfo etc. It loads UI values such as libraries, realms, forms etc. from the server and translates them into defined language using HTTP.

If user is not authenticated and makes an attempt to authenticate himself by providing credentials, session and authentication request is sent to IdP containing default realm, entityID, requestID etc. The browser checks user in session using id and if the session is not available for any user request is forwarded to authenticate and get session.

The credentials provided by user are verified in OpenAM IdP data store and tries to authenticate. The OpenAM datastore is an embedded or external OpenDJ directory server, which stores user identity information such as profiles, credentials, login-logout information etc. When request is made to OpenAM IdP, it fetch user information from OpenDJ server and verify the credentials submitted by user in login form with the stored values. This GET request shows an attempt to authenticate the user using data store.
6.2 Outcome of bridging between OpenAM IdP and OpenAM SP

In this section we cover how a user is authenticated while using Single Sign-On service. We go through the fundamental backend process which takes place via browser for communication between OpenAM IdP and OpenAM SP for SSO.

1. When a user tries to access the secured resource protected by OpenAM SP and use SSO. The link communicates between one instance of OpenAM i.e. IDP at port no. 8082 and SP at port
CHAPTER 6. RESULT

8083.

http://openam.thesissib.net:8083/OpenAM-12.0.0/saml2.jsp;sp/SSOInit.jsp?
metaAlias=/sp&dp=EntityID=http://openam.thesissib.com:8082/OpenAM-12.0.0

Figure 6.8: Tomcat log: HTTP GET resource information

2. The browser redirects SSO session requests from service provider to identity provider. The redirects and requests response from IdP using SAML redirect/response.

3. After successful redirection, the browser fetches login page containing UI, locales, serverinfo etc. It loads GUI values such as libraries, realms, forms etc. from the server and translates them into defined language using HTTP.

4. If user is not authenticated and makes an attempt to authenticate himself by providing credentials, session and authentication request is sent to IdP containing default realm, entityID, requestID etc. The browser checks user in session using id and if the session is not available for any user request is forwarded to authenticate and get session. These POST messages

Figure 6.9: Tomcat log: HTTP GET requests information from IdP

Figure 6.10: Tomcat log: HTTP GET request to fetch data from IdP login
6.2. OUTCOME OF BRIDGING BETWEEN OPENAM IDP AND OPENAM SP

containing users with session information, entityID, information about where to redirect and to get response and cookie information.

5. The credentials provided by user are verified in OpenAM IdP data store and tries to authenticate.

6. If credentials are valid the user is authenticated and session is requested by browser using session token sent by IdP.

7. OpenAM gets user pranay from user input and requests validation from IdP.

8. OpenAM creates valid session and POST request to validate it, Figure 6.14.
CHAPTER 6. RESULT

9. SSO session is established and request is redirected back to user.

10. After successful authentication, message appears on browser "Single Sign-On succeed".

6.3 OAuth 2.0 authorization with OpenAM

OpenAM provides several options to use OAuth 2.0 authorization protocol using Facebook, Google and other service providers. We used Facebook service provider to authorize users.

1. When user clicks on Facebook login button, the request goes via browser.

2. The user is redirected to facebook to validate user credentialsFigure 6.17.

3. Provided successful authentication at Facebook page, user is redirected back to open secured page as seen in Figure 6.18.

4. In Figure 6.19 OAuthProxy.jsp? file is used to connect public URL to OAuth 2.0 authorization server and this later returns back using OAuth2 redirect_uri parameter to an internal URL.

5. These parameters are then processed via OpenAM to authorize access to protected application.
6.3. OAUTH 2.0 AUTHORIZATION WITH OPENAM

10.1.1.12 - [21/Jul/2015:02:33:53 +0200] "POST /OpenAM-12.0.0/json/users?_action=getIdFromSession HTTP/1.1" 401 62
10.1.1.12 - [21/Jul/2015:02:33:54 +0200] "POST /OpenAM-12.0.0/json/authenticate?service=FacebookSocialAuthenticationService&authIndexType=service&authIndexValue=FacebookSocialAuthenticationService HTTP/1.1" 200 882
10.1.1.12 - [21/Jul/2015:02:34:08 +0200] "POST /OpenAM-12.0.0/json/authenticate?service=FacebookSocialAuthenticationService&authIndexType=service&authIndexValue=FacebookSocialAuthenticationService HTTP/1.1" 200 916

