

Autorisierung im Enterprise OpenSSO: Theorie und Praxis

Christoph Mathys
christoph.mathys at hslu.ch

CC Informations und Software Sicherheit
Hochschule Luzern - Technik & Architektur

10. Juni 2008

Zusammenfassung

OpenSSO ist eine Java Server Applikation, mit der zentralisierte Authentisierung und Autorisierung möglich wird. Durch die zugrunde liegende JavaEE Technologie ist eine gute Skalierbarkeit gegeben. OpenSSO bietet neben der Unterstützung für zahlreiche Authentisierungs-Backends auch Support für alle wichtigen ID Federation Protokolle (SAML 1/2, WS-Fed, Liberty). Mit dem Autorisierungsservice von OpenSSO kann auf verschiedenen Webservern (Apache, IIS, Tomcat, ...) Zugriffskontrolle implementiert werden, die zentral verwaltet wird.

Das Dokument ist in zwei Teile gegliedert. Thema des ersten Teils ist die Architektur und der Aufbau von OpenSSO. Im praktisch ausgerichteten zweiten Teil geht es darum, verschiedene Szenarien aufzubauen. Ein Szenario startet jeweils mit einer Beschreibung, welche Funktion realisiert wird, und leitet danach Schritt für Schritt zum Ziel.

Horw, 10. Juni 2008
INHALTSVERZEICHNIS

Inhaltsverzeichnis

1	Der SUN Access Manager / OpenSSO	3
1.1	OpenSSO und Sun Access Manager	3
1.2	Wie's funktioniert	3
1.3	Architektur von OpenSSO	4
1.3.1	Services von OpenSSO	4
1.3.2	Authentication Service	5
1.3.3	Session Service	6
1.3.4	Authorization Service	7
1.3.5	Identity Repository	9
2	Szenarios	9
2.1	Installation von OpenSSO auf Apache Tomcat 5.5	10
2.1.1	Anforderungen	10
2.1.2	Vorgehen	10
2.2	Installation der Kommandozeilen Tools für OpenSSO	11
2.2.1	Anforderungen	11
2.2.2	Vorgehen	12
2.3	OpenSSO Authentisierung/Autorisierung für einen Apache Webserver	12
2.3.1	Anforderungen	12
2.3.2	Vorgehen	12
2.4	OpenSSO und Active Directory	14
2.4.1	Anforderungen	15
2.4.2	Vorgehen (AD als DataStore)	15
2.5	OpenSSO Identity Federation mit SAMLv1.1	16
2.5.1	Anforderungen	17
2.5.2	Vorgehen	17
A	Akronyme	18
B	HTTP Cookies	19
C	SAML Source ID	19
C.1	SAML Source ID im Standard	19
C.2	SAML Source ID in Shib und OpenSSO	20

Horw, 10. Juni 2008

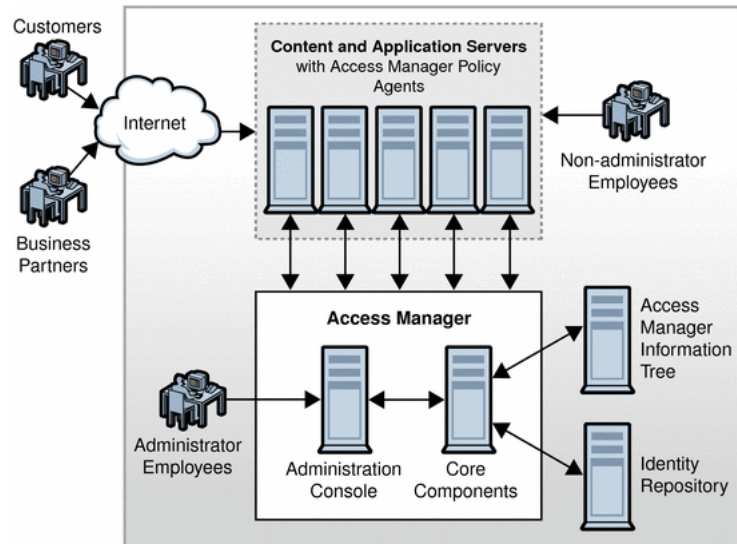


Abbildung 1: OpenSSO als Wächter der Daten einer Firma

1 Der SUN Access Manager / OpenSSO

Access Manager dient dazu, User zu authentisieren und zu autorisieren, und zwar sowohl innerhalb einer Firma als auch über die Firmengrenzen hinweg mittels Identity Federations. Weiter wird auch Single Sign On möglich. Access Manager kann Webapplikationen, Webserver und sonstige Applikationen schützen (Client SDK). Die Authentisierung und Authorisierung von Usern erfolgt zentral und für verschiedene Programme einheitlich. Bereits authentisierte Benutzer können alle Applikationen verwenden, auf die sie Zugriff haben, ohne sich erneut anmelden zu müssen.

1.1 OpenSSO und Sun Access Manager

Im Verlauf von 2006 hat Sun ihr kommerzielles Produkt Access Manager unter dem Namen OpenSSO als frei zugängliches Open Source Projekt unter der Common Development and Distribution License (CDDL) veröffentlicht. Neue Features werden in OpenSSO implementiert und getestet. In regelmässigen Abständen forkt Sun den Tree, fügt teilweise proprietären 3rd Party Code hinzu und macht daraus eine Release Version des Access Managers. Das gleiche Modell wird auch bei Sun Application Server 9 eingesetzt, der regelmässig aus dem Projekt Glassfish erzeugt wird.

1.2 Wie's funktioniert

Wenn ein Benutzer oder eine externe Applikation versucht, auf Daten des Firmenservers zuzugreifen, fängt ein Policy Agent die Anfrage ab und leitet sie um zu einem OpenSSO Server. OpenSSO fordert nun vom User Credentials wie beispielsweise Username und Passwort an. Falls die Credentials mit denjenigen im Identity Repository übereinstimmen, gelten die User Credentials als authentisch.

Nach dieser Authentisierung prüft der Policy Agent die Policies, die mit der Benutzeridentität verknüpft sind, um so zu erfahren ob der User zum Zugriff autorisiert ist. Policies werden mit dem Access Manager erzeugt und regeln, welche Benutzer (oder Gruppen von Benutzern) auf welche Ressourcen Zugriff haben. Weiter können sie auch Conditions spezifizieren, die zusätzliche Beschränkungen enthalten, z.B. zu welchen Uhrzeiten der Zugriff erfolgen darf, von welcher IP aus usw. Basierend auf dem Ergebnis dieser Policy Evaluation erlaubt oder verbietet der Agent den Zugriff des Benutzers auf

