

SAML2 Burp Plugin

«SAML Raider»

Bachelorarbeit
FS 2015
Abteilung Informatik
Hochschule für Technik Rapperswil
<http://www.hsr.ch/>

Autoren: Roland Bischofberger, Emanuel Duss
Betreuung: Cyrill Brunswiler
Arbeitsumfang: 12 ECTS bzw. 360 Arbeitsstunden pro Student
Arbeitsperiode: 16. Februar bis 12. Juni 2015

Inhaltsverzeichnis

I. Einführung	5
1. Abstract	6
2. Management Summary	7
2.1. Ausgangslage	7
2.2. Vorgehen, Technologien	7
2.3. Ergebnisse	8
2.4. Ausblick	8
3. Aufgabenstellung	9
3.1. Einführung	9
3.2. Aufgabe	9
3.3. Vorgehen	10
3.4. Randbedingungen	10
3.5. Infrastruktur	10
3.6. Erwartete Resultate	10
3.7. Termine	11
3.8. Betreuung	11
3.9. Referenzen	12
3.10. Unterschriften	12
II. Technischer Bericht	13
4. Einleitung und Übersicht	14
4.1. Problemstellung	14
4.2. Aufgabenstellung	14
4.3. Inhalt dieses Berichts	14
5. Einführung in SAML	15
5.1. OASIS	15
5.2. Was ist SAML?	15
5.3. Der SAML Standard	16
5.4. SAML Profiles	21
5.5. SAML Implementationen	28
6. SAML Security	31
6.1. X.509 Public Key Infrastructure	31
6.2. XML Security	39
6.3. Schlussfolgerung	45

7. Angriffe auf SAML	46
7.1. Angriffe auf die X.509 Public Key Infrastructure	46
7.2. Angriffe auf die XML Signaturen	49
7.3. Angriffe auf SAML	53
8. Analyse	54
8.1. Benutzercharakteristik	54
8.2. Workflow und Scope des Actors	54
8.3. Requirements	56
8.4. Use Cases	57
8.5. User Interface	63
9. Burp Suite Pluginentwicklung	66
9.1. Burp	66
9.2. Burp Extender	66
9.3. Workflow für ein simples Plugin	68
10. Architektur	69
10.1. Überlegungen und geplante Architektur	69
10.2. Implementierte Architektur	70
10.3. Vergleich zur geplanten Architektur	71
10.4. Bewertung der Architektur	72
11. Implementation	74
11.1. Vorgehensweise	74
11.2. Entwicklungsumgebung	74
11.3. Requirements	75
11.4. X.509 Verwaltung	76
11.5. SAML Message Editor	87
11.6. Testing	90
11.7. Eingesetzte Libraries	91
11.8. Build	92
11.9. Lizenz	92
11.10 Probleme	93
12. Testumgebung	95
12.1. Aufbau	95
12.2. Installation	96
12.3. Beispiel einer SAML Authentifizierung	98
13. Ergebnisse	102
13.1. Zertifikatsverwaltung	102
13.2. SAML Editor	103
14. Schlussfolgerungen	105
14.1. Bewertung der Ergebnisse	105
14.2. Vergleich mit anderen Produkten	105
14.3. Ausblick	105

III. Anhang	107
A. Glossar	108
B. Literaturverzeichnis	111

Teil I.
Einführung

1. Abstract

Der SAML Standard wird für die verteilte Authentifizierung vielfach eingesetzt. Penetration Tester müssen diese Installationen auf ihre Sicherheit prüfen können. Typische Aufgaben sind die Durchführung von XSW Angriffen oder die Signierung von manipulierten SAML Assertions mit gefälschten X.509 Zertifikaten. Für einen Penetration Tester ist es wichtig, dass er eine SAML Umgebung einfach und korrekt auf Schwachstellen oder Fehlkonfigurationen prüfen kann.

Da eine SAML Assertion nur eine kurze Zeit gültig ist, reicht die Zeit oft nur knapp aus um eine Assertion zu manipulieren. Zudem ist diese Aufgabe sehr Fehleranfällig, da alles genau stimmen muss. Um den Penetration Tester dabei zu unterstützen, schrieben wir ein Plugin für die Burp Suite.

Das Plugin ist in zwei Teile unterteilt. Im ersten Teil können X.509 Zertifikate verwaltet werden. Dazu gehören folgende Funktionen:

- Zertifikate importieren und exportieren
- Details zu den Zertifikaten anzeigen
- Private Schlüssel importieren und exportieren
- Zertifikate klonen, bearbeiten und neue erstellen

Der zweite Teil widmet sich den SAML Messages. Die Zertifikate aus dem ersten Teil werden für das Signieren der SAML Messages verwendet. Der zweite Teil bietet folgende Funktionen:

- SAML Messages decodieren und encodieren
- Wichtigste Informationen der SAML Messages anzeigen
- SAML Messages darstellen (Schön eingerückt mit Syntax Highlighting)
- SAML Messages manuell editieren
- SAML Responses und SAML Assertions signieren
- Signatur aus den SAML Messages entfernen
- Durchführen von acht häufigen XSW Attacken

Somit steht dem Penetration Tester ein gutes Werkzeug zur Verfügung, um SAML Installationen durch die Unterstützung unseres Plugins "SAML Raider" effizient zu testen.

2. Management Summary

2.1. Ausgangslage

Grund der Projekt lancierung

Security Assertion Markup Language (SAML) ist ein Standard für die verteilte Authentifizierung und geteilte Identitäten über mehrere Organisationen. Dieser Standard wurde in Produkten wie Shibboleth oder Microsoft ADFS implementiert. Diese Implementationen werden von Penetration Tester auf ihre Sicherheit getestet.

Das Testen von SAML Implementationen ist ein aufwändiger Prozess. Dazu gehört beispielsweise das Dekodieren, Manipulieren, Signieren und wieder Zurückkodieren von SAML Nachrichten. Diese Vorgänge sind recht fehleranfällig. Der Tester darf sich deshalb keine Fehler erlauben, da sonst der Test nicht funktioniert. Da eine SAML Nachricht nur eine kurze Gültigkeitsdauer hat, müssen diese Schritte unter Zeitdruck durchgeführt werden, was ein korrektes und sorgfältiges Prüfen zusätzlich erschwert.

Projektziele

Damit die notwendigen Schritte automatisiert werden können, ist es das Ziel ein Plugin für die Burp Suite zu schreiben. Die Burp Suite ist eine Software, welche von vielen Penetration Tester genutzt wird um Webapplikationen auf ihre Sicherheit zu testen. Da diese Software jedoch keine speziellen SAML Funktionen anbietet, soll eine Erweiterung dafür geschrieben werden.

Um zu verstehen, wie der SAML Standard funktioniert, sollen die Technologiegrundlagen zu SAML erarbeitet und dokumentiert werden. Dazu gehört ein Studium der Funktionsweise von SAML sowie eine Recherche über mögliche Angriffe von einem Penetration Tester.

Mit einer selbst entworfenen Übungsaufgabe und dazugehöriger Musterlösung soll unser Plugin für Schulungszwecke beispielsweise im Hacking-Lab verwendet werden können.

2.2. Vorgehen, Technologien

Zu Beginn des Projektes wurde eine Projektplanung anhand des RUP Projektmanagementmodells erstellt.

In einem ersten Teil des Projektes eigneten wir uns das Wissen zur Funktionsweise von SAML und den möglichen Angriffen an. Nachdem die Anforderungen an das Plugin mit dem Projektpartner definiert wurde, begannen wir mit der Planung der Architektur. Die Architektur setzt auf das bekannte Model-View-Controller Pattern.

Bei der Entwicklung setzten wir auf die Programmiersprache Java, da wir Java sehr gut kennen und für die Benutzung des Plugins in Burp keine weitere Software nachinstalliert werden muss.

Damit wir unser Plugin auf korrekte Funktionsweise in einer realen Umgebung testen konnten, setzten wir eine Testumgebung auf. Diese Umgebung besteht aus einem Identity Provider, welcher den Benutzer authentifiziert und zwei Service Providern, welche eine Webseite für den Benutzer anbieten.

2.3. Ergebnisse

Anhand der Anforderungen entstand ein Tool, welches in zwei Teile aufgeteilt ist: Der erste Teil dient der Verwaltung von Zertifikaten und der zweite Teil dient der Manipulierung der SAML Nachrichten.

Von 20 funktionalen Anforderungen decken wir 14 ab. Alle funktionalen Anforderungen mit der Priorität "sehr hoch", sowie die meisten der Priorität "hoch" sind erfüllt. Einige weitere nicht hoch priorisierte Anforderungen wurden ebenfalls implementiert. Während der Entwicklung und der Zwischenpräsentation kamen weitere Anforderungen dazu, welche wir ebenfalls implementierten. Die Implementation ist sehr nahe an der geplanten Architektur.

Zusammengefasst kann man sagen, dass die grössten beziehungsweise wichtigsten Teile des Plugins funktionstüchtig implementiert wurden. Wir mussten nur kleine Abstriche bei den Funktionen machen. Ein Penetration Tester kann unser Plugin bereits zum Testen einsetzen und die wichtigsten Aufgaben durchführen.

2.4. Ausblick

Anschliessend an dieses Projekt können die nicht abgedeckten Anforderungen implementiert werden. Die grafische Oberfläche des Plugins ist gut benutzbar, jedoch noch nicht vollkommen ausgereift. Deshalb müsste man dort nochmals ein bis zwei Manntage investieren um die Oberfläche zu perfektionieren. Des Weiteren fehlt eine grössere Funktion, welche es dem Tester erlaubt, eine spezifische Art von Attacken vollautomatisch durchführen zu lassen. Diese Funktion selbst zu implementieren würde jedoch sicherlich die Hälfte der Zeit in Anspruch nehmen, die wir für unser Projekt benötigten.

3. Aufgabenstellung

Die Aufgabenstellung gibt einen Überblick über die möglichen Tätigkeiten und Rahmenbedingungen dieser Arbeit.

3.1. Einführung

SAML2 etabliert sich je länger je mehr als der de-facto Standard für verteilte Authentisierung. Zudem sind weitere, sehr ähnliche Standards wie Shibboleth und Microsoft Claims resp. Active Directory Federation Services am Markt vertreten. Die Authentisierungsverfahren werden mehrheitlich für HTTPs basierte Dienste eingesetzt und machen Gebrauch von der XML-Technologie. Dabei werden bekannte Trust-Konzepte als auch etablierte Codierungs-, Verschlüsselungs- und Signaturverfahren angewandt. Das Testen von solchen Verfahren und den enthaltenen Nachrichten gestaltet sich aufgrund derer Komplexität aber etwas schwierig und umständlich.

3.2. Aufgabe

Die Arbeit soll nun die erwähnten Verfahren gegenüberstellen und ein Test-Plugin realisieren, das die Prüfung der Verfahren vereinfacht. D.h. das Plugin soll den Tester nicht nur mit Codierungs- und Cryptoverfahren unterstützen, sondern auch mit bereits bekannten Schwachstellen in der Verarbeitung der Nachrichten.

Eine Testumgebung für Microsoft Claims basierte Authentisierung ist bereits vorhanden.

3.2.1. Mögliche Tätigkeiten

- Studium der 3 Protokolle (SAML2, Shibboleth und Microsoft Claims)
- Studium vorhandener Unterlagen zu SAML, resp. XML Schwachstellen
- Erarbeitung einer Übersicht und Erklärung von bekannten Angriffsszenarien
- Übersicht von möglichen Schwachstellen in Implementationen (IDP und SP)
- Extrakt von Testfällen (step-by-step) und Formulierung von Gegenmassnahmen
- Studium des Burp Plug-in Mechanismus
- Programmierung eines Burp-Plugins, das die Testfälle abdeckt oder unterstützt
- Erstellung von Übungsaufgaben und Musterlösungen um die Angriffe und Abwehrmechanismen zu trainieren

3.2.2. Benötigtes Skillset

- Neue Authentisierungsverfahren verstehen
- Bekannte Angriffskonzepten verstehen und Testfälle erarbeiten
- Erarbeitung eines Burp Plug-in für die Unterstützung von Penetration Tester

3.3. Vorgehen

Im Rahmen der allgemeinen Richtlinien zur Durchführung von Studien- und Bachelorarbeiten gemäss eigenem Projektmanagementplan. Dieser Projektmanagementplan ist als Erstes von den Studenten zu erstellen und enthält insbesondere:

- Die Beschreibung des dem Projektcharakter angepassten Vorgehensmodells.
- Eine erste Aufteilung der Aufgabe in gemeinsam und einzeln zu bearbeitende Teile unter Berücksichtigung der vorgegebenen Teilaspekte. Die genaue Aufteilung muss spätestens nach der Technologiestudie erfolgen.
- Den Projektplan (Zeitplan) und die Meilensteine.

3.4. Randbedingungen

- Wo möglich sollten Open Source Produkte eingesetzt werden
- Das entwickelte Burp-Plugin sollten Plattform-Unabhängig betrieben werden können (Windows, Mac OSX und Linux)

3.5. Infrastruktur

Die Arbeiten werden auf den zugeteilten Rechnern an der HSR mit der Standardinstallation durchgeführt. Zusätzlich benötigte Software oder Hardware wird bei Bedarf und nach Rücksprache mit Compass Security zur Verfügung gestellt.

3.6. Erwartete Resultate

3.6.1. In elektronischer Form

- Installationskit (alle Dateien für eine Installation und Installationsanweisung)
- kompletter Source Code
- komplette Software Dokumentation (Use Cases, Klassenmodell, Sequenzdiagramme usw. in UML)
- komplettes Use Cases und Success Scenarien resp. Musterlösungen
- alle Dokumente
- alle Protokolle

3.6.2. Auf Papier

Gemäss der Anleitung der HSR: <https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html>

Es muss aus den abgegebenen Dokumenten klar hervorgehen, wer für welchen Teil der Arbeit und der Dokumentation verantwortlich war.

3.7. Termine

3.7.1. Start/Ende

Gemäss Vorgabe der HSR: <https://www.hsr.ch/Termine-Bachelor-und-Studiena.5142.0.html>

16.02.15	Beginn der Bachelorarbeit, Ausgabe der Aufgabenstellung durch die Betreuer.
Mai 2015	Fotoshooting. Genauere Angaben erteilt die Kommunikationsstelle rechtzeitig.
05.06.15	Die Studierenden geben den Abstract für die Diplomarbeitsbroschüre zur Kontrolle an ihren Betreuer/Examinator frei. Die Studierenden erhalten vorgängig vom Studiengangsekretariat die Aufforderung und die Zugangsdaten zur Online-Erfassung des Abstracts für die Broschüre. Die Studierenden senden per Email das A0-Poster zur Prüfung an ihren Examinator/Betreuer. Vorlagen sowie eine ausführliche Anleitung betreffend Dokumentation stehen unter den allgemeinen Infos Diplom-, Bachelor- und Studienarbeiten zur Verfügung.
10.06.15	Der Betreuer/Examinator gibt das Dokument mit dem korrekten und vollständigen Abstract für die Broschüre zur Weiterverarbeitung an das Studiengangsekretariat frei.
12.06.15	Abgabe des Berichtes an den Betreuer bis 12.00 Uhr. Fertigstellung des A0-Posters bis 12.00 Uhr.
12.06.15	Präsentation der Bachelorarbeiten, 16 bis 20 Uhr
03.08.15 - 21.08.15	Mündliche BA-Prüfung
25.09.15	09.00 Uhr Bachelorfeier und Ausstellung der Bachelorarbeiten

3.7.2. Zeitplan und Meilensteine

Zeitplan und Meilensteine für das Projekt sind von den Studenten selber zu erarbeiten und zusammen mit dem Projektmanagementplan abzuliefern. Die Meilensteine sind bindend. Der erste Meilenstein ist vorgegeben. Mit den Betreuern werden regelmässige Sitzungen zur Fortschrittskontrolle durchgeführt.

Abgabetermin Projektmanagementplan mit Zeitplan: 2. März 2015

3.8. Betreuung

Die Arbeiten werden durch Cyrill Brunswiler betreut.

3.8.1. Kontakt

Cyrill Brunswiler, Managing Director, Compass Security Schweiz AG
Werkstrasse 20, 8645 Jona, Switzerland
<http://www.csnc.ch/>

Please submit files securely using Filebox <https://secure.csnc.ch/inbox/DshpdeDVVP68Pp>

3.9. Referenzen


- Vorgaben HSR <https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html>
- Burp Proxy <http://portswigger.net/burp/proxy.html>
- Breaking SAML https://owasp.org/images/2/28/Breaking_SAML_Be_Whoever_You_Want_to_Be_-_Juraj_Somorovsky%2BChristian_Mainka.pdf
- Analysis of Signature Wrapping Attacks and Countermeasures <http://lists.w3.org/Archives/Public/public-xmlsec/2009Nov/att-0019/Camera-Ready.pdf>
- Secure SAML validation to prevent XML signature wrapping attacks <http://arxiv.org/ftp/arxiv/papers/1401/1401.7483.pdf>
- Shibboleth Federated Identity Solution <https://shibboleth.net/>
- Microsoft Active Directory Federation Services Overview <https://technet.microsoft.com/en-us/library/hh831502.aspx>
- Microsoft Introduction to Claims <https://msdn.microsoft.com/en-us/library/ff359101.aspx>

3.10. Unterschriften

Jona, 16. Februar 2015



Cyrill Brunschwiler



Emanuel Duss



Roland Bischofberger

Teil II.

Technischer Bericht

4. Einleitung und Übersicht

4.1. Problemstellung

Testet ein Penetration Tester eine Security Assertion Markup Language (SAML) [65] Umgebung auf Schwachstellen, beinhaltet dies sehr viele manuelle Schritte wie beispielsweise das Dekodieren, Manipulieren, Signieren und wieder Enkodieren von SAML Nachrichten. Diese Schritte sind zum Teil mühsam auszuführen. Bei so vielen manuellen Schritten besteht eine erhöhte Wahrscheinlichkeit, dass ein Fehler gemacht wird und somit der Test nicht korrekt durchgeführt wird. Ausserdem besteht je nach zu testender Konfiguration ein Zeitdruck, da SAML Nachrichten unter meist nur wenige Minuten oder gar Minuten Bruchteile gültig sind. Dies erschwert korrektes und sorgfältiges Prüfen zusätzlich.

4.2. Aufgabenstellung

In dieser Arbeit sollen die Grundlagen von SAML erarbeitet werden, die bereits erforschten Angriffe auf SAML zusammengetragen und ein Plugin für die Burp Suite geschrieben werden. Dieses Plugin soll einige Schritte im Testablauf für den Penetration Tester automatisieren. Die detaillierte Aufgabenstellung ist im Kapitel 3 zu finden.

4.3. Inhalt dieses Berichts

Dieser technische Bericht erklärt wie SAML funktioniert und worauf die Sicherheit von SAML beruht. Hierfür wird auf die X.509 Zertifikate [16] eingegangen. Danach werden Angriffe auf SAML und die eingesetzte X.509 Zertifikatinfrastruktur gezeigt. Dabei handelt es sich um bereits bekannte Angriffe.

In der zweiten Hälfte geht es um das Herzstück der Arbeit: Unser Plugin mit dem Namen "SAML Raider". Die Analyse der Anforderungen stellen diesem Teil vor. Desweiteren wird erklärt, wie mit der Burp Suite Plugins entwickelt werden und wie wir bei der Entwicklung vorgegangen sind.

In unserer Arbeit visualisieren wir jeweils XML Bäume beziehungsweise SAML Nachrichten mit einem Baumdiagramm. Dabei werden die einzelnen Elemente wie in der Abbildung 4.1 dargestellt.

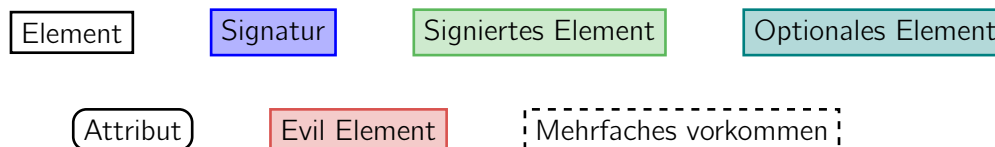


Abbildung 4.1.: Legende für die XML Bäume

5. Einführung in SAML

Dieses Kapitel bietet einen Überblick über SAML. Im Abschnitt 5.1 wird OASIS, als SAML Standardersteller erläutert. Was SAML ist und wie der Standard aufgebaut ist, wird in den Abschnitten 5.2 und 5.3 erläutert. Die verschiedenen Use Cases von SAML beziehungsweise die Profiles sind im Abschnitt 5.4 illustriert und beschrieben. Eine Auswahl von Software, welche SAML implementiert, wird im Abschnitt 5.5 verglichen und beschrieben.

5.1. OASIS

Die Organization for the Advancement of Structured Information Standards (OASIS) ist eine non-profit Organisation, welche offene Standards entwickelt. Die Standards befassen sich beispielsweise mit Informationssicherheit, dem Internet of Things oder mit Cloud Computing [41]. Auf der Webseite der OASIS (<https://www.oasis-open.org>) sind 116 Standards frei abrufbar. Einige bekannte davon sind DocBook für die Erstellung von Büchern oder das OpenDocument Format (ODF), das freie Dateiformat für Büroanwendungen, welches LibreOffice und OpenOffice verwendet wird und auch von Microsoft Word unterstützt wird. Viele weitere Standards beschäftigen sich mit Webservices wie Beispielsweise Web Services Security (WSS) [44]. Die OASIS besteht aus 91 Untergruppen, so genannte Technical Committee (TC), welche die Standards entwickeln. Der PKCS #11 Standard für Smart Cards wird beispielsweise von der TC "PKCS#11 TC" verwaltet und weiterentwickelt. SAML, womit sich unsere Arbeit beschäftigt, kommt aus der TC "Security Services (SAML)" [42].

5.2. Was ist SAML?

SAML ist ein auf XML (Extensible Markup Language) basiertes Framework für den Austausch von Sicherheitsinformationen wie Angaben über die Authentifizierung, Eigenschaften oder Berechtigungen von Benutzern oder Systemen [38].

Der Standard von SAML ist plattformunabhängig geschrieben. Für Benutzer bietet es den Vorteil, dass sich ein Benutzer nur noch einmal für beliebig viele Applikationen einloggen muss und sich somit nur ein Passwort merken muss. Da das Passwort Serverseitig nur noch an einem Ort abgelegt werden muss, vermindert sich der administrative Aufwand für die Benutzerverwaltung pro Applikation [38].

SAML 1.0 wurde 2002 von der OASIS spezifiziert. 2003 folgte die Version 1.1 und hatte grossen Erfolg im Finanz- und Bildungswesen sowie Regierungen und in der Industrie. SAML wurde von vielen Herstellern implementiert und demonstrierte die Interoperabilität zwischen unterschiedlichen Herstellern [4].

In SAML 2.0 kamen viele weitere Features hinzu: So kann als Benutzer ID eine zufällige Zahl verwendet werden um die Privatsphäre zu Schützen. Eine Neuerung ist auch die Verschlüsselung von Assertions. Durch neue Attribute Profiles, ist es möglich LDAP Informationen in Assertions zu hinterlegen. Das Single Logout Protocol wurde auch in SAML 2.0 eingeführt, welches ein globales Ausloggen von allen Diensten ermöglicht [38]. Diese Neuerungen waren für SAML ein wichtiger Schritt um zu einem anerkannten und weit verbreiteten Standard zu werden [38].

SAML wird in folgenden drei Einsatzgebieten eingesetzt [38]:

Single Sign-On (SSO) Mit SAML ist es möglich über mehrere Domains hinweg ein Single Sign-On einzurichten. Ein Benutzer muss sich nur einmal an einem Dienst anmelden, damit ihm weitere Dienste ebenfalls den Zugriff gewähren. Somit muss sich ein Benutzer nur noch ein Passwort für viele Dienste merken und die Service Provider brauchen kein eigenes Benutzer- und Passwort Management. [38]

Federated Identity Mit SAML ist es möglich ein einziges Login in mehreren Organisationen zu verwenden. Dies ist nicht nur für den Benutzer praktisch, sondern auch für die Administratoren, da somit die Benutzerdaten nicht auf mehreren System aktuell gehalten werden müssen. [38]

Web Services Mit SAML können die kommunizierenden Parteien mit einem Webservice authentifiziert werden, um weitere Webservices abzusichern. [38]

SAML wird von vielen Applikationen eingesetzt. Darunter befinden sich Google Apps, Office365, SuisseID oder Amazon Webservices [62]. Auch im universitären Umfeld in der Schweiz betreibt die SWITCH mit SWITCH AAI eine SAML Umgebung [64]. Benutzer der HSR Umgebung können beispielsweise mittels SAML am Moodle der HSR angemeldet werden.

5.3. Der SAML Standard

Auf der Homepage der OASIS [43] sind die Dokumente für dem SAML Standard [44] frei abrufbar: "Assertions and Protocols" [30], "Bindings" [32], "Profiles" [36], "Metadata" [35], "Authentication Context" [31], "Conformance Requirements" [33], "Security and Privacy Considerations" [37] "Glossary" [34] und "Errata" [40]. Der Standard macht regen Gebrauch von XML und definiert eigene XML Namespaces. Die dazugehörigen XML Schemas werden ebenfalls auf der Webseite zur Verfügung gestellt [44]. Im Gegensatz zum sehr umfangreichen Standard bietet der "Technical Overview" [39] einen guten Einstieg in die Thematik.

In diesem Unterkapitel werden die wichtigsten Begriffe und Verfahren erklärt.

5.3.1. SAML Teilnehmer

Eine SAML Kommunikation findet immer zwischen einer SAML Asserting Party und einer SAML Relying Party statt. Die Asserting Party, auch SAML Authority genannt, macht SAML Assertions. Die Relying Party nutzt diese Assertions. Ob die Relying Party der Assertion vertraut, hängt von der Vertrauensbeziehung (trust relationship) mit der Asserting Party und von einer gültigen Signatur ab. Der Teilnehmer, welcher eine Anfrage stellt, heisst SAML Requester. Der antwortende Teilnehmer heisst SAML Responder. [39]

In SAML gibt es verschiedene Rollen. Für Single Sign-On (SSO) gibt es beispielsweise zwei Rollen: Ein Identity Provider (IdP) und ein Service Provider (SP). [39]

SAML Assertions werden immer über ein Subject, welches sich authentifizieren kann, gemacht. Ein Subject kann ein Mensch oder ein Computer sein. Man spricht auch von einem Principal. [39]

Hier sind die wichtigsten Begriffe über die Teilnehmer zusammengefasst [39]:

SAML Asserting Party Stellt eine SAML Assertion aus.

SAML Authority Synonym für Asserting Party.

SAML Identity Provider Stellt eine Assertion aus.

SAML Principal Über den Principal wird eine Assertion gemacht.

SAML Requester Fragt nach einer SAML Assertion.

SAML Responder Sendet die SAML Assertion zurück.

SAML Service Provider Bietet einen Service falls Assertion gültig ist.

SAML Subject Synonym für Principal.

5.3.2. SAML Assertion

Dieser Teil des Standards erklärt, wie eine SAML Assertion aufgebaut ist. Eine SAML Assertion beinhaltet Aussagen oder Behauptungen über ein SAML Subject. Diese Angaben sind immer in eine XML Datei eingebettet. Die Assertion wird von der Asserting Party, aufgrund eines Requests von der Relying Party, erstellt. Die Assertion kann aber auch ohne Request von der Asserting Party zur Relying Party versendet werden. Es gibt drei Typen von Assertions [39]:

Authentication Statements Ein Benutzer wurde erfolgreich authentifiziert. Beispiel: Benutzer Samuel hat sich erfolgreich mit einem Passwort angemeldet. [39]

Attribute Statements Ein Benutzer besitzt ein bestimmtes Attribut. Beispiel: Der Benutzer Samuel ist HSR Student im Studiengang Informatik. [39]

Authorization decision Statements Ein Benutzer darf eine bestimmte Aktion ausführen: Der Benutzer Samuel darf auf die Netzwerkfreigabe "Skripteserver" zugreifen. [39]

Der Aufbau einer SAML Assertion ist im Dokument "Assertions and Protocols" des Standards [30] beschrieben. Dies ist ein Auszug aus dem Schema `saml-schema-assertion-2.0.xsd` von Zeile 57 bis 74 [44]:

```
<element name="Assertion" type="saml:AssertionType"/>
<complexType name="AssertionType">
  <sequence>
    <element ref="saml:Issuer"/>
    <element ref="ds:Signature" minOccurs="0"/>
    <element ref="saml:Subject" minOccurs="0"/>
    <element ref="saml:Conditions" minOccurs="0"/>
    <element ref="saml:Advice" minOccurs="0"/>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="saml:Statement"/>
      <element ref="saml:AuthnStatement"/>
      <element ref="saml:AuthzDecisionStatement"/>
      <element ref="saml:AttributeStatement"/>
    </choice>
  </sequence>
  <attribute name="Version" type="string" use="required"/>
  <attribute name="ID" type="ID" use="required"/>
  <attribute name="IssueInstant" type="dateTime" use="required"/>
</complexType>
```

Listing 5.1: Schema einer SAML Assertion [44]

Aus diesem XML Schema lässt sich der folgende Baum ableiten:

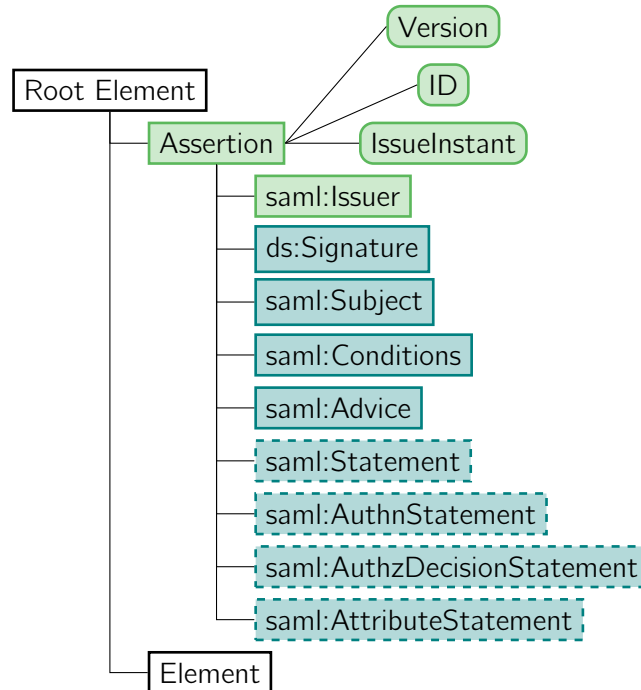


Abbildung 5.1.: Aufbau einer SAML Assertion

Dabei bedeuten die Felder folgendes [30]:

- `saml:Issuer`: Infos über den Aussteller der Assertion.
- `ds:Signature`: Die Signatur authentifiziert die Assertion und bringt Integrität. Die Signatur muss in der "enveloped" Form vorliegen (vgl. 6.2).
- `saml:Subject`: Gibt an Subject an, für die alle angegebenen Assertions gelten.
- `saml:Conditions`: Zusätzliche Bedingungen wie beispielsweise `NotBefore`, `NotOnOrAfter` oder `OneTimeUse`.
- `saml:Advice`: Zusätzliche Infos, welche die SAML Authority mitteilen will. Beispielsweise weitere Beweise für die Assertion oder weitere Verteilungspunkte für Assertion Updates.
- `saml:Statement`: Erweiterung, welche anderen Applikationen erlaubt die Assertion für weitere Zwecke zu verwenden. Dieses Element ist ein abstrakter Datentyp und kann nicht direkt verwendet werden. Die folgenden drei Statement Typen können verwendet werden.
- `saml:AuthnStatement`: Aussage dass das Subjekt zu einem Zeitpunkt sich richtig authentifiziert hat.
- `saml:AuthzDecisionStatement`: Beschreibt, warum das Subjekt sich erfolgreich Authentifiziert hat, mit einem optional angegebenen Beweis.
- `saml:AttributeStatement`: Gibt ein Attribut an, mit welchem das Subjekt assoziiert wird.

Im Listing 5.2 ist eine Beispiellassertion aus der Testumgebung zu sehen.

```
<Assertion ID="_643167e5-6ef4-4011-b5c7-c9d948752874"
  IssueInstant="2015-03-07T10:25:02.833Z" Version="2.0"
  xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
  <Issuer>http://SAMLWIN.saml.lan/adfs/services/trust</Issuer>

  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
```

```

<ds:CanonicalizationMethod
  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
<ds:SignatureMethod
  Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
<ds:Reference URI="#_643167e5-6ef4-4011-b5c7-c9d948752874">
  <ds:Transforms>
    <ds:Transform
      Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-✓
signature" />
    <ds:Transform
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
  </ds:Transforms>
  <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
  <ds:DigestValue>2o5nmG+qAhBHbYggoyu8DFXkuRSnp1Ww=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue> ... gekuerzt ... </ds:SignatureValue>
  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
      <ds:X509Certificate> ... gekuerzt </ds:X509Certificate>
    </ds:X509Data>
  </KeyInfo>
</ds:Signature>

<Subject>
  <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <SubjectConfirmationData InResponseTo="_b8378b2a3d44043465cd20191bad3139"
      NotOnOrAfter="2015-03-07T10:30:02.833Z"
      Recipient="https://samlcent/Shibboleth.sso/SAML2/POST" />
  </SubjectConfirmation>
</Subject>
  <Conditions NotBefore="2015-03-07T10:25:02.827Z"
    NotOnOrAfter="2015-03-07T11:25:02.827Z">
    <AudienceRestriction>
      <Audience>https://samlcent/shibboleth</Audience>
    </AudienceRestriction>
  </Conditions>
  <AttributeStatement>
    <Attribute Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn">
      <AttributeValue>browser@saml.lan</AttributeValue>
    </Attribute>
    <Attribute Name="http://schemas.xmlsoap.org/claims/Group">
      <AttributeValue>Domaenen-Benutzer</AttributeValue>
    </Attribute>
    <Attribute Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6"
      NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
      <AttributeValue>browser@saml.lan</AttributeValue>
    </Attribute>
  </AttributeStatement>
  <AuthnStatement AuthnInstant="2015-03-07T10:25:02.776Z">
    <AuthnContext>
      <AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:✓
PasswordProtectedTransport</AuthnContextClassRef>
    </AuthnContext>
  </AuthnStatement>
</Assertion>

```

Listing 5.2: Assertion aus unserer Testumgebung

Um die Sicherheit zu gewährleisten, muss der ganze Teilbaum "Assertion" mit einer XML Signatur signiert werden. Dieses Vorgehen wird in Kapitel 6.2 erklärt.

5.3.3. SAML Protocols

In den SAML Protocols wird beschrieben, welche Requests und welche dazugehörigen Responses es gibt. SAML definiert sechs verschiedene Protokolle [39]:

Authentication Request Protocol Ein SAML Principal kann eine Assertion mit einem Authentication Statement über sich anfordern.

Single Logout Protocol Ein SAML Principal kann sich gleichzeitig aus allen offenen Sessions ausloggen.

Assertion Query and Request Protocol Eine Asserting Party kann nach einer bereits existierenden Assertion fragen.

Artifact Resolution Protocol SAML Nachrichten können über eine kurze Referenz, SAML Artifact genannt, übermittelt werden. Über das Artifact Resolution Protocol kann das Artifact in die vollständige Nachricht aufgelöst werden.

Name Identifier Management Protocol Damit kann die ID, welche auf ein Principal verweist, geändert werden.

Name Identifier Mapping Protocol Damit können mehrere IDs, welche auf ein Principal verweisen, miteinander verknüpft werden.

5.3.4. SAML Bindings

SAML Nachrichten werden über bestehende Protokolle wie HTTP oder SOAP versendet. Wie SAML in die bestehenden Protokollen eingebettet und transportiert werden kann, ist in den SAML Bindings beschrieben. SAML definiert folgende Bindings [39]:

HTTP Redirect Binding Eine SAML Nachricht kann über HTTP Redirect (HTTP Status Code "302 Found") transportiert werden.