Figure 6.17: Tomcat log: Redirection to Facebook page requesting Login

10.1.1.12 - [21/Jul/2015:02:34:09 +0200] "GET /OpenAM-12.0.0/XUI/templates/openam/authn/OAuth2.html HTTP/1.1" 404 1055
10.1.1.12 - [21/Jul/2015:02:34:13 +0200] "POST /OpenAM-12.0.0/json/authenticate HTTP/1.1" 401 72
10.1.1.12 - [21/Jul/2015:02:34:13 +0200] "POST /OpenAM-12.0.0/json/authenticate?service=FacebookSocialAuthenticationService&authIndexType=service&authIndexValue=FacebookSocialAuthenticationService HTTP/1.1" 200 916

Figure 6.18: Tomcat log: Redirection from Facebook after authentication

10.1.1.12 - [21/Jul/2015:02:34:13 +0200] "GET /OpenAM-12.0.0/XUI/images/span_error.png HTTP/1.1" 304
10.1.1.12 - [21/Jul/2015:02:34:13 +0200] "GET /OpenAM-12.0.0/XUI/templates/openam/authn/OAuth2.html HTTP/1.1" 404 1055
10.1.1.12 - [21/Jul/2015:02:34:15 +0200] "GET /OpenAM-12.0.0/oauth2c/OAuthProxy.jsp?code=AOeDKC6zvR3/KkOz/YnMqD8A15h8v8YvX3bavNX5_1UqVv1pIdHRNhIGxjEUTP4Gal0/AhLS5Pa4DxOjis7UErz4y05__b2AXI67AmZrWmWvW33F70YdB2P54E2Gs__S1lRZHSHd9WpQdp5vyYcD-XlB959le7Ob516TCyCdGHiwZNhjWyPpxeE3B3X66C5CY6-BbTlV2D3cZ399hYenQwqTOOhBCg4y45S-3DZdHuQvvmPxQeXw15GAIazy-6DuuFZ3MqWlEviv_TphlKR8zJUc661SFZTVYUJFEw7fz1DXWJ900VvGhQy4-OI047AqyZULIFZg_FQkMx_Kz_vD0RyO2ZrRz6ZOa&state=9y2w04juw86odek21js2cu2xtgr07z HTTP/1.1" 200 824

Figure 6.19: Tomcat log: Redirection from Facebook after authentication
Chapter 7

Conclusion and Future Work

In this thesis, we presented OpenAM “all-in-one” solution. This solution specifically focuses on achieving multi-protocol federations and Single Sign-On solutions. To better understand the solution, we developed several use cases that might be useful later in production state. This elicit our key implementations:
First where OpenAM acts as an identity Provider and Shibboleth as service provider,
Second, OpenAM working as access management solution and Shibboleth serving as identity provider and service provider,
Third, OpenAM acting as identity provider and a service provider, And Fourth, SSO using WindowsDesktopSSO module in OpenAM.
By implementing these scenarios, we focused on achieving higher productivity as users would only need to use their credentials once and this might save some time.

Currently, the study focuses on using only OpenAM and OpenDJ. In future, other products such as OpenIG or similar can be used for making communications between different protocols and OpenAM, this might increase the overall performance of the system. Secondly, GWDG as a data center, could use analytical tools such as Elasticsearch, Logstash and Kibana in current systems that tracks login and logouts, events, sessions, problem detection’s etc. using OpenAM.
Appendix A

Installation Guide

A.1 OpenAM and OpenDJ Installation

A.1.1 Basic installation

1. install tomcat7 tomcat7-admin
2. Add tomcat-manager roles in tomcat-users.xml
   1. go to /etc/tomcat7/server.xml
   2. check for port 8080

A.1.2 OpenDJ Installation

1. Goto forgerock and download latest version of OpenDJ and OpenAM after registering
2. Install OpenDJ
   (a) sudo dpkg -L opendj*.deb (* = version)
3. Go to /opt folder
   (a) sudo ./setup
   (b) agree license and put password
   (c) click next and create your own directory (dc=example,dc=com)
4. start control panel and click start OpenDJ
A.1.3 Install OpenAM