Horw, 10. Juni 2008
1.3 Architektur von OpenSSO

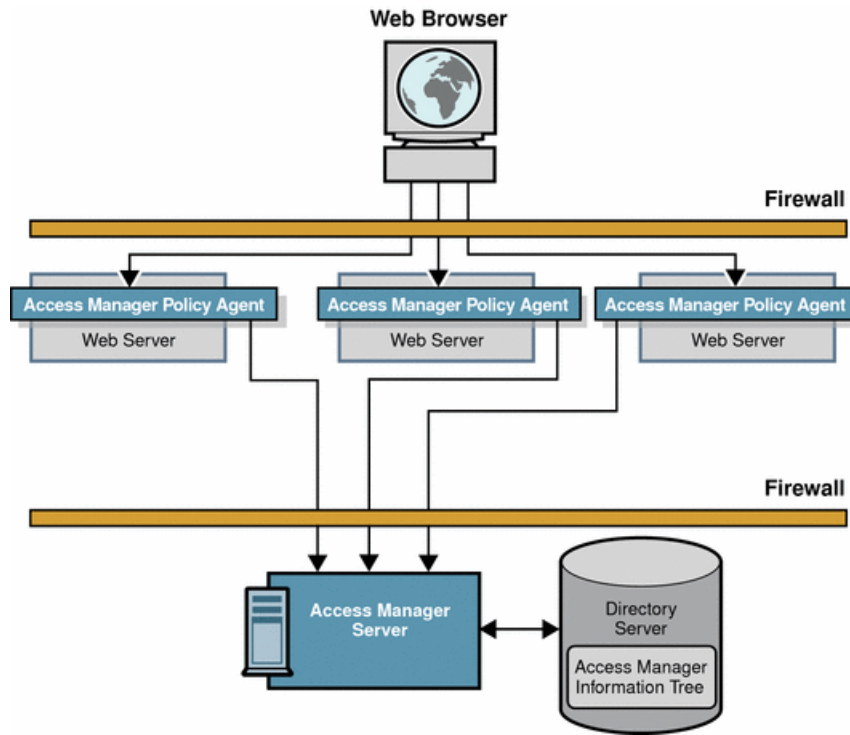


Abbildung 2: OpenSSO mit Agents

die Resource. Bild 1 stellt eine mögliche Konfiguration von OpenSSO als Wächter der Firmendaten dar.[3, p.16]

1.3 Architektur von OpenSSO

OpenSSO wurde als J2EE-Applikation entwickelt, die auf einem Web Container (Application Server, Tomcat, ...) deployed werden kann. Dadurch kann OpenSSO das Load Balancing von JavaEE ausnutzen. Kerntechnologien von OpenSSO umfassen HTML, XML und SOAP. Um andere Applikationen zu schützen hat SUN den Agent-Ansatz gewählt. Das bedeutet, dass auf jedem zu schützenden Webserver ein entsprechender Agent installiert werden muss, wie in Abbildung 2 dargestellt. Dieser Agent kommuniziert via Webservices mit OpenSSO. Für eigene Applikationen kann das Java oder C ClientSDK verwendet werden, welche die wichtigsten Dienste von OpenSSO als API zur Verfügung stellen.

1.3.1 Services von OpenSSO

OpenSSO bietet verschiedene Schnittstellen nach aussen als Webservice an:

1. Authentication
2. Authorization (Policy)
3. Session (SSO)
4. Auditing/Logging
5. Identity Repository Access
6. Federation

Horw, 10. Juni 2008

1.3 Architektur von OpenSSO

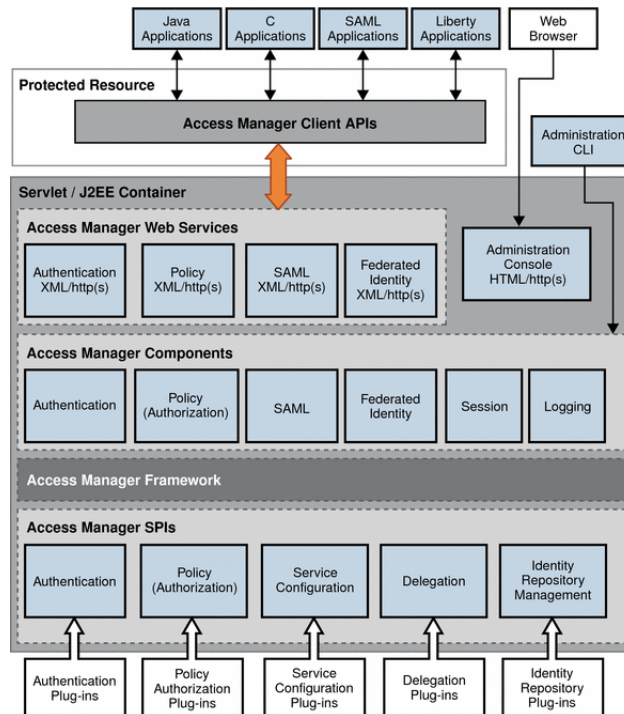


Abbildung 3: Architektur von OpenSSO

7. Web Services Security

Die Komponenten der Dienste 1-5 sind einfache Servlets, die XML Requests entgegennehmen und Antworten in XML zurückgeben. Die Dienste 6 und 7 basieren auf den Federation- und Webservice-standards SAML, Liberty Alliance und WS-Fed. Die Clientkomponente von jedem dieser Dienste ist als Java-API verfügbar. Die Dienste 1-4 sind zusätzlich als C-API verfügbar. Policy Agents basieren auf diesen APIs. Ein Skizze der Gesamtarchitektur ist in Abbildung 3 dargestellt. Die folgenden Abschnitte gehen auf die wichtigsten Komponenten genauer ein.

1.3.2 Authentication Service

Dieser Dienst prüft User Credentials. Er unterstützt eine Reihe von Plugins (Authentication Modules), um mit verschiedenen User Repositories zu kommunizieren, z.B. LDAP, RADIUS oder SecureID. Der Service interagiert mit der Autentisierungsdatenbank um die Credentials zu validieren und mit dem Identity Repository um Profile-Attribute des Benutzers zu laden. Wenn die Credentials des Users gültig sind, wird ein Session-Token erzeugt.

Die Authentication Modules können als Stack konfiguriert werden und besitzen jeweils ein Attribute OPTIONAL, REQUIRED, REQUISITE, SUFFICIENT welches besagt, was geschieht wenn der Benutzer korrekte resp. falsche Credentials liefert. Beim Login wird dieser Stack von oben nach unten abgearbeitet, bis die Bedingungen für einen erfolgreichen oder fehlgeschlagenen Login gegeben sind. PAM unter Unix funktioniert nach demselben Muster.

Der Authentication Service unterstützt neben der einfachen Validierung von Credentials verschiedene weitere Funktionen:

- Es gibt verschiedene Authentisierungstypen, die Konfiguriert werden können, beispielsweise User

Horw, 10. Juni 2008

1.3 Architektur von OpenSSO

oder Module basierte Authentisierung. Bei ersterem wird der User an einem spezifisch für ihn festgelegten Modul angemeldet. Bei der modulbasierten Authentisierung werden alle Benutzer an einem bestimmten Modul (z.B. LDAP) angemeldet.

- Mit Account Locking kann ein Account nach einer definierbaren Anzahl fehlgeschlagenen Anmeldeversuchen gesperrt werden.
- Authentication Chaining ermöglicht das Hintereinanderschalten von verschiedenen Authentisierungsmodulen (analog zu PAM). Das kann beispielsweise so aussehen, dass sich der User zuerst am AD anmeldet und anschliessend eine Zertifikat präsentieren muss.
- Mit Session Upgrade kann der Authentisierungslevel des Users erhöht werden, indem sich der User bei einem weiteren Modul mit einem höheren Level anmeldet.

1.3.3 Session Service

Der Session Service erzeugt eine neue Session, wenn sich ein User erfolgreich eingeloggt hat. In dieser Session werden die Interaktionen des Benutzers mit anderen Webapplikationen gespeichert. Der Session Service kann dadurch die Applikationen benachrichtigen, wenn die Session beendet wird oder Eigenschaften geändert werden. Die Session ID ist ein String, welcher eine bestimmte Instanz einer Session identifiziert. Wenn eine Applikation diesen String kennt, kann sie auf Eigenschaften der Session zugreifen. Die Session ID (auch Session Token genannt) wird in einem Cookie auf dem Client gespeichert. Dieser Cookie wird beim Zugriff auf Server in derselben Domain immer mitgeschickt.

Initiale Anmeldung

Policy Agents prüfen bei einer Anfrage, ob eine gültige Session vorhanden ist. Falls nicht, wird der Browser zum Authentication Service umgeleitet. Der Authentication Service holt beim Session Service ein neues Session Token. Der Session Service erzeugt eine Session im Zustand 'Invalid' und gibt deren ID, einen zufälligen String zur Identifikation der Session, als Session Token zurück. Dieses Session Token wird als Cookie beim Browser gesetzt.

