HAAGA-HELIA
University of Applied Sciences

OpenIG SSO Authentication Implementation for Web Portal

Timi Kohonen

Thesis
Degree programme in
Information Technology
2013

HAAGA-HELIA
ammattikorkeakoulu

Päivämäärä

Tietojenkäsittelyn koulutusohjelma

Tekijä tai tekijät Timi Kohonen	**Ryhmä tai aloitusvuosi** 2010
Opinnäytetyön nimi Kertakirjautumisen toteutus OpenIG:n avulla verkkopalveluun Case: Trusteq	**Sivu- ja liitesivumäärä**
Ohjaaja tai ohjaajat Petri Hirvonen – Haaga-Helia Saila Suvanto – Trusteq OY	

Tämä työ tutkii kertakirjautumisen toteuttamista Redmine verkkopalveluun OpenIG:n avulla, joka haluttiin suojata käyttäjien tunnistamiseen ja valtuuttamiseen käytettävällä Security Assertion Markup Language -standardilla. Opinnäytetyössä keskitytään erityisesti OpenIG-tuotteeseen ja sillä toteutettavaan Federaatioon eli luottamusverkostoon. OpenIG on avoimen lähdekoodin edustapalvelin, jota voidaan hyödyntää erilaisissa tunnistautumiseen liittyvissä ratkaisuissa. OpenIG asennetaan aina kohde web-sovelluksen edustapalvelimeksi.

Opinnäytetyön tehtävänä on selvittää miten OpenIG soveltuu tietoturvajärjestelmiin erikoistuneen yrityksen käyttöön. Työ tehtiin itsenäisenä projektina ja toteutettiin virtuaaliympäristössä, jossa OpenIG asennettiin toimimaan Trusteq Connect -tunnistautumispalvelun ja Redmine web-sovelluksen kanssa.

Työn ensimmäinen puolisko keskittyy projektin motivaatioon, sekä työssä käytettävään termistöön. Siinä käydään läpi myös työssä käytettäviä ohjelmia ja sovelluksia. Jälkimmäisessä osassa esitellään OpenIG sekä sillä toteutettu projekti, joka keskittyy kehityspuolella tehtyihin määrityksiin ja esittelee myös työssä tarvittavat konfiguraatiot. Lopussa olevista liitteistä löytyy tietoa liittyen tehtyihin asennustoimenpiteisiin.

Informaatioteknologiassa toteutettavat täysin uudet projektit ovat haasteellisia ja tarkkoja aikatauluja voi olla hankala arvioida uusia tuotteita kokeiltaessa. OpenIG osoittautui toimivaksi tuotteeksi ja sen avulla toteutettu Federaatio-malli saatiin valmiiksi suunnitellussa aikataulussa. Projektista saatuja tuloksia voidaan mahdollisesti hyödyntää myös tulevaisuudessa.

Asiasanat
OpenIG, Federaatio, Kertakirjautuminen, SAML, Tunnistautuminen

HAAGA-HELIA
University of Applied Sciences

Date of presentation

Degree programme in Information Technology

Author(s) Timi Kohonen	**Group or year of entry** 2010
The title of thesis OpenIG SSO Authentication Implementation for web portal Case: Trusteq	**Number of report pages and attachment pages**
Advisor(s) Petri Hirvonen – Haaga Helia Saila Suvanto – Trusteq OY	

This thesis studies Open Identity Gateway (OpenIG) Single Sign-On authentication implementation for Redmine web portal secured with Security Assertion Markup Language. The focus of the study is in the OpenIG product and Federation that was implemented by using the gateway. The OpenIG is an open source high-performance reverse proxy server. The product has credential replay and session management functionality and it is always configured to operate as a reverse proxy server for the target web application.

The motivation of the study was to discover how suitable OpenIG is for the use of information security company. The work was implemented as an independent project. The OpenIG was configured to operate with the Trusteq Connect authentication service and Redmine web application.

The first part of the thesis focuses to the motivation of the work. The major concepts, applications and software are also included there. The last part introduces OpenIG and the project that was implemented by using it. The project focus is on the development environment and it presents configurations that were made. The appendices in the end contain information about the configurations.

Completely new projects are challenging in the information technology industry and it is very difficult to compose accurate schedules for them. The OpenIG product proved to be a working Federation tool and the project was implemented within the prescribed time limit. The project results may offer some benefits in the future.

Key words
OpenIG, Federation, Single Sign-On, SAML, Authentication

Table of contents

1 Introduction

The purpose of this thesis was to discover OpenIG product and its functionality for the use of information security enterprise Trusteq and its services. The project was started by learning how OpenIG operates. The first proto type was built in a virtual machine environment, where all the testing was executed. The final result was a working proxy server using the Single Sign-On (SSO) secured with the Secure Assertion Markup Language (SAML). The working product was planned to be transferred in the organizations Federation Service that runs on Amazon Web Services (AWS).

This project was accomplished for Trusteq Oy that is a company specializing in identity and access management (IAM). Trusteq builds and designs solutions for its customers where the focus is on privacy and data protection. The company was founded in 2003 and had approximately 40 employees during the time this project was implemented. Trusteq consists of different departments and project groups that produce services for their specific customers. (Trusteq 2013.)

This project was executed to facilitate login to the target web application. It was produced under the project group Team Connect that was composed of identity management experts and developers. Team Connect has developed a Cloud computing and Identity Management Service (IMS) product Trusteq Connect. Redmine is a project management web application that is used for debugging in Trusteq development. The goal of this study was to explore the functionality of OpenIG and produce an exact documentation how to implement a working Federation Service using OpenIG product between the Trusteq Connect identity provider and a Redmine web application in Trusteq's network environment.

The first chapter of this thesis introduces the project motivation and defines the process. All the major concepts are determined in the second chapter "Theoretical concepts". The next chapter presents OpenIG and its functionality. Fourth chapter "Project implementation" describes accurately the installation part of the process in the virtual machine environment and components functionality. The last chapter summaries all previous and presents the conclusion.

Terminology

API	Application Program Interface
AWS	Amazon Web Services
CoT	Circle of Trust
CRUD	Create, Read, Update, Delete
Federation	Certain kind of Information Security Implementation
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IAM	Identity and Access Management
IDM	Identity Management Service
IDP	Identity Provider
ISV	Independent Software Vendor
Java SPIs	Java Service Provider Interfaces
JCE	Java Cryptography Extension
JDK	Java Development Kit
JSON	Java Script Object Notation
NAT	Network Address Translati**on**
OpenIG	Open Identity Gateway
REST	Representational State Transfer
SaaS	Software-as-a-Service
SAML	Security Assertion Mark Language
SP	Service Provider
SSL	Security Sockets Locker
SSO	Single Sign-On
STS	Security Token Service
URI	Uniform Resource Identifier
WS	Web Service
XML	Extensible Markup Language

2 Motivation and Case introduction

This chapter introduces the project starting point and describes the motivation of the project. It also contains the research problem description and the presentation of the implemented use case. The project is defined accurately in the later part of this chapter. The last part presents hardware and software that were used for this implementation.

2.1 Starting Point

At the beginning of the project the OpenIG was a quite new acquaintance for all the members of Team Connect. However the technical advisors of Trusteq had discovered that the OpenIG offers a well-designed open source solution for SSO authentication with multiple features. Other individual Identity Gateway products such as IBM Tivoli Federated Identity Manager Business Gateway are enterprise versions. Additionally, the OpenIG is very transforming for different kind of Federation[1] services. OpenIG can be configured for various functions by using Java programming language. Nearly all of the OpenIG documentation was found from ForgeRock's official website and forum.

The research problem was that the SSO connection to the Redmine web application did not work with the SAML and the login to the target web application desired to be accomplished straight using the same identity credentials as the Identity Provider Trusteq Connect service. The start-up plan was to implement the working OpenIG environment into a virtual machine and execute a functional testing there. The accomplished platform was planned to be transferred into organization's production use.

The project goal was to discover how to implement a Single Sign-On (SSO) authentication secured with Security Assertion Markup Language (SAML) using OpenIG proxy server, where Trusteq Connect operated as an Identity Provider (IDP) and the target web application was a Redmine. Both of the products were running in Trusteq's Amazon Web Services (AWS). When the project was started, it was not confirmed that the OpenIG implementation would work.

[1] Term Federation is determined in the next chapter "Theoretical concepts"

2.2 Use Case

In this project the OpenIG was configured to work as a Federation Service with additional Java configuration in the authentication that parses the SAML assertion and associates with the Redmine. The use case contains the description how the OpenIG proxy server operates when the user attempts to get into the target web application (Figure 1 Use Case).

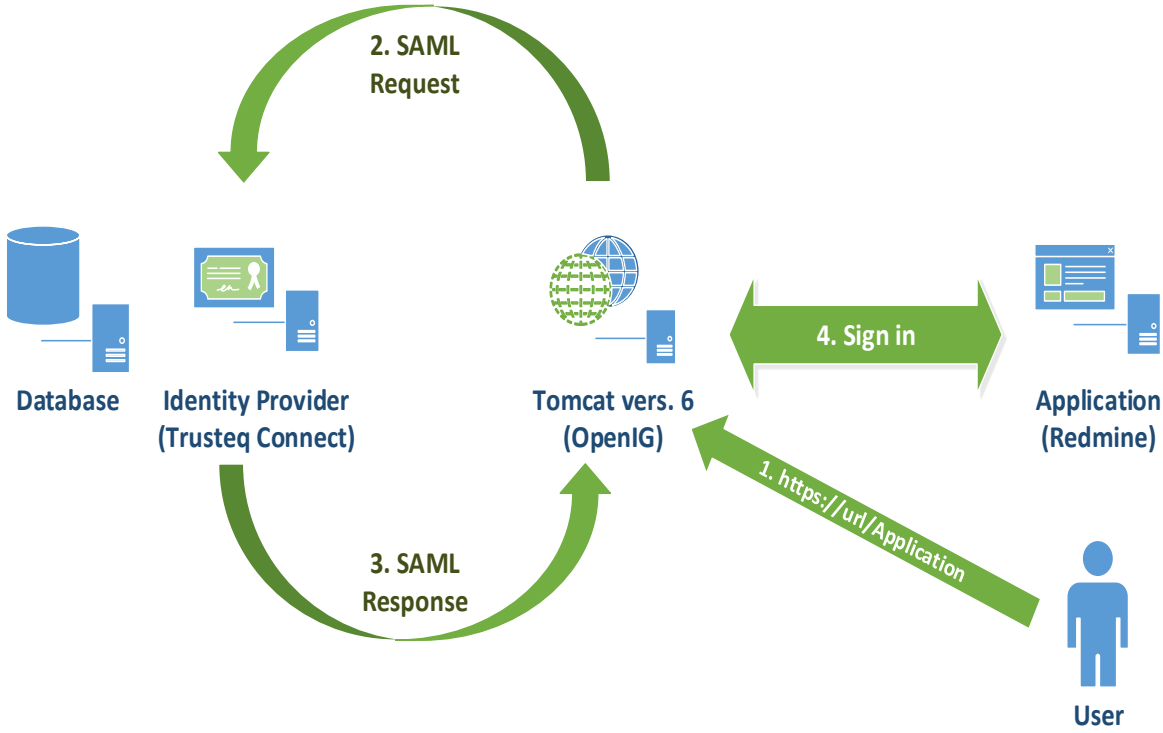


Figure 1 Use Case

First the user will create a connection to the Tomcat[2] server by typing the Internet address to a browser (1), which sends the SAML Request to Identity Provider (IDP) (2). Secondly the IDP will process the Request that is redirected to the proxy server as the SAML Response (3). Finally, the OpenIG server will execute all configured login operations in the exchange and sign in the user to the target application (4).

