

Secologic

Web Service Security

Best Practice Guide

Version 1.0 – Februar 2007

Dieses Material wird von der SAP AG zur Informationszwecken zur Verfügung gestellt, ohne Haftung für Fehler oder Auslassungen in dieser Publikation. Des Weiteren übernimmt SAP keine Garantie für die Exaktheit oder Vollständigkeit der Informationen, Texte, Grafiken, Links und sonstigen in dieser Publikation enthaltenen Elementen. Diese Publikation wird ohne jegliche Gewähr, weder ausdrücklich noch stillschweigend, bereitgestellt. Dies gilt u. a., aber nicht ausschließlich, hinsichtlich der Gewährleistung der Marktgängigkeit und der Eignung für einen bestimmten Zweck sowie für die Gewährleistung der Nichtverletzung geltenden Rechts.

SAP haftet nicht für entstandene Schäden. Dies gilt u. a. und uneingeschränkt für konkrete, besondere und mittelbare Schäden oder Folgeschäden, die aus der Nutzung dieser Materialien entstehen können. Diese Einschränkung gilt nicht bei Vorsatz oder grober Fahrlässigkeit.

Die gesetzliche Haftung bei Personenschäden oder Produkthaftung bleibt unberührt. Die Informationen, auf die Sie möglicherweise über die in diesem Material enthaltenen Hotlinks zugreifen, unterliegen nicht dem Einfluss von SAP, und SAP unterstützt nicht die Nutzung von Internetseiten Dritter durch Sie und gibt keinerlei Gewährleistungen oder Zusagen über Internetseiten Dritter ab.

Dieses Material wurde von der SAP mit Unterstützung des Fraunhofer Institut ‚Sichere Informations-Technologie‘ (SIT) im Rahmen des Forschungsprojektes *secologic* erstellt. *Secologic* (Laufzeit 2005 - März 2007) wird gefördert vom Bundesministerium für Wirtschaft und Technologie (BMWi). Wir danken dem BMWi für die Unterstützung des Projektes.

Weitere Informationen zu dem Projekt finden Sie unter www.secologic.de. Bei Fragen und/oder Anregungen zu diesem Dokument wenden Sie sich bitte an Rosemaria Giesecke (rosemaria.giesecke@sap.com).

Inhaltsverzeichnis

INHALTSVERZEICHNIS	3
ABBILDUNGSVERZEICHNIS.....	5
TABELLENVERZEICHNIS	5
EINLEITUNG	6
1 ALLGEMEINES MODELL VON WEB SERVICES.....	7
1.1 KERN-STANDARDS	8
1.1.1 XML.....	8
1.1.2 SOAP	8
1.1.3 WSDL	8
1.1.4 UDDI.....	9
1.2 STANDARDS UND STANDARDISIERUNGSORGANISATIONEN – EIN KURZÜBERBLICK	9
1.3 SCHUTZZIELE.....	11
1.3.1 Vertraulichkeit.....	11
1.3.2 Integrität	12
1.3.3 Authentizität	12
1.3.4 Verfügbarkeit	12
1.3.5 Verbindlichkeit.....	12
1.3.6 Anonymität.....	12
2 POTENTIELLE SCHWACHSTELLEN UND ANGRIFFSMÖGLICHKEITEN.....	13
2.1 ALLGEMEINE ANGRIFFSFORMEN	13
2.1.1 Denial of Service.....	13
2.1.2 Capture Replay Attack	14
2.1.3 Man-in-the-Middle Attack	15
2.2 STANDARDANGRIFFE AUF WEBANWENDUNGEN	15
2.2.1 SQL Code Injection.....	16
2.2.2 Buffer Overflows	16
2.3 WEB SERVICE-SPEZIFISCHE ANGRIFFE	17
2.3.1 WSDL und Access Scanning	17
2.3.2 External Entity Attacks	18
2.3.3 XML Bombs.....	19
2.3.4 Large Payloads.....	20
2.3.5 XPath Injections	20
2.3.6 XML Rewriting.....	22
3 SICHERHEITSANFORDERUNGEN.....	25
3.1 PHASEN DER WS-ENTWICKLUNG	25
3.2 BESTIMMUNG DER GRUNDLEGENDEN SICHERHEITSANFORDERUNGEN.....	26
3.3 REFERENZMODELL.....	30
3.4 SICHERHEITSANFORDERUNGEN AN DEN REFERENZPUNKTEN.....	30
4 LÖSUNGEN ZUR UMSETZUNG DER SICHERHEITSANFORDERUNGEN.....	33
4.1 ÜBERBLICK ÜBER WEB SERVICE SICHERHEITSSPEZIFIKATIONEN	33
4.1.1 XML Encryption	34
4.1.2 XML-Signature	34
4.1.3 WS-Security	34
4.1.4 WS-SecurityPolicy	35
4.1.5 WS-Trust.....	35
4.1.6 WS-Privacy	35
4.1.7 WS-SecureConversation	35