Der Benutzer muss nun dem Authentication Service seine Credentials präsentieren. Sind sie gültig, aktiviert der Service beim Session Service über eine Reihe von Calls die Session, was soviel bedeutet wie dass ihr Zustand auf 'Valid' gesetzt wird und einige Attribute in der Session gespeichert werden (Login Time, Level, ...). Die Session Datenstruktur (SSOToken) wird dem Agent geschickt, welcher den User ursprünglich zum Authentication Service umgeleitet hat. Der Browser des Users wird nach erfolgreicher Authentisierung ebenfalls zu diesem Agent umgeleitet, nun aber mit einem gültigen Session Token. Dieser Vorgang ist in Abbildung bla in einem Sequenzdiagramm dargestellt.

Der Agent erhält vom Browser bei diesem zweiten Request das Session Token und validiert es beim Session Service. Ist das Token gültig geht der Ablauf beim Prüfen der Policies weiter. Das SSOToken, welches die Daten der Session enthält, hat der Agent während der Authentisierung des Users bereits vom Session Service erhalten.

Single Sign On

SSO beginnt dann, wenn der User auf eine weitere durch einen Agent geschützte Applikation zugreift. Wiederum fängt der Agent den Request ab und prüft, ob ein Session Token (auch SSOTokenID) vorhanden ist. Er findet dieses Token in Form des Browser Cookies, welches bei der ersten Anmeldung gesetzt wurde. Mit diesem Token kann der Agent über die SSO API das SSOToken, also die Session Datenstruktur, laden. Anschliessend prüft der Agent die Gültigkeit des Session Tokens beim Session Service. Ist das Token gültig, kann der Zugriff abhängig von den Policies erfolgen.

Horw, 10. Juni 2008
1.3 Architektur von OpenSSO

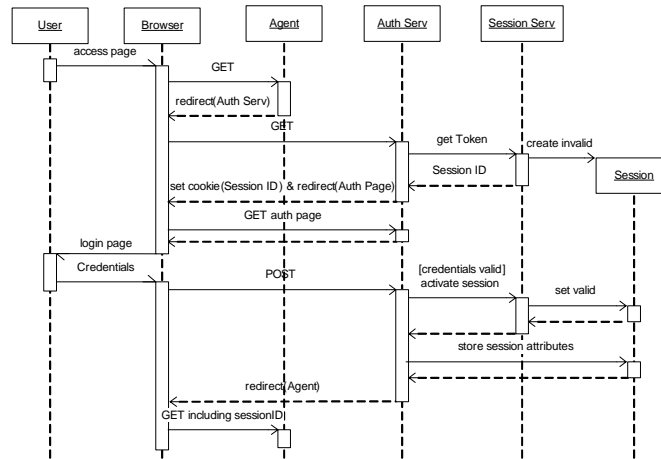


Abbildung 4: Authentisierung eines Users beim Request einer Website

Security Issue: Session Cookie Hijacking

Ein grosses Problem von obigem Ansatz ist das Risiko des sogenannten Session Cookie Hijacking, das heisst Malory gelangt in den Besitz des Session Tokens von Bob. Hat Malory erst das Cookie erhalten, kann er sich bei jeder Applikation als Bob ausgeben und erhält dieselben Zugriffsrechte. Malory hat verschiedene Möglichkeiten, in den Besitz des Cookies zu gelangen:

- Er kann Fehler in einer Applikation ausnutzen um eigenen Code einzuschleusen, der ihm den Cookie übermittelt.
- Er kann eine eigene Applikation innerhalb der Cookie Domain laufen lassen, die ihm das Cookie von Bob übermittelt.
- Bei ungeschützten Verbindungen kann er den Cookie sniffen.

Um diese Probleme zu lösen kann das in OpenSSO vorhandene Cross Domain SSO (CDSSO) verwendet werden. Eigentlich für den Fall entworfen, dass SSO in Applikationen in unterschiedlichen DNS-Domains ermöglicht werden soll, leistet es für die obigen Bedrohungen gute Dienste, auch wenn nur eine einzelne Domain verwendet wird. Jeder Agent und der OpenSSO Server selber müssen allerdings für die Verwendung von CDSSO speziell konfiguriert werden. Der Grundgedanke hinter dieser in [2] vorgeschlagenen Lösung ist, dass für jeden Server eine eigene Session erzeugt wird. Damit SSO trotzdem möglich ist, wird der Browser beim ersten Zugriff auf einen neuen Agent zum OpenSSO Server umgeleitet. Von diesem erhält er ein proprietäres HTML Form, welches er wieder dem Agent schickt. Der Agent setzt schliesslich das Cookie im Browser für genau seinen Server.

1.3.4 Authorization Service

Der Authorization oder Policy Service ist ein Dienst, der basierend auf Regeln entscheidet, wer auf welche Ressourcen zugreifen darf. Der Service unterstützt binäre (allow/deny) und nicht binäre Entscheidungen (ein Attribute, z.B. mailboxQuote). Generell definiert eine Policy, was ein Benutzer mit welchen Ressourcen unter bestimmten Bedingungen tun kann.

In Abbildung 5 ist der Ablauf dargestellt, wenn ein authentisierter Benutzer auf eine geschützte Resource zugreifen will. Die Teilnehmer haben dabei die folgenden Rollen und Aufgaben:

Policy Agent Der PA fungiert als Policy Enforcement Point (PEP), d.h. ihm obliegt die Aufgabe, dem User den Zugriff zu ermöglichen oder zu verbieten. Dazu fragt er beim PDP nach.

Horw, 10. Juni 2008

1.3 Architektur von OpenSSO

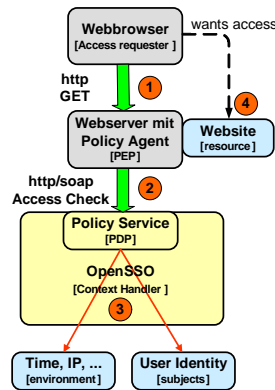


Abbildung 5: PDP und PEP

OpenSSO Der OpenSSO-Server arbeitet als Policy Decision Point (PDP) und Context Handler. Als PDP muss der Server entscheiden, ob eine Anfrage erlaubt wird oder nicht. Für diese Entscheidung kann der PDP über den Context Handler weitere Attribute abfragen, beispielsweise die Uhrzeit oder die Art der Anmeldung des Benutzers.

Die folgenden Abläufe sind in Abbildung 5 dargestellt:

1. Der Webbrowser schickt ein http GET an den Webserver.
2. Der Policy Agent fängt den Request ab und schickt den Namen der angeforderte Resource, die UserID und weitere Parameter an den Policy Service (Kommunikation via Webservice)
3. Der Policy Service entscheidet, ob der Request erlaubt wird oder nicht. Dazu kann er weitere Informationen abfragen (IP des Webbrowsers, Gruppen-Informationen des Users, Tageszeit usw.).
4. Die Entscheidung wird dem Agent geschickt. Ist der Zugriff erlaubt, wird der Request dem Webserver weitergeleitet. Ansonsten erhält der Benutzer eine entsprechende Fehlermeldung.

Aufbau einer Policy

Eine Policy ist aus vier verschiedenen Elementen aufgebaut, wobei drei auf die Entscheidung des Policy Service Einfluss haben:

- Eine Rule gibt an, welche Aktionen auf ein bestimmte Ressourcen angewendet werden dürfen.
- Die Subjects definieren, für welche Benutzer/Gruppen/Rollen die oben definierten Rules zutreffen.
- Bei den Conditions können zusätzliche Bedingungen angegeben werden, beispielsweise die Zeitspanne, während der Zugriff möglich ist, IPs für welche der Zugriff erlaubt ist usw. Sind mehrere Conditions vorhanden, werden sie folgendermassen verknüpft: Conditions derselben Kategorie werden mit einem logischen OR zusammengefasst, anschliessend werden die verschiedenen Kategorien AND-Verknüpft¹.
- Die Response Providers können der Antwort weitere Attribute hinzufügen, beispielsweise Infos aus dem Profil des zugreifenden Users.

