SAML-basierte Single Sign On Frameworks

Eugen Weiss

Seminar-Arbeit

am

Lehrstuhl für Netz- und Datensicherheit
Prof. Dr. Jörg Schwenk

betreut durch Juraj Somorovsky

20.08.2010

Horst-Görtz Institut Ruhr-Universität Bochum

Zusammenfassung

Der Single-Sign On (SSO) ist eine Technik, die es erlaubt durch einmalige Authentifizierung mehrere Dienste, die nicht unter einer Domain erreichbar sein müssen, zu nutzen. In dieser Seminararbeit geht es um den SAML-basierten SSO und bekannte Frameworks, die diesen implementiert haben. Es werden die Frameworks von Google, WSO2 und Shibboleth vorgestellt und wichtige Aspekte bezüglich SAML und Sicherheit heraus gearbeitet.

Inhaltsverzeichnis

1 Einleitung

Unternehmen wollen mit ihren Web-Diensten viele Nutzer erreichen und Ihnen eine möglichst komplette und komfortable Lösung anbieten. Deshalb entschließen sie sich oft zur Kooperation mit anderen Unternehmen, die entweder die Anwendung um wichtige Funktionen ergänzen oder aber einen anderen Dienst anbieten, der für den Nutzer interessant sein könnte.
Damit die Übergänge zwischen den einzelnen Anwendungen für den Nutzer transparent bleiben und er sich nicht jedes Mal beim betreten neu authentifizieren muss, gibt es bestimmte Mechanismen um eine einmalige Authentifizierung, ein so genanntes SSO (Single-Sign On), zu ermöglichen. Ein Standard auf diesem Gebiet stellt das SAML(Security Assertion Markup Language) dar. Es wurde innerhalb von OASIS vom Security Services Technical Commitee(SSTC) entwickelt. Die erste Version (SAML 1.0) ist im Jahr 2002 publiziert worden und ist seitdem kontinuierlich weiterentwickelt worden. Die aktuelle Version SAML 2.0 bringt viele nützliche Erweiterungen mit, die in Zusammenarbeit mit anderen Projekten, wie z.B. das „Internet2 Shibboleth project“, entwickelt wurden.
Eine grundlegende Beschreibung der wichtigsten Aspekte von SAML2.0 gibt es im zweiten Kapitel dieser Seminararbeit. Das Hauptaugenmerk der Arbeit liegt auf den SAML2.0-basierten SSO Frameworks. Im dritten Kapitel wird zunächst ein Überblick über einige bekannte Frameworks gegeben. Dazu gehören Shibboleth, WSO2 und das von Google zur Verfügung gestellte Framework für den SAML-basierten SSO für Google Apps. Es wird herausgestellt, welche Lösungen diese anbieten und wodurch sie sich unterscheiden. Dabei wird vor allem darauf eingegangen welche wichtigen Eigenschaften der SAML2.0 Spezifikation umgesetzt worden sind. Das folgende Kapitel beschäftigt sich mit der Sicherheit bei der Übertragung der Nachrichten im SSO-Prozess. Wir schauen uns an welche Mechanismen die Frameworks nutzen um die Kommunikation zu sichern.

2 Grundlagen SAML2.0

Dieses Kapitel behandelt einige Grundlagen von SAML, die wichtig für das Verständnis des SAML-basierten SSO sind.

2.1 SAML Einführung

SAML stellt einen Standard dar um mit Hilfe von so genannten „Assertions“ (deutsch:Behauptung) Aussagen über Parteien zu treffen. Eine Partei ist dabei eine Person, kann aber auch ein Unternehmen sein. Die Assertion kann abhängig vom Anwendungsfall Angaben zur Authentizität, Berechtigungen und Eigenschaften einer Partei liefern. SAML ist ein XML-basiertes Framework, folglich werden alle Anfragen und Antworten als XML-Dokumente geliefert. Das hat den Vorteil, dass es unabhängig von der verwendeten Plattform und Infrastruktur eingesetzt werden kann. Neben den Assertions gibt es weitere Bausteine, die in SAML definiert werden:
Der Ablauf der Anfragen und Antworten zwischen den Parteien ist durch Protokolle spezifiziert. Weiterhin wird durch die Bindung (Binding) definiert, wie das Zusammenspiel von SAML-Protokollen mit Standard-Protokollen, wie z.B. SOAP oder HTTP, funktioniert. Schließlich beschreibt ein Profil (Profile), wie SAML-Protokolle, Bindungen und Assertions in einem bestimmten Anwendungsfall, z.B. Web Single-Sign-On, eingesetzt werden (vgl. [OAS05d]). Die einzelnen Bausteine werden in Abbildung 2.1 veranschaulicht.

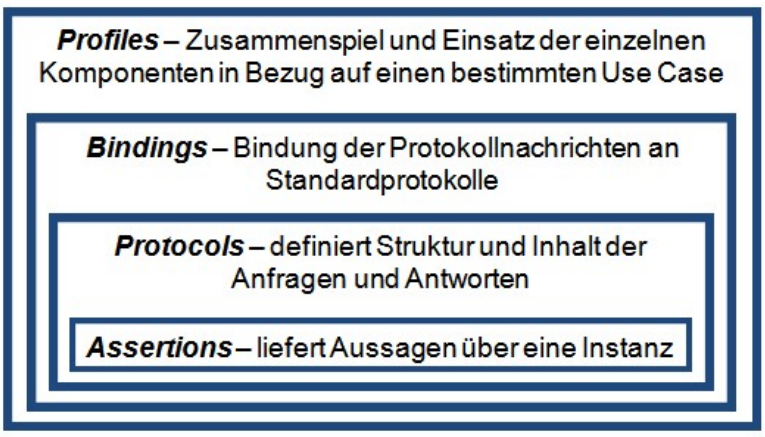

Abbildung 2.1: Service Provider initiierter SSO

In dieser Seminararbeit liegt der Schwerpunkt auf der SSO Funktionalität, die im Folgenden

genauer beschrieben wird.

2.2 SAML Web SSO

Der Web SSO (Single-Sign-On) ist ein Mechanismus, der es ermöglicht durch eine einmalige Authentifizierung auf einer Webseite, Zugang zu weiteren Web-Diensten zu bekommen, ohne sich nochmal zu authentifizieren. Es ist dabei nicht wichtig, dass die Dienste unter einer Domain laufen. Der Übergang muss für den Nutzer transparent bleiben.
In SAML2.0 wird unterschieden zwischen dem Identity Provider und dem Service Provider. Unter dem Identity Provider versteht man eine vertrauenswürdige Partei, wo man sich mit seinen Benutzerdaten authentifizieren kann. Wenn die Anmeldung erfolgreich war, erfolgt im Hintergrund eine Anfrage von Identity Provider zum Service Provider, die eine Assertion mit den Authentifizierungsangaben enthält. Wenn der Benutzer auf den Dienst des Service Providers zugreifen darf wird er nun direkt dorthin weitergeleitet (vgl. [OAS05e]).

