Konfiguration X509-Authentifizierung mit Shibboleth-IdP V3.X

Erstellt für

DFN-CERT Services GmbH

Version 1.0

Hauptverantwortlicher:

David Hübner

Dateiname: Anleitung-X509-Authn-ShibIdP-v1.0.odt

Erstellt am: 24.04.2018

Letzte Änderung: 07.05.2018

Versionskontrolle:

Version	Datum	Autor	durchgeführte Änderungen
0.01	12.02.18	David Hübner	Grobfassung Anleitung
0.02	24.04.18	David Hübner, Jennifer Vosseler	Ergänzungen zu anderen Systemkomponenten, Überarbeitung, Endredaktion
1.0.	07.05.18	Karin Gietz	Tippfehler korrigiert und 1.0 erstellt

Finale Abnahme:

Version	Datum	abgenommen durch	Rolle / Position
1.00	07.05.18	Jürgen Brauckmann	Projektleiter Kunde

Inhaltsverzeichnis

1 Einleitung

1.1 Zielsetzung

Ziel dieser Dokumentation ist die Evaluation und Beschreibung verschiedener Szenarien, wie eine Authentifizierung mit X509-Client-Zertifikaten in Shibboleth umgesetzt werden kann.

1.2 Zielgruppe

Das Dokument setzt fortgeschrittene Kenntnisse in der Konfiguration von Apache Webserver sowie der Shibboleth-Identity-Provider(IdP)-Software voraus. Zielgruppe sind in erster Linie Administratoren, die den IdP so konfigurieren möchten, dass sich die Nutzer mit Client-Zertifikaten authentifizieren können.

1.3 Überblick

In Kapitel 2 folgt zunächst ein Überblick über den in Shibboleth integrierten X509-LoginHandler, sowie über die generelle Architektur der darüber stattfindenden Authentifizierung. In Kapitel 3 folgt schließlich eine detaillierte Anleitung, wie die einzelnen Systemkomponenten konfiguriert werden können, um die Authentifizierung über X509-Client-Zertifikate einzurichten. Hierfür wird der häufig eingesetzte Software-Stack bestehend aus Shibboleth- IdP, Tomcat als Servlet-Container und Apache 2 als Webserver, der als Reverse-Proxy fungiert, verwendet. Die Anleitung zeigt die notwendigen Schritte anhand von Beispielzertifikaten auf. Auf Anpassungs- oder Erweiterungsmöglichkeiten für eine abweichende Umgebung wird eingegangen.

2 X509-Authentifizierung in Shibboleth

Ein Login-Vorgang am Shibboleth-IdP besteht im Allgemeinen aus verschiedenen *Flows*. Nachdem eine Anfrage von einem Service Provider (SP) empfangen wurde, geht der IdP in einen der verfügbaren Flows zur Authentifizierung über. Dabei handelt es sich beispielsweise um den Password-Flow authn/Password, wobei sich der Nutzer durch die Eingabe von Nutzername und Passwort authentifizieren kann. Diese Daten werden dann gegen eine angeschlossene Datenquelle (häufig LDAP-Server) geprüft. Am Ende dieses Flows steht die *Subject-Canonicalization*, die das Erzeugen eines Benutzernamens (auch *Principal* genannt) für den Nutzer, der sich gerade erfolgreich authentifiziert hat, beschreibt. Unter diesem Benutzernamen ist der Nutzer im weiteren Verlauf schließlich im IdP bekannt. Normalweise (und vor allem bei Verwendung des Password-Flows) entspricht dieser Name schlichtweg dem Nutzernamen.

Neben dem authn/Password-Flow stellt der Shibboleth-IdP auch den sogenannten authn/X509-Flow zur Verfügung, welcher die Verwendung von X509-Client-Zertifikaten als Authentifizierungsmethode erlaubt. Die Logik auf Seiten des IdP ist dabei vergleichsweise simpel, da schlichtweg die Umgebungsvariable „javax.servlet.request.X509Certificate“ im

Servlet-Container verwendet wird, um zu prüfen, ob die Authentifizierung erfolgreich ist oder nicht. Wie diese Variable befüllt wird, spielt für den IdP dabei keine Rolle [Shib-X509AuthnConfiguration].