4.1.8	WS-Federation	35
4.1.9	WS-Authorization	36
4.1.10	SAML	36
4.1.11	XACML	36
4.2	UMSETZUNG DER SICHERHEITSANFORDERUNGEN DURCH WS*-SPEZIFIKATIONEN	37
4.2.1	Gewährleistung der Vertraulichkeit und Integrität, sowie Sicherheit der Kommunikation über mehrere Anfragen hinweg (SA1 & SA2)	37
4.2.2	Gewährleistung der Authentizität, sowie Vertrauensbeziehungen und Vereinigungen (SA3).....	38
4.2.3	Gewährleistung der Autorisation (SA4).....	39
4.2.4	Gewährleistung der Verfügbarkeit (SA5)	39
4.2.5	Gewährleistung der Verbindlichkeit (SA6).....	40
4.2.6	Gewährleistung der Anonymität (SA7).....	40
4.2.7	Sicherheitsrichtlinien.....	40
4.2.8	Zusammenfassung.....	41
4.3	ZUORDNUNG DER WS*-STANDARDS ZU DEN REFERENZPUNKTEN.....	42
5	FRAMEWORKS ZUR ERSTELLUNG VON SICHEREN WEB SERVICES.....	43
5.1	ÜBERBLICK.....	43
5.2	SAP NETWEAVER	43
5.2.1	Web Service Sicherheit in SAP NetWeaver.....	44
5.3	OPEN SOURCE FRAMEWORKS	46
5.3.1	GLASSFISH	46
5.3.2	APACHE AXIS	48
5.3.2.1	Axis Architektur	49
5.3.2.2	WSS4J	50
5.3.2.3	Axis 2	52
5.4	INFRASTRUKTURELLE MAßNAHMEN.....	52
5.4.1	Sicherer Betrieb des NetWeaver Application Servers Java.....	53
5.4.2	Sicherer Betrieb eines GlassFish Application Servers	53
5.4.3	Sicherer Betrieb von Axis	53
6	FAZIT	54
	REFERENZEN.....	55
	ABKÜRZUNGSVERZEICHNIS.....	58

Abbildungsverzeichnis

Abbildung 1: Kommunikationsdreieck der WS-Nutzung.....	7
Abbildung 2: Phasen der WS-Entwicklung (in Anlehnung an [Har03])	25
Abbildung 3: Referenzpunkte im WS-Dreieck	30
Abbildung 4: Web Service Sicherheitsspezifikationen im Überblick.....	34
Abbildung 5: NetWeaver Sicherheitseinstellungen	45
Abbildung 6: NetWeaver Web Service Konfiguration	46
Abbildung 7: Axis Server Architektur [Axis]	50

Tabellenverzeichnis

Tabelle 1: Übersicht der relevanten Standards und Spezifikationen.....	11
Tabelle 2: Sicherheitsanforderungen.....	29
Tabelle 3: Sicherheitsanforderungen an den Referenzpunkten.....	32
Tabelle 4: Umsetzung der Sicherheitsanforderungen durch WS*-Spezifikationen.....	41
Tabelle 5: Zuordnung der WS*-Spezifikationen zu den Referenzpunkten.....	42
Tabelle 6: Überblick Web Services Frameworks.....	43
Tabelle 7: Sicherheitskonfigurationen NetWeaver	45

Einleitung

Unternehmen aller Größenordnung haben in den vergangenen Jahren ihren klassischen Webauftritt durch interaktive Webshops im Frontoffice Bereich erweitert und sich so neue Absatzwege erschlossen. Mit Web Services steht eine neue Technologie bereit, die einen weiteren Schritt in Richtung Automatisierung von Geschäftsprozessen bietet. Sie erlaubt die Vernetzung von Backoffice Anwendungen über Unternehmensgrenzen hinweg, deren Funktionalitäten über standardisierte Schnittstellen angeboten werden. Web Services haben mittlerweile die Einsatzreife erreicht und die großen Anbieter von Weblösungen, wie z.B. SAP, IBM, BEA und Microsoft, setzen auf eine *Service Orientierte Architektur (SOA)* in ihren Laufzeitumgebungen. Neben den kommerziellen Lösungen gibt es aber auch eine Reihe von Open Source Frameworks, die eine Entwicklungs- und Laufzeitumgebung für Web Services bieten.