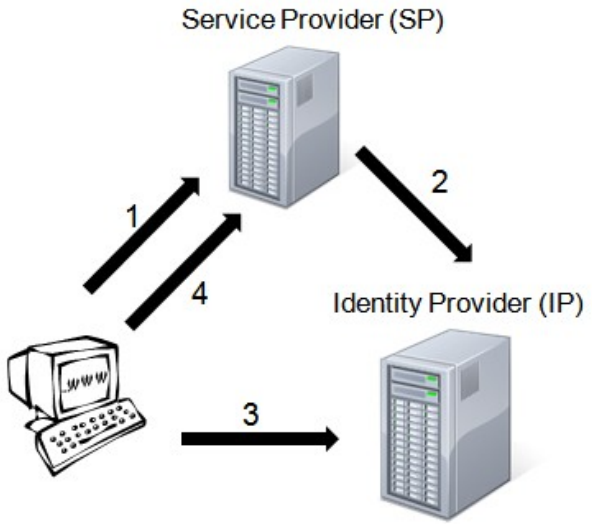

Abbildung 2.2: Service Provider initiierter SSO

In Abbildung 2.2 ist der Service Provider initiierte SSO dargestellt. Im ersten Schritt greift ein Nutzer auf den Service Provider zu. Es erfolgt eine Authentifizierungsanfrage an den Identity Provider und die Weiterleitung des Nutzers auf dessen Seite (Schritt 2). Nun meldet sich der Benutzer beim Identity Provider an (Schritt 3). Im Hintergrund wird eine Antwort vom Identity Provider zum Service Provider mit der Assertion geschickt. Schließlich landet der Benutzer im Schritt vier wieder auf der Ausgangsseite, die er ursprünglich aufgerufen hat.
Es gibt auch einen Identity Provider initiierten SSO, wo der Nutzer zunächst mit dem Identity Provider in Kontakt tritt und von dort aus, nach einer erfolgreichen Authentifizierung, weitere Dienste erreichen kann.

3 Vorstellung der Frameworks

In diesem Kapitel sollen drei bekannte Frameworks vorgestellt werden. Dabei handelt es sich um das Google SSO Framework, Shibboleth 2.0 und WSO2.

3.1 Google SSO Framework

Google bietet für die Entwickler eine Referenz-Implementierung an, welche sie dabei unterstützt den SSO Mechanismus für ihre Applikation zu realisieren. Google übernimmt dabei die Rolle des Service Providers. Anhand der Referenz-Implementierung kann nun ein Partner-Unternehmen den Prozess eines SAML2.0 basierten SSO nachvollziehen und ein Produkt entwickeln, wo sich Nutzer über Ihre Seite einloggen können und anschließend Zugriff auf Dienste von Google Apps haben (vgl. [Goo10d] und [Goo10c]). Das Framework wird in drei verschiedenen Sprachen zur Verfügung gestellt: .NET, Java und PHP. Diese Arbeit konzentriert sich auf die Analyse der Java-Lösung.
Der Ablauf eines Web SSO besteht immer aus Assertion-Anfragen bzw. -Antworten. Um dieses Konzept umzusetzen verwendet das Google Framework Servlets. Ein Servlet wird auf einem Web Server ausgeführt und kann dort Anfragen entgegennehmen, verarbeiten und beantworten. Der Unterschied zwischen einem Servlet und einer normalen Java-Klasse besteht darin, dass das Servlet die Schnittstelle `javax.servlet.Servlet` implementiert, die dem Entwickler, die grundlegenen Methoden für die Arbeit mit Servlets, zur Verfügung stellt (vgl. [Zei99]). In dem Framework erben die Servlets von der Klasse `javax.servlet.http.HttpServlet`, da für uns im Web nur das Protokoll HTTP von Bedeutung ist. Es werden insgesamt zwei Servlets verwendet. Für das Erstellen der Authentifizierungsanfragen vom Service Provider an den Identity Provider wird das CreateRequestServlet verwendet. Hier wird unter anderem der Provider Name und die ACS(Assertion Consumer Service)-URL übermittelt, an die die anschließende Antwort gehen soll. Die Antwort wird in dem ProcessResponseServlet generiert. Diese enthält den Namen des authentifizierten Benutzers. Da das Servlet nur die notwendigsten Funktionen zur Verfügung stellt um einen Identity Provider zu simulieren, muss ein Entwickler, der nach einer vollständigen Lösung, die einen Identity Provider beinhaltet, sucht auf andere Open-Source bzw. kommerzielle Produkte ausweichen (z.B. Shibboleth 2.0).
Sowohl für die Assertion Anfragen als auch für die Antworten existieren Templates, welche Platzhalter für die einzelnen Parameter beinhalten. Diese werden durch die bereits erwähnten Servlets ausgefüllt.

3.2 Shibboleth 2.0

Das Shibboleth 2.0 Framework (`http://shibboleth.internet2.edu/`) ist ein Open-Source Projekt, welches ein Web SSO System zur Verfügung stellt. Die Version 2.0 basiert auf SAML2.0 und bietet sowohl Lösungen für einen Identity Provider als auch für einen Service Provider an. Shibboleth Implementierungen sind in den Sprachen C++ und Java verfügbar.

3.2.1 Identity Provider

Der Identity Provider ist in Java implementiert und ist daher auf allen Plattformen lauffähig. Zur Installation wird ein installiertes Java 6 JDK sowie ein Servlet Container vorausgesetzt. Der Servlet Container wird benötigt da die Implementierung des IDPs auf Java Servlets basiert. Ein bekannter und oft eingesetzter Servlet Container ist Apache Tomcat.
Sobald man einen IDP auf seinem System installiert hat, kann die Konfiguration durchgeführt werden. Man kann sämtliche Einstellungen bequem in XML-Dateien durchführen. Dort wird festgelegt wie eingehende und ausgehende Nachrichten verarbeitet werden sollen. Außerdem kann man Konfigurationen bzgl. Transformation und Kodierung von Attributen in Nachrichten vollziehen.
Ein wichtiger Prozess in einem Web SSO use case ist der Authentifizierungsprozess und die Verwaltung von Identitäten beim Identity Provider. Bei der Authentifizierung werden von Shibboleth verschiedene Methoden unterstützt. Ein Nutzer kann sich z.B. mit seinem Benutzernamen und Passwort anmelden oder es wird die IP überprüft und der Zugang wird gewährt falls diese in einem bestimmten Bereich liegt (vgl. [Shi09b]). Die Verwaltung der Benutzerdaten erfolgt in einem LDAP Verzeichnis. Shibboleth ist mit einem LDAP Modul ausgestattet um wichtige LDAP Funktionen ausführen zu können (vgl. [Shi09a]).

3.2.2 Service Provider

Der Service Provider (SP) ist in C++ implementiert. Die Installation erfolgt abhängig vom eingesetzten Betriebssystem und Web Server. Am einfachsten ist die Konfiguration mit einem Apache Web Server.
Für den SP existiert eine zentrale Konfigurationsdatei, wo Eigenschaften des SPs, aber auch die Kommunikation mit dem Identity Provider eingestellt werden kann. Damit ein SP Authentifizierungsanfragen an einen IP stellen kann muss man diese einander bekannt machen. Das geschieht über das Laden von so genannten Metadaten. Diese enthalten unter anderem die Schlüsselinformationen zum Signieren der einzelnen Anfragen.

