Jussi Jäppinen

Single Sign-On Federation Instance
Providing Data about Authenticated Users to Web Applications

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology and Communications
Bachelor’s Thesis
21.04.2017
This thesis entails the process of installing and configuring a Single Sign-On instance that enables its users to access web applications and services without the need of multiple authentications.

The end-users will access the service through a portal which they have already logged into, the agent application will mostly only validate the pre-existing cookies and allow access to the integrated applications. The authentication itself is done externally by creating a connection to a pre-existing environment built for the purpose, using an “Agent”-application that handles the related queries and responses. The session cookies are saved in the user’s web browser, so the user will not have to log in again until the cookie has expired. As long as a valid cookie has been created for the user, they should be able to access the applications and services integrated into this environment. The main purpose of the environment is transferring user information to the applications, i.e. federating, which was implemented here using the Secure Assertion Markup Language (SAML).

Keywords: Authentication, Single Sign-On, Federation
Contents

1 Introduction 1

2 Environment 1

2.1 Background 1
2.2 Theory and Protocols 2
2.2.1 Parties 2
2.2.2 Traffic Flow During Authentication 2
2.2.3 Service Environment 5
2.3 Pre-requisites 7
2.3.1 Operating System and Network 7
2.3.2 Ensuring Availability 7

3 Installation 8

3.1 Service Account Creation 8
3.2 Installing OpenSSL 8
3.3 Installing Apache HTTP Server 9
3.4 Installing Apache Tomcat 10
3.5 Installing Java JDK 11
3.6 CA Single Sign-On WebAgent 12
3.6.1 Role 12
3.6.2 Installation 12

4 Component Configuration 15

4.1 Apache Tomcat 15
4.2 Apache HTTPD Server 15
4.3 CA WebAgent 17
4.4 General Configurations to Shibboleth IdP 23
4.4.1 Adaptations to Environment 23
4.4.2 Changes to idp.properties 24
4.4.3 Changes to logback.xml 25
4.4.4 Changes to relying-party.xml 26
4.4.5 Changes to metadata-providers.xml 27
4.4.6 Changes to access-control.xml 27
4.4.7 Building Shibboleth 28

5 Integrating Service Providers into Shibboleth IdP 28
1 Introduction

This thesis covers the setup of an instance federating the authentications of external business users for a client firm of KPMG. The contents of the thesis have been altered to keep the client anonymous, which limits the validity of the text, but on the other hand makes the information more generalized and adaptable to other environments. The thesis was written to depict the installation and configuration process of the instance, with explanations of the components and their functions. As the federation instance was an addition to an existing environment, the attached systems are also discussed, but in very general terms as to limit the possibility of the client's recognition.

2 Environment

2.1 Background

This project was built into an existing environment, where there are multiple pre-configured user databases, services and applications. Due to this, the document mostly includes information that is specific to this environment, but can in theory be implemented elsewhere as well. Pre-existing environments exist to authenticate internal users to third-party applications and to authenticate business users to internal applications. This project was based on the need to authenticate business users outside the Internal LDAP, into applications hosted by third-party owners outside the organization's intranet.

Before building the environment, the business users would have to login manually, using their username and password as long as the accounts are delivered from the business LDAP to the application by means of data delivery. The goal of the new solution was to allow the users to login by simply pressing a link to the application, while the system authenticates the user in the background automatically. This solution can be referred to as Single Sign-On, or SSO, which is a fast-growing field, especially in the world of large organizations and enterprises.
2.2 Theory and Protocols

2.2.1 Parties

Single Sign-On is a process in which a user is Authenticated into multiple applications or resources, while not having to input their credentials into each separate entity as long as their session is valid. The Shibboleth Identity Provider uses the Security Assertion Markup Language to communicate to the applications that have been integrated into it as federated services.

Security Assertion Markup Language (SAML) is an XML-based open standard for exchanging authentication and authorization data between security domains, i.e., an identity provider and a service provider. Using SAML, an online service provider contacts an online identity provider which authenticates users who are trying to access secure content. [1, 2.4.2]

Setting up the SAML connection requires the Identity Provider (IdP) and Service Provider (SP) to exchange metadata about the respective services, based on which the parties can communicate and validate the incoming SAML messages. The metadata files are usually exchanged between the customer when a new federation is needed, or sometimes when large modifications are made to the service provider’s application. The Metadata includes all the needed information and URL address for the two parties to successfully communicate and can also include certificates for validating the messages between the parties. The built environment as a whole will be referred to as “Bisfed”, an abbreviation from the words “Business Federation”.

2.2.2 Traffic Flow During Authentication

Most of the traffic that happens during a user’s authentication process uses the HTTPS protocol, which is a common protocol used by web browsers to transfer files and data over the internet. The process starts when the user clicks on an HTTPS link to an application that has been federated using the instance. As a result, they will be redirected to either the application, or directly to the Bisfed service, depending on whether a Service Provider-initiated, or an Identity Provider-initiated link is used.
In a Service Provider-initiated scenario, the user is first sent to the web application itself, where a web server listening to the incoming HTTP traffic notices the user’s attempt to log in and formulates a request meant for the Identity Provider. Once ready, an authentication request is passed back on to the user’s web browser and they are redirected with it to the IdP’s HTTP REDIRECT URL, that the Service Provider gets from the IdP’s Metadata. The transferred SAML message is called an AuthnRequest and looks similar to the excerpt below.

```xml
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
   xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
   ID="identifier_1"
   Version="2.0"
   IssueInstant="2004-12-05T09:21:59Z"
   AssertionConsumerServiceIndex="1">
   <saml:Issuer>https://sp.example.com/SAML2</saml:Issuer>
   <samlp:NameIDPolicy
      AllowCreate="true"
      Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"/>
</samlp:AuthnRequest>
```

An example AuthnRequest provided by Oasis [2, 5.1.2]

The AuthnRequest should include all the necessary information for the authentication process to start. It specifies the used protocols, which versions of them should be used, the creation time of the request, which Service Provider it is meant for and if used, what the format of the NameID attribute should be.

Once the Bisfed’s Apache HTTP server gets the request, it will send the request forward to Tomcat due to the AJP Proxy and Reverse Proxy configurations that are added to the web server’s httpd.conf file. Once Tomcat receives the request it will pass it onto the Shibboleth Identity Provider instance running as a webapp under the specific URL path. Shibboleth checks the AuthNRequest and as a result will start to formulate an answer according to the configuration established with the application team. The final result is called an assertion and its content aims to provide the required data for user to be connected to their respective account inside the application itself. As Shibboleth gets the data from LDAP servers behind a load-balancer, the connection there has to be set up, as well. There is a pre-determined set of attributes that Shibboleth will fetch from the LDAP for all the users, however the ones that are actually included in the assertion are
determined by the filter configuration for each respective application. Usually the information sent is limited to just a few attributes, as to minimize possible data leakage to unwanted third-parties. Figure 1 illustrates a service provider initiated process.

Figure 1. A flowchart showing a Service Provider-initiated authentication process.