Jeder Login-Flow wird über einen dedizierten Endpunkt am IdP zur Verfügung gestellt. Bei authn/X509 handelt es sich dabei um den Pfad /Authn/X509. Ein naheliegender Weg besteht nun darin, diesen Endpunkt über den Apache-Webserver zu schützen. Sobald der IdP dann in seinem Ablauf in den authn/X509-Flow springt (wodurch der User auf den geschützten Pfad auf dem Webserver weitergeleitet wird), greift die Zugriffskontrolle von Apache. Da in diesem Fall als Authentifizierungsmethode X509-Zertifikate eingesetzt werden sollen, kann hierfür das Apache-Modul „mod_ssl“ entsprechend konfiguriert [Apache-mod-ssl] und das Client-Zertifikat validiert werden. Ist diese Validierung erfolgreich, kann der Nutzer den Endpunkt des IdP aufrufen. Dort ist „javax.servlet.request.X509Certificate“ nun korrekt befüllt und stellt Informationen über das verwendete Zertifikat bereit. Wenn die Zugriffskontrolle von Apache den Nutzer nicht durchlässt (also die Validierung des Zertifikats nicht erfolgreich war), kommt der Nutzer gar nicht erst zum X509-Endpunkt.

Im Gegensatz zum authn/Password-Flow, wo die *Subject-Canonicalization* im Allgemeinen sehr simpel ist, muss beim authn/X509-Flow unter Umständen mehr konfiguriert werden. Normalerweise lässt sich der zu verwendende Name aber aus dem *Subject-DN* des Client-Zertifikats extrahieren.

3 Anleitung mit Apache, Tomcat und Shibboleth

In diesem Kapitel soll nun die beschriebene Kombination von Apache-Webserver (Apache 2), Tomcat Java-Servlet-Container (Tomcat 7 bzw. 8) und Shibboleth IdP V3.X beschrieben werden. In der Anleitung wird davon ausgegangen, dass die prinzipielle Infrastruktur bereits aufgebaut und lauffähig ist.

In diesem Beispiel werden zwei Client-Zertifikate verwendet:

Zertifikat 1: emailAddress=mustermann@dfn-cert.de, CN=Erika Mustermann, OU=Test, O=Organisation 1, C=DE

Zertifikat 2: emailAddress=mustermann@dfn-cert.de, CN=Erika Mustermann, OU=Test, O=Organisation 2, C=DE

Die Zertifikate sind nun durch folgende Zertifikat-Hierarchie ausgestellt:

C=DE,O=Test,CN=Test Eins CA

C=DE,O=Test,CN=Test Intermediate

C=DE,O=Test,CN=Test Root

Bei „CN=Test Root“ handelt es sich in diesem Beispiel um die Root CA.

Die beiden Zertifikate unterscheiden sich also nur durch die Zugehörigkeit zu Organisation 1 bzw. Organisation 2.

Beide Zertifikate wurden als Client-Zertifikate in den Browser importiert.

3.1 Konfiguration der Authentifizierung in Shibboleth

Die Konfiguration des X509-Flows zur Authentifizierung im Shibboleth-IdP erfordert lediglich eine Anpassung in der Konfigurationsdatei {idp.home}/conf/idp.properties. Hier lässt sich der Login-Flow mit `idp.authn.flows = X509` aktivieren.

Weiterhin werden laut [Shib-X509AuthnConfiguration] noch drei optionale Konfigurationsmöglichkeiten des Flows in {idp.home}/conf/authn/x509-authn-config.xml angeboten. In der Praxis dürfte in den meisten Fällen jedoch nur einer davon relevant sein:

`shibboleth.authn.X509.externalAuthnPath` erlaubt das Festlegen des Pfads, auf den weitergeleitet werden soll, sobald der Login-Flow aufgerufen wird. Standardmäßig wird der Nutzer dabei auf eine vorgeschaltete Seite weitergeleitet (siehe Abbildung 1) und muss explizit bestätigen, dass er nun einen Login per X509-Zertifikat wünscht (was dann zur Weiterleitung auf den Endpunkt /Authn/X509 führt). Wenn dieses Verhalten nicht gewünscht ist, kann dieser Parameter direkt auf den Endpunkt gesetzt werden.