¹Siehe dazu den Kommentar der Methode `getConditionDecision(SSOToken token, Map env)` in der Klasse `com.sun.identity.policy.Conditions` (CVS Version 1.3).

Horw, 10. Juni 2008

Neben den oben beschriebenen Normal Policies gibt es auch die Referral Policy. Diese Art von Policy delegiert sowohl Verwaltung wie auch Evaluation von Policies an eine andere Realm in OpenSSO (für die ein anderer Admin zuständig sein kann). Mit den entsprechenden Plugins kann die Delegation sogar zu anderen Produkten erfolgen.

1.3.5 Identity Repository

Die User-Identities in OpenSSO können aus verschiedenen Quellen bezogen werden. Die Basis ist dabei der Datastore von OpenSSO, welcher direkt über das Webfrontend verwaltet werden kann. Welche weiteren Dienste als Quellen für Identitäten verfügbar sind, hängt vom jeweiligen OpenSSO Service ab. Der Policy Service unterstützt Identitäten von einem LDAP-Server oder aus dem Datastore. Der Authentication Service unterstützt neben LDAP und Datastore noch eine ganze Reihe von weiteren Authentisierungsdiensten, beispielsweise RADIUS oder SecureID.

Datastore

Ein Datastore speichert alle Identitätstypen von OpenSSO. Dazu gehören UserIDs, AgentIDs, Rollen und Gruppen. Über das Webfrontend können neue Einträge erzeugt und bestehende modifiziert werden. Die im Store gespeicherten Identitäten stehen zur Authentisierung und als Identitäten für die Authorisierung zur Verfügung (als Access Manager Identities, wobei je nach Typ des Datastore z.B. Rollen nicht zur Verfügung stehen). Die AM Identities können auch in der J2EE Declarative Security² verwendet werden. Als DataStore kann OpenSSO entweder Files oder einen LDAPv3-kompatiblen Directory Service verwenden (z.B. OpenLDAP oder Active Directory).

2 Szenarios

OpenSSO ist ein umfangreiches und mächtiges Tool. Seit dem Start dieses Projektes wurden viele Erfahrungen mit Solaris, den Tools von Sun, JavaEE und vor allem Access Manager/OpenSSO gemacht. Daraus werden in diesem Kapitel eine Reihe von Szenarios konstruiert, die typische Anwendungsfälle von OpenSSO wiedergeben sollen. Die Szenarios hier bauen auf Tomcat, OpenSSO und Linux. Diese Kombination eignet sich vorzüglich zum experimentieren, da einfach aufgebaut und schnell konfiguriert (im Gegensatz zu einer mit dem JES-Installer konfigurierten Umgebung).

Einige allgemeine Bemerkung zur Infrastruktur: smb://htad03/workgroup/ISIS/Projekte/_1120228_Authorisierung_im_EUmfeld/software

- Die **Systemzeit** auf den verschiedenen Hosts mit OpenSSO-Software (Server und Agents) muss übereinstimmen. Sessions reagieren sensibel auf falsche Systemzeiten. Bei falsch eingestellten Systemuhren kann es passieren, dass ein Server denkt, der User sei korrekt angemeldet und ihn an den Nächsten verweist, dieser aber befindet, dass die Session bereits abgelaufen ist und ihn zurück schickt. Eine einfache Möglichkeit zum Zeitabgleich stellt NTP dar.
- Alle Hosts mit OpenSSO oder Agents sollten über Forward Lookups zu finden sein, d.h. entweder muss DNS eingerichtet werden oder das Hosts-File muss auf allen Rechnern die entsprechenden Einträge enthalten. Am Besten wird für alle Rechner dasselbe File verwendet, damit die Einträge konsistent sind. DNS Aliases (CNAME) funktionieren ebenfalls, falls der Host bereits in einem anderen DNS eingetragen ist (was beispielsweise im Enterpriselab der Fall ist). Bei der Namensgebung ist zu beachten, dass alle Hosts einen gemeinsamen Teil im DNS-Namen haben müssen, die sogenannte Cookie Domain. Hosts innerhalb dieser Cookie Domain können mit Agents und OpenSSO gesichert werden.

²Die J2EE Declarative Security erlaubt es, in Programmen die Rollen eines Benutzers abzufragen. Es ist dann möglich, im Deskriptor die erforderlichen Rollen zum Verwenden eines Servlets festzulegen oder direkt im Code die Rollen eines Users auszuwerfen.

Horw, 10. Juni 2008

2.1 Installation von OpenSSO auf Apache Tomcat 5.5

Die referenzierten Dateien sind alle im Intranet auf folgendem Server zu finden:

2.1 Installation von OpenSSO auf Apache Tomcat 5.5

In diesem Szenario werden wir den OpenSSO Server auf einem Tomcat 5.5 deployen. Damit ist der Grundstein gelegt, um weitere Szenarien aufzubauen. Das hier beschriebene Deployment ist eines der einfachsten überhaupt. Konfigurationsdaten und Identities liegen direkt im Filesystem und Tomcat läuft mit root-Rechten. Die Beschreibung hier enthält auch einige spezifische Hinweise für Ubuntu Linux. Die Installation funktioniert allerdings auf Solaris oder Windows sehr ähnlich.

2.1.1 Anforderungen

Infrastruktur Der Server sollte einen Hostnamen haben, in diesem Beispiel werden wir *opensso.test.local* in */etc/hosts* verwenden. Der Client, auf dem der Browser läuft, muss den Server über den Namen erreichen können.

Betriebssystem Linux (Ubuntu Server 7.10 mit OpenSSH Server)

Software FAM Build 1 (*fam_build1.zip*), Sun Runtime Library (*fam8-sun-runtime.zip*), Apache Tomcat 5.5,

2.1.2 Vorgehen

1. Als erstes installieren wir das Sun Java 6 JDK und unzip. Dabei wird auch der avahi-daemon installiert, der Apples Bonjour-Protokoll spricht. Für unsere Zwecke stört der Daemon, also schalten wir ihn aus.

```
# aptitude install sun-java6-jdk unzip
# update-rc.d -f avahi-daemon remove
# /etc/init.d/avahi-daemon stop
```

2. Nachdem das JDK installiert ist, müssen wir JAVA_HOME setzen, da viele Programme, darunter auch Tomcat, auf diese Umgebungsvariable angewiesen sind. Dazu öffnen wir mit einem Editor die environment-Datei und fügen die Definition von JAVA_HOME zuunterst hinzu:

```
# vim /etc/environment
JAVA_HOME="/usr/lib/jvm/java-6-sun"
```

3. Jetzt kopieren wir FAM Build 1, Apache Tomcat 5.5 und die Sun Runtime Library aufs System und entpacken die Archive mit unzip. Den Apache Webserver entpacken wir direkt ins opt-Verzeichnis:

```
# unzip apache-tomcat-5.5.25.zip -d /opt/
```

4. Apache bringt nicht ganz alle Standard-Bibliotheken mit, die OpenSSO benötigt. Daher müssen wir die Datei *webservices-api.jar* aus der Sun Runtime Library ins Verzeichnis für endorsed Libraries kopieren (quasi nachinstallieren):

```
# cp webservices-api.jar /opt/apache-tomcat-5.5.25/common/endorsed
```