Der Einsatz von Web Services bietet neben einer erhöhten Flexibilisierung und Automatisierung einen weiteren wichtigen Vorteil. Vorhandene Software muss nicht ersetzt werden, sondern kann durch Wrapperfunktionen für Web Services standardisiert nutzbar gemacht werden. Doch im gleichen Maße wie diese Form der Automatisierung und Öffnung der Geschäftsprozesse nach außen Flexibilität verspricht, treten auch Bedrohungen und Sicherheitsfragen in den Vordergrund. Wenn Aufträge, Preisanfragen, Bestellungen und andere Geschäftsdaten zwischen einer Vielzahl von teilweise miteinander konkurrierenden Unternehmen mitunter halbautomatisch ausgetauscht werden, dann sind die Ansprüche der betreffenden Unternehmen und ihrer Kunden an die Sicherheit der Kommunikation und Datenverarbeitung hoch. Preisantworten sollen vertraulich sein, Angebotsabgaben verbindlich, der Zugriff auf bestimmte Web Services soll autorisiert sein, die Integrität und Authentizität der Kommunikation muss gewährleistet und der Datenschutz beachtet werden, um nur die vordringlichsten Sicherheitsanforderungen zu erwähnen.

Das vorliegende Dokument geht diesen Fragen nach und stellt die aktuellen Standards und Technologien zu Web Services und ihrem Umfeld vor. Der besondere Schwerpunkt liegt dabei auf den Standards und Entwicklungen im Bereich Web Service Security und deren Unterstützung in SAP Netweaver und den Open Source Frameworks Axis und GlassFish.

Zielgruppe für dieses Dokument sind Softwareentwickler und Softwarearchitekten, die zukünftig Web Services *sicher* erstellen und nutzen wollen.

Dieses Dokument ist wie folgt aufgebaut. In Kapitel 1 wird ein Kurzeinstieg in Web Services gegeben und es werden die allgemeinen Schutzziele definiert, die man bei der Kommunikation über offene Netze sicherstellen möchte. Kapitel 2 stellt mögliche Angriffe gegen Web Services vor, die die Einhaltung der Schutzziele gefährden. In Kapitel 3 werden die Schutzziele in Form von konkreten Sicherheitsanforderungen für Web Services genauer gefasst, deren Durchsetzung mittels geeigneter Standards und Mechanismen in Kapitel 4 vorgestellt wird. Den Abschluss bildet Kapitel 5, das untersucht, welche WS Security Standards in den oben erwähnten Frameworks umgesetzt sind. Empfehlungen zum sicheren Programmieren und Konfigurieren von Web Services runden diesen Abschnitt ab.

1 Allgemeines Modell von Web Services

Web Services bieten gegenüber anderen Ansätzen zur Kommunikation in verteilten Systemen, wie dem *Remote Procedure Call (RPC)* oder klassischer Middleware wie CORBA zwei entscheidende qualitative Vorteile. Sie sind zum einen wirklich lose gekoppelt, d.h. es werden keine Anforderungen an die Art der Plattform und Implementierung der Web Services auf Sender- und Empfängerseite gestellt. Zum anderen basieren Web Services auf allgegenwärtigen Protokollen und Standards, wie XML, HTTP, SMTP, etc., die sehr gut verstanden und allgemein akzeptiert sind.

Abbildung 1 zeigt das Kommunikationsdreieck von Dienstanbieter (Service Provider), Dienstanwender (Service Requester) und Dienstregistry.