In and IdP-Initiated process, the first part where the user is sent to the application to the IdP carrying an AuthNRequest, is cut off. Instead, the user will click a link that starts the process directly at the IdP. This is done by using an link with the EntityID from the application’s Metadata, based on which the Identity Provider can start the same exact process as would happen in a SP-initiated authentication. Figure 2 illustrates the afore mentioned process.
Figure 2. A flowchart showing an Identity Provider-initiated authentication process.

While much of the process is the same, the SP-initiated process can be regarded as more secure due to the additional steps performed during an authentication. The service provider has to make a request and connect it to the received answer, whereas in the IdP-initiated process the SP just accepts the messages without making the connection.

### 2.2.3 Service Environment

The new environment will consist of two servers running Apache HTTP server instances, CA WebAgent, Apache Tomcat, and Shibboleth Identity Provider instances. Shibboleth handles building the assertion that in the end is used to verify the authentication of the user on the application side. Depending on what attributes the customer binds with the application, the IdP can be configured to fetch them from the connected LDAP directory and send them in the assertion. Because of the configured trust relationship between the IdP and the application, the application can trust the assertion it receives to be authentic and can log in the user without the need to input a password. Figure 3 shows a visualization of the network components that are attached to, or a part of the environment.
Figure 3. A visualization of the Bisfed environment and its connected servers and directories. Information regarding the system has been changed to not represent the actual values.

If the application does not have the respective accounts imported via data-delivery, Just-In-Time Provisioning can be used to create new accounts based on the information provided in the assertion. Due to this, the most common Attributes sent are the first and last name of the user, their username, as well as their phone number and email.

The Apache Tomcat instances, on which Shibboleth is running on, are configured to use external authentication. They will therefore deliver all the requests to the Apache front-end servers, which in turn pass it onto the attached SiteMinder WebAgents using an automatically configured module. SiteMinder is an application developed by CA, that is used as the backbone of the SSO system in this environment. The WebAgents provide the login pages to the user through the Apache front-end, takes the user-inputted credentials and communicates with the Policy Servers to authenticate the user using either Windows Active Directory, or Radius authentication.
2.3 Pre-requisites

2.3.1 Operating System and Network

The installation described in this document was done on a server running on the Red Hat Enterprise Linux 6.6 (Santiago). Since the service needs to be able to access policy servers over the network, all the necessary firewall openings were requested beforehand to ensure the service’s functionality once installed. The CA WebAgent needs the connections to be open so it can request the authentications from the Policy server. If the user already has a browser cookie from a previous authentication, some of these connections are needed to verify its validity. All the necessary files need to be downloaded beforehand for the installation process to be as smooth as possible and to allow rollback, if needed.

2.3.2 Ensuring Availability

Since the service was built to run in a corporate production environment, some steps are to be taken to ensure the services availability at all times. Identical federation instances will be set up on two separate virtual servers running in the company’s data center. The system is connected to multiple Policy Servers in different cells ensuring that Authentication is possible at all times. The connection to the LDAP Directories was established through a Global Load-Balancer, which in turn divides the traffic to multiple Local Load-Balancers, which are connected to multiple nodes.
3 Installation

Once all the files needed in the installation are gathered in the source folder, the installation is ready to begin. This chapter goes through the steps taken in the installation command-by-command with an explanation of what they are for and what how they affect the final installation. The folder used for the installation was based on previously installed environmental variables, in which the folder /app has been used. The new IdP will be installed into the folder /app/b2bidp/ based on the app federating business-to-business, or b2b users.

Even though the installations could be done automatically, a downloaded packed file was used and saved under a /source directory, in case something goes wrong in the future and a component needs to be reinstalled.

3.1 Service Account Creation

A service account needs to be environment helps to spot the applications belong to it, when grepping based on application names, for example. In this case, if the service account will be named 'bisfed', condensed from 'business federation'. The user was given a UID of 17000 to make it easier to recognize from others.

The account was created with the following command;

```
/usr/sbin/useradd -m -d /app/bisfed -u 17000 bisfed
```

3.2 Installing OpenSSL

The installation should start with installing the OpenSSL using the created bisfed account. OpenSSL is needed to handle the certificates that Apache and Shibboleth will use for creating trust relationships between different parties. The version used for this installation is 1.0.2f. The downloaded file will usually be in tar format and needs to be unpacked using the ‘tar’ command.

```
tar xvf openssl-1.0.2f.tar.gz
```
This should create a directory /openssl-1.0.2f. The 'cd'-command is used to navigate between folders.

Running the OpenSSL configuration file with --prefix= and --openssldir defines the installation locations to be used once the application has been compiled. In this case the folder will be under the default one as /openssl.

```
./config --prefix=/app/b2bidp/openssl --openssldir=/app/b2bidp/openssl
shared
```

The actions performed during the compilation are first outputted into a “makeOutput.txt” file, which is useful in tracing problems if some arise during the actual compilation of the application. When encountering an error, checking the contents of this output file might bring the issue to light, by showing what was done prior to error event. The output file can be created with the following command:

```
make > makeOutput.txt
```

After the compiling, everything should be ready for the installation. The procedure can be started with the following command.

```
make install
```

3.3 Installing Apache HTTP Server

After unpacking the downloaded Apache HTTP server file and navigating to the folder, running the configuration command is required. Prefixes are used to install certain functions and features required in the environment. For example, the installed SSL libraries are needed in securing internet connections and the proxies in setting up communications to Tomcat. Running the below command will take a relatively long amount of time, since it defines all installation locations for the application. The version used here will be Apache 2.2.31.

```
./configure --enable-so --enable-ssl --enable-proxy --enable-proxy-http
--enable-proxy-connect --enable-headers --enable-ext-filter --with-
mpm=worker --enable-rewrite --prefix=/app/b2bidp/apache-2.2.31 --with-
ssl=/app/b2bidp/openssl --enable-module="shared all"
```
After the configuration has finished, outputting the compilation to a text file can be started.

```make
make > makeOutput.txt
```

Compiling Apache will not take as long as OpenSSL, but can still take a few minutes. After the process has completed, the installation can be started with the following command;

```make
make install
```

After the installation has been completed, Apache should be ready to use. Lines for OpenSSL need to be added to the environmental variables, which can be done by editing a file under the Apache installation’s /bin directory. The following lines defines root and OpenSSL paths and ‘exports’ them for Apache to use.

```bash
LD_LIBRARY_PATH="/app/b2bidp/apache-2.2.31/lib:$LD_LIBRARY_PATH"
export LD_LIBRARY_PATH
LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/app/b2bidp/openssl/lib
export LD_LIBRARY_PATH
```

### 3.4 Installing Apache Tomcat

Installing Apache Tomcat is done by unpacking the downloaded file and copying or moving it to where the installation is wanted. All the connections to the other applications are created when the configuration is done.

```bash
tar xvf apache-tomcat-8.5.5.tar.gz
cp -R apache-tomcat-8.5.5 /app/b2bidp/
```

By executing the commands above, Apache Tomcat should be installed. To ensure that all the unpacked files are accessible and editable, full rights are added to the service account with the first command. If it is also desired to limit usage from other accounts, the second command can be used to grant them only read and execution rights.

```bash
chmod -R u+rwX apache-tomcat-8.5.5/bin/
chmod -R 755 apache-tomcat-8.5.5/bin/
```
3.5 Installing Java JDK