5. Im nächsten Schritt müssen wir die Grösse des Java Heaps von Tomcat erhöhen, damit die Installation von OpenSSO überhaupt durchlaufen kann. Dazu editieren wir *catalina.sh* mit einem Editor und fügen die entsprechenden JAVA_OPTS unterhalb des einführenden Kommentars ein.

```
# vim /opt/apache-tomcat-5.5.25/bin/catalina.sh
JAVA_OPTS="-Xms256m -Xmx1024m"
```

Horw, 10. Juni 2008

2.2 Installation der Kommandozeilen Tools für OpenSSO

6. Bevor wir den Server starten, markieren wir die verschiedenen Shell-Scripts von Tomcat noch als ausführbar.

```
# chmod +x /opt/apache-tomcat-5.5.25/bin/*.sh
```

7. Damit sind die nötigen Anpassungen an Tomcat vorgenommen und wir starten den Server wieder, um anschliessend *fam.war* aus dem Archiv FAM Build 1 zu deployen. Zum deployen der Applikation müssen wir das war-File ins richtige Tomcat-Verzeichnis kopieren. Der Server bemerkt das neue Archiv und führt das Deployment durch.

```
# /opt/apache-tomcat-5.5.25/bin/startup.sh
# cp deployable-war/fam.war /opt/apache-tomcat-5.5.25/webapps/
```

8. Als nächstes konfigurieren wir den OpenSSO-Server. Wenn alles funktioniert hat ist das Config-Servlet von OpenSSO unter der URL *http://opensso.test.local:8080/fam/* mit einem normalen Webbrowser zu erreichen. Wir wählen die Variante Custom zur Konfiguration und tragen in den Feldern des GUI folgende Werte ein:

Password	admin123
Server URL	http://opensso.test.local:8080
Cookie Domain	.test.local
Configuration Directory	/opt/etc/opensso
Server Settings	Embedded
Suffix to store configuration Data	dc=opensso,dc=java,dc=net
Service Management config Data	
Name	localhost
Port	50389
Directory Administrator DN	cn=Directory Manager
Password	adminadmin

Tabelle 1: Config von OpenSSO

9. Nachdem sich OpenSSO fertig konfiguriert hat erscheint eine entsprechende Meldung. Wird jetzt wieder die fam-Website vom Server aufgerufen erscheint ein Login-Dialog, an dem wir uns mit *amadmin/admin123* anmelden und so auf die Konfigurationsdialoge von OpenSSO zugreifen können.

2.2 Installation der Kommandozeilen Tools für OpenSSO

Neben dem Webfrontend stehen zur Administration von OpenSSO auch Kommandozeilen-Programme zur Verfügung. Einige Funktionen sind ausschliesslich über diese Tools erreichbar. Dieses Szenario beschreibt, wie die Tools auf dem OpenSSO-Server installiert werden.

2.2.1 Anforderungen

Betriebssystem Linux (Ubuntu Server 7.10 mit OpenSSH Server)

Software FAM Build 1 (*fam_build1.zip*)

Szenarien Szenario 2.1

Horw, 10. Juni 2008

2.3 OpenSSO Authentisierung/Autorisierung für einen Apache Webserver

2.2.2 Vorgehen

1. Als erstes wird FAM Build 1 entpackt.
2. Als nächstes legen wir einen neuen Ordner für die Kommandozeile-Tools an und entpacken die Datei mit den Tools in dieses Verzeichnis.

```
# mkdir /opt/fam_bin
# unzip tools/famAdminTools.zip -d /opt/fam_bin
```

3. Unter den entpackten Dateien befindet sich ein setup-Shellscript, welches wir zur Konfiguration der Tools benutzen.

```
# cd /opt/fam_bin
# ./setup
Path to config files of server (example:
/opt/OpenSSO/config):/opt/etc/opensso
```

4. Die Installation ist damit schon abgeschlossen. Zum Test, obs auch funktioniert, müssen wir das Passwort von amadmin in ein Datei speichern (keine Passworte auf der Kommandozeile!) und anschliessend eines der Kommandos aus *fam/bin* aufrufen.

```
# echo admin123 > /root/fampwd
# /opt/fam_bin/fam/bin/famadm list-auth-instances \
--realm / --adminid amadmin --password-file /root/fampwd
```

2.3 OpenSSO Authentisierung/Autorisierung für einen Apache Webserver

In diesem Szenario wird der Inhalt auf einem Apache Webserver gegen unbefugten Zugriff abgesichert. Dazu wird ein Policy Agent installiert, welcher alle Zugriffe abfängt und zunächst überprüft, ob der Benutzer die entsprechenden Berechtigungen besitzt. Die Benutzer werden vom OpenSSO-Server authentisiert. Die Verwaltung der Berechtigungen geschieht ebenfalls dort. Abbildung 6 zeigt die verschiedenen Systeme, die beteiligt sind. Auf jedem Webserver, der via OpenSSO gesichert werden soll, muss ein Policy Agent installiert werden. In diesem Szenario werden wir zuerst den Webserver und den Policy Agent installieren und konfigurieren. Danach werden wir Benutzer und Policies in OpenSSO anlegen, die den Zugriff regeln.

2.3.1 Anforderungen

Infrastruktur Für dieses Szenario werden wir einen zweiten Linux-Server verwenden, der unter dem Namen *apache.test.local* erreichbar ist. D.h. die drei involvierten Rechner Client, OpenSSO Agent und OpenSSO Server müssen im Hosts-File den Rechner *apache* und den Rechner *opensso* stehen haben.

Betriebssystem Linux (Ubuntu Server 7.10)

Software Policy Agent für Apache 2.2 (*apache_v22_Linux_agent.zip*)

Szenarien Szenario 2.1

2.3.2 Vorgehen

1. Zuerst müssen wir den Apache Webserver installieren. Weiter brauchen wir unzip zum entpacken der Dateien und eine JRE zum Ausführen des agentadmin-Programms. Den avahi-daemon schalten wir nach der Installation ab.

Horw, 10. Juni 2008

2.3 OpenSSO Authentisierung/Autorisierung für einen Apache Webserver

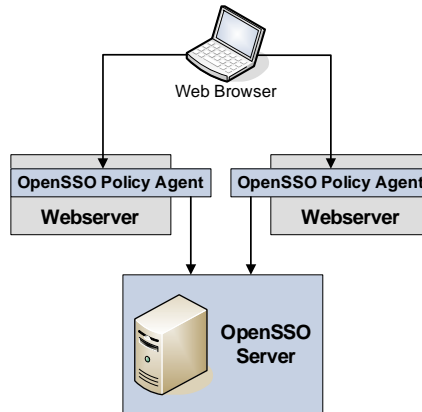


Abbildung 6: Struktur eines typischen OpenSSO-Deployments

```
# aptitude install apache2 unzip sun-java6-jre
# update-rc.d -f avahi-daemon remove
# /etc/init.d/avahi-daemon stop
```

2. Als nächstes kopieren wir den Policy Agent für Apache 2.2 auf das System und entpacken ihn. Dabei können wir auch gleich die enthaltenen Scripts als ausführbar markieren und eine Datei mit dem Passwort von amAdmin anlegen.

```
# unzip apache_v22_Linux_agent.zip -d /opt/
# chmod +x /opt/web_agents/apache22_agent/bin/*
# echo admin123 > /fam_pwd.txt
```

3. Als nächstes starten wir das Installationsskript des Agents. Es stellt eine ganze Reihe von Fragen und passt am Ende die Config vom Apache an.

```
# /opt/web_agents/apache22_agent/bin/agentadmin --install
```

Apache Config Directory	/etc/apache2/
Access Manager Service Host	opensso.test.local
Access Manager Services port	8080
AM Services protocol	http
AM Deployment URI	/fam
Agent Host Name	apache.test.local
Port for Web Server Instance	80
Preferred Protocol for Instance	http
Agent Profile name	UrlAccessAgent
path to password file	/fam_pwd.txt