HTTP POST Binding Eine SAML Nachricht kann Base64 Kodiert über einen HTTP POST Request transportiert werden.

HTTP Artifact Binding Ein SAML Artifact kann über ein HTML Formular oder über ein HTTP GET Request transportiert werden.

SAML SOAP Binding Eine SAML Nachricht kann in einer SOAP Nachricht transportiert werden.

Reverse SOAP (PAOS) Binding Ein HTTP Client kann ein SOAP Responder sein.

SAML URI Binding Eine bestehende SAML Assertion kann über eine URL aufgelöst werden.

5.3.5. SAML Profiles

In den SAML Profiles werden die Anforderungen beschrieben, um ein bestimmten Use Case zu erfüllen. Für diese Beschreibungen werden bestehende Assertions, Protocols und Bindings verwendet. SAML definiert folgende Profiles [36]:

Webbrowser SSO Profile Beschreibt, wie mit einem Webbrowser Single Sign-On umgesetzt werden kann. Hierbei wird beispielsweise auf die Bindings HTTP Redirect, HTTP POST, HTTP Artifact und SOAP Bindings zurückgegriffen.

Enhanced Client and Proxy (ECP) Profile Beschreibt, wie Single Sign-On von einem Endpunkt, welcher kein Browser ist, funktioniert. Beispielsweise kann dies eine Desktopapplikation oder serverseitiger Code sein.

Identity Provider Discovery Profile Beschreibt, wie ein Benutzer dem Service Provider seinen Identity Provider angeben kann.

Single Logout Profile Beschreibt, wie das Single Logout Protocol umgesetzt werden kann.

Assertion Query/Request Profile Beschreibt verschiedene Query Typen für Assertions.

Artifact Resolution Profile Beschreibt, wie das Artifact Resolution Protocol umgesetzt werden kann.

Name Identifier Management Profile Beschreibt, wie das Name Identifier Management Protocol umgesetzt werden kann.

Name Identifier Mapping Profile Beschreibt, wie das Name Identifier Mapping Protocol umgesetzt werden kann.

5.4. SAML Profiles

In diesem Abschnitt beschreiben wir die SAML Profiles "Single Sign-On", "Single Logout" und "Identity Federation Services" aus dem SAML Standard genauer [36].

5.4.1. Single Sign-On Profile

Mit Single Sign-On (SSO) loggt sich der Benutzer beim Identity Provider ein, dieser gibt dem Benutzer eine Assertion, mit welcher er sich beim Service Provider ausweisen kann. In SAML 2.0 wird der Vorgang entweder durch den Service Provider (Service Provider Initiated Single Sign-On) oder durch den Identity Provider (Identity Provider Initiated Single Sign-On) gestartet. Die gebräuchlichste Form ist SP Initiated SSO [36].

Service Provider initiated Single Sign-On (Redirect/POST Bindings)

Beim Service Provider initiated Single Sing-On mit dem Redirect/POST Bindings kann sich ein Benutzer wie folgende Schritte zeigen bei einem Service Provider anmelden. Die Figur 5.2 stellt die Schritte als Sequenzdiagramm dar [36].

1. Der Benutzer besucht mit dem Browser eine Webseite und führt einen HTTP GET Request aus.
2. Der Benutzer ist beim Service Provider nicht angemeldet. Deshalb schickt der Service Provider als Antwort ein HTTP Redirect "302 Found" an den Client. In der URL Variable `SAMLRequest` steht ein SAML `AuthnRequest`. Darin sind der Issuer (Service Provider) und die Zeit enthalten.

3. Der erhaltene `AuthnRequest` wird an den Identity Provider weitergeleitet.
4. Der Identity Provider prüft, ob der Benutzer bereits angemeldet ist.
5. Falls der Benutzer noch nicht angemeldet ist, wird nach dem Benutzernamen und Passwort gefragt.
6. Der Benutzer meldet sich mit Benutzernamen und Passwort an.
7. Der Identity Provider prüft, ob die eingegebenen Zugangsdaten stimmen.
8. Der Security Context des Benutzers wird in eine SAML Assertion geschrieben. Diese Assertion wird digital signiert und als `SAMLResponse` Variable in ein HTML Formular eingepackt. Dieses HTML Formular wird per an den Browser geschickt.
9. Entweder klickt der Benutzer auf den `Submit` Button um die Assertion an den Service Provider zu schicken oder ein JavaScript Code löst den Versand automatisch aus.
10. Der Service Provider überprüft die digitale Signatur auf ihre Gültigkeit.
11. Der Service Provider prüft, ob der korrekt authentifizierte Benutzer auch die nötigen Rechte für die angeforderte Seite hat.
12. Falls alles OK ist, wird der Seiteninhalt ausgeliefert.

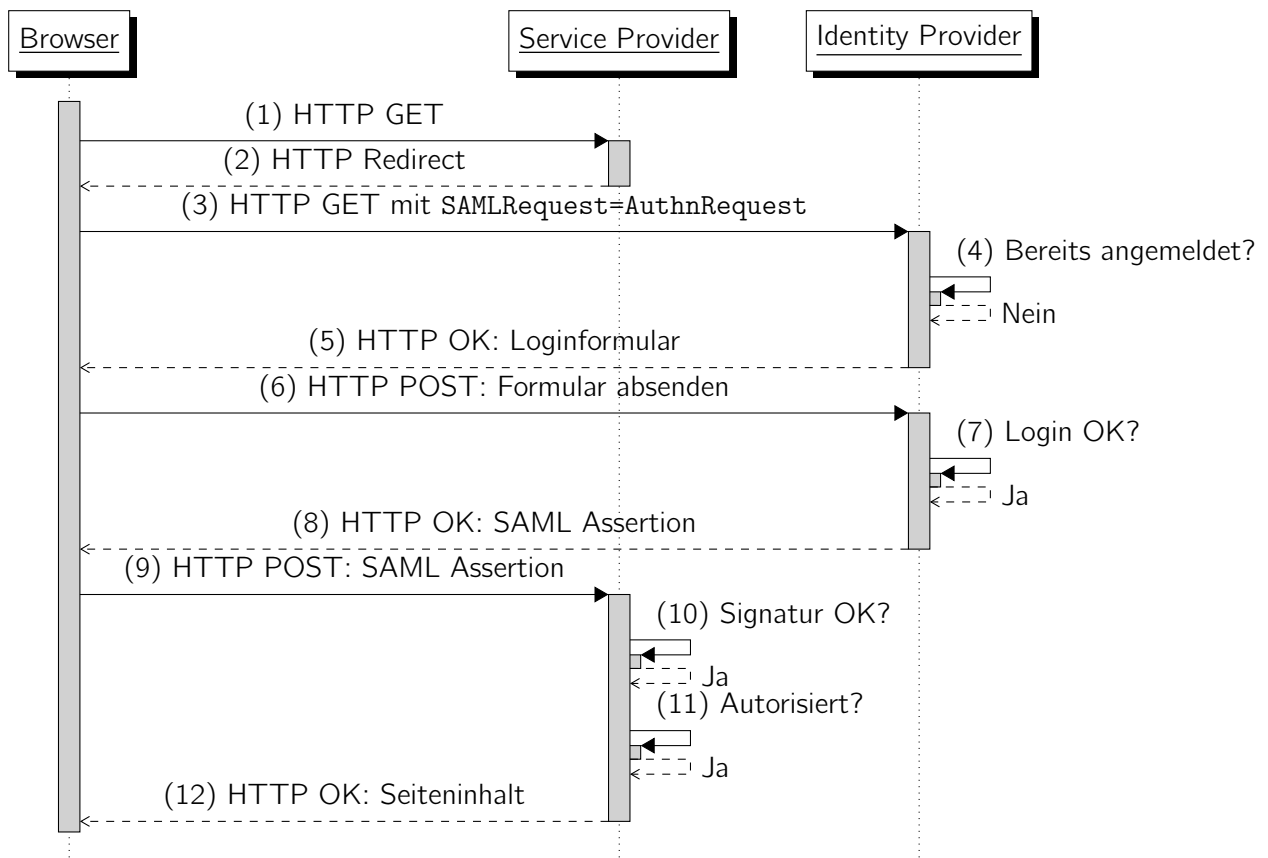


Abbildung 5.2.: Service Provider Initiated Single Sign-On (Redirect/POST Bindings) [36]

Service Provider initiated Single Sign-On (POST/Artifact Bindings)

In diesem Beispiel findet auch ein Service Provider initiated Single Sign-On statt, jedoch dieses Mal mit den POST/Artifact Bindings [36].

Der erste Unterschied ist, dass die erste Antwort vom Service Provider kein HTTP Redirect ist, eine normale HTTP Antwort mit dem Status Code 200. Der nachfolgende Request wird über ein Formular abgeschickt [36].

Der zweite Unterschied besteht darin, dass die Assertion nicht direkt an den Service Provider geschickt wird, sondern nur ein "Artifact", welches der Service Provider dazu nutzen kann die Assertion beim Identity Provider anzufordern [36].

1. Der Benutzer besucht mit dem Browser eine Webseite und führt einen HTTP GET Request aus.
2. Der Benutzer ist beim Service Provider nicht angemeldet. Der Service Provider schickt ein HTML Formular, welches einen SAML `AuthnRequest` enthält. Dieser ist im Formular in der Variable `SAMLRequest` zu finden.
3. Entweder klickt der Benutzer auf den `Submit` Button um den `AuthnRequest` an den Identity Provider zu schicken oder ein JavaScript Code löst den Versand automatisch aus.
4. Der Identity Provider prüft, ob der Benutzer bereits angemeldet ist.
5. Falls der Benutzer noch nicht angemeldet ist, muss dieser sich authentifizieren.
6. Der Benutzer meldet sich mit Benutzernamen und Passwort an.
7. Der Identity Provider prüft, ob die eingegebenen Zugangsdaten stimmen.
8. Der Security Context des Benutzers wird in eine SAML Assertion geschrieben. Diese Assertion muss nicht zwingend digital signiert sein, da das HTTP Artifact Binding verwendet wird. Dabei wird die Assertion direkt vom Identity Provider an den Service Provider übertragen.
9. Entweder klickt der Benutzer auf den `Submit` Button um das Artifact an den Service Provider zu schicken oder ein JavaScript Code löst den Versand automatisch aus.
10. Der Service Provider fragt mit einer `ArtifactResolve` Nachricht nach der Assertion. Dies wird über das SOAP Protokoll gemacht.
11. Der Identity Provider entpackt die `ArtifactResolve` Nachricht aus dem SOAP Body und sucht nach der dazugehörenden Assertion. Die Assertion wird wieder per SOAP dem Service Provider übermittelt.
12. Der Service Provider überprüft die digitale Signatur auf ihre Gültigkeit. Dieser Schritt wird nur gemacht, falls die Assertion signiert wurde. Das Signieren der Assertion ist optional.
13. Der Service Provider prüft, ob der korrekt authentifizierte Benutzer auch die nötigen Rechte für die angeforderte Seite hat.
14. Danach schickt der Service Provider eine HTTP Redirect Nachricht, welche den Benutzer auf die zu Beginn gewünschte Seite weiterleitet.
15. Der Benutzer fordert die Seite an.
16. Der Seiteninhalt wird ausgeliefert.

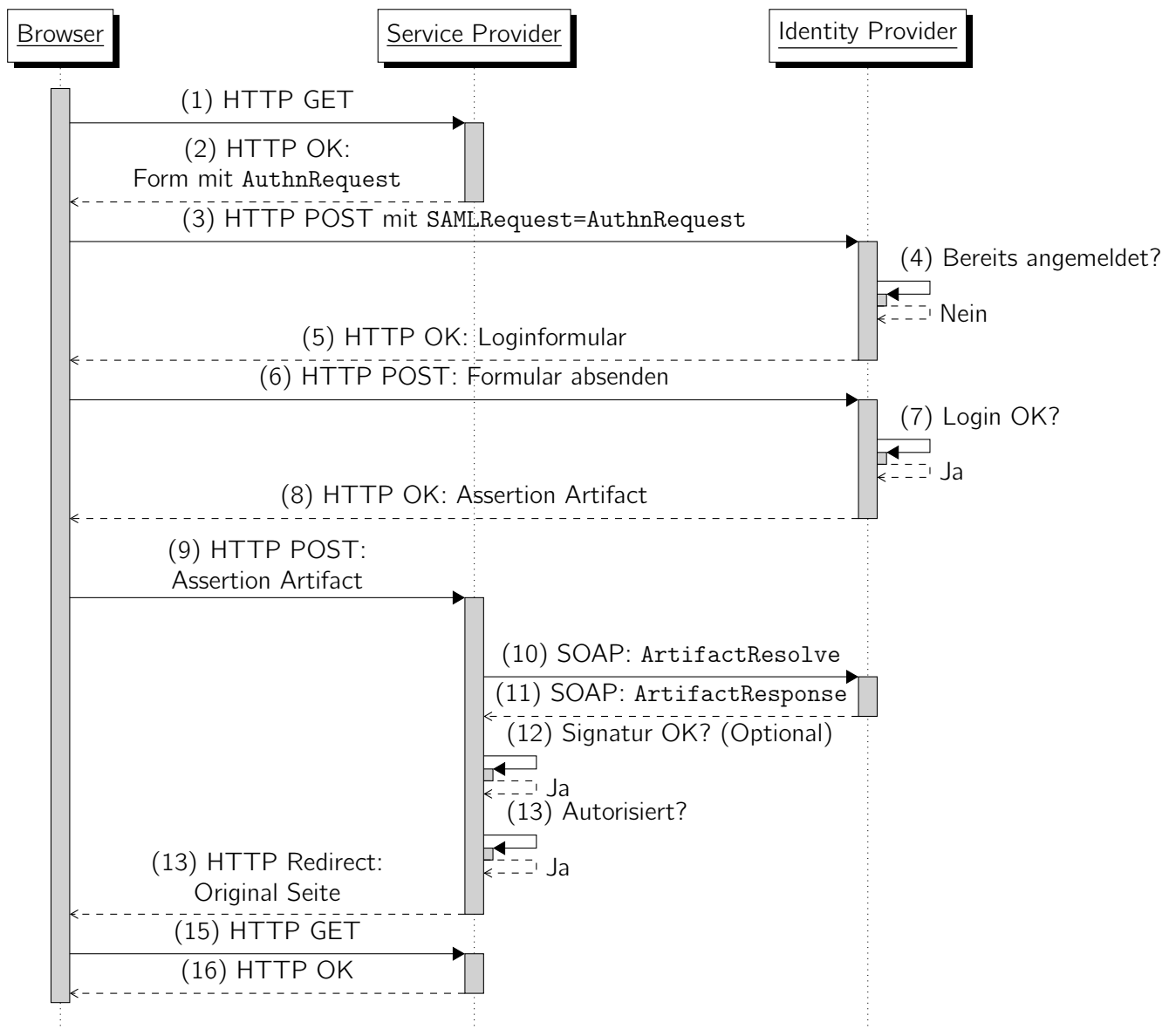


Abbildung 5.3.: Service Provider Initiated Single Sign-On (POST/Artifact Bindings) [36]

Identity Provider initiated Single Sign-On (POST Binding)

Bei dieser Variante wird zuerst die Verbindung zum Identity Provider hergestellt und dort ein Login vorgenommen. Dieses Login kann später dazu verwendet werden sich bei einem oder auch mehreren Service Provider anzumelden [36].

1. Der Benutzer meldet sich auf der Seite des Identity Providers an.
2. Der Identity Provider prüft den Benutzernamen und das Passwort.
3. Sind die Zugangsdaten gültig, wird dies bestätigt und der Benutzer ist eingeloggt.
4. Der Benutzer ruft auf dem Identity Provider einen Link auf, welcher schlussendlich zu einem Service Provider führt.
5. Der Identity Provider erstellt eine signierte SAML Assertion und sendet diese in einer HTML Form an den Browser zurück.

6. Entweder klickt der Benutzer auf den Submit Button um die Assertion an den Service Provider zu schicken oder ein JavaScript Code löst den Versand automatisch aus.
7. Der Service Provider überprüft die digitale Signatur auf ihre Gültigkeit.
8. Der Service Provider prüft, ob der korrekt authentifizierte Benutzer auch die nötigen Rechte für die angeforderte Seite hat.
9. Danach schickt der Service Provider eine HTTP Redirect Nachricht, welche den Benutzer auf die zu Beginn gewünschte Seite weiterleitet.
10. Der Benutzer fordert die Seite an.
11. Der Seiteninhalt wird ausgeliefert, da die Session jetzt authentifziert.

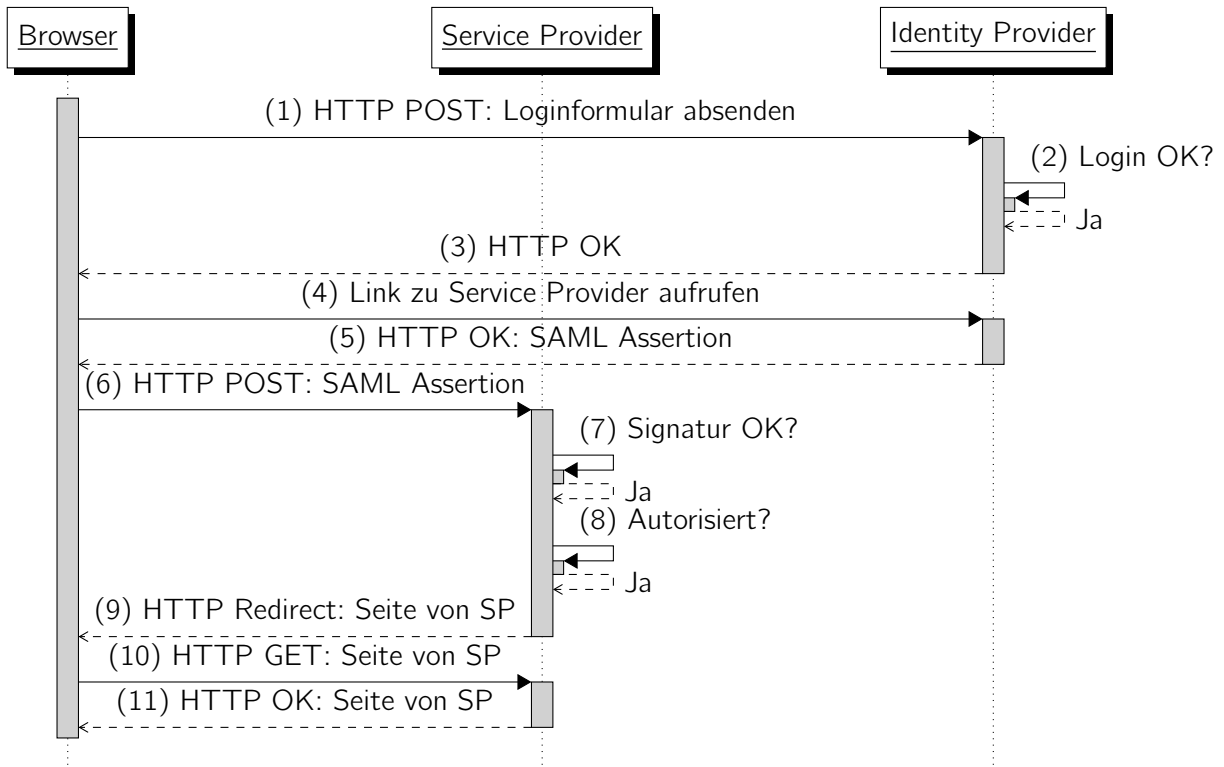


Abbildung 5.4.: Identity Provider Initiated Single Singn-On (POST Binding) [36]

Single Sign-On bei bereits bestehender Session zum IdP

Hat man sich beim IdP erfolgreich angemeldet, wird ein Cookie gesetzt. Dieses Cookie enthält eine Session ID, welche der Identity Provider kennt. Werde ich vom Service Provider an den Identity Provider weitergeleitet, kann der Identity Provider entscheiden, ob man sich in einem Formular mit Benutzername und Passwort einloggen kann oder ob das Session Cookie für die Authentifizierung noch gültig ist [59].

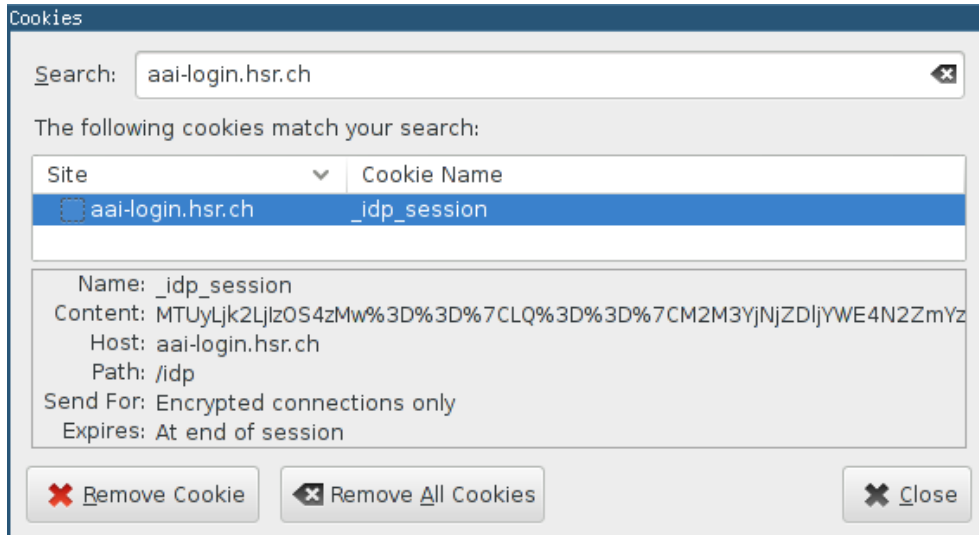


Abbildung 5.5.: Session Cookie `_idp_session` mit dem IdP der HSR

5.4.2. Single Logout

Um sich bei jedem Session Participant auf einmal abzumelden, gibt es in SAML das Single Logout Profile [36]. Dabei kontaktiert der User oder ein Administrator einen Session Participant, welcher ein SP oder IdP sein kann. Dieser Session Participant stösst nun den Single Logout Prozess an, wie in der Figur 5.6 zu sehen.

Jeder der folgenden `LogoutRequests` sollte laut Standard signiert und vom Protocol Binding in Ihrer Integrität gesichert sein [36].

1. Ein Session Participant beginnt den Single Logout Prozess und schickt einen `LogoutRequest` an den IdP, von welchem er die zugehörige Assertion erhalten hat.
2. Der IdP bestimmt die Session anhand des Principals im Single Logout Request und stellt fest, welche anderen Session Participants vorhanden sind in dieser Session.
3. Erhält aus seinen Daten die anderen Session Participants. Sollte kein anderer Session Participant vorhanden sein, wird direkt mit Schritt 6 fortgefahren.
4. Der IdP schickt einen `LogoutRequest` an alle anderen Session Participants.
5. Alle anderen Session Participants senden eine `LogoutResponse` zurück.
6. Der IdP sendet eine `LogoutResponse` an den ursprünglichen SessionParticipant, welcher den Logout Prozess gestartet hat.

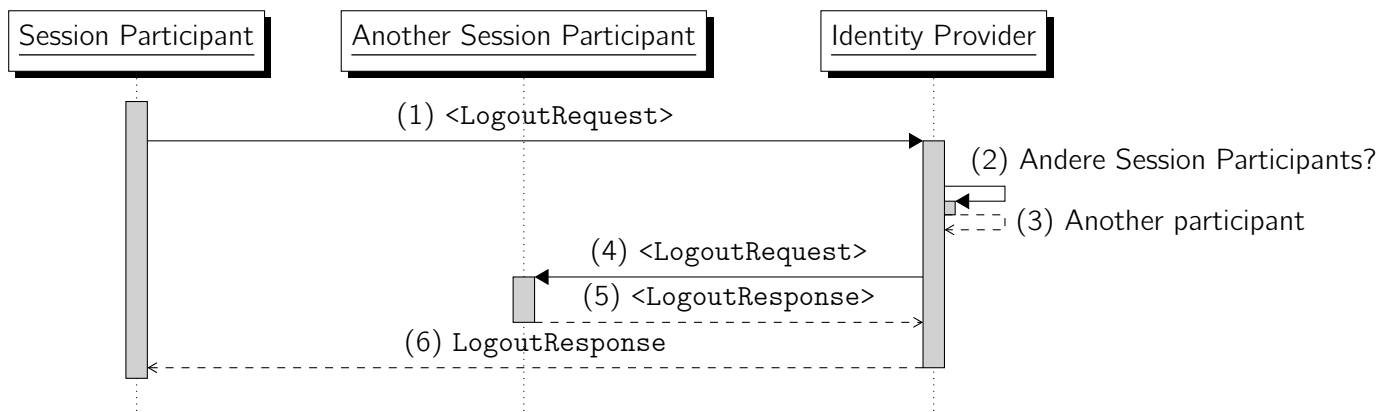


Abbildung 5.6.: Single Logout Profile [36]

5.4.3. Identity Federation Services

Provider können eine Vereinbarung erstellen, welche eine ID oder ein Attribut einem Benutzer zuordnet. Verbünden sich zwei oder mehrere Provider zusammen, spricht man von einer *Federation*. Bei jedem Provider kann diese gemeinsame ID mit einem lokalen Benutzer verknüpft werden, welcher bei jedem Provider unterschiedlich heissen kann. Man sagt, dass eine Local Identity mit einer Federated Identity verlinkt wird. Durch SAML können diese Informationen über die Benutzer dynamisch zur Laufzeit ausgetauscht werden. [39]

Es gibt folgende Federation Techniken [39]:

Out-of-Band Account Linking Beim Service Provider und beim Identity Provider existiert pro Benutzer ein Account, welcher ausserhalb von SAML synchronisiert wird.

Persistent Pseudonym Identifiers Beim Service Provider und beim Identity Provider existiert pro Benutzer ein Account, der aber anders heissen kann. Die Lokale ID (beispielsweise der Benutzername) wird nicht gegenseitig ausgetauscht, sondern nur eine "Linked ID", welche für eine eindeutige Identifizierung verwendet wird.

Transient Pseudonym Identifiers Hier muss nicht auf beiden Seiten ein Benutzeraccount gepflegt werden. Der Identity Provider erstellt für jeden erfolgreich eingeloggteten Benutzer eine temporäre ID, welche in der SAML Assertion an den Service Provider zurückgegeben wird. Dieser speichert in einer Tabelle, welche ID von welchem Identity Provider kommt. Mit diesem Ansatz kann der Benutzer anonym beim Service Provider erscheinen.

5.5. SAML Implementierungen

5.5.1. Shibboleth

Internet2 ist eine Community und Non-Profit Organisation aus Amerika [21], welche aus Universitäten wie dem MIT, grossen Unternehmungen wie Cisco und IBM, Partner wie das NIST und Bildungsnetzwerke besteht [22].

Shibboleth ist ein freies Federated Identity Software Paket für Single Sign-On. Sie wurde von Internet2 Organisation entwickelt und steht mit der Apache Software License unter einer freien Lizenz. Shibboleth implementiert den SAML 2.0 Standard von OASIS, welcher um Datenschutzfunktionen ergänzt wurden. So kann für jede Applikation separat festgelegt werden, auf welche Informationen (SAML Attributes) zugreifen darf. [23]

Shibboleth besteht aus folgenden Produkten [24]:

Centralized Discovery Service Webinterface für Service Provider, welche das SAML Discovery Service protocol unterstützen, auf dem die Benutzer ihren Identity Provider auswählen können. Eine Installation kann für mehrere Service Provider verwendet werden.

Embedded Discovery Service Webinterface für Service Provider auf dem die Benutzer ihren Identity Provider auswählen können. Eine Installation macht jeder Service Provider für sich.

Identity Provider Authentifiziert einen Benutzer und kann weitere Attribute über diese bekanntgeben. Diese Komponente funktioniert mit jeder bekannten SAML Implementierung.

Metadata Aggregator CLI Tool und REST Schnittstelle um Metadaten einzulesen, zu transformieren und wieder auszugeben. Über eine REST Schnittstelle können Metadaten abgefragt werden. Internet2 hat noch keine Implementation veröffentlicht.

OpenSAML-C++ Eine in C++ geschriebene API für SAML Nachrichten um diese beispielsweise zu generieren, lesen oder zu signieren. Diese API ist für Entwickler von SPs oder IdPs gedacht und ist keine Implementation von diesen. Sehr Lowlevel und nur knapp dokumentiert. Um damit zu arbeiten müssen sehr gute C++ Kenntnisse vorhanden sein.

OpenSAML-Java Eine in Java geschriebene API für SAML Nachrichten um diese beispielsweise zu generieren, lesen oder zu signieren. Diese API ist für Entwickler von SPs oder IdPs gedacht und ist keine Implementation von diesen. Sehr Lowlevel und nur knapp dokumentiert. Um damit zu arbeiten muss man gutes Hintergrundwissen haben.

Service Provider Ermöglicht ein Single Sign-On für Webapplikationen durch die Webserver Apache und IIS sowie durch FastCGI.

Für Shibboleth gibt es drei Mailinglisten (`{announce,users,dev}@shibboleth.net`), ein Wiki (<http://wiki.shibboleth.net/>), einen Bugtracker (<https://issues.shibboleth.net>), Security Advisories, eine Roadmap aber auch kommerziellen Support. Neben Internet2 als Principal Member ist auch SWITCH als Federation Member Mitglied des Shibboleth Consortiums. [24]

Security

Es wurde nichts im Wiki von Shibboleth [60] dazu gefunden, ob ein Shibboleth SP auf eine Replay Attacke anfällig ist. Es wurde deshalb mit der Burp Suite getestet. Das Ergebnis war, dass Shibboleth nicht anfällig darauf ist. In der Abbildung 5.7 ist die Response des SP zu sehen.

```
<p>Please include the following message in any email:</p>
<p class="error">opensaml&#58;&#58;SecurityPolicyException at
(https&#58;://samlcent/Shibboleth.sso/SAML2/POST)</p>

<p>Rejecting replayed message ID &#40;_6278a32e-dd5c-4bd3-ad1c-c97d737dddbc&#41;.</p>

</body>
</html>
```

Abbildung 5.7.: Untersuchung der Replay Attacke auf einen Shibboleth SP

5.5.2. Microsoft Claims

Microsoft Claims ist eine kommerzielle Single Sign-On und Federated Identity Lösung von Microsoft. Mit Windows Server 2008 R2 hat man automatisch eine Lizenz dazu. Claims implementiert den SAML 2.0 Standard. Als Identity Provider wird die Microsoft Serverrolle Active Directory Federation Services (ADFS) verwendet. ADFS stellt SAML Assertions aus, welche von Microsoft als Claim bezeichnet werden. [27]

Security

Mit ADFS werden die Möglichkeiten, wie man sich beim IdP authentifizieren kann, erweitert. Es ist damit möglich sich mit Kerberos, X.509 Zertifikaten oder Smartcards zu authentifizieren [26]. ADFS unterstützt verschiedene Mechanismen um die SAML Umgebung abzusichern. Es gibt eine Token Replay Detection welche sich verwendete Assertions merkt, damit diese kein zweites Mal verwendet werden können. Diese Sicherheitsmassnahme ist per default aktiviert. Die SAML Tokens können optional auch verschlüsselt übermittelt werden, damit ein Man-in-the-Middle diese nicht mitlesen kann. Dies muss manuell aktiviert werden. Zusätzlich wird über die Extended Protection for Authentication [29] ein zusätzliches Token, das Channel Binding Token, übermittelt, womit Man-in-the-Middle Angriffe erkannt werden können. Microsoft weist zudem darauf hin, dass sich sensible Informationen in den Token befinden können und deshalb die Logdateien nur für die Administratoren lesbar zu machen, welche diese wirklich brauchen. Standardmässig werden keine Informationen geloggt, welche es erlauben Personen zu identifizieren. [28]

5.5.3. Vergleich

Die Produkte von Shibboleth und von Microsoft Claims sind miteinander kompatibel, da beide den SAML 2.0 Standard implementieren. Somit kann jede Partei intern eigene Protokolle benutzen ohne auf eine verteilte Authentifizierung mit SAML zu verzichten [61].

Damit jedoch ein Shibboleth Service Provider mit einem ADFS Identity Provider ohne Probleme funktioniert, muss man in den Metadaten, welche vom IdP generiert werden, noch den Namespace für Shibboleth im EntityDescriptor eintragen. Ausserdem fehlen in den generierten Metadaten die Scope Informationen beim IDPSSODescriptor. Zu beachten ist hierbei, dass dies eine einmalige

Änderung ist. In den übertragenen Assertions ist keine Änderung mehr nötig. [1] Diese Schritte sind im Detail in unserer Testumgebung durchgeführt worden und in dem Kapitel 12 beschrieben.

Die Namensgebungen für die einzelnen Teile von SAML sind bei Shibboleth und Microsoft ADFS unterschiedlich. Die Tabelle 5.5.3 listet dabei die verschiedenen Namensgebungen auf. Wir benutzen in unserer Arbeit die Namen von Shibboleth, da diese gleich wie im SAML Standard sind.

ADFS Name	Shibboleth Name	SAML Konzept dahinter
Claims Provider, Issuer	Identity Provider	Identity Provider, Asserting Party, Authority
Relying Party	Service Provider	Service Provider
Security Token	Assertion	Assertion
Claims	Assertion Attributes	Attributes
Subject	Principal	Principal, Subject

Tabelle 5.1.: Vergleich der Namensgebungen [1]

5.5.4. SAML Bibliotheken

Die zwei vorgestellten Lösungen, Shibboleth und Claims, müssen SAML Nachrichten parsen, generieren, etc. Dazu werden bei beiden Produkten Bibliotheken eingesetzt. Shibboleth baut dabei auf OpenSAML auf, welche auch von Internet2 entwickelt wird [23]. Microsoft setzt dabei auf eine eigene Bibliotheken aus dem .NET Bereich [27].

6. SAML Security

Dieses Kapitel bietet eine Übersicht über die Technologien, welche eingesetzt werden um die Sicherheit von SAML zu gewähren. Der erste Abschnitt 6.1 befasst sich mit der X.509 Public Key Infrastructure und der zweite Abschnitt 6.2 mit XML Security.

6.1. X.509 Public Key Infrastructure

In diesem Abschnitt soll eine kurze Übersicht über die X.509 Public Key Infrastructure (PKI) gegeben werden. X.509 ist ein Standard der die Struktur und Funktion von digitalen Zertifikaten im X.509 Format definiert. Mit einem solchen Zertifikat können Public Keys an Subjekte gebunden werden. Diese Bindung wird von einer Certificate Authority signiert und somit beglaubigt. [16]

Es gibt weitere Standards für PKIs. Der Standard PKIX (Public Key Infrastructure X.509) der IETF beschreibt in vielen RFCs wichtige Funktionen von PKIs. PKIX basiert auf den X.509 Zertifikaten. Ein weiterer Standard ist die "Common PKI", welche sich auf den rechtlichen Hintergrund des Signierungsgesetzes konzentriert. [57]

6.1.1. Aufbau

Ein X.509 Zertifikat ist im RFC 5280 "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" spezifiziert und setzt sich aus folgenden drei Abschnitten zusammen [16]:

Zertifikat

- Version: Version von X.509 entweder v1 (0x0), v2 (0x1) oder v3 (0x2). Die hexadezimale Schreibweise ist jeweils um 1 kleiner, da bei 0 mit zählen beginnt wird.
- Seriennummer: Maximal 20 Byte grosse ID, welche von der CA zugewiesen wird.
- Algorithmus: Algorithmus, welcher für die Signatur verwendet wird. Dies kann RSA, DSA oder auch eine elliptische Kurve sein.
- Issuer: X500 Name der Certificate Authority.
- Gültigkeit: Besagt ab wann und bis wann ein Zertifikat gültig ist.
- Subject: X500 Name für den Zertifikatsinhaber.
- Subject Public Key Info: Enthält den Public Key und den verwendeten Algorithmus.
- Issuer UID und Subject UUID: Diese Optionalen Felder können ein Subject eindeutig identifizieren, falls mehrere Zertifikate das gleiche Subject haben.