[2] Tomcat is a web container where the OpenIG has been configured

2.3 Defining Project

This project explores how OpenIG works with the target web application Redmine and the Identity Provider Trusteq Connect. All the testing was made in Trusteq's external network. The project was narrowed down to only concern Redmine even though it could be used in other environments as well. OpenIG probably acts very differently with various components and software. The focus will be on the virtual machine system where the OpenIG was installed in this project. This thesis will not present how the configured implementation was transferred into Amazon Web Services. Modifications into the identity provider and the Redmine servers were made by the technical advisors and not by the researcher of OpenIG.

2.4 Equipment and Software

This project was implemented by using Oracle VM VirtualBox virtualization software that was configured into a laptop computer running Windows 7 Operating System. OpenIG was configured into the virtual environment using Linux CentOS 6.4 platform. All of the programs used for the OpenIG implementation in the virtual environment had an open source license. All the hardware equipment, information processing tools and software were provided by the Trusteq.

3 Theoretical concepts

This chapter introduces the main concepts contained in the OpenIG implementation. The first part contains information about the Federation and all the primary concepts involved. The latter part presents the tools that were used during the project which includes the target virtualization software, operating system, web application and Identity Provider.

3.1 Federation

In the information technology a Federation term indicates to a certain kind of Circle of Trust (CoT) where all actors have been associated. The federation provides communicating between different organization services by using only one authentication. The federation reduces login credentials and improves the information security eventfully in organizations. Corporations are allowed to share specified applications, with resultant cost savings and consolidation of resources. The federation works as a SSO extension and it makes the SSO possible to internal services. The main federation standards are SAML, WS-Federation and Shibboleth. (J. Lauhia, L. Tielinen. 2009.)

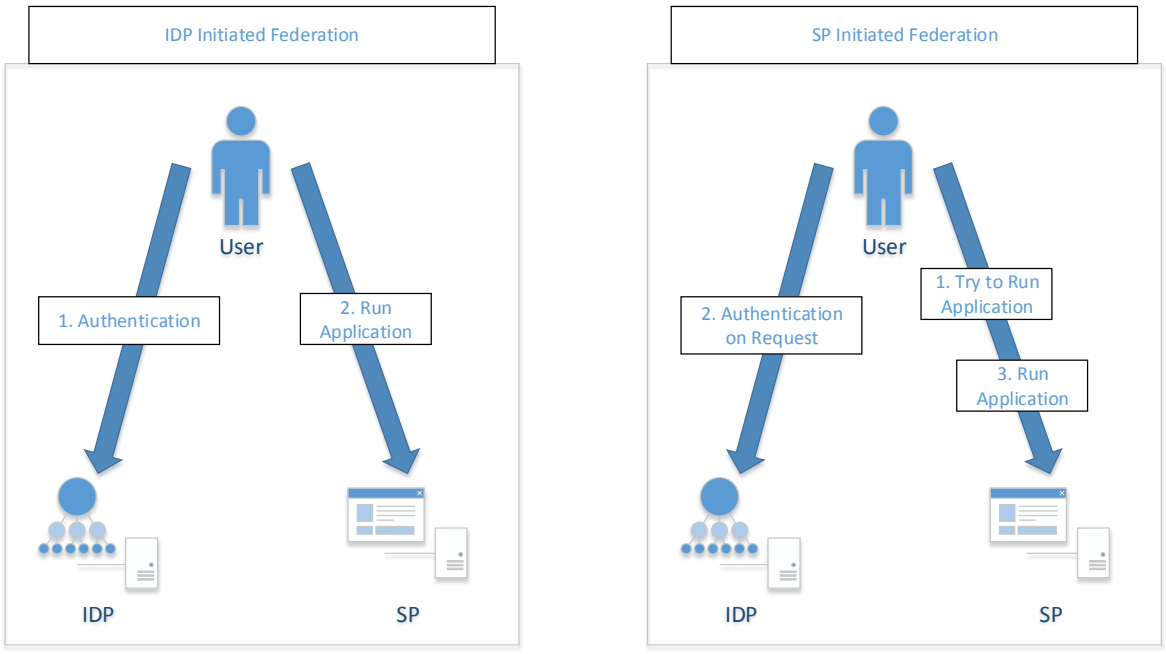


Figure 2 Regular Federation Models (J. Lauhia, L. Tielinen 2009)

IDP Initiated Federation and SP Initiated Federation are the most typical federation models (Figure 3 Regular Federation Models). In the IDP Initiated Federation model the federation is activated on the IDP. The federation is activated on the SP in the SP Initiated Federation model where the user is sent to IDP for authentication. (J. Lauhia, L. Tielinen. 2009.)

Federation gathers together processes and technologies that support the secured Cicle of Trust within the context of SSO. Federated SSO allows user to login once and then access to defined resources. The federation is "An association comprising any number of service providers and identity providers." according to Organization for the Advancement of Structured Information (OASIS), the creator of SAML (empowerID 2013).

3.1.1 Security Sockets Layer (SSL)

Security Sockets Layer (SSL) is a cryptographic protocol for managing the message transmission security on the Internet developed by Netscape Corporation. It confirms the privacy between an application and its users to protect from the third-party software. SSL usually operates between server and client to ensure a secure the connection. The secure hypertext transfer protocol (HTTPS) is using SSL that is invoked on Web Server. The SSL guarantees that the authentication, message privacy and message integrity are safe. It allows secured exchanges sensitive information which increases the business. (M S.Bhiogade 2002.)

Transport Layer Security (TLS) is based-on Security Sockets Layer (SSL). TLS and SSL are an integral part of most Web browsers and servers but the protocols security measures varies. TLS and SSL are widely implemented in several open source software projects such as OpenSSL. (IBM. Transport Layer Security 2012.)

3.1.2 Extensible Markup Language (XML)

Extensible Markup Language (XML) is a markup language that is developed by World Wide Web Consortium (W3C). It defines an array of rules for encoding documents in a very flexible format. XML is very similar to Hypertext Markup Language (HTML), but it has been designed for the representation of arbitrary data structures on the Internet. XML describes a class of data objects called XML documents that include a parsed or unparsed data. XML processor operates in the application and is a software module that provides the access to XML documents. (T. Bray, Jean Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. 2006.)

3.1.3 Security Assertion Markup Language (SAML)

Security Assertion Markup Language (SAML) is the most used protocol for federated identity deployments. Many enterprises, government agencies and service providers have chosen to their standard protocol for communicating identities across the internet. SAML is based-on Extensible Markup Language (XML) which makes it a very flexible in various situations. Two operations can communicate and share every kind of identity attributes in a SAML assertion while they are using the XML. The SAML has a large flexibility and it can be incorporated into other standards (Ping Identity 2013.)

The large compatibility of SAML makes it preferred in the SSO industry. Enterprises are able to create SSO implementations by using the SAML that can also be used with several federation partners. It will conform easily to Software-as-a-Service (SaaS) applications and many various external service providers. (Ping Identity 2013).

3.1.4 Service Provider and Identity Provider

A Service Provider (SP) is a corporate entity that provides various Web Services (WS) and it is usually referred to a third party or outsourced suppliers including Application Service Providers (ASP), Storage Service Providers (SSP), and Internet Service Providers (ISP). An authorization and authentication comes from a trusted Identity Provider (IDP) or Security Token Service (STS). A Security Token Service also known as Identity Tokens is a web service that is used for authorizing and authenticating. The

security information is routed between a SP and IDP by using SAML that ensures the protected connection. (empowerID 2013.)

An Identity Provider (IDP) or Identity Assertion Provider is a website or network service that authenticates users on the Internet. An authentication module verifies a security token such as SAML. Most of the major website providers including Google, Facebook and Yahoo are also identity providers. The Identity Provider works as a Security Token Service in the WS-Federation Model. OASIS organization has determined Identity Provider as:

> **"A kind of provider that creates, maintains, and manages identity information for principals and provides principal authentication to other service providers within a federation, such as with web browser profiles."**

Both IDP and SP can be related to Identity Management, but IDP has more functionality. (empowerID 2013.)

3.1.5 WS-Federation

WS-Federation (Web Services Federation) is a federation protocol developed by the several technology corporations such as IBM and Microsoft Corporation. Microsoft is the largest WS-Federation vendor which has invested a lot of effort into incorporating WS-Federation into its products. Even though the WS-Federation is very similar to SAML, the protocols are still incompatible. Furthermore, WS-Federation terminology differs from the SAML and for example the Service Provider is called a Relying Part (RP) in the WS-Federation Model. (empowerID 2013 & AssureBridge 2012.)

3.1.6 Shibboleth

Shibboleth is an open source federated Identity Management (IM) system developed by the Internet2 community. It allows user data to be sent to remote services where the user data is generally used to authorization, customization or authentication in different services. Identity Provider and Service Provider are the main components in Shibboleth implementation. The Providers communicate by using Metadata files that

determine the Provider properties and location. Generally Federation associated Providers are configured to rewrite the metadata regularly to ensure that the members of Circle of Trust are updated. (Shibboleth 2013).

3.2 VirtualBox

Oracle VM VirtualBox is a high performance virtualization product developed by Sun Microsystems, but nowadays it is part of Oracle Corporation. VirtualBox is an application that is installed on an existing host operating system which allows operating systems to be run and loaded. VirtualBox has several platform packages for Windows, Linux, OS X and Solaris operating systems which all are free. Virtualization allows running multiple operating systems simultaneously and infrastructure unification. Furthermore, it facilitates testing, disaster recovery and it makes software installation more flexible. (VirtualBox 2013.)

3.3 CentOS

Community Enterprise Operating System (CentOS) is a free enterprise class computing platform based on rebuild of Red Hat Enterprise Linux open sources. The first stable version of CentOS was released in 2004. The purpose of CentOS is to offer a free capable operating system for enterprise use. CentOS versions are updated regularly approximately in every 6 months which makes it a secure, reliable, low-maintenance, predictable and reproducible Linux environment for its users. The community mainly provides the technical support for CentOS users and the information is distributed in the official web forums and mailings lists. (CentOS 2013.)

3.4 Redmine

Redmine is an Open source project management and bug-tracking tool, which is completely free. The first release was founded in 2006 and it was developed by using Runs on Rails framework. The design of the web view is remarkably influenced by Trac (Trac 2012). Enterprises are able to manage multiple projects with the Redmine that includes issue tracking, integrated project management features and support for different version control systems. (Redmine 2013.) Trusteq uses the Redmine for internal project managing by writing new features, bugs and tasks there.

Redmine supports REST API and exposes part of its data through it. This function provides create, read, update and delete (CRUD) request methods for specified resources. The REST API interacts with cURL that is a command-line tool for transferring data between separate protocols. Redmine REST API supports only JSON and XML protocols. (Redmine. Rest API. 2013.)

3.5 Trusteq Connect

Trusteq connect is an open source web application project using SSO into the target web applications. It is a SaaS product which provides identity and access management as a service to all sorts of companies. Trusteq Connect enables flexible and secure access to the digital (web) services the organization is using. This means that the service providers can enable access to their services without creating complicated identity and access management solutions. End-users need only one login to access all the services their organization has set in their network. (Trusteq 2013.)