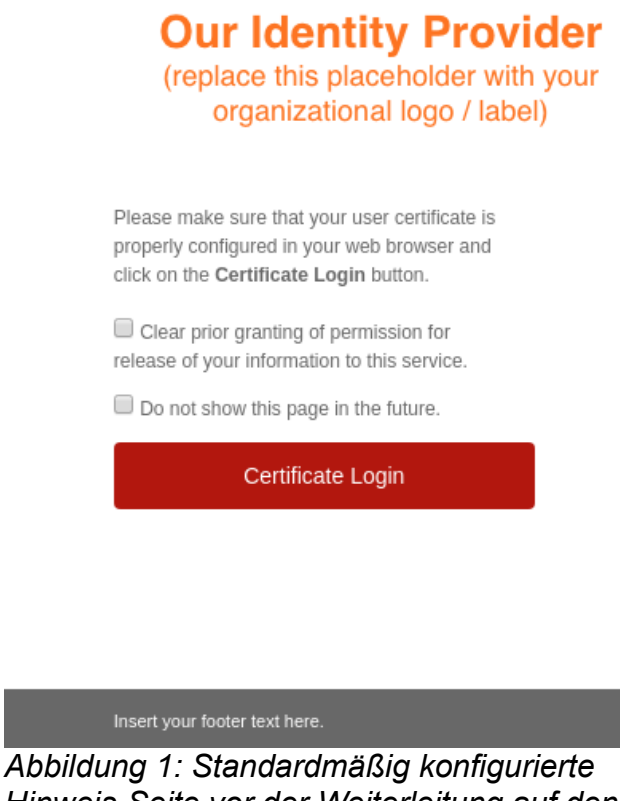

Abbildung 1: Standardmäßig konfigurierte Hinweis-Seite vor der Weiterleitung auf den geschützten Endpunkt

3.2 Validierung der Zertifikate in Apache

Nachdem nun im IdP der authn/X509-Flow aktiviert und ggf. konfiguriert wurde, muss Apache so konfiguriert werden, dass der IdP-Endpunkt /Authn/X509 nur nach erfolgreicher Prüfung des Client-Zertifikats aufgerufen werden darf.

3.2.1 Einbinden der CA zur Validierung des Zertifikats

Zunächst muss das Zertifikat zur Validierung des Root-Zertifikats in der Hierarchie angegeben werden:

```
SSLCACertificateFile /etc/pki/tls/certs/dfn-test-root.crt
```

Zu beachten ist, dass dies nicht innerhalb einer Location-Angabe in Apache erfolgen darf, da insbesondere Apache in der Version 2.4 und mit aktueller OpenSSL-Version damit nicht zurechtkommt. Stattdessen muss diese Angabe direkt im Virtual-Host gesetzt werden.

3.2.2 Schutz des X509-Endpunkts des IdP

Der eigentliche Schutz des authn/X509-Endpunkts am IdP erfolgt dann über den Schutz der entsprechenden Location auch innerhalb der Virtual-Host-Definition, etwa wie folgt:

```
<Location /idp/Authn/X509>
    SSLVerifyClient require
    SSLVerifyDepth 3
    SSLOptions +StdEnvVars +ExportCertData
    Require expr (%{SSL_CLIENT_S_DN_O} == 'Organisation 1')
</Location>
```

Während hier prinzipiell die in [Apache-mod-ssl] aufgeführten Parameter gesetzt werden können, sind insbesondere die folgenden Direktiven interessant:

`SSLVerifyClient` ermöglicht erst die Authentifizierung per Client-Zertifikat. Zudem macht im hier betrachteten Anwendungsfall auch nur `require` Sinn, da nur damit sichergestellt wird, dass der Client ein valides Zertifikat präsentieren muss.

`SSLVerifyDepth` drückt die maximale Anzahl an Intermediate-CAs aus, die von Apache verfolgt werden, bis das Zertifikat abgewiesen wird. Im vorliegenden Beispiel ist hier der Wert 3 ausreichend. In anderen Szenarien muss ggf. angepasst werden.

`SSLOptions` ermöglicht das Festlegen von weiteren Parametern, insbesondere welche Daten aus dem Zertifikat als Umgebungsvariablen zur Verfügung gestellt werden.