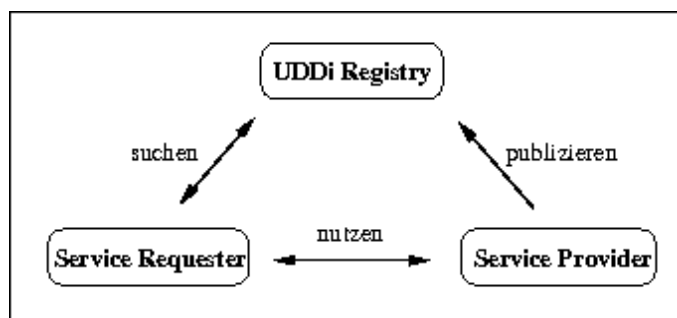


Abbildung 1: Kommunikationsdreieck der WS-Nutzung

Im ersten Schritt publiziert der Dienstanbieter (Service Provider) seinen neuen Dienst in der Registry, wo interessierte Dienstanwender (Service Requester) nach geeigneten Diensten für ihre Anforderungen suchen können. Hat der Nutzer einen Dienst mit passenden Attributen gefunden, folgt er einer URL, die er von der Registry bekommen hat und holt sich die genaue Dienstbeschreibung vom Dienstanbieter. Im nächsten Schritt nutzt er den Dienst des Anbieters wie er in der Dienstbeschreibung spezifiziert ist. Für die Beschreibung der Dienste wird üblicherweise die *Web Service Description Language (WSDL)* verwendet, während die Parteien ihre Anfragen und Antworten über das *SOAP* Protokoll austauschen. Zusammen mit der UDDI API bilden diese Standards den Kern der Web Service Welt. In Abschnitt 1.1 werden diese Spezifikationen detaillierter betrachtet.

Die Beachtung dieser grundlegenden Standards reicht aus, um funktionsfähige Web Services implementieren zu können, die aber keine Sicherheitseigenschaften garantieren. D.h., simple SOAP Nachrichten können unbemerkt mitgelesen und verändert werden, und es ist dem Empfänger nicht möglich die Identität des Senders zuverlässig zu prüfen. Diese nicht nur wünschenswerten, sondern für elektronische Geschäftsprozesse auch unabdingbaren Schutzziele werden im Abschnitt 1.3 genau definiert. Darauf aufbauend werden anschließend Sicherheitsanforderungen für Web Services formuliert (Kap. 3), deren theoretische und praktische Umsetzung durch die entsprechenden Web Service Ergänzungen in den Kapiteln 4 und 5 gezeigt wird.

1.1 Kern-Standards

1.1.1 XML

XML steht für *Extensible Markup Language* [XML06] und ist eine Sprache zur effizienten Kapselung von Daten. Im Wesentlichen bietet XML eine textbasierte Beschreibungssprache, mit deren Hilfe beliebige Datenstrukturen definiert und plattform- und anwendungsunabhängig verarbeitet werden können. Nutzinformationen werden in XML von Tags umschlossen, welche die Bedeutung und logische Struktur der Daten beschreiben. Diese Tags werden außerhalb der XML Dokumente entweder in *Document Type Definitions* (DTDs) oder XML Schemas spezifiziert. XML ist vom *World Wide Web Consortium* (W3C) standardisiert und bildet die Basis für die Interoperabilität von Web Services.

1.1.2 SOAP

Die vier Buchstaben SOAP standen bis zur Version 1.2 [SOAP03] dieses Protokolls als Abkürzung für *Simple Object Access Protocol*. Seitdem ist SOAP ein eigenständiger Begriff, da die Langform den Gegenstand des Protokolls nicht korrekt erfasst.

Im Prinzip definiert SOAP lediglich ein Nachrichtenformat für unidirektionale und zustandslose Kommunikation mittels XML Dokumenten. Für dieses Nachrichtenformat kennt SOAP zwei Interaktionsstile – einen RPC-ähnlichen und einen Dokumentenstil – um eine Anfrage und eine Antwort zwischen dem Dienstanutzer und dem Dienstanbieter zu kommunizieren. Die Datentypen der Aufrufparameter und der Dienstanantwort müssen in beiden Fällen in XML so kodiert werden, dass die Kommunikationspartner die Nachrichten verstehen können. SOAP 1.2 stellt mit dem *SOAP Encoding* ein Format zur Kodierung von Datenstrukturen zur Verfügung, dessen Verwendung aber freigestellt ist. Beide Seiten können sich auf eine beliebige Form der Datenkodierung einigen.

Ebenso generisch ist der Transport der XML-Nachrichten gehalten. SOAP bestimmt kein Transportprotokoll, obwohl HTTP üblicherweise das Protokoll der Wahl ist. Es kann aber genauso gut SMTP oder jedes andere Transportprotokoll gewählt werden, das Dienstanutzer und -anbieter verstehen.