Tabelle 2: Konfiguration des Policy Agents

4. Der Agent ist nun installiert, es fehlen ihm allerdings noch einige Shared Libraries zum Funktionieren. Welche das sind können wir mit dem Kommando `ldd` überprüfen:

```
# ldd /opt/web_agents/apache22_agent/lib/libamapc22.so
```

Horw, 10. Juni 2008

2.4 OpenSSO und Active Directory

- Wir installieren nun die fehlenden Bibliotheken. Anschliessend müssen wir einige Symlinks machen, damit die Bibliotheken auch wirklich gefunden werden. Der Agent wurde für Redhat Linux geschrieben, und offenbar tragen die Libs dort teilweise andere Namen als in unserem Ubuntu Server.

```
# aptitude install libxml2 libnss3-0d
# cd /usr/lib
# ln -s libnss3.so.0d libnss3.so
# ln -s libssl3.so.0d libssl3.so
# ln -s libplc4.so.0d libplc4.so
# ln -s libplds4.so.0d libplds4.so
# ln -s libnspr4.so.0d libnspr4.so
```

- Nun sind alle Bibliotheken vorhanden, die benötigt werden. Jetzt müssen wir im OpenSSO Server noch ein Profil für den Agent anlegen, damit beispielsweise den Policy Service benutzen kann. Dazu beim OpenSSO Server mit amAdmin:admin123 anmelden. Im Tab Access Control die opensso-Realm anklicken. Nun zum Tab Subjects gehen. Dort wählen wir den Tab Agent, dann New. ID ist "UrlAccessAgent", Type "Policy Agent" und das Passwort lautet "admin123". Mit OK bestätigen.
- Ein Neustart des Apache-Server steht nun noch an, dann sind wir fertig.

```
/etc/init.d/apache2 restart
```

- Zum Testen rufen wir mit einem Webbrowser eine Website vom Apache ab: `apache.test.local`. Zunächst werden wir nun zum OpenSSO-Server weitergeleitet und müssen gültige Credentials angeben. Danach werden wir zurück zum Apache-Server geschickt und erhalten dort eine Zugriffsverweigerung, solange wir im Policy-Service keine Regeln erstellen, die den Zugriff erlauben. **Vorsicht:** Der OpenSSO Server muss gestartet sein, ehe wir Apache starten, sonst stürzt Apache beim Start ab.

2.4 OpenSSO und Active Directory

Microsofts Active Directory findet sich in zahlreichen Firmen als User Repository. In diesem Szenario werden wir OpenSSO so konfigurieren, dass sich User aus dem Active Directory bei OpenSSO anmelden können. In den Policies zur Autorisierung werden direkt Benutzer und Gruppen im AD referenziert.

Grundsätzlich gibt es zwei Varianten, AD zur Authentisierung und Authorisierung zu verwenden:

- Die Authentisierung von Benutzern erfolgt über das Active Directory Authentication Module. Der Policy-Service kann entweder mit OpenSSO Identities oder mit Identities von einem externen LDAP-Server arbeiten. Genau davon kann für dieses Szenario gebrauch gemacht werden. Nachteil dieser Variante ist, das AD an zwei verschiedenen Orten referenziert werden muss (Authentisierungs-Modul und Policy-Service). Auch J2EE-Security funktioniert damit nicht.
- Als Datastore wird AD einmal konfiguriert und fortan stehen alle AD-User als OpenSSO Identities zur Verfügung. Rollen und spezielle Agents-Accounts gibt es allerdings nicht. Bei der Konfiguration der Authentisierung wird einfach auf den DataStore verwiesen. Diese Variante hat den Vorteil, dass auch ein begrenztes User-Management (ändern von Attributen/Gruppenzugehörigkeit) auf der AM-Webkonsole möglich ist.

Als Username wird bei einem Standard-Konformen LDAP das Attribut UID verwendet. Im AD bleibt das Attribut leider leer. Zwei andere Attribute kommen in Frage:

Horw, 10. Juni 2008

2.4 OpenSSO und Active Directory

sAMAccountName Diese Kennung wird meist verwendet, um sich bei Windows anzumelden und kann z.B. `jdoe` lauten. Sie ist allerdings in Multidomain-Umgebungen nicht eindeutig.

UserPrincipalName Das ist die Eindeutige Kennung in einer AD-Umgebung. Sie setzt sich aus dem `sAMAccountName` und der Domain zusammen, also beispielsweise `jdoe@hta.fhz.ch`.

Bei diesem Szenario wird der `sAMAccountName` verwendet, da wir ohnehin nur eine Domain benutzen. OpenSSO unterhält Benutzerprofile in seinem Datastore. Dort sind beispielsweise die Frage und Antwort zum Passwortreset oder eine benutzerspezifische Loginkonfiguration (andere Reihenfolge von Loginmodulen) gespeichert. Ob dieses Profil nötig ist und allenfalls beim ersten Login angelegt werden soll, entscheidet die Eigenschaft "User Profile". Die folgenden Einstellungen sind möglich:

Dynamic Der Account und einige definierte Attribute werden in den DataStore von OpenSSO kopiert, der Benutzer steht ab dem ersten Login als AM-Identity zur Verfügung.

Dynamic with User Aliases ???

Required Die Credentials des Users werden beim Modul geprüft. Falls sie korrekt sind und der Benutzer auch im OpenSSO Store existiert, ist die Anmeldung erfolgreich.

Ignored Username und Passwort werden beim Loginmodul verifiziert. Weder wird ein Profil für den Benutzer angelegt noch ist es von Bedeutung, ob ein Benutzer-Profil im OpenSSO DataStore vorhanden ist.

2.4.1 Anforderungen

Infrastruktur Für dieses Szenario brauchen wir einen Active Directory Server. Beim Test habe ich Windows 2003 R2 Enterprise verwendet. Der Server heisst `dc.test.local`, der Administrator hat das Passwort `isis`. Die Domain hat das LDAP-Suffix `dc=test,dc=local`.

Szenarien Szenario 2.1, Szenario 2.3

2.4.2 Vorgehen (AD als DataStore)

- Wir werden die Konfiguration des AD DataStores in einer neuen Realm vornehmen. Im Tab Access Control legen wir eine neue Realm mit dem Namen "AdStore" an. Ein Klick auf die neu angelegte Realm führt uns zur Konfiguration derselben.
- Nun wechseln wir zum Tab Data Stores. Hier löschen wir den bestehenden Files-Datastore (keine Angst, er wird nur in dieser Realm gelöscht). Danach erzeugen wir einen neuen DataStore durch einen Klick auf New. Wir wählen als Typ "Active Directory" und geben den Namen "AD". Next!
- In diesem Screen gibt es eine ganze Reihe von Werten zu ändern/ergänzen:

LDAP Server	dc.test.local:389
LDAP Bind DN	CN=Administrator,CN=Users,dc=test,dc=local
LDAP Bind Password	isis
LDAP Organization DN	dc=test,dc=local
LDAPv3 Search Scope	SCOPE_SUB
Users Search Attribute	sAMAccountName
Attr Name for Group Membership	memberOf
LDAP People Container Naming Attr	<leer>
LDAP People Container Value	<leer>
ID Types That Can Be Authenticated	User
Authentication Naming Attribute	sAMAccountName