The java installation is quite straightforward, as it mostly consists of just copying files from one place to another and adding a few lines of text into the user’s bash profile. The installation starts with unpacking the downloaded java file and copying the resulting folder to the /app/b2bidp directory.

```
tar xvf jdk-8u72-linux-i586.gz
cp -R jdk1.8.0_72/ /app/b2bidp
```

To add the environmental variables, modify the .bash_profile in the /app/bisfed/ folder.

```
nano /app/bisfed/.bash_profile
```

Adding these lines will define the paths to OpenSSL and Java and export them for other applications to use. Defining JAVA_HOME is also needed, so it can be referenced to easily.

```
#BFED SPECIFIC
PATH=/app/b2bidp/openssl/bin:/app/b2bidp/java/bin:$PATH:$HOME/bin
export PATH

JAVA_HOME=/app/b2bidp/java
export JAVA_HOME
```

One can test if the installation has succeeded by logging out and logging back in again. If defining the Java installation has been successful, an output similar to this should appear, once the command has been entered.

```
java -version
java version "1.8.0_72"
Java(TM) SE Runtime Environment (build 1.8.0_72-b15)
Java HotSpot(TM) Server VM (build 25.72-b15, mixed mode)
```

The default Java installation also needs to be changed by including the Java Cryptography Extension files to allow encryption and keys to be used in the applications running on the service account. Shibboleth, for example, can use the JCE to encrypt the assertions it sends to the application and needs it to decrypt messages, as well. Once the files are downloaded from Oracle, they can be installed to under /app/b2bidp/java/jre/lib/ by overwriting the existing files.
gzip /app/b2bidp/Source/jce_policy-8.zip

cp /app/b2bidp/Source/UnlimitedJCEPolicy/* /app/b2bidp/java/jre/lib/security/

After replacing the files, the Java installation is complete, and the service should be able to use the extensions to cypher and decipher encrypted information.

3.6 CA Single Sign-On WebAgent

3.6.1 Role

For the service to function as a whole, Shibboleth has to have a trusted source confirming the identities of the people logging in. The environment in which the federation instance is being built, SiteMinder Policy Servers are used as the backbone of authentication. It gets authentication and authorization requests from different instances and validates them against configured database sources. SiteMinder can also be used to grant and block access from different applications, or “domains”, by creating rules and responses as needed. This is useful in situations such as corporation mergers, where the databases of two companies are combined, but access to resources still needs to be restricted. In this case however, these rules can also be made to the Shibboleth configurations.

Usually rules are set up so that they filter out any users who have a certain attribute in their user account, such as a Company name or location. SiteMinder is also responsible for creating and confirming the authenticity of the browser cookies that are used for keeping a session open after logging in. The environment here uses cookies specifically for business users, which are not usable in the company’s internal side.

3.6.2 Installation

The installation itself is started by unpacking the latest version of the Linux web agent to the /app/b2bidp/source directory. The installer is located inside the archive as an executable binary file. Rights for executing the file should be added for the current user. These commands are be used to unpack the file and begin the installation process.
Inputting these commands starts the installation in the Unix console window. The output of the installer will be shown here as it would be presented on screen, but with some parts omitted for the sake of clarity. Proposed answers to the questions asked in the dialog are shown in an enlarged and bold font. Note that the answers shown here might not reflect the answers inputted when the installation was done to the environment.

Preparing to install...
Extracting the JRE from the installer archive...
Unpacking the JRE...
Extracting the installation resources from the installer archive...
Configuring the installer for this system's environment...

Launching installer...

Preparing CONSOLE Mode Installation...
PRESS <ENTER> TO CONTINUE:

DO YOU ACCEPT THE TERMS OF THIS LICENSE AGREEMENT? (Y/N): Y

Answering Y (yes) to this dialog indicates that the user installing has read the license agreement and agrees with its terms. Answering N (no) will not allow the installation to continue. After agreeing to the terms the installation will ask for an installation path and confirmation for the Web Agent client. In this case, /app/bisfed/CA/webagent is used.

Choose Install Location

Specify a location for the Web Agent. If the path does not contain the word "webagent," the installation program will create a folder called "webagent" and appends it to the end of your path.

Where would you like to install?
Default Install Folder: /app/bisfed/CA/webagent
ENTER AN ABSOLUTE PATH, OR PRESS <ENTER> TO ACCEPT THE DEFAULT:
INSTALLED FOLDER IS: /app/b2bidp/CA/webagent

IS THIS CORRECT? (Y/N): Y
After pressing enter in this dialog, the installation process starts and the screen will show a summary of the actions that it will run.

Pre-Installation Summary

Please Review the Following Before Continuing:

Product Name:
CA SiteMinder Web Agent

Install Folder:
/app/b2bidp/CA/webagent

Disk Space Information (for Installation Target):

- Required: 212,201,391 bytes
- Available: 78,647,975,936 bytes

PRESS <ENTER> TO CONTINUE:

Installing...

[--------------------|--------------------|--------------------|--------------------]
[-------------------|-------------------|-------------------|-------------------]

Install Complete

Congratulations. CA SiteMinder Web Agent v12.50 has been successfully installed
to: /app/b2bidp/CA/webagent

To configure the web agent, run the command

/app/b2bidp/CA/webagent/install_config_info/ca-wa-config.bin

PRESS <ENTER> TO EXIT THE INSTALLER:

The installation wizard should have copied the necessary files it needs to run to the correct folders. However, the components still need to be loaded in the Apache web server’s configuration files for the Agent to be activated.
4 Component Configuration

4.1 Apache Tomcat

As Apache Tomcat is only used to as a platform to run Shibboleth, its configuration consists mainly of creating an AJP connector, which it will use to communicate to Apache with. The other changes are to disable the option for HTTP connectivity and to change the values regarding the default server port, to slightly increase security. These modifications are done to the server.xml file, which resides under /app/b2bidp/apache-tomcat-8.5.5/conf/. Placing text between <!-- and --> in XML removes the content from being executed.

```xml
<!--<Server port="8080" shutdown="SHUTDOWN">-->
<Server port="8105" shutdown="SHUTDOWN9000RIP">

<!-- <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" /> -->

<!-- Define an AJP 1.3 Connector on port 8209 -->
<Connector port="8209" protocol="AJP/1.3" redirectPort="42443" address="localhost" scheme="https" proxyPort="443" tomcatAuthentication="false" />
```

The default files that reside under /apache-tomcat-8.5.5/webapps/ can also be removed, even though they can not be accessed unless an AJP connector is created to share them in the Apache HTTPD Server configuration.