Ab dem X.509v3 Standard wurde mit Extensions eine Möglichkeit geschaffen, weitere Informationen in den Zertifikaten abzulegen. Im RFC 5280 sind bereits einige Erweiterungen definiert [16]. Folgend werden einige davon erklärt:

- Authority Key Identifier: Falls eine CA mehrere Keys für die Signatur verwendet, kann hier angegeben werden, welcher verwendet wurde.

- Subject Key Identifier: Gleiches für den Zertifikatsinhaber.
- Key Usage: Hier wird angegeben, für welche Zwecke das Zertifikat genutzt werden kann. Im RFC werden folgende definiert:
 - Digital Signieren (`digitalSignature`): Der Public Key vom Zertifikat darf für die Validierung von Signaturen verwendet werden.
 - Non Repudiation (`nonRepudiation`, Nichtabstreitbarkeit): Die signierten Daten können von einer dritten Partei als gültig erklärt werden.
 - Schlüssel verschlüsseln (`keyEncipherment`): Der öffentliche Schlüssel darf für die Verschlüsselung von privaten Schlüssel verwendet werden. Beispielsweise für die Übertragung eines symmetrischen Schlüssels.
 - Daten verschlüsseln (`dataEncipherment`): Mit dem Public Key dürfen direkt Daten verschlüsselt werden. Diese Option wird sehr selten verwendet, da meist auf symmetrische Kryptografie gesetzt wird.
 - Schlüssel Agreement (`keyAgreement`): Der Public Key darf für ein Schlüsselaustauschverfahren wie beispielsweise Diffie-Hellman verwendet werden.
 - Zertifikate Signieren (`keyCertSign`): Der Public Key vom Zertifikat darf für die Verifizierung von Signaturen von anderen Zertifikaten verwendet werden. Ist das CA Bit der Basic Constraint Erweiterung gesetzt, muss dies aktiviert sein.
 - CRLs signieren (`cRLSign`): Der Public Key darf für die Verifizierung von Signaturen von CRLs verwendet werden.
 - Nur verschlüsseln (`encipherOnly`): Für das Schlüsselaustauschverfahren dürfen Parameter nur mit dem Public Key verschlüsselt werden.
 - Nur entschlüsseln (`decipherOnly`): Für das Schlüsselaustauschverfahren dürfen Parameter nur mit dem Public Key entschlüsselt werden.
- Subject Alternative Name: Weitere X500 Namen, für welche das Zertifikat gültig ist. Hier gibt es verschiedene Typen:
 - `otherName`: Weitere X500 Namen.
 - `rfc822Name`: E-Mail Adressen.
 - `dNSName`: DNS Namen.
 - `x400Address`: x400 Adressen.
 - `directoryName`: Verzeichnisname für CRLs.
 - `uniformResourceIdentifier`: URI oder URL.
 - `iPAddress`: IP Adressen.
 - `registeredID`: Spezielle ID
- Issuer Alternative Name: Weitere X500 Namen vom Issuer. Hier gibt es die selben Typen wie beim Subject Alternative Name.
- Basic Constraints: Hier wird angegeben, ob das Zertifikat eine CA ist und wie viele CAs in der Kette unter diesem Zertifikat sein dürfen. Ist diese Pfadlänge 0, kann eine CA keine weiteren Zertifikate signieren, welche als CA fungieren.
- Extended Key Usage: Hier können weitere Key Usages angegeben werden. Beispiele hierfür sind:
 - `serverAuth`: Für TLS Webserver
 - `clientAuth`: Um Clients an einem TLS Webserver zu authentifizieren.
 - `codeSigning`: Ausführbaren Code signieren
 - `emailProtection`: E-Mails verschlüsseln / signieren. Wird beispielsweise für S/MIME Zertifikate verwendet.
 - `OCSPSigning`: Um OCSP Antworten zu signieren.
- CRL Distribution Points: Hier wird eine oder mehrere Liste angegeben, welche die revozierte Zertifikate beinhaltet.

Eine Extension kann als kritisch markiert sein. Dies bedeutet, dass jedes Programm, welches ein Zertifikat verarbeitet, die kritischen Extensions auch verstehen muss. [16]

Zertifikat Signatur Algorithmus

In diesem Abschnitt ist die ID des Kryptografischen Algorithmus enthalten, mit welchem die CA dieses Zertifikat unterschrieben hat [16].

Zertifikat Signatur

Hier ist eine digitale Signatur enthalten, welche anhand des Zertifikates, erster Abschnitt hier, berechnet wurde [16].

6.1.2. Vertrauensmodell einer PKI

Das Vertrauensmodell einer PKI basiert auf einem hierarchischen System. Hierbei kann einer CA vertraut werden. Wenn diese CA weitere Zertifikate signiert, traue ich denen Zertifikaten über die vertrauenswürdige CA. Zwischen der CA welcher ich traue und einem Zertifikat können mehrere CAs sein. Kette an Zertifikaten, welche überprüft werden muss. [57]

Bei der Überprüfung eines Zertifikats ist darauf zu achten, dass die Signatur korrekt ist, dass das Zertifikat zum aktuellen Zeitpunkt gültig ist und dass Zertifikat nicht widerrufen ist. [57]

Ein Zertifikat kann über sogenannte Certificate Revocation List (CRL) widerrufen und somit für ungültig erklärt werden. Diese Listen können heruntergeladen und gegen ein Zertifikat verglichen werden. Eine weitere Methode für die Widerrufung ist OCSP (Online Certificate Status Protocol), bei dem ein Server nach der Gültigkeit angefragt wird. Hier bietet sich der Vorteil, dass nicht die gesamte Liste mit revozierten Zertifikaten heruntergeladen werden muss. [57]

6.1.3. Beispiel: Erstellen einer CA und signieren eines Zertifikats

Certificate Authority

Um ein XML Dokument zu signieren, müssen wir zuerst ein Zertifikat erstellen. Hierzu erstellen wir zuerst mit OpenSSL eine neue Certificate Authority (CA) [20]:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:4096 \
  -keyout cakey.pem -out cacert.pem
[...]
Country Name (2 letter code) [AU]:CH
State or Province Name (full name) [Some-State]:St. Gallen
Locality Name (eg, city) []:Rapperswil
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SAML2 Burp Plugin
Organizational Unit Name (eg, section) []:BA Team Emanuel und Roland
Common Name (e.g. server FQDN or YOUR name) []:SAML2 Burp Plugin CA
Email Address []:
```

Listing 6.1: Erstellen einer CA mittels OpenSSL

Durch den OpenSSL Befehl `req` können Certificate Signing Requests nach dem PKCS #10 Standard [58] erstellt und signiert werden. Durch die Option `-x509` sagt man aber, dass der Output ein komplettes Zertifikat und nicht nur ein Certificate Signing Request sein soll. Die Option `-nodes` lässt den Private Key ohne Passwort speichern. Mit der Option `-days 365` erhält das Zertifikat eine Gültigkeitsdauer von einem Jahr. Die Schlüsselstärke der RSA Schlüssel wird auf 4096 Bit gestellt, was 142 Bit

Sicherheit entspricht [12]. Es wird ein Private Key mit `-keyout cakey.pem` in die Datei `cakey.pem` geschrieben und ein Zertifikat mit `-out cacert.pem` in die Datei `cacert.pem` [20].

Certificate Signing Request für den Benutzer

Als nächstes wollen wir für den Benutzer `bauser` ein neues Zertifikat erstellen. Hierfür erstellen wir ein Certificate Request nach dem PKCS#10 Standard [58] mit OpenSSL [20]:

```
$ openssl req -new -newkey rsa:4096 -nodes \  
-keyout bauserkey.pem -out bausercsr.pem  
[...]  
Country Name (2 letter code) [AU]:CH  
State or Province Name (full name) [Some-State]:St. Gallen  
Locality Name (eg, city) []:Rapperswil  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:XML Signature Test  
Organizational Unit Name (eg, section) []:BA  
Common Name (e.g. server FQDN or YOUR name) []:Bauser XML  
Email Address []:bauser@example.com  
  
Please enter the following 'extra' attributes  
to be sent with your certificate request  
A challenge password []:  
An optional company name []:
```

Listing 6.2: CSR für Benutzer erstellen

Durch den OpenSSL Befehl `req` mit der Option `-new` wird ein neuer Certificate Signing Request erstellt. Die RSA Schlüsselgröße wird mit `-newkey rsa:4096` ebenfalls auf 4096 Bit gestellt und eine Verschlüsselung mit einer Passphrase mittels `-nodes` deaktiviert. Der Private Key heisst `bauserkey.pem` und der Certificate Signing Request `bausercsr.pem` [20].

CSR von der CA signieren lassen

Jetzt muss der für den Benutzer `bauser` erstellte CSR von der CA signiert werden [14]:

```
$ openssl x509 -req -in bausercsr.pem -CA cacert.pem -CAkey cakey.pem \  
-CAcreateserial -out bausercert.pem  
Signature ok  
subject=/C=CH/ST=St. Gallen/L=Rapperswil/O=XML Signature Test/OU=BA  
/CN=Bauser XML/emailAddress=bauser@example.com  
Getting CA Private Key
```

Listing 6.3: CSR signieren

Durch den OpenSSL Befehl `req` mit der Option `-in` wird der CSR `bausercsr.pem` von der CA welche mit der Option `-CA cacert.pem` angegeben wurde signiert. Der Private Key der CA wird mit der Option `-CAkey cakey.pem` angegeben. Da noch keine Datei mit den Seriennummern der Zertifikate existiert, muss diese mit der Option `-CAcreateserial` erstellt werden. Das Zertifikat für den Benutzer `bauser` wird mit `-out` in die Datei `bausercert.pem` geschrieben [20].

Überprüfen des Zertifikats

Jetzt kann mit OpenSSL überprüft werden, von wem und für wen das Zertifikat ausgestellt wurde [20]:

```
$ openssl x509 -in bausercert.pem -noout -text
Certificate:
  Data:
    Version: 1 (0x0)
    Serial Number: 9281090331331178257 (0x80cd0e7ba74eaf11)
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=CH, ST=St. Gallen, L=Rapperswil,
           O=SAML2 Burp Plugin, OU=BA Team Emanuel und Roland,
           CN=SAML2 Burp Plugin CA
    Validity
      Not Before: Mar  5 12:09:16 2015 GMT
      Not After : Apr  4 12:09:16 2015 GMT
    Subject: C=CH, ST=St. Gallen, L=Rapperswil,
           O=XML Signature Test, OU=BA,
           CN=Bauser XML/emailAddress=bauser@example.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (4096 bit)
      Modulus:
        00:c3:9e:94:53:b5:94:2b:a7:c9:ee:21:06:5e:99:
[...]
```

Listing 6.4: Zertifikat vom Benutzer überprüfen

Das Zertifikat wurde von unserer CA "SAML2 Burp Plugin CA" für den Benutzer Bauser erstellt.

6.1.4. Analyse von Zertifikaten der schweizer Webseiten

Um ein Gefühl zu bekommen, welche Eigenschaften X.509 Zertifikate aufweisen, wurde eine Analyse von schweizer Zertifikaten vorgenommen. Das Ziel war es herauszufinden, welche Extensions eingesetzt werden und welche Algorithmen für den Public Key und die Signatur verwendet werden. So können, je nach dem wie die Ergebnisse ausfallen, Funktionen in unserem Plugin bei der Implementierung unterschiedlich priorisiert werden. Beispielsweise soll nicht zu viel Zeit in eine Extension gesteckt werden, welche gar nie verwendet wird.

Vorbereitung

Zuest wurden die Top 1 Million Webseiten von Alexa heruntergeladen. Alexa ist eine Firma von Amazon, welche den weltweiten Internettraffic analysiert und daraus Statistiken wie beispielsweise die TOP 1 Million Webseiten ableitet. [11]

```
$ curl -LO http://s3.amazonaws.com/alexastatic/top-1m.csv.zip
$ unzip top-1m.csv.zip
$ du -chs top-1m.csv*
22M top-1m.csv
9.7M top-1m.csv.zip
```

In diesem Archiv sind die Top 1 Million Webseiten von Alexa.

```
$ wc -l top-1m.csv
1000000 top-1m.csv
```

Davon hat es 2824 Domains aus der Schweiz:

```
$ grep -E '\.ch$' top-1m.csv | wc -l
2824
```

Von diesen Domains wird das TLS Zertifikat heruntergeladen und abgespeichert. Diesen Job erledigt ein kleines Bash Script:

```
#!/bin/bash

grep -E "\.ch$" top-1m.csv | cut -d, -f2 | while read domain
do
    echo -n "Trying $domain..."
    echo "" | timeout 2 \
        openssl s_client -connect $domain:443 -servername $domain 2>/dev/null \
        | openssl x509 -text > certs/${domain,,}.pem 2>/dev/null \
        && echo "  Got certificate for $domain." \
        || echo "  No certificate for $domain."
done

find certs/ -empty -exec rm {} \;
```

Im Verzeichnis certs sind 1977 Zertifikate verfügbar. Das sind 70% der angefragten Webseiten.

```
$ ls certs/ | wc -l
1977
```

Analyse

Diese Zertifikate können jetzt nach interessanten Eigenschaften abgefragt werden.

Folgende Extensions werden in den 1977 Zertifikaten verwendet. Die erste Spalte zeigt, wie oft die Extension vorkommt.

```
$ CERTDIR=certs/
$ echo "Extensions"
$ grep -h -E "^[0-9a-zA-Z]+.*: (critical)?$" $CERTDIR/* \
  | sed 's/critical//' | sort | uniq -c | sort -nr
Extensions
 1767          X509v3 Authority Key Identifier:
 1674          X509v3 Basic Constraints:
 1607          X509v3 Extended Key Usage:
 1605          X509v3 Key Usage:
 1599          X509v3 Subject Alternative Name:
 1599          X509v3 CRL Distribution Points:
 1591          Authority Information Access:
 1562          X509v3 Certificate Policies:
 1343          X509v3 Subject Key Identifier:
  115          X509v3 Issuer Alternative Name:
  111          Netscape Comment:
   60          CT Precertificate SCTs:
   10          Netscape Cert Type:
    4          1.3.6.1.4.1.311.21.10:
    1          1.3.6.1.4.1.311.21.7:
```

Die meisten Zertifikate besitzen einen RSA Public Key. Nur wenige setzen auf die eher modernen elliptischen Kurven:

```
$ echo "Public Key Algorithms"
$ grep -h "Public Key Algorithm" $CERTDIR/* | sort | uniq -c | sort -nr
Public Key Algorithms
  1945          Public Key Algorithm: rsaEncryption
    32          Public Key Algorithm: id-ecPublicKey
```

Die grosse Mehrheit der Schlüssel sind 2048 Bit gross. Die tiefe Bitzahl von 256 kommt von den elliptischen Kurven und stimmt mit deren Anzahl überein. Zwei Leute haben sich bei 4056 und 2084 wohl vertippt.

```
$ echo "Public Key Size"
$ grep -h "Public-Key:" $CERTDIR/* | sort | uniq -c | sort -nr
Public Key Size
  1644          Public-Key: (2048 bit)
   161          Public-Key: (1024 bit)
   138          Public-Key: (4096 bit)
    32          Public-Key: (256 bit)
     1          Public-Key: (4056 bit)
     1          Public-Key: (2084 bit)
```

Von den RSA Keys haben alle denselben Exponenten:

```
$ echo "Exponent"
$ grep -h "Exponent:" $CERTDIR/* | sort | uniq -c | sort -nr
Exponent
  1945          Exponent: 65537 (0x10001)
```

Für die Signatur wird am häufigsten SHA2 mit 256 Bit benutzt, knapp gefolgt von SHA1. SHA2 mit 512 Bit ist eher selten. Einige wenige setzen noch auf MD5.

```
$ echo "Signature Algorithms"
$ grep -m1 -h "Signature Algorithm" $CERTDIR/* | sort | uniq -c | sort -nr
Signature Algorithms
   980          Signature Algorithm: sha256WithRSAEncryption
   840          Signature Algorithm: sha1WithRSAEncryption
   101          Signature Algorithm: sha512WithRSAEncryption
    32          Signature Algorithm: ecdsa-with-SHA256
    24          Signature Algorithm: md5WithRSAEncryption
```

Total haben 67 Zertifikate die Aufgabe einer CA:

```
$ echo "Basic Constraint"
$ grep -h -E "CA:(TRUE|FALSE)" $CERTDIR/* | sort | uniq -c | sort -nr
Basic Constraint
  1607          CA:FALSE
    66          CA:TRUE
     1          CA:TRUE, pathlen:1
```

Folgende Key Usages sind vertreten. Die am deutlichst verbreitetsten Key Usages sind "Digital Signature" und "Key Encipherment".

```
$ echo "Key Usage"
$ awk '/X509v3 Key Usage/{ getline; print }' $CERTDIR/* | sort | uniq -c \
| sort -nr
Key Usage
 1429   Digital Signature, Key Encipherment
  114   Digital Signature, Key Encipherment, Key Agreement
   32   Digital Signature
   10   Digital Signature, Key Encipherment, Data Encipherment
    8   Digital Signature, Non Repudiation, Key Encipherment, Data ↵
        Encipherment
    8   Digital Signature, Non Repudiation, Key Encipherment
    2   Key Encipherment, Data Encipherment
    1   Digital Signature, Key Encipherment, Data Encipherment, Certificate ↵
        Sign
    1   Digital Signature, Key Encipherment, Certificate Sign
```

Folgende Extended Key Usages sind vertreten. Die am deutlich verbreitetsten Extended Key Usages sind "Web Server Authentication" und "TLS Web Client Authentication".

```
$ echo "Extended Key Usage"
$ awk '/X509v3 Extended Key Usage/{ getline; print }' $CERTDIR/* | sort \
| uniq -c | sort -nr
Extended Key Usage
 1453   TLS Web Server Authentication, TLS Web Client Authentication
  102   TLS Web Client Authentication, TLS Web Server Authentication
   30   TLS Web Server Authentication
   10   TLS Web Server Authentication, TLS Web Client Authentication,
        Netscape Server Gated Crypto
    7   TLS Web Server Authentication, TLS Web Client Authentication,
        Microsoft Server Gated Crypto, Netscape Server Gated Crypto
    2   TLS Web Server Authentication, TLS Web Client Authentication,
        Netscape Server Gated Crypto, Microsoft Server Gated Crypto
    1   TLS Web Client Authentication, TLS Web Server Authentication,
        Netscape Server Gated Crypto, Microsoft Server Gated Crypto
    1   Netscape Server Gated Crypto, TLS Web Server Authentication,
        TLS Web Client Authentication
    1   Microsoft Server Gated Crypto, Netscape Server Gated Crypto,
        TLS Web Server Authentication
```

6.2. XML Security

6.2.1. Einführung

Das World Wide Web Consortium (W3C) veröffentlichte 2002 einen Standard um XML Dateien zu signieren. Die aktuellste Version ist aus dem Jahr 2008 [67]. Das Dokument beschreibt Regeln wie XML Dateien oder Teile davon signiert und auf ihre Signatur überprüft werden können. Der Standard definiert einen neuen XML Namespace `xmlns=http://www.w3.org/2000/09/xmldsig#`. [67]

6.2.2. Aufbau einer XML Signatur

Die Signatur ist folgendermassen im Standard [67] definiert, die Kommentare wurden ergänzt:

```
<Signature ID?>                                <!-- Signatur Tag -->
  <SignedInfo>                                  <!-- Diese Information wird signiert -->
    <CanonicalizationMethod/>                  <!-- Wie SignedInfo normalisiert wurde -->
    <SignatureMethod/>                         <!-- Algorithmus fuer die Signatur -->
    (<Reference URI? >                         <!-- Was signiert wird -->
      (<Transforms>)?                          <!-- Eventuelle Anpassungen -->
      <DigestMethod>                           <!-- Hashtyp -->
      <DigestValue>                            <!-- Hash -->
    </Reference>)+
  </SignedInfo>
  <SignatureValue>                             <!-- Signatur -->
  (<KeyInfo>)?                                 <!-- Welcher Schluessel verwendet wurde -->
  (<Object ID??>)*                             <!-- Zusaeztliche Infos -->
</Signature>
```

Listing 6.5: Struktur einer XML Signatur [67]

Hierbei bedeutet ? ein optionales Element, + einmaliges oder mehrmaliges Vorkommen und * kein bis mehrmaliges Vorkommen. Die Tags haben folgende Bedeutung [67]:

- Im `Signature` Tag ist die ganze Signatur enthalten.
- Im `SignedInfo` Tag sind die Informationen, welche signiert werden.
- Im `CanonicalizationMethod` Tag ist der Algorithmus zu finden, mit welchem die Daten innerhalb vom `SignedInfo` Tag normalisiert (canonicalized) werden.
- Im `SignatureMethod` Tag steht der Algorithmus, mit welchem der normalisierte `SignedInfo` Inhalt in den `SignatureValue` umgewandelt wird. Dieser Wert besteht aus einer Hash- und einer Verschlüsselungsfunktion. Dieser Wert wird signiert, damit die Signatur resistent gegen eine Downgrade Attacke ist.
- In jedem `Reference` Tag steht der Hashtyp (`DigestMethod`) und der resultierende Hash (`DigestValue`).
- Eventuell wurde vor dem Hashen noch Anpassungen vorgenommen. Diese Operationen stehen im `Transforms` Tag.
- Im `KeyInfo` Tag steht der Key, welcher für die Validierung der Signatur verwendet werden soll. `KeyInfo` befindet sich ausserhalb von `SignedInfo` und ist deshalb selber nicht signiert. Dieses Tag kann aber durch einen weiteren `Reference` Eintrag signiert werden.
- Im `Object` Tag können zusätzliche Informationen wie beispielsweise ein Zeitstempel angegeben werden.

Zur besseren Visualisierung wird der XML Baum für eine Signatur folgend als Baum dargestellt:

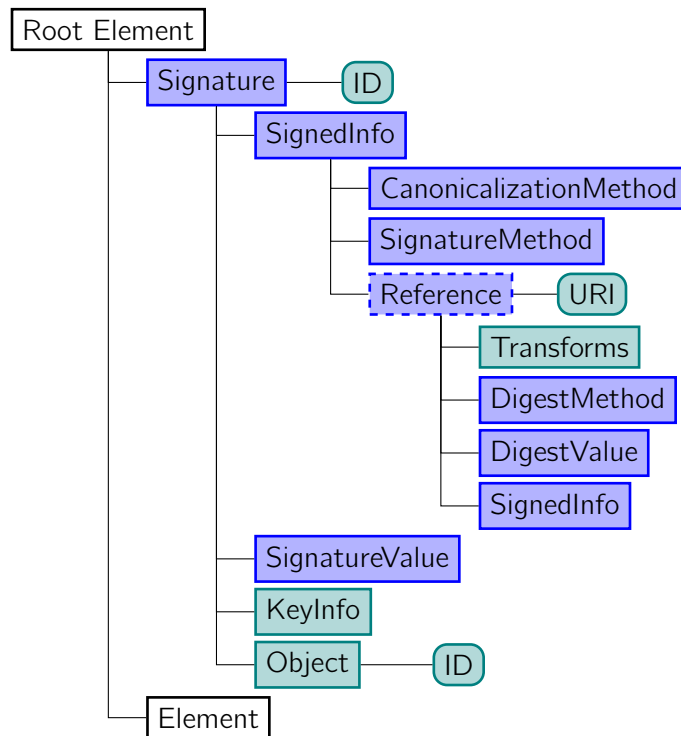


Abbildung 6.1.: Aufbau einer XML Signatur

Die farbigen Elemente sind Teil der Signatur, wobei runde Elemente Attribute darstellen. Die grünen Elemente sind optional und gestrichelte dürfen mehrmals vorkommen. Die Legende dazu ist auch im Abschnitt 4.3 zu finden.

Es gibt drei Arten um signierte Informationen in einer XML Datei zu speichern:

Enveloping Signatur

Bei der Enveloping Signatur werden die signierten Daten Data innerhalb der Signatur Signature abgelegt [67].

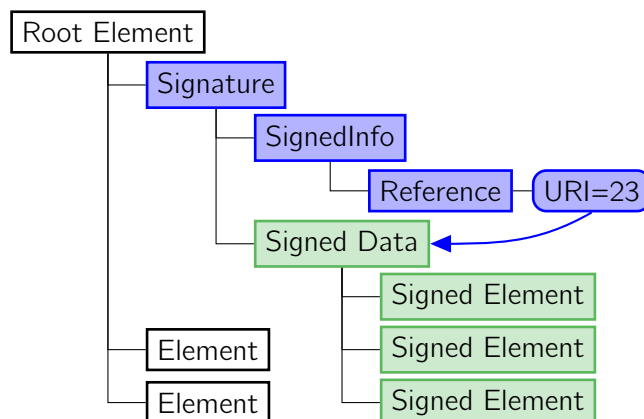


Abbildung 6.2.: Enveloping XML Signatur [67]

Enveloped Signatur

Bei der Enveloped Signatur wird die Signatur *Signature* innerhalb der signierten Daten abgelegt [67].

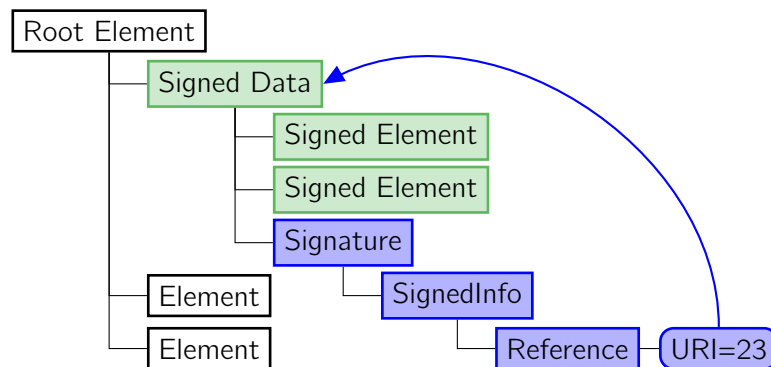


Abbildung 6.3.: Enveloped XML Signatur [67]

Detached Signatur

Bei der Detached Signatur sind die Signatur und die signierten Dateien komplett voneinander getrennt [67]:

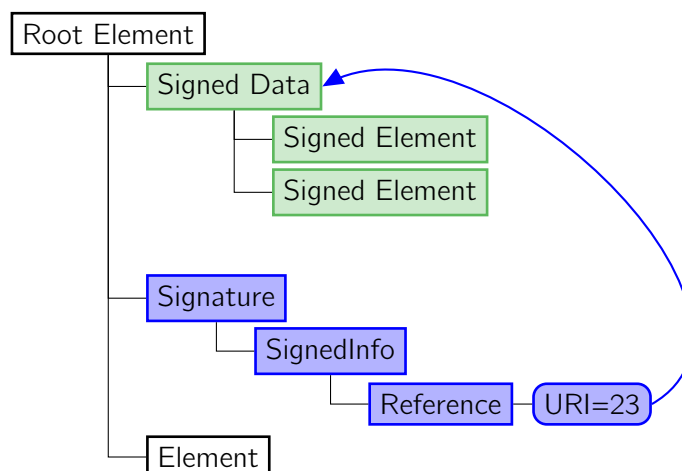


Abbildung 6.4.: Detached XML Signatur [67]

6.2.3. SAML Assertions signieren

Der SAML Standard sagt, dass SAML Assertions welche von einer Asserting Party ausgestellt werden auch signiert sein sollten. Sofern ein Profile keine andere Vorgabe macht, muss die XML Signatur nach dem enveloping Verfahren erstellt werden. [30]. "Sollte" und "muss" ist hier im Sinne von RFC 2119 [15] gemeint. Somit sieht eine Assertion mit entsprechender XML Signatur folgendermassen aus:

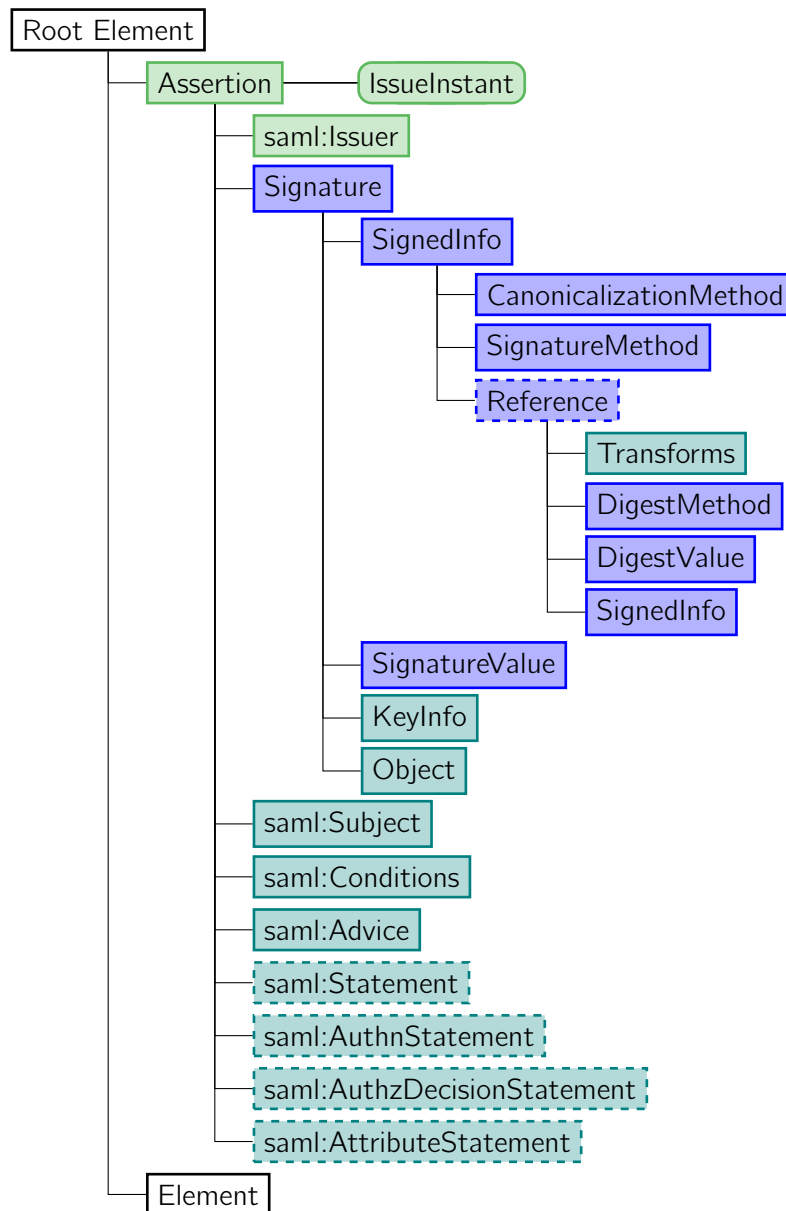


Abbildung 6.5.: Aufbau einer SAML Assertion [67]

6.2.4. Beispiel: XML Signieren

An folgendem Beispiel wird das Signieren einer XML Datei gezeigt. In der XML Datei soll das Element `/root/element/item[@id=secure]` signiert werden.

```

<?xml version="1.0"?>
<root>
  <element>
    <item>Content Item 1</item>
    <item id="secure">Ich bin signiert</item>
  </element>
</root>

```

Listing 6.6: examples/X.509/test.xml

Dazu wird ein Signature Template in das XML eingebettet, damit das Tool `xmlsec1` aus dem Paket `xmlsec` [8], welches zum Signieren verwendet wird, weiss wo die Signatur platziert werden muss. Dieser Schritt wird "von Hand" gemacht, also manuell eingefügt.