Trusteq Connect provides strong secured authentication for its users using different authenticate methods such as Google Authenticator mobile application. Trusteq Connect is a solution to build a working IAM and SSO environment for a company. It can be configured to use an existing user data. Access management is based-on the identity data. The service determines the permissions to each user and group. In other words, the product can determine individual permissions to every user in the organization. (Trusteq 2013.)

4 OpenIG product

This chapter introduces the OpenIG and its components that are represented with a detailed information. ForgeRock , the developer of OpenIG, is also represented in a short section. The chapter also presents the functionality of OpenIG in the different cases and two ordinary Federation illustrations are introduced in the end.

4.1 Open Identity Gateway

OpenIG (Open Identity Gateway) is a component that is developed by ForgeRock. OpenIG has been evolved as a part of OpenAM that is an open source entitlements, access management and federation server platform product (ForgeRock 2013). It can be used in several identity cases. By default it works as a proxy server, which enables SSO for a target application. (OpenIG 2013).

Without making any major changes into the configuration, OpenIG is a Java-based reverse proxy which enables access to a web application. The reverse proxy is a type of proxy server that associates with a client to make it more efficient (IBM 2010). All HTTP traffic to target web applications goes through the configured OpenIG server, which enables transformation, close inspection and filtering for every request. (OpenIG 2013.)

By using the Form-Filter module, OpenIG automatically log users in when an authentication or timeout page is recognized. Enabling the SAML2 service makes it an endpoint of SAML2. After this operation it can receive and verify the SAML2 requests, which allow users to log in to the target application without using any other authentication. (OpenIG 2013.)

There is no definite standard to introduce functionality of OpenIG and its usage depends on implementation. The main concept is that all the requests to the target application are routed through the reverse proxy. This requires modifying the DNS entry for the application. The following case above (Figure 4 OpenIG Functionality) introduces how the connection proceeds between the user, OpenIG and portal while the user attempts to login to the web application. (OpenIG 2013.)

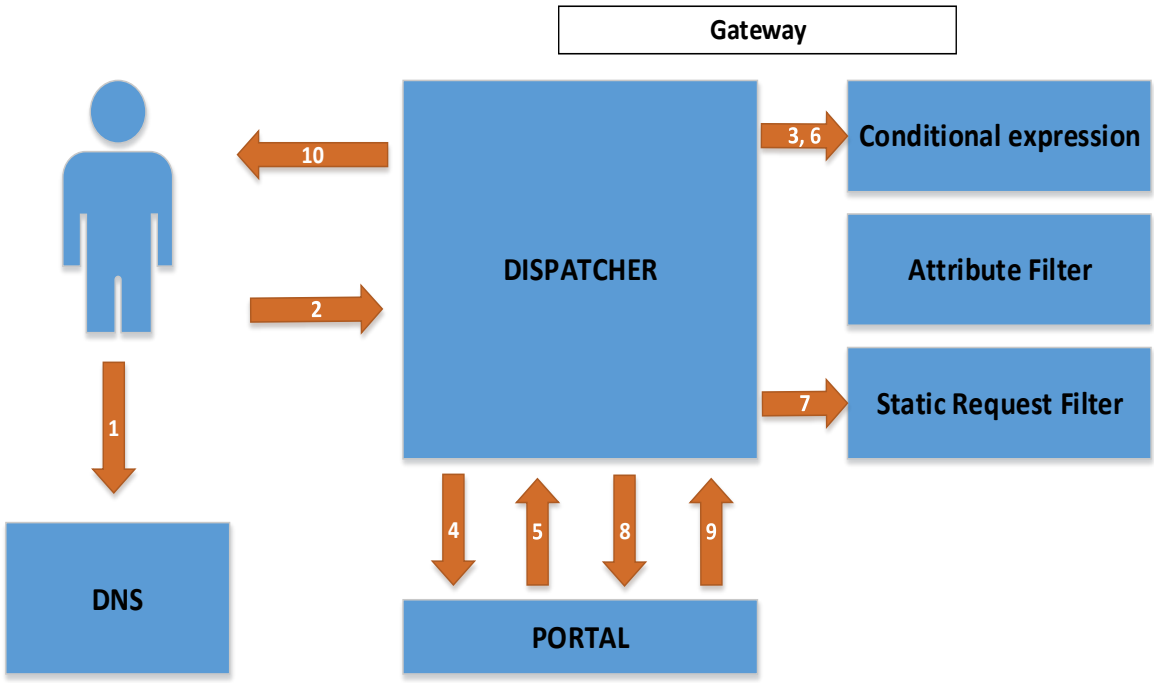


Figure 4 OpenIG Functionality (OpenIG 2013)

Figure 4 is a detailed flow which presents OpenIG functionality when the user tries to login into the portal at first time. Portal is the end page of the target web application.

1. Browser host creates a DNS request by typing the URI to achieve the IP address for OpenIG. DNS will returns the IP address of OpenIG.
2. Browser dispatches the request to OpenIG
3. OpenIG supervises the request and detects that no login page match.
4. The original request is forwarded to the Portal
5. Portal does not detect any local application session and redirects to the Portal login page.
6. OpenIG handles the redirect request and detects a login page match.
7. OpenIG produces the login form.
8. OpenIG sends (POST) the login form to the Portal.
9. Authentication is validated and forwarded to the Portal end user page.
10. The final redirect is in the OpenIG that sends it back to the Browser.

(OpenIG 2013)

4.2 ForgeRock

ForgeRock is an Independent Software Vendor (ISV) that was founded in 2010. The founder developers consisted of former Sun business and technology experts and their goal was to deliver the finest open source identity stack that is also very well secured. Nowadays Forgerock is the global open source producer. The enterprise has developed several identity-oriented service solutions for thousands of the world's largest companies and government organizations. Their primary Open Identity Stack consists of three identity and access management products (OpenAM, OpenDJ and OpenIDM), which have a common application program interfaces (APIs) and user interfaces. ForgeRock technology is built on open standards and deployed by a worldwide network of system integrator, training partners and consulting. (ForgeRock 2013.)

4.3 OpenIG Core and Configuration

The purpose of OpenIG core is to work as a reverse proxy to the target application. It is a standard Java EE servlet implementation of a reverse proxy. OpenIG core is able to transform, search and process HTTP traffic that goes or comes to the target application. It has ability to submit login forms, login pages, transform or filter content and possibility to work as a Federation endpoint for the application. This all can be produced without configuring the application or its container. (OpenIG 2013.)

OpenIG can be configured easily, because it is very self-contained and transforming. Configuration will be produced by JavaScript Object Notation (JSON) language. JSON is an open standard format that is easy for humans to write and read, but also easy for machines to generate and parse. JSON is a completely language independent data format which uses conventions in a large variety of programming languages and the filename extension is ".json". By default OpenIG looks for config.json file in the home directory. (OpenIG 2013 & JSON 2009.)

Every module stores its own data in JSON representation and the data locates in flat files. The OpenIG features have ability to be modified straight by making changes into the JSON flat files. Each configured OpenIG JSON module has a Heaplet connected

into it. Heaplet takes care for reading the JSON configuration and it also creates configuration in OpenIG's runtime heap for the relevant module. The modules can read their configuration from the heap or make shared configuration details open to some other modules. (OpenIG 2013.)

4.3.1 Exchange & Dispatcher

OpenIG uses Exchange wrapper around the HTTP response and request objects. These objects pass through the OpenIG system and all responses and requests can be modified or accessed there. It is also possible to add any kind of data into the Exchange to help the data going through filters and handlers. (OpenIG 2013.)

The Dispatcher works like the internal router for OpenIG. All requests are handled and sent forward on to the modified chain of filters and handlers. The component (for example the target host, URL, headers, URL parameters, cookies etc.) of a request will determine how it will be forwarded in the router. (OpenIG 2013.)

```
55      {
56          "name": "DispatchHandler",
57          "type": "DispatchHandler",
58          "config": {
59              "bindings": [
60
61                  {
62                      "condition": "${exchange.request.uri.path != '/redmine/login'}",
63                      "handler": "ClientHandler"
64                  },
65                  {
66                      "condition": "${empty exchange.session.username attribute}",
67                      "handler": "SPInitiatedSSORedirectHandler"
68                  },
69                  {
70                      "condition": "${exchange.request.uri.path == '/redmine/login'}",
71                      "handler": "LoginChain"
72                  },
73                  {
74                      "handler": "OutgoingChain"
75                  }
76
77              ]
78          }
79      },
```

Figure 5 DispatcherHandler – config.json

The Dispatcher (Figure 5) was written to config.json in the complete OpenIG implementation. The exchange is dispatched to associate with four different handlers.

4.3.2 Chain, Handlers and Filters

A Chain includes at least one Filter and a Handler. The handler looks over an incoming request from the Dispatcher. For further processing the handler gives the request on to a different Chain or to the configured application. For example requests are sent to Login Chain (Figure 6) that runs a list of Filters and then dispatches the exchange to the next handler. (OpenIG 2013.)

```
90    {
91        "name": "LoginChain",
92        "type": "Chain",
93        "config": {
94            "filters": ["SwitchFilter","RedmineUserFilter","HiddenValueExtract","LoginRequestFilter2"],
95                "handler": "OutgoingChain"
96        }
97    },
```

Figure 6 LoginChain – config.json

OutgoingChain (Figure 7) is regularly the last Chain and it dispatches the exchange to ClientHander. Capture Filter writes a log and is typically removed when the OpenIG is transferred in production so that it does not fill the available disk space. (OpenIG 2013.)

```
172     {
173         "name": "OutgoingChain",
174         "type": "Chain",
175         "config": {
176             "filters": ["CaptureFilter"],
177             "handler": "ClientHandler",
178             "file": "/tmp/outgoing.log"
179         }
180     },
```

Figure 7 OutgoingChain –config.json

Every Chain makes its final processing by sending a call to a Handler. A Handler will call some other Chain or it has ability to send the request on to the configured web application. There are only three Handlers that come with the OpenIG and others must be configured into it. ClientHandler (Figure 8) always does the last request on to the target application. (OpenIG 2013.)

```
196     {
197         "name": "ClientHandler",
198         "type": "ClientHandler",
199         "config": {
200             "filters": ["LocationRewriter"]
201     }
202     }
203     ]
204   },
```

Figure 8 ClientHandler –config.json

Filters take care of HTTP requests in OpenIG. Developers are able to bound Filters together and use them to act as the input stream coming from the browser or the reverse stream that returns back from the determined application. Filters can process simple tasks like a logging and more complicated tasks such as transforming content or retrieving user attributes. OpenIG offers several different kinds of Filters that are ready to use. Those filters can be combined in chains. Developers can write custom filters by using the Java Service Provider Interfaces (Java SPIs). (OpenIG 2013.)

```
156     {
157         "name": "LoginRequestFilter",
158         "type": "StaticRequestFilter",
159         "config": {
160             "method": "POST",
161             "uri": "http://localhost/redmine/login",
162         "form": {
163                 "username": ["GET-username"],
164                 "password": ["GET-password"],
165                 "authenticity_token": ["${exchange.hiddenValue.value}"],
166                 "back_url": ["http://localhost/redmine/"],
167                         "login": ["login attribute"],
168                         "utf8": ["utf8 attribute"]
169               }
170         }
171     },
```

Figure 9 LoginRequestFilter –config.json

LoginRequestFilter (Figure 9) is a StaticRequestFilter that creates a new login request and replaces any other request that already exists in the exchange. This filter includes a form parameter which contains specified field names from the target application. (OpenIG 2013.)

```
113     {
114     "name": "HiddenValueExtract",
115     "type": "EntityExtractFilter",
116     "config": {
117         "messageType": "response",
118         "target": "${exchange.hiddenValue}",
119         "bindings":  [
120             {
121             "key": "value",
122             "pattern": "authenticity_token\"\s.*value=\"(.*)\"",
123             "template": "$1"
124             }
125         ]
126     }
127     },
```

Figure 10 HiddenValueExtract –filter – config.json

HiddenValueExtract (Figure 10) is an EntityExtractFilter that extracts hidden value form the login page. The hidden value is contained in the login form that is used in the POST method.