Da SOAP XML-basiert ist, können die bekannten Mechanismen aus dem XML-Security Standard zur Verschlüsselung und Signatur von XML Dokumenten angewendet werden. Abschnitt 4.1.1 geht tiefer auf SOAP Security ein.

1.1.3 WSDL

Die *Web Services Description Language* (WSDL) [WSDL06] ist eine Beschreibungssprache für die Schnittstellen, die von Web Services angeboten werden. Eine Servicebeschreibung in WSDL besteht aus einem abstrakten Teil, der die Dienstschnittstelle in Form von Datentyp- und Operationsdefinitionen beschreibt und einem konkreten Teil, der Web Service-spezifische *Bindings* ergänzt. Die Bindings eines Web Service bestehen aus Informationen über die Kodierung der Daten, die Protokolle zu ihrem Transport und Informationen zur Dienstlokalisierung. Diese Trennung von abstrakter und konkreter Schnittstelle erlaubt die Wiederbenutzung der abstrakten WSDL Spezifikation, die auf verschiedene Weisen implementiert werden kann.

WSDL ist damit den *Interface Description Languages* (IDL) klassischer Middleware sehr ähnlich. Allerdings verursacht die lose Kopplung von Web Services einige Unterschiede zu üblicher Middleware und somit auch zu üblichen IDLs. So müssen im konkreten Teil einer WSDL Beschreibung insbesondere die Zugriffsmechanismen für Web Services beschrieben werden, die bei klassischer Middleware durch einen impliziten Kontext gegeben sind.

1.1.4 UDDI

Der *Universal Description, Discovery, and Integration (UDDI)* Standard [UDDI05] definiert ein Framework, um Web Services kategorisieren und katalogisieren, registrieren und finden zu können. Dazu stellt UDDI einen Satz von Datenstrukturen und APIs bereit, um Metainformationen zu repräsentieren, zu durchsuchen und abzufragen. Diese Daten werden in einer *Business Registry* vorgehalten, deren Zugriffsoperationen wiederum mit einem Web Service implementiert sein können.

Es ist wichtig festzustellen, dass eine UDDI Registry keine Dienstbeschreibungen oder andere technische Informationen über einzelne Web Services speichert und dass der UDDI Standard keine Sprache zur Beschreibung von Web Services anbietet oder festlegt. UDDI definiert vielmehr einen erweiterbaren Metarahmen, um Schnittstellen, Beschreibungen, Dokumentationen, technische Konzepte oder ganz allgemeine Klassifikationen beschreiben zu können. Der Kern dieses Metarahmens ist das Technische Modell (*Technical Model* oder *tModel*), das so unterschiedliche Dinge wie eine Web Service Schnittstelle oder eine Eigenschaft, wie „deutschsprachige Dokumentation“ repräsentieren kann. Die tatsächliche Information, d.h. die Schnittstellenspezifikation, die in WSDL repräsentiert sein kann, oder die Definition der Kategorie „deutschsprachige Dokumentation“ liegt außerhalb der UDDI Registry auf Seiten des Diensteanbieters oder einer Standardisierungsorganisation und wird über eine URL referenziert.

Der UDDI Standard definiert sechs APIs von denen die *UDDI Inquiry API* und die *UDDI Publishers API* die wichtigsten Schnittstellen der Registry darstellen. Die Inquiry API kann von Applikations- und Dienstentwicklern bei der Implementierung von WS Clients genutzt werden, um geeignete Dienste und Anbieter zu finden. Clientprogramme können aber auch zur Laufzeit über diese API Web Services suchen, um sich dann dynamisch an diese zu binden. Die Publishers API richtet sich an Diensteanbieter, um neue Dienste in eine Registry einzutragen, vorhandene Dienste zu verwalten oder zu entfernen oder um Metainformationen über den Diensteanbieter selbst zu modifizieren.