```
<?xml version="1.0"?>
<root>
  <element>
    <item>Ich bin ein Item.</item>
    <item id="signed">Und ich bin signiert</item>
  </element>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="#signed">
        <Transforms>
          <Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <DigestValue/>
        </Reference>
      </SignedInfo>
      <SignatureValue/>
      <KeyInfo>
        <KeyValue/>
      </KeyInfo>
    </Signature>
  </root>
```

Listing 6.7: `examples/X.509/test_signature_template.xml`

Dann kann folgendermassen mit dem Schlüsselmaterial von `bauser` das XML signiert werden [8]:

```
$ xmlsec1 --sign --privkey-pem bauserkey.pem --output test_signed.xml \
  --id-attr:id item test_signature_template.xml
```

Das signierte XML sieht danach so aus:

```
<?xml version="1.0"?>
<root>
  <element>
    <item>Ich bin ein Item.</item>
    <item id="signed">Und ich bin signiert</item>
  </element>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="#signed">
        <Transforms>
          <Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <DigestValue>RAB0cS5nPolkUUqng8a5/QSX1U8=</DigestValue>
        </Reference>
      </SignedInfo>
```

```

    <SignatureValue>lQZRvAd6D8IYvIzA7xZ4SJP+kg1yUHkmBT2DMx4rXsp9/
      i6T9rNCpktgLgjWZVer
r2TwVUyL1h1LkKEYw8JG0J1bN2Ek1jZQ6znozokDBla7TI8/j+z/TFBAVRJLHSh4v
6M0bFuB/Zp+KJ9P5jg8arH060n+/BDcvBawoGf3XGNzZaU05i0HyHegXQnqBvShi
XfBDbMoTlirM3onGpU4BZyhBccR955SrIYlFAHiK2fweseY7YTXRkf4yjnKC0i18
ep5FxtgCcgvTV2oo8X7rHg7PUdYhbFnWIORYTsVw19NtHbbl3qU1gWTWsaMWAV3P
X1xJyBcXxoA2zGYXPv58pk8YfpxFpPVsW8gMGxcQXGDdgTKECg/Euq3W+F5nRV02
FLP58S+4VVI7DM3x4XpL2qiJWh2HArDbIDXFwBvUCkM6Tvz7QXBtLn1X9+NgZEkm
WKyNopR760mXKIpHFkSp6Wnqh1BLb75TXc71AhriJITgSz8dfXqQVK1PKFmt5/KM
KsqL/g5eZQ6kYvx31q3JrhcdOFiSsu/mG0AHMChbU0bkyf/PbCW1Fj+Dq7dn/pc1
fKavocLgOdoAZrubH2Hu6/tsBihfdv08y0oZtLKabkTqec4ZZ2S0Ewxh/C4kEwKW
15R344ZIUd6qCKHu0q2qg+dEgkgkQ7MmRfBkJ4xaHX0=</SignatureValue>
  <KeyInfo>
    <KeyValue>
<RSAKeyValue>
<Modulus>
w56UU7WUK6fJ7iEGXpkA+o5xUYFu6wYeUrcUd5mmSkYlJKGaZeIHSgpWryeIDqub
9INxzhC2a47xs5vU3bv3amGtMPaMe+aY3s0Qd4A0GT8zhcsNOd1Gu4QXaKnpJw80
grMpsGf0Xk+IGQmocQiHxbXtM3+rnyq2qmtfty2DS9a+RG+ZhSYPbJs4dnHpa/Er
uw0hGU04qIK4q50dU6Kd8VFMEkC164ky2vCNx+yajZE+DoHyI4wm2YoL8eJ838Ji
JvxEIqeskJibi330Eic+GksqJuUMsF4qZ/XJQa0dWn0Cf95HGeqy7JXK0yHpr75
XRdh1bsAPf3AYMw+3+eIAsCRWstizLB7FexibwftLKn0t7Dg1tyQff2yU2fqze0a
tKIZXTn0sNIBji1wbgvflBII9WH4guNrfQaB71pilbpC5NxpWKbwTvmv0dftXD9W
1/WlNJe2ukQ1UpWG4K3tM0ofoNKw/nVAuiJFVIS61LLOWDrHxBSUBzIanYNfZxI4
tWwnJLlqQ3knnF8I/jMzFN01XP3dpiIB8i/f/2PLq4x+NyBh+yPhPXJ4brHD+LIq
0Jxyo8Zd7YZlL19qBcU3nG13KqvZDDim4BnWYukqCf4eq6xsPQnyWqV0y9E+XgaB
WqCWixDp00NP2e9fhekT29+0ZxGGcSP9qM+4dNzQxsM=
</Modulus>
<Exponent>
AQAB
</Exponent>
</RSAKeyValue>
</KeyValue>
  </KeyInfo>
</Signature>
</root>

```

Listing 6.8: examples/X.509/test_signed.xml

6.2.5. Schlechte Algorithmen

Die ECRYPT II gibt empfehlungen für Sicherheitslevels ab. Für eine Longterm Protection müssen laut den ECRYPT II Empfehlungen die Anforderungen in der untenstehenden Liste erfüllt sein. Die Longterm Protection von ECRYPT II bedeutet eine anwendungsunabhängige Empfehlungen bis ins Jahr 2040. Diese Informationen stammen von der Empfehlungsseite <http://www.keylength.com>. [12]

- Die Schlüsselgröße für RSA Schlüssel, muss mindestens 3248 Bit betragen.
- Als Hashsumme muss mindestens eine Länge von 256 Bit verwendet werden (Beispielsweise SHA-2 mit 256, 384 oder 512 Bit).
- Schlüsselgröße für Elliptische Kurven müssen mindestens 256 Bit betragen.

6.3. Schlussfolgerung

Eine wichtige Anforderung bei SAML ist, dass die Assertions integer und authentisch sind. SAML stellt die Integrität und Authentizität von Assertions über eine XML-Signatur und dabei auch über X.509 Zertifikate sicher. Die Zertifikatskette bis zu einem vertrauten Zertifikat sorgt dafür, dass der Assertion vertraut werden kann.

Somit ist es ausserordentlich wichtig, dass die Überprüfung der Signatur und des Zertifikats inklusive der gesamten Zertifikatskette korrekt durchgeführt wird.

7. Angriffe auf SAML

In diesem Kapitel wird beschrieben, welche Angriffe auf SAML möglich sind. Dabei wird jeweils aufgeführt was bei diesem Angriff ausgenutzt wird, was bereits vor dem Angriff vorhanden sein muss, der Ablauf des Angriffs und auch was das Ergebnis eines erfolgreichen Angriffs ist. Des Weiteren wird beschrieben, wie der Angriff verhindert werden kann. Diese Angriffe sind jeweils Szenarien die möglich wären, wenn ein Teil der Sicherheitsmechanismen nicht korrekt implementiert wurden.

Im Abschnitt 7.1 werden mögliche Angriffe auf die von SAML verwendete X.509 Infrastruktur beschrieben. Der Abschnitt 7.2 befasst sich mit Angriffen auf XML Signaturen und im Abschnitt 7.3 werden potentielle Angriffe auf SAML selbst beschrieben.

7.1. Angriffe auf die X.509 Public Key Infrastructure

Die Angriffe auf die X.509 Public Key Infrastruktur konzentrieren sich auf Fehler in der Überprüfung des Zertifikats, welches zur Signierung benutzt wurden.

7.1.1. Use Cloned Self-signed Certificate

Das Zertifikat, mit welchem die XML Signatur erstellt wurde, wird geklont. Das heisst, dass die Felder exakt den gleichen Inhalt haben müssen. Bei diesem Zertifikat handelt sich um ein selbst signiertes Zertifikat, was bedeutet, dass das Zertifikat sich selbst signiert hat und keine CA dies vorgenommen hat.

Vorbedingungen

Ein gültiges Zertifikat eines IdPs zum Klonen ist vorhanden.

Ablauf der Attacke

1. Alle Zertifikatsfelder werden kopiert.
2. Zertifikat wird durch sich selbst signiert.
3. SAML Nachricht mit diesem Zertifikat signieren und Zertifikat in XML einbinden.
4. Editierte Nachricht an SP senden.

Erwartetes Resultat

Der Angriff ist erfolgreich, wenn die selbst signierte SAML Nachricht akzeptiert wurde.

Gegenmassnahme

Es sollen nur Zertifikate einer gewissen CA akzeptiert werden. Ausserdem muss die Zertifikatskette geprüft werden und nicht nur ob die Felder in Ordnung sind.

7.1.2. Use Cloned Certificate Chain

Ähnlich wie im vorherigen Angriff, soll ein Zertifikat geklont werden. Nun wird jedoch die gesamte Zertifikatskette kopiert.

Vorbedingungen

Eine gültige Zertifikatskette eines IdPs zum Klonen ist vorhanden.

Ablauf der Attacke

1. Root Zertifikat kopieren.
2. Root Zertifikat wird durch sich selbst signiert.
3. Nächst unterstes Zertifikat kopieren und durch Eltern Zertifikat signieren lassen.
4. Den vorherigen Schritt so lange wiederholen bis kein Kindzertifikat in der Kette mehr verfügbar ist.
5. SAML Nachricht mit dem untersten Zertifikat signieren und Zertifikat in XML einbinden.
6. Editierte Nachricht an SP senden.

Erwartetes Resultat

Wenn die selbst signierte SAML Nachricht akzeptiert wird, ist der Angriff erfolgreich.

Gegenmassnahme

Es müssen nicht nur die Zertifikatsfelder sondern auch die gesamte Zertifikatskette bis zum Root auf Gültigkeit überprüft werden.

7.1.3. Use valid Certificate of other CA

Bei diesem Angriff wird ein gültiges Zertifikat, welches jedoch von einer anderen CA wie der erlaubten unterschrieben wurde, benutzt. Dabei ist das CA Zertifikat von einer als allgemein vertrauenswürdig eingestuften CA unterschrieben worden.

Vorbedingungen

Das gültige Zertifikat ist vorhanden.

Ablauf der Attacke

1. mit gültigem Zertifikat die SAML Assertion signieren.
2. SAML Nachricht an SP senden.

Erwartetes Resultat

Wenn die selbst signierte SAML Nachricht akzeptiert wird, ist der Angriff erfolgreich.

Gegenmassnahme

Es dürfen nur bestimmte CAs als gültige CAs akzeptiert werden.

7.1.4. Use valid Certificate with incorrect Key Usage

Die SAML Assertion soll mit einem Zertifikat signiert werden, welches jedoch nicht für Signaturen von XML Nachrichten erlaubt wäre. Das heisst, dass man SAML Nachrichten z.B. mit einem S/MIME E-Mail Zertifikat, das von einer vertrauten CA stammt, signieren könnte.

Vorbedingungen

Ein Zertifikat ist vorhanden, dessen Usage es nicht erlaubt zu signieren und das von einer vom SP akzeptierten CA stammt.

Ablauf der Attacke

1. Mit dem Zertifikat die SAML Assertion signieren.
2. SAML Nachricht an SP senden.

Erwartetes Resultat

Wenn die durch das unerlaubte Zertifikat signierte SAML Nachricht akzeptiert wird, ist der Angriff erfolgreich.

Gegenmassnahme

Die erlaubten Zwecke des Zertifikats müssen überprüft werden.

7.1.5. Use revoked Certificate

Bei diesem Angriff wird ein revoziertes Zertifikat zum unterzeichnen der Nachricht benutzt.

Vorbedingungen

Das revozierte Zertifikat ist vorhanden.

Ablauf der Attacke

1. Mit dem revoziertem Zertifikat die SAML Assertion signieren.
2. SAML Nachricht an SP senden.

Erwartetes Resultat

Wenn die mit dem revozierten Zertifikat signierte SAML Nachricht akzeptiert wird, ist der Angriff erfolgreich.

Gegenmassnahme

Das Überprüfen auf revozierte Zertifikate muss korrekt implementiert sein.

7.2. Angriffe auf die XML Signaturen

Die folgenden Angriffe auf XML Signaturen bezüglich SAML wurden von Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen im Jahr 2012 am 21st USENIX Security Symposium unter dem Titel "On Breaking SAML" vorgestellt. [62]

Bei den folgenden Angriffen wird als Vorbedingung meist eine gültige SAML Assertion benötigt. Diese kann auf verschiedene Arten besorgt werden. Man kann sich bei einem Identity Provider registrieren und erhält somit eine SAML Assertion, welche für sich selbst ausgestellt wurde. Durch Man-in-the-Middle-Angriffe können Assertions abgefangen werden. Durch Google Hacking, also durch das Suchen über eine normale Suchmaschine, kann man ebenfalls an Assertions gelangen [62].

7.2.1. Signature Exclusion Attack

Aus einer gültigen SAML Nachricht wird die Signatur vollständig entfernt. Damit soll ausgenutzt werden, dass möglicherweise nur eine Überprüfung der Signatur stattfindet, wenn diese auch vorhanden ist. Ansonsten wird die SAML Nachricht einfach als gültig gekennzeichnet [62].

Vorbedingungen

Eine korrekte SAML Assertion ist vorhanden oder kann konstruiert werden. Diese kann folgendermassen aussehen:

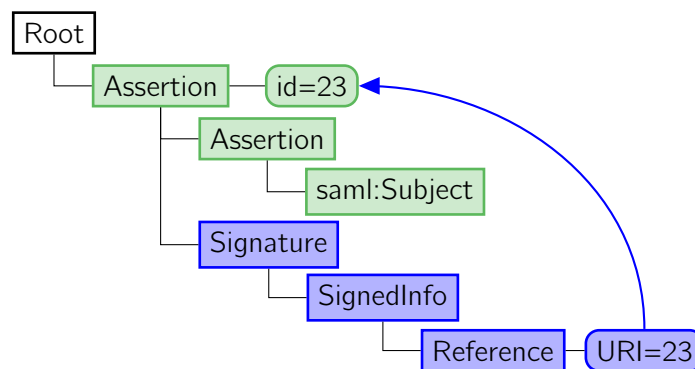


Abbildung 7.1.: SAML Assertion vor der XML Signature Exclusion [30]

Ablauf der Attacke

1. Die XML Signatur wird aus der Nachricht entfernt oder eine neue Nachricht ohne Signatur wird generiert.
2. Optional: Die SAML Nachricht kann bearbeitet werden. Beispielsweise wird das SAML Subject verändert.
3. SAML Nachricht wird an den Service Provider gesendet und von dem akzeptiert.

Erwartetes Resultat

Wenn die SAML Nachricht ohne Signatur akzeptiert wird, ist der Angriff erfolgreich. Die SAML Assertion ohne Signatur sieht folgendermassen aus:

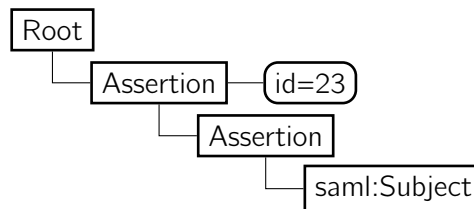


Abbildung 7.2.: SAML Assertion nach der XML Signature Exclusion Attack

Laut dem Forscherteam waren folgende Frameworks auf die Signature Exclusion Attacke anfällig: [62]

- Apache Axis 2
- JOSSO
- OpenAthens

Gegenmassnahme

SAML Nachrichten ohne Signatur dürfen nicht akzeptiert werden.

7.2.2. XML Signature Wrapping Attack (XSW)

Erhält ein Service Provider eine SAML Assertion, durchläuft diese nacheinander zwei Schritte: [62]

1. Signature Verification: In der XML Signatur wird nach der URI gesucht und geprüft, ob die Signatur des XML Teilbaums, welche mit der URI referenziert wird, gültig ist. Ist die Signatur gültig, kommt der nächste Schritt.
2. Assertion Evaluation: Unabhängig vom ersten Schritt wird die SAML Assertion ausgewertet.

Das Problem dabei ist, dass nicht sichergestellt wird, dass nur die signierte Assertion evaluiert wird. Durch die Signature Wrapping Attacken wird eine zusätzliche SAML Assertion in den XML Baum eingefügt, welche im zweiten Schritt ausgewertet wird. Viele verschiedenen SAML Frameworks haben in den Tests der Forschungsgruppe zwar die Signatur korrekt auf ihre Richtigkeit überprüft, aber danach im zweiten Schritt eine andere, zusätzliche, Assertion ausgewertet. Die Forschergruppe hat über 1000 Permutationen bei 14 Produkten getestet. Dabei waren nur zwei der 14 Produkte gar nicht verwundbar. [62]

Vorbedingungen

Eine korrekte und gültig signierte SAML Assertion ist vorhanden oder kann konstruiert werden. Diese sieht in diesem Beispiel wie im Unterabschnitt 7.2.1 in Abbildung 7.1 aus.

Ablauf der Attacke

1. Die vorhandene Assertion wird im XML Baum verschoben.
2. Die vorhandene XML Signatur wird im XML Baum verschoben.
3. Es wird eine zusätzliche Assertion eingefügt.
4. Die neue SAML Assertion wird an den Service Provider verschickt.

Die ersten zwei Schritte sind optional und werden je nach Implementation der Software des Service Providers anders vorgenommen. Mit den folgenden einfachen Permutationen der XML Elemente hat das Forschungsteam die folgenden XSW Attacks entdeckt. Wir nennen diese Angriffe in unserer Arbeit "Common XSW" mit einer aufsteigenden Nummer, damit wir später im Dokument besser darauf Bezug nehmen können. [62]

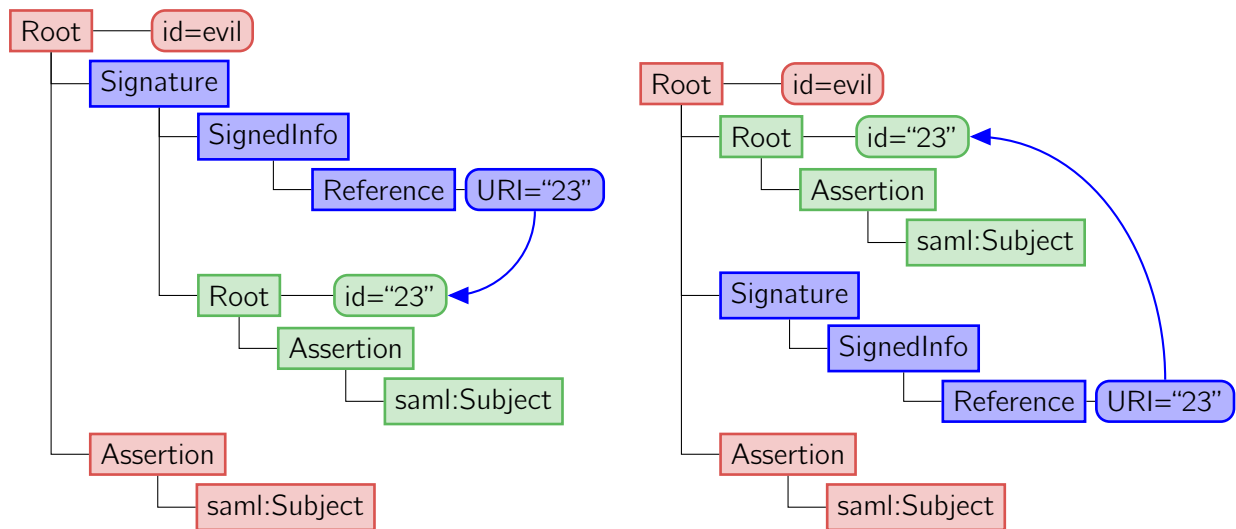


Abbildung 7.3.: Common XSW Nr. 1 gegen Guany Abbildung 7.4.: Common XSW Nr. 2 gegen WSO2 und JOSSO [62]

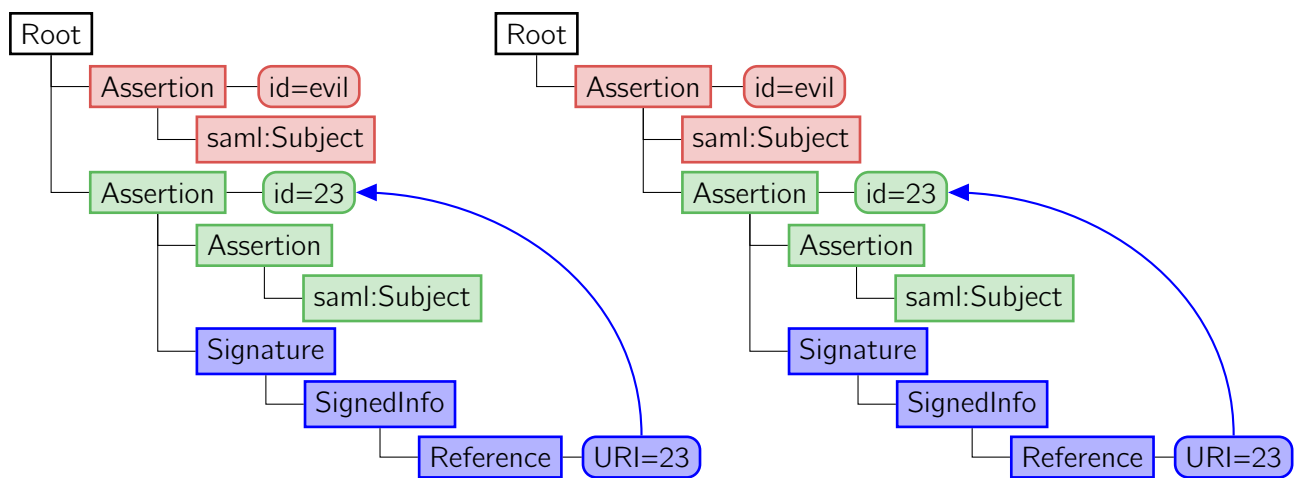


Abbildung 7.5.: Common XSW Nr. 3 gegen Higg- Abbildung 7.6.: Common XSW Nr. 4 gegen Higg- ins, Apache Axis 2 und IBM XS40 [62]

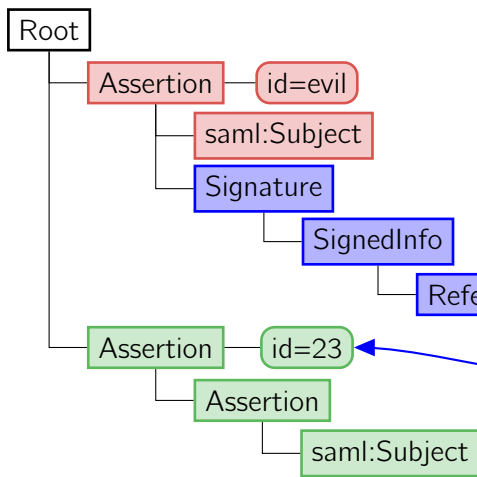


Abbildung 7.7.: Common XSW Nr. 5 gegen OIOSAML [62]

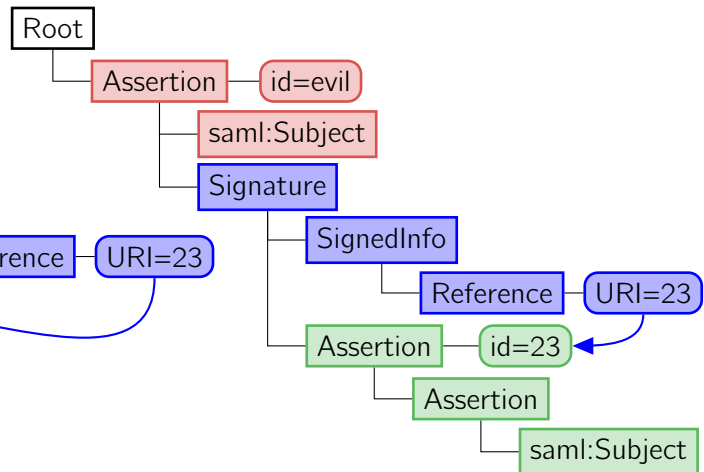


Abbildung 7.8.: Common XSW Nr. 6 gegen OpenAM und Salesforce [62]

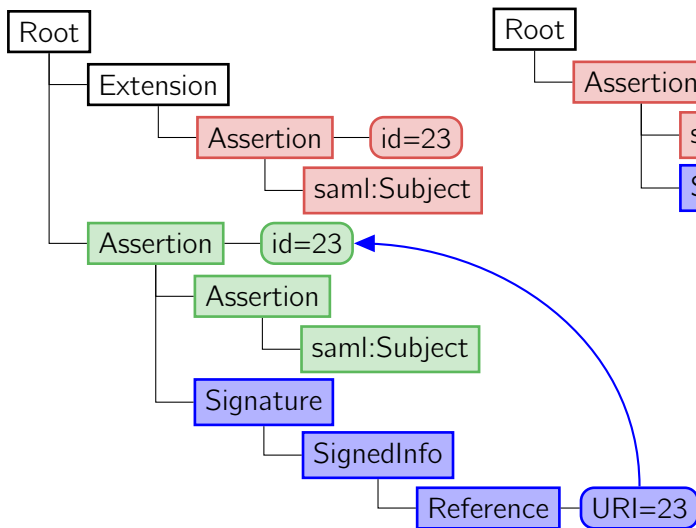


Abbildung 7.9.: Common XSW Nr. 7 Attacke gegen OpenSAML in C++ [62]

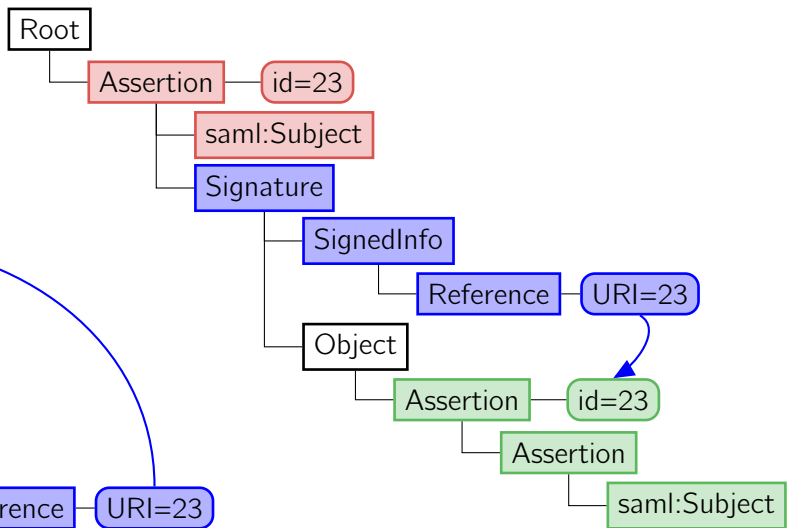


Abbildung 7.10.: Common XSW Nr. 8 Attacke gegen OpenSAML in Java [62]

Gegenmassnahmen

OpenSAML hat geprüft, ob die evaluierte Assertion dieselbe ID hat wie die signierte. Durch eine XML Schema Validierung wurde sichergestellt, dass jede ID nur einmal vorkommt. Das Forscherteam konnte jedoch durch XML Erweiterungen trotzdem eine zusätzliche Assertion mit gleicher ID hinzufügen. Das Team empfiehlt folgende Gegenmassnahmen: [62]

1. See-what-is-signed: Alles was nicht signiert ist, aus dem XML Dokument löschen. So wird nur der wirklich signierte Inhalt weiterverarbeitet.
2. Unique Identification (Tainting) of Signed Data: Alle signierten Elemente mit einem neuen Attribut versehen, welches mit einer Zufallszahl initialisiert wurde. Diese Zufallszahl soll in den

weiteren Verarbeitungsschritten wieder überprüft werden. Da dies aber das XML Schema von den Assertions verletzen würde, muss man das Schema erweitern.

7.3. Angriffe auf SAML

7.3.1. Logout other Users

Sollte es einem Angreifer möglich sein die Session ID und auch den User beim AuthResponse auszu-lesen, könnte er versuchen den User auszuloggen.

Vorbedingungen

Eine Authentication Response ist vorhanden.

Ablauf der Attacke

1. SessionID und User aus Authentication Response auslesen
2. Einen Logout Request erstellen
3. Logout Request an IdP senden

Erwartetes Resultat

Der Angriff ist erfolgreich, wenn der User aus allen oder einem Teil der Sessions ausgeloggt wird. Das heisst der Logout Response enthält den Status Success.

Gegenmassnahme

Nur signierte Logout Requests akzeptieren.

7.3.2. Replay Attacke

Sollte es einem Angreifer möglich sein eine SAML Nachricht abzufangen, kann er versuchen diese nochmals an den Service Provider zu senden.

Vorbedingungen

Eine SAML Nachricht eines IdP ist vorhanden.

Ablauf der Attacke

1. Nachricht des IdP kopieren.
2. Die kopierte Nachricht an SP senden.

Erwartetes Resultat

Der Angriff ist erfolgreich, wenn die kopierte SAML Nachricht akzeptiert wurde.

Gegenmassnahme

Mit einem Identifier, der auch signiert ist, soll sichergestellt werden, dass eine SAML Nachricht nur einmal akzeptiert wird.

8. Analyse

Dieses Kapitel definiert die Eigenschaften der Benutzer des Plugins und Anforderungen an die Software. In diesem Kapitel wird im Abschnitt 8.1 die Benutzercharakteristik beschrieben. Der Abschnitt 8.2 bietet eine Beschreibung wie der Benutzer das Plugin benutzt und in welchem Scope er dies tut. Die Requirements an das Plugin und die dazugehörigen Use Cases die das Plugin abdecken soll, werden in den Abschnitten 8.3 und 8.4 behandelt.

8.1. Benutzercharakteristik

Der typische Benutzer für unser Plugin ist ein Penetration Tester. Dieser besitzt bereits das technische Wissen über die Technologien und versteht die Hintergründe und Zusammenhänge. Er weiss was X.509 Zertifikate, XML Signature, SAML Nachrichten und XSW Attacken sind. Deshalb ist eine Anleitung zu dem Plugin nicht nötig. Der Benutzer kennt ausserdem bereits die Burp Suite und kennt deren Eigenschaften und die Bedienung derer. Typischerweise hat der Benutzer vorher dieselben Aufgaben ohne die Burp Suite durchgeführt und darf sich jetzt mit dem Plugin unterstützen lassen.

8.2. Workflow und Scope des Actors

Der Actor ist ein Penetration Tester, welcher das Plugin im Burp Intercepting Proxy verwendet. In dem Sequenzdiagramm 8.1 ist zu sehen, wie ein typischer Benutzer beim Testen von SAML Installationen mittels unserem Plugin vorgeht.

Der Penetrationstester hat mit Burp die Möglichkeit ausgehende Nachrichten, welche vom Browser an den Service Provider und Identity Provider geschickt werden, mit Burp abzufangen und manipuliert zuzustellen. Dies passiert an vier Stellen:

1. Die Login Seite wird vom Service Provider angefragt.
2. Dem Identity Provider wird gesagt, dass man sich einloggen möchte.
3. Dem Identity Provider werden die Zugangsdaten zugestellt.
4. Die Assertion, welche man vom Identity Provider erhalten hat, wird dem Service Provider zugestellt.

Die ersten beiden Punkte sind gar nicht kritisch. Dort hat man keine Möglichkeit durch eine manipulierte Anfrage an eine gültige Assertion zu kommen. Bei der dritten Stelle könnte man durch ausprobieren von Benutzernamen und Passwort eine gültige Assertion erhalten. Dies ist jedoch nicht Ziel unserer Arbeit.

Unsere Arbeit befasst sich ausschliesslich mit dem vierten Punkt, den Angriffen auf Service Provider. Burp sendet die Assertion an den Service Provider, welcher die Signatur auf ihre Gültigkeit überprüft und anhand der Assertion entscheidet, ob die Person für die angeforderte Seite autorisiert ist. Unser Burp Plugin soll diese Assertion so manipulieren können, dass ein verwundbarer bzw. ein falsch konfigurierter Service Provider die gefälschte Assertion als gültig anerkennt.

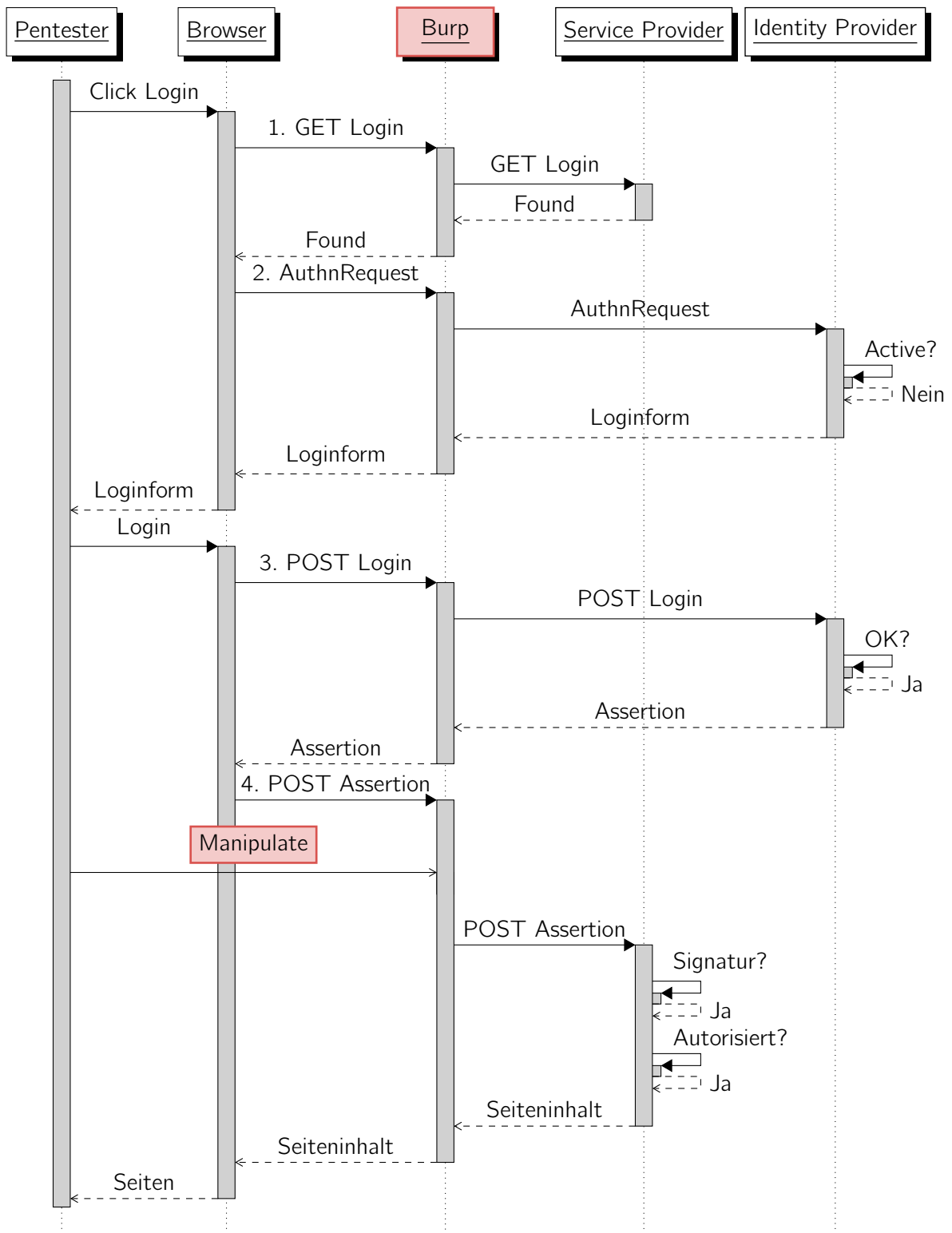


Abbildung 8.1.: Workflow eines Pentesters beim Testen einer SAML Installation

Wenn in den folgenden Beschreibungen von einer SAML Nachricht die Rede ist, ist die SAML Nachricht mit einer Assertion von Punkt 4 aus der Abbildung 8.1 gemeint.

8.3. Requirements

8.3.1. Functional Requirements

Durch ein Brainstorming des BA Teams, während einer Sitzung mit unserem Betreuer und mithilfe von Unterlagen der Compass Security wurden folgende Requirements zusammengestellt und priorisiert:

Nr.	Name	Beschreibung	Priorität
1	Common XSW	Die bekannten XSW Attacken [62] abbilden	High
2	Permuted XSW	XSW Permutationen zur dynamisch generieren und ausführen	Medium
3	Weak XML Signature	Schwache XML-Signaturen kennzeichnen	High
4	XML Signature exclusion	XML Signatur aus Assertion entfernen	Very high
5	Replace XML Signature	Bestehende XML Signatur durch eine neue ersetzen	Very high
6	Create XML Signature	Ein XML Element signieren	Very high
7	Clone Certificate	Zertifikat mit gleichen Inhalten klonen	Very high
8	Clone Certificate Chain	Ganze Zertifikatskette mit gleichen Inhalten klonen	Very high
9	Export Certificates	Ein Zertifikat oder die ganze Kette exportieren	Medium
10	Edit SAML Messages	SAML Nachrichten editieren	High
11	Edit Certificate	Zertifikatsfelder editieren z.B. OCSP, CRL	Low
12	Weak Certificate	Schwache Zertifikate kennzeichnen	Medium
13	Information display	Schlüsselinformationen der SAML Kommunikation darstellen	High
14	Framework Fingerprinting	Application Hersteller und Produkt identifizieren	Low
15	Artifact Binding support	Artifact Binding wird unterstützt	Low
16	POST Binding support	POST Binding wird unterstützt	High
17	Redirect Binding support	Redirect Binding wird unterstützt	High
18	SOAP Binding support	Unterstützung von SOAP Nachrichten	Medium
19	Single Logout Profile support	Single Logout Profile wird unterstützt	High
20	Webbrowser SSO Profile support	Webbrowser SSO Profile wird unterstützt	High

Tabelle 8.1.: Auflistung der functional Requirements

8.3.2. Non Functional Requirements

Folgende Non Functional Requirements sind für unser Plugin wichtig:

Nr.	Name	Beschreibung
1	Plattformunabhängigkeit	Plugin soll Plattform-Unabhängig betrieben werden können (Windows, Mac OSX und Linux)
2	Installationsfreundlich	Das JAR-Plugin soll wie andere Java Erweiterungen einfach über den Extender installiert werden können.
3	Keine zusätzlichen Libraries	Es müssen keine zusätzlichen Libraries für die Nutzung unseres Plugins installiert werden.
4	Userfreundlich	Die Benutzung des Plugins soll für einen technisch versierten Benutzer selbstverständlich sein.

Tabelle 8.2.: Auflistung der Non functional Requirements

8.4. Use Cases

Auf eine visuelle Darstellung der Use Cases mit einem Use Case Diagramm verzichtet wurde bewusst verzichtet, da es für das Plugin nur einen Actor, den Penetration Tester, gibt. Der Actor wird in den Use Cases auch nicht jedes mal erneut erwähnt, da es immer derselbe ist.

8.4.1. Use Case 01: Execute Common XSW Attack

Beschreibung

Der Actor kann eine bekannte XSW Attacke auswählen und durchführen lassen.

Precondition

Eine signierte SAML Nachricht wurde von Burp abgefangen (trapped).

Main Success Scenario

1. Der Actor wählt eine bekannte XSW Attacke aus.
2. Das Burp Plugin fügt an der entsprechenden Stelle eine zusätzliche Assertion ein um die XSW Attacke auszuführen. Je nach XSW Attacke werden zusätzlich Elemente im XML Baum verschoben.
3. Der Actor löst den Versand der neuen Nachricht aus.

8.4.2. Use Case 02: Execute permuted XSW Attacks

Beschreibung

Es werden weitere mögliche XSW Angriffe generiert. Diese sind nicht im Voraus fix einprogrammiert, sondern werden zur Laufzeit dynamisch anhand eines Algorithmus generiert.

Precondition

Eine signierte SAML Nachricht wurde von Burp abgefangen (trapped).

Main Success Scenario

1. Der Actor wählt auf einer SAML Nachricht aus, dass er permuted XSW Attacken durchführen will.
2. Das Burp Plugin generiert mehrere Assertions und sendet diese ans Ziel.

8.4.3. Use Case 03: Rate XML Signature Algorithms

Beschreibung

Der Actor sieht, ob eine XML Signatur schwache Algorithmen verwendet.

Precondition

Eine signierte SAML Nachricht wurde von Burp abgefangen (trapped).

Main Success Scenario

1. Der Actor ruft die Funktion zur Bewertung der Signatur auf.
2. Das Burp Plugin zeigt eine Bewertung der XML Signatur an (Bewertungsschema siehe Kapitel 6.2.5).

8.4.4. Use Case 04: Exclude XML Signature

Beschreibung

Der Actor kann eine Signatur aus einer Assertion entfernen.

Precondition

Eine signierte SAML Nachricht wurde von Burp abgefangen (trapped).

Main Success Scenario

1. Der Actor ruft die Funktion zur Entfernung der Signatur auf.
2. Das Burp Plugin entfernt die XML Signatur.

8.4.5. Use Case 05: Create XML Signature

Beschreibung

Der Actor kann eine XML Signatur erstellen.

Precondition

Es steht privates X.509 Schlüsselmaterial für die Signierung zur Verfügung. Eine SAML Nachricht wurde von Burp abgefangen (trapped).

Main Success Scenario

1. Der Actor wählt privates Schlüsselmaterial aus.
2. Der Actor ruft die Funktion zur Erstellung der Signatur auf einer Nachricht auf.
3. Das Burp Plugin signiert die Elemente mit dem Schlüsselmaterial.

8.4.6. Use Case 06: Replace XML Signature

Beschreibung

Eine vorhandene XML Signatur wird entfernt und durch eine selbsterstellte ersetzt.

Precondition

Es steht privates X.509 Schlüsselmaterial für die Signierung zur Verfügung. Eine signierte SAML Nachricht wurde von Burp empfangen.

Main Success Scenario

1. Siehe Use Case Exclude XML Signature
2. Siehe Use Case Create XML Signature

8.4.7. Use Case 07: Clone Certificate

Beschreibung

Ein Zertifikat wird mit gleichem Inhalt nachgebaut.

Precondition

Es ist ein bestehendes Zertifikat verfügbar.

Main Success Scenario

1. Der Actor wählt ein Zertifikat aus.
2. Der Actor ruft die Funktion zur Duplizierung des Zertifikats auf.
3. Das Burp Plugin baut ein Zertifikat mit den gleichen Feldinhalten nach. Für das neue Zertifikat wird neues Schlüsselmaterial mit der selben Schlüsselgröße generiert. Das neue Zertifikat ist self-signed.