4.2 Apache HTTPD Server

The Apache HTTPD Server will work as the front-end server, listening to the traffic arriving to both the configured IP address and the hostname. Configuration for this requires editing the httpd.conf file under /apache-2.2.31/conf/.

```conf
Listen 192.254.100.16:80
#For Production:
ServerName https://bisfed.ext.company.com
UseCanonicalName On
```

Changing the user and group parameters to reflect the name of the Service Account is also needed. Apache allows the software to run under accounts specified here.

```
[.output omitted..]
User bisfed
Group bisfed
```

Log rotation is added, so that logs are not overwritten, Apache HTTPD comes with its own rotatelogs program that is used for this feature. Logs are added to collect errors and access information. The parameters -l and -f are used to make sure that the logs entries are using local time and that the files are created instantly once the server starts. The rotated log names are marked with %Y for year, %m for month and %d day. The value 86400 is used to make sure that the logs are rotated each day at midnight. [4]

```
[.output omitted..]
ErrorLog "|/app/b2bidp/apache-2.2.xx/bin/rotatelogs -l -f /app/b2bidp/apache-2.2.xx/logs/error_log_%Y_%m_%d 86400"
CustomLog "|/app/b2bidp/apache-2.2.xx/bin/rotatelogs -l -f /app/b2bidp/apache-2.2.xx/logs/access_log_%Y_%m_%d 86400" common
```

AJP Connectors are configured to pass through files hosted on Tomcat under its webapps. The connectors are added with ProxyPass and ProxyPassReverse to localhost:port to allow communication in and out, respectively. Shibboleth can be reached from under /idp/ and the healthcheck.html files will be under the root path.

```
[.output omitted..]
#AJP CONNECTOR configurationsProxyRequests Off
ProxyPass /idp ajp://localhost:8209/idp
ProxyPassReverse /idp ajp://localhost:8209/idp
ProxyPass /healthcheck.html ajp://localhost:8209/healthcheck.html
ProxyPassReverse /healthcheck.html ajp://localhost:8209/healthcheck.html
```

Removing folder indexing is also recommended to lower the possibility of file probing and vulnerability exploiting by malicious parties. This is easily done by removing searching the file for the word “Indexes” and removing the any occurrences.
4.3 CA WebAgent

The SiteMinder WebAgent is configured using a tool that CA provider with the installed package. The Configuration Wizard also communicates the necessary information to the SiteMinder Policy Server to allow the registration of the Agent. The Configuration Wizard requires some environmental variables to be set beforehand, for which a script exists as well. Inputting these commands sets the variables and starts the configuration tool in the SSH console.

```
./ca_wa_env.sh
./ca-wa-config.sh -i console
```

As the Host Registration is chosen to be done, the Configuration Wizard will then prompt for an Administrator username and Password that has the rights to perform registrations in SiteMinder. After inputting correct credentials, the Wizard asks for the Host Configuration Object that, which the Agent should use.

```
Enter the name of the host configuration object. The name must match a host configuration object name already defined on the Policy Server.

Trusted Host Name (DEFAULT: ): iamprod-b2bidp-w1
Host Configuration Object (DEFAULT: ): iamprod-europe-helsinki
```

An IP address and registration port for one of the Policy Servers is then entered. The rest of the Policy Servers are manually added to a configuration file after the Wizard has completed.
Policy Server IP Address
----------------------------------------
Enter the IP Address of the Policy Server where you are registering this host.
Multiple IP addresses must separate by comma. The IP address should be in the form <server_address:port>, where the port represents a Policy Server behind the firewall. For Example:
111.12.12.2:1234 or myserver:1234 (IPv4)
[2001:db8::1428:57ab]:1234 (IPv6)

NOTE: Include the port number in the IP address only if your Policy Server is behind a firewall.

Policy Server IP Address (DEFAULT: ): 127.0.0.0:60442

After entering the IP Address, a selection for FIPS mode is shown. The Federal Information Processing Standards are a set of standards developed by an US government agency. For this environment, these standards do not need to be used so Compatibility Mode is selected.

FIPS Mode Setting
------------------
The use of FIPS-compliant algorithms is optional. If your organization does not require the use of FIPS-compliant algorithms, leave FIPS Compatibility Mode selected. If they are required, select either FIPS Migration Mode or FIPS Only Mode. For more information about selecting the appropriate mode, see the Web Agent Installation Guide.

--> 1- FIPS Compatibility Mode
    2- FIPS Migration Mode
    3- FIPS Only Mode

ENTER THE NUMBER FOR YOUR CHOICE, OR PRESS <ENTER> TO ACCEPT THE DEFAULT:: 1

The Configuration Wizard will then ask what name should be used for the created configuration file and where to save it. The defaults can be used, as it automatically suggests a suitable path by itself.
Enter a file name and location to store Host Configuration information or accept the default location /app/b2bidp/CA/webagent/config and file-name SmHost.conf.

Enter file name (DEFAULT: SmHost.conf): SmHost.conf

Enter location (DEFAULT: /app/b2bidp/CA/webagent/config):

The next few selection screens are in regards to the Web Server used, which in this case is the Apache Web Server. Then its path, version and server type are selected, which are /app/b2bidp/apache-2.2.31, Apache 2.2.X and HTTP Apache, respectively.

------------------------------------------------------------------------
Select Web Server(s)  
---------------------
Select which Web Server(s) you want to configure as a Web Agent. You will have to enter a path for each selected web server.
Note: If you have an Apache-based Web server, please select the Apache Web Server option.

1- Apache Web Server
2- Domino Web Server
3- iPlanet or Sun ONE Web Server

ENTER A COMMA-SEPARATED LIST OF NUMBERS REPRESENTING THE DESIRED CHOICES, OR PRESS <ENTER> TO ACCEPT THE DEFAULT: 1

------------------------------------------------------------------------
Apache Web Server path  
----------------------
Enter the root path of where Apache Web server installed. Please enter path (DEFAULT: ): /app/b2bidp/apache-2.2.31

------------------------------------------------------------------------
Apache Version  
--------------
Please select a choice for the Apache version.

1- Apache version 1.x
2- Apache version 2.x
3- Apache version 2.2.x
ENTER THE NUMBER OF THE DESIRED CHOICE: 3

Apache Server Type

Please select one of the following appropriately match your previous selection

1- Strong Hold
2- Oracle HTTP Server
3- IBM HTTP Server
4- Covalent Enterprise-Ready Apache or FastStart Server
5- HP Apache
6- HTTP Apache

ENTER THE NUMBER FOR YOUR CHOICE, OR PRESS <ENTER> TO ACCEPT THE DEFAULT:: 6

Next the Agent Configuration Object is defined. The object contains information that the agent will use to configure itself and the Apache connection with. These object are pre-set on the SiteMinder Policy Servers.

Agent Configuration Object

Enter the name of an Agent Configuration Object that defines the configuration parameters which the Web Agent will use for Apache 2.2.31.

Agent Configuration Object (DEFAULT: AgentObj): iamprod-b2bidp

The SSL Connection is then defined to be HTTP Basic over SSL, which means that as long as the connections used are HTTPS, the traffic can be validated. The other option would be to use client certificates, but such is not required for this environment.

SSL Authentication

The following SSL configurations are available for this web server. If the Web Agent will be providing advanced authentication, select which configuration it will use to configure Apache 2.2.25.

--> 1- HTTP Basic over SSL
2- X509 Client Certificate
The last two screens are to enable the WebAgent, and to summarize the configurations that will be done with the Wizard. Selecting to Enable the Agent is preferred here, so it will not have to be done at a later time separately.

Webagent Enable option

Please select Yes to Enable the WebAgent

1- Yes
2- No (DEFAULT)

Web Server Configuration Summary

Please confirm the configuration selection. Accept the configuration and press 'Enter' to continue. To change one or more settings, select 'Previous'. Select 'Cancel' will exit the configuration.