Horw, 10. Juni 2008

2.5 OpenSSO Identity Federation mit SAMLv1.1

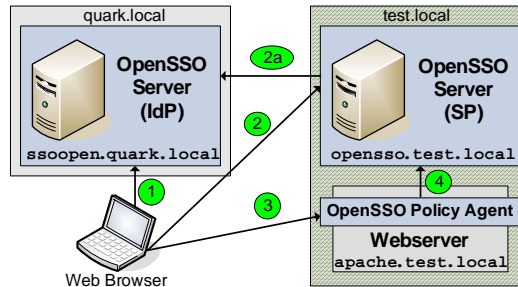


Abbildung 7: Identity Federation: Beispielablauf mit SAMLv1.1

Tabelle 3: Konfigurationswerte für den AD Datastore

4. Nach dem obige Werte eingegeben wurden, den Screen mit Finish verlassen. Zum Testen, ob die Anbindung funktioniert, können wir zum Tab Subjects wechseln. Erscheinen dort unter User und Group die Einträge aus dem AD, so war die Konfiguration erfolgreich.
5. Damit wir nun die User aus dem AD für Policies nutzen können, müssen wir in der Root-Realm eine Delegation (Referral Policy) zu unserer eben kreierten Realm einrichten. Dazu klicken wir auf den Button Back to Realms und wählen die Realm "opensso". Dort zum Tab Policies, dann New Referral. Neue URL-Policy Rule für `http://apache.test.local:80/*`, dann ein neues Referral zur Realm "AdStore". Nun können wir im Tab Policies der Realm AdStore regeln für diesen Server anlegen.
6. Nun ist noch eine Änderung an der Konfiguration des Agents nötig. Das Login-Servlet, welches Username/Passwort abfragt, muss wissen, bei welcher Realm es einen Benutzer anmelden muss. Dazu dient der Servlet Parameter "realm". Auf dem Rechner `apache.test.local` öffnen wir die Datei `AMAgent.properties` und passen die Zeile mit der Servlet URL an. Danach müssen wir den Webserver neu starten.

```
# vim /opt/web_agents/apache22_agent/Agent_001/config/AMAgent.properties
com.sun.am.policy.am.login.url =
    http://opensso.test.local:8080/fam/UI/Login?realm=AdStore
# /etc/init.d/apache2 restart
```

2.5 OpenSSO Identity Federation mit SAMLv1.1

In diesem Szenario werden wir eine einfache Identity Federation mit zwei OpenSSO-Servern und einem durch einen Agent geschützten Apache aufbauen. Die beiden OpenSSO-Server werden die Identitätsinformationen über SAMLv1.1 (Artifact Profile) austauschen. Das Ziel ist, dass Benutzer der Domain `quark.local` auf den Apache in der Domain `test.local` zugreifen können. Der Clue dabei ist, dass nur der OpenSSO-Server von `quark.local` die Benutzer kennt, `test.local` vertraut `quark.local`, dass er die Benutzer korrekt identifiziert. `quark.local` ist also in diesem Szenario der Identity Provider (IdP), während `test.local` einen Dienst (Apache Webserver) anbietet und somit als Service Provider (SP) funktioniert.

Abbildung 7 stellt den Ablauf für Identity Federating mit SAMLv1.1 (Artifact Profile) vereinfacht dar:

1. Als erstes ruft der Client in `quark.local` den Intersite Transfer Service auf und meldet seinem OpenSSO Server, auf welche Resource er in `test.local` zugreifen will. Dabei wird er auch gleich vom IdP authentisiert.

Horw, 10. Juni 2008

2.5 OpenSSO Identity Federation mit SAMLv1.1

2. Nach erfolgreicher Anmeldung wird der Client zum OpenSSO-Server von *test.local* weitergeleitet, in der URL trägt er eine Referenz zur SAML Assertion. Der OpenSSO-Server von *test.local* holt die Assertion vom IdP (2a) und erzeugt eine Session in der Domain *test.local*.
3. Der Client wird nun zur angeforderten Resource (Apache Webserver) weitergeleitet.
4. Der Agent führt wie gewohnt die Session Validierung durch und nutzt anschliessend den Policy-Service zur Entscheidung, ob der Zugriff erlaubt wird (also wie üblich).

2.5.1 Anforderungen

Infrastruktur Für dieses Szenario benötigen wir zwei OpenSSO Instanzen und einen mit Agent geschützten Apache. In der Domain *test.local* stehen die Rechner *opensso* und *apache*. Die Domain *quark.local* enthält den zweiten OpenSSO-Server, *ssoopen*.

Szenarien Szenario 2.1 (2x), Szenario 2.3

2.5.2 Vorgehen

Als erstes brauchen wir die SAML Site Identifier der beiden OpenSSO-Rechner. Der Site Identifier wird von der Asserting Party zur Relying Party übertragen, damit diese mit dem Artifact die Assertion holen kann. Um sie nachzuschauen gehen wir zum Tab Federation, dann unter SAML 1.x den Button Local Site Properties klicken. Unter Site Identifiers klicken wir auf die Instance ID (<http://opensso.test.local:8080>). Den Wert unter Site ID brauchen wir (hier dYipZeolmbx-uHlnSvQDP0PC1ND0=). Im weiteren Text werden die Site IDs mit `$OPENSSEO_ID` für die ID von *opensso.test.local* und `$$SSOOPEN_ID` für *ssoopen.quark.local* referenziert. Für nähere Informationen zur SAML Site ID siehe Apeendix C.

ssoopen.quark.local

1. Als erstes konfigurieren wir den Rechner *ssoopen* als IdP, d.h. SAML Source. Dazu im Tab Federation unter SAML 1.x Trusted Partners auf New klicken. Unter Destination Artifact und SOAP Query auswählen. Wir sind die Source, unser Trusted Partner, den wir hier definieren, ist also die Destination. Next!
2. Nun müssen wir einige Werte in die Felder einfüllen. Die Namen der Servlets, welche die Aufgaben übernehmen, sind aus der SUN Dokumentation.

Source ID	<code>\$OPENSSEO_ID</code>
Host List	<code>opensso.test.local</code>
Target	<code>apache.test.local:80</code>
SAML URL	<code>http://opensso.test.local:8080/fam/SAMLAwareServlet</code>

Tabelle 4: SAML Config von ssoopen.quark.local

3. Ein Klick auf Finish sind wir bereits fertig mit der Konfiguration des IdP.

opensso.test.local

1. Nun richten wir die Trustbeziehung vom Service Provider (*opensso*) zum Identity Provider (*ssoopen*) ein. Wiederum unter Federation, SAML 1.x unter "Trusted Partners" auf New klicken. Diesmal wählen wir unter Source Artifact aus. Next!
2. Die Werte in der folgenden Tabelle eintragen:

Horw, 10. Juni 2008

Source ID	\$SSOOPEN_ID
SOAP URL	http://ssoopen.quark.local: 8080/fam/SAMLSOAPReceiver

Tabelle 5: SAML Config von opensso.test.local

3. Als nächstes müssen wir die Authentisierung so konfigurieren, dass der Benutzer kein Profil beim SP erhält. Dazu im Tab Access Control die opensso-Realm auswählen.
4. Nun im Tab Authentication den Button Advanced Properties... betätigen.
5. User Profile umstellen auf ignored. Das heisst der SP ignoriert es, wenn er in seinem DataStore kein Profil für den Benutzer findet und legt auch keines an.
6. Unter den Module Instances muss das Modul "Federation" vorhanden sein. Falls nicht muss eine neue Modulinstanz davon konfiguriert werden.
7. Zum Schluss legen wir noch im Tab Policies eine Policy an, welche allen Authentisierten Benutzern den Zugriff auf den Apache Webserver erlaubt. Wir fügen dem allerdings die Condition bei, dass sie sich beim Federation-Modul angemeldet haben. Das bedeutet dann, dass alle Benutzer von *ssoopen.quark.local* auf den Apache Webserver zugreifen dürfen, allerdings niemand sonst.