8.4.8. Use Case 08: Clone Certificate Chain

Beschreibung

Die gesamte Zertifikatskette wird mit gleichem Inhalt nachgebaut.

Precondition

Es ist ein bestehendes Zertifikat verfügbar.

Main Success Scenario

1. Der Actor wählt ein Zertifikat aus.
2. Der Actor ruft die Funktion zur Duplizierung der Zertifikatskette auf.
3. Das Burp Plugin baut die ganze Zertifikatskette mit den gleichen Feldinhalten nach.

8.4.9. Use Case 09: Export Certificate

Beschreibung

Ein Zertifikat kann exportiert werden.

Precondition

Ein Zertifikat ist in Burp vorhanden.

Main Success Scenario

1. Der Actor wählt ein Zertifikat aus.
2. Der Actor ruft die Funktion zum Export der Zertifikate auf.
3. Das Burp Plugin exportiert das Zertifikat.

8.4.10. Use Case 10: SAML Nachrichten editieren

Beschreibung

Der Actor kann eine SAML Nachricht manuell editieren.

Precondition

Eine SAML Nachricht wurde von Burp abgefangen (trapped).

Main Success Scenario

1. Der Actor ruft die Funktion zum Editieren der SAML Nachricht auf.
2. Der Actor kann die SAML Nachricht nach belieben bearbeiten.
3. Das Burp Plugin fügt die Änderungen in die SAML Nachricht ein.

8.4.11. Use Case 11: Zertifikatsfelder editieren

Beschreibung

Der Actor kann die Felder von einem Zertifikat abändern.

Precondition

Extension 2.a: Eine eigene selbstsignierte CA ist verfügbar.

Main Success Scenario

1. Der Actor wählt ein Zertifikat aus.
2. Der Actor ruft die Funktion zum Editieren eines Zertifikats auf.
3. Der Actor editiert die Felder im Zertifikat.
4. Der Actor speichert die Änderungen, worauf ein neues Zertifikat mit den editierten Werten erstellt wird. Dieses Zertifikat ist Self-Signed.

Extension

- 3.a. Das editierte Zertifikat wird von einer eigenen selbstsignierten CA unterschrieben.

8.4.12. Use Case 12: Rate Algorithms

Beschreibung

Schwache kryptografische Algorithmen werden gekennzeichnet.

Precondition

Ein Zertifikat oder eine Signatur ist vorhanden.

Main Success Scenario

1. Der Actor ruft die Funktion zur Bewertung der Zertifikate auf.
2. Das Burp Plugin zeigt eine Bewertung der Zertifikate an (Bewertungsschema siehe Kapitel 6.2.5).

8.4.13. Use Case 13: Display Information

Beschreibung

Der Actor kann sich zusammengefasste Informationen bezüglich der SAML Nachricht anzeigen lassen.

Precondition

Eine SAML Nachricht wurde von Burp abgefangen (trapped).

Main Success Scenario

1. Der Actor ruft die Funktion zur Informationsanzeige auf.
2. Das Burp Plugin zeigt dem Actor zusätzliche Informationen zur SAML Nachricht an.

8.4.14. Use Case 14: Single Logout für User auslösen

Beschreibung

Der Actor kann aus einer Assertion einen Logout Request generieren lassen.

Precondition

Eine SAML Assertion wurde von Burp abgefangen (trapped).

Main Success Scenario

1. Der Actor ruft die Funktion zur Generierung einer Single Logout Nachricht auf.
2. Das Burp Plugin erstellt anhand einer empfangenen Assertion einen Single Logout Request.
3. Der Actor löst das Absenden der Nachricht aus.
4. Das Burp Plugin sendet die Nachricht an den IdP.

8.4.15. Use Case 15: Framework Fingerprinting

Beschreibung

Der Actor kann sich anhand einer empfangenen SAML Nachricht ein Framework Fingerprinting anzeigen lassen.

Precondition

Eine SAML Nachricht wurde von Burp abgefangen (trapped).

Main Success Scenario

1. Der Actor ruft die Funktion zur Anzeige eines Fingerprints auf.
2. Das Burp Plugin zeigt falls möglich die Applikation an, welche die SAML Nachricht versendet hat.

8.5. User Interface

Anhand der Requirements und den Use Cases wurden Überlegungen zum User Interface abgeleitet. Folgende Mockups für das User Interface wurden mit Balsamiq [63] erstellt. Das Plugin erweitert Burp um zwei Tabs.

Das erste Tab, zu sehen in der Abbildung 8.2, zeigt die dekodierte SAML Nachricht als XML Code und bietet Optionen um die Signatur der SAML Nachricht zu kontrollieren. Im rechten Teil des Tabs werden diverse Informationen zur Nachricht angezeigt.

Der Zweite Teil, zu sehen in der Abbildung 8.3, beschäftigt sich mit der Erzeugung und Verwaltung der Zertifikate.

8.5.1. SAML Tab

Das SAML Tab befindet sich unter dem Tab Proxy → Intercept. Wenn der Benutzer eine Assertion an den Service Provider schickt, wird diese durch den Burp Proxy festgehalten. Der Benutzer kann im Tab verschiedene Aktionen mit dieser Assertion durchführen. Das Tab sieht wie in Abbildung 8.2 aus.

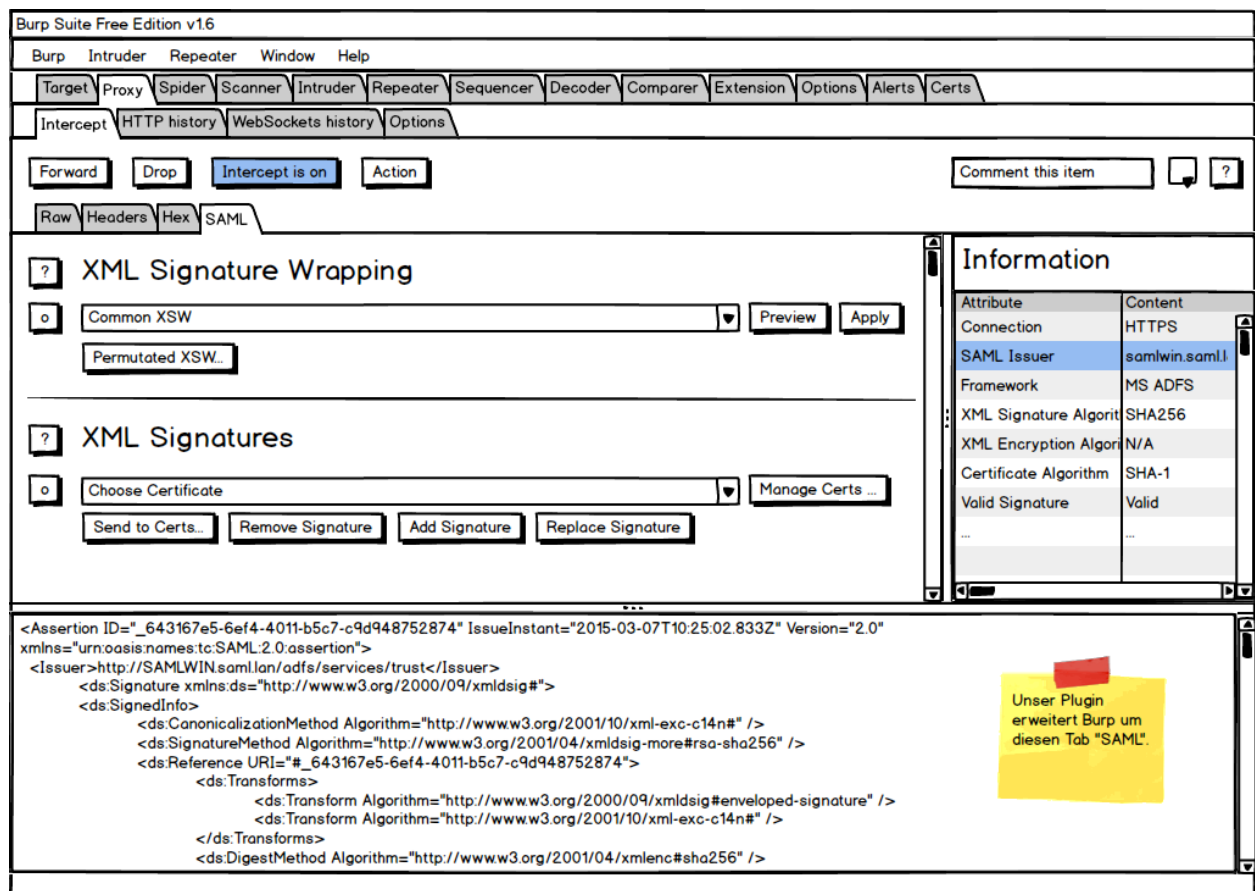


Abbildung 8.2.: User Interface Mockup vom SAML Tab

XML Signature Wrapping

Im oberen Teil können XSW Attacken ausgeführt werden. Über das Fragezeichen kann eine Beschreibung zu XSW und den möglichen Angriffen angezeigt werden. Bevor die Attacke ausgeführt wird, kann über den Button "Preview" eine Vorschau angezeigt werden, damit man die veränderte mit der originalen Nachricht vergleichen kann. Das Dropdown bietet die Optionen für die in Kapitel 7.2.2 beschriebenen XSW Angriffe. Durch einen Klick auf "Apply" wird die SAML Nachricht im unteren Teil an die ausgewählte XSW Attacke angepasst.

Durch den "permuted XSW" Button können über die Repeater Funktion von Burp mehrere XSW Attacken automatisiert ausgeführt werden.

XML Signatures

Der zweite Teil im oberen Abschnitt des UI beschäftigt sich mit den X.509 Zertifikaten der XML Signatur. Bestehende Signaturen können entfernt werden. Durch einen Klick auf "Send to Certs..." wird das Zertifikat, welches sich in der SAML Nachricht befindet, an das "Certs" Tab gesendet (Beschreibung siehe weiter unten). Falls privates Schlüsselmaterial verfügbar ist, besteht die Möglichkeit durch ein Klick auf "Add Signature" die SAML Nachricht mit dem ausgewählten Zertifikat zu signieren. Beide Schritte (löschen und signieren) können über den Button "Replace Signature" durchgeführt werden.

Information

Im linken Teil unter Informationen werden folgende Informationen angezeigt:

- SAML Issuer: Wer hat die Assertion ausgestellt?
- XML Signature Algorithm: Mit welchem Algorithmus die XML Nachricht signiert wurde.
- XML Digest Algorithm: Mit welchem Algorithmus wurde der Hash gebildet?
- XML Encryption Algorithm: Ob oder mit welchem Algorithmus die XML Nachricht verschlüsselt wurde.
- Condition Before, After: Wie lange ist die Assertion gültig?
- Subject Confirmation Before, After: Wie lange ist das Statement über das Subject gültig?

SAML Message

Zu unterst in diesem Tab sieht man die SAML Nachricht in dekodierter Form, sprich die Nachricht ist als XML lesbar. Durch die Funktionen im oberen Teil wird die Nachricht on-the-fly bearbeitet. Auch ein manuelles Bearbeiten der Nachricht ist möglich.

Beim Klick auf "Forward" wird die Nachricht wieder in die richtige Form gebracht und an das Zielsystem weitergeleitet.

8.5.2. Zertifikate Tab

Das Plugin hat viel mit X.509 Zertifikaten zu tun. Da dies keinen direkten Zusammenhang mit den übertragenden SAML Nachrichten hat, wird dies in einem separaten Tab dargestellt. Das Zertifikate Tab befindet sich in der obersten Ebene und sieht wie in der Abbildung 8.3 aus:

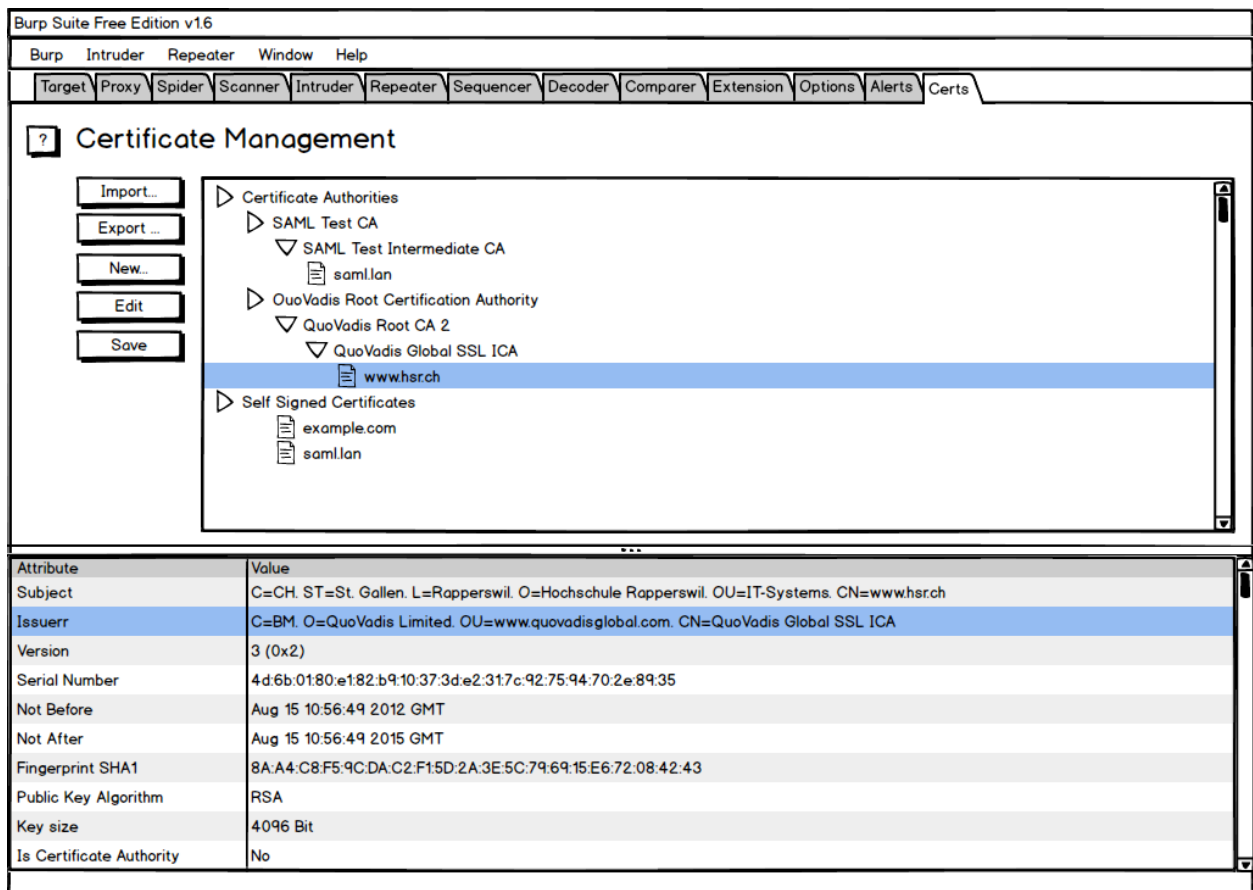


Abbildung 8.3.: User Interface Mockup vom Zertifikate Tab

Certificate Management

In der obigen Baumstruktur werden die Zertifikate hierarchisch dargestellt. Certificate Authorities können eine Intermediate CA signieren, welche wiederum ein Zertifikat beispielsweise für eine Domain signiert. Im unteren Teil werden Informationen zum selektierten Zertifikat angezeigt. Diese Informationen beinhaltet etwa dieselben Informationen, welche mit `openssl x509 -noout -text` eines X.509 Zertifikats angezeigt werden können.

Über die Buttons links des Zertifikatbaumes können Zertifikate importiert und exportiert werden. Mit "New" wird ein neues Zertifikat erstellt. Dabei kann der Benutzer im unteren Bereich die Inhalte des neuen Zertifikats selber bestimmen. Durch ein Klick auf "Save" wird neues Schlüsselmaterial generiert und daraus ein self-signed Zertifikat mit den Werten, welche im Formular stehen, generiert.

Wird der "Edit" Button ausgewählt, geschieht eigentlich das selbe wie beim erstellen eines neuen Zertifikats, mit dem Unterschied, dass das aktuell ausgewählte Zertifikat als Vorlage übernommen wird.

Zertifikate, welche selber generiert wurden und deshalb von keiner offiziellen CA, welche in den Betriebssystemen vorinstalliert ist, enthalten ist, werden entsprechend gekennzeichnet. Ebenfalls werden Zertifikate mit privatem Schlüsselmaterial gekennzeichnet.

9. Burp Suite Pluginentwicklung

9.1. Burp

Die Burp Suite (kurz Burp) ist eine Software um Webapplikationen auf ihre Sicherheit zu testen. Sie wurde von Dafydd Stuttard der Firma PortSwigger Ltd. entwickelt. Burp beinhaltet mehrere Tools, welche die Arbeit eines Penetrationstesters unterstützen. Folgende Komponenten sind in Burp enthalten: [55]

- Intercepting Proxy: Traffic mittels Man-in-the-Middle zwischen Browser und Webapplikation analysieren und modifizieren.
- Spider: Webseiten nach Inhalt und Funktionalität automatisch durchsuchen.
- Scanner: Automatische Erkennung von Verwundbarkeiten in Webapplikationen.
- Intruder: Benutzerdefinierte Attacken erstellen und spezielle Verwundbarkeiten ausnutzen. Beispielsweise können durch Fuzzing SQL Injections, Cross Site Scripting und Buffer Overflows gefunden werden, IDs nummeriert und Bruteforce Angriffe durchgeführt werden.
- Repeater: Requests manipulieren und wieder absenden. In einer History hat man Zugriff auf alle abgesendeten Requests.
- Sequencer: Zufälligkeit von Session Tokens testen, indem Statistiken über Session Tokens ausgewertet werden.

Die erledigten Arbeiten können in Burp gespeichert und wieder importiert werden. Durch den Burp Extender können eigene Plugins geschrieben werden. [55]

Von Burp gibt es zwei Versionen: Eine "Free Edition" und eine "Professional Edition". In der Free Edition ist die Intruder Komponente nur limitiert und die Scanner Komponente gar nicht vorhanden. Zudem gibt es keine Möglichkeit die bestehende Arbeit zu speichern und wieder zu laden. Weitere Unterschiede sind in der Abbildung 9.1 zu finden. [55]

9.2. Burp Extender

Mit Burp Extender kann die Burp Suite mit eigenen Funktionalitäten erweitern werden. Erweiterungen (Extensions) können in Java, Python und Ruby geschrieben werden, wobei bei Python und Ruby aus dem Code dynamisch Java Klassen generiert werden. Dies erfordert neben der Installation vom Ruby bzw. Python Interpreter, die zusätzliche Installation von Bibliotheken. Plugins, welche in Python geschrieben werden, erfordern beispielsweise die zusätzliche Installation von Jython. [53]

In einem Store unter <https://pro.portswigger.net/bappstore/> können Extensions von anderen Leuten heruntergeladen werden. Diese Extensions sind direkt in der Burp Suite installierbar. Zum heutigen Zeitpunkt sind im BAppStore 58 Plugins verfügbar. [52]

Mit der Burp Extender API kann man beispielsweise Anfragen und Antworten manipulieren, auf Laufzeitdaten wie Proxy History zugreifen, andere Komponenten starten, eigene Scans definieren, eigene Intruder Payloads schreiben, Cookies verändern oder das User Interface mit Tabs und Menüs ergänzen. Die API [54] ist als JavaDoc Seite unter <http://portswigger.net/burp/extender/api/index>.


	Free Edition	Professional Edition \$299 per user per year
Burp Proxy	✓	✓
Burp Spider	✓	✓
Burp Repeater	✓	✓
Burp Sequencer	✓	✓
Burp Decoder	✓	✓
Burp Comparer	✓	✓
Burp Intruder	?	✓
Burp Scanner	?	✓
Save and Restore	?	✓
Search	?	✓
Target Analyzer	?	✓
Content Discovery	?	✓
Task Scheduler	?	✓
Release Schedule	?	✓
	Time-throttled demo	
	Major point releases	Frequent updates, earlier releases, beta versions
	Download now 	Buy now 

Abbildung 9.1.: Vergleich der zwei Burp Versionen [55]

html verfügbar. Für die Community gibt es ein Support Center unter der URL <https://support.portswigger.net/customer/portal/topics/719885-burp-extensions/questions>, welches allerdings sehr spärlich genutzt wird und relativ klein ist. Der Entwickler Dafydd Stuttard ist auch via Twitter über den Username @PortSwigger erreichbar. [53]

Im Dezember 2012 erschien eine neue Version von Burp und die API wurde komplett überarbeitet [50]. Deshalb ist bei der Pluginentwicklung darauf zu achten, dass allfälliger bestehender Code von anderen Entwicklern oder Plugins auf der alten API basiert.

9.2.1. Eigenes Plugin in den BApp Store stellen

Da keine detaillierten Infos auf der Webseite verfügbar sind, wie eigene Plugins in den BApp Store kommen, wurde der Entwickler kontaktiert. Es ist möglich dem Entwickler den Code unseres Plugins zuzustellen, worauf er überprüft wird. Ist alles in Ordnung, wird er von ihm in den BApp Store geladen. Dies kann bis zu zwei Wochen dauern. Kommen später Updates, kann man den Entwickler erneut benachrichtigen und er lädt die neue Version in den Store. Dies geschieht schneller, da nicht mehr der ganze Code überprüft wird.

9.3. Workflow für ein simples Plugin

Eine einfache Burp Extension wird erstellt, indem man ein neues Projekt in der bevorzugten IDE erstellt mit einem Package `burp`. In dieses Package kopiert man die Interface Files von Burp und erstellt in diesem Package eine neue Klasse `BurpExtender`, welche von `IBurpExtender` erbt und die Methode `registerExtenderCallbacks` überschreibt. Dann kann man in dieser Klasse Callbacks registrieren, welche Burp bei verschiedenen Events aufruft. Als JAR exportiert kann das Plugin in die Burp Suite geladen werden. [\[51\]](#)

Im Blog von PortSwigger befinden sich unter dem Label `burp extender` einige Posts, welche sich mit der Pluginerstellung auseinandersetzen. So wird erklärt wie man ein neues Projekt aufsetzt und einfache Plugins entwickelt. [\[56\]](#)

10. Architektur

10.1. Überlegungen und geplante Architektur

Das Diagramm 10.1 gibt eine Übersicht über die Packages und Klassen, welche im Projekt verwendet werden.

Die Architektur wurde nach dem Model-View-Controller (MVC) Modell aufgebaut. Die Entscheidung wurde so getroffen, weil damit die Logik an einem Ort, sprich in den Controllern, gebündelt wird. Somit enthalten die View und das Model wenig bis gar keine Logik.

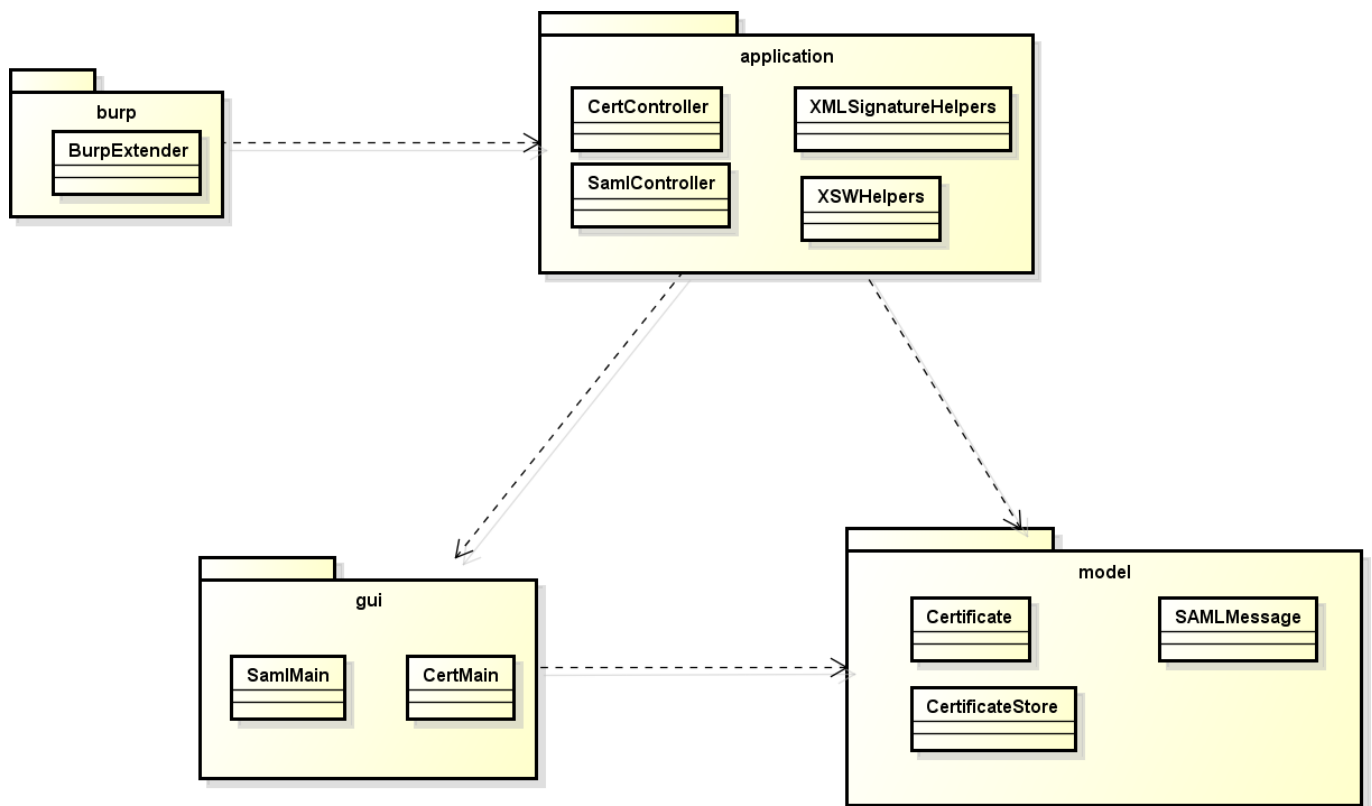


Abbildung 10.1.: Geplante Packages und Klassen

Das burp Package mit der Klasse BurpExtender ist durch den Burp Plugin Mechanismus vorgegeben.

Aufteilung MVC

Das `application` Package beinhaltet die Controller. Die Controller verwalten die Daten aus dem Model und kontrollieren die Views. Zusätzlich werden die Helper Klassen in diesem Package abgelegt, weil diese auch Logik enthalten.

Das `gui` Package beinhaltet die Klassen für die zwei Views unseres Plugin. Jeder Teil, also der SAML Message Editor und das SAML Certificate Management wird jeweils in einer eigenen Klasse abgelegt.

Das `model` Package enthält die benötigten Models. Hier werden die einzelnen Zertifikate von der Klasse `Certificate` in der Klasse `CertificateStore` abgelegt. Die einzelnen SAML Messages werden in der Klasse `SAMLMessage` gespeichert.

10.2. Implementierte Architektur

Folgende Packages mit den darin enthaltenen Klassen haben wir für das Plugin implementiert.

10.2.1. Package burp

Package, welches von Burp für ein Plugin gefordert wird, damit das Plugin überhaupt gestartet wird.

- `BurpExtender`: Dient Burp als Einstiegspunkt, Callbacks werden hier registriert.

10.2.2. Package application

Das `application` Package entspricht dem Controller Teil aus dem MVC Modell und enthält die Logik des Plugins.

- `BurpCertificateBuilder`: Factory Klasse für die Erstellung eines neuen Zertifikats. Schlussendlich wird daraus ein `BurpCertificate` geholt.
- `CertificateHelper`: Hilfsklasse, welche für die Zertifikate verwendet werden (Beispiel: Umwandeln von Hex in `BigInteger`, `ByteArray` in Hex darstellen, etc.).
- `CertificateTabController`: Logik zur Steuerung der Abläufe des Zertifikatstabs (Beispiel: Klonen eines Zertifikats.).
- `SamlTabController`: Logik zur Steuerung der Abläufe des SAML Editor Tabs (Beispiel: Nachricht signieren).
- `X509KeySelector`: Hilfsklasse für die Validierung von Signaturen.
- `XMLHelpers`: Hilfsklasse für die Manipulationen von XML.
- `diff_match_patch`: Library [18] um den Diff der XSW Attacken zu erzeugen.

10.2.3. Package gui

Das `gui` Package entspricht der View aus dem MVC Modell und enthält die User Interfaces für den SAML Editor und das SAML Certificate Management.

- `CertificateTab`: UI für das SAML Certificate Management Tab.
- `ImagePanel`: Panel um Bilder anzuzeigen und zu skalieren. Wird für die grafische Anzeige von den verschiedenen XSW Varianten verwendet.
- `SamlMain`: UI für den SAML Editor Tab.
- `SamlPanelAction`: Panel für die Aktionen, Buttons vom SAML Editor Tab.

- `SamlPanelInfo`: Panel für die Infos einer SAML Nachricht vom SALM Editor Tab.
- `SignatureHelpWindow`: Fenster zur Anzeige der XML Signatur Hilfe.
- `XSWHelpWindow`: Fenster zur Anzeige der XSW Hilfe.

10.2.4. Package model

Das `model` Package entspricht dem Model Teil aus dem MVC Modell und enthält Daten.

- `BurpCertificate`: Bildet Zertifikate ab (Wrapper für `X509Certificate` aus `java.security.cert` und speichern von zusätzlichen Attributen)
- `BurpCertificateStore`: Bildet die Beziehungen zwischen den Zertifikaten als Baum ab. Wichtig für die Darstellung von Zertifikatsketten.
- `BurpCertificateExtension`: Value Object um Extensions unabhängig von dessen Inhalt zu speichern. Dies bedeutet, dass nur die OID der Extension, ob diese als kritisch markiert ist und der Wert der Extension als Byte Array gespeichert wird. Also ohne die Bedeutung der Extension zu kennen.
- `ObjectIdentifier`: Beinhaltet verschiedene Maps mit Methoden um OIDs in dessen Namen als String und umgekehrt zu konvertieren. Beispielsweise wird aus `2.5.29.15` der Name `KeyUsage` und umgekehrt.

10.2.5. Package test

Hier befinden sich die JUnit Tests zu den einzelnen Klassen.

10.3. Vergleich zur geplanten Architektur

In der Abbildung 10.2 ist die umgesetzte Architektur mit ihren Klassen und Packages zu sehen.

Die Packages wurden wie geplant umgesetzt, wobei in der Planung das `Package test` für die Tests vergessen ging und dieses deshalb noch hinzugefügt wurde.

10.3.1. Package application

Im `application` Package kamen noch ein `CertificateBuilder`, eine `CertificateHelper` Klasse, die `Diff and Patch Library` und der `KeySelector` dazu. Diese Klassen haben sich während der Entwicklung als sinnvoll erwiesen oder waren durch Abhängigkeiten vorausgesetzt.

Beispielsweise war die `CertificateBuilder` nötig, da ein Zertifikat nach dem Hinzufügen von Attributen signiert werden muss. Somit ist es nicht möglich, ein einziges Zertifikatobjekt zu halten, welches verändert werden kann. Es muss jedes Mal ein neues Zertifikat generiert werden. Würde man ein Zertifikat nachträglich ändern, würde die Signatur nicht mehr gültig sein.

Die Klassen `XSWHelpers` und `XMLSignatureHelpers` haben wir in der Klasse `XMLHelpers` zusammengefasst.

10.3.2. Package gui

Im gui Package kamen die Panels, in welche das `SamlMainWindow` unterteilt ist dazu. Dies wurde so gemacht, damit sich das gesamte GUI nicht in einer Klasse befindet und diese riesig wird. Es wurde in der Planung ausserdem vergessen die Hilfe Fenster einzuplanen.

10.3.3. Package model

Für den SAML Message Tab ist kein Model nötig. Dies ist dadurch bedingt, da die Nachrichten von Burp selbst persistiert werden und bei jedem Aufruf des Tabs neu gesetzt werden. Somit ist das SAML Tab sozusagen stateless. Es merkt sich nur die aktuelle Nachricht und deren Eigenschaften. Die aktuelle Nachricht lässt sich gut in der in Java bereits vorhandenen Klasse `Document` speichern.

Dazu kamen noch die Klassen `BurpCertificateExtension` und `ObjectIdentifier`. Diese Klassen wurden hinzugefügt, um Funktionen und Daten an einem Ort zu kapseln und die Kohäsion hoch zu halten.

10.4. Bewertung der Architektur

Leider wurde die geplante Architektur, nicht komplett kompromisslos umgesetzt. Es wurde meist bewusst ein Auge zugedrückt, da die Machbarkeit und Implementation der schwierigsten Teile im Mittelpunkt der Entwicklung standen. Ein Refactoring einiger Teile ist bereits für nach der Arbeit geplant.

Die Abhängigkeiten der einzelnen Packages sind zum Teil bidirektional. Das heisst, dass es gegenseitige Abhängigkeiten gibt, was die Wartbarkeit vermindert. Es sollten Observer und Eventlistener eingesetzt werden um diese Abhängigkeiten zu entkoppeln.

Im Grossen und Ganzen ist die Architektur jedoch so wie geplant implementiert. Die Logik befindet sich in den Controllern und konnte somit aus dem GUI und dem Model ferngehalten werden.

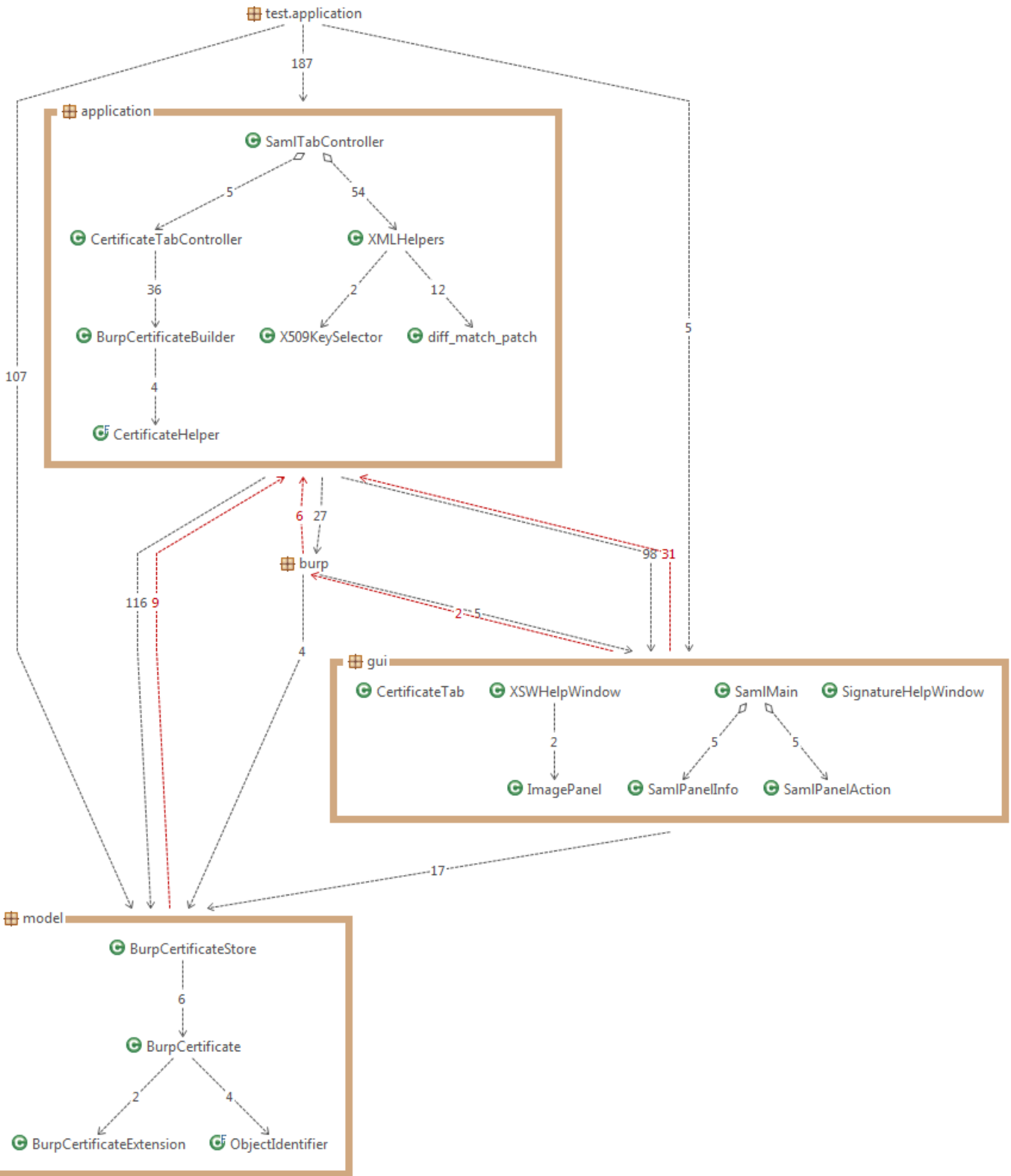


Abbildung 10.2.: Umgesetzte Architektur

11. Implementation

11.1. Vorgehensweise

Die Implementation der Features wurde auf die Projektmitarbeiter aufgeteilt. Emanuel hat die Zertifikatsverwaltung dazugehörigen Logik implementiert und Roland hat das SAML Message Editor Tab implementiert. Dabei haben wurde jeweils darauf geachtet, dass die sehr hoch priorisierten Requirements zuerst abgedeckt werden. Tiefer priorisierte Requirements wurden dann implementiert falls genügend Zeit vorhanden war und diese sich als sinnvolle Ergänzung zu den sehr hoch priorisierten Features anboten.

11.2. Entwicklungsumgebung

Das Plugin wurde in der Entwicklungsumgebung Eclipse entwickelt. Der Source Code wird mit Git in einem privaten Repository der HSR verwaltet. Emanuel entwickelt auf einem Linux System mit Arch Linux und Roland auf einem Windows System mit Windows 7.

11.3. Requirements

Geplante Functional Requirements

Nr.	Name	Priorität	Status
1	Common XSW	High	Implementiert
2	Permuted XSW	Medium	Nicht implementiert
3	Weak XML Signature	High	Nicht implementiert
4	XML Signature exclusion	Very high	Implementiert
5	Replace XML Signature	Very high	Implementiert
6	Create XML Signature	Very high	Implementiert
7	Clone Certificate	Very high	Implementiert
8	Clone Certificate Chain	Very high	Implementiert
9	Export Certificates	Medium	Implementiert
10	Edit SAML Messages	High	Implementiert
11	Edit Certificates	Low	Implementiert
12	Weak Certificate	Medium	Nicht implementiert
13	Information display	High	Implementiert
14	Framework Fingerprinting	Low	Nicht implementiert
15	Artifact Binding support	Low	Nicht implementiert
16	POST Binding support	High	Implementiert
17	Redirect Binding support	High	Implementiert
18	SOAP Binding support	Medium	Implementiert
19	Single Logout Profile support	High	Nicht implementiert
20	Webbrowser SSO Profile support	High	Implementiert

Tabelle 11.1.: Erfüllte Requirements

Geplante Non Functional Requirements

Nr.	Name	Status
1	Plattformunabhängigkeit	Teilweise erfüllt: Getestet auf Windows und Linux
2	Installationsfreundlich	Erfüllt: Über Extender installierbar
3	Keine zusätzlichen Libraries	Erfüllt: Keine zusätzlichen Libraries nötig
4	Userfreundlich	Erfüllt: Durch Use Case Tests bestätigt

Tabelle 11.2.: Auflistung der Non functional Requirements

Zusätzlich Funktionen

Während der Entwicklungsphase und auch an der Zwischenpräsentation kamen weitere Ideen auf, welche für einen Penetration Tester nützlich sind. Folgende Funktionen wurden zusätzlich zu den Requirements implementiert, da sie für das Entwicklungsteam wichtig erschienen:

- X.509 Zertifikat importieren (PEM und DER Format)
- X.509 Zertifikatskette importieren
- X.509 Zertifikate löschen
- Infos von X.509 Zertifikaten anzeigen
- Privates Schlüsselmaterial importieren

- PKCS#8, DER Format
- Traditional RSA, PEM Format
- Privates Schlüsselmaterial exportieren (traditioneller RSA Key Format)
- Syntaxhighlighting der SAML Nachricht

11.4. X.509 Verwaltung

Im Zertifikate Tab unseres Plugins gibt es die Möglichkeit folgende Aktionen durchzuführen:

- X.509 Zertifikat importieren (PEM und DER Format)
- X.509 Zertifikatskette importieren
- X.509 Zertifikate exportieren (PEM Format)
- X.509 Zertifikate löschen
- Infos von X.509 Zertifikaten anzeigen
- Privates Schlüsselmaterial importieren (PKCS#8 im DER Format und traditional RSA im PEM Format)
- Privates Schlüsselmaterial exportieren (traditional RSA Key Format)
- X.509 Zertifikate klonen
- X.509 Zertifikatskette klonen
- X.509 Zertifikate erstellen / bearbeiten

11.4.1. X.509 Zertifikat importieren

Mit OpenSSL können Zertifikate heruntergeladen und in verschiedene Formate konvertiert werden. Folgendermassen wird das Zertifikat von www.hsr.ch heruntergeladen. Dieses Zertifikat liegt im PEM Format vor. Mit OpenSSL kann dies zusätzlich ins DER Format konvertiert werden.