Configure the following webserver(s):
Apache Server:
Apache 2.2.25
Agent Configuration Object: iamprod-b2bidp
SSL Authentication type: HTTP Basic over SSL
IS WebAgent Enabled: YES

Please enter a choice.
-> 1- Continue
    2- Previous
    3- Cancel
ENTER THE NUMBER OF THE DESIRED CHOICE, OR PRESS <ENTER> TO ACCEPT THE DEFAULT

Press enter to close the installer.

-------------------------------------------------------------------------------------------------------------- Installing...
---------------------------
[==================================|=================================|==================================|===================]
[==================================|=================================|==================================|===================]  
Configuration Complete

Congratulations! CA SiteMinder Web Agent has been successfully configured.
PRESS <ENTER> TO EXIT THE INSTALLER:

After completion, the Agent should be mostly configured. Some additional manual changes are however required for it to properly function. First, the CA WebAgent environmental variables need to be included into the Apache HTTP Servers envvars file under /app/b2bidp/apache-2.2.31/bin. This can be done easily with the cat command.

```
cat /app/b2bidp/CA/webagent/ca_wa_env.sh >> /app/b2bidp/apache-2.2.31/bin/envvars
```

The envvars file should afterwards be appended with the correct variables.

The next part is to modify the WebAgent.conf file created by the Configuration Wizard. In this file there are general configurations regarding the WebAgent, which specify for example file locations for other config files and Apache plugins. In this file, the localconfig.xml is uncommented to allow it to be loaded, since a parameter in the file needs to be edited.

```
# WebAgent.conf - configuration file for SiteMinder Web Agent
# Web Agent Version = 12.50, Build = 875, Update = 03
#agentname="<AgentName>, <IPAddress>"
HostConfigFile="/app/b2bidp/CA/webagent/config/SmHost.conf"
AgentConfigObject="iamprod-b2bidp"
EnableWebAgent="YES"
ServerPath="/app/b2bidp/apache-2.2.31/conf"
```
localconfigfile="/app/b2bidp/apache-2.2.31/conf/LocalConfig.conf"
LoadPlugin="/app/b2bidp/CA/webagent/bin/libHttpPlugin.so"
#LoadPlugin="/app/b2bidp/CA/webagent/bin/libAffiliate10Plugin.so"
#LoadPlugin="/app/b2bidp/CA/webagent/bin/libSAMLAffiliatePlugin.so"
#LoadPlugin="/app/b2bidp/CA/webagent/bin/libeTSSOPlugin.so"
#LoadPlugin="/app/b2bidp/CA/webagent/bin/libIntroscopePlugin.so"
#LoadPlugin="/app/b2bidp/CA/webagent/bin/libSAMLDataPlugin.so"
#LoadPlugin="/app/b2bidp/CA/webagent/bin/libOpenIDPlugin.so"
#LoadPlugin="/app/b2bidp/CA/webagent/bin/libDisambiguatePlugin.so"
AgentIdFile="/app/b2bidp/apache-2.2.31/conf/AgentId.dat"

The local configuration file itself should be edited to include only the DefaultAgentName, which refers to the WebAgent itself, and options for creating logs if needed. All other options can be either commented out or removed.

DefaultAgentName="iamprod-b2bidp-w1"
LogAppend="NO"
LogFile="NO"
LogFileName="/app/b2bidp/CA/webagent/log/webagent.log"
TraceAppend="NO"
TraceConfigFile="/app/b2bidp/CA/webagent/config/WebAgentTrace.conf"
TraceFile="NO"
TraceFileName="/app/b2bidp/apache-2.2.31/logs/Web-Agent-Trace.log"

After these changes the Agent is configured and ready to use. Rest of the configurations regarding the Agent are done on the SiteMinder Policy Servers, which this thesis does not touch upon.

4.4 General Configurations to Shibboleth IdP

4.4.1 Adaptations to Environment

As Shibboleth is installed in a non-default location, an addition is needed to the the web.xml file, that is used to define parameters for Shibboleth. The parameter that needs to be defined is idp.home and its value should be the installation location. If this is not done, Tomcat tries to start Shibboleth from the location /opt/shibboleth-idp/ It would
also be possible to define the parameter on an OS level, but as there are multiple environments running on the same server, it is better to define in this file. The file is placed under Shibboleth’s /shibboleth-idp/webapp/WEB-INF and /shibboleth-idp/edit-webapp/WEB-INF folders.

```
<context-param>
    <description>Enable debugging for the application</description>
    <param-name>debug</param-name>
    <param-value>true</param-value>
</context-param>
```

An example of a context-parameter from the Apache Software Foundation [5]

The Apache Software Foundation’s wiki for web.xml shows that such context parameters can be defined as in the example shown above.

```
<context-param>
    <param-name>idp.home</param-name>
    <param-value>/app/bisfed/shibboleth-idp</param-value>
    <description>Defines the installation path for the Shibboleth IdP</description>
</context-param>
```

By changing the "param-name" to "idp.home" and its value "param-value" to "/app/bisfed/shibboleth-idp". A short description is added for future reference, as well.

### 4.4.2 Changes to idp.properties

This file contains general, but very important configurations about the IdP related for example to the certificates and the name the IdP is recognized with. The first thing to set is the EntityID, which is the Identifier used for the IdP. This is referred for example in the metadata files provided for Service Providers when configuring trust relationships. The scope is used to identify to which domain level the service provides Authentication to. In this case, as no subdomain is specified, the Authentication is valid for the whole domain.

```
idp.entityID= https://bisfed.ext.company.com/idp/shibboleth
idp.scope= company.com
```

The next settings are related to the Signing and Encryption certificates used by the service to provide validation to the assertions sent. The signing certificate will be included
in every assertion sent by the IdP, and is verified by each Service Provider. It should also
be manually included to the idp-metadata.xml file, so that it can be imported to the software easily, if it supports it. Some Service providers however require that the certificates in .pem or .cer formats. In this system, the certificate files themselves were renamed to reflect the default names in these configurations.

```
idp.signing.key= %{idp.home}/credentials/idp-signing.key
idp.signing.cert= %{idp.home}/credentials/idp-signing.crt
```

The third modification to the file is to change the default Authentication flow from Password to RemoteUser, as is expected by SiteMinder. The RemoteUser value depicts a login flow, the handling of which is controlled by SiteMinder. Multiple different Authentication flows can be defined and assigned to applications based on their specific needs.

The flows are defined to present the user with a login screen served by the SiteMinder WebAgent running on the Bisfed server. Some resources are also protected and unprotected with SiteMinder to either limit or allow access. These settings relate to the file /Shibboleth-idp/conf/authn/general-authn.xml where the method is defined.

```
idp.authn.flows= RemoteUser
idp.authn.flows.initial = RemoteUser
```

### 4.4.3 Changes to logback.xml

This file contains configurations about how Shibboleth logs its behaviour and inbound requests. The main configuration change here is to add the logging of assertions into a new file. The assertion is the final outbound message sent by Shibboleth to the SP and contains all the data required to validate an authentication on the application side.