`Require` erlaubt die Konfiguration von Bedingungen, die erfüllt sein müssen, damit der Zugriff gestattet wird und ermöglicht damit die Einschränkung der Authentifizierung auf bestimmte Nutzergruppen. In Kapitel 3.2.3 wird diese Direktive genauer betrachtet. `Require expr` ersetzt im Allgemeinen die veraltete `SSLRequire`-Direktive.

3.2.3 Konfigurationsmöglichkeiten Require

Die Verwendung von Require ist unter [Apache-Require] vollständig dokumentiert. Weitere Informationen zur Syntax der alten Direktive SSLRequire lassen sich unter [Apache-SSLRequire] finden. Die Syntax ist mit einigen Ausnahmen identisch.

In dem Beispiel aus Kapitel 3.2.2 wird beispielsweise die Einschränkung `(%{SSL_CLIENT_S_DN_O} == 'Organisation 1')` verwendet. Dabei wird mit `%{SSL_CLIENT_S_DN_O}` zunächst auf eine Umgebungsvariable zugegriffen (in diesem

Fall, der O-Komponente des Subject-DN des Client-Zertifikats) welche dann auf Übereinstimmung mit einem bestimmten String verglichen wird. Eine Übersicht über die nutzbaren Umgebungsvariablen findet sich unter [Apache-SSL-Envvars]. In diesem Beispiel würde also Zertifikat 1 (O=Organisation 1) Zugriff erhalten, während Zertifikat 2 (O=Organisation 2) nicht akzeptiert werden würde.

Diese Einschränkungen können nun nach Belieben kombiniert werden, etwa wie im folgenden Beispiel:

```
Require expr ( (%{SSL_CLIENT_S_DN_O} == 'Organisation 1')  \
             &&  (%{SSL_CLIENT_I_DN_CN} == 'Test Zwei CA') )
```

Hier wird mit `%{SSL_CLIENT_I_DN_CN}` der CN des Issuer des Client-Zertifikats überprüft. Da beide Zertifikate von „Test Eins CA“ ausgestellt wurden, werden beide Zertifikate nicht akzeptiert.

3.3 Weiterverarbeitung des Ergebnisses in Shibboleth

Nachdem nun in Apache die Validierung des Client-Zertifikats vorgenommen und dieses akzeptiert wurde, kann der User auf den authn/x509-Endpunkt am IdP zugreifen. Dort steht dem IdP über die Umgebungsvariable „javax.servlet.request.X509Certificate“ der Inhalt des Client-Zertifikats zur Verfügung.

Im Folgenden werden einige Punkte beschrieben, die nun üblicherweise am IdP nach der eigentlichen Authentifizierung zu beachten sind:

3.3.1 Extrahieren des „Principal“ aus dem Zertifikat

Zunächst muss über die *Subject-Canonicalization* ein Name für den Nutzer erstellt werden. Dies kann in {idp.home}/conf/c14n/x500-subject-c14n-config.xml konfiguriert werden. Hierbei stehen standardmäßig zwei Möglichkeiten zur Verfügung:

Zunächst kann über eine Prioritätenliste auf etwaige Attribute im SubjectAltName des Zertifikats zugegriffen werden. Hier ist die Angabe von Werten zwischen 0 und 8 möglich, welche in [RFC-5280] definiert werden. 1 steht hier beispielsweise für E-Mail.

Alternativ kann über eine zweite Prioritätenliste auf die einzelnen Komponenten im Subject-DN des Client-Zertifikats zugegriffen werden, wobei in der IdP-Konfiguration die jeweiligen OIDs angegeben werden müssen. Um also beispielsweise in den vorliegenden Zertifikaten die emailAddress – und wenn diese nicht im Subject-DN ist, alternativ den CN zu verwenden – könnte die folgende Regel definiert werden:

```
<util:list id="shibboleth.c14n.x500.ObjectIDs">
    <value>1.2.840.113549.1.9.1</value>
    <value>2.5.4.3</value>
</util:list>
```

Weiterhin ist es auch möglich, einige vorgegebene Transformationen anzuwenden (z. B. den Namen in Kleinbuchstaben transformieren) oder eigene Transformationen zu formulieren.

Generell muss hier eine passende Strategie gefunden werden, mit der anhand des erzeugten Namens weitergearbeitet werden kann. Wenn also beispielsweise weitere Attribute aus einem LDAP-Server abgefragt werden sollen, muss hier bereits darauf geachtet werden, einen geeigneten Namen für den Suchfilter zu generieren.