```
$ echo "" | openssl s_client -connect www.hsr.ch:443 > /tmp/hsr.pem
$ openssl x509 -in /tmp/hsr.pem -outform der -out /tmp/hsr.der
$ file /tmp/hsr.{pem,der}
/tmp/hsr.pem: ASCII text
/tmp/hsr.der: data
```

Beispiel Java

Über die Klasse `CertificateTabController` können X.509 Zertifikate im DER und PEM Format importiert werden.

```
CertificateTabController c = new CertificateTabController();
BurpCertificate c1 = c.importCertificate("/tmp/hsr.pem");
BurpCertificate c2 = c.importCertificate("/tmp/hsr.der");
```

Beispiel GUI

Über den Button "Import ..." können diese Zertifikate in beiden Formaten importiert werden. Diese werden danach im Zertifikatsbaum angezeigt (Abbildung 11.1). Die Source wird auf "Imported" gesetzt.

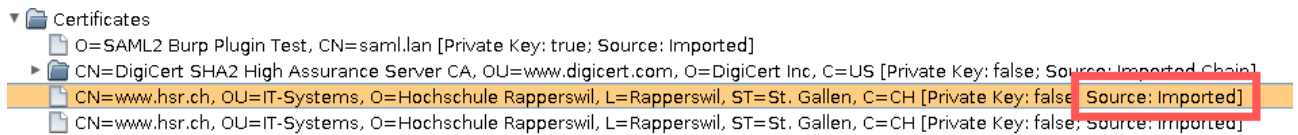


Abbildung 11.1.: Baumansicht der Zertifikate

11.4.2. X.509 Zertifikatskette importieren

In das Plugin kann eine ganze Zertifikatskette importiert werden. Eine Zertifikatskette bekommt man typischerweise bei einer TLS Verbindung mitgeliefert, wenn das Serverzertifikat von einem intermediate Zertifikat signiert wurde, welches nicht auf dem System installiert ist. Dieses mitausgelieferte intermediate Zertifikat ist von einer CA signiert, welche auf dem System installiert ist, um die Trust Chain zu bilden. Somit kann das Serverzertifikat auf Korrektheit überprüft werden. [25]

Eine Zertifikatskette kann mit der Option `-showcerts` von `s_client` aus OpenSSL heruntergeladen angesehen werden:

```
$ echo "" | openssl s_client -connect example.org:443 -showcerts > /tmp/chain.pem
```

Darin sind in diesem Falle zwei Zertifikate enthalten:

```
$ grep -A2 -i begin /tmp/chain.pem
-----BEGIN CERTIFICATE-----
MIIF6DCCBNCGAwIBAgIQBBHej100YvalqGG3EuxrWTANBgkqhkiG9w0BAQsFADBw
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
--
-----BEGIN CERTIFICATE-----
MIIEsTCCA5mgAwIBAgIQBOHnpNxc8vNtwCtCuF0VnzANBgkqhkiG9w0BAQsFADBs
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
```

Laut RFC 5246 ist die Reihenfolge so definiert, dass das erste Zertifikat das nächste signiert. Deshalb kommt als erstes das intermediate Zertifikat und am Schluss das Serverzertifikat. [66]

Beispiel Java

Die Zertifikate können über den `CertificateTabController` importiert werden. Diese sind danach in einer Liste von `BurpCertificate` verfügbar.

```
CertificateTabController c = new CertificateTabController();
List<BurpCertificate> chain = c.importCertificateChain("/tmp/chain.pem");
```

Beispiel GUI

Importiert man mit dem Button "Import Chain..." eine Zertifikatskette in das Plugin, wird diese hierarchisch in den Baum eingefügt und die Source entsprechend gekennzeichnet, damit der Tester sieht, woher diese Zertifikate kommen:

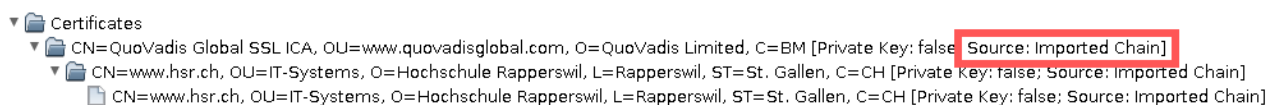


Abbildung 11.2.: Importierte Zertifikatskette

11.4.3. X.509 Zertifikate exportieren

Die Zertifikate, welche in Burp importiert oder generiert wurden, können im PEM Format exportiert werden.

Beispiel Java

Die Methode `exportCertificate` der Klasse `CertificateTabController` exportiert das angegebene Zertifikat auf den angegebenen Pfad auf das Dateisystem im PEM Format.

```
CertificateTabController c = new CertificateTabController();
BurpCertificateBuilder b = new BurpCertificateBuilder("CN=example.net");
BurpCertificate certificate = b.generateSelfSignedCertificate();
c.exportCertificate(certificate, "/tmp/exported.pem");
```

Dieses Zertifikat liegt als X.509 Zertifikat auf dem Dateisystem und kann wie jedes Zertifikat verwendet werden. Es wird also kein eigenes Dateiformat verwendet.

Beispiel GUI

Über den Button "Export..." kann das Zertifikat exportiert werden. Das Zertifikat liegt danach PEM Codiert auf dem Dateisystem:

```
$ file /tmp/exported.pem
/tmp/exported.pem: PEM certificate
```

11.4.4. X.509 Zertifikat löschen

Über den "Delete" Button kann ein Zertifikat aus der Zertifikatsliste entfernt werden.

11.4.5. Infos von X.509 Zertifikaten anzeigen

Damit man überhaupt sieht, was die Zertifikate für Eigenschaften haben, können diese abgefragt werden.

Beispiel Java

Zertifikate werden als Objekt der Klasse `BurpCertificate` gespeichert. Diese Klasse bietet einem die Möglichkeit viele Informationen abzufragen. Einige Methoden der Klasse `BurpCertificate`:

```
burpCertificate.getIssuer();
burpCertificate.getSubject();
burpCertificate.getSubjectAlternativeNames();
```

Es werden alle allgemeinen Felder von einem X.509 Zertifikat und folgende Extensions von unserem Plugin unterstützt:

- `AuthorityKeyIdentifier`
- `BasicConstraints`
- `ExtKeyUsage`
- `IssuerAlternativeName`
- `KeyUsage`
- `SubjectAlternativeName`
- `SubjectKeyIdentifier`

Beispiel GUI

In der Detailansicht im GUI werden diese Informationen angezeigt. Zuerst werden Informationen angezeigt, welche nicht im Zertifikat selber gespeichert sind, sondern separat vom Plugin verwendet und verwaltet werden. Beispielsweise ob ein Private Key verfügbar ist oder wie das Zertifikat in unser Plugin kam. Diese Informationen sehen folgendermassen aus:

Plugin Specific

Source Imported

Private Key Private Key Import: Export:

Edit Certificate

Abbildung 11.3.: Pluginspezifische Details des Zertifikats

Danach folgen direkt die generellen Informationen zum Zertifikat:

General

Version

Serial Number (Hex)

Signature Algorithm

Issuer

Not Before

Not After

Subject

Public Key Algorithm

Key Size

Modulus

Exponent

Signature

Abbildung 11.4.: Generelle Informationen des Zertifikats

Die unterstützten Extensions werden folgendermassen dargestellt:

Supported Extensions

Basic Constraints CA Path Limit No Path Limit Don't copy

Key Usage Digital Signature Non Repudiation Key Encipherment Data Encipherment Key Agreement Key Certificate Signing CRL Signing

Extended Key Usage Server Authentication OCSP Signing E-Mail Protection Code signing Client Authentication Timestamping

Subject Alternative Names

www.example.org (DNS)	Delete	E-Mail	Add
example.com (DNS)			
example.edu (DNS)			
example.net (DNS)			

Issuer Alternative Names

	Delete	E-Mail	Add
--	--------	--------	-----

Subject Key Identifier B0:00:A7:F4:22:E9:B1:CE:21:61:17:C4:C4:6E:71:64:C8:E6:0C:55 Auto generate form Public Key

Authority Key Identifier 51:68:FF:90:AF:02:07:75:3C:CC:D9:65:64:62:A2:12:B8:59:72:3B Auto generate from Issuer Public Key

Abbildung 11.5.: Unterstützte Extensions des Zertifikats

Falls es weitere Extensions gibt, welche das Plugin nicht selber bearbeiten kann, werden diese als "Unsupported Extensions" deklariert und wie folgt in einer Listbox dargestellt. Kennt das Plugin auch den Namen der Extension nicht, wird die OID angezeigt:

Unsupported Extensions

AuthorityInfoAccess	<input checked="" type="checkbox"/> Copy unsupported Extensions
CRLDistributionPoints	
CertificatePolicies	

Abbildung 11.6.: Nicht unterstützte Extensions des Zertifikats

11.4.6. Privates Schlüsselmaterial importieren

Ist ein Zertifikat in Burp verfügbar, besitzt dieses noch keinen privaten Schlüssel. Dieser ist jedoch für die Erstellung einer Signatur notwendig. Deshalb gibt es die Möglichkeit privates Schlüsselmaterial zu importieren. Unser Plugin unterstützt zwei Formate für Private Keys:

- PKCS#8 im DER Format, ohne Verschlüsselung
- Traditional RSA im PEM Format

Ein PKCS#8 Schlüssel im PEM Format sieht folgendermassen aus:

```
$ (head -3 private_key_pkcs8.pem; echo ...; tail -3 private_key_pkcs8.pem)
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBCwggSjAgEAAoIBAQCqi+IYNkXrd6k1
HtR67/j6EJ5fQbMydy22B9N263XmBlqYTBQrVo3z0atLNxLPikUwI+8DpH/JApQv
...
uRiCynorRoEmMYb1Pa5bP0yx83YSnoAr+GouKpb16talIyCYLYRLX80vdREuqEi9
8MnaDCpqrEFMV0sFXsUxI+g=
-----END PRIVATE KEY-----
```


Diesen PKCS#8 Schlüssel im DER Format erhält man, indem man den base64 codierten Teil decodiert und binär speichert:

```
$ grep -v "^-----" private_key_pkcs8.pem | base64 -d > private_key_pkcs8.der
```

Auch mit OpenSSL kann der Key vom PEM ins DER Format umgewandelt werden:

```
$ openssl pkcs8 -inform PEM -outform DER -in private_key_pkcs8.pem \
-out private_key_pkcs8.der -nocrypt
```

Zusätzlich können die PKCS#8 Schlüssel mit einem Passwort geschützt werden. Dies wird von unserem Plugin nicht unterstützt. Mit der Option `-nocrypt` kann das Passwort vom Private Key entfernt werden.

So sehen Schlüssel aus, welche im traditionellen RSA Format in PEM Codierung gespeichert werden:

```
$ (head -3 private_key_rsa.pem; echo ...; tail -3 private_key_rsa.pem)
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA1SQVamCE9y5gZ/eCCzC9m3bu+A6gFrg5G/Ha1JYy79VW0Ba8
Yq/e3xDM04xMvHNchqmK0Jmegpw0zebRtyKDbYjbTQ5A09WZb2e+Bf5krT0wiQH1
...
tUF108G5a+q4y+xajCgSBsRZg3ztDvteq9Pz4cr8RQgaGeGDeA4wjWXzrLfuX5ei
tAN7WmJwxw34pkgXS6JanU8LcLeqxKkZbtPsZo7pS3p4GcKEEQp+5A==
-----END RSA PRIVATE KEY-----
```

Beispiel Java

Der Import eines Private Keys wird mit dem `CertificateTabController` durchgeführt. Die jeweilige Methode fügt dem angegebenen Zertifikat (`BurpCertificate`) den Private Key hinzu.

```
CertificateTabController c = new CertificateTabController();
BurpCertificateBuilder b = new BurpCertificateBuilder("CN=example.net");
BurpCertificate certificate = b.generateSelfSignedCertificate();
certificateTabController.importPKCS8(b, "/tmp/private_key_pkcs8.pem");
certificateTabController.importPrivateKey(b, "/tmp/private_key.pem");
```

Beispiel GUI

Im GUI kann über die Buttons "PKCS#8 DER..." und "Traditional RSA PEM..." Private Keys importiert werden. Dies ist danach im Zertifikatsbaum als Text sowie in der Detailansicht als Checkbox sichtbar.

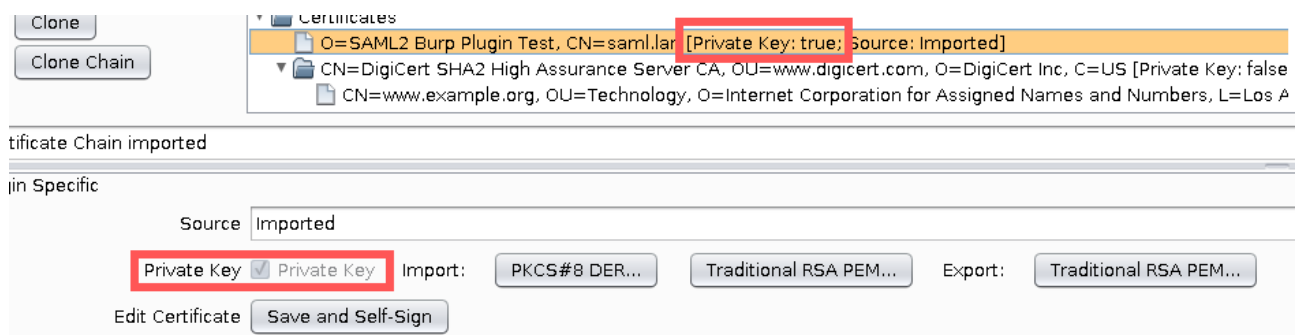


Abbildung 11.7.: Private Key ist importiert

11.4.7. Privates Schlüsselmaterial exportieren

Ist ein Zertifikat mit privatem Schlüsselmaterial im Plugin geladen, so kann auch der private Schlüssel exportiert werden. Unterstützt wird nur das traditionelle RSA Format.

Beispiel Java

Wird ein selbstsigniertes Zertifikat erzeugt, so besitzt dies einen Private Key. Dieser wird wie folgt exportiert:

```
CertificateTabController c = new CertificateTabController();
BurpCertificateBuilder b = new BurpCertificateBuilder("CN=example.net");
BurpCertificate certificate = b.generateSelfSignedCertificate();
c.exportPrivateKey(certificate, "/tmp/key.pem");
```

Dieser liegt als traditioneller RSA Key im PEM Format auf dem Dateisystem:

```
$ head -3 /tmp/key.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAptT3NP6T6WKkPB02mKijLajKPCETrOUrXCNTOWD3LpzdsqZ6I
axoUgZgPQ0++STXgzxPxWuVwyOhYXgUykp0sr/43gWQndp2okgt5MC/T32h2w72u
```

Beispiel GUI

Über den Button "Traditional RSA Key..." kann der RSA Key exportiert werden.

11.4.8. X.509 Zertifikate klonen

Ist ein Zertifikat im Plugin geladen, so kann dieses geklont werden. Das bedeutet, es werden alle Felder 1:1 kopiert, daraus ein neues Zertifikat inklusive Schlüsselmaterial erstellt und von sich selbst signiert.

Beispiel

Die Methode `cloneCertificate` aus der Klasse `CertificateTabController` bietet die Möglichkeit ein Zertifikat 1:1 zu klonen. Hierbei werden alle X.509 Felder inklusive allen Extensions kopiert. Es werden auch die Extensions kopiert, welche nicht manuell erstellt werden können und im Plugin als "Unsupported Extension" vermerkt werden.

Zuerst wird das Zertifikat der HSR heruntergeladen:

```
echo "" | openssl s_client -connect www.hsr.ch:443 2>/dev/null \
| openssl x509 > /tmp/hsr.pem
```

Jetzt kann man das Zertifikat importieren und klonen:

```
String originalCert = "/tmp/hsr.pem"
String clonedCert = "/tmp/cloned.pem"
String clonedKey = "/tmp/key.pem"

CertificateTabController c = new CertificateTabController();

BurpCertificate original = c.importCertificate(originalCert);
BurpCertificate cloned = c.cloneCertificate(original,
    new BurpCertificateBuilder(original.getSubject()));
```



```

emanuel@eris:~/Daten/Studium/Semester_6_FS15/BA_Bachelorarbeit/BA_eduss_rbischof
2 /p/2/f/62
1 Certificate:
2   Data:
3     Version: 3 (0x2)
4     Serial Number:
5       4d:6b:01:80:e1:82:b9:10:37:3d:e2:31:7c:92:75:94:70:2e:89:35
6     Signature Algorithm: sha1WithRSAEncryption
7     Issuer: C=BM, O=QuoVadis Limited, OU=www.quovadisglobal.com, CN=QuoVadis Global SSL ICA
8     Validity
9       Not Before: Aug 15 10:56:49 2012 GMT
10      Not After : Aug 15 10:56:49 2015 GMT
11      Subject: C=CH, ST=St. Gallen, L=Rapperswil, O=Hochschule Rapperswil, OU=IT-Systems, CN=www.hsr.ch
12      Subject Public Key Info:
13        Public Key Algorithm: rsaEncryption
14        Public-Key: (4096 bit)
15        Modulus:
16          00:b1:46:71:7f:fc:ec:4c:1f:26:79:71:94:0d:83:
17          2a:58:c7:7a:8f:49:f8:4e:93:54:43:d7:cd:3d:55:
/proc/23488/fd/62 1,1 Top
1 Certificate:
2   Data:
3     Version: 3 (0x2)
4     Serial Number:
5       4d:6b:01:80:e1:82:b9:10:37:3d:e2:31:7c:92:75:94:70:2e:89:35
6     Signature Algorithm: sha1WithRSAEncryption
7     Issuer: C=BM, O=QuoVadis Limited, OU=www.quovadisglobal.com, CN=QuoVadis Global SSL ICA
8     Validity
9       Not Before: Aug 15 10:56:49 2012 GMT
10      Not After : Aug 15 10:56:49 2015 GMT
11      Subject: C=CH, ST=St. Gallen, L=Rapperswil, O=Hochschule Rapperswil, OU=IT-Systems, CN=www.hsr.ch
12      Subject Public Key Info:
13        Public Key Algorithm: rsaEncryption
14        Public-Key: (4096 bit)
15        Modulus:
16          00:99:38:68:a8:af:91:7e:d6:68:70:aa:50:62:22:
17          7d:c8:01:d5:11:47:55:f1:2d:88:eb:d6:fa:b0:8f:
/proc/23488/fd/63 1,1 Top
[3] 1:vim 2:scrot- 3:bash 4:vimdiff* | emanuel@eris | 0.84 0.61 0.49 | 2015-05-10 15:35

```

Abbildung 11.9.: Die X.509 Felder unterscheiden sich nur im Public Key

```

emanuel@eris:~/Daten/Studium/Semester_6_FS15/BA_Bachelorarbeit/BA_eduss_rbischof
2 /p/1/f/63
44 df:f1:52:79:16:a6:6f:bf:4a:69:35:6a:40:25: 44 90:f8:18:88:b0:2c:13:d0:d7:f3:3e:e6:e0:5e:
45 c7:8a:8d:44:4b:5d:7f:8f:e8:10:61:b8:0c:ff: 45 59:cc:d0:83:b5:10:20:11:73:09:da:f9:1e:2a:
46 ca:dd:5a:3d:45:be:2f:55:ef:ad:f5:9b:4e:94: 46 1d:cd:70:b9:3c:11:d9:5e:ff:dd:6f:ec:36:52:
47 0b:18:36:0e:aa:11:42:9f:28:0c:30:37:57:99: 47 a5:ef:15:af:be:7b:8c:82:39:3a:ea:4a:29:bf:
48 85:38:d4:09:27:0c:41:3b:12:9e:b9:4f:23:f0: 48 ba:66:b6:18:8e:37:75:8a:ff:8d:3f:1a:aa:1b:
49 69:0c:71:fd:12:31:df:67:d3:55:be:e5:6d:5e: 49 7b:59:dd:3a:e9:ac:82:7e:2b:bd:ec:ec:c7:4d:
50 47:08:2f 50 23:5c:61
51 Exponent: 65537 (0x10001) 51 Exponent: 65537 (0x10001)
52 X509v3 extensions: 52 X509v3 extensions:
53 Authority Information Access: 53 Authority Information Access:
54 OCSP - URI:http://ocsp.quovadisglobal.com 54 OCSP - URI:http://ocsp.quovadisglobal.com
55 CA Issuers - URI:http://trust.quovadisglobal.c 55 CA Issuers - URI:http://trust.quovadisglobal.c
56 56
57 X509v3 Subject Key Identifier: 57 X509v3 Subject Key Identifier:
58 78:D1:C7:2B:90:09:82:0F:C7:58:26:02:93:A8:77:D 58 78:D1:C7:2B:90:09:82:0F:C7:58:26:02:93:A8:77:D
59 X509v3 Subject Alternative Name: 59 X509v3 Subject Alternative Name:
60 DNS:www.hsr.ch, DNS:log.hsr.ch, email:root@hsr 60 DNS:www.hsr.ch, DNS:log.hsr.ch, email:root@hsr
61 X509v3 CRL Distribution Points: 61 X509v3 CRL Distribution Points:
62 62
63 Full Name: 63 Full Name:
64 URI:http://crl.quovadisglobal.com/qvsslica.c 64 URI:http://crl.quovadisglobal.com/qvsslica.c
65 65
66 X509v3 Certificate Policies: 66 X509v3 Certificate Policies:
67 Policy: 1.3.6.1.4.1.8024.0.2.100.1.1 67 Policy: 1.3.6.1.4.1.8024.0.2.100.1.1
68 CPS: http://www.quovadisglobal.com/repositor 68 CPS: http://www.quovadisglobal.com/repositor
69 69
70 X509v3 Key Usage: critical 70 X509v3 Key Usage: critical
71 Digital Signature, Key Encipherment 71 Digital Signature, Key Encipherment
72 X509v3 Extended Key Usage: 72 X509v3 Extended Key Usage:
73 TLS Web Server Authentication, TLS Web Client 73 TLS Web Server Authentication, TLS Web Client
74 X509v3 Authority Key Identifier: 74 X509v3 Authority Key Identifier:
75 keyid:32:4D:A1:4F:EA:F0:AE:99:B6:EE:9B:07:2C:8 75 keyid:32:4D:A1:4F:EA:F0:AE:99:B6:EE:9B:07:2C:8
76 X509v3 CRL Distribution Points: 76 X509v3 CRL Distribution Points:
77 Full Name: 77 Full Name:
78 URI:http://crl.quovadisglobal.com/qvsslica.c 78 URI:http://crl.quovadisglobal.com/qvsslica.c
79 X509v3 Subject Key Identifier: 79 X509v3 Subject Key Identifier:
80 78:D1:C7:2B:90:09:82:0F:C7:58:26:02:93:A8:77:D 80 78:D1:C7:2B:90:09:82:0F:C7:58:26:02:93:A8:77:D
81 Signature Algorithm: sha1WithRSAEncryption 81 Signature Algorithm: sha1WithRSAEncryption
82 74:ee:2f:97:89:0c:45:cc:21:7c:04:2c:bb:46:2d:9b:47:d6 82 58:ed:06:3b:77:c2:4c:1a:dd:6a:a4:16:ba:20:2d:6c:d5:8d
83 e7:67:82:d6:5d:f1:d2:70:0a:31:8a:9f:2e:d2:bd:fd:e8:9d 83 f9:0d:4e:4a:0f:9b:42:f1:c7:95:3e:1c:01:26:5d:09:6c:c5
/proc/17076/fd/63 47,21 76% /proc/17076/fd/62 47,21 61%
[3] 1:vim 2:bash 3:bash- 4:vimdiff* | emanuel@eris | 0.42 0.37 0.40 | 2015-05-10 15:22

```

Abbildung 11.10.: Die Extensions haben eine andere Reihenfolge, sind aber identisch

11.4.9. X.509 Zertifikatskette klonen

Es ist auch möglich, eine gesamte Zertifikatskette zu klonen. Hierbei wird die erste CA in der Kette als erstes erstellt und mit dem Private Key dieser CA das darunterliegende Zertifikat signiert. Dies geht so in der Kette weiter bis zum letzten Zertifikat in der Kette.

Beispiel Java

Der Aufruf über den CertificateTabController sieht folgendermassen aus:

```
CertificateTabController c = new CertificateTabController();  
List<BurpCertificate> chain = c.importCertificateChain("/tmp/hsr_chain.pem");  
List<BurpCertificate> clones = c.cloneCertificateChain(chain);
```

In der Liste clones sind jetzt alle neuen Zertifikate verfügbar.

Beispiel GUI

Im GUI kann man auf den Button "Clone Chain" klicken um vom ausgewählten Zertifikat aus rekursiv alle Zertifikate zu klonen. Dies kann je nach Anzahl Zertifikaten, Schlüsselgrösse und Rechenleistung etwas dauern. Im Zertifikatsbaum werden diese neu dargestellt. In folgendem Bild ist oben die importierte Zertifikatskette und unten die geklonte Zertifikatskette. Dies ist auch an den verfügbaren Private Keys und an der eingetragenen Source sichtbar.

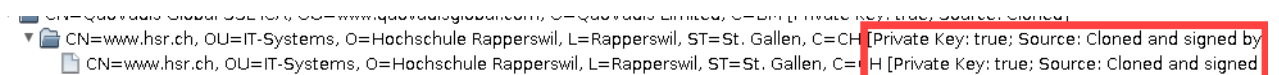


Abbildung 11.11.: Geklonte Zertifikatskette

11.4.10. X.509 Zertifikate erstellen / bearbeiten

Mit der Klasse BurpCertificateBuilder können neue Zertifikate erstellt werden. Diese werden schlussendlich als BurpCertificate im BurpCertificateStore in einer Baumstruktur abgelegt, welche oben im Tab dargestellt wird.

Es werden alle generellen Felder eines X.509 Zertifikats unterstützt:

- Version (nur Version 3 (0x2))
- Seriennummer
- Issuer
- Not Before
- Not After
- Subject
- Signaturalgorithmus (Es kann nur mit RSA Schlüssel signiert werden)
- Public Key
 - Die Grösse der Schlüssel hängt von der eingestellten Schlüsselgrösse ab.
 - Es werden nur RSA Keys unterstützt.

Desweiteren wird eine Auswahl an Extensions unterstützt:

- Basic Constraint
- Key Usage

- Extended Key Usage
 - Kann manuell gesetzt werden
 - Kann automatisch aus dem Public Key berechnet werden.
- Subject Alternative Name
- Subject Key Identifier
 - Kann manuell gesetzt werden
 - Kann automatisch aus dem Public Key berechnet werden.
- Issuer Alternative Name
 - Kann manuell gesetzt werden
 - Kann automatisch aus dem Public Key des Issuers berechnet werden.
- Authority key Identifier (entweder berechnet aus dem Public Key des Issuers oder auch direkte Eingabe eines Hex-Strings)

Weitere Extensions werden zur Zeit nicht unterstützt.

Beispiel

Folgendermassen lässt sich ein neues Zertifikat erstellen:

```
String subject = "O=SAML2 Burp Plugin Test, CN=saml.lan";
BurpCertificateBuilder b = new BurpCertificateBuilder(subject);
b.setVersion(3);
b.setSerial("11:22:33:44:55:66:77:88:99:AA:BB:CC:DD:EE:FF");
b.setNotAfter("May 23 23:23:23 2023 GMT");
b.setNotBefore("May 23 23:23:23 2005 GMT");
b.setKeySize(1024);
b.setSignatureAlgorithm("SHA256withRSA");
b.setIssuer(subject); // Self-signed

b.setBasicConstraintsPathLimit(23);

b.addKeyUsage("Key Encipherment");
b.addKeyUsage("Digital Signature");

b.addExtendedKeyUsage("Server Authentication");
b.addExtendedKeyUsage("Client Authentication");

b.addSubjectAlternativeName("DNS", "foobar.hsr.ch");
b.addSubjectAlternativeName("E-Mail", "studenten@hsr.ch");
b.addSubjectAlternativeName("E-Mail", "dozenten@hsr.ch");

b.addIssuerAlternativeName("DNS", "issuer.example.net");
b.addIssuerAlternativeName("E-Mail", "issuer23@example.net");
b.addIssuerAlternativeName("E-Mail", "issuer5@example.org");

b.setAuthorityKeyIdentifier("01:02:03:04:05:06:07:08:09:0A:0B:0C:0D:0E:0F↵
:10:11:12:13:14");

b.setSubjectKeyIdentifier(true);