1.2 Standards und Standardisierungsorganisationen – ein Kurzüberblick

Die Standards und die Spezifikationen, die in den Abschnitten 1.1 und 4 vorgestellt wurden bzw. werden, sind nicht von nationalen (wie z.B. DIN und NIST) oder internationalen (wie z.B. ISO und ETSI) Standardisierungsorganisation erlassen worden. Sie sind das Ergebnis einer Zusammenarbeit von Unternehmen und Instituten in offenen Organisationen wie dem *World Wide Web Consortium (W3C)* oder der *Organization for the Advancement of Structured Information Standards (OASIS)*. Während die OASIS die Endprodukte ihres Vereinheitlichungsprozesses Standards nennt, spricht die W3C von *Recommendations*, also Empfehlungen. Auch wenn letzteres deutlicher schwächer klingt besitzen die Ergebnisse beider Organisationen die Wirkungskraft von De-facto-Standards, da an ihrer Entwicklung und Umsetzung nahezu alle relevanten IT Unternehmen beteiligt sind.

Neben diesen Organisationen gibt es im Web Services Umfeld einige Spezifikationen, die von Koalitionen von Unternehmen ausgearbeitet worden sind, ohne sie einem offiziellen Standardisierungsprozess zu unterwerfen.

Tabelle 1 listet die für dieses Dokument relevanten Standards und Spezifikationen mit ihrer Versionsnummer, ihrem Status und der urhebenden Organisation auf. Alle Spezifikationen, deren Standardisierung noch nicht abgeschlossen sind, bekommen dort den Status 'Draft' zugewiesen. Weiterhin werden in diesem Dokument OASIS Standards und W3C Recommendations als Standards bezeichnet, während alle anderen Spezifikationen unabhängig vom Urheber oder Status des Drafts schlicht unter dem Begriff *Spezifikation* geführt werden.

Tabelle 1: Übersicht der relevanten Standards und Spezifikationen

<i>Name</i>	<i>Langform</i>	<i>Vers.</i>	<i>Datum</i>	<i>Urheber</i>	<i>Status</i>
WSDL [WSDL06]	Web Service Description Language	2.0	03.2006	W3C	Draft
XML [XML06]	Extensible Markup Language	1.1	08.2006	W3C	Standard
SOAP [SOAP03]	SOAP	1.2	06.2003	W3C	Standard
UDDI [UDDI05]	Universal Description, Discovery, and Integration	3.0.2	02.2005	OASIS	Standard
WSS [WSS06]	Web Services Security: SOAP Message Security	1.1	02.2006	OASIS	Standard
XMLDSig [XMLSig02]	XML-Signature Syntax and Processing	o.V.	02.2002	W3C	Standard
XMLEnc [XMLEnc02]	XML Encryption Syntax and Processing	o.V.	12.2002	W3C	Standard
SAML [SAML05]	Security Assertion Markup Language	2.0	05.2006	OASIS	Standard
WS-Policy [WS-Pol06]	Web Services Policy 1.5 - Framework	1.5	07.2006	W3C	Draft
WS-SecurityPolicy [WS-SecPol05]	Web Services Security Policy Language	1.1	07.2005	IBM, Microsoft, RSA, Verisign	Spezifikation
WS-Trust [WS-Trust06]	WS-Trust	1.3	09.2006	OASIS	Draft
WS-SecureConversation [WS-SC06]	WS-SecureConversation	1.3	09.2006	OASIS	Draft
WS-Federation [WS-Fed03]	Web Services Federation Language	1.0	07.2003	IBM, Microsoft, RSA, Verisign	Spezifikation

1.3 Schutzziele

Dieser Abschnitt führt die grundlegenden Eigenschaften ein, die man für gewöhnlich an geschäftliche Kommunikation stellt und deren Zusicherung man auch von einem IT-System verlangt. Die folgenden Definitionen schärfen diese im allgemeinen Sprachgebrauch zu vagen Begriffe, um die Einhaltung der von ihnen bezeichneten Eigenschaften in einem IT-System nachweisen oder widerlegen zu können. Dies erlaubt es, in Abschnitt 2 die Bedeutung von Angriffen auf IT-Systeme im Allgemeinen und Web Services im Besonderen besser zu beschreiben und deren Abwehr in Kapitel 4 vorzustellen.

1.3.1 Vertraulichkeit

Unter *Vertraulichkeit* oder *Informationsvertraulichkeit* (Confidentiality) von Daten oder Nachrichten versteht man die Zusicherung, dass diese nur einem autorisierten Kreis von Personen bekannt wird.

Ein IT-System gewährleistet Vertraulichkeit, wenn es Mechanismen bereitstellt, um Daten vor unautorisiertem Zugriff zu schützen. Dazu müssen Berechtigungen zur Verarbeitung

dieser Daten vergeben und entzogen werden können und Kontrollen vorhanden sein, die die Einhaltung dieser Berechtigungen durchsetzen. Weiterhin müssen Nachrichten während der Übertragung zum Empfänger vor unberechtigtem Mitlesen geschützt werden können.