The rolling policies used for archiving the logs into packed files is also edited, so that the files are saved under /archive/ in the log folder. The Appender below is used to generate the log files themselves and to enable log rotation on a daily basis;

```
<appender name="IDP_ASSERTION" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <File>${idp.logfiles}/idp-assertion.log</File>
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
    <MaxHistory>31</MaxHistory>
    <FileNamePattern>${idp.logfiles}/archive/idp-assertion-%d{yyyy-
A logger also needs to be tied to the "IDP_ASSERTION" appender identifier. In Shibboleth the assertions flow under the "PROTOCOL_MESSAGE" logger, so the configuration would be as follows;

```xml
<logger name="PROTOCOL_MESSAGE" level="DEBUG" additivity="false">
  <appender-ref ref="IDP_ASSERTION"/>
</logger>
```

With the additional configuration Shibboleth now understands to direct the generated assertions into idp-assertion log files, new ones of which are generated daily. This both saves disk space and alleviates the need of doing the process manually.

### 4.4.4 Changes to relying-party.xml

The relying-party.xml file is used for customizing the handling of Entities inside the Shibboleth IdP instances. The parameters included in the configuration below are to disable the encryption of NameID formatted attributes and assertions for the all entities using the Default configuration. Configuring a relaying party for each application is possible and sometimes needed, but if no specific configuration exists for an entity, the default is used. More edits to this file might be needed based on the application's requirements for NameID attributes, but these are explained under the corresponding section "NameID configuration".

```xml
<bean id="shibboleth.DefaultRelyingParty" parent="RelyingParty">
  <property name="profileConfigurations">
    <list>
      <bean parent="Shibboleth.SSO" p:postAuthenticationFlows="attribute-release"/>
      <bean parent="SAML2.SSO" p:encryptAssertions="false" p:encryptNameIDs="false">
        <property name="defaultAuthenticationMethods">
          <list>
          </list>
        </property>
      </bean>
    </list>
  </property>
</bean>
```
4.4.5 Changes to metadata-providers.xml

This file contains the configurations on where the metadata for each Service Provider is stored. In this environment a single file is used, so only one file needs be edited and used. Commenting out the default configuration and making a new line with the new configuration is used, as shown below;

```xml
<!-- <MetadataProvider id="LocalMetadata" xsi:type="FilesystemMetadata" metadataFile="PATH_TO_YOUR_METADATA"/> -->
<MetadataProvider id="LocalMetadata" xsi:type="FilesystemMetadata" metadataFile="/app/b2bidp/shibboleth-idp/conf/sp_metadatas.xml"/>
```

4.4.6 Changes to access-control.xml

As the system's uptime and availability is constantly monitored by another server, an entry granting access to the Status pages needs to be added.

```xml
<util:map id="shibboleth.AccessControlPolicies">
  <entry key="AccessByIPAddress">
    <bean parent="shibboleth.IPRangeAccessControl"
      p:allowedRanges="#{ {'127.0.0.1/32', '::1/128', '192.168.0.80/32'} }" />
  </entry>
</util:map>
```
4.4.7 Building Shibboleth

After all the general and personalization configurations are done, the Shibboleth application can be packed into a war file for Tomcat to use. Shibboleth includes its own shell script for this under the /shibboleth-idp/bin path called build.sh. Running the script creates a file called idp.war under the /shibboleth-idp/war/ folder. From there, the newly built file needs to be copied under Tomcat's webapp folder.

```bash
cd /app/bisfed/shibboleth-idp/bin
./build.sh

Installation Directory: [/app/bisfed/shibboleth-idp/]
Rebuilding /app/bisfed/shibboleth-idp/war/idp.war ...
...done
BUILD SUCCESSFUL
Total time: 3 seconds

cp -R /app/bisfed/shibboleth-idp//war/idp.war /app/bisfed/apache-tomcat-8.5.5/webapps/
```

Tomcat will be able to automatically unpack the war file and then host the folder so that it can be shared through the configured AJP connectors.

5 Integrating Service Providers into Shibboleth IDP

In a new integration, the first step is to exchange metadata information with the Service Provider, and to inquire what Attributes will be used to bind the users to their corresponding accounts after authentication. Just one attribute can be enough to connect an authenticated user to their account so that they are logged in correctly. It is also possible that the application uses Just-In-Time provisioning, in which case more attributes might be requested. The application might need the attributes in a specific format, depending on which software they use to read and process the SAML messages sent from the IdP. There are also two types of attributes, so called “normal” attributes that are included in the attribute statement and Name Identifier attributes, which are included earlier, in the assertion’s subject section.
5.1 Exchanging and Configuring Metadata Files

The metadata includes information about the respective parties that is needed for the creation of a trust-relationship between the IdP and the SP. The metadata files are usually exchanged between the customer when a new federation is needed, or sometimes when large modifications are made to the service provider's application. The files include URLs for the HTTP connections and the certificates required for establishing secure connections between the IdP and the application. The allows self-signed key pairs for web connections, messages and for the encryption of data.

```xml
<EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  entityId="https://ServiceProvider.com/SAML">
  <ds:Signature>...</ds:Signature>
  <SPSSODescriptor AuthnRequestsSigned="true"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <KeyDescriptor use="signing">
      <ds:KeyInfo>
        <ds:KeyName>ServiceProvider.com SSO Key</ds:KeyName>
      </ds:KeyInfo>
    </KeyDescriptor>
    <KeyDescriptor use="encryption">
      <ds:KeyInfo>
        <ds:KeyName>ServiceProvider.com Encrypt Key</ds:KeyName>
      </ds:KeyInfo>
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    </KeyDescriptor>
    <SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
      Location="https://ServiceProvider.com/SAML/SLO/Soap"/>
    <SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
      Location="https://ServiceProvider.com/SAML/SLO/Browser"
      ResponseLocation="https://ServiceProvider.com/SAML/SLO/Response"/>
    <NameIDFormat
      urn:oasis:names:tc:SAML:2.0:nameid-format:transient
    </NameIDFormat>
    <AssertionConsumerService
      isDefault="true" index="0"
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
      Location="https://ServiceProvider.com/SAML/SSO/Artifact"/>
    <AssertionConsumerService
      index="1"
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"/>
  </SPSSODescriptor>
</EntityDescriptor>
```
Location="https://ServiceProvider.com/SAML/SSO/POST"/>
<AttributeConsumingService index="0">
  <ServiceName xml:lang="en">Academic Journals R US</ServiceName>
  <RequestedAttribute NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"
    Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.7" FriendlyName="eduPersonEntitlement">
    <saml:AttributeValue>
      https://ServiceProvider.com/entitlements/123456789
    </saml:AttributeValue>
  </RequestedAttribute>
</AttributeConsumingService></SPSSODescriptor>

<Organization>
  <OrganizationName xml:lang="en">Academic Journals RUS</OrganizationName>
  <OrganizationDisplayName xml:lang="en">Academic Journals R US, a Division of Dirk Corp.</OrganizationDisplayName>
  <OrganizationURL xml:lang="en">https://ServiceProvider.com</OrganizationURL>
</Organization>
</EntityDescriptor>

Example Metadata provided by Oasis [3, 2.6]

A Shibboleth metadata file's content is enclosed in a EntityDescriptor, which starts by stating the formats and standards used, which in this case are SAML 2.0, W3 XML Signatures schema and the W3 XML Namespace. An IDPSSODescriptor then states the supported protocols, changing this to disable the version 1.1 of SAML, for example. The Extensions section of the metadata is modified with the used domain scope of the service. As the user-interface for Shibboleth is not used, the lines used for its configuration are commented out.

The KeyDescriptor sections store the X509 signing certificates used for the encryption. Entries used for message and connection encryption are marked with an use of "signing" and the one for data with "encryption". Encryption can be enabled or disabled for any entity, in the filter by configuring the used signing certifications. This is useful in testing the configuration of assertions when the customer still has not provided the needed metadata.
5.2 Adding Service Providers to Shibboleth

The contents of the customer metadata file, either exported from the application or compiled by hand, are usually added to the /shibboleth-idp/conf/SpMetadatas/ folder and then referenced to in the metadata-providers.xml file. For simplification, the metadata in this case is added as a new entity to a sp_metadatas.xml file created in Shibboleth /conf directory, which is then referred to in the metadata-providers.xml file. The file includes metadata for testshib.org by default, which functions as an example on how to configure the file. Shibboleth can connect the metadata to the attribute filter and resolver, that are used so configure sent parameters, based on the EntityID alone.

5.2.1 Attribute Resolver Configuration

The attribute resolver in Shibboleth is responsible for fetching data from the respective LDAP entries for each authenticated user. It is automatically run for each user after Shibboleth receives their username from the SiteMinder WebAgent after authentication. Based on the username, Shibboleth makes an LDAP query using an account with enough privileges and requests a configured list of attributes to be returned. The LDAP account used, the fetched attributes and the LDAP connections themselves are established by adding Data connectors to the bottom of the attribute-resolver.xml file.