3.3 WSO2

WSO2 (`http://wso2.com/`) läuft unter der Apache Lizenz und ist deshalb ebenso wie Shibboleth ein Open-Source Projekt. Es wurde in August 2005 gegründet und entwickelt seitdem Lösungen für eine SOA (service-oriented architecture). In diesem Rahmen wurde auch der WSO2 Identity Server bereitgestellt, der zur Authentifizierung in einem Web SSO genutzt werden kann (vgl. [WSO10b]). Daneben wird auch eine etwas einfachere und flexible Lösung

angeboten, die so genannte „WSO2 Cloud Identity“. Hierbei wird die Funktionalität eines Identity Providers an WSO2 ausgelagert, die dafür Rechenkapazität und Infrastruktur zur Verfügung stellen. Die Aufsetzung eines Systems, welches administriert und gewartet werden muss, entfällt bei dieser Lösung (mehr Informationen auf der WSO2-Seite [WSO10a]). Dadurch kann ein Unternehmen viele Kosten einsparen.
Um den SSO für einen bestimmten Service mit Hilfe von WSO2 Cloud Identity einzurichten muss zunächst ein Account bei WSO2 erstellt werden. Dabei ist es notwendig in Besitz einer gültigen Domain zu sein, die bei der Registrierung verifiziert wird. Ist die Registrierung erfolgreich abgeschlossen, kann mit der Konfiguration des SSO begonnen werden. Dazu muss lediglich ein Keystore importiert und die ACS-URL des Service Providers konfiguriert werden (eine Beispielkonfiguration von WSO2 für Google Apps wird in diesem HowTo [Mah09] beschrieben).
Zum Betreiben des Identity Servers muss ein System verfügbar sein, wo dieser installiert werden kann. Die Installation ist aber unkompliziert, da WSO2 im Paket alles Notwendige mit liefert, wie z.B. einen Web Server oder eine Datenbank. Nach dem Entpacken des Pakets wird der Server über ein Script gestartet und über ein Web-Interface können alle Einstellungen durchgeführt werden. Der SSO mit SAML wird analog, wie bei der Cloud Identity eingerichtet. Der Identity Server bietet, eine komplette Infrastruktur für die Verwaltung von Identitäten und verschiedene Mechanismen zur Authentifizierung, an (z.B. OpenID). Die Infrastruktur wird in der folgenden Abbildung dargestellt.

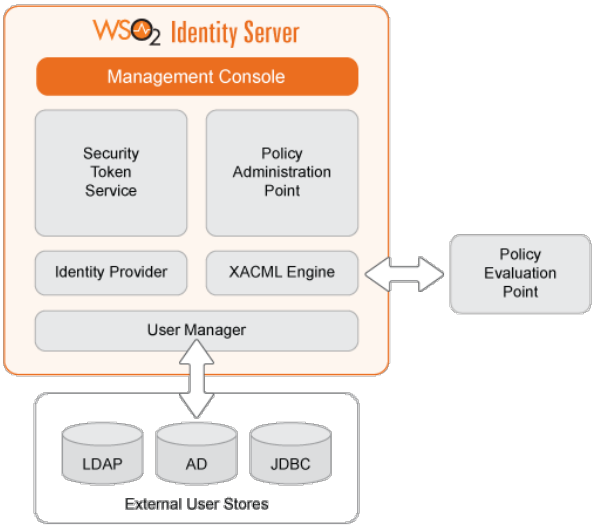

Abbildung 3.1: WSO2 Identity Provider Infrastruktur[1]

[1] http://wso2.com/wp-content/themes/wso2ng-v2/images/is-archi.gif

4 Umsetzung wichtiger SAML2.0 Aspekte

Es wird nun detaillierter auf die Umsetzung der SAML 2.0 Spezifikation eingegangen. Dazu wird das Format der Assertions, die mit den einzelnen Frameworks generiert werden, analysiert. Weiterhin werden die einzelnen Parameter, die bei den Anfragen und Antworten zwischen den Parteien ausgetauscht werden, betrachtet.

4.1 Service Provider initiierte SSO Kommunikation

In diesem Abschnitt wird auf die einzelnen Anfragen und Antworten, in einem Service Provider initiierten SSO, eingegangen.

4.1.1 SAML Authentication Request

Es wird im Folgenden der Authentication Request, der mit dem Google SAMLTool(vgl. [Goo10b]) generiert wurde, analysiert. Wie schon im Kapitel 3 dargestellt, wird ein Nutzer nach dem Aufruf einer Service Provider Seite, in diesem Fall Google Apps, auf die Seite des Identity Provider weitergeleitet, wo er sich authentifizieren muss. Dazu wird eine SAML Anfrage an den Identity Provider, welcher von der Referenzimplementierung simuliert wird, geschickt. Das erzeugte XML-Dokument ist im Folgenden Listing dargestellt.

Listing 4.1: Authentication Request

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <samlp:AuthnRequest
3 xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
4 ID="fpagejpkbhmddfodlbmnfhdginimekieckijbeei"
5 Version="2.0"
6 IssueInstant="2006-05-02T08:49:40Z"
7 ProtocolBinding="urn:oasis:names.tc:SAML:2.0:bindings:HTTP-Redirect"
8 ProviderName="google.com"
9 AssertionConsumerServiceURL="https://www.google.com/hosted/psosamldemo.net/acs"/>
```

- Zeile 4: Die `ID` ist ein zufälliger 160-Bit String, der für die Anfrage generiert wird.
- Zeile 5: `Version` ist ein statischer Parameter, wo die SAML-Version abgelegt ist.
- Zeile 6: `IssueInstant` enthält einen Zeitstempel im speziellen SAML Format YYYY-MM-DDTHH:MM:SSZ
- Zeile 7: Im `ProtocolBinding` ist das Binding HTTP-Redirect definiert. Dieses Binding ist in SAML spezifiziert und besagt, dass der Authentication Request durch einen HTTP Redirect mit HTTP Status 302 bzw. 303 ausgeführt wird (vgl. [OAS05e] S.25 ff.).

- Zeile 8: Hier wird der Domain-Name des Service Providers eingetragen.
- Zeile 9: Es wird die URL festgelegt, die zur Verifikation der SAML Antwort genutzt werden soll.

4.1.2 Encoded Redirect

Die Authentifizierungsanfrage wird via HTTP GET an den Identity Provider geschickt. Vorher findet aber ein Deflate Encoding der Nachricht statt, welches durch das folgende Binding definiert ist: `urn:oasis:names:tc:SAML:2.0:bindings:URL-Encoding:DEFLATE` (vgl. [OAS05b] S.17-18).Dazu muss die Anfrage zunächst mit Hilfe der DEFLATE-Methode(RFC-1951) [Deu96] komprimiert werden. Die Java-Bibliothek stellt dafür die notwendigen Funktionen zur Verfügung. Anschließend findet ein base64 Encoding und ein URL Encoding statt, wobei u.a. Leerzeichen entfernt werden. Die Anfrage wird nun als query-String mit dem Parameter `SAMLRequest` an die URL angefügt. Zusätzlich wird die ursprünglich aufgerufene Adresse als weiterer query-String im Parameter `RelayState`, ebenfalls nach einem URL Encoding, angefügt (vgl. [OAS05e] S.25 ff.). Im Folgenden ist ein Beispiel einer bereits kodierten Anfrage dargestellt:

Listing 4.2: kodierte Anfrage an den IDP