1. Goto web browser
2. Open tomcat control panels
3. Browse OpenAM war file and click deploy in deploy section
4. After deployment click on newly create URL
5. Agree the license
6. Enter username password
   (a) Username = amAdmin and Password = password
7. Change root suffix
   (a) dc=forgerock,dc=example,dc=com
8. Connect to OpenDJ
   (a) Enter Root suffix : dc=example,dc=com
   (b) password = password
9. Click next
10. If you want to configure load balancing website you can enter the URL, Else
11. Click Next and OpenAM will install

A.2 Shibboleth IdP and SP

A.2.1 Basic Configuration

1. Install a Virtual Box on Ubuntu Version 14.14 (64-Bit) and after installing, proceed to next steps.
   (a) sudo apt-get update
   (b) sudo apt-get upgrade
2. Install Java
   (a) sudo apt-get install openjdk-7-jre-headless default-java
3. Install Tomcat7
4. Install Java
A.2. **SHIBBOLETH IDP AND SP**

(a) `sudo apt-get install tomcat7 tomcat7-admin tomcat7-examples tomcat7-docs tomcat7-user`

(b) `sudo addgroup ssl-cert`

(c) `sudo adduser tomcat7 ssl-cert`

5. Install Apache2

(a) `sudo apt-get install apache2`

(b) `sudo a2ensite default-ssl`

(c) `sudo a2enmod ssl`

(d) `sudo a2enmod proxy_ajp`

(e) `sudo service apache2 restart`

6. Install VirtualBox Guest Additions for full resolutions and reboot

(a) `sudo apt-get install dkms build-essential`

7.

**A.2.2 IDP Installation**

Check log files simultaneously for errors (If any)

1. `tomcat7: /var/log/tomcat7/`

2. `shibboleth-idp: /opt/shibboleth-idp/logs/`

3. Download IDP from the following link


5. Extract the zip file

(a) `tar shibboleth-identity-provider-3.1.1.zip`

6. Set variables

(a) `sudo -i`

(b) `export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64`

(c) `export IDP_SRC=/Downloads/shibboleth-identity-provider-3.1.1a`

7. Install using `sudo ./install.sh` command

8. Use these values during installation:
APPENDIX A. INSTALLATION GUIDE

(a) /opt/shibboleth-idp
(b) idp.thesisshib.com
(c) password: password

9. Tomcat configuration

```
<Proxy ajp://localhost:8009/idp/*>
Allow from all
</Proxy>
ProxyPass /idp/ ajp://localhost:8009/idp/
```

Table A.1: tomcat configuration

10. sudo chown tomcat7 /opt/shibboleth-idp/metadata
11. sudo chown tomcat7 /opt/shibboleth-idp/logs

12. Setting debug mode

(a) sudo nano /opt/shibboleth-idp/conf/logging.xml

```
<logger name="edu.internet2.middleware.shibboleth" level="DEBUG"/>
```

Table A.2: debug mode configuration

13. Activate Login via Unix Username/Password

(a) vim /etc/shibboleth-idp/login.config

```
within ShibUserPassAuth put:
com.sun.security.auth.module.UnixLoginModule required debug=true;
```

Table A.3: Configuration login.config file

14. Deploy the IdP war-file

(a) create idp.xml in /etc/tomcat6/Catalina/localhost/idp.xml

15. https://localhost/idp/status

16. Configure relying-party.xml

(a) vim /etc/shibboleth-idp/relying-party.xml
A.2. SHIBBOLETH IDP AND SP

<?xml version="1.0" encoding="UTF-8"?>
<Context
docBase="/opt/shibboleth-idp/war/idp.war"
privileged="true"
antiResourceLocking="false"
antiJARLocking="false"
unpackWAR="false"
swallowOutput="true">
</Context>