4.3.3 Services and Federation

The Services gives permission to the integration of additional features. This gives the services, such as Federation, opportunity to be a part of OpenIG implementation without massive configuration into it. For example SAML2 is one of the additional services that OpenIG supports. (OpenIG 2013).

OpenIG can act as the Service Provider by enabling the Federation Service. It creates a Circle of Trust with a SAML2 adaptable Identity Provider, where both IDP (Identity Provider) and SP (Service Provider) initiated SAML2 profiles are supported. This means that the IDP and SP can be separate companies or domains like in the standard Federation use case. The remote Identity Provider gives an unsolicited Authentication statement to OpenIG in IDP initiated Single Sign-On. In Service Provider initiated Single Sign-On the configured OpenIG calls the Federation Service to initiate Federated SSO with the determined Identity Provider. In both cases, the Federation Services task is to validate the user and let through the required attributes to Identity Gateway to make the logging possible to the web application. (OpenIG 2013.)

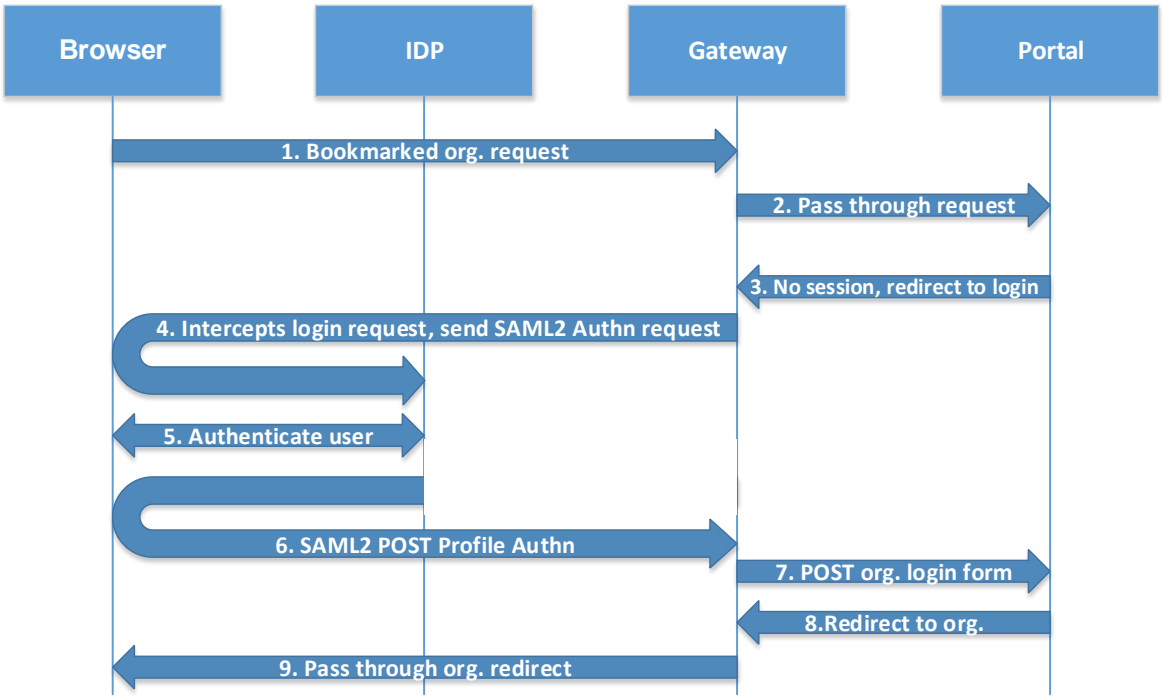


Figure 11 Federation Gateway SP Initiated SAML2 SSO (OpenIG 2013).

In the following figure above (Figure 12), the Federation Gateway provides SAML2 endpoint features operating as Service Provider in an SP initiated Single Sign-On implementation and the target application is provided by an external supplier. First user creates a connection to the target web application. The Federation Gateway recognizes the request and sends it through because no login page is found. The target application redirects the request to its login page. Federation Gateway finds a match for the login page and issues an SP initiated SSO SAML2 request to the target IDP service. The SAML2 authentication request is sent to IDP that authenticates the user. After that the authentication credentials are dispatched to OpenIG. The Federation Gateway validates the assertion and transforms those attributes available to the OpenIG login chain. The chain receives the user credentials and dispatches the login form to the web application. The target application handles the credentials and redirects the user to its home page. (OpenIG 2013.)

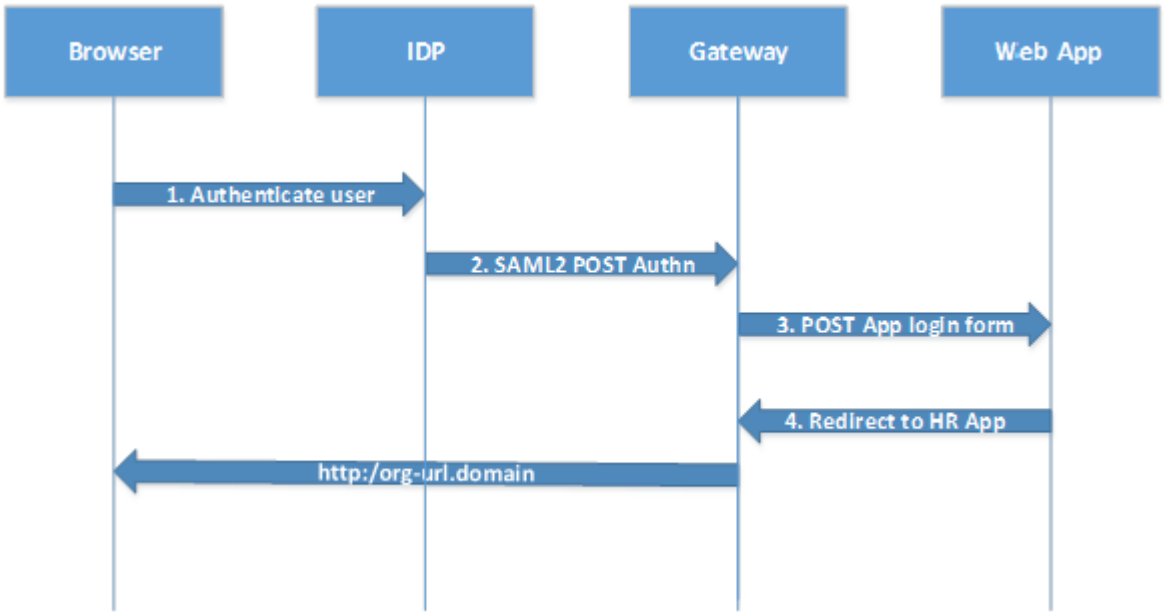


Figure 12 Federation Gateway IDP Initiated SAML2 SSO (OpenIG 2013).

In the following figure above (Figure 11), the OpenIG Federation Gateway provides SAML2 endpoint features in an IDP initiated Single Sign-On configuration while operating as a Service Provider. First user navigates to the target application address and is redirected to the IDP authentication. IDP dispatches authentication response to the target web application. Federation Gateway receives and validates the SAML assertion. It also does the attributes available to the OpenIG login chain that retrieves the user credentials and dispatches the login form to the target web application using POST method. The target web application validates the login credentials and the user is redirected to applications primary page. (OpenIG 2013.)

5 Project implementation

The following chapter introduces the installation of OpenIG and its functionality in the implemented virtual environment. Additionally, it presents how the OpenIG environment was built and detailed information about the RedmineUserFilter that is a custom heap object Filter developed by using Java.

The implementation was started with the virtual environment configuration which was built in the host computer by using virtualization software VirtualBox. The operating system CentOS 6.4 was installed into the virtual machine and all the necessary development kits including network and software updates were configured there. The next task was to get acquainted with the OpenIG product and explore how it should be configured to function exact in the Trusteq's identity management environment. The OpenIG required a web container to operate and in consequence of that Apache Tomcat 6 was configured into the virtual host. After all the server features were configured and the OpenIG core installation was implemented, the proxy server was determined to support SAML protocol. Additionally, a custom Java Filter feature was developed to discriminate Requests and Responses. The custom Filter was inserted into the OpenIG core. In this implementation "$HOME" is the home directory location of the user running the web container.

5.1 VirtualBox and CentOS

The first task of the implementation was to install the virtualization software. The newest version of VirtualBox (4.2.18) was configured into the host computer for this project. Later in the project the previous version of VirtualBox (4.2.16) was installed to remove the existing version due to a technical issue that Snapshots corrupted the installed virtual machine. A new empty virtual machine was created in the VirtualBox to run the defined operating system that was CentOS 6.4. Image Guide for installation can be found in Appendix 1. After these steps the virtual machine appeared in VirtualBox Manager. There are five different virtual machines installed in the VirtualBox in the Figure 13.

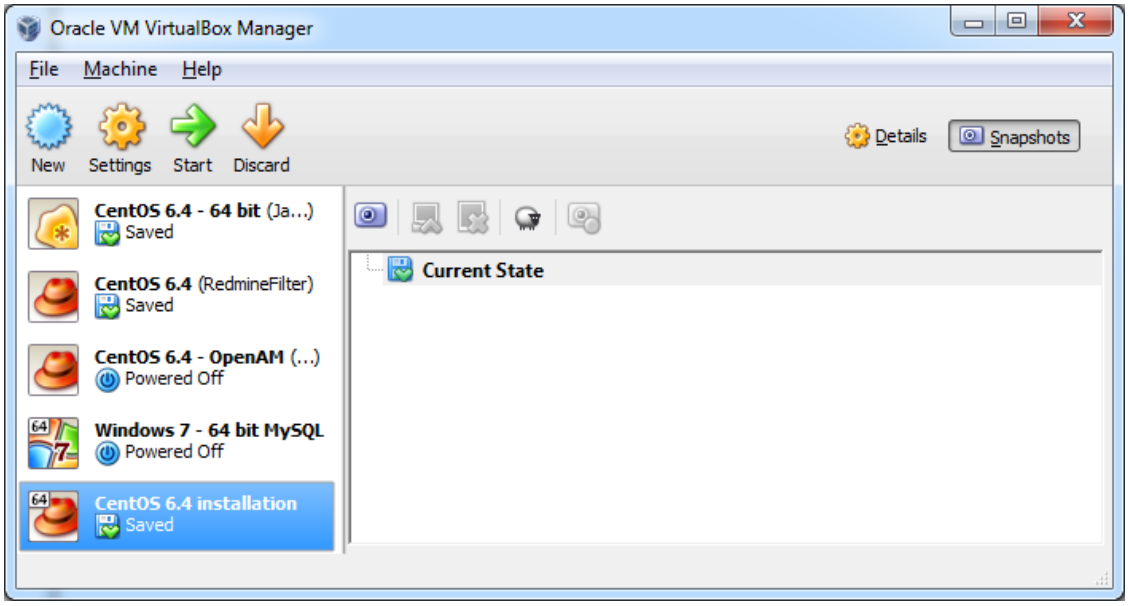


Figure 13 VirtualBox Manager View

VirtualBox also includes a Snapshot function. Snapshot captures a particular state of a virtual machine and user can revert back to that state if something critical occurs. This project was implemented step by step and Snapshots were taken constantly when something new was configured correctly in the virtual machine. The snapshots that were taken during the implementation are presented in Appendix 2.

It is necessary to receive enough memory and hard drive space so that the system operates fluently in the virtual environment. Inspect the system requirements before the installation. Additionally, the VirtuBox tool Guest Additions was configured to facilitate the usage. The installation is started from a boot manager that locates under the **Devices** menu of a target virtual machine (Appendix 3). The Guest Additions has several functions that make the usage of virtualization more flexible such as mouse pointer integration and better video support (VirtualBox 2013).

The target operating system installation was started by selecting **Start** and choosing the correct **Start-up disk** (Figure 14). In this project the CentOS 6.4 operating system was installed into the virtual machine using 64-bit computing that had to be defined previously while creating the empty virtual machine. CentOS was installed with a basic video driver and Desktop software.

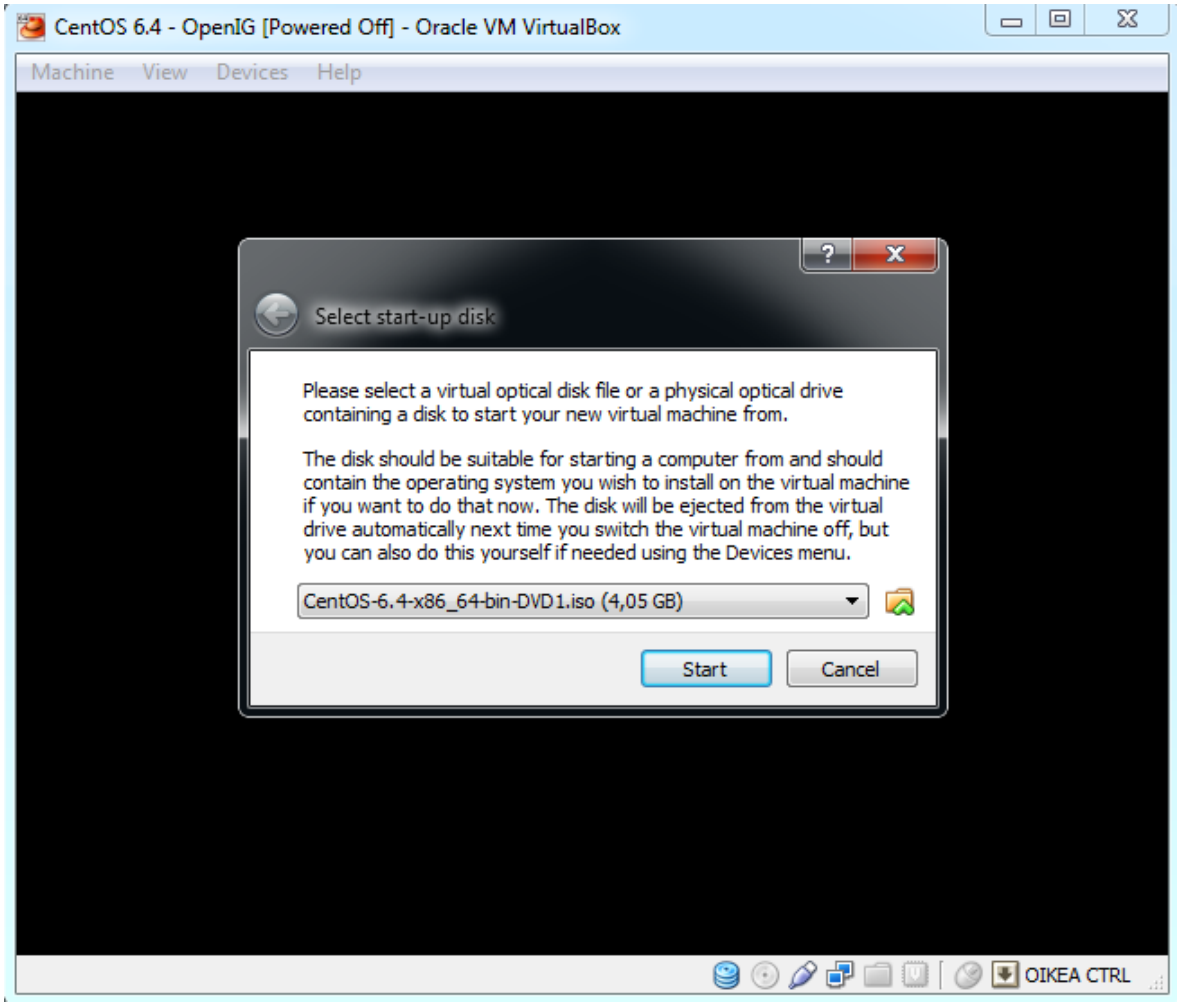


Figure 14 Select start-up disk VirtualBox

In the virtual environment the CentsOS required a network configuring to enable the network. The Ethernet Controller file was determined and the network service was restarted and the file configuration is presented in the Appendix 4. Network Time Protocol (NTP) was used to synchronize the clock automatically.

5.2 Tomcat and Java

OpenIG requires the web container and the correct version of Java to operate accurately. OpenIG is only able to work with Java SE Development Kit 6u21 or later versions of Java 6 environment (OpenIG 2013). The newest version of Java 6 environment (during the time Java SE Development Kit 6u45) was downloaded and configured to the virtual environment. The Java Development Kit (JDK) required registered Oracle account.

OpenIG supports various deployment containers from different developers (OpenIG 2013) and therefore Apache Tomcat (version 6.037) was configured to operate as the application server. Apache Tomcat is an open source servlet container and web servlet, which executes Java Servlets and Java Server Pages technologies. It is developed by Apache Software Foundation. (Apache Tomcat 2013.)

Apache Tomcat 6 and Java 6 environment were installed under /opt directory and the server was defined to execute the Java. Last task in the web container environment was to configure Security Socket Layer (SSL) protocol for managing secure communication. The Java security extension Java Cryptography Extension (JCE) files were downloaded and inserted into the security path $JAVA_HOME/jre/lib/security. JCE is a Java application program interface (API) that provides a framework for the security features implementation in Java 6 and Java 7 (Oracle JCE 2010).

Symbolic links (symlinks) were created to Tomcat and Java directories to facilitate the updating in the future. The name of the symbolic link directory persists even if the data in the target folder is configured. It reduces the need of deleting or changing file names or data. A symbolic link file or directory contains exactly the same data as its original target file. The target file remains even if the symbolic link is deleted. (Microsoft Symbolic Links 2013.)

5.3 OpenIG.war

The core content of OpenIG is archived into ".war" file. The OpenIG "Federation.war" file (full name - gateway-2.2.0-SNAPSHOT.war) was downloaded and deployed to the web application directory $HOME/opt/tomcat/webapps/. The web container instantly recognized the ".war" file and unloaded its core content to the root context directory. The trunk of the unmodified Federation.war is represented in the Appendix 5. Without any configurations OpenIG seeks config.json file in the $HOME/.ForgeRock/OpenIG/ directory. The path does not exist and it must be created by the user.

5.4 Config.json

In this OpenIG implementation the proxy server was first configured to work with the target application using the "Login Which Requires a Hidden Value From the Login Page" – config.json template. The Hidden Value is a required credential that is extracted in the exchange (OpenIG 2013). The final JSON-encoded configuration file contained more JSON Objects and had more functionality. The last version of working config.json is presented in the Appendix 6. Only the IP addresses, Federation and Redmine attributes have been modified to the generic form for the security reasons.

ConsoleLogSink writes the log to the error stream where the level (TRACE) determines the logs that are displayed. DispatchServlet sends requests to mapped servlets and filters based on the extra path of request information. Pattern is the existing path to match against the incoming requests. Object is required value that is the name of servlet filter or HTTP servlet name. FederationServlet implements the IDP initiated SSO with OpenIG. HandlerServlet translates a servlet request and sends it to a handler. BaseURI is the destination of the target application and Handler is next the heap object.

DispatcherHandler dispatches the array of handlers. Condition operates with the expression configuration parameter values and it ensures that the heap object handler dispatches to the correct destination. SPInitiatedSSORedirect composes a static response in HTTP exchange. Status code 302 performs a redirection and status reason is an optional value. Location determines the header fields that are set in the response.

Login Chain dispatches an exchange to an ordered list of filters and the OutGoingChain handler in the end. RedmineUserFilter filters a request of the exchange by extracting regular expression patterns from a message entity. HiddenValueExtract extracts authenticity token in the exchange. The token is found from the login page of Redmine and varies on each login. LoginRequestFilter2 creates a new request and replaces all the previous requests in the exchange. It uses "GET" method for capturing the resource from the Redmine login page.

SwitchFilter changes the exchange direction to other handler (LoginRequestHandler). Condition 200 is the standard response for successful HTTP requests. LoginRequestHandler is another chain that operates with the different filters. LoginRequestFilter implements request's "POST" method to set the login details in exchange. It includes the specified form parameter that consists of login page values of Redmine. This form parameter requires the username, password, the authenticity token and back_url that is the domain name of the target application. "Login" and "utf-8" are additional attributes to clarify the message in the request.

OutGoingChain is last Chain in the heap, which dispatches the exchange to ClientHandler. Additionally, it calls the CaptureFilter and writes outgoing traffic log (/tmp/outgoing-log) to verify that message is going through the OutGoingChain. CaptureFilter captures responses and requests and writes HTTP traffic log to /tmp/gateway.log destination.

LocationRewriter rewrites Location headers on responses and ensures that the user goes through the OpenIG proxy server even if the SSL protocol is active. ClientHandler submits exchange requests to the Redmine server that is the target remote server. All the incoming requests in the servlet heap object are dispatched to DispatchServlet (ServletObject).

5.5 Federation Service

The Federation service was built up by modifying the OpenIG Federation specific XML files and config.json. This OpenIG Federation IDP Initiated Service was configured by using Fedlet configuration files that were copied to the Gateway directory. The OpenIG redirects the exchange to the Federation Service once is has detected the Redmine's login page. The Federation Service redirects back to the target application after the Federation Services has validated the SAML assertions with the Trusteq Connect.

The Federation Service JSON (FederationServlet) template was copied from OpenIG tutorial and modified to associate with the Trusteq Connect. OpenIG Federation.war-file contained the necessary data files for federated implementation but the attributes

were configured to accomplish a correct JSON functionality. The following JSON configuration receives a SAML assertion from Trusteq Connect and then logs the user into Redmine using the determined credentials. Modifications must also be configured to the XML files and the metadata have to be copied to remote servers to achieve a working Federation Service.

```
28      {
29          "name": "FederationServlet",
30          "type": "org.forgerock.openig.saml.FederationServlet",
31          "config": {
32              "assertionMapping": {
33                  "userName":"userName attribute",
34                  "mail":"mail attribute",
35                  "firstname":"firstname attribute",
36                  "lastname":"lastname attribute"
37              },
38              "subjectMapping":"subjectName",
39          "sessionIndexMapping":"sessionIndex",
40              "redirectURI":"https://localhost:8443/redmine/login",
41              "logoutURI":"https://localhost:8443/redmine/logout",
42          "assertionConsumerEndpoint":"fedletapplication",
43              "SPinitiatedSSOEndpoint":"SPInitiatedSSO",
44              "singleLogoutEndpoint":"fedletSlo"
45           }
46      },
```

Figure 15 FederationServlet – config.json

The following FederationServlet above (Figure 15) is the final version of the config.json configuration used in the OpenIG implementation. Only the assertionMapping, redirectURI and logoutURI attributes required modifying from the originally copied template.

Type is the Classname path of the FederationServlet which is located in the OpenIG flat data files. The assertionMapping defines attributes that are dispatched in the incoming assertion. "Username", "mail", "firstname" and "lastname" were the target attributes for this implementation. SubjectMapping value in the assertion is set in the session. SessionIndexMapping defines the Trusteq Connect's "sessionIndex" for the user in the assertion. RedirectURI is the login page of the target application which is detected by a Form-Filter. The Form-Filter retrieves the user data from the session and executes the logging. LogoutURI is the logout point but it had no functionality in this

project. AssertionConsumerEndpoint value “fedletapplication” is configured to match with the Fedlet. “SPInitiatedSSO” and “fedletSlo” are determined in the metadata.

Fedlet endpoint was already specified to URI https://host.domain:port/saml and therefore all the Federation Service XML files were transferred to $HOME/.ForgeRock/SAML/ directory. The Federation files are listed in the Figure 16 below.

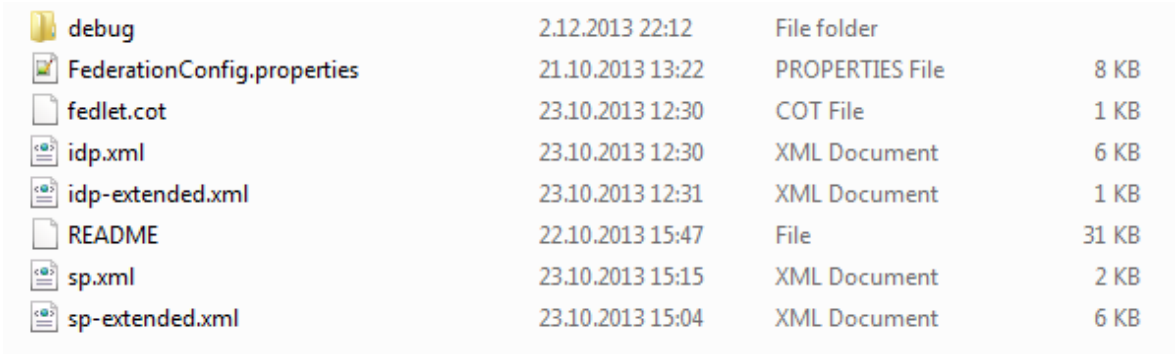


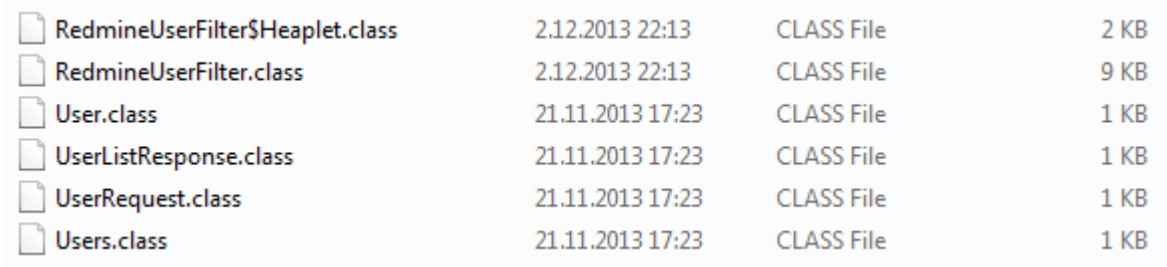
Figure 16 $HOME/.ForgeRock/SAML/

OpenIG Federation XML files associate with IDP, SP and the OpenIG and have the following functionality. Federation logging details are written to libSAML file that locates in the debug-directory. FederationConfig.properties includes advanced features of the openFed library. Fedlet.cot creates The Circle of Trust between the Identity Provider and OpenIG - original file name gateway.cot. Idp.xml is the Federation Service XML and generated by the Identity Provider. Idp-extended.xml contains the standard extensions for the Identity Provider. Sp,xml includes the standard metadata for the OpenIG Federation Service and sp-extended.xml contains the metadata extensions for sp-xml.

Fedlet.war was downloaded from OpenAM installation zip-file and the metadata (idp.xml) was exported from the Identity Provider (Trusteq Connect). Only the FederationConfig.properties and Idp.xml did not require any modification. Other files had to be configured to match against the IDP and SP details. Idp-extended.xml was copied to Identity Provider server and Service Provider metadata files were transferred to Service Provider server after the files were modified correctly.

5.6 RedminUserFilter

The project implementation required more functionality than OpenIG provided in advance and for that reason "RedmineUserFilter" was developed to execute several different XML associated operations. RedmimeUserFilter is a JSON based custom filter that was developed using Java Service Provider Interface (SPI). Java code files were written using Eclipse IDE that includes tools for developing Java EE. Completed Java code files (.java) were transformed to Java compiled class files (.class) and transferred to the correct web container location. RedmineuserFilter.class calls user contained Java classes listed in the Figure 18 below.



RedmineUserFilter$Heaplet.class	2.12.2013 22:13	CLASS File	2 KB
RedmineUserFilter.class	2.12.2013 22:13	CLASS File	9 KB
User.class	21.11.2013 17:23	CLASS File	1 KB
UserListResponse.class	21.11.2013 17:23	CLASS File	1 KB
UserRequest.class	21.11.2013 17:23	CLASS File	1 KB
Users.class	21.11.2013 17:23	CLASS File	1 KB

Figure 17 $TOMCAT_HOME/webapps/ROOT/WEB-INF/classes/com/trusteq/openigfilter

RedmineUserFilter JSON operates in the identical way with the FederationServlet. It calls for the RedmineUserFilter.class to receive attributes that are listed in the specific object configuration (config). Attribute functions are determined in the Java class. The working config.json class (RedmineUserFilter) is displayed in the Figure 19.

```
98      {
99      "name": "RedmineUserFilter",
100     "type": "com.trusteq.openigfilter.RedmineUserFilter",
101     "config": {
102         "redmineuri": "http://redmineURI-IP/redmine",
103         "messageType": "request",
104         "login": "login attribute",
105         "firstname": "firstname attribute",
106         "lastname": "lastname attribute",
107         "mail":"mail attribute",
108         "apikey":"apikey attribute",
109         "redminePassword": "password attribute",
110         "target": "target attribute"
111         }
112     },
```

Figure 18 RedmineUserFilter - config.json

RedmineUserFilter.java (Appendix 7) was developed utilizing OpenIG examples and already existing Filters. OpenIG required setting up Java 6 EE environment in Eclipse to provide working Java classes. First task was to implement a custom Filter and after that begin to increase its functionality step by step. The design a working Filter required previous knowledge of the target application and Identity Provider. The logging was written to web container log (catalina.out) for debugging.

RedmineUserFilter parses the upcoming SAML request and modifies the login credentials to authenticate the connecting user into the target application. The Filter was configured to work with the Redmine REST API protocol to associate with users.json resources which requires an authentication into web application using valid credentials. The REST API function was enabled by a Redmine administrator in the admin panel and authentication was implemented by using a Redmine API key that was hardcoded in the OpenIG config.json file.

RedmineUserFilter.class (Appendix 7) first verifies that the connecting user already exists in the Redmine. If the user exists, the filter updates the basic user information credentials such as firstname, lastname and mail. If the user does not exists, the filter creates a new basic user based-on the username using the user contained information. In both cases, the Filter generates a new password. The user must have a unique mail address, otherwise the new user is not created. The filter generates a new 45 character long password on every login. The user is never able to see the password, because it is written in assertion metadata.

5.7 Testing and debugging

Testing was implemented in the CentOS virtual environment using various operating system tools and command lines. Typically the configurations were straight edited to OpenIG or the new piece of data was transferred into its server and tested in the virtual environment. The cURL command-line tool was used to execute CRUD operations into the Redmine.

Logging was written to the OpenIG file /tmp/gateway.log and Apache Tomcat 6 file opt/tomcat/logs/catalina.out. HTTP traffic also does logging to Identity Provider and Service Provider web container. Most of the failures referred client errors and required information of HTTP response status codes.

Figure 19 Firefox SSO Tracer

Firefox SSO Tracer (Figure 17) is Mozilla Firefox browser Add-on. It filters HTML requests, displays SAML messages and flags authentication Messages (SSO Tracer 2013). SSO Tracer debugging tool was used to discover SAML messages in the OpenIG SAML responses and requests. SAML requires that the time is synchronized precisely with a host and associated remote servers.

6 Conclusion

The OpenIG project was successfully implemented within the prescribed time limit. The working OpenIG platform was configured into Trusteq's identity service and the project report was published in their Wiki-site. The OpenIG most probably will be used in the future with other applications and projects that the Trusteq enterprise will execute. This thesis established that the product can be configured to implement a working Federation service secured with SAML protocol. The original OpenIG files included most of the required functionality used in this implementation and it is intensely customizable using Java 6 development kits. The coding does not differ much from other products running Java 6.

The challenging part of the project was that there was so slightly information about the OpenIG. It was difficult to define the time schedules and prophecy the amount of time that each part would require. One of the reasons was that I did not have previous experience working with the identity and access management field and many of the concepts and habits were unfamiliar. Working and cases become more acquainted when the project progressed.

The OpenIG is pretty unknown product even if it has been released already in 2011. Its support is primarily taken care of the OpenIG community and its developers, which is very common to all the open source software. It was very hard to find any information about OpenIG outside the official OpenIG website and I never joined the ForgeRock's mailing list. As well as this OpenIG project, I reckon that other open source software also require knowledge of the similar kind of products and years of experience of the industry.

Even though Federation has become more and more popular during the last few years, there are hardly similar products as OpenIG in the market. This may occur considering that only larger companies use Federation and Federation solutions are quite expensive. Smaller corporations always need to consider pros and cons carefully before purchasing anything. Generally the Federation is implemented by another large technology company.

Nearly all the great technology corporations such as Microsoft, Google, Facebook and Yahoo are using Federation for their services. It facilitates the usage and reduces identity credentials which makes the authentication more secured and facilitates maintenance. While working in the information security Trusteq, I noticed the importance of planning secured network at the beginning. Many of the big companies have implemented their identity and access management carelessly which can cause major problems in the future.

Most of the concepts were unfamiliar to me before I started to work with the project. During the process I faced deadlocks and required help from my technical advisors but each time the solution was found and the project was able to go forward. Most issues proceed from incorrect configurations that were determined in the OpenIG files. Occasionally errors appeared due to the server side problems. Once I had to start the project all over again, because the VirtualBox version was corrupted and virtual machines did not work. . Fortunately that happened in the beginning of the project and making a fresh start was the fastest way to go forward. The logs were in an active use for debugging and I learned a lot of about the tactics that were used to locate errors in the software and server side.

I have gained a much better understanding of the Federation and its concepts. The study has also taught extremely lot of the information technology industry and working in the medium-sized technology enterprise. Trusteq also provided me a new vision to work when I worked as a part of project group Connect that consisted of technology experts and developers. The study gave me considerably better acquirements to work in the industry and the benefits are obviously visible. If I would start the same kind of project now, I certainly would get into it much faster. Now I would know how to begin to work more efficient way, even though it would be a different company.

References

Apache Tomcat. 2013

http://tomcat.apache.org/

Accessed: 10 December 2013

AssureBridge. WS-Federation. Publication date: June 28, 2012.

http://www.assurebridge.com/sso/saml-vs-ws-federation-for-single-sign-on/

Accessed 19 December 2013

CentOS. Wiki. October 15 2013

http://wiki.centos.org/

Accessed: 21 November 2013

empowerID. Identity Management. 2013

http://www.empowerid.com/learningcenter/technologies/service-identity-providers

Accessed: 7 November 2013

empowerID. Federation Services. 2013

http://www.empowerid.com/learningcenter/technologies/service-identity-providers

Accessed: 19 December 2013

IBM. Reverse Proxy. 2010

http://pic.dhe.ibm.com/infocenter/sametime/v8r5/index.jsp?topic=%2Fcom.ibm.help.sametime.v851.doc%2Fconfig%2Fst_adm_port_rvprxy_overview_c.html

Accessed: 19 December 2013

IBM. Transport Layer Security. June 6 2012

www.ibm.com/developerworks/webservices/library/ws-ssl-security/

Accessed: 19 December 2013

Forgerock website. OpenAM. 2013

http://forgerock.com/products/open-identity-stack/openam/

Accessed: 28 October 2013

Forgerock website. Who We Are. 2013
http://forgerock.com/who-we-are/ Accessed: 28 October 2013
J. Lauhia, L. Tielinen. 2009. Federoijan PikaopasTM. Trusteq OY Espoo.
Accessed: 11 November 2013

Microsoft. Symbolic Links. 16 November 2013
http://msdn.microsoft.com/en-us/library/windows/desktop/aa365680%28v=vs.85%29.aspx
Accessed: 11 December 2013

M S.Bhiogade. June 2002. Secure Sockets Layer.
http://www.proceedings.informingscience.org/IS2002Proceedings/papers/Bhiog058Secur.pdf
Accessed: 19 December 2013

OpenIG website. Publication date: October 14, 2013.
http://openig.forgerock.org/doc/gateway-guide/
Accessed: 25 October 2013

Oracle Java Cryptographic Service Provider. 2010.
http://docs.oracle.com/javase/1.4.2/docs/guide/security/CryptoSpec.html#ProviderArch Accessed: 5 November 2013

Oracle VM VirtualBox website. 2013
https://www.virtualbox.org/manual/ch01.html#virtintro
Accessed 19 November 2013

Ping Identity website. SAML. 2013
https://www.pingidentity.com/resource-center/SAML-Tutorials-and-Resources.cfm
Accessed: 29 October 2013

Redmine. Rest API. 2013
http://www.redmine.org/projects/redmine/wiki/Rest_api
Accessed: 16 December 2013

Redmine website. 2013
http://www.redmine.org
Accessed: 10 November 2013

Shibboleth. 2013.
http://shibboleth.net/about/
Accessed: 13 December 2013

SSO Tracer. Firefox. 2013
https://addons.mozilla.org/En-us/firefox/addon/sso-tracer/?src=cb-dl-created
Accessed: 3 January 2013

T. Bray, Jean Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau. 16 August 2006.
Extensible Markup Language (XML).
http://www.w3pdf.com/W3cSpec/XML/2/REC-xml11-20060816.pdf
Accessed: 17 December 2013

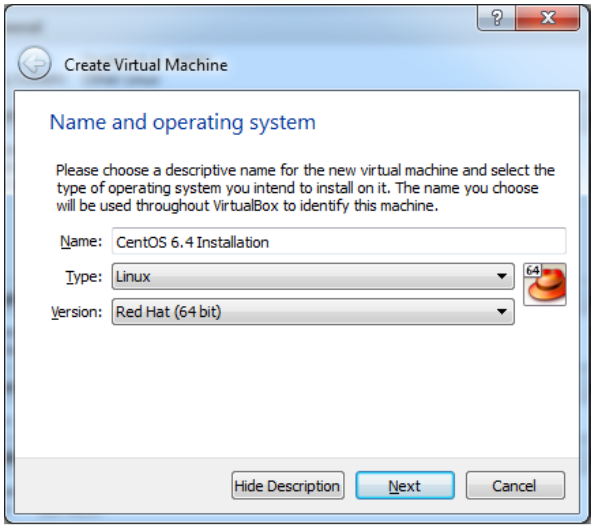
Trac website. Redmine. 2012
http://trac.edgewall.org/wiki/RedMine
Accessed: 17 November 2013

Trusteq Connect Service Description. Publication date: July 19, 2013.
Accessed: 31 October 2013

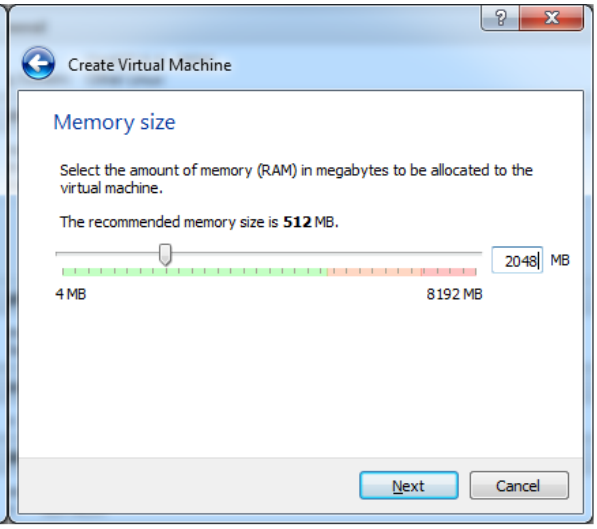
World Wide Web Consortium. XML. 29 October 2013
http://www.w3.org/XML/
Accessed: 16 December 201