```
identity_provider.html?SAMLRequest=eJxdkE1PwzAMhs%2F9F1XubcM00IjWTQOEmDTQtA8O3NrEa7M1d
onTjZ9P2UAgrraf1489nn64Jj6CZ0uYi6tUihhQk7FY5WK7eUxGYjoZc%2BGaVs26UOMK3jvgEEc9iKzOnVx0H
hUVbFlh4YBV0Go9e16oQSpV6ymQpkZE84dc7Nqign17KGtnzI5MUzrc1aayaB0cLOiD3ZcAVsTR649Wn9LDzB3
MkUOBoS9JeZPI60QONnKkhrdqKN9EtPxedWfxcsE%2Fr%2FSvV3kZYvW02SyTFRjrQYdzyNEa8C89kYuKqGog1
eRENGMGH3qle0LuHPg1%2BKPVsF0tclGH0LLKstPplP5CWU0cwGQt09erDDhKEUJWaBbZ5BPy1YRc&RelaySta
te=http%3A%2F%2Fwww.google.com%2Fhosted%2Fpsosamldemo.net%2FDashboard
```

4.1.3 SAML Response

Als Nächstes muss der Identity Provider die Authentifizierungsanfrage wieder dekomprimieren und dekodieren. Es muss überprüft werden ob die Anfrage gültig ist und die ACS-URL muss extrahiert werden, da die anschließende Antwort an diese URL gehen soll. Es wird nun die generierte Antwort des IDPs betrachtet.

Listing 4.3: SAML Response

```
1  ï»¿<?xml version="1.0" encoding="UTF-8"?>
2  <samlp:Response ID="dbbifblnkdcmcmjaiolbgiekfjjocndkmdggabpj"
3  IssueInstant="2010-06-20T08:46:51.048Z"
4  Version="2.0" xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
5  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
6  <ds:SignedInfo>
7  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
8  <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
9  <ds:Reference URI="#dbbifblnkdcmcmjaiolbgiekfjjocndkmdggabpj">
10 <ds:Transforms>
11 <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
12 <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
13 <ec:InclusiveNamespaces PrefixList="ds␣saml␣samlp" xmlns:ec="http://www.w3.org/2001/10/
       xml-exc-c14n#"/>
14 </ds:Transform></ds:Transforms>
15 <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```
16 <ds:DigestValue>TmAYpj5h%2BZnX1jiNA/4JEgaw2xs=</ds:DigestValue>
17 </ds:Reference>
18 </ds:SignedInfo>
19 <ds:SignatureValue>jSsrYc0Y5NdGENVjHHTCzyZIb6CVKli0Eabc1GOmGJ42sUn4R%2B0kpOaMH4C%2
       BHJDQw%2BU0R5XIo3wr7RQ1LUuskqAXB5cnoypRVfcWtVf7hA8zsdM765/
       bsyS2JeiK4ttE5WpQ5X4MzBFyiWM69GlSWD%2BWkoQ8oYaG3UY6%2BgCcXGM=
20 </ds:SignatureValue>
21 <ds:KeyInfo>
22 <ds:X509Data>
23 <ds:X509Certificate>
       MIICSzCCAbSgAwIBAgIEawGsiDANBgkqhkiG9w0BAQQFADBqMTQwMgYDVQQDEytzYW1sc3NvdGVzdG
24 dvb2dsZW1haWxjb20uc2FtcGxlLWdhLmNvbS10ZXN0MQ0wCwYDVQQLEwROb251MRQwEgYDVQQKEwtO
25 b25lIEw9Tm9uZTENMAsGA1UEBhMETm9uZTAeFw0xMDA1MjAxNDA4MThaFw0yMDA2MTYxNDA4MThaMG
26 oxNDAyBgNVBAMTK3NhbWxzc290ZXN0Z29vZ2xlbWFpbGNvbS5zYW1wbGUtZ2EuY29tLXRlc3QxDTAL
27 BgNVBAsTBE5vbmUxFDASBgNVBAoTC05vbmUgTD1Ob25lMQ0wCwYDVQQGEwROb251MIGfMA0GCSqGSI
28 b3DQEBAQUAA4GNADCBiQKBgQDEDSPQOeGw4lLnUOOEX2XVNjrcPhvzSSqh8vzX5b9FPA6aRPdxqvoz
29 koVgf1Lrj21uK1Xvz91qtgLwPe%2B1UC%2BFCgh2GdKqIhRzrAjuk2oY6Ww5AG2g87b9Jj/nhkoNxH
30 l3EEVXdQ8%2BuezZdWAAgKheEH%2B%2BUz4l1HPxNKx8w4J3FQIDAQABMA0GCSqGSIb3DQEBBAUAA4
31 GBAIxXo%2BpYxxlNglyAmaapi3WnxVSLz9PzzJ5n3rEpMC2l8UYDX/f0VM60kIymFFz0uw2DGeg7EC
32 Vj8teMf3RM0u%2Bh%2BhCqpgzUf5RGfeFFnVr0LG5KjRwlsgZwaPHLufiOgkYqKNR9K6/qCs4/DTKi
33 8pjoHnd41x82G8alpeD/tS0y</ds:X509Certificate>
34 </ds:X509Data>
35 </ds:KeyInfo>
36 </ds:Signature>
37 <samlp:Status><samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/></
       samlp:Status>
38 <saml:Assertion ID="cbkcaajkhnenoijehchhdjinkgkjhhfenglnlccg" IssueInstant="2010-06-20
       T08:46:51.048Z" Version="2.0" xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
39 <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">https://identity
       .cloud.wso2.com/samlsso</saml:Issuer>
40 <saml:Subject><saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
       format:emailAddress">admin</saml:NameID>
41 <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
42 <saml:SubjectConfirmationData InResponseTo="onpadpgiimoldgfijijioifimabpggoefgcchjno"
       NotOnOrAfter="2010-06-20T08:51:51.048Z" Recipient="https://www.google.com/a/
       samlssotestgooglemailcom.sample-ga.com/acs"/>
43 </saml:SubjectConfirmation>
44 </saml:Subject>
45 <saml:Conditions NotBefore="2010-06-20T08:46:51.048Z" NotOnOrAfter="2010-06-20T08:51:51
       .048Z">
46 <saml:AudienceRestriction><saml:Audience>google.com/a/samlssotestgooglemailcom.sample-
       ga.com</saml:Audience>
47 </saml:AudienceRestriction></saml:Conditions><saml:AuthnStatement AuthnInstant="
       2010-06-20T08:46:51.048Z"><saml:AuthnContext>
48 <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</
       saml:AuthnContextClassRef></saml:AuthnContext>
49 </saml:AuthnStatement>
50 </saml:Assertion>
51 </samlp:Response>
```

Das Listing zeigt die Response, die bei einem SSO zu Google Apps nach einem erfolgreichen Login über WSO2 Cloud Identity, erstellt wurde. Im Folgenden wird detaillierter auf einige wichtige Parameter eingegangen:

- Zeile 39: Ein eindeutiger Bezeichner (meistens die Domain) für den Aussteller der Nachricht wird hier angegeben. In unserem Fall ist WSO2 der Aussteller.
- Zeile 40: Das Subjekt stellt den Inhaber (engl. bearer) der Assertion dar (vgl. [OAS05c] S.13). Über ihn werden Aussagen getroffen.

- Zeile 42: Das `SubjectConfirmationData`-Element besitzt das Attribut `InResponseTo` mit der ID der Authentifizierungsanfrage für die diese Antwort erstellt wurde. Das `NotOnOrAfter`-Attribut begrenzt die Gültigkeit der Nachricht. Ist dieser Zeitpunkt überschritten, wird die Assertion verworfen. Schließlich definiert das Attribut `Recipient` die ACS-URL, wo die Anfrage verifiziert werden soll.
- Zeile 45: Das `Conditions`-Element legt den Zeitraum der Gültigkeit für die Assertion fest.
- Zeile 46: Ein `AudienceRestriction`-Element muss vorhanden sein, falls ein Inhaber-Subjekt definiert wurde. Das Element enthält den eindeutigen Bezeichner des Service Providers. In diesem Fall Google.
- Zeile 48: Hier wird angegeben mit welcher Methode der Nutzer sich authentifiziert hat.

4.2 Metadaten

Bevor ein Identity Provider und ein Service Provider miteinander in Kontakt treten können, um z.B. ein Web SSO zu ermöglichen, müssen diese Informationen über die jeweils andere Partei erlangen. Zum Beispiel müssen die Zertifikate zur Verifikation der Signaturen von Assertions importiert werden.
Seit der Veröffentlichung des SAML2.0 Standards werden Metadaten unterstützt damit Informationen, über die an der Kommunikation teilnehmenden Parteien, einfach ausgetauscht werden können (vgl. [OAS07]). Shibboleth 2.0 hat dieses Konzept bereits umgesetzt. Dort können, sowohl für den IDP als auch für den SP, XML-Dokumente mit Metadaten angelegt werden. Wenn nun ein IDP mit einem neuen SP kommunizieren will, muss er zunächst dessen Metadaten herunterladen. Dazu wird ein neuer, so genannter „Metadata Provider“, konfiguriert, wo die URL, zum XML-Dokument mit den Metadaten des SPs, angegeben werden muss. Nach Abschluss der Konfiguration wird das Dokument heruntergeladen und lokal abgelegt. Umgekehrt muss man genauso vorgehen.
Nun wird genauer auf den Inhalt des Metadaten-Dokuments eingegangen. Bei der Installation eines Shibboleth IDP, wird automatisch eine passendes Metadaten XML-Dokument dafür angelegt. Das folgende Listing zeigt ein mit Shibboleth generiertes Metadaten-Dokument (vgl. [Shi09c]).

Listing 4.4: IDP Metadaten