Der Test

1. Für den Test müssen wir im Grunde nur den Intersite Transfer Service von ssoopen ansteuern und uns unter Angabe unserer Credentials weiterleiten lassen. Die Service URL lautet folgendermassen: *http://ssoopen.quark.local:8080/fam/SAMLAwareServlet?TARGET=http://apache.test.local:80*
2. Nach der Eingabe der Credentials werden wir zum Apache Webserver weitergeleitet.

A Akronyme

AM Access Manager

AD Microsoft Active Directory

LDAP Lightweight Directory Access Protocol

FIM Federated Identity Management

FAM Federated Access Manager

SSO Single Sign On

NTP Network Time Protocol

SAML Security Assertion Markup Language

ID-FF Identity Federation Framework

WS-FED Webservice Federation

RADIUS Remote Authentication Dial-In User Service

PAM Pluggable Authentication Modules

CDDL Common Development and Distribution License

Horw, 10. Juni 2008

B HTTP Cookies

Ein HTTP-Cookie, auch Browser-Cookie genannt, bezeichnet Informationen, die ein Webserver zu einem Browser sendet oder die clientseitig durch JavaScript erzeugt werden. Der Client sendet die Informationen in der Regel bei späteren Zugriffen unverändert an den selben Webserver im Hypertext-Transfer-Protocol-Header an den Server. Cookies sind clientseitig persistente/gespeicherte Daten.

Cookies werden in den Kopfzeilen (Header) von Anfragen und -Antworten via HTTP übertragen. Das HTTP ist per Definition ein zustandsloses Protokoll, daher ist für den Webserver jeder Zugriff völlig unabhängig von allen anderen. Eine Webanwendung, die sich über einen längeren Zeitraum hinwegzieht, muss mit Zusätzen auf der Anwendungsschicht (im Browser) arbeiten, um den Teilnehmer über mehrere Zugriffe hinweg identifizieren zu können.[4]

Cookies können neben einer Name-Value Property noch einige Attribute enthalten, unter anderem einen Domain-Namen. Der Cookie wird dann mitgeschickt, wenn ein Request zu einem Server in dieser Domain geschickt wird. Aus Sicherheitsgründen werden Cookies nur akzeptiert, wenn der Server selber mitglied der Domain ist, die im Domain String angegeben ist.

C SAML Source ID

Die SAML Source ID identifiziert eindeutig den Identity Provider gegenüber einem Service Provider im Artifact Profile. Ist die Source ID falsch konfiguriert, kann der SP die zu einem Artifact gehörende Assertion nicht anfordern und der Ablauf schlägt fehl. Dieses Kapitel beschäftigt sich damit, wie die ID innerhalb von SAML verwendet wird, wie sie aufgebaut ist und wo sie in den Produkten verwendet wird.

C.1 SAML Source ID im Standard

Beim SAML Artifact Profile wird die Assertion nicht direkt gePOSTed. Stattdessen wird der URL beim Redirect zur Relying Party ein Parameter (SAMLart, das Artifact) angehängt, welcher aus der ID der Asserting Party und der ID der Assertion selber besteht. Mit dem Artifact kann die Relying Party die Assertion anfordern. Das Artifact ist ein Base64-kodierter String, welcher sich aus den folgenden drei Teilen zusammensetzt (aus [1, S.17]):

```
TypeCode           := 0x0001
RemainingArtifact := SourceID AssertionHandle
SourceID           := 20-byte-sequence
AssertionHandle    := 20-byte-sequence
```

Die Source ID identifiziert eine SAML Asserting Party, daher müssen alle Sites mit gemeinsamen Destinations verschiedene IDs besitzen. Die Relying Party sucht mit dieser ID in einer lokalen Liste die URL des SAML Responders der Source Site, um die zum Artifact gehörende Assertion anzufordern. Diese Liste mit dem ID-URL Mapping wird out-of-band ausgetauscht, d.h. zum Beispiel durch die Administratoren konfiguriert oder mithilfe von Metadaten eingespeist. Typischerweise wird die SourceID aus einer URL berechnet, im Standard wird der SHA-1 Hash der Site URL als Wert für SourceID vorgeschlagen.

Innerhalb der Source Site wird die Assertion durch das AssertionHandle referenziert. Dieses Handle sollte aus einer kryptografisch starken Zufalls- oder Pseudozufallszahl abgeleitet werden.

Der TypeCode identifiziert die obige Art von Artifact. Durch das Hinzufügen von weiteren TypeCodes können weitere Formen von Artifacts definiert werden. Die oben beschriebene Form von Artifact ist allerdings für alle SAML 1.1-konformen Implementation ein MUST.

Horw, 10. Juni 2008

C.2 SAML Source ID in Shib und OpenSSO

C.2 SAML Source ID in Shib und OpenSSO

Die SourceID wird in verschiedenen SAML Produkten unterschiedlich konfiguriert, teilweise direkt, teilweise nicht.

Shibboleth berechnet die Source ID aus dem Konfigurationswert `providerId` im Config-File des Identity Providers. Für einen IdP in der `aaitest`-Federation von Switch kann dieses Feld beispielsweise den Wert `urn:mace:switch.ch:aaitest:el.hta.fhz.ch` haben. Intern berechnet Shibboleth die Source ID durch das Berechnen des SHA-1 Hashes aus diesem Feld.

Access Manager/OpenSSO erfolgt die SAML Administration direkt über die Source ID. Bei der Installation wird die Source ID aus dem Hostnamen berechnet (inklusive `http[s]://` und Port, als z.B. `https://opensso.test.local:8443`). Zur Berechnung des SHA-1

In Shibboleth bekommt man die Erzeugte ID nie zu Gesicht, in Access Manager/OpenSSO muss man sie sogar herunkopieren. In Shibboleth werden IdPs (Asserting Party) durch die `providerId` (z.B. `urn:mace:switch.ch:aaitest:el.hta.fhz.ch`) identifiziert. OpenSSO erzeugt beim Installieren aus dem Hostnamen die Source ID.

Wenn nun ein Shibboleth IdP mit einem OpenSSO SP zusammenarbeiten soll, muss der Base64-kodierte SHA-1 Hash der Shib `providerId` des IdP beim Trusted Partner eingetragen werden. Umgekehrt, also wenn ein OpenSSO IdP mit einem Shib SP arbeiten soll, müsste entsprechend der Hostname von OpenSSO in den Metadaten des Shib SP eingetragen werden.

Um den SHA-1 Hash eines Strings zu berechnen bringt OpenSSO ein Tool mit. Es ist im Client SDK zu finden (`am_sdk.jar/openssclientsdk.jar`) und berechnet aus einem String den Base64-kodierten SHA-1 Hash. Das Tool wird folgendermassen verwendet:

```
# java -cp openssclientsdk.jar com.sun.identity.saml.common.SAMLSiteID \  
    urn:mace:switch.ch:aaitest:el.hta.fhz.ch \  
    kdyXeTgDL+Hos61HGwU5soHXA7E=
```

Literatur

- [1] Eve Maler, Prateek Mishra, and Robert Philpott. Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1, September 2003. `oasis-sstc-saml-bindings-1.1`.
- [2] *Technical Note: Precautions Against Cookie Hijacking in an Access Manager Deployment*. Sun Microsystems, Inc., Sun Microsystems, Inc. 4150 Network Circle Santa Clara, CA 95054 U.S.A., 819-7849-10 edition, Februar 2006.
- [3] *Sun Java System Access Manager 7.1 Technical Overview*. Sun Microsystems, Inc., Sun Microsystems, Inc. 4150 Network Circle Santa Clara, CA 95054 U.S.A., 819-4669 edition, März 2007.
- [4] Wikipedia. HTTP-Cookie — Wikipedia, Die freie Enzyklopädie. <http://de.wikipedia.org/w/index.php?title=HTTP-Cookie&oldid=37908878>, 2007. [Online; Stand 25. Oktober 2007].