Table A.4: Create idp.xml file

<metadata:MetadataProvider id="ShibbolethMetadata" xsi:type="metadata:ChainingMetadataProvider">
<!– Load the IdP’s own metadata. This is necessary for artifact support. –>
<metadata:MetadataProvider id="IdPMD" xsi:type="metadata:FilesystemMetadataProvider"
metadataFile="/opt/shibboleth-idp/metadata/idp-metadata.xml"
maxRefreshDelay="P1D" />
<!– Example metadata provider. –>
<!– Reads metadata from a URL and store a backup copy on the file system. –>
<!– Validates the signature of the metadata and filters out all by SP entities in order to save memory –>
<!– To use: fill in ‘metadataURL’ and ‘backingFile’ properties on MetadataResource element –>
<metadata:MetadataProvider id="URLMD_SP1" xsi:type="metadata:FileBackedHTTPMetadataProvider"
metadataURL="http://sp1.intern.de/Shibboleth.sso/Metadata"
backingFile="/opt/shibboleth-idp/metadata/sp1-metadata.xml"/>
<!–
<metadata:MetadataFilter xsi:type="metadata:ChainingFilter">
<metadata:MetadataFilter xsi:type="metadata:RequiredValidUntil"
maxValidityInterval="P7D" />
<metadata:MetadataFilter xsi:type="metadata:SignatureValidation"
trustEngineRef="shibboleth.MetadataTrustEngine"
requireSignedMetadata="true" />
<metadata:MetadataFilter xsi:type="metadata:EntityRoleWhiteList"
</metadata:MetadataFilter>
<metadata:MetadataFilter>
</metadata:MetadataProvider>
</metadata:MetadataProvider>

Table A.5: Modify relaying-party.xml file

<metadata:MetadataProvider id="ShibbolethMetadata" xsi:type="metadata:ChainingMetadataProvider">
<!– Load the IdP’s own metadata. This is necessary for artifact support. –>
<metadata:MetadataProvider id="IdPMD" xsi:type="metadata:FilesystemMetadataProvider"
metadataFile="/opt/shibboleth-idp/metadata/idp-metadata.xml"
maxRefreshDelay="P1D" />
<!– Example metadata provider. –>
<!– Reads metadata from a URL and store a backup copy on the file system. –>
<!– Validates the signature of the metadata and filters out all by SP entities in order to save memory –>
<!– To use: fill in ‘metadataURL’ and ‘backingFile’ properties on MetadataResource element –>
<metadata:MetadataProvider id="URLMD_SP1" xsi:type="metadata:FileBackedHTTPMetadataProvider"
metadataURL="http://sp1.intern.de/Shibboleth.sso/Metadata"
backingFile="/opt/shibboleth-idp/metadata/sp1-metadata.xml"/>
<!–
<metadata:MetadataFilter xsi:type="metadata:ChainingFilter">
<metadata:MetadataFilter xsi:type="metadata:RequiredValidUntil"
maxValidityInterval="P7D" />
<metadata:MetadataFilter xsi:type="metadata:SignatureValidation"
trustEngineRef="shibboleth.MetadataTrustEngine"
requireSignedMetadata="true" />
<metadata:MetadataFilter xsi:type="metadata:EntityRoleWhiteList"
</metadata:MetadataFilter>
<metadata:RetainedRole>samlmd:SPSSODescriptor</metadata:RetainedRole>
</metadata:MetadataProvider>
</metadata:MetadataProvider>

Table A.5: Modify relaying-party.xml file
A.2.3 Installing SP

1. apt-get install libapache2-mod-shib2

2. configure shibboleth2.xml

   (a) vim /etc/shibboleth/shibboleth2.xml

   ```xml
   <!– The ApplicationDefaults element is where most of Shibboleth’s SAML bits are defined. –>
   <ApplicationDefaults entityID="https://sp1.intern.de/shibboleth"
   REMOTE_USER="eppn persistent-id targeted-id">
   <Sessions lifetime="28800" timeout="3600" relayState="ss:mem"
   checkAddress="true" handlerSSL="false" cookieProps="https"
   handlerURL="/Shibboleth.sso" exportLocation="https://sp1.intern.de/Shibboleth.sso/GetAssertion">
   <SSO entityID="https://idp.intern.de/idp/shibboleth">
   SAML2 SAML1
   </SSO>
   </Sessions>
   <Logout>SAML2 Local</Logout>
   <Handler type="MetadataGenerator" Location="/Metadata" signing="true" https="true" http="false"/>
   [...]</Sessions>
   [...]</Logout>
   [...]</ApplicationDefaults>
   [...]<MetadataProvider type="XML" uri="https://idp.intern.de/idp/profile/Metadata/SAML"
   backingFilePath="federation-metadata.xml" reloadInterval="7200">
   [...]<MetadataFilter type="RequireValidUntil" maxValidityInterval="2419200"/>
   <MetadataFilter type="Signature" certificate="fedsigner.pem"/>
   -></MetadataProvider>
   ```

3. Generate keys

   (a) shib-keygen -h sp1.intern.de -e https://sp1.intern.de/shibboleth