```
1  <EntityDescriptor entityID="https://idp.machine.test/idp/shibboleth"
2                    xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
3                    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
4                    xmlns:shibmd="urn:mace:shibboleth:metadata:1.0"
5                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
6      <IDPSSODescriptor protocolSupportEnumeration="urn:mace:shibboleth:1.0␣
           urn:oasis:names:tc:SAML:1.1:protocol␣urn:oasis:names:tc:SAML:2.0:protocol">
7          <Extensions>
8              <shibmd:Scope regexp="false">machine.test</shibmd:Scope>
9          </Extensions>
10         <KeyDescriptor>
11             <ds:KeyInfo>
```

```
                <ds:X509Data>
                    <ds:X509Certificate>
MIIDKzCCAhOgAwIBAgIUSh38ymBUKBZ3JpI6AY7oX9cZQbQwDQYJKoZIhvcNAQEFBQAwGzEZMBcGA1
UEAxMQaWRwLm1hY2hpbmUudGVzdDAeFw0xMDA4MTMwNzQwMDdaFw0zMDA4MTMwNzQwMDdaMBsxGTAX
BgNVBAMTEGlkcC5tYWNoaW5lLnRlc3QwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCZ/o
gfLbgPdgAKCtB5StZD/GNzIxm2ahTF+/3XeT1K4NHvZsalHvjr/RJNPsLPNN/blcjaoRnAC+2COAbL
r3OKc9lJdrVuzNARMs4UFXob7ePavuHVBNCKmkkwwdPSH/p4zVEr/hsNFBfxmSwJB9WWXoUWdhJ3bs
BaMDVtfx08JO4bmvHsKKWBMP6Zg/RSMs6lspCYwXuK97V14qmRAd6CsSCQjirL0sxI3jxDRmvD5i8K
LSGpm2dj1YQz3GSHhPlQjjanS7dRkGv6PAkxFt3keP9DFkVAdiXZY+Sl3WnmP7sKMC3l2rJuW+WPcq
oARsNMB1muoYYuuixLBQ24egQ/AgMBAAGjZzBlMEQGA1UdEQQ9MDuCEGlkcC5tYWNoaW5lLnRlc3SG
J2h0dHBzOi8vaWRwLm1hY2hpbmUudGVzdC9pZHAvc2hpYmJvbGV0aDAdBgNVHQ4EFgQUy2CpEPhIUd
7J8LBPyM2P3pPIFyowDQYJKoZIhvcNAQEFBQADggEBAHI0smZ/nLUHYbu++rD/QNkXm4zFSVJZ5gwO
5G8aGBriQDmVcfOO7SnmYKuWxTXcgLulNZQt95kiMzWu34kyizNNymCqoyM0dMcyrpxRhO7CvEVeWc
84FJtSFYTD0OFRh4OwydSmfLftODmzM65fEXUuV5IfzWz7LovlpEdkWp0P0oey6cHK8sBqH96uNSE3
ghKrtvjc8TlQ/L3NA1y9zN8qYNDms1ADHXVoYopY5Mc4gaWyZVcuh90TKH/5vE5T2Sq7oi5zJvuROI
xwQXYjUA8dpaKVVM6Pgkhw6jLlKzfpLqt60j/HnbmBEm0KBXjbJin7Pz95g7g+2LIfcBJt+X8=
                    </ds:X509Certificate>
                </ds:X509Data>
            </ds:KeyInfo>
        </KeyDescriptor>
        <ArtifactResolutionService Binding="urn:oasis:names:tc:SAML:1.0:bindings:SOAP-
            binding"
                                   Location="https://idp.machine.test:8443/idp/profile/
                                       SAML1/SOAP/ArtifactResolution"
                                   index="1"/>
        <ArtifactResolutionService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
                                   Location="https://idp.machine.test:8443/idp/profile/
                                       SAML2/SOAP/ArtifactResolution"
                                   index="2"/>
        <NameIDFormat>urn:mace:shibboleth:1.0:nameIdentifier</NameIDFormat>
        <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</NameIDFormat
            >
        <SingleSignOnService Binding="urn:mace:shibboleth:1.0:profiles:AuthnRequest"
                             Location="https://idp.machine.test/idp/profile/Shibboleth/
                                 SSO" />
        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
                             Location="https://idp.machine.test/idp/profile/SAML2/POST/
                                 SSO" />
        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST-
            SimpleSign"
                             Location="https://idp.machine.test/idp/profile/SAML2/POST-
                                 SimpleSign/SSO" />
        <SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
            Redirect"
                             Location="https://idp.machine.test/idp/profile/SAML2/
                                 Redirect/SSO" />
    </IDPSSODescriptor>
    <AttributeAuthorityDescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:1
        .1:protocol␣urn:oasis:names:tc:SAML:2.0:protocol">
        <Extensions>
            <shibmd:Scope regexp="false">machine.test</shibmd:Scope>
        </Extensions>
        <KeyDescriptor>
            <ds:KeyInfo>
                <ds:X509Data>
                    <ds:X509Certificate>
MIIDKzCCAhOgAwIBAgIUSh38ymBUKBZ3JpI6AY7oX9cZQbQwDQYJKoZIhvcNAQEFBQAwGzEZMBcGA1
UEAxMQaWRwLm1hY2hpbmUudGVzdDAeFw0xMDA4MTMwNzQwMDdaFw0zMDA4MTMwNzQwMDdaMBsxGTAX
BgNVBAMTEGlkcC5tYWNoaW5lLnRlc3QwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCZ/o
gfLbgPdgAKCtB5StZD/GNzIxm2ahTF+/3XeT1K4NHvZsalHvjr/RJNPsLPNN/blcjaoRnAC+2COAbL
r3OKc9lJdrVuzNARMs4UFXob7ePavuHVBNCKmkkwwdPSH/p4zVEr/hsNFBfxmSwJB9WWXoUWdhJ3bs
BaMDVtfx08JO4bmvHsKKWBMP6Zg/RSMs6lspCYwXuK97V14qmRAd6CsSCQjirL0sxI3jxDRmvD5i8K
```