1.3.2 Integrität

Die *Integrität* oder *Datenintegrität* (Integrity) einer Nachricht oder eines Datums bezeichnet deren Unverfälschtheit bzw. Vertrauenswürdigkeit.

Ein IT-System gewährleistet Integrität, wenn es einem unautorisierten Dritten nicht möglich ist, die zu schützenden Daten oder Nachrichten unbemerkt zu manipulieren.

1.3.3 Authentizität

Man spricht von der *Authentizität* (Authenticity) von Nachrichten oder Daten im Allgemeinen, wenn sie tatsächlich von dem Subjekt oder aus der Quelle kommen, aus der sie zu stammen vorgeben. Authentizität bedeutet also sichere Identifizierung der Kommunikationspartner.

Ein IT-System sichert Authentizität zu, wenn es Mechanismen bereitstellt, die die behauptete Authentizität der zu schützenden Nachrichten und Daten entweder bestätigen oder widerlegen. Das heißt, es ist in einem solchen System nicht möglich, Nachricht und Daten im Namen eines anderen Subjektes zu erstellen.

1.3.4 Verfügbarkeit

DIN 40042 definiert *Verfügbarkeit* (Availability) als „die Wahrscheinlichkeit, ein System zu einem gegebenen Zeitpunkt in einem funktionsfähigen Zustand anzutreffen.“ Ein IT-System soll seinen Usern den vereinbarten Zugriff gewährleisten und in seiner Verfügbarkeit nicht durch unautorisierte Aktionen oder gezielte Angriffe beeinträchtigt werden können.

1.3.5 Verbindlichkeit

Die Begriffe *Verbindlichkeit* und *Nicht-Zurückweisbarkeit* (Non-Repudiation) einer Nachricht oder Aktion werden synonym verwendet, um anzuzeigen, dass deren Sender bzw. Initiator seine Urheberschaft nicht zurückweisen kann. Verbindlichkeit transportiert daher auch immer die Identität des Urhebers der Aktion.

Der Begriff Verbindlichkeit wird in der IT sehr oft als Voraussetzung für die Rechtsverbindlichkeit von elektronischen Geschäftstransaktionen genannt. Diese Forderung ist im Allgemeinen zu stark.

1.3.6 Anonymität

Schutz der *Anonymität* bzw. *Privatsphäre* (Privacy) ist der sperrigste Begriff unter den Sicherheitseigenschaften, da das, was Privatsphäre ausmacht, nicht eindeutig fassbar ist und sowohl subjektiv als auch in der Rechtsprechung sehr unterschiedlich interpretiert wird. Die Privatsphäre des Einzelnen soll durch das *Bundesdatenschutzgesetz (BDSG)* und das *Teledienstschutzgesetz (TDDSG)* gesichert werden.

Ein IT-System, das die Privatsphäre seiner Nutzer schützt, sollte nur so viele Daten über seine Benutzer erheben und speichern, wie für die Erbringung des Dienstes notwendig sind (Datensparsamkeit) und diese auch nur für autorisierte Personen einsehbar machen. Insbesondere sollten technische und organisatorische Maßnahmen sicherstellen, dass keine Profile über das Verhalten der Benutzer erstellt werden können. Die anonyme Nutzung von Diensten ist die strikteste Form der Privatsphäre.

2 Potentielle Schwachstellen und Angriffsmöglichkeiten

In diesem Kapitel werden Schwachstellen und Angriffsmöglichkeiten besprochen, die beim Einsatz von Web Services zum Tragen kommen können. Nicht alle dieser Angriffe sind spezifisch für Web Services, aber alle können gegen Web Services angewendet werden. Abschnitt 2.1 führt zuerst einige ganz allgemeine Angriffsformen ein, die auf sehr unterschiedliche Arten durchgeführt werden können. In Abschnitt 2.2 werden gängige Angriffe vorgestellt, die auf Schwachstellen in Technologien und Programmcode beruhen. Den Abschluss bildet Abschnitt 2.3, der sich auf Web Service- und XML-spezifische Angriffe konzentriert.

2.1 Allgemeine Angriffsformen

2.1.1 Denial of Service

Ausgangssituation

Web Services und Webