Identity Federation
with SAML 2.0

Josua Trösch


Supervising tutors
Prof. Dr. Andreas Steffen, HSR
Roman Pletka, AdNovum
Alex Suzuki, AdNovum

University of Applied Sciences
HSR Hochschule für Technik Rapperswil

Industry partner
AdNovum Informatik AG, Zürich
Abstract

Nowadays, IT systems are not isolated entities anymore but heavily rely on information exchanged with other systems. Usually, an authorization verification needs to be accomplished before getting access to another system’s data. Nevis is a security infrastructure for the protection of sensitive data, services and applications. It is developed and maintained by AdNovum Informatik AG. Like many other vendors of security software, AdNovum implemented a proprietary protocol and format to provide security information over network connections. As more and more Internet services started to collaborate, the need for an open standard to describe and exchange security information has emerged. The SAML 2.0 standard serves this need by providing a suite of protocols and message formats to describe and exchange security information.

In a first phase of this thesis, an overview of the current support of the SAML 2.0 standard by some software products was compiled. The two application containers BEA WebLogic and IBM WebSphere include broad support for the SAML 2.0 standard. JBoss currently brings support for the SAML 1.0 standard. The open Java and C++ library OpenSAML 2.0 provides Software developers with functionality for validating and handling SAML 2.0 XML documents and supports various profiles and bindings.

In a second phase, selected features of the OASIS SAML 2.0 standard were implemented into AdNovum’s Nevis framework. The Nevis framework was enabled to handle SAML 2.0 Authentication Requests. Based on the Request a SAML 2.0 conforming Response is composed and sent back to the requesting service. If access is granted, this Response contains a SAML 2.0 Assertion with the requested security information to log in a user in the remote system. Performance and profile conformance were tested against a BEA WebLogic Server instance.

The implementation has been merged into the Nevis Framework and can be used in future projects.
# Contents

1 **Introduction** iv  
   1.1 Structure of this document iv  

2 **Management Summary** vii  
   2.1 Initial situation vii  
      2.1.1 Motivation vii  
      2.1.2 Goals viii  
   2.2 Proceeding ix  
      2.2.1 Related work ix  
      2.2.2 Implementation ix  
      2.2.3 Involved Mentors ix  
   2.3 Results ix  
   2.4 Outlook x  

3 **SAML 2.0** 1  
   3.1 Introduction to SAML 1  
      3.1.1 Basic SAML Concepts 2  
   3.2 SAML Web Browser SSO Profile 3  

4 **Nevis middleware** 4  
   4.1 AdNovum Nevis Middleware 4  
      4.1.1 nevisBox 4  
      4.1.2 nevisProxy 6  
      4.1.3 nevisAuth 6  

5 **Software support for SAML 2.0** 9  
   5.1 Liberty Interoperable™Event 9  
   5.2 Application Containers 9  
      5.2.1 BEA Weblogic Server 9  
      5.2.2 IBM Websphere 10  
      5.2.3 JBoss 10  
   5.3 OpenSAML 10  
      5.3.1 OpenSAMLDemo 11
CONTENTS

6 Results 12
   6.1 Implementation ................................................. 12
      6.1.1 Conceptual functionality .................................. 12
      6.1.2 SAMLRequestProcessor .................................... 14
      6.1.3 Provider .................................................... 15
   6.2 Tests ............................................................. 16
      6.2.1 Load tests .................................................. 16
      6.2.2 Unit tests .................................................. 19

7 Conclusions 20

8 Experience Report 21

Appendix A: WebLogic Setup and Configuration of Nevis 22

Appendix B: Project management 32

Appendix C: Glossary 40

References 40
Chapter 1

Introduction

1.1 Structure of this document

This documentation is composed of the following chapters:

**Conceptual Formulation** The conceptual formulation that defines the goals of this diploma thesis.

**Management Summary** Is directed to management members and explains the motivation for the project, the proceeding and an outlook to possible future work on the project.

**SAML 2.0** An introduction to the Security Assertion Markup Language

**Nevis middleware** An introduction to AdNovum’s Nevis middleware and it’s components relevant for this project.

**Software support for SAML 2.0** Overview of current SAML 2.0 support in selected application servers and inspection of the open developer library, OpenSAML

**Results** Describes the implemented features and the tests that were conducted.

**Experience Report** In this chapter the diploma thesis is reviewed from a personal perception.

**Conclusions** Conclusions and a review of the results.

**Appendix A: WebLogic Setup and Configuration of Nevis** An installation guide for BEA WebLogic Server TechPreview 10.3 and configuration samples for Nevis can be found there.

**Appendix B: Project management** A project plan and meeting protocols.

**Appendix C: Glossary** Specific terms and acronyms explained.
Identity Federation mit SAML

Student: Josua Trösch
Betreuer: Prof. Dr. Andreas Steffen
Industriepartner: AdNovum

Einführung

One of the main security questions addressed in Single Sign-On (SSO) environments is the usage and management of identity propagation between identity providers and consumers. The propagation of identities is usually done by so called security tokens or assertions which convey signed authentication and authorization information. A service provider which can be a J2EE application then acts as a consumer of such assertions. The identity propagation process derives directly from the trust model defined for the given environment.

The Security Assertion Markup Language (SAML) is an XML-based open standard which has been standardized by the OASIS Security Services Technical Committee for exchanging authentication and authorization information between identity providers and consumers in a security domain. Identity Federation addresses the cross-domain interactions that allow for the exchange of identity information and hence implement a specific trust-relationship model.

In the scope of this diploma thesis the student will first study the SAML 2.0 standard in order to be able to draw a detailed overview of the current functional range in the Nevis middleware framework from AdNovum. Based on this insight, the students will design and implement a SAML-based mechanism that allows to piggy-back arbitrary attributes to be propagated in an assertion - a fundamental functionality in identity federation.

The thesis will include the composition of a prototype environment acting as a proof-of-concept for identity federation. This prototype will implement the handling of direct SAML requests and hence act as a SAML responder. The solution will be based on the Nevis middleware framework and a current off-the-shelf J2EE container supporting SAML.
Aufgabenstellung

- Compile an overview of the current support of the SAML 2.0 standard, based on the following J2EE application containers (BEA WebLogic, IBM Websphere, JBoss, etc.).
- Extend the Nevis Middleware to support part of the SAML 2.0 standard, i.e. implement the Web Browser SSO Profile, thus enabling identity federation for web services.

Links

- Einführung in SAML 2.0
  http://en.wikipedia.org/wiki/SAML_2.0
- SAML 2.0 Spezifikation
  http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip
- Uebersicht über die AdNovum Middleware Komponenten
- Nevis Whitepaper

Rapperswil, 3. März 2008

Prof. Dr. Andreas Steffen
Chapter 2

Management Summary

2.1 Initial situation

2.1.1 Motivation

Nowadays, IT systems are not isolated entities anymore, but heavily rely on network connections to other systems. Usually, an authorization verification needs to be accomplished before getting access to another system’s data. Providing a username and password or a digital certificate are ways of proving one’s identity. As more and more internet services collaborate, Single Sign On (SSO) solutions have become popular. This relieves users from logging in separately for every service they use. Instead, their identity is propagated from one service to the other. Many vendors implemented their proprietary SSO solution. In heterogeneous environments, the need for a standard for the exchange of security data has emerged.

Identity Federation is the concept of building a trust relationship between two entities and enables an automated access control. The SAML 2.0 standard from the OASIS consortium includes extensive specifications for Identity Federation and Single Sign On scenarios. SAML 2.0 entities that have a Trust-Relationship can exchange arbitrary security-relevant data about a principal, including access rights, entitlement information and arbitrary attributes.

Figure 2.1 shows a basic Identity Federation Use Case. The user has an authentication context in company.com and when he tries to access the protected resource at the partner site, his identity information can be retrieved from company.com automatically. This is possible because the two security domains have a business agreement and the user’s identity is federated. To simplify the use of a web service provided by different security domains, SAML defines a Single Sign On protocol for Web Browser users. This standard allows products from different vendors to propagate logged in users to other security domains, enabling a more seamless user experience.

Nevis is used in different companies to ensure security and confidentiality of
sensitive data, services and applications. For the communication between its
components this middleware enables SSO, using a proprietary mechanism. For the
propagation of identities to other vendors’ products, a translation mechanism had to
be implemented. AdNovum’s Ninja project, for example, enables Java application
containers to communicate with the Nevis security infrastructure. The goal of
implementing SAML functionality in Nevis is to enable integration with arbitrary
other systems.

2.1.2 Goals

There were two main goals set for this diploma thesis:

- Compile an overview of the current support of the SAML 2.0 standard, based
  on these J2EE application containers: BEA WebLogic, IBM Websphere,
  JBoss.

- Implement a prototype that handles direct SAML requests and hence acts as
  a SAML responder. The solution has to be based on the Nevis middleware
  framework.

The SAML specification supports various modes (called SAML Bindings) for the
exchange of SAML Requests and Responses. The prototype shall support the
HTTP-POST Binding. Optional features are the support of the other two possible
Bindings, HTTP Redirect and Artifact.
2.2 Proceeding

2.2.1 Related work

As I was on new ground with the Identity Federation subject, SAML and the Nevis Framework, the first part of the thesis included a lot of reading. An overview of the SAML 2.0 standard was drafted. As the specification is very extensive, I soon concentrated on the Single Sign On scenario.

In a first phase of this thesis, an overview of the current support of the SAML 2.0 standard by selected software products was compiled. One of the inspected Containers, the BEA WebLogic server, was set up as a reference implementation. It was chosen because a Technical Preview version was freely available and installable with a simple setup.

In a second phase, the Nevis functionality regarding SAML 2.0 was extended. After getting an overview of the whole Nevis framework and the nevisBox, I focused on nevisProxy and nevisAuth. It became clear that the main functionality would be placed within nevisAuth, as this component handles the authentication procedures. In a next step, the functional requirements for nevisAuth were extracted.

2.2.2 Implementation

Before I started with the implementation, I set up a Nevis environment. A VMWare-Image of a nevisBox could be used. During the implementation process, the server components that were adapted could be replaced in the VMWare with the new ones. Some problems were encountered setting up the WebLogic server as a SAML Service Provider. The configuration described in the official manual did not result in the targeted situation: A WebLogic server that would act as a SAML Requester. Help was searched in an Internet-Forum and by requesting official support from BEA. A reply in the Forum described a solution that solved the problem. With this environment the added functionality could be regularly tested. Code was added to nevisAuth whilst maintaining backward compatibility.

2.2.3 Involved Mentors

For questions regarding the project’s requirements and AdNovum’s software, Alex Suzuki and Roman Pletka from AdNovum were the primary contacts. In meetings held every week, the progress of the work and the remaining open items were discussed with Prof. Dr. Andreas Steffen, Roman Pletka and Alex Suzuki.

2.3 Results

An overview of the functional range of popular server software regarding SAML 2.0 could be drawn. The leading product is IBM’s WebSphere, followed by BEA
WebLogic server. JBoss includes SAML 1.0 functionality, but does not currently support SAML 2.0. This gives an indication of the importance and acceptance of the standard for the industry. Support for SAML is growing and the standard will be used as it addresses many companies’ needs.

A prototype that serves as a SAML-2.0-compliant Identity Provider has been implemented in the AdNovum Nevis Middleware. This was achieved by extending the existing implementation. The prototype was successfully tested against a WebLogic server instance. Also, a load test was conducted. It showed that the bottleneck is the nevisAuth server.

Unfortunately, the spare time reserved in the project plan was consumed by server configuration problems. Therefore, only a basic scenario could be implemented in the prototype. Nevertheless, the defined goals of the diploma thesis were achieved. The quality of the written code was assured using common software engineering methods.

2.4 Outlook

In the future, Nevis could be extended to support the other Bindings (HTTP Redirect and Artifact). It could also be enabled to act as a Service Provider, i.e. generate Requests and consume Assertions received in response. The moderate load test results could be further investigated.
Chapter 3

SAML 2.0

This chapter describes the SAML 2.0 standard (Security Assertion Markup Language).

3.1 Introduction to SAML

The SAML standard has been developed by the OASIS consortium (Organization for the Advancement of Structured Information Standards), an organization devoted to the development, convergence and adoption of open standards for information technology. SAML is an open standard for exchanging security information. The following aspects of the standard are of interest for this thesis:

**Single Sign On** Collaborating organizations want to allow their users a seamless transfer from one to the other web service. If users are authenticated at a web service, they shouldn’t have to manually reauthenticate on the other organizations service. Existing solutions typically relied on browser cookies for exchanging identity information. Browser cookies created from one security domain, for security reasons, can’t be read from another one. Therefore, proprietary mechanisms to pass the authentication data between security domains have been used. This solution works fine for a single enterprise but as soon as different organisations, using different products collaborate, this approach gets impractical. SAML provides a standard protocol and message format to exchange this security information.

**Federated Identity** To exchange the authentication in a collaborative application environment, there is another issue than the message exchange format; the different security domains must find a way to ensure they are referring to the same principal in this exchange. Usually users are known in one security domain by a certain ID which might be different in another security domain. When partners have agreed on a common alias the referred user is said to have a Federated Identity. This alias can then be used in SAML messages. SAML 2.0 also specifies a protocol to give the user control over the creation of a federation of his identity, greatly reducing costs for a company’s security administration.
3.1.1 Basic SAML Concepts

SAML consists of building block components that can be combined to support a number of use cases. Basically the components allow transfer of identity, authentication, attribute, and authorization information between trusted security domains.

SAML Assertions carry statements about a principal. They are usually received from an Asserting Party in Response to a Request. In some cases unsolicited Assertions arrive. In either case there are several methods to verify if the information contained in the Assertion is trusted before establishing a security context. Different ways to exchange messages are defined by SAML Protocols. The structure and content of these protocol messages are defined in XML schemas.

SAML Bindings define the use of lower level communication or messaging protocols (e.g. HTTP, SOAP) to transport SAML protocol messages.

A SAML Profile describes a particular business use case. It typically defines which protocols and bindings can be combined and which data an Assertion must include. The Web Browser SSO Profile was implemented in this thesis and is described in the next section.

Figure 3.1 illustrates the relationship between the basic SAML components.
3.2 SAML Web Browser SSO Profile

The Web Browser Single Sign On Profile provides a variety of options, describing how to use SAML messages and bindings to achieve Single Sign On over different authentication domains.

Figure 3.2 illustrates a basic SAML 2.0 SSO scenario which is initiated by the SP. The process starts with the user requesting a resource. As this user does not yet have a security context at this domain, a request triggered to an Identity Provider. This request is not sent directly but via the users Web Browser. The request is packed in a POST Form that, with Java Script enabled, is automatically sent to the Identity Provider. If the Identity Provider has not yet done so, it authenticates the user. A SAML Assertion is generated. If the HTTP-POST binding is used, the Response containing the generated Assertion, is packed in a POST Form that is sent back to the user. This Form submits the Response to the Service Provider. The Service Provider processes the Assertion and decides to grant or deny access to the resource.

The process differs if the SAML 2.0 Artifact Binding is used. Instead of the actual request, a so called SAML Artifact is sent to the Identity Provider. This Artifact is a reference to the request. Upon receiving the Artifact, the Identity Provider will contact the Service Provider directly to obtain the actual Request. The same principle applies for the Assertion when using the Artifact Binding. Instead of the actual Assertion, a reference to it is sent to the Service Provider via the users Web Browser.
Chapter 4

Nevis middleware

4.1 AdNovum Nevis Middleware

In this section an introduction into AdNovum’s Nevis Middleware is given. Nevis is used in many different companies to securely attach Web-Applications to the internet. By design, it is pluggable and built on Open Source Software, thus supporting common industry-standards. It prevents unauthorized access to critical data, services and applications from outside and inside a company’s network. Within one domain this middleware enables Single Sign On (SSO). This simplifies users workflow and enhances their work experience by automatically logging them into different applications they use throughout their working processes. The main components are:

- **nevisProxy** Application Firewall and encryption
- **nevisAuth** Authentication, Authorization and Session management
- **nevisRum** Resource and User management
- **nevisAdmin** Administration of components via Webinterface
- **nevisBox** Container for the above components

Figure 4.1 shows the architecture of the Nevis Middleware.

4.1.1 nevisBox

The nevisBox is an appliance setup of the above mentioned nevis components. It acts as a Container for one or more Nevis components and can be administered via a web console. A single-, two- or three-box (running on one or multiple machines) setup is possible.

A nevisBox can be administered via a web console provided by nevisAdmin. NevisAdmin communicates with the other services using JMX (Java Management Extensions). For environments were firewall rules do not allow connections to the
outside, the nevisAdmin configuration communication is reverse. Instead of sending configuration updates, it makes MBeans, containing the updated configuration data, available to registered components. In certain time intervals every component checks if there is a configuration update (i.e. an MBean) available on nevisAdmin and if any, collects it.

All manageable components are discovered automatically by the nevisOsAgent, a JMX Server. NevisOsAgent allows to perform queries on the Operating System and to set up and control Nevis services. Figure 4.2 depicts the configuration flow.

Figure 4.1: The Nevis Middleware Architecture (Source: AdNovum)

Figure 4.2: Communication flow for Nevis administration and configuration
4.1.2 nevisProxy

NevisProxy is a reverse proxy and, combined with nevisAuth, acts as an application firewall. It’s main tasks are:

**Session handling**  Associating multiple requests with a client

**Authentication & authorization**  Ensuring authentication (delegated to nevisAuth) and enforcing access restrictions

**Reverse proxying**  Forwarding HTTP(S) requests to content providers and enabling content caching to reduce the application load for static content.

It’s software architecture, based on the J2EE servlet API (see [Sm]), separates the functional components from the carrier server’s environment and provides an environment that enables efficiently writing customer-specific plugins. Filters and servlets are configured in the nevisProxy’s `web.xml` configuration file with the appropriate initialization parameters. They are then mapped to URL-patterns. With a mapping entry in `web.xml` a filter or servlet is associated with a specific URL-pattern (e.g. `/applOne/*`). Different filter/servlet settings can therefore just be accomplished for distinct URL-patterns. If a request for an application (e.g. `http://www.company.com/applOne`) arrives at nevisProxy every mapped filter for this URL-pattern is applied. Filters are applied in the order they are defined in `web.xml`. Every filter can use the request data (e.g. for authentication) and make changes to it before sending it to the next filter. At the end of every filter chain one servlet has to be mapped. A Servlet can retrieve a response from a remote service (e.g. an application server). Figure 4.3 shows the filter and servlet architecture on a basic example.

NevisProxy is implemented in C++. To communicate with nevisAuth, CORBA is used. For using Java objects, a library simulating a Java environment called *Base Component* was written by AdNovum. This allows to use the nevisAuth Java-Objects in nevisProxy.

4.1.3 nevisAuth

NevisAuth is responsible for authentication. The following list explains the important authentication events:

**authenticate**  This is the initial login procedure. It associates a user and his/her credentials with a global session or a set of security roles.

**step up**  If an already logged in user does not have the necessary privileges for accessing a resource, this method is called.

**step down**  User privileges that were previously granted with a step up are removed from the login session.
logout Terminates the global session. All session members are notified to remove resources associated with that global session or user.

Incoming authentication requests from nevisProxy are dispatched by the Authentication engine. It provides a session for stateful multistep authentication. On the session, the current user’s authentication data (e.g. the HTTP request) can be accessed. The AuthEngine contains one or more AuthStates. An AuthState is a logical step in the authentication procedure and can return a result. Every AuthState is configured with its parameters in the nevisAuth configuration file (esauth4.xml). By combining multiple AuthStates, complex authentication procedures can be constructed which are easily manageable.

Figure 4.4 shows a configuration sample containing one domain and several authentication states. For the initial authentication the LoginUidPwd state is called. This state must implement the authenticate() method. If additional privileges are needed the stepup() method from the class configured in the LoginOneTimePw state is called.
To identify a correctly authenticated user within the Nevis Middleware, nevisAuth issues a token called SecToken mainly containing the following authentication data:

- a user id
- the strength of the authentication (e.g. weak, strong)
- a reference to the authentication service that verified the user credentials
- a global authentication session identifier
- a creation timestamp and an absolute lifetime

The SecToken is an AdNovum proprietary format. It is signed to allow the receiver to verify the data has not been changed. This SecToken was designed when an open standard like SAML didn’t exist. To communicate with services other than nevis, SAML is used. Functionality for producing and consuming SAML 2.0 Assertions is implemented, but it doesn’t include the option to deliver Assertions in reply to SAML 2.0 Requests. The SAML 2.0 WebBrowser SSO Profile standardizes this exchange and was implemented in this diploma thesis.
Chapter 5

Software support for SAML 2.0

In this chapter the current support of the SAML 2.0 standard by some software products is examined.

5.1 Liberty Interoperable\textsuperscript{TM} Event

Liberty Alliance is a global identity consortium. Technology vendors, consumer service providers and educational and government organizations are members of it. In the Liberty Interoperable\textsuperscript{TM} Event in December 2007, the participating vendors products were tested for SAML 2.0 interoperability, as stated in a press release [All]

One of the passing products was the RSA Federated Identity Manager v4.0. The solution can be integrated with WebLogic server and IBM Websphere and supports SAML 1.1 and 2.0.

5.2 Application Containers

5.2.1 BEA Weblogic Server

The current WebLogic Server 10.0 does not support SAML 2.0 but the upcoming version WebLogic Server 10.3 supports the SAML 2.0 Web Browser SSO profile and the related standard Web Services Security (WS-Security) SAML Token profile 1.1. A TechPreview of WebLogic Server 10.3 could be downloaded and installed for testing purposes.

BEA Weblogic TechPreview 10.3

For testing purposes, a WebLogic Server was set up. After initial problems installing and configuring the WebLogic Server Tech Preview 10.3, some fundamental information about BEA WebLogic was collected and a step by step installation guide
was created (see Appendix A).

5.2.2 IBM Websphere

The IBM Websphere application server needs an extension for handling SAML messages, the Tivoli Federated Identity Manager, a component of the IBM Tivoli Software. The Tivoli solutions provide centralized authentication, policy management and access control services for Web resources, systems and hosted applications. A typical installation consists of the reverse proxy IBM Tivoli Access Manager and a WebSphere Application Server. TFIM can then be integrated with any Web application via an HTTP/HTTPS connection.

IBM Tivoli Federated Identity Manager version 6.1.1 (TFIM) provides concurrent support for SAML 1.0, 1.1 and 2.0. IBM has passed Liberty Alliance testing for SAML 2.0 interoperability with its TFIM version 6.2 in November 2007. It conforms to SAML, WS-Federation and Liberty ID-FF federation protocol standards.

5.2.3 JBoss

The JBoss application server currently supports the SAML 1.0 standard. SAML 2.0 will be implemented in the future, but there is no roadmap yet.

The JBoss Federated SSO Framework is developing features for Identity Federation, making use of the OpenSAML libraries. For current information about the project, visit: http://labs.jboss.com/jbosssso/

5.3 OpenSAML

OpenSAML is an open-source project developed by Internet2. The Java and C++ libraries, currently supporting the SAML 2.0 standard, help developers implementing SAML functionality in their software. It is possible to create objects with the individual information fields that make up a SAML message and build the correct XML representation, as well as parsing SAML XML messages back into objects. Functionality to help developers using various SAML profiles and bindings is included.

The OpenSAML Java library was inspected regarding the Web Browser SSO profile. It currently does not bring support for the entire profile, meaning that AuthnRequests and Responses can be parsed and validated but certain fields that are required by the Web Browser SSO profile pass the validation unnoticed as they are not required by the basic Request-Response protocol. Therefore a profile specific validation needs to be performed by the developer using the library.
5.3.1 OpenSAML Demo

To demonstrate OpenSAML’s capability, a small commandline utility was written. It can parse and validate SAML 2.0 AuthnRequests and Responses and generates an output, informing if the XML Files are valid.
Chapter 6

Results

6.1 Implementation

This chapter mainly describes the extensions made to the Nevis Framework. Functionality to produce SAML Assertions was included before. In this diploma thesis, support for the SAML 2.0 Web Browser SSO Profile was implemented. This profile specifies the use of SAML AuthnRequests to obtain a Response containing a SAML Assertion. Therefore, functionality to parse AuthnRequests and generate an according Assertion was integrated into NevisAuth. The implementation was realized according to the SAML 2.0 specification [Ser05], taking into account the Errata Document [Ser07].

6.1.1 Conceptual functionality

This section gives an overview on how an incoming Request is processed by the different Nevis components and an according Assertion is generated. A SAML Web Single Sign On Use Case is described in Chapter 3. It points out the requirements for an implementation of a SAML participant. In our case, the Nevis middleware is the Identity providing participant. To reliably test the implemented features a Service Provider partner instance was set up. A WebLogic Server (TechPreview 10.3) was taken as a reference implementation.

The following existing Filters and AuthStates were used for the implementation:

- NevisProxy

  IdentityCreationFilter Handles the authentication process of a user and establishes an authenticated session.

- SAMLProviderFilter After having passed the IdentityCreationFilter, this filter triggers the issuance and distribution of a SAML Response. It was used in former projects to deliver unsolicited SAML Assertions.

- NevisAuth
LoginState An arbitrary AuthState that performs an authentication of the user (e.g. UseridPasswordAuthenticateState).

For further information see the NevisProxy Reference Guide [AG07b] and the NevisAuth Reference Guide [AG07a]. A configuration example is found in Appendix A.

![Diagram of SAML Request Processing](image)

**Figure 6.1: Processing of a SAML Request**

Figure 6.1 illustrates the processing of a Request triggered by a user requesting a protected resource at a Service Provider. For simplicity the indirections via the user’s web browser are not shown in this diagram.

1. **Request arrives** An `<AuthnRequest>` is received at the NevisProxy. At first, the IdentityCreationFilter assures that an authentication context exists. If an authenticated session already exists, steps 2 and 3 are skipped.

2. **Login** If the user is not logged in yet, the configured state in NevisAuth conducts a login.

3. **Return principal** NevisAuth returns the authenticated principal to NevisProxy.

4. **Call SAMLProviderFilter** If the login succeeded, the SAMLProviderFilter is called.

5. **Process `<AuthnRequest>`** The `<AuthnRequest>` is handed over to the RequestProcessor in NevisAuth. The SAMLRequestProcessor in NevisAuth validates the `<AuthnRequest>` and extracts certain information.
6. Call Provider  The SAML-Provider state uses the information from step 5 to generate a SAML Assertion which is packed into a SAML Response.

7. Return  The Response is sent to the SAMLProviderFilter.

8. Send  The SAMLProviderFilter sends the Response to the Service Provider.

All SAML messages are XML documents. Although the OpenSAML library includes support for parsing such messages, the XMLBeans approach was used to be consistent with the existing Provider AuthState implementation. XMLBeans is an apache technology for accessing XML by binding it to Java types.

6.1.2 SAMLRequestProcessor

The SAMLRequestProcessor AuthState processes SAML AuthnRequests as specified in the WebBrowser SSO Profile (see SAML profile specification in [Ser05]) and the underlying Authentication Request Protocol (see SAML core specification in [Ser05]). As for now, it only supports the HTTP-POST binding and therefore expects the Request to be passed in a form control called $ \text{SAMLRequest}$. The received Request is base64-decoded and validity checks are performed on the resulting XML document.

Service Providers are configured via Metadata XML Files (see SAML meta specification in [Ser05]). If no Metadata Files are found, the state can not initialize and throws an Exception. The following parameters in the nevisAuth configuration file ($ \text{esauth4.xml}$) allow the configuration of the RequestProcessor AuthState:

"saml.metadata.path" (required) All files with the suffix .xml in this folder are searched for containing SP Metadata.

"saml.request.accept_unsigned" (optional) Accepts unsigned AuthnRequests if set to true or rejects them if set to false (Default)

If all checks are passed without an error, the information necessary to construct the Response and Assertion is saved in session-notes:

"saml.SP.entityID" Entity ID of the sender.

"saml.request.ID" ID of the Request.

"saml.SP.URL" URL the Response will be sent to. If a Target URL was declared in the AuthnRequest and the Request was signed it is used. If this is not the case, the target URL defined in the SP configuration is used.

"saml.request.subject" If the AuthnRequest declared a Subject the value is saved in this note.

If an error occurred while processing the AuthnRequest the session-note "saml.errors" is set with the according second level error code (e.g. "urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified").
6.1.3 Provider

The existing Provider class that generates SAML 2.0 Assertions was extended, while maintaining complete backward compatibility. It tries to extract information from the session notes. If successful, the values are set in the according Response and Assertion elements. If an error code was set in the SAMLRequestProcessor, no Assertion is generated and the Response only contains an error message. The additional state configuration options are:

"saml.Target" *(required)* This option was included before and needs a special setting. To enable the state to dynamically generate Assertions in Response to AuthnRequests it has to be set with this value: "$notes:saml.SP.URL". This instructs the Provider to extract the target from the session-note that was set by the SAMLRequestProcessor state. The SAML Response will be sent to this target.

"saml.response.sign" *(optional)* If set to *true* the Response element is signed. By default it is set to *false*.

"saml.error.secondLevelCodes" *(optional)* If a SAML-error was handed over from the AuthnRequest processing, this option specifies if the second level error code is included in the Response message. The basic error message ("urn:oasis:names:tc:SAML:2.0:status:Responder") only states that any error has occurred at the Responder side. The second level error code gives more specific information about the exact nature of the occurred error. Attackers may use this information and therefore it can be turned off.

Sample configurations for nevisAuth and nevisProxy are listed in Appendix A.
6.2 Tests

6.2.1 Load tests

For load tests the ProxySniffer tool was used. It was run on the local machine. In the web browser, the service was registered as a proxy server. Via a web interface a recording session could be started. In another browser window the login procedure was executed. All the URL’s that were called in this procedure were recorded by ProxySniffer. After the necessary adjustments in the recorded session were made, the tool was able to execute this procedure repeatedly. For every test run a variety of parameters could be configured. Tests were run with different amounts of users accessing the resources concurrently. Figure 6.2 shows the setup for the load tests.

![Load Test Setup Diagram]

The following charts show the comparison of 4 load tests where 10, 30, 70 and 100 users tried to login concurrently during 10 minutes. The connecting lines between the 4 measuring points are interpolated. To simulate real circumstances every login attempt was configured with so called "think times". At every login procedure a "user" was idle for 4 seconds before submitting the password at Nevis, simulating a real user’s time to think. Before the logout another 6 seconds were added. This "think time" is indicated by the red line in the diagram.
Chart 6.3 shows the average time for a complete login procedure.

![Average login time](image)

**Figure 6.3: Load Test - Average login time**

In chart 6.4 we see the successfully completed logins per minute. We can see that more than 70 concurrent users result in a decreasing rate. Also the number of logins starts to get unstable at 30 concurrent users as the standard deviation bars show. This standard deviation is comparatively high because a lot of the requested logins are not served immediately. After 10 minutes of testing no more requests were sent but the pending logins were finished. As samples were only saved every minute, this caused a low number of completed logins at the first minute but a high number at the end. All the logins in between completed at a constant rate.

![Completed Logins](image)

**Figure 6.4: Load Test - Completed logins per minute**
Chart 6.5 shows the failed login attempts in percent. We can see that the session failures start to increase with 70 concurrent users and accordingly the completed logins per minute (see chart 6.4) decrease.

![Login failure rate](image1)

Figure 6.5: Load Test - Login failure rate

Chart 6.6 reveals the impact of the tests on every server when 100 concurrent users tried to log in. In this test scenario, the nevisAuth machine clearly was the bottleneck.

![CPU loads of the servers in percent](image2)

Figure 6.6: Load Test - CPU loads of the servers in percent

The conducted load tests showed that the system did not perform as well as expected under stress. In former non SAML tests, the nevis framework could handle around
300 logins per minute. Unfortunately, the remaining time did not allow a further investigation of the reasons for this moderate performance. Nevertheless, the results were discussed and some possible explanations came up:

The conducted load test showed that the bottleneck is the nevisAuth server. Where the other machines had a CPU load ranging from 1% to 6%, the nevisAuth server was on its limits with a 94% CPU load. Compared to the WebLogic server, the two nevis services nevisAuth and nevisProxy were running on slower machines. The signing of every response and assertion was a cpu consuming task. The used standard sun algorithm could have been exchanged with a more performant one.

The testing environment was not set up as it would be in a production environment. It could not be ruled out that other developers were using these machines for other tests as well at that time.

6.2.2 Unit tests

To assure correct functionality JUnit tests were written, testing the whole SAML procedure in NevisAuth. A fake Authentication engine is set up and fed with Requests. Depending on the testcase, the resulting Assertion or Exception is inspected.
Chapter 7

Conclusions

An overview of the current SAML 2.0 support by selected software products was compiled. IBM’s WebSphere brings the broadest support for the standard, followed by BEA WebLogic server. Both products make Identity Federation and Single Sign On possible. JBoss does not currently support SAML 2.0, but is planning to implement it. This gives an indication of the importance and acceptance of the standard for the industry. Identity Federation enables more seamless cross-domain business interactions. Support for SAML is growing and the standard will be used as it addresses many companies’ needs.

The existing SAML implementation in the Nevis framework was used and extended. The result is a prototype that serves as a SAML-2.0-compliant Identity Provider. The prototype was successfully tested against a WebLogic server instance. Unfortunately, the spare time reserved in the project plan was consumed by server configuration problems. These problems consumed a lot of time because I had to wait for answers in the forum. Therefore, the optional features (i.e additional Bindings) could not be implemented in the prototype. Nevertheless, this implementation can be used as a basis for further SAML development. Additional functionality can easily be integrated into the solution. Current projects at AdNovum indicate further application of the SAML 2.0 standard.

The conducted load test showed that the bottleneck is the nevisAuth server. Where the other machines had a CPU load ranging from 1% to 6%, the nevisAuth server was on its limits with a 94% CPU load. Possible explanations and solutions for this behaviour were discussed. First, nevisAuth had to sign every Assertion and Response with an RSA signature. Sun’s standard RSA algorithm was used for this procedure. There are more efficient algorithms available or the signing could even be delegated to a Hardware module. Second, the testing environment was not set up as it would be in a production environment as other developers use these machines for their tests as well.
Chapter 8

Experience Report

I chose this diploma thesis because I was interested in the security topic and because it was a great opportunity to gain insight into the working methods of a software developing company. The first part of the thesis included a lot of reading; the SAML 2.0 standard specification, the Nevis middleware documentation and the existing SAML code in the Nevis middleware. As all the computers in AdNovum run on Linux I had to get familiar with this Operating System. The weekly held meetings where the progress was discussed and necessary actions were decided helped keeping me on track.

A standard nevisBox was set up in a reasonable time, as an existing VMWare-Image could be used. The problems started when I tried to set up a WebLogic server. As it was a technical preview version the documentation for the new SAML 2.0 functionality was very basic. This problem consumed a lot of time, which was rare in this project lasting eight weeks. After trying different things, I tried to find help in a user forum and from the official BEA support site. The solution for the problems was posted in the forum after a few days. In retrospect I spent to much time trying to solve the problem myself. In a future project I would contact support earlier and meanwhile work up other items.

Setting up a performance testing environment in week 7 – again – was not as easy as expected, as every server had to be configured individually. An expired WebLogic testing license was just one of the encountered problems. All in all I underestimated the effort necessary to configure these server systems.

The time I spent in AdNovum was very instructional. It was the first time I could gain insight into a software development process in a greater scale and had an opportunity to study and implement an interesting security standard. Thanks to the support of my mentors and the employees at AdNovum I could eventually achieve a satisfactory result in my diploma thesis.
Appendix A:
WebLogic Setup and Configuration of Nevis

Installing and Configuring WebLogic

General WebLogic directory structure

**autodeploy** directory for quickly deploying applications on a development server. Any applications or modules that are placed in this directory are automatically deployed, when the server instance is running in development mode. In production mode, those are not available.

**bin** contains scripts for starting and stopping the Administration Server and optionally Managed Servers.

**config** contains the following:

- config.xml (domain-specific configuration file), specifying the name of the domain and the configuration parameter settings for each server instance, cluster, resource, and service in the domain.

- Subdirectories containing the configuration for various system modules (e.g. deployments, jdbc, lib, security). They contain configuration files that are referenced in the central config.xml file.

**lib** contains the domain library. Any jar files placed in this directory are added dynamically to the end of the server classpath at server start-up.

**security** contains common security files for all servers in the domain.

**servers** contains a subdirectory for each server in the domain. In these server subdirectories, in turn, are server-instance specific files and folders (e.g. bin, cache, logs, security).

*(More information can be found in the WebLogic Server 10.0 Documentation)*

**Installation**

The following is a step-by-step installation and configuration guide for the WebLogic 10.3 tech preview.

1. Execute installer: `./server103tp_linux32.bin`
2. Full installation
3. Insert the following lines in `server/bin/startWLS.sh`
WL_HOME="/local/bea/wlserver_10.3tp"
JAVA_HOME="/local/bea/jrockit_160_02"
FEATURES_DIR="/local/bea/modules/features"
SERVER_NAME="testServer"
WLS_USER="weblogic"
WLS_PW="weblogic"

4. Create a directory in `<Weblogic-home>` for test-domain (e.g. "testDomain")

5. Change to this directory (this causes the domain specific files to be created in this folder)

6. Start the (previously edited!) startup-script: `./../server/bin/startWLS.sh`

7. Confirm when asked: "Would you like the server to create a default configuration and boot? (y/n)"

The Server is now up and ready. Applications can be deployed in the administration console web interface (see Administration)

For starting the server instance from now on, the generated script in the domain-directory (`./<domain-directory>/bin/startWebLogic.sh`) is to be used.

**Administration**

This section guides you through administration console (web interface) configuration screens. Only the necessary modifications for a minimal configuration are listed. For more detailed information about the Weblogic Server TechPreview 10.3, see [ Sysb].

On a default installation, the administration console is found on: `http://localhost:7001/console/`

**Deploying the demobank web-application**

In the test-domain directory, a "webapps"-directory was created and the demobank-application was copied in there. In the Administration Console the demobank-application was added with the following security settings: “Custom Roles and Policies: Use only roles and policies that are defined in the Administration Console.”

The easiest way to deploy an application would be to copy it in `<domain-directory>/autodeploy/`. Whenever an .ear-File or exploded application-directory structure is copied in there it is automatically deployed. However, for autodeployed applications no extended configuration options like security constraints can be defined!

**Configuring SAML 2.0 functionality**

Configure Security: Security Realms → <select realm>

Create SAML2 Idp & Authenticator
Authentication providers are responsible for verifying the user’s identity. The simplest authentication mechanism would be a username and password. Multiple authenticators are allowed. The JAAS control flags (OPTIONAL, SUFFICIENT, REQUIRED, and REQUISITE) control the order in which authenticators are accessed and the behavior of the authentication process with multiple authenticators.

Providers → Authentication → New
Create two Providers, one of type SAML2Identity Asserter and another one of type SAML Authenticator

<activate the changes and restart the server>

**Configure SAML2Identity Asserter**

Identity assertion providers are similar to authenticators but the authentication of a user is based upon a perimeter token (e.g., a SAML token). Instead of locally, this user was authenticated indirectly by a different system. Only identities from trusted (i.e., configured) Identity Provider partners are accepted. Multiple identity asserters are allowed, one for every token type.

1. **Create IdP Partner**

   <select SAML Identity Assertion Provider> → New (Web Single Sign on IdP Partner)

   The web console asks for an XML configuration file. Listing 8.1 shows a minimal Configuration file.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  entityID="https://172.16.254.130/SAML">
  <md:IDPSSODescriptor WantAuthnRequestsSigned="false"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <md:SingleSignOnService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
      Location="https://172.16.254.130/saml-idp"/>
  </md:IDPSSODescriptor>
  <md:Organization>
    <md:OrganizationName xml:lang="en">AdNovum</md:OrganizationName>
    <md:OrganizationDisplayName xml:lang="en">MyNevisBox</md:OrganizationDisplayName>
  </md:Organization>
</md:EntityDescriptor>
```

Listing 8.1: Minimal IdP-metadata configuration File
2. Settings for IdP Partner

Security realms → <select realm> → Providers → Authentication → <choose the SAML2Identity Asserter> → Management → <choose Partner> → General

Enabled: true (check)

Redirect URI's: Defines a list of URIs that trigger the SAML assertion sequence when someone tries to access them. Relative path (e.g. "http://localhost:7001/demobank/login.jsp")

Configure WebLogic as a SAML2 Service Provider:

Environment → Servers → <choose server instance> → Federation Services → SAML 2.0 Service Provider:

Enabled: check (true)

Preferred Binding: POST

Default URL: [http://localhost:7001/demobank](http://localhost:7001/demobank)

*Note: For using WebLogic as a SAML Authority (Identity Provider), a WebLogic Credential Mapping Provider has to be configured.*

Configure general SAML2 settings:

Environment → Servers → <choose server instance> → Federation Services → SAML 2.0 General → SAML 2.0 General:

Following fields need to be filled out: Contact Person Given Name, Contact Person SurName, Contact Person Company, Organization URL

**Published Site URL: has to end with saml2 (e.g. [http://localhost:7001/saml2/](http://localhost:7001/saml2/)).**

Entity ID: Usually the Organization URL is used for this field (e.g. [http://localhost:7001/](http://localhost:7001/))

**Logging:**

Although some logging can be activated in the web console, enabling it there did not work for the SAML 2.0 specific logs. To get the complete SAML log messages, input the following in the server startup script (`./<domain-directory>/bin/startWebLogic.sh`):

```
JAVA_OPTIONS="$\{JAVA_OPTIONS\}
-Dweblogic.debug.DebugSecuritySAMLAtn=true
-Dweblogic.debug.DebugSecuritySAMLlib=true
```

25
–Dweblogic.debug.DebugSecuritySAML2Service=true
–Dweblogic.debug.DebugSecuritySAML2CredMap=true
–Dweblogic.debug.DebugSecuritySAML2Atn=true”
Sample configurations

The following two listings show the relevant configuration in nevisProxy and nevisAuth to enable the SAML 2.0 Web Browser SSO functionality.

nevisProxy

This configures nevisProxy to process SAML 2.0 Assertions at http://www.company.com/saml-idp. NevisAuth needs to provide the SAMLTEST-Domain.

Listing 8.2: web.xml

```xml
<!-- ********** filters ********** -->
<!-- ********** filters ********** -->

<filter>
  <filter-name>SAML AuthenticationFilter</filter-name>
  <filter-class>:
  ::ch::nevis::isiweb4::filter::auth::IdentityCreationFilter
</filter-class>

<init-param>
  <param-name>AuthenticationServlet</param-name>
  <param-value>EsAuth4Connector</param-value>
</init-param>

<init-param>
  <param-name>LoginRendererServlet</param-name>
  <param-value>BuiltinLoginRenderer</param-value>
</init-param>

<init-param>
  <param-name>InactiveInterval</param-name>
  <param-value>14400</param-value>
</init-param>

<init-param>
  <param-name>ReauthInterval</param-name>
  <param-value>99999</param-value>
</init-param>

<init-param>
  <param-name>Realm</param-name>
  <param-value>SAMLTEST</param-value>
</init-param>

<init-param>
  <param-name>EntryPointID</param-name>
  <param-value>fulvia.adnovum.ch</param-value>
</init-param>
</filter>
```
<filter>
  <filter-name>SAMLFilterBrowserPOST</filter-name>
  <filter-class>
    ch::nevis::isiweb4::filter::saml::SAMLProviderFilter
  </filter-class>

  <init-param>
    <param-name>AuthenticationServlet</param-name> <!-- any servlet can be specified here as it is never reached -->
    <param-value>EsAuth4Connector</param-value>
  </init-param>

  <init-param>
    <param-name>BindingType</param-name>
    <param-value>BrowserPOST</param-value>
  </init-param>

  <init-param>
    <param-name>ParameterName</param-name>
    <param-value>SAMLResponse</param-value>
  </init-param>

  <init-param>
    <param-name>AudienceRestriction</param-name>
  </init-param>

  <init-param>
    <param-name>Target</param-name>
    <param-value>
      http://192.168.8.185:7001/saml2/sp/acs/post
    </param-value>
  </init-param>
</filter>

<!— filter mapping —>
This configures the SAMLTEST-Domain. It will authenticate the user with the TestUseridPassword state. Any other login state could be configured instead. The stepup method is called by the SAMLProviderFilter from nevisProxy. Here the SAMLRequestProcessor needs to be configured to process the Authentication Request.

Listing 8.3: esauth4.xml

```xml
<Domain name="SAMLTEST" default="true" reauthInterval="0" inactiveInterval="1800">
  <Entry method="authenticate" state="TestUseridPassword" />
  <Entry method="stepup" state="SAMLRequestProcessor" />
</Domain>

<AuthState name="SAMLRequestProcessor"
  class="ch.nevis.esauth.auth.states.saml.SAMLRequestProcessor">
  <ResultCond name="default" next="SAMLRequestProcessor" />
  <ResultCond name="ok" next="SAMLProvider" />
  <Response value="AUTH_DONE">
    <Gui name="ErrorDialog" label="errordialog.label">
      <GuiElem name="lasterror" type="error"
        label="${notes.lasterrorinfo}" value="${notes.lasterror}" />
    </Gui>
  </Response>

  <property name="saml.metadata.path" value="/var/opt/nevisauth/default/conf/serviceProviders/" />
  <property name="saml.request.accept_unsigned" value="true" />
  <property name="keystoreref" value="DefaultKeyStore" />
```
<property name="keyobjectref" value="DefaultTrust" />
</AuthState>

<AuthState name="SAMLProvider"
class="ch.nevis.esauth.auth.states.saml.Provider" final="false">
  <Response value="AUTH_DONE">
    <Gui name="ErrorDialog" label="errordialog.label">
      <GuiElem name="lasterror" type="error"
        label="\{notes.lasterrorinfo\}" value="\{notes.lasterror\}" />
    </Gui>
    <Arg name="isiweburl" value="\{inargs.isiwebuserid\}" />
  </Response>
  <property name="keystoreref" value="DefaultKeyStore" />
  <property name="keyobjectref" value="DefaultSigner" />

  <property name="saml.assertion.issuer" value="http://fulvia.adnovum.ch:880" />
  <property name="saml.assertion.subject" value="\{sess:ch.nevis.session.userid\}" />
  <property name="saml.assertion.include_cert" value="false" />
  <property name="saml.assertion.audience_restriction" value="\{inctx:ch.nevis.isiweb4.auth.saml.AudienceRestriction\}" />
  <property name="saml.response.sign" value="true" />
  <property name="saml.response.target" value="\{notes:saml.SP.URL\}" />
  <property name="saml.assertion.tolerance" value="60" />
</AuthState>

<!-- this one for testing -->

<AuthState name="TestUseridPassword"
class="ch.nevis.esauth.auth.engine.UidPwLoginTest" authLevel="auth.test">
  <ResultCond name="ok" next="AuthDone" />
  <ResultCond name="firstlogin" next="AuthDone" />
  <ResultCond name="pwchange" next="AuthDone" />
</AuthState>
<Response value="AUTH_CONTINUE">
  <Gui name="AuthUidPwDialog" label="login.test.label">
    <GuiElem name="lasterror" type="error"
      label="${notes.lasterrorinfo}" value="${notes.lasterror}" />
    <GuiElem name="info" type="info" label="login.test.text" />
    <GuiElem name="isiwebuserid" type="text"
      label="userid.label" value="${notes.loginid}" />
    <GuiElem name="isiwebpasswd" type="pw-text"
      label="password.label" />
    <GuiElem name="submit" type="button"
      label="submit.button.label" value="Login" />
  </Gui>
</Response>
</AuthState>
Appendix B: Project management

Project Plan

A rough project plan was made during the first week. Because this was an individual project without division of work, no detailed project plan was created.

<table>
<thead>
<tr>
<th>Week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Related work, reading</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Set up environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Finish work &amp; docu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.1: Initial project plan

Because of problems setting up the server environment in the third week the project plan was adjusted. The setup phase was extended to the end of week 3 and the implementation phase was extended to a length of 3 weeks.

<table>
<thead>
<tr>
<th>Week</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Related work, reading</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Set up environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Finish work &amp; docu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.2: Adapted project plan
Meeting protocols

Kick-Off Meeting

Date Tuesday March 4 2008

Attendants Alex Suzuki
Andreas Steffen
Josua Trösch
Roman Pletka

Objectives for week 1

• Basically, Identity Federation for Webservices should be enabled.

• nevisProxy & nevisAuth: Mechanisms for handling SAML 2.0 Assertions exist but have to be extended.

• The Nevis Framework should be enabled to support the SAML 2.0 WebBrowser SSO Profile.

• Current SAML 2.0 support of common Application containers shall be documented.

• Milestones; It has been agreed on the following 4 Basic Milestones:

  2 weeks Familiarize with the Nevis Framework and the SAML 2.0 Standard
  3 weeks Implementation
  2 weeks Tests, Performance analyses
  1 week Documentation, Final Report

• Benefit of this work for AdNovum: Get an insight into the SAML 2.0 possibilities as a basis for later development.

• Documentation: should contain 50-60 pages, in an arbitrary format

• Meetings are held weekly on Tuesday, 8:00 o’clock

Next meeting: Tuesday, March 11, 8:00

33
Meeting week 2

Date Tuesday March 11 2008

Attendants Andreas Steffen
Josua Trösch
Roman Pletka

Done

- Application Container - Found out about SAML Support

  **WebLogic** Version 10.3 supports the SAML 2.0 Web Single Sign-on profile (= Web Browser SSO Profile?) and the WS-Security SAML Token profile 1.1.

  **Websphere** No information yet *(ongoing)*

  **JBoss** Supports SAML 1.0. SAML 2.0 will be implemented in the future, but no Roadmap yet.

- Studied the relevant part of the SAML 2.0 specification
- Set up and configured NevisBox in a local VM
- Familiarized with the important Nevis framework concepts
- Familiarized with the NevisProxy and NevisAuth Code *(ongoing)*
- Know the AdNovum culture

Objectives for week 2

- Find out if WebLogic sufficiently supports the Web Browser SSO Profile, more info about IBM Websphere
- Order an appropriate Application Container to complete the testing environment
- Conceptual formulation is read by all participants. It will serve as an exact agreement on this diploma thesis’ scope.
- Related work finished (Familiarize with environment)
- Documents to discuss will be sent to participants before Tuesday

Next meeting: Tuesday, March 18, 10:30
Meeting week 3

Date: Tuesday March 18 2008

Attendants: Alex Suzuki, Andreas Steffen, Josua Trösch, Roman Pletka

Done

• BEA Weblogic Server: Problem importing XML-File (the File validates against the Schema but Weblogic won’t accept it)

• Documentation: Needs to be worked up to the actual stand of work

• Implementation: Rough outline of the Implementation discussed with Alex Suzuki

Objectives week 3

• Solve Weblogic problems, configure as a SAML Service Provider

• Have a look at OpenSAML 2.0, it could be used in the implementation

• Documents: Will hand in a rough project plan for the next meeting

Next meeting: Tuesday, March 25, 8:00
Meeting week 4

Date Tuesday March 25 2008

Attendants Andreas Steffen
Roman Pletka
Josua Trösch

Done

- Weblogic konfigured, SAML Requests are now triggered when trying to access a resource
- Projectplan discussed at meeting, ok

Objectives week 4

- Have a closer look at the SAMLProvider Filter in nevisProxy, it probably has to be adapted to be able to handle SAML Requests
- Tests to be discussed with Alex Suzuki sometime, we want to conduct Performance tests

Next meeting: Tuesday, April 1, 8:00
Meeting week 5

Date Tuesday April 1 2008

Attendants Alex Suzuki
Andreas Steffen
Roman Pletka
Josua Trösch

Done

• Request are parsed and validated, Assertion is generated accordingly

Problems

• WebLogic is not accepting the SAML Assertion, (HTTP 404)

Objectives week 5

• Try the following to solve the weblogic problem
  jconsole to watch and configure MBeans for WebLogic
  ask Daniel Spoerndli for help, he worked with WebLogic and SAML 1.0
  try finding help in forums
  access WebLogic configuration via WLST

Next meeting: Tuesday, April 8, 8:00
Meeting week 6

Date Tuesday April 8 2008

Attendants Alex Suzuki
Andreas Steffen
Roman Pletka
Josua Trösch

Done

• WebLogic accepts Assertions

Objectives week 6

• Validate Requests regarding
  Check signature
  Check against configured Service Providers

• Partnerconfiguration: read Metadata XML Files

• OpenSAML 2.0 Library, what is the exact support

Next meeting: Tuesday, April 15, 8:00
Meeting week 7

Date Tuesday April 15 2008

Attendants Alex Suzuki
Andreas Steffen
Roman Pletka
Josua Trösch

Done

• Implementation ready

• Existing Unit Tests run successfully

Problems

• ProxySniffer seems to ignore the initial request: Probably localhost is not diverted in Proxy-BrowserConfig

Objectives week 7

• Complete Unit Test for added functionality

Presentation: Tuesday, April 22, 9:00
## Appendix C: Glossary

<table>
<thead>
<tr>
<th>Word / Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion</td>
<td>A SAML message containing security information about a principal</td>
</tr>
<tr>
<td>AuthnRequest</td>
<td>A SAML message containing a request for an Assertion</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>IdP</td>
<td>Identity Provider</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java Enterprise Edition</td>
</tr>
<tr>
<td>JAAS</td>
<td>Java Authentication and Authorization Service</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management Extensions</td>
</tr>
<tr>
<td>HTTP-Artifact Binding</td>
<td>SAML Binding specifying rules for referencing SAML messages</td>
</tr>
<tr>
<td>HTTP-Redirect Binding</td>
<td>SAML Binding using HTTP-Redirect to exchange messages</td>
</tr>
<tr>
<td>HTTP-POST Binding</td>
<td>SAML Binding using HTTP-POST to exchange messages</td>
</tr>
<tr>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>SecToken</td>
<td>An AdNovum proprietary format containing security information</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider</td>
</tr>
<tr>
<td>SSO</td>
<td>Single Sign On</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Identity Federation Use Case</td>
<td>viii</td>
</tr>
<tr>
<td>3.1</td>
<td>Basic SAML concepts</td>
<td>2</td>
</tr>
<tr>
<td>3.2</td>
<td>Basic SSO scenario</td>
<td>3</td>
</tr>
<tr>
<td>4.1</td>
<td>The Nevis Middleware Architecture <em>(Source: AdNovum)</em></td>
<td>5</td>
</tr>
<tr>
<td>4.2</td>
<td>Communication flow for Nevis administration and configuration</td>
<td>5</td>
</tr>
<tr>
<td>4.3</td>
<td>The nevisProxy Filter and Servlet Architecture</td>
<td>7</td>
</tr>
<tr>
<td>4.4</td>
<td>AuthEngine: state processing and configuration</td>
<td>8</td>
</tr>
<tr>
<td>6.1</td>
<td>Processing of a SAML Request</td>
<td>13</td>
</tr>
<tr>
<td>6.2</td>
<td>Load Test Setup</td>
<td>16</td>
</tr>
<tr>
<td>6.3</td>
<td>Load Test - Average login time</td>
<td>17</td>
</tr>
<tr>
<td>6.4</td>
<td>Load Test - Completed logins per minute</td>
<td>17</td>
</tr>
<tr>
<td>6.5</td>
<td>Load Test - Login failure rate</td>
<td>18</td>
</tr>
<tr>
<td>6.6</td>
<td>Load Test - CPU loads of the servers in percent</td>
<td>18</td>
</tr>
<tr>
<td>8.1</td>
<td>Initial project plan</td>
<td>32</td>
</tr>
<tr>
<td>8.2</td>
<td>Adapted project plan</td>
<td>32</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Listings

8.1 Minimal IdP-metadata configuration File ........................................ 24
8.2 web.xml ................................................................. 27
8.3 esauth4.xml ............................................................... 29
Bibliography