Appendices

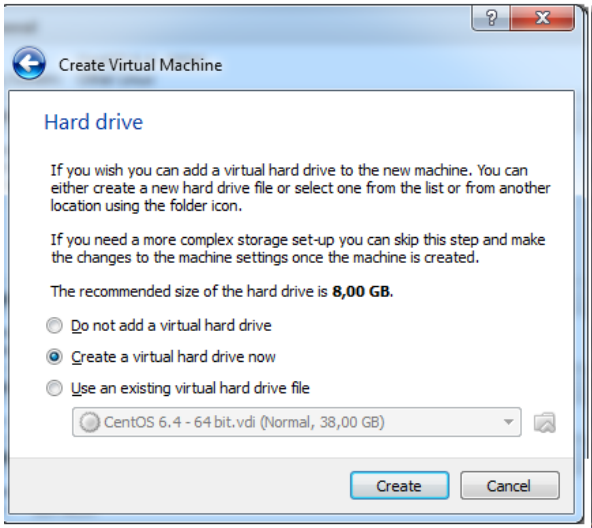
7.1 Create a new empty virtual machine



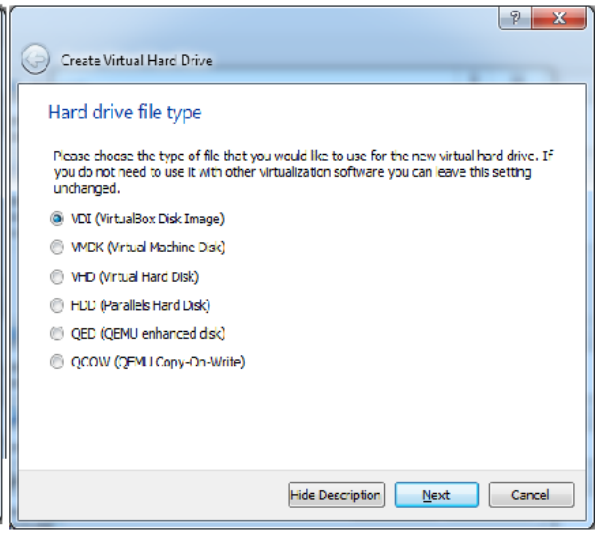
1. Determine name and operating system



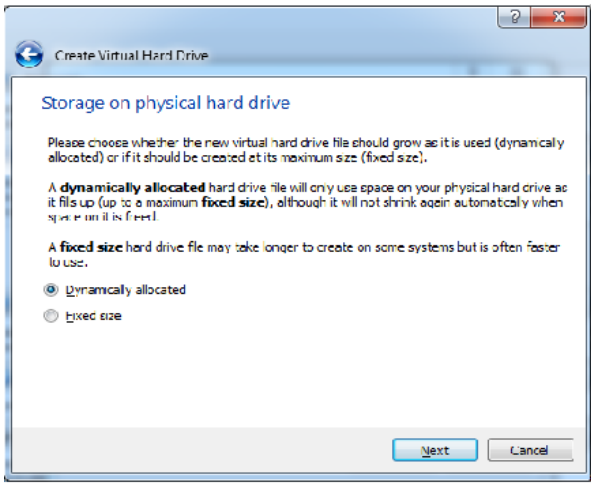
2. Choose Memory Size



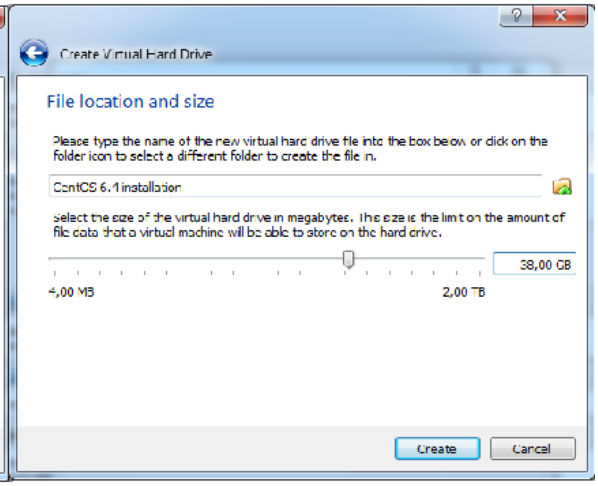
3. Determine Hard Drive



4. Choose Hard Drive file type

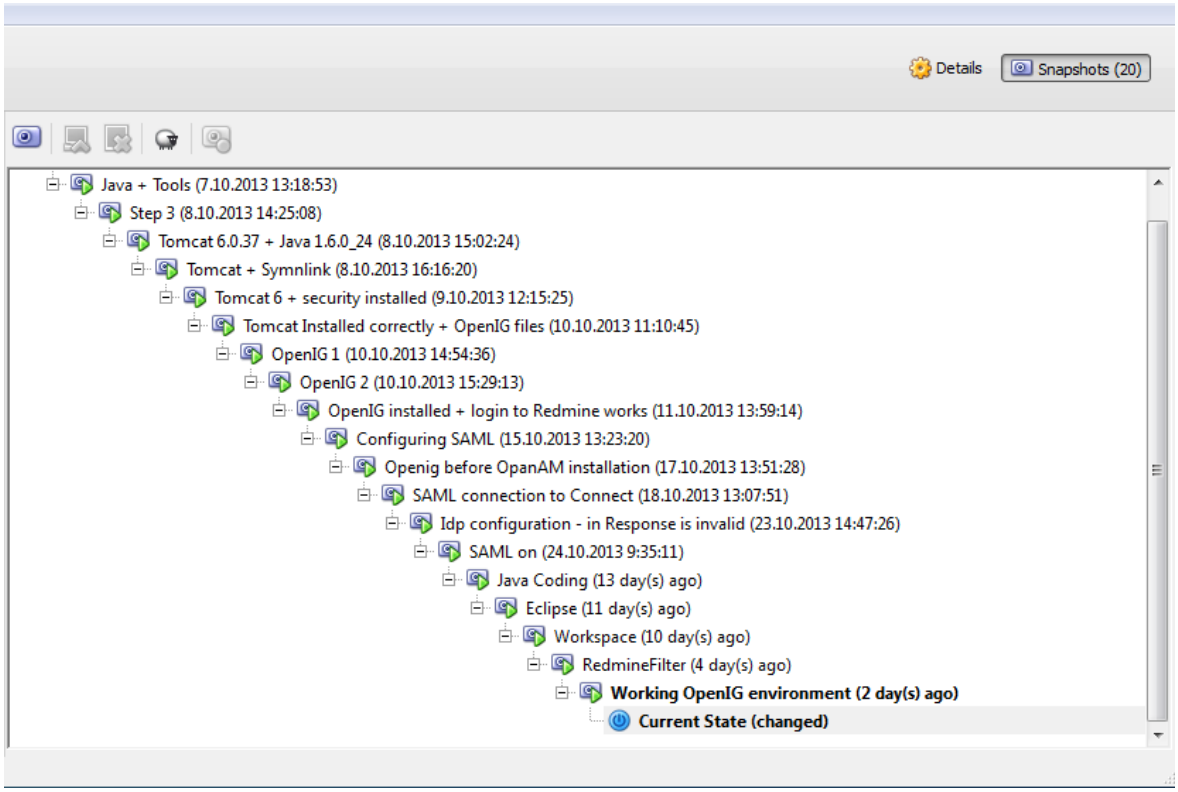


5. Set the storage on physical hard rive

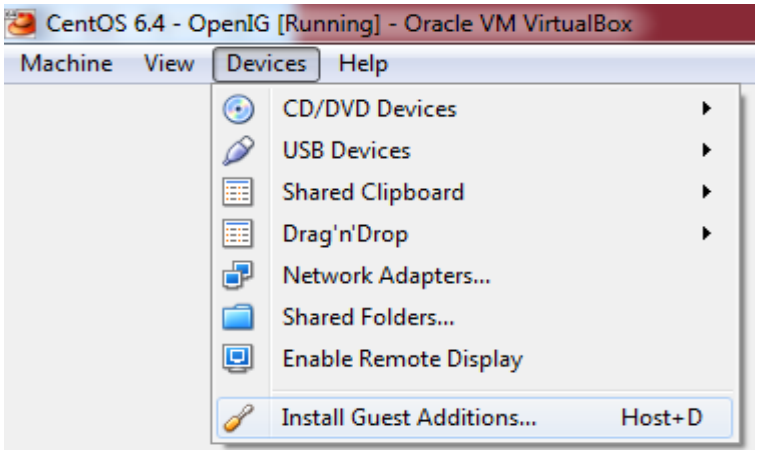


6. Determine the file location and size and press **Create**

7.2 VirtualBox Snapshots



7.3 VirtualBox Install Guest Additions

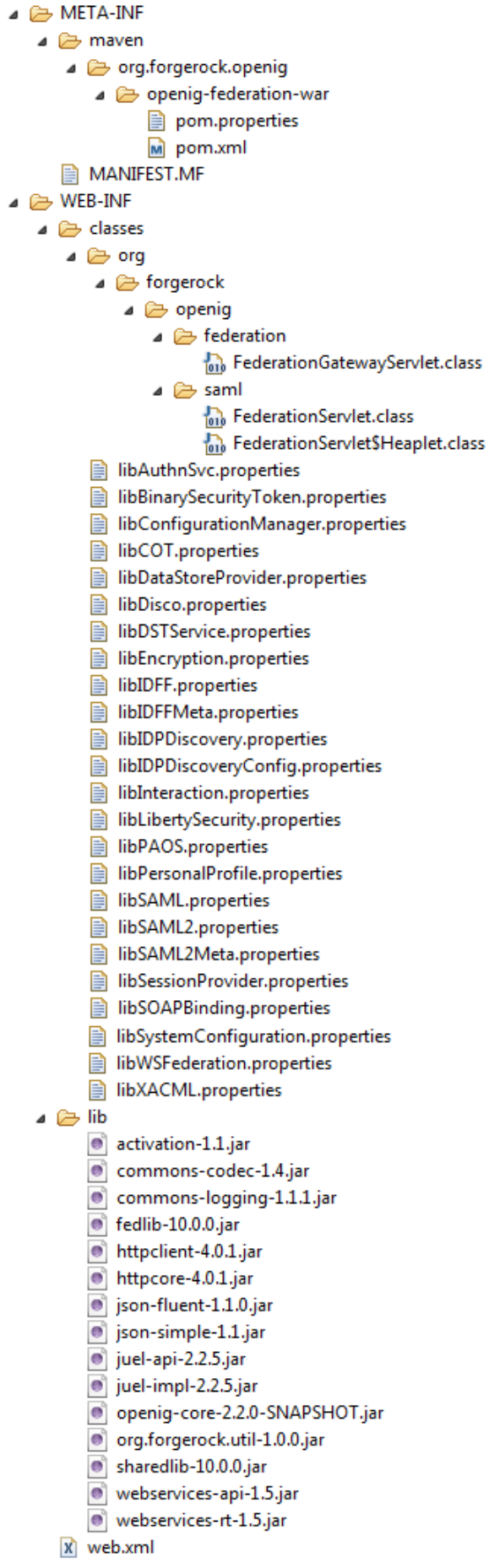


7.4 The Ethernet Controller path

```
  GNU nano 2.0.9   File: /etc/sysconfig/network-scripts/ifcfg-eth0

DEVICE=eth0
HWADDR=08:00:27:81:51:83
TYPE=Ethernet
UUID=27c11697-85f6-4136-a1da-6c36f6b40be9
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=dhcp
```

7.5 OpenIG Federation.war - trunk



7.6 OpenIG config.json

```
{
  "heap": {
    "objects": [
     {
     {
        "name": "LogSink",
        "comment": "Default sink for logging information.",
        "type": "ConsoleLogSink",
        "config": {
            "level": "TRACE"
        }
     },
     {
         "name": "DispatchServlet",
         "type": "DispatchServlet",
         "config": {
             "bindings": [
                 {
                     "pattern":"^/redmine/saml",
                     "object":"FederationServlet"
                 },
                 {
                     "pattern":"",
                     "object":"HandlerServlet"
                 }
             ]
         }
     },
     {
         "name": "FederationServlet",
         "type": "org.forgerock.openig.saml.FederationServlet",
         "config": {
             "assertionMapping": {
                 "userName":"userName attribute",
                 "mail":"mail attribute",
                 "firstname":"firstname attribute",
                 "lastname":"lastname attribute"
             },
             "subjectMapping":"subjectName",
         "sessionIndexMapping":"sessionIndex",
             "redirectURI":"https://localhost:8443/redmine/login",
             "logoutURI":"https://localhost:8443/redmine/logout",
         "assertionConsumerEndpoint":"fedletapplication",
             "SPinitiatedSSOEndpoint":"SPInitiatedSSO",
             "singleLogoutEndpoint":"fedletSlo"
          }
     },
```

```
    {
        "name": "HandlerServlet",
        "type": "HandlerServlet",
        "config": {
            "handler": "DispatchHandler",
            "baseURI":"http://baseURI-IP/redmine"
        }
    },
    {
        "name": "DispatchHandler",
        "type": "DispatchHandler",
        "config": {
            "bindings": [

                {
                    "condition": "${exchange.request.uri.path != '/redmine/login'}",
                    "handler": "ClientHandler"
                },
                {
                    "condition": "${empty exchange.session.username attribute}",
                    "handler": "SPInitiatedSSORedirectHandler"
                },
                {
                    "condition": "${exchange.request.uri.path == '/redmine/login'}",
                    "handler": "LoginChain"
                },
                {
                    "handler": "OutgoingChain"
                }

            ]
        }
    },
        "name": "SPInitiatedSSORedirectHandler",
            "type": "StaticResponseHandler",
            "config": {
                "status": 302,
                "reason": "Found",
                "headers": {
                    "Location": ["https://localhost/redmine/saml/SPInitiatedSSO"]
                }
            }
    },
{
    "name": "LoginChain",
    "type": "Chain",
    "config": {
        "filters": ["SwitchFilter","RedmineUserFilter","HiddenValueExtract","LoginRequestFilter2"],
            "handler": "OutgoingChain"
    }
},
```

```
{
"name": "RedmineUserFilter",
"type": "com.trusteg.openigfilter.RedmineUserFilter",
"config": {
    "redmineuri": "http://redmineURI-IP/redmine",
    "messageType": "request",
    "login": "login attribute",
    "firstname": "firstname attribute",
    "lastname": "lastname attribute",
    "mail":"mail attribute",
    "apikey":"apikey attribute",
    "redminePassword": "password attribute",
    "target": "target attribute"
    }
},
```

```
  {
  "name": "HiddenValueExtract",
  "type": "EntityExtractFilter",
  "config": {
      "messageType": "response",
      "target": "${exchange.hiddenValue}",
      "bindings":  [
          {
          "key": "value",
          "pattern": "authenticity_token\"\s.*value=\"(.*)\"",
          "template": "$1"
          }
      ]
  }
  },
```

```
  {
      "name": "LoginRequestFilter2",
      "type": "StaticRequestFilter",
      "config": {
          "method": "GET",
          "uri": "http://localhost-URI/redmine/login"
              }
  },
```

```
  {
   "name": "SwitchFilter",
   "type": "SwitchFilter",
   "config": {
       "onResponse": [
           {
               "condition": "${exchange.response.status == 200}",
               "handler": "LoginRequestHandler"
           }
       ]
   }
  },
```

```
148     {
149         "name": "LoginRequestHandler",
150         "type": "Chain",
151         "config": {
152                 "filters": ["LocationRewriter","LoginRequestFilter"],
153                 "handler": "OutgoingChain"
154     }
155     },
```

```
156     {
157         "name": "LoginRequestFilter",
158         "type": "StaticRequestFilter",
159         "config": {
160             "method": "POST",
161             "uri": "http://localhost/redmine/login",
162         "form": {
163                 "username": ["GET-username"],
164                 "password": ["GET-password"],
165                 "authenticity_token": ["${exchange.hiddenValue.value}"],
166                 "back_url": ["http://localhost/redmine/"],
167                         "login": ["login attribute"],
168                         "utf8": ["utf8 attribute"]
169             }
170         }
171     },
```

```
172     {
173         "name": "OutgoingChain",
174         "type": "Chain",
175         "config": {
176             "filters": ["CaptureFilter"],
177             "handler": "ClientHandler",
178             "file": "/tmp/outgoing.log"
179         }
180     },
```

```
181     {
182         "name": "CaptureFilter",
183         "type": "CaptureFilter",
184         "config": {
185             "captureEntity": true,
186             "file": "/tmp/gateway.log"
187         }
188     },
```

```
189      {
190       "name": "LocationRewriter",
191       "type": "RedirectFilter",
192       "config": {
193           "baseURI": "https://localhost/redmine/"
194        }
195      },
```

```
196         {
197             "name": "ClientHandler",
198             "type": "ClientHandler",
199             "config": {
200                 "filters": ["LocationRewriter"]
201         }
202         }
203         ]
204     },
205     "servletObject": "DispatchServlet"
206 }
```

7.7 RedmineUserFilter.java (Secret)