3.3.2 Abfrage von Attributen aus einem LDAP-Server

Schließlich kann nun der in Kapitel 3.3.1 erzeugte *Principal* dazu verwendet werden, weitere Attribute aus einem LDAP-Server abzufragen. In {idp.home}/conf/ldap.properties wird ein Suchfilter für den Attribute-Resolver definiert:

```
idp.attribute.resolver.LDAP.searchFilter =
(uid=$resolutionContext.principal)
```

Hierbei bezieht sich `$resolutionContext.principal` auf den in Kapitel 3.3.1 erzeugten Namen. Dieser Filter wird dann normalerweise in den Data Connectors im Attribute-Resolver verwendet, um Informationen von einem LDAP-Server abzufragen. Die weitere Konfiguration unterscheidet sich dann nicht weiter von einer herkömmlichen Attribut-Abfrage.

Eine weitere mögliche Verwendung des Principal ist eine Attribut-Definition in {idp.home}/conf/attribute-resolver.xml, die direkt auf diesen Wert zugreift:

```
<AttributeDefinition id="uid" xsi:type="PrincipalName">
  <AttributeEncoder xsi:type="SAML1String"
      name="urn:mace:dir:attribute-def:uid" encodeType="false" />
  <AttributeEncoder xsi:type="SAML2String"
      name="urn:oid:0.9.2342.19200300.100.1.1" friendlyName="uid"
      encodeType="false" />
</AttributeDefinition>
```

Hier wird direkt der in Kapitel 3.3.1 erzeugte Principal als Eingangswert für das Attribut uid verwendet.

4 Anmerkungen zu weiteren Systemkomponenten

In diesem Kapitel sollen alternative Systemkomponenten analysiert werden, die über den in Kapitel 3 erläuterten „Standardfall“ hinaus gehen.

4.1 Verwendung von Nginx, statt Apache als Webserver

Prinzipiell bietet auch Nginx die Möglichkeit einen Zugriffsschutz mit Client-Zertifikaten zu konfigurieren. Dies ist über das SSL-Modul möglich [Nginx-ssl-verify-client]. Eine minimale

Konfiguration beinhaltet die Direktiven `ssl_client_certificate`, `ssl_verify_client` und `ssl_verify_depth`. Die einzelnen Parameter entsprechen dabei `SSLCACertificateFile`, `SSLVerifyClient` und `SSLVerifyDepth` aus der Apache-Konfiguration. Die genauen Konfigurationsmöglichkeiten lassen sich der Nginx-Dokumentation [Nginx-ssl-module] entnehmen.

Es sind allerdings zwei Einschränkungen zu berücksichtigen:

1. Bei der Konfiguration von Apache in Kapitel 3.2 wurde nur eine bestimmte `<Location>` geschützt. Dies führt dazu, dass der Webserver prinzipiell ohne Authentifizierung über die Client-Zertifikate erreichbar ist, aber die Location /idp/Authn/X509 entsprechend geschützt ist. Diese Unterscheidung ist so in Nginx nicht möglich. Stattdessen wäre es notwendig, verschiedene `Server`-Blöcke zu definieren; einmal für den IdP an sich (kein Schutz) und einmal für den konkreten Pfad zur Authentifizierung. Dies wird etwa in [Serverfault-nginx] diskutiert.
2. Es existiert keine direkte Entsprechung zu den `Require`-Direktiven in Apache, um den Zugriff feiner zu regeln. Jedoch stellt auch Nginx Informationen aus dem Client-Zertifikat als Variablen zur Verfügung [Nginx-ssl-variables], beispielsweise den Subject-DN über `$ssl_client_s_dn`. So lassen sich nun bestimmte Werte in der Nginx-Konfiguration abfangen, etwa so:
   ```
   if ($ssl_client_s_dn != "...") { return 403; }
   ```

4.2 Verwendung von Jetty, statt Tomcat als Servlet-Container

In der Anleitung aus Kapitel 3 wird vom Java-Servlet-Container lediglich die Variable „javax.servlet.request.X509Certificate“ ausgelesen. Dies ist nicht Tomcat-spezifisch und es ist davon auszugehen, dass dies genauso auch in Jetty funktioniert.