```
63 LSGpm2dj1YQz3GSHhPlQjjanS7dRkGv6PAkxFt3keP9DFkVAdiXZY+Sl3WnmP7sKMC3l2rJuW+WPcq
64 oARsNMB1muoYYuuixLBQ24egQ/AgMBAAGjZzBlMEQGA1UdEQQ9MDuCEGlkcC5tYWNoaW5lLnRlc3SG
65 J2h0dHBzOi8vaWRwLm1hY2hpbmUudGVzdC9pZHAvc2hpYmJvbGV0aDAdBgNVHQ4EFgQUy2CpEPhIUd
66 7J8LBPyM2P3pPIFyowDQYJKoZIhvcNAQEFBQADggEBAHI0smZ/nLUHYbu++rD/QNkXm4zFSVJZ5gwO
67 5G8aGBriQDmVcfOO7SnmYKuWxTXcgLulNZQt95kiMzWu34kyizNNymCqoyM0dMcyrpxRhO7CvEVeWc
68 84FJtSFYTD0OFRh4OwydSmfLftODmzM65fEXUuV5IfzWz7LovlpEdkWp0P0oey6cHK8sBqH96uNSE3
69 ghKrtvjc8TlQ/L3NA1y9zN8qYNDms1ADHXVoYopY5Mc4gaWyZVcuh90TKH/5vE5T2Sq7oi5zJvuROI
70 xwQXYjUA8dpaKVVM6Pgkhw6jLlKzfpLqt60j/HnbmBEm0KBXjbJin7Pz95g7g+2LIfcBJt+X8=
71                         </ds:X509Certificate>
72                     </ds:X509Data>
73                 </ds:KeyInfo>
74             </KeyDescriptor>
75             <AttributeService Binding="urn:oasis:names:tc:SAML:1.0:bindings:SOAP-binding"
76                               Location="https://idp.machine.test:8443/idp/profile/SAML1/
                                      SOAP/AttributeQuery" />
77             <AttributeService Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
78                               Location="https://idp.machine.test:8443/idp/profile/SAML2/
                                      SOAP/AttributeQuery" />
79             <NameIDFormat>urn:mace:shibboleth:1.0:nameIdentifier</NameIDFormat>
80             <NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</NameIDFormat
                   >
81         </AttributeAuthorityDescriptor>
82 </EntityDescriptor>
```

Das Wurzelelement eines Metadaten Dokuments muss, `EntityDescriptor` bzw. `EntitiesDescriptor`(bei mehreren Entitäten), heißen und der Namespace `urn:oasis:names:tc:SAML:2.0:metadata` ist zu deklarieren. Nun werden einige Daten des Dokuments genauer betrachtet.

- Zeile 6: Es werden Protokolle definiert, die vom IDP unterstützt werden. In dem Beispiel handelt es sich dabei um SAML 1.0, SAML 2.0 und Shibboleth 1.0.
- Zeile 10: Ein `KeyDescriptor`-Element enthält Schlüsselinformationen. Zum Beispiel ein X509 Zertifikat.
- Zeile 38-39: Hier werden NameID-Formate festgelegt, die vom IDP unterstützt werden.
- Zeile 44+46: Für den SSO werden hier die Bindings definiert und die URL, wo man den SSO Service für das entsprechende Binding aufrufen kann.

Nach bisherigen Recherchen bieten weder Google noch WSO2 eine Funktion zum Laden von Metadaten an.
Eine Datei mit Metadaten für den IDP ist, in dem installierten WSO2 Identity Server Paket, nicht vorhanden. Es gibt anscheinend nur die Möglichkeit diese Konfiguration manuell über das Web-Interface vorzunehmen.

5 Maßnahmen zur Sicherung der Kommunikation

Bei der Übertragung der SAML Anfragen und Antworten ist es wichtig, dass die Integrität, Authentizität und Vertraulichkeit der Daten gesichert ist. Das SSO ist eine sehr bequeme Funktionalität für den Benutzer, sie birgt aber auch Gefahren. Ein Identity Provider, der sämtliche Benutzer Login-Daten verwaltet, stellt einen „Single-Point of Failure“ dar. Falls es einem Angreifer gelingt an die Benutzerdaten zu kommen, hat er Zugang zu sämtlichen Diensten, die durch die Service Provider angeboten werden. Dieses Kapitel befasst sich deshalb mit den Mechanismen, die das SAML2.0 zur Sicherung der Kommunikation anbietet und die Umsetzung dieser in den einzelnen Frameworks.

5.1 SAML 2.0 Sicherheit

Für die Kommunikation via HTTP zwischen zwei Parteien empfiehlt sich der Einsatz von SSL 3.0 oder TLS 1.0, so werden Datenpakete auf Netzwerkebene geschützt. In diesem Fall erfolgt die Authentifizierung des Servers mit Hilfe eines X509 v3 Zertifikats. Diese Anforderung muss bei der Installation des Web Servers beachtet werden und hat im Allgemeinen nichts mit dem SAML 2.0 Standard zu tun.

5.1.1 XML-Signatur und XML-Verschlüsselung

Um einen zusätzlichen Schutz der ausgetauschten Nachrichten zu ermöglichen können diese signiert werden. Dabei ist es wichtig, dass der RSA Algorithmus, von den an der Kommunikation teilnehmenden Parteien, unterstützt wird. Für das Signieren von XML-Dokumenten gibt es drei verschiedene Methoden: enveloped, enveloping und ditached (vgl. [BRS+08]). Im folgenden werden die Unterschiede der einzelnen Methoden herausgestellt und in Abbildung 5.1 veranschaulicht.

- *enveloped*: Bei dieser Methode wird das gesamte Dokument signiert. Als Referenz wird beim Signieren die Wurzel des Dokuments gewählt.
- *enveloping*: Hier wird ein bestimmtes referenziertes Objekt im Dokument signiert. Ein eindeutiger Bezeichner muss für dieses Objekt existieren.
- *detached*: Das Objekt, dass signiert wird, befindet sich außerhalb des „Signature“-Elements.

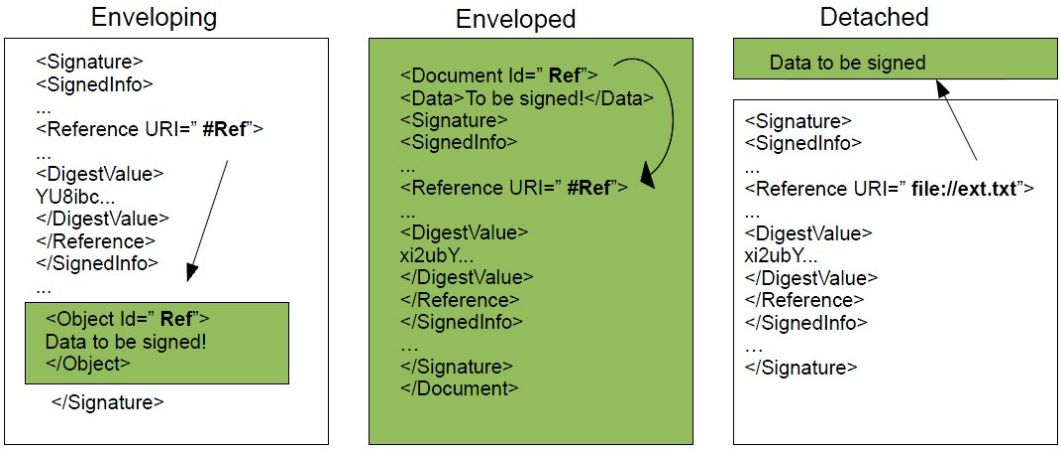

Abbildung 5.1: Signatur-Methoden[1]

Im SAML2.0 ist festgelegt, dass die enveloped-Methode verwendet werden soll. Außerdem muss in der Wurzel, des zu signierenden Teils, das ID-Attribut gesetzt sein (vgl. [OAS05a] S.69 ff.). Ist das ID-Attribut nicht gesetzt, so wird bei der enveloped-Methode automatisch das Wurzel-Element des XML-Dokuments für die Signaturberechnung ausgewählt. Dies muss aber nicht immer das richtige Element sein, falls man zum Beispiel eine Assertion signieren will, die in einer SOAP-Nachricht eingebettet ist, wird fälschlicherweise das SOAP-Element als Wurzel mit signiert. Bei der Berechnung des Signaturwerts mit der enveloped-Methode muss man darauf achten, dass man den eigenen `SignatureValue` nicht in die Berechnung einbezieht. Dies führt zu Fehlern bei der Verifizierung.