```xml
<!-- START LDAP DIRECTORY CONNECTOR -->
<resolver:DataConnector id="Production-Dir" xsi:type="dc:LDAPDirectory"
ldapURL="ldaps://ldap-helsinki.sub.company.com"
baseDN="ou=Users,o=Company"
principal="cn=Shibboleth_User, ou=AdminUsers, ou=Accounts, o=Company"
principalCredential="0mitt3dCr3denti4ls">
  <dc:FilterTemplate>
    <![CDATA[(username=$requestContext.UserPrincipalName)]]>
  </dc:FilterTemplate>
  <dc:ReturnAttributes>Username AccountNumber InternalID Surname FirstName Email CompanyID</dc:ReturnAttributes>
</resolver:DataConnector>
<!-- STOP LDAP DIRECTORY CONNECTOR -->
```

In the example above, the connector would answer to internal Shibboleth requests with the id "Production-Dir" and would get fetch the Username, AccountNumber, InternalID, Surname, FirstName, Email and CompanyID from the configured URL using the account
Shibboleth_User. The search is performed based on the username, which corresponds to the UserPrincipalName in the LDAP Directory.

Attribute definitions are then used to create usable attributes from the returned values. The below examples are to create attributes with the SAML2.0 formats "emailAddress" from the raw Email attribute, and "UserID" from the raw attribute Username. The IDs configured here can be afterwards used to call the attribute in an application's filter entry and the configured name allows the Service Provider to easily fetch the values for binding. Even if the raw source attribute identifier is the same as a definition's output would be, as is the case with the Username attribute below, it needs to be defined so Shibboleth call it during filtering. Some Service Providers might require the attribute format to be unspecified, which is not preferable, but sometimes required.

<!-- Email to SAML2.0 emailAddress -->
<resolver:AttributeDefinition xsi:type="ad:Simple"
    id="Email_to_emailAddress_Mail" sourceAttributeID="Email">
    <resolver:Dependency ref="Production-dir" />
    <resolver:AttributeEncoder xsi:type="enc:SAML2StringNameID"
        xmlns="urn:mace:shibboleth:2.0:attribute:encoder" nameFormat=
        "urn:oasis:names:tc:SAML:2.0:attrname-format:emailAddress"
        name="Mail"/>
</resolver:AttributeDefinition>

<!-- Username to SAML2.0 UserID -->
<resolver:AttributeDefinition xsi:type="ad:Simple" id="Username_to_unspecified_Username" sourceAttributeID="Username">
    <resolver:Dependency ref="Production-dir" />
    <resolver:AttributeEncoder xsi:type="enc:SAML2StringNameID"
        xmlns="urn:mace:shibboleth:2.0:attribute:encoder" nameFormat=
        "urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified"
        name="Username"/>
</resolver:AttributeDefinition>

Static definitions can also be set up to send any value needed. For example in a case where a Service Provider is setup with multiple IdPs, delivering information about the user's company can be of use. Sending Static values does require source attributes to exists in the LDAP directory.
Defining such an attribute requires a Static Data Connector to be setup, so an Attribute
definition has a source to get the value from. This example above would send an Attribute
named "SourceCompany" with the value of "Great-Company". In the filter it can be called
with the id "Static_Company".

5.2.2 Attribute Filter Configuration

After the Attributes have been resolved, Shibboleth resumes to what EntityID the Au-
thentication was done for and locates the respective Attribute Filter. The filters them-
selves are configured in the file attribute-filter.xml and include the EntityID for recognizion
and Attribute Rules for defining which resolved attributes Shibboleth can include to the
assertion. In the filter below, there are three configured attributes, two of which are nor-
mal attributes that can be released without further configuration.

<!-- Application1 ServiceProvider1 || General comment and date -->
  <afp:PolicyRequirementRule xsi:type="basic:AttributeRequesterString"
    value="https://Application1.ServiceProvider1.com"/>
  <afp:AttributeRule
    attributeID="nameid_InternalID"/>
This configuration calls the attributes `nameid_InternalID`, `Email_to_emailAddress_Mail` and `Username_to_unspecified_Username` to be released for the entity. If for some reason the resolver fails to return a value for the attribute, it is simply left out of the Assertion, as the filter cannot pass an empty one through. Note that the first attribute rule refers to an attribute with a NameID format, the configuration of which is discussed in the next chapter.

5.2.3 NameId Configuration

The NameId generation in Shibboleth IdP v3 follows the SAML2 specification tightly by default, and because of this the resultant values from the default generators are opaque hashed values. For some customer implementations, this can be problematic, as their application can use the value of the NameId parameter to bind the authenticated user to an account. In some installations the application team does not even have the possibility to select the attributes used for binding. A typical use case would be to bind the user with their email address, while requesting it in an opaque format, such as a Persistent or Transient ID. Due to this, custom generators need to be added for the creation of transparent NameId attributes.

While the parameter controlling the supported NameID formats can be added per-application, a better way might be to add all of them to the default SAML2.SSO bean. This way all the entities using the SAML2 Default method can use any of the added formats, and in most integrations, the relaying-party.xml file can be left unaltered. The useable NameID formats are defined with the parameter “nameIDFormatPrecedence”. 
Per-application NameID formats can be added by using the following method.