BurpCertificate certificate = b.generateSelfSignedCertificate();
```

Dieses Zertifikat kann jetzt in unserem Burp Plugin verwendet werden. In OpenSSL sieht das Zertifikat folgendermassen aus:

```

emanuel@eris:~
emanuel@eris:~
$ openssl x509 -in /tmp/exported.pem -noout -text
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=saml.lan, O=SAML2 Burp Plugin Test
    Validity
      Not Before: May 23 23:23:23 2005 GMT
      Not After : May 23 23:23:23 2023 GMT
    Subject: CN=saml.lan, O=SAML2 Burp Plugin Test
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (1024 bit)
      Modulus:
        00:a2:d5:3d:a5:fe:ad:27:fc:47:a5:67:79:55:9a:
        15:2c:d2:63:1e:23:6e:aa:bc:ae:1b:14:2b:fa:83:
        80:a8:29:5f:bd:01:30:af:73:49:15:7f:b6:bf:bd:
        2c:c5:e9:c0:98:dd:99:11:2f:cc:b4:a1:15:90:2c:
        0c:de:55:17:54:c4:51:6f:78:f3:be:6a:69:f3:fa:
        72:3c:fe:be:c2:f0:ec:81:a1:bc:ac:2f:c7:7c:29:
        9e:c0:17:1e:6c:2a:af:4d:b4:72:27:38:e9:45:6b:
        fe:e7:4f:ef:78:50:7b:2f:7f:d2:1a:c1:0a:f5:c7:
        5c:a0:f1:5f:64:20:8a:b5:3f
      Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Basic Constraints: critical
        CA:TRUE, pathlen:23
      X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
      X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
      X509v3 Subject Alternative Name: critical
        DNS:foobar.hsr.ch, email:studenten@hsr.ch, email:dozenten@hsr.ch
      X509v3 Subject Key Identifier:
        EC:78:AE:AA:38:15:F6:C4:A3:5D:7E:8F:D4:A8:4C:3C:3D:28:29:E7
      X509v3 Issuer Alternative Name: critical
        DNS:issuer.example.net, email:issuer23@example.net, email:issuer5@example.org
      X509v3 Authority Key Identifier: critical
        keyid:01:02:03:04:05:06:07:08:09:0A:0B:0C:0D:0E:0F:10:11:12:13:14

    Signature Algorithm: sha256WithRSAEncryption
      07:49:9a:61:07:06:cb:c7:f1:c9:12:64:70:d3:7a:8e:db:c6:
      88:c1:a3:55:86:55:72:3a:89:d4:d2:8d:6b:92:4e:74:aa:6c:
      5b:2d:8b:3c:71:64:d7:c1:43:1c:d1:1a:14:99:c6:d2:24:6d:
      25:19:8a:67:d4:df:40:50:ae:e6:6a:e4:37:0c:00:23:2f:21:
      cc:5a:a0:8a:9a:be:01:3c:47:cd:7a:7a:5f:5d:72:35:d6:fa:
      d4:14:30:d1:f2:a4:da:93:73:0f:f6:cc:c7:f5:62:e2:35:9d:
      96:63:b1:5d:20:f8:4e:48:ee:41:a3:38:23:a5:68:0a:53:0e:
      f5:05
emanuel@eris:~
$

```

Abbildung 11.12.: Selbst erstelltes Zertifikat in OpenSSL

Wie man sieht, wurden alle Felder korrekt eingetragen und auch die Extensions richtig erstellt.

Beispiel GUI

Wie die GUI Elemente zu erstellen eines neuen Zertifikats aussehen, ist bereits im Abschnitt 11.4.5 "Infos von X.509 Zertifikaten anzeigen" beschrieben.

11.5. SAML Message Editor

Im SAML Message Editor können folgende Aktionen durchgeführt werden.

- SAML Nachricht signieren
- SAML Assertion signieren
- Signaturen entfernen
- Nachricht manuell editieren

- 8 Arten von XSW ausführen
- Vorschau der XSW Transformationen anzeigen
- In der SAML Nachricht enthaltenes Zertifikat an die SAML Zertifikatsverwaltung senden
- SAML Nachricht auf original interceptete Message zurücksetzen

Es können mit dem Plugin nur SAML Nachrichten bearbeitet werden, die als Request ausgehen. Wie in der Abbildung 8.1 beschrieben, ist der POST an den Service Provider der Ort wo wir die SAML Nachricht manipulieren. Es werden das Redirect, POST und das SOAP Binding unterstützt.

Das fertige GUI ist im Kapitel Ergebnisse in der Abbildung 13.2 zu finden. Die Funktionen sind selbst-erklärend über die jeweiligen Buttons oder ComboBoxen auszulösen.

Die folgenden Kernfunktionen sind alle in der Klasse `XMLHelpers` implementiert. Die Beispiele zeigen die Aufrufe von ausserhalb der Klasse, in diesem Fall dem `SamlTabController`.

11.5.1. SAML Nachricht / Assertion signieren

Bei der SAML Nachricht kann die Assertion und die SAML Response signiert werden. Wird nur die Assertion signiert, wird eine allfällige Signatur der Response entfernt, da diese ungültig wird. Zur Signatur wird bei einer zu ersetzenden Signatur der original benutzte Algorithmus benutzt. Ist der Algorithmus jedoch nicht bekannt oder war die originale Nachricht / Assertion nicht signiert, wird RSA-SHA256 benutzt.

Beispiel

```
Document doc = xmlHelpers.getXMLDocumentOfSAMLMessage(string);
xmlHelpers.signAssertion(doc, signAlgorithm, digestAlgorithm, cert.getCertificate(),
    cert.getPrivateKey());
```

11.5.2. Signaturen entfernen

Mit einem Klick können alle Signaturen entfernt werden, um die Signature Exclusion Attack zu testen.

Beispiel

```
Document doc = xmlHelpers.getXMLDocumentOfSAMLMessage(string);
xmlHelpers.removeAllSignatures(document)
```

11.5.3. XSW ausführen

Die ausgewählte XSW Attacke wird aus dem GUI ausgelesen und danach auf die originale Nachricht angewendet. Eine XSW Attacke wird immer auf die originale Nachricht und nicht auf die editierte Nachricht angewendet, da sonst die Richtigkeit des XSW nicht mehr gewährleistet wäre.

Beispiel

```
document = xmlHelpers.getXMLDocumentOfSAMLMessage(orgSAMLMessage);
xmlHelpers.applyXSW(samlGUI.getActionPanel().getSelectedXSW(), document);
```


11.5.4. XSW Vorschau anzeigen

Hier wird eine ein "unified diff" erzeugt, welcher die Nachricht vor und nach der Transformation anzeigt. Die Rückgabe der Methode `diffLineMode` gibt direkt HTML Code zurück, welcher nun in ein temporäres HTML File geschrieben und anschliessend im Browser angezeigt wird. In der Abbildung 11.13 ist ein Ausschnitt aus einer solchen Vorschau zu sehen.

Beispiel

```
Document document = xmlHelpers.getXMLDocumentOfSAMLMessage(orgSAMLMessage);
xmlHelpers.applyXSW(samlGUI.getActionPanel().getSelectedXSW(), document);
String after = xmlHelpers.getStringOfDocument(document, 2, true);
String diff = xmlHelpers.diffLineMode(orgSAMLMessage, after);
desktop.browse(uri);
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?> ¶
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" Consent="urn:oasis:names:tc:SAML:2.0:consentunspecified" Destination="f
IssueInstant="2015-04-06T06:42:39.213Z" Version="2.0"> ¶
<Issuer xmlns="urn:oasis:names:tc:SAML:2.0:assertion">http://SAMLWIN.saml.lan/adfs/services/trust</Issuer> ¶
<samlp:Status> ¶
<samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/> ¶
</samlp:Status> ¶
<Assertion xmlns="urn:oasis:names:tc:SAML:2.0:assertion" ID="_f27d6403-32f3-45ec-8b24-8b2fb4ca99b0" IssueInstant="2015-04-06T06:42:3
<Assertion xmlns="urn:oasis:names:tc:SAML:2.0:assertion" ID=" evil assertion ID" IssueInstant="2015-04-06T06:42:39.212Z" Version="2.0"> ¶
<Issuer>http://SAMLWIN.saml.lan/adfs/services/trust</Issuer> ¶
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"> ¶
<ds:SignedInfo> ¶
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc14n#"> ¶
<ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"> ¶
<ds:Reference URI="#_f27d6403-32f3-45ec-8b24-8b2fb4ca99b0"> ¶
<ds:Transforms> ¶
<ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"> ¶
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc14n#"> ¶
</ds:Transforms> ¶
```

Abbildung 11.13.: Diff zur Vorschau der XSW Attacke

11.5.5. Zertifikat an SAML Zertifikatsverwaltung senden

Das in einer SAML Nachricht integrierte Zertifikat wird aus der SAML Nachricht extrahiert und an den `CertificateTabController` gesendet, welcher dieses in seinen Zertifikatbaum importiert.

Beispiel

```
Document document = xmlHelpers.getXMLDocumentOfSAMLMessage(textArea.getText());
String cert = xmlHelpers.getCertificate(document.getDocumentElement());
if(cert != null){
    certificateTabController.importCertificateFromString(cert);
}
```

11.6. Testing

Um die Richtigkeit der Implementationen zu gewährleisten wurde auf der einen Seite auf Unittests mit JUnit gesetzt, welche die Kernfunktionen testen. Jedoch wurden auch Tests mit externen Applikationen durchgeführt. Die Unit Tests sind im Source Code zu finden. Die zusätzlichen Tests sind in den folgenden Unterkapiteln genauer beschrieben.

11.6.1. Zertifikatsverwaltung

Um die Funktionen der Zertifikatsverwaltung zu testen, wurden vor allem Unittests mit JUnit und das Kommandozeilentool OpenSSL eingesetzt.

Mit OpenSSL konnten Zertifikate für die Tests heruntergeladen und angeschaut werden. So konnten die X.509 Felder und Extensions ausgelesen und mit den implementierten Methoden verglichen werden. War eine Methode richtig implementiert, wurden dazu ein oder mehrere Unittests mit JUnit geschrieben. Somit kann sichergestellt werden, dass bei einer Änderung bestehende Funktionalitäten weiterhin funktionieren. Unter anderem wurden für folgende wichtigen Funktionalitäten Unittests erstellt:

- Zertifikate/Zertifikatskette importieren
- Zertifikate inkl. Extensions auslesen
- Zertifikate selber erstellen
- Zertifikat exportieren
- Zertifikat/Zertifikatskette klonen

Als Kontrolle wurden immer wieder manuelle Tests mit OpenSSL durchgeführt, um zu sehen, ob die Resultate wirklich den erwarteten Ergebnisse entspricht und ob die Zertifikate so valide generiert werden, damit fremde Software damit umgehen kann. Dieses Vorgehen stellte sich vor allem bei den Extensions als sehr wichtig heraus, da dort beispielsweise die Fehler nur in OpenSSL sichtbar wurden. Was bei den Extensions genau war, ist im Abschnitt 11.10 beschrieben.

Da bei vielen Unittests neues Schlüsselmaterial generiert werden musste, dauerten die Tests sehr lange. Schnell wurde klar, dass dies nicht zumutbar ist. Somit wurde für die Methode zur Generierung von Schlüsselmaterial ein Fake geschrieben und der Methode, welche Schlüsselmaterial generiert, übergeben. Jetzt wird für die Unittests immer derselbe RSA Schlüssel mit 512 Bit Schlüssellänge verwendet, damit dieser nicht jedes mal neu generiert werden muss. Die tiefe Bitzahl beschleunigt auch das Signierungsverfahren, welches nicht gefaked werden kann.

11.6.2. SAML Nachricht Manipulation

Die Manipulation der SAML Nachrichten wurde unter anderem mit der im Kapitel 12 beschriebenen Testumgebung getestet. Vor allem die Korrektheit des Signierens der SAML Nachrichten wurde dadurch verifiziert, dass eine SAML Nachricht mit dem Plugin intercepted wurde, danach mit dem korrekten Identity Provider Zertifikat erneut signiert und an den Service Provider weitergeleitet wurde. Wenn die Nachricht angenommen wird, ist die Implementation der Signierung korrekt. Die grundlegende Korrektheit der produzierten XML Signatur wird bereits in einem Unittest getestet.

Das XSW wird in Unittests so getestet, indem eine manuell manipulierte Nachricht mit der von unserem Plugin produzierten Nachricht verglichen wird. Ausserdem sind noch grundlegende Funktionen auf SAML Nachrichten mit Unittests getestet.

11.6.3. Use Case Testing

Um die Erfüllung der Use Cases zu testen, hat der Entwickler des einen Teils des Plugins, die Use Cases des anderen Teils auf Ihre Vollständigkeit und Funktion hin getestet.

Nr.	Name	Ergebnis
01	Execute Common XSW Attack	Funktioniert
02	Execute permuted XSW Attacks	Nicht implementiert.
03	Rate XML Signature Algorithms	Nicht implementiert.
04	Exclude XML Signature	Funktioniert
05	Create XML Signature	Funktioniert
06	Replace XML Signature	Funktioniert
07	Clone Certificate	Funktioniert
08	Clone Certificate Chain	Funktioniert
09	Export Certificate	Funktioniert
10	SAML Nachrichten editieren	Funktioniert
11	Zertifikatsfelder editieren	Funktioniert, Save Button nicht sehr intuitiv platziert
12	Rate Algorithms	Nicht implementiert.
13	Display Information	Funktioniert
14	Single Logout für User auslösen	Nicht implementiert
15	Framework Fingerprinting	Nicht implementiert.

Tabelle 11.3.: Ergebnisse des Use Case Tests

11.7. Eingesetzte Libraries

Bei der Implementation wurde so gut es ging auf Java Bordmittel gesetzt, damit die externen Abhängigkeiten so klein wie möglich gehalten werden können. Da dies jedoch nicht immer ohne enormen Aufwand möglich war, wurden die folgenden Libraries eingesetzt. Alle diese Libraries sind unter einer Open Source Lizenz lizenziert.

Bouncy Castle

Bouncy Castle [46] wird verwendet um die Zertifikatsverwaltung zu unterstützen. Diese Library wurde benutzt, da sie X509 Zertifikate sehr gut unterstützt und die meist verbreitete Library für Java und X509 ist.

Bouncy Castle ist unter MIT license lizenziert.

RSyntaxTextArea

RSyntaxTextArea [13] wird genutzt um die SAML Nachricht mit schönem Syntax Highlighting darzustellen. Es ist die einzige Library die dies in Java so einfach zur Verfügung stellt. RSyntaxTextArea ist unter modified BSD license lizenziert.

Diff Match and Patch

Wir nutzen Diff Match and Patch [18] um den Vergleich der Message vor dem applizieren des XSW und danach zu berechnen und anzuzeigen. Diff Match and Patch wurde gewählt, da diese Library sehr einfach einen Diff erzeugen und visualisieren kann. Diese Library ist unter Apache License 2 lizenziert.

11.8. Build

Die nötigen Libraries hatten wir zu Beginn ebenfalls im Git Repository, was nicht sehr ideal ist: Das Repository wird dadurch relativ gross, falls eine Library durch eine neue ersetzt wird, bleibt die alte Library in der Git History und das Austauschen ist durch das Ersetzen der Datei nicht sehr elegant. Deshalb haben wir uns für das Build System Maven entschieden. Somit müssen die Libraries nicht im Repository committed sein, da Maven die nötigen Abhängigkeiten selbstständig herunterlädt.

Das gesamte Eclipse Projekt wurde in ein Maven Projekt umgewandelt und auch die Verzeichnisstruktur an die Vorgaben von Maven angepasst. Somit wissen auch spätere Entwickler der Software, wie unser Verzeichnis zu verwenden ist. Die Verzeichnisstruktur sieht jetzt wie folgt aus:

- ./README.md: Beschreibung des Projekts
- ./AUTHORS: Autoren
- ./pom.xml: Konfigurationsdatei inklusive Abhängigkeiten für Maven
- ./src/main/resources: Dateien für das Plugin
- ./src/main/java: Java Packages und Klassen für das Plugin
- ./src/main/scripts: Shellscripts um X.509 Zertifikate zu erstellen (wird nicht direkt vom Plugin verwendet)
- ./src/test/resources: Dateien für die JUnit Tests
- ./src/test/java: Packages und Klassen für die JUnit Tests
- ./gitignore: Nicht zu trackende Dateien für Git
- ./LICENSE: Lizenz (MIT)
- ./target: Hier wird das gebildete Jar abgelegt

Ein neuer Entwickler muss jetzt nur das Git Repository klonen und kann sofort mit Maven die nötigen Abhängigkeiten herunterladen und aus dem Code eine Jar Datei bilden, welche direkt in der Burp Suite verwendet werden kann:

```
$ git clone https://eduss@git.hsr.ch/git/SAMLBurpPlugin
$ mvn clean install
```

Listing 11.1: Befehle um Repository zu klonen und Abhängigkeiten zu installieren

Der Buildvorgang führt auch gleich alle JUnit Tests durch und legt das gebildete Jar im Verzeichnis targets ab. Die Libraries, welche vom Plugin verwendet werden, sind in der Jar Datei mit dem Postfix jar-with-dependencies eingebettet, damit diese nicht zusätzlich installiert werden müssen.

11.9. Lizenz

Die Apache License 2 [17] von "Diff Match and Patch", die MIT License [49] von Bouncy Castle und die BSD License [48] von RSyntaxTextArea sind mit der MIT License kompatibel. Wir haben uns deshalb und aus folgenden Gründen für die MIT Lizenz [49] entschieden:

- Lizenz für freie Software.
- Die Lizenz und das Copyright muss erwähnt werden
- Die Software darf kopiert, verändert und weiterverbreitet werden.
- Die Lizenz ist sehr kurz und einfach gehalten.
- Falls ein Schaden wegen der Software entsteht, ist man nicht haftbar.

11.10. Probleme

11.10.1. Encoding der ausgehenden Nachricht

Zu Beginn der Arbeit bestand das Problem, dass Umlaute in der SAML Nachricht beim Weiterleiten der Nachricht nicht korrekt encodiert wurden. Das Problem war, dass kein Encoding beim Erstellen von Strings aus Byte Arrays und umgekehrt angegeben wurde.

Als Lösung wurde das Encoding konsequent bei allen Methoden angegeben und danach funktioniert das Encoding korrekt.

11.10.2. XML valide Signieren

Es wurde getestet, ob ein XML valide signiert wird, indem eine gültige Assertion mit dem korrekten Private Key signiert wurde und danach an den Service Provider weitergeleitet wurde. Es hat am Anfang nicht funktioniert, dass die Nachricht beim SP angenommen wird. Deshalb wurde die Library OpenSAML2 [10] benutzt um die Nachricht zu signieren und zu überprüfen ob die eigene Signaturmethode und OpenSAML sich im Ergebnis unterscheidet. Dies war der Fall.

Ein Hauptunterschied war, dass nach der Signierung noch Leerschläge und Linebreaks vorhanden waren, die entfernt wurden, was nach dem Erstellen einer Signatur, diese ungültig gemacht hat. Nachdem auch die letzten Attribute korrekt waren und auch die `Canonicalization` Method übereinstimmte, waren die mit OpenSAML und mit der eigenen Methode signierten Nachrichten identisch. Die Nachricht wurde nun jedoch immer noch nicht angenommen beim SP.

Es wurde nun gerüft ob die originale Nachricht vom IdP von unserer Nachricht abweicht und inwiefern. Der einzige Unterschied war nun das Zertifikat, welches gleich sein sollte.

Der ADFS Identity Provider erstellt im Verlaufe des Einrichtungsassistenten ein Zertifikat mit einem Private Key, von welchem danach zwei weitere Zertifikate und Private Keys abgeleitet werden. Diese werden für die Signierung und die Verschlüsselung der Nachrichten benötigt. Es wurde für die Neusignierung mit dem Plugin das Hauptzertifikat und dessen Schlüssel benutzt und nicht das für die Signatur abgeleitete.

Als Lösung wurde ein neues Zertifikat erzeugt, im ADFS für die Signierung der Assertions importiert und den Service Provider bekanntgegeben. Nun mit dem korrekten Zertifikat funktionierte das Signieren mit dem Plugin.

11.10.3. Auslesen von X.509v3 Extensions nur schwer möglich

Da von der Bouncy Castle Library grossen Gebrauch gemacht wird, wurden die Zertifikate in einem Objekt der Bouncy Castle Klasse `X509CertificateHolder` gespeichert. Diese bietet viele Methoden um die Details der Zertifikate auszulesen, doch bei den Extensions gelangten wir schnell an die Grenzen. Es gab nur eine generische Methode um alle oder einzelne Extensions abzurufen. Diese kamen aber in einem Format daher, das nicht nutzbar war, sprich die Daten daraus konnten nicht geparkt und weiterverwendet werden. So war es nicht möglich die einzelnen Extensions sinnvoll auszulesen.

Die Lösung war die Verwendung der Klasse `X509Certificate` aus dem in Java mitgelieferten Package `java.security.cert`. Diese Klasse bietet viel bessere Methoden für den Umgang mit Extensions. Der Wechsel von `X509CertificateHolder` zu `X509Certificate` hat sich gelohnt. Für viele Extensions steht eine Methode zum Auslesen zur Verfügung.

Diese Methoden sind auch nicht immer perfekt, aber immerhin ist es möglich die Eigenschaften der Extensions auszulesen. So kann über diese Klasse beispielsweise direkt die "KeyUsage" ausgelesen werden. Hierzu bekommt man ein boolean Array zurück. Der Wrapper verarbeitet dieses Array und macht daraus eine Liste, mit sprechenden Namen, welche danach weiterverwendet werden kann.

11.10.4. Unittests brauchen lange wegen Schlüsselgenerierung

Wie bereits im Abschnitt 11.6 genauer beschrieben, dauerten die Unittests, bei welchen ein Zertifikat generiert wird, sehr lange, da bei jedem Unittest neues Schlüsselmaterial generiert wurde. Durch einen Fake mit statisch einprogrammierten Keys konnte dieses Problem elegant gelöst werden.

11.10.5. Cloning der Extensions misslingt

Das Cloning der Extensions funktioniert so, dass diese 1:1 ausgelesen und direkt ins neue Zertifikat gespeichert werden, also ohne den Inhalt der Extensions zu parsen und daraus die neue Extension zu erstellen. Als Input für eine neue Extension erwartete die Methode ein Byte Array. Die Extensions konnten ausgelesen und in einem Byte Array gespeichert werden. Dieses Array wurde der Methode für die Erstellung einer neuen Extension übergeben, wobei am Byte Array nichts verändert wurde. Dies wollte einfach nicht funktionieren. Die Ergebnisse in OpenSSL sahen immer so aus:

```
X509v3 extensions:
  Authority Information Access:
    .k..h0f0*..+.....0...http://ocsp.quovadisglobal.com08..+.....0...http://trust.gu
X509v3 Subject Key Identifier:
    .....x...+.....X&...w..8..
X509v3 Subject Alternative Name:
    ..*'0%.
www.hsr.ch.
log.hsr.ch..root@hsr.ch
X509v3 CRL Distribution Points:
    .7..40200...*http://crl.quovadisglobal.com/qvsslica.crl
X509v3 Certificate Policies:
    .M..J0H0F..+.....X..d..0604..+.....(http://www.quovadisglobal.com/repository
X509v3 Authority Key Identifier:
    .....0...2M.O.....P..~
X509v3 Extended Key Usage:
    .....0...+.....+.....
X509v3 Key Usage:
    .....0...+.....+.....
```

Abbildung 11.14.: Zertifikat mit fehlerhaften Extensions

Scheinbar wurden die Extensions gespeichert, aber irgendwie stimmte etwas noch nicht. Als ob die Extensions im falschen Format vorliegen würden. Die API von Bouncy Castle, war nutzlos, da dort nur von einem Byte Array die Rede war. Nach vielen Stunden debuggen und ausprobieren fand ich die Lösung im Source Code vom X509v3CertificateBuilder aus Bouncy Castle selber: Das Byte Array musste in den Datentyp `ANSIEncodable` umgewandelt werden. Wäre Bouncy Castle nicht Open Source, hätte die Lösungssuche wohl noch länger gedauert. Die API Beschreibung hätte aber besser sein dürfen.

12. Testumgebung

12.1. Aufbau

In der Abbildung 12.1 ist eine Übersicht über die Testumgebung illustriert.

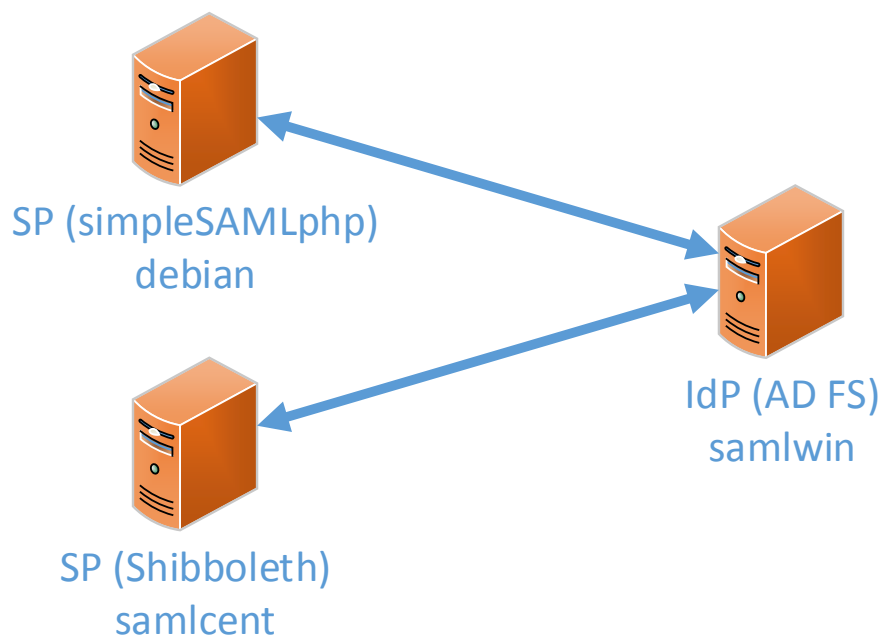


Abbildung 12.1.: Aufbau der Testumgebung

12.1.1. ADFS Identity Provider

Betriebssystem	Windows Server 2008 R2
Hostname	SAMLWIN
IP	192.168.123.1
User	bauser
Aufgabe	IdP
Software	Active Directory, AD Federation Services, IIS

Besonderheiten:

- Domänencontroller für Domäne: saml.lan

12.1.2. Service Provider Shibboleth

Betriebssystem	CentOS 6
Hostname	samlcent
IP	192.168.123.20
User	bauser
Aufgaben	SP
Software	Apache, Shibboleth SP
Testlink	https://samlcent/secure/

12.1.3. SP SimpleSAMLphp

Betriebssystem	Debian
Hostname	debian
IP	192.168.123.30
User	bauser
Aufgaben	SP
Software	Apache, SimpleSAMLphp SP
Testlink	https://debian/simplesaml/module.php/core/authenticate.php

12.2. Installation

Um Active Directory Federation Services als IdP und Shibboleth als SP zu nutzen sind wir wie folgt vorgegangen [3] [5] [1] :

Auf dem ADFS Identity Provider:

1. Rolle "AD Domainservices" installieren
2. Rolle "IIS" installieren
3. Server als Domänencontroller einrichten
4. ADFS 2.0 installieren
5. Im IIS self-signed Zertifikat erstellen für Default Seite auf CN `samlwin.saml.lan`
6. Bei Defaultsite Binding auf Port 443 mit vorher erstelltem SSL Zertifikat hinzufügen.
7. Nun über die ADFS Konfigurationskonsole einrichten: Neuen Verbunddienst, Eigenständiger Verbundserver, erstelltes SSL Zertifikat angeben
8. Die Metadaten befinden sich nun unter <https://samlwin/FederationMetadata/2007-06/FederationMetadata.xml>

Auf dem Service Provider Shibboleth:

1. Installation von Shibboleth aus dem Repository http://download.opensuse.org/repositories/security://shibboleth/CentOS_CentOS-6/security:shibboleth.repo
2. Installation vom Apache Webserver: `yum install httpd`
3. Self-signed Zertifikat erstellen und in `/etc/httpd/conf.d/ssl.conf` angeben

4. In etc/httpd/conf.d/shib.conf prüfen ob Location für secure Verzeichnis vorhanden ist
5. Restart
6. In /etc/selinux/config den Wert SELINUX auf permissive setzen
7. Status sollte nun unter <https://samlcent/Shibboleth.sso/Status> verfügbar sein
8. In der Datei /etc/shibboleth/shibboleth2.xml die entityID auf <https://samlcent/shibboleth> ändern und beim Tag CredentialResolver das self-signed Zertifikat mit Key angeben: <CredentialResolver type='File' key='localhost.pem' certificate='localhost.pem' />

Auf dem Identity Provider ADFS

1. Zertifikat von SP in Internet Explorer unter Trusted Root Certification Authorities importieren.
2. <https://samlcent/Shibboleth.sso/Metadata> als Datei speichern und bei ADFS als Relying Party importieren
3. In der ADFS Konfiguration die Claim Rules für samlcent editieren. Dabei eine Issuance Transform Rule hinzufügen für LDAP Attribute. Rule Name ist Get Data das Mapping der LDAP Attribute ist wie folgt: User-Principal-Name → UPN, Token-Groups- Unqualified Names → Group. [1]
4. Nun noch eine Custom Rule erstellen die heisst Transform UPN to epPN. Der folgende Text stellt die Custom Rule dar [1]:

```
c:[Type == "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn"]
=> issue(Type = "urn:oid:1.3.6.1.4.1.5923.1.1.1.6", Value = c.Value, ↵
  Properties["http://schemas.xmlsoap.org/ws/2005/05/identity/↵
    claimproperties/attributename"] = "urn:oasis:names:tc:SAML:2.0:↵
    attrname-format:uri");
```

5. Nun noch eine Custom Rule mit folgendem Text erstellen, damit die Group auf das epSA Attribut gemappt wird. Name der Regel: Transform Group to epSA [1]

```
c:[Type == "http://schemas.xmlsoap.org/claims/Group", Value == "Domain ↵
  Users"]
=> issue(Type = "urn:oid:1.3.6.1.4.1.5923.1.1.1.9", Value = "↵
  member@contoso.com", Properties["http://schemas.xmlsoap.org/ws↵
    /2005/05/identity/claimproperties/attributename"] = "urn:oasis:names:↵
    tc:SAML:2.0:attrname-format:uri");
```

6. Die Metadaten vom IdP <https://fsweb.contoso.com/FederationMetadata/2007-06/FederationMetadata.xml> lokal abspeichern und editieren.
7. Im EntityDescriptor den Namespace xmlns:shibmd='urn:mace:shibboleth:metadata:1.0' hinzufügen.
8. Den Tag IDPSSODescriptor wie folgt abändern: [1]

```
<IDPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML↵
  :2.0:protocol"><Extensions><shibmd:Scope regexp="false">samlwin.saml.↵
  lan</shibmd:Scope></Extensions><KeyDescriptor use="encryption">
```

9. Die nun editierte Datei editedMetadata.xml auf SP übertragen
10. Auf dem IdP in den Server Features sicherstellen, dass bei .Net die WCF Activation installiert ist und in den Diensten der Net.TcpListeneradapter läuft.
11. In IIS ein Port Binding für net.tcp hinzufügen.

12. in der Datei C:/inetpub/adfs/ls/web.config den Authmode auf Forms ändern
<authentication mode='Forms' />
13. Auch in der web.config Datei, den Abschnitt localAuthenticationTypes so ändern damit Forms zu oberst ist [2]:

```
<localAuthenticationTypes >
  <add name="Forms" page="FormsSignIn.aspx" />
```

Auf Service Provider Shibboleth:

1. Die Datei editedMetadata.xml in shibboleth2.xml angeben: [1] <MetadataProvider type='XML' file='/etc/shibboleth/editedFedMetadata.xml' /> Dabei alle anderen Metadataprovider auskommentieren.

Service Provider SimpleSAMLphp Installation

1. Apache mit PHP und SSL einrichten.
2. SimpleSAMLphp von <https://simplesamlphp.org/> herunterladen und nach /var/simplesaml/ extrahieren.
3. Anleitung von SimpleSAMLphp zur Installation eines SP durchführen. [6]
4. Ein self-signed Zertifikat erstellen und unter /var/simplesaml/cert ablegen. [7]
5. Mit dem Tool debian/simplesaml/admin/metadata-converter.php die Metadaten des IdP in die php-Form bringen und in die Datei /var/simplesaml/metadata/saml20-idp-remote.php einfügen.. [7]
6. In der Datei /var/simplesaml/authsources.php das Attribut Idp auf die Entity ID des Idp [7]
7. Über das Tool http://debian/simplesaml/module.php/core/frontpage_federation.php die SP Metadaten exportieren und auf den IdP transferieren. [7]

Identity Provider für SimpleSAMLphp Service Provider anpassen

1. In ADFS einen neuen Relying Party Trust erstellen und dabei die Metadaten des SP importieren.
2. Die gleichen Claim Rules wie bei SP Shibboleth hinzufügen.
3. Nun noch die diese Claim Rule hinzufügen, da SimpleSAMLphp einen transient identifier erwartet. [9]


```
c: [Type == 'http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn'] =>
  issue(Type = 'http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier',
  Issuer = c.Issuer, OriginalIssuer = c.OriginalIssuer, Value = c.Value,
  ValueType = c.ValueType, Properties['http://schemas.xmlsoap.org/ws/2005/05/identity/cla
  = 'urn:oasis:names:tc:SAML:2.0:nameid-format:transient');x
```

12.3. Beispiel einer SAML Authentifizierung

Wie im Kapitel 5.4.1 unter dem Abschnitt "Service Provider Initiated Single Sign-On (Redirect/POST Bindings)" theoretisch beschrieben, haben wir einen Login Prozess zwischen dem Browser, dem Service Provider samlcent und dem Identity Provider samlwin nachgestellt und dokumentiert. Folgend werden die SAML Teile aufgeführt. Dabei entspricht die Nummerierung der Nachrichten jener der Abbildung 5.2. In der Abbildung 12.2 sind die zwei SAML Nachrichten illustriert.

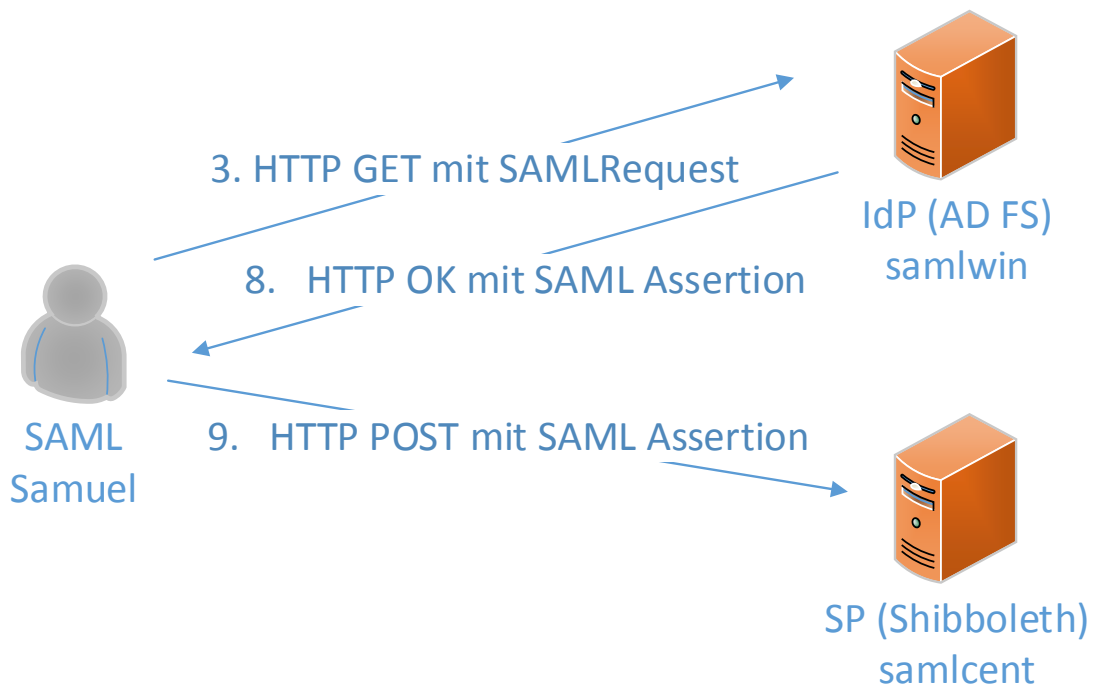


Abbildung 12.2.: Gekürzter Ablauf einer Anmeldung per SAML

3. HTTP GET mit SAMLRequest (Browser → IdP)

Im Listing 12.1 die SAML Nachricht zu sehen, bei der der Browser von SAML Samuel den AuthnRequest an den IdP sendet. Den AuthnRequest hat der Browser per 302 Redirect vom SP erhalten.

```
<samlp:AuthnRequest
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  AssertionConsumerServiceURL="https://samlcent/Shibboleth.sso/SAML2/POST"
  Destination="https://samlwin.saml.lan/adfs/ls/"
  ID="_b8378b2a3d44043465cd20191bad3139"
  IssueInstant="2015-03-07T10:24:15Z"
  ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
  Version="2.0">

  <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
    https://samlcent/shibboleth
  </saml:Issuer>
  <samlp:NameIDPolicy AllowCreate="1"/>
</samlp:AuthnRequest>
```

Listing 12.1: SAML Request Browser → IdP

8. HTTP OK mit SAML Assertion (IdP → Browser)

Der IdP sendet nach erfolgreicher Überprüfung der angegebenen Benutzerdaten eine signierte Assertion, im Listing 12.2 zu sehen, zurück an den Browser. Diese Assertion wird im Schritt 9 vom Browser an den SP weitergeleitet.

```

<samlp:Response
  ID="_a3d9aec5-4854-4a6b-990e-5634697e0ecb"
  Version="2.0"
  IssueInstant="2015-03-07T10:25:02.833Z"
  Destination="https://samcent/Shibboleth.sso/SAML2/POST"
  Consent="urn:oasis:names:tc:SAML:2.0:consent:unspecified"
  InResponseTo="_b8378b2a3d44043465cd20191bad3139"
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">

  <Issuer xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
    http://SAMLWIN.saml.lan/adfs/services/trust
  </Issuer>

  <samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success" />
  </samlp:Status>

  <Assertion
    ID="_643167e5-6ef4-4011-b5c7-c9d948752874"
    IssueInstant="2015-03-07T10:25:02.833Z"
    Version="2.0"
    xmlns="urn:oasis:names:tc:SAML:2.0:assertion">

    <Issuer>http://SAMLWIN.saml.lan/adfs/services/trust</Issuer>

    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod
          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        <ds:SignatureMethod
          Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
        <ds:Reference
          URI="#_643167e5-6ef4-4011-b5c7-c9d948752874">
          <ds:Transforms>
            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature" />
            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
          <ds:DigestValue>2o5nmG+qAhBH7ih87aBK9F1bYggoyu8DFXkuRSnp1Ww=</ds:
DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue> ... gekuerzt </ds:SignatureValue>
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <ds:X509Data>
          <ds:X509Certificate> ... gekuerzt </ds:X509Certificate>
        </ds:X509Data>
      </KeyInfo>
    </ds:Signature>

    <Subject>
      <SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <SubjectConfirmationData
          InResponseTo="_b8378b2a3d44043465cd20191bad3139"
          NotOnOrAfter="2015-03-07T10:30:02.833Z"
          Recipient="https://samcent/Shibboleth.sso/SAML2/POST" />
        </SubjectConfirmation>
      </Subject>

```

```

<Conditions
  NotBefore="2015-03-07T10:25:02.827Z"
  NotOnOrAfter="2015-03-07T11:25:02.827Z">
  <AudienceRestriction>
    <Audience>https://samlcent/shibboleth</Audience>
  </AudienceRestriction>
</Conditions>

<AttributeStatement>
  <Attribute
    Name="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn">
    <AttributeValue>browser@saml.lan</AttributeValue>
  </Attribute>

  <Attribute
    Name="http://schemas.xmlsoap.org/claims/Group">
    <AttributeValue>Domaenen - Benutzer</AttributeValue>
  </Attribute>

  <Attribute
    Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <AttributeValue>browser@saml.lan</AttributeValue>
  </Attribute>
</AttributeStatement>

<AuthnStatement
  AuthnInstant="2015-03-07T10:25:02.776Z">
  <AuthnContext>
    <AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:
PasswordProtectedTransport</AuthnContextClassRef>
  </AuthnContext>
</AuthnStatement>
</Assertion>
</samlp:Response>

```

Listing 12.2: SAML Request IdP → Browser

13. Ergebnisse

13.1. Zertifikatsverwaltung

In der Burp Suite gibt es auf der Root Ebene ein neues Tab. In der Abbildung 13.1 als Überblick das gesamte Tab:

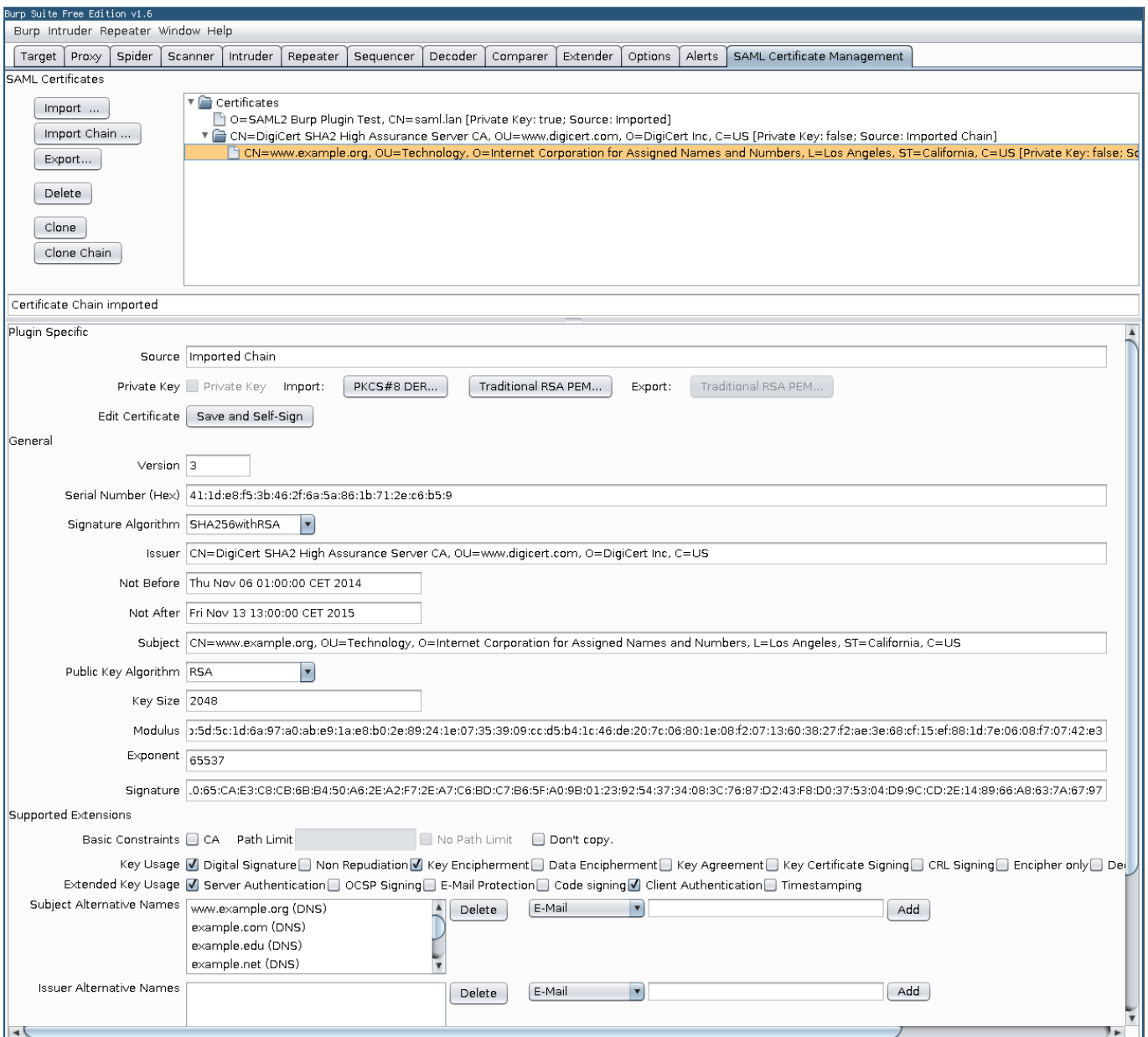


Abbildung 13.1.: Übersicht über das SAML Certificate Management

Dort findet die gesamte Verwaltung der X.509 Zertifikate statt. Im oberen Teil können Zertifikate importiert, exportiert und geklont werden. In einer Baumansicht werden die Zertifikate hierarchisch dargestellt.

Im unteren Teil werden Details zum selektierten Zertifikat angezeigt. Hierzu wurden alle allgemeinen Felder eines X.509 Zertifikats und häufig auftretende Extensions im GUI abgebildet. Es ist möglich privates Schlüsselmaterial zu importieren und zu exportieren. Die Zertifikatsfelder können zudem editiert werden. Über einen speichern Button werden die Felder ausgelesen und daraus ein neues Zertifikat generiert.

Eine genauere detailliertere Auflistung aller Funktionen und eine dazugehörige Beschreibung ist im Kapitel 11.4 zu finden.

13.2. SAML Editor

Ein Message Editor Tab für die Burp Suite wurde erstellt. Damit können SAML Nachrichten editiert, neu signiert, Signaturen entfernt, das Zertifikat extrahiert und XSW Attacken appliziert werden. Ausserdem bietet es eine Übersicht über die wichtigsten Informationen der Assertion.

Es werden das Redirect, POST und SOAP Binding von SAML unterstützt. Eine volle Übersicht über den Funktionsumfang ist in der Auflistung 11.5 im Unterkapitel 11.5 zu finden. In der Abbildung 13.2 ist das Userinterface zu sehen, zum Zeitpunkt wenn eine SAML Nachricht interceptet wurde. Im oberen linken Bereich können verschiedene Aktionen wie eine Nachricht zu signieren, mit einem Klick angewendet werden. Rechts oben ist ein Überblick über die SAML Nachricht zu sehen. Im unteren Bereich ist die SAML Nachricht mit Syntax Highlighting zu sehen.

13.2.1. Nicht erreichte Ergebnisse

Wir konnten in der verfügbaren Zeit nicht alle Requirements implementieren. Folgt erläutern wir weshalb manche Requirements nicht erfüllt wurden nebst dem Fehlen von Zeit.

Das Requirement permuted XSW haben wir, aus dem Grund, dass dies viel zu viel Zeit in Anspruch nehmen würde, nicht umgesetzt. Weiter haben wir eine Kennzeichnung von schwachen Signaturalgorithmen nicht implementiert. Dies wurde hauptsächlich ausgelassen, da der Penetration Tester mit unserem Plugin den Algorithmus nun auf einen Blick sieht und selbst beurteilen kann ob dies ein guter Algorithmus ist.

Ein Fingerprinting der SAML Frameworks wurde nicht umgesetzt, da sich dies nach kurzer Analyse als nicht trivial herausstellte und die verfügbare Zeit nicht auch noch für dies ausreichte.

Das Single Logout Profile wird nicht unterstützt, da wir uns zuerst auf die Login Profiles konzentrieren wollten und dort so gut wie möglich, viele Funktionen abdecken.

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts SAML Certificate Management

Intercept HTTP history WebSockets history Options

Request to http://samlcent:80 [unknown host]

Forward Drop Intercept is on Action

Raw Params Headers Hex SAMLUEL

XSW Attacks

XSW1 Preview in Browser... Reset Message

Permutated XSW... Apply XSW

XML Signature

CN=ADFS Signing - SA... Remove Signatures (Re)Sign Assertion

Send Certificate to SAML Certificates (Re)Sign Message

Assertion

Condition Not Before 2015-04-06T06:42:39.210Z

Condition Not After 2015-04-06T07:42:39.210Z

Issuer http://SAMLWIN.saml.lan/adfs/services/trust

Signature

Signature Algorithm http://www.w3.org/2001/04/xmldsig-more#rsa-sha256

Digest Algorithm http://www.w3.org/2001/04/xmlenc#sha256

Subject

Subject Conf. Not Before

Subject Conf. Not After 2015-04-06T06:47:39.213Z

Encrypted with

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" Consent="urn:oasis:names:tc:SAML:2.0:consent:unspecified" Destination=
"https://samlcent/Shibboleth.sso/SAML2/POST" ID="_fd601e21-5f81-469e-88c7-da72dcf1357" InResponseTo="_545e60fe3602a06d25f241b622c5a773" IssueInstant=
"2015-04-06T06:42:39.213Z" Version="2.0">
3   <Issuer xmlns="urn:oasis:names:tc:SAML:2.0:assertion">http://SAMLWIN.saml.lan/adfs/services/trust</Issuer>
4   <samlp:Status>
5     <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
6   </samlp:Status>
7   <Assertion xmlns="urn:oasis:names:tc:SAML:2.0:assertion" ID="_f27d6403-32f3-45ec-8b24-8b2fb4ca99b0" IssueInstant="2015-04-06T06:42:39.212Z" Version="2.0">
8     <Issuer>http://SAMLWIN.saml.lan/adfs/services/trust</Issuer>
9     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
10      <ds:SignedInfo>
11        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
12        <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
13        <ds:Reference URI="#_f27d6403-32f3-45ec-8b24-8b2fb4ca99b0">
14          <ds:Transforms>
15            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
16            <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
17          </ds:Transforms>
18          <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
19          <ds:DigestValue>foKK0crQsYCouYU2pt9dvyDdI9Z4s5Z0WAHrplAFa8</ds:DigestValue>
20        </ds:Reference>
21      </ds:SignedInfo>
22      <ds:SignatureValue>
23        5a/BEGAYZFArapDrrhKpYcB7wADxpN1rwBoY5ADyMs1FDZ2Xbrk6ILBVdWqH78Xd50QtAXgap+Zsx8dIVF5TN407s8TDT3UkGERQu4eTisjhJaNjnc+HNXtkubKnQ2jpoGdoDfpgf2UJIVq7b9zXqXkI4V4DcMOJc1hbiIw
24        I2GXFLzm70fWYDAkuAkbaA0wX716jb6xkmHhA4kEDysz0xFLlublKp92H74D0wIhnIqP2k60NzuTmlfjMGN5FZengZyJUG6IX79mfFpCG6tFM9wRzaehThGRLIQ2QtYh4McBYwAq1JrL2QXurSpH061rAzk0D79HKDBPR6
25        2Zws55Jw==</ds:SignatureValue>
26      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
27        <ds:X509Data>
28          <ds:X509Certificate>
29            MIIC3DCCAcSgAwIBAgIQN0u7JfaKFrXpPoGuP0EeVjTANBqkqhkiG9w0BAQsFADAQMScwJgYDVQQExe9BREZTIFNpZ225pbmcgLSBTQU1Mv010LnNhbWubwubGFuMB4XDTE1MDIyNTE3MTE0N1owKjE
30            oMCYGAUEAxMfQURGUyBTakduaw5nIC0gU0FTFfdJT15zYn1sLmxbhjCCASIWdQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAPLTYkbBivPa2+KrOvxo1a10cOnxzFR1ZELyhyicj2j0hKuQd+fb+OgP4fNuaH/dEbSiZ
31            0fd3HtQ0nrc65NTrXpPqAasMEGpVVOem14kaKwxrYOD3NbFoFvXjvjmV9UQt2RaBe160sFe5805clvVvXA2Sf81fIeH1S8EMavFOQFqkQbDU/XmGtW0XjQhyiJ4MEy7Zwgu2HmxiwNa6wSf1DXZIUyq3gUZ+eFr8kTg

```

Abbildung 13.2.: Übersicht des SAML Editor Tabs.

14. Schlussfolgerungen

14.1. Bewertung der Ergebnisse

Es wurde eine Software erstellt, die mit keiner anderen Software in ihrem Funktionsumfang vergleichbar ist. Somit ist es sehr erfreulich, dass ein Plugin mit einem solch grossen Funktionsumfang erstellt wurde, welches bereits im produktiven Einsatz verwendet werden kann. Das Zusammenspiel der Zertifikatsverwaltung und des SAML Message Editors bietet ein optimales Werkzeug zum Testen von SAML Umgebungen. Einzig im Bereich der Codequalität mussten Abstriche gemacht werden.

14.2. Vergleich mit anderen Produkten

Das WS-Attacker Framework [45] ist ein vergleichbares Produkt, welches sich jedoch auf permutierte XSW Attacken, XML Encryption Attacks und SOAP Services konzentriert. Ein Produkt, wie das erstellte Plugin, das sich auf SAML konzentriert, gibt es nicht auf dem Markt.

14.3. Ausblick

14.3.1. Verbesserung der Architektur

Das Produkt sollte einer Architekturoptimierung unterzogen werden. Dies beinhaltet das Refactoring damit keine wechselseitigen Abhängigkeiten mehr bestehen, wie Sie in der Abbildung 10.2 im Kapitel 10 mit roten Pfeilen dargestellt sind. Dies bedeutet überall und nicht nur an einzelnen Stellen das Observer Pattern und Eventlistener zu verwenden.

14.3.2. Weitere Features

Alle noch nicht erfüllten Requirements sind noch zu implementieren. In einer zukünftigen Version wäre es ausserdem vorteilhaft, WSS mit dem SAML Token Profile [47] zu unterstützen. Damit könnte die bereits vorhanden Hacking-Lab Aufgabe gelöst werden.

14.3.3. Weiterentwicklung

Da wir eine funktionierende Software erstellten, die hoffentlich auch von Penetration Tester eingesetzt wird, haben wir uns entschieden das Projekt auch nach der Bachelorarbeit weiterzuführen.

Wir veröffentlichen unser Projekt nach einem Refactoring auf GitHub [19]. Somit besteht die Möglichkeit, dass auch andere Leute an unserem Projekt teilnehmen und entweder Bugs melden, Featurewünsche einbringen oder sogar Code beisteuern. Wir haben uns für GitHub entschieden, weil GitHub eine sehr grosse Community hat und im Gegensatz zu einer eigenen Projektseite schnell fremder Code

in Form eines Pull Requests eingepflegt werden kann und auch Hilfsmittel wie Bugtracker und Wiki bereits zur Verfügung stehen.

Zudem werden wir auch das Plugin in den BApp Store [\[52\]](#) stellen, damit die Installation direkt aus der Burp Suite heraus funktioniert und somit unser Plugin von noch mehr Testern verwendet werden kann.

14.3.4. Fazit

Zusammenfassend kann man somit sagen, dass das Plugin funktional ausgereift ist, in seiner Architektur und der Codequalität jedoch noch verbessert werden kann.

Teil III.
Anhang

A. Glossar

- ADFS** Active Directory Federation Services. Microsoft Windows Server Rolle für Identity Provider.
- Artifact** SAML Daten gehen direkt als Artifact vom IdP zum SP, ohne beim User vorbeizukommen.
- Assertion** Aussage über ein Subjekt.
- Base64** Kodierung um binäre Daten in einem ASCII String darzustellen.
- CA** Certificate Authority. Stelle die digitale Zertifikate ausstellt.
- CLR** Certificate Revocation List. Liste von Zertifikaten die als revoziert gekennzeichnet sind.
- CSR** Certificate Signing Request. Format in der eine Anfrage zur digitalen Signatur an eine CA gesendet wird.
- Canonicalization** Daten in eine Standardform bringen
- Claims** SSO und Identity Federation Lösung von Microsoft.
- Cookie** Daten die vom Webserver an den Browser und wieder zurück gesendet werden um Daten zu einer Session zu speichern.
- DSA** Digital Signature Algorithm. Ein Algorithmus für digitale Signaturen.
- Downgrade Attacke** Eine HTTPS Verbindung wird auf eine schlechtere Verschlüsselung gebracht.
- Elliptische Kurve** Spezielle algebraische Kurven die auch in der Kryptografie benutzt werden.
- FIPS** Federal Information Processing Standards, öffentliche Standards entwickelt von den USA.
- Federation** Zusammenschluss mehrerer Organisationen.
- Framework** Software die als Grundgerüst für weitere Software dient
- GUI** Graphical User Interface. Grafische Schnittstelle zum Benutzer.
- Google Hacking** Angriffe mit Unterstützung der Google Suche durchführen
- HTTP POST** Ein HTTP Request der Daten beinhaltet, die vom Webserver ausgelesen werden sollen.
- HTTP Redirect** HTTP Status zur Umleitung auf eine andere Seite.
- HTTP Status Code** Statusnummer die als Antwort vom Webserver an den Client übertragen wird.
- HTTP** Hypertext Transfer Protocol. Protokoll um Webseiten abzurufen.
- IdP** Identity Provider. Stelle an der sich der Benutzer authentifiziert.
- Internet of Things** Alltagsgeräte, die mit Elektronik ausgestattet sind und über ein Netzwerk verbunden sind.
- MVC** Model-View-Controller. Software Architektur Pattern.
- Man-in-the-Middle** Angriff bei dem ein Angreifer sich zwischen Opfer und Ziel platziert.
- NFR** Non Functional Requirement. Anforderungen an ein System, die nicht das Verhalten der Software spezifizieren.

OASIS Organization for the Advancement of Structured Information Standards. Spezifizierte unter Anderem SAML.

OCSP Online Certificate Status Protocol. Wird zur online Überprüfung von revozierten Zertifikaten benutzt.

ODF Open Document Format. Freies Dateiformat für Büroanwendungen.

OpenSSL Freie SSL/TLS Library.

PKCS#10 Certificate Request <https://tools.ietf.org/html/rfc2986>

PKCS#11 Plattformunabhängige API für kryptografische Tokens.

PKCS Public-Key Cryptography. Gruppe von kryptografischen Standards.

PKI Public-Key Infrastructure. Infrastruktur um Public-Key Material zu verwalten.

Private Key Privater Schlüssel in einem Public-Key Umfeld

RSA Public-Key Kryptosystem.

S/MIME Standard für Public-Key Verschlüsselung.

SAML Asserting Party Stellt eine SAML Assertion aus.

SAML Authority Synonym für Asserting Party.

SAML Bindings Setzt fest wie eine SAML Nachricht transportiert werden soll

SAML Identity Provider Stellt eine Assertion aus.

SAML Principal Über den Principal wird eine Assertion gemacht.

SAML Profiles Legt eine Menge von Anwendungen fest, wie SAML Protokolle genutzt werden können.

SAML Protocols Definition von SAML Requests und Responses auszusehen haben.

SAML Requester Fragt nach einer SAML Assertion.

SAML Responder Sendet die SAML Assertion zurück.

SAML Service Provider Bietet einen Service falls Assertion gültig ist.

SAML Subject Synonym für Principal.

SAML Security Assertion Markup Language. Standard für verteilte Authentifizierung.

SOAP Simple Object Access protocol. Protokollspezifikation auf Basis von XML Nachrichten.

SP Service Provider. Bietet einen Service mit einer SAML Authentifizierung an.

SSO Single Sign-On. Erlaubt Anmeldung an mehreren Systemen mit dem gleichen Login.

Self-Signed Zertifikat, welches durch sich selbst signiert wurde.

Signatur Digitale Signatur, die mathematisch die Authentizität von Daten bestätigt.

Switch Managt .ch und .li Domains und das Netzwerk zwischen den schweizer Hochschulen.

TC Technical Committee. Arbeitsgruppen der OASIS.

UI User Interface. Schnittstelle zum Benutzer. Oft GUI.

X.500 Standard für Verzeichnisdienste.

X.509 Standard für Zertifikate.

XML Namespace Identifizieren das Vokabular eindeutig.

XML Schema Empfehlung zum Definieren von Strukturen von XML Dokumenten.

XML Signatur Signatur über ein XML oder einen Teilbaum dessen.

XML Extensible Markup Language. Sprache um hierarchisch strukturierte Daten darzustellen.

XSD XML Schema Definition. Abkürzung für XML Schema.

XSW XML Signature Wrapping. Angriffstyp auf Endpunkte die eine XML Signatur überprüfen.

B. Literaturverzeichnis

- [1] AD FS 2.0 Step-by-Step Guide: Federation with Shibboleth 2 and the InCommon Federation. <https://technet.microsoft.com/en-us/library/gg317734%28v=ws.10%29.aspx>. zuletzt besucht am 24.02.2015.
- [2] Föderationsbenutzer ist beim Anmelden bei Office 365, Azure oder Intune wiederholt nach Anmeldeinformationen gefragt. <http://support.microsoft.com/kb/2461628>. zuletzt besucht am 07.03.2015.
- [3] Install and Configure Shibboleth in CentOS 6. <https://coolpandaca.wordpress.com/2013/04/04/install-and-configure-shibboleth-in-centos-6-idp-and-sp/>. zuletzt besucht am 07.03.2015.
- [4] Security Assertion Markup Language. https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language. zuletzt besucht am 08.06.2015.
- [5] Shibboleth IdP and SP Installation and Configuration. <http://csrdu.org/blog/2011/07/04/shibboleth-idp-sp-installation-configuration/>. zuletzt besucht am 07.03.2015.
- [6] simpleSAMLphp Installation and Configuration. <https://simplesamlphp.org/docs/stable/simplesamlphp-install>. zuletzt besucht am 25.04.2015.
- [7] SimpleSAMLphp Service Provider QuickStart. <https://simplesamlphp.org/docs/stable/simplesamlphp-sp>. zuletzt besucht am 25.04.2015.
- [8] XML Security Library. <https://www.aleksey.com/xmlsec/>. zuletzt besucht am 05.03.2015.
- [9] ADFS 2.0 InvalidNameIDPolicy. <http://stackoverflow.com/questions/19816803/adfs-2-0-invalidnameidpolicy>, 2013. zuletzt besucht am 25.04.2015.
- [10] Internet 2. OpenSAML 2, Home . <https://wiki.shibboleth.net/confluence/display/OpenSAML/Home>. zuletzt besucht am 02.06.2015.
- [11] Alexa. About Us. <http://www.alexa.com/about>, 2015. zuletzt besucht am 08.06.2015.
- [12] BlueKrypt. ECRYPT II Recommendations (2012). <http://www.keylength.com/en/3/>, 2015. zuletzt besucht am 26.03.2015.
- [13] bobbylight. RSyntaxTextArea. <https://github.com/bobbylight/RSyntaxTextArea>, 2015. zuletzt besucht am 25.05.2015.
- [14] Tony Bourke. Creating Your Own SSL Certificate Authority (and Dumping Self Signed Certs). <http://datacenteroverlords.com/2012/03/01/creating-your-own-ssl-certificate-authority/>, 2012. zuletzt besucht am 05.03.2015.
- [15] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. <https://tools.ietf.org/html/rfc2119>, 1997.
- [16] D. Cooper, NIST, S. Santesson, Microsoft, S. Farrell, Trinity College Dublin, S. Boeyen, Entrust, R. Housley, Vigil Security, W. Polk. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. <http://tools.ietf.org/html/rfc5280>, 2008. zuletzt besucht am 11.03.2015.

- [17] Apache Software Foundation. Apache License. <http://www.apache.org/licenses/LICENSE-2.0>, 2004. zuletzt besucht am 03.06.2015.
- [18] Neil Fraser. google-diff-match-patch. <https://code.google.com/p/google-diff-match-patch/>, 2015. zuletzt besucht am 25.05.2015.
- [19] Inc GitHub. GitHub. <https://github.com/>, 2015. zuletzt besucht am 09.06.2015.
- [20] Paul Heinlein. OpenSSL Command-Line HOWTO. <https://www.madboa.com/geek/openssl/>, 2014. zuletzt besucht am 05.03.2015.
- [21] Internet2. The Internet2 Community: Enabling the Future. <http://www.internet2.edu/about-us/>, 2014. zuletzt besucht am 18.02.2015.
- [22] Internet2. Communities & Groups. <http://www.internet2.edu/communities-groups/members/>, 2015. zuletzt besucht am 18.03.2015.
- [23] Internet2. Internet2 Products: Shibboleth. <http://www.internet2.edu/products-services/trust-identity-middleware/shibboleth/>, 2015. zuletzt besucht am 18.03.2015.
- [24] Internet2. Products. <https://shibboleth.net/products/>, 2015. zuletzt besucht am 19.03.2015.
- [25] Aetion LLC. What is the SSL Certificate Chain? <https://support.dnssimple.com/articles/what-is-ssl-certificate-chain/>, 2008.
- [26] Microsoft. Introduction to ADFS. <https://technet.microsoft.com/en-us/library/cc786469%28v=ws.10%29.aspx>, 2005. zuletzt besucht am 25.03.2015.
- [27] Microsoft. A Guide to Claims based Identity and Access Control, 2nd Edition. <https://msdn.microsoft.com/en-us/library/ff423674.aspx>, 2011. zuletzt besucht am 19.03.2015.
- [28] Microsoft. Best Practices for Secure Planning and Deployment of AD FS. <https://technet.microsoft.com/en-us/library/ff630160.aspx>, 2012. zuletzt besucht am 25.03.2015.
- [29] Microsoft. Extended Protection for Authentication Overview. <https://msdn.microsoft.com/en-us/library/vstudio/dd767318%28v=vs.90%29.aspx>, 2015. zuletzt besucht am 25.03.2015.
- [30] OASIS. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. <http://docs.oasis-open.org/security/saml/v2.0/>, 2005. zuletzt besucht am 26.02.2015.
- [31] OASIS. Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0. <http://docs.oasis-open.org/security/saml/v2.0/>, 2005. zuletzt besucht am 26.02.2015.
- [32] OASIS. Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0. <http://docs.oasis-open.org/security/saml/v2.0/>, 2005. zuletzt besucht am 26.02.2015.
- [33] OASIS. Conformance Requirements for the OASIS Security Assertion Markup Language (SAML) V2.0. <http://docs.oasis-open.org/security/saml/v2.0/>, 2005. zuletzt besucht am 26.02.2015.
- [34] OASIS. Glossary for the OASIS Security Assertion Markup Language (SAML) V2.0. <http://docs.oasis-open.org/security/saml/v2.0/>, 2005. zuletzt besucht am 26.02.2015.
- [35] OASIS. Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0. <http://docs.oasis-open.org/security/saml/v2.0/>, 2005. zuletzt besucht am 26.02.2015.

- [36] OASIS. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. <http://docs.oasis-open.org/security/saml/v2.0/>, 2005. zuletzt besucht am 26.02.2015.
- [37] OASIS. Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0. <http://docs.oasis-open.org/security/saml/v2.0/>, 2005. zuletzt besucht am 26.02.2015.
- [38] OASIS. SAML V2.0 Executive Overview. http://www.oasis-open.org/committees/documents.php?wg_abbrev=security, 2008. zuletzt besucht am 19.02.2015.
- [39] OASIS. Security Assertion Markup Language (SAML) V2.0 Technical Overview. <http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.pdf>, 2008. zuletzt besucht am 19.02.2015.
- [40] OASIS. SAML Version 2.0 Errata 05. <http://docs.oasis-open.org/security/saml/v2.0/errata05/os/saml-v2.0-errata05-os.odt>, 2012. zuletzt besucht am 26.02.2015.
- [41] OASIS. About Us. <https://www.oasis-open.org/org>, 2015. zuletzt besucht am 19.02.2015.
- [42] OASIS. Committees. <https://www.oasis-open.org/committees>, 2015. zuletzt besucht am 19.02.2015.
- [43] OASIS. OASIS. <https://www.oasis-open.org>, 2015. zuletzt besucht am 19.02.2015.
- [44] OASIS. Standards. <https://www.oasis-open.org/standards>, 2015. zuletzt besucht am 19.02.2015.
- [45] Chair of Network and Ruhr-University Bochum Datasecurity. WS-Attacker. <http://sourceforge.net/projects/ws-attacker/>, 2015. zuletzt besucht am 03.06.2015.
- [46] Legion of the Bouncy Castle Inc. The Legion of the Bouncy Castle. <https://www.bouncycastle.org/>, 2015. zuletzt besucht am 25.05.2015.
- [47] OASIS Open. Web Services Security: SAML Token Profile 1.1 . <https://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLSecurityTokenProfile.pdf>, 2006. zuletzt besucht am 02.06.2015.
- [48] Opensource.org. The BSD 3-Clause License. <http://opensource.org/licenses/BSD-3-Clause>, 2015. zuletzt besucht am 03.06.2015.
- [49] Opensource.org. The MIT License (MIT). <http://opensource.org/licenses/MIT>, 2015. zuletzt besucht am 03.06.2015.
- [50] PortSwigger. Draft new extensibility API. <http://blog.portswigger.net/2012/12/draft-new-extensibility-api.html>, 2012. zuletzt besucht am 30.03.2015.
- [51] PortSwigger. Writing your first Burp Suite extension . <http://blog.portswigger.net/2012/12/writing-your-first-burp-extension.html>, 2012. zuletzt besucht am 30.03.2015.
- [52] PortSwigger. BApp Store. <https://pro.portswigger.net/bappstore/>, 2015. zuletzt besucht am 30.03.2015.
- [53] PortSwigger. Burp Extender. <http://portswigger.net/burp/extender/>, 2015. zuletzt besucht am 30.03.2015.
- [54] PortSwigger. Burp Extender API. <http://portswigger.net/burp/extender/api/index.html>, 2015. zuletzt besucht am 30.03.2015.
- [55] PortSwigger. Burp Suite. <http://portswigger.net/burp/>, 2015. zuletzt besucht am 30.03.2015.

- [56] PortSwigger. PortSwigger Web Security Blog. <http://blog.portswigger.net/search/label/burp%20extender>, 2015. zuletzt besucht am 30.03.2015.
- [57] Klaus Schmeh. *Kryptografie: Verfahren, Protokolle, Infrastrukturen*. dpunkt.verlag, 5 edition, 2013.
- [58] RSA Security. RFC 2986: PKCS #10: Certification Request Syntax Specification. <https://tools.ietf.org/html/rfc2986>, 2000.
- [59] Shibboleth. IdPCookieUsage. <https://wiki.shibboleth.net/confluence/display/SHIB2/IdPCookieUsage>, 2015. zuletzt besucht am 27.03.2015.
- [60] Shibboleth. Wiki Shibboleth. <https://wiki.shibboleth.net>, 2015. zuletzt besucht am 27.03.2015.
- [61] Deb Shinder. Claims Based Identity: What does it Mean to You? http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/Claims-Based-Identity-What-does-Mean-You-Part2.html. zuletzt besucht am 24.02.2015.
- [62] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. On Breaking SAML: Be Whoever You Want to Be, 2012. Paper und Video vom 21st USENIX Security Symposium: <https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/somorovsky>.
- [63] Balsamiq Studios. Balsamiq. <http://balsamiq.com/>, 2015. zuletzt besucht am 08.04.2015.
- [64] SWITCH. SWITCHCaai. <http://www.internet2.edu/about-us/>, 2015. zuletzt besucht am 24.03.2015.
- [65] OASIS Security Services TC. Security Assertion Markup Language (SAML) v2.0. <https://www.oasis-open.org/standards#samlv2.0>, 2005. zuletzt besucht am 18.02.2015.
- [66] T. Dierks und E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. <https://tools.ietf.org/html/rfc5246>, 2008.
- [67] W3C. XML Signature Syntax and Processing (Second Edition). <http://www.w3.org/TR/xmldsig-core/>, 2008. zuletzt besucht am 05.03.2015.