Eine weitere Vorgabe durch den SAML2.0 Standard ist die Verwendung der „exclusive Canonicalization“. Durch die Canonicalization wird das Dokument in eine Standardform gebracht, denn es muss sichergestellt werden, dass sowohl die Signierung als auch die Verfikation auf exakt denselben Daten stattfindet.
Seit SAML 2.0 wird die XML-Verschlüsselung (vgl. [IDS02]) unterstützt. Es ist nun möglich komplette Assertions oder einzelne Attribute innerhalb der Assertions zu verschlüsseln und so die Vertraulichkeit der Daten auf Nachrichtenebene sicher zu stellen

5.1.2 Weitere Sicherheitsmechanismen

Ein weiterer Aspekt, der nicht außer Acht gelassen werden sollte, ist die Gültigkeit der Nachrichten. Wenn die Gültigkeits-Zeitspanne klein genug gewählt ist, werden so genannte „Replay-Attacken“ erschwert, die es einem Angreifer ermöglichen mit geschnittene Nachrichten aus vergangenen Kommunikationen einfach wieder zu versenden.
Bei SAML 2.0 gibt es Maßnahmen um sich vor solchen Angriffen zu schützen. Die Nachrichten sollten immer das Attribut `IssueInstant` mit einem aktuellen Timestamp enthalten. Weiterhin können Bedingungen (Conditions) hinzugefügt werden, wo eine Zeitspanne für die Gültig-

[1] http://www.nds.rub.de/media/nds/downloads/ws0910/2.3_XML_XMLSignature_v09.pdf

keit definiert wird. Eine eindeutige ID, die bei jeder Nachricht vergeben wird, sorgt dafür dass ein wiederholtes Versenden derselben Nachricht erkannt wird (vgl. [OAS05f] S.18). Dies ist auch wichtig um den Server vor DoS Attacken zu schützen. Dabei ist das Ziel des Angreifers einem Server so oft Anfragen zu schicken bis diese nicht mehr verarbeitet werden können und der Server keine Anfragen mehr entgegen nehmen kann bzw. abstürzt. So ein Szenario ist beim Umgang mit XML-Dokumenten durchaus möglich. Das Parsen kann sehr aufwendig und Speicher- und Ressourcenlastig sein. Wurde hier bei der Implementierung nicht aufgepasst, kann dies schwerwiegende Folgen haben.

5.2 Google SSO Sicherheit

Das Google Framework verzichtet auf die Signierung der Authentifizierungsanfrage. Diese wird lediglich komprimiert und base64 kodiert. Man kann also durch mithören der HTTP Nachrichten die Anfrage abfangen und dekodieren, was noch kein Sicherheitsrisiko darstellt, da hier noch keine sensible Daten übertragen werden. Die anschließende Antwort vom Identity Provider muss aber vor Manipulationen geschützt werden, da hier die NameID des Nutzers übertragen wird und dem Service Provider signalisiert wird, dass die Anmeldung erfolgreich war. Um diese Nachricht zu schützen verwendet Google die XML-Signatur mit der enveloped-Methode. Es werden zwei Algorithmen unterstützt, RSA und DSA (vgl. [Goo10a]). Eine Verschlüsselung, des XML-Dokuments bzw. einzelner Attribute, findet nicht statt.

5.3 Shibboleth 2.0 Sicherheit

In der Shibboleth 2.0 Dokumentation wird die Sicherung via SSL/TLS empfohlen. Dies entspricht auch den Vorgaben von SAML 2.0. Allerdings ist der Nutzer selbst dafür verantwortlich den Web Server richtig abzusichern.
Um Ressourcen zu sparen kann man die Signatur bzw. Verschlüsselung von Nachrichten an- bzw. abschalten (vgl. [Shi10b]). Bei folgenden Nachrichten lässt sich das Signieren über ein Parameter festlegen:

- Antwortnachrichten
- Assertions
- Anfragenachrichten

Die Verschlüsselung kann für die NameIDs und für die gesamte Assertion konfiguriert werden. Das Verschlüsseln von NameIDs macht insofern Sinn, als dass dort wichtige Identifikationsmerkmale über den Benutzer, der sich authentifiziert hat, enthalten sind (z.B. E-Mail Adresse).
Bei der Verwendung von SSL/TLS, XML-Signaturen und XML-Verschlüsselung, ist die Frage nach den Key Management sehr wichtig. Es gibt diesbezüglich keine Vorgaben im SAML Standard, weshalb Shibboleth hier ein eigenes Konzept entwickelt hat, die „Explicit Key Trust Engine“ (vgl. [Shi10a] und [Shi10c]). Das Konzept basiert auf den schon im vorherigen Kapitel behandelten Metadaten, wo in `md:KeyDescriptor` Elementen Schlüssel definiert sind.

Diese Schlüssel werden beim verifizieren einer XML-Signatur oder beim überprüfen eines TLS Zertifikats ausgewertet. Der Vorteil dabei ist, dass diese Daten zentral verwaltet und gesichert werden können. Ein Systemadministrator muss dafür sorgen, dass die Metadaten über einen sicheren Kanal geladen (SSL/TLS) und geschützt abgelegt werden.

5.4 WSO2 Sicherheit

Der WSO2 Identity Server ist nach der Installation stadardmäßig mit TLS gesichert, was dem Benutzer aufwendige Konfigurationsschritte erspart. Ein gültiges Zertifikat ist im Installationspaket auch vorhanden.
Eine Möglichkeit XML-Verschlüsselung von Assertions im WSO2 Identity Server zu aktivieren, wird nicht zur Verfügung gestellt. Das Signieren von Nachrichten ist dagegen standardmäßig aktiviert. Bei einem Test eines SSO bei Google Apps wurde der generierte SAML Response signiert und auch die in der SAML 2.0 Spezifikation beschriebenen Maßnahmen gegen Replay Attacken wurden umgesetzt. Zeitspanne für die Gültigkeit einer Assertion ist standardmäßig auf 5 Minuten gesetzt.