```xml
<bean id="shibboleth.DefaultRelyingParty" parent="RelyingParty">
  <property name="profileConfigurations">
    <list>
      <bean parent="Shibboleth.SSO"/>
      <ref bean="SAML1.AttributeQuery"/>
      <ref bean="SAML1.ArtifactResolution"/>
      <bean parent="SAML2.SSO"/>
        <property name="includeAttributeStatement="true"
        assertionLifetime="PT5M"
        signResponses="true"
        signAssertions="false"
        encryptNameIds="false"
        nameIDFormatPrecedence="#{
          'urn:oasis:names:tc:SAML:2.0:nameid-format: persistent',
          'urn:oasis:names:tc:SAML:2.0:nameid-format: transient',
          'urn:oasis:names:tc:SAML:2.0:nameid-format: emailAddress',
          'urn:oasis:names:tc:SAML:2.0:nameid-format: kerberos',
          'urn:oasis:names:tc:SAML:1.1:nameid-format: emailAddress',
          'urn:oasis:names:tc:SAML:1.1:nameid-format: unspecified',
          'urn:oasis:names:tc:SAML:1.1:nameid-format: X509SubjectName',
          'urn:oasis:names:tc:SAML:1.1:nameid-format: WindowsDomainQualifiedName'}">
          <property name="defaultAuthenticationMethods">
            <list>
              <bean parent="shibboleth.SAML2AuthnContextClassRef" c:classRef="urn:oasis:names:tc:SAML:2.0:ac:classes:Password"/>
            </list>
          </property>
        </bean>
      <ref bean="SAML2.ECP"/>
      <ref bean="SAML2.Logout"/>
      <ref bean="SAML2.AttributeQuery"/>
      <ref bean="SAML2.ArtifactResolution"/>
      <ref bean="Liberty.SSOS"/>
    </list></property>
  </bean>
</bean>

Per-application NameID formats can be added by using the following method.

```xml
<bean parent="RelyingPartyByName">
  <property name="profileConfigurations">
    <list>
    </list>
  </property>
</bean>
```
The NameID generators use entries from the attribute-resolver.xml file, of which an example is shown below. Note that there is no attribute formatting, as this will be done later on in the generator. Here the email address of the user is fetched from the connected LDAP directory using this resolver. Due to normal attributes in the Resolver having an Attribute Encoder attached to them, a new attribute is added so the NameID generator can specify the format itself.

```xml
<!-- NameID PersonEmail_LDAP -->
<resolver:AttributeDefinition xsi:type="ad:Simple"
  id="nameid_InternalID" sourceAttributeID="InternalID">
  <resolver:Dependency ref="Production-dir" />
</resolver:AttributeDefinition>
```

The generators themselves need to be created into the saml-nameid.xml file. They consist of a bean referencing an internal Shibboleth dependency used in the generation process and variables such as the format of the generated NameId. Here the variable "useUnfilteredAttributes" is also set to true to allow the generator to use source attributes not listed in the application's Filter entry.

```xml
<!-- Generator from InternalID to Persistent NameID -->
<bean parent="shibboleth.SAML2AttributeSourcedGenerator"
  p:format="urn:oasis:names:tc:SAML:2.0:nameid-format:X509SubjectName"
  p:useUnfilteredAttributes="true"
  p:attributeSourceIds="#{ { 'nameid_InternalID' } }">  
  <property name="activationCondition">
    <bean parent="shibboleth.Conditions.RelyingPartyId"
      c:candidates="#{
      'https://Application1.ServiceProvider1.com',
      'https://Application2.ServiceProvider2.com'} }" />
  </property>
</bean>
```

The source attribute is set with "attributeSourceIds" and can be appended to with backup attributes to try in order, in the case that the Resolver cannot return a value for the first one.
6 End Results

After the installation and configuration processes were completed, all the components can be started and an Authentication attempted. The default scripts for starting the Apache HTTP Server and Apache Tomcat are used.

As the related configurations have been done on both the IdP and SP sides, the federation should be successful. The normal use case for the service is to access through a Portal to which the user has logged into, the they would have a valid SMSession cookie for the WebAgent to validate. In the testing that was done here however, the process was started from the IdP, so the Agent will prompt for credentials as no session exists. Figure 4 shows an example prompt that an user accessing the service with a Finnish installation of Firefox would be shown.

Figure 4. A prompt shown for an user accessing the service without a pre-existing session.

To see the traffic flow that is happening during the user’s authenticating, a Firefox-plugin called SAML Tracer is used. The plugin tracks all HTTP GET and HTTP POST requests and can decode the sent SAML messages, as well. Figure 5 shows SAML Tracer results after a login process has completed using and IdP-initiated link.
Figure 5. SAML Tracer results after a successful login using an IdP-initiated link.

The process can be started with an IdP-initiated link, which includes the EntityID of the application in the URL, referred to by “providerID”. The process can be started from any web browser that has network access to the service. For “Application1” the link would look similar to https://bisfed.ext.company.com/idp/profile/SAML2/Unsolicited/SSO/providerID=https://Application1.ServiceProvider1.com. Once accessed, an AuthNRequest message is simulated by Shibboleth and the authentication process begins.

6.1 Assertion

An assertion will at this point be created by Shibboleth based on the configuration that was done for the application. It should contain all the information needed to log the user in to the application itself. All certificates such have been omitted from the assertion. The important values have been bolded and enlarged for better visibility.

```xml
    ID="23000073Bbc799fa2000f30008e92000"
    IssueInstant="2017-01-07T12:00:17.958Z" Version="2.0"
    xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
        https://bisfed.ext.company.com/idp/shibboleth
    </saml2:Issuer>
</saml2p:Response>
```
If the assertion is accepted on the application side, the user should be logged into the system, as illustrated in Figure 5. At this stage, all the processes related to the instance
have been successfully completed and the service has functioned as intended. Figure 6 shows an example page that a service provider could show after a user has logged in.

![Example Page](image)

**Figure 6.** An example of a page describing a successful login.

After the solution was implemented in the production environment, there has been constant use to the integrated applications, and the service has performed well, offering users access to the web applications. Running on a load-balanced data centre infrastructure, the service’s performance should not be an issue, even when under stress from a large number of active users.

### 6.2 Possible Improvements

In an environment, such as this, where a centralized login page is used, there is little need for expanding the service further. It is possible however, to modify Shibboleth to support multiple login pages by adding additional external login paths that would then be protected with the SiteMinder agent. Applications could be configured to use a specific authentication path and the SiteMinder WebAgent would then offer the corresponding login page. The functionality would offer a solution for more complex authentication schemes, such as multifactor, or machine certificate authentication, if needed.

It would also be good to install the software using a package manager, instead of manually downloading the packets. At least for the Apache web server and Tomcat, the process should be quite simple using RPM Package Manager or another piece of similar
software. This way the software could be updated quite easily, without the need for installing it manually each time the version needs to be changed. This solution might however be harder to configure, as the installation locations in this environment are customized.

6.3 Conclusions

The federation environment should offer SAML2-based authentication to any compliant service providers, including cloud providers such as Azure and Salesforce. The service is quite easy to maintain and should only require admin intervention for an application integration, or in the case of component malfunctions. As there are two separate servers running behind a load-balancer, the availability of the environment is also ensured.

The built environment allows users of the business portal to be logged into the integrated applications automatically. As the need of users inputting their credentials is removed, and secure connections are used, the new login process can also be thought to be more secure than the previous. The number of needed logins was minimized to just one, increasing the ease of use of the business portal while simultaneously saving time.

The client has been satisfied with the end result and the federation environment has been in constant use after the first integrations were done.
References

The references mentioned in the thesis are listed here in the order of their appearance.