4.3 Direkte Authentifizierung in Tomcat

Der IdP lässt sich prinzipiell auch ohne vorgeschalteten Webserver (Apache oder Nginx) betreiben. Dabei übernimmt der Java-Servlet-Container direkt die Terminierung der SSL-Verbindung. Tomcat bietet dabei die bereits bekannten Möglichkeiten zur Konfiguration an [Tomcat-ssl], nämlich `caCertificateFile` zur Konfiguration der CA-Chain, `certificateVerification` zur Aktivierung der Validierung und `certificateVerificationDepth`. Diese Parameter lassen sich im `<Connector>` setzen, der in conf/server.xml definiert wird. Auch hier hat man jedoch wieder das Problem, dass sich die Client-Authentifizierung nicht auf bestimmte (Unter)-Verzeichnisse beschränken lässt.

4.4 Direkte Authentifizierung in Jetty

Auch Jetty bietet prinzipiell die Möglichkeit der Client-Authentifizierung über Zertifikate an [Jetty-ssl]. Jedoch ist es auch hier nicht ohne Weiteres möglich, den Zugriffsschutz nur für einzelne Verzeichnisse zu konfigurieren.

5 Fazit

Alle hier betrachteten Webserver und Java-Servlet-Container bieten zumindest rudimentären Support für die Validierung von Client-Zertifikaten. Diese Funktion wird letztlich von OpenSSL zur Verfügung gestellt. Für den Einsatz in Kombination mit dem Shibboleth-IdP, der auf den Inhalt von javax.servlet.request.X509Certificate angewiesen ist und die eigentliche Validierung und damit Authentifizierung an den Webserver auslagert, sind jedoch eine Reihe von weiteren Funktionen erforderlich. Zum einen ist es notwendig, dass sich der Schutz auf bestimmte Verzeichnisse einschränken lässt und nicht immer für den gesamten Server gilt. Andernfalls würde der Nutzer direkt beim Zugriff auf den IdP nach seinem Zertifikat gefragt werden. Damit wäre es dann nicht mehr sinnvoll möglich, verschiedene Login-Methoden am IdP anzubieten. Zum anderen ist es wünschenswert, die Validierung der Client-Zertifikate weiter einschränken zu können (vgl. `Require`-Direktive bei Apache in Kapitel 3.2.3).

Diese Anforderungen lassen sich mit der Kombination von Apache als Reverse-Proxy Webserver und einem Java-Servlet-Container (Jetty oder Tomcat) sehr effizient umsetzen. Gerade die sehr umfangreichen `Require`-Direktiven in Apache erlauben eine sehr detaillierte Abbildung von Anforderungen. Die Anwendung des Zugriffsschutzes auf ausgewählte Verzeichnisse ist in Apache deutlich einfacher umsetzbar, als in Nginx.

6 Quellenverzeichnis

[Shib-AuthenticationConfiguration]
https://wiki.shibboleth.net/confluence/display/IDP30/AuthenticationConfiguration

[Shib-X509AuthnConfiguration]
https://wiki.shibboleth.net/confluence/display/IDP30/X509AuthnConfiguration

[Apache-mod-ssl] https://httpd.apache.org/docs/2.4/mod/mod_ssl.html

[Apache-Require] https://httpd.apache.org/docs/2.4/mod/mod_authz_core.html#reqexpr

[Apache-SSLRequire] https://httpd.apache.org/docs/2.4/mod/mod_ssl.html#sslrequire

[Apache-SSL-Envvars] https://httpd.apache.org/docs/2.4/mod/mod_ssl.html#envvars

[RFC-5280] https://tools.ietf.org/html/rfc5280#section-4.1.2.6

[Nginx-ssl-module] http://nginx.org/en/docs/http/ngx_http_ssl_module.html

[Nginx-ssl-verify-client]
http://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_verify_client

[Nginx-ssl-variables] http://nginx.org/en/docs/http/ngx_http_ssl_module.html#variables

[Serverfault-nginx] https://serverfault.com/questions/721572

[Tomcat-ssl] https://tomcat.apache.org/tomcat-8.5-doc/config/http.html#SSL_Support

[Jetty-ssl] https://www.eclipse.org/jetty/documentation/9.4.x/jetty-ssl-distribution.html