6 Fazit

Die einzelnen vorgestellten Frameworks haben alle ihre Vorzüge und eignen sich für bestimmte Zielgruppen. Das Google SSO Framework ist sehr einfach gehalten und bietet einen guten Einstieg in das Thema SSO. Ein Entwickler, der nicht vertraut ist mit dem Ablauf des Protokolls kann hier schnell und unkompliziert eine lokale Anwendung aufsetzen, die den SSO simuliert. Darauf basierend kann er nun eigene Lösungen implementieren. Grundlagen bzgl. SAML und XML werden hier aber schon vorausgesetzt.
Wenn man mit dem Thema Web SSO bereits vertraut ist und eine komplette und sichere Lösung sucht, die dazu annähernd alle neuen Erweiterungen des SAML 2.0 Standards unterstützt, ist Shibboleth 2.0 zu bevorzugen. Der Einstieg ist hier relativ schwierig, aber es existieren viele Test-Seiten und Tutorials um mit dem System vertraut zu werden. Daneben ist bei Shibboleth, sowohl ein Identity als auch ein Service Provider verfügbar, so dass ein kompletter Web SSO use case aufgesetzt werden kann.
WSO2 hält mit der „Cloud Identity“ Anschluss an einen neuen Trend. Es bietet Dienste in einer Cloud an. Das Einrichten einer Identitätsverwaltung auf einem eigenen System und die Administration und Wartung entfällt dabei. Man nutzt die Infrastruktur und Rechenkapazität von WSO2 und hat über ein Web Interface Zugriff auf ein Identitäts-Management System. Hier ist der Einstieg wohl am einfachsten. Man muss aber auch darauf vertrauen, dass WSO2 die zur Verfügung gestellten Daten nicht für andere Zwecke missbraucht.

Literaturverzeichnis

[BRS+08] BARTEL, Mark ; REAGLE, Joseph ; SOLO, David ; HIRSCH, Frederick ; ROESSLER, Thomas: *XML Signature Syntax and Processing (Second Edition)*. Version: 2008. http://www.w3.org/TR/xmldsig-core/

[Deu96] DEUTSCH, Peter: *DEFLATE Compressed Data Format Specification version 1.3*. Version: 1996. http://www.w3.org/Graphics/PNG/RFC-1951

[Goo10a] GOOGLE: *Frequently Asked Questions*. Version: 2010. http://code.google.com/intl/de-DE/googleapps/faq.html

[Goo10b] GOOGLE: *SAML-based Single Sign-On for Google Hosted Services - Demo Tool*. Version: 2010. http://code.google.com/intl/de-DE/apis/apps/sso/saml_static_demo/saml_demo.html

[Goo10c] GOOGLE: *SAML Single Sign-On (SSO) Service for Google Apps*. Version: 2010. http://code.google.com/intl/de-DE/googleapps/domain/sso/saml_reference_implementation.html

[Goo10d] GOOGLE: *Web-based Reference Implementation of SAML-based SSO for Google Apps*. Version: 2010. http://code.google.com/intl/de-DE/googleapps/domain/sso/saml_reference_implementation_web.html

[IDS02] IMAMURA, Takeshi ; DILLAWAY, Blair ; SIMON, Ed: *XML Encryption Syntax and Processing*. Version: 2002. http://www.w3.org/TR/xmlenc-core/

[Mah09] MAHESH, Thilina: *How to integrate Google Apps with WSO2 Cloud Identity ?* Version: 2009. http://wso2.org/library/articles/integrate-google-apps-wso2-cloud-identity

[OAS05a] OASIS: *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*. Version: 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[OAS05b] OASIS: *Bindings for the OASIS Security Assertion Markup Language(SAML)V2.0*. Version: 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf

[OAS05c] OASIS: *Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0*. Version: 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf

[OAS05d] OASIS: *SAML V2.0 Executive Overview*. Version: 2005. http://www.oasis-open.org/committees/download.php/13525/sstc-saml-exec-overview-2.0-cd-01-2col.pdf

[OAS05e] OASIS: *SAML V2.0 Technical Overview*. Version: 2005. http://www.oasis-open.org/committees/download.php/11511/sstc-saml-tech-overview-2.0-draft-03.pdf

[OAS05f] OASIS: *Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0*. Version: 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf

[OAS07] OASIS: *Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0*. Version: 2007. http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf

[Shi09a] SHIBBOLETH INTERNET2: *IdPAuthUserPass*. Version: 2009. https://spaces.internet2.edu/display/SHIB2/IdPAuthUserPass

[Shi09b] SHIBBOLETH INTERNET2: *IdPUserAuthn*. Version: 2009. https://spaces.internet2.edu/display/SHIB2/IdPUserAuthn

[Shi09c] SHIBBOLETH INTERNET2: *Metadata*. Version: 2009. https://spaces.internet2.edu/display/SHIB2/Metadata

[Shi10a] SHIBBOLETH INTERNET2: *ExplicitKeyTrustEngine*. Version: 2010. https://spaces.internet2.edu/display/SHIB2/ExplicitKeyTrustEngine

[Shi10b] SHIBBOLETH INTERNET2: *IdPXMLSigEnc*. Version: 2010. https://spaces.internet2.edu/display/SHIB2/IdPXMLSigEnc

[Shi10c] SHIBBOLETH INTERNET2: *TrustManagement*. Version: 2010. https://spaces.internet2.edu/display/SHIB2/TrustManagement

[WSO10a] WSO2: *WSO2 debuts Cloud Identity to simplify Identity Management and Authentication in the Cloud*. Version: 2010. http://wso2.com/about/news/wso2-debuts-cloud-identity-to-simplify-identity-management-and-authentication-in-the-cloud/

[WSO10b] WSO2: *WSO2 Identity Server*. Version: 2010. http://wso2.com/products/identity-server/

[Zei99] ZEIGER, Stefan: *Servlet Essentials*. Version: 1999. http://www.novocode.com/doc/servlet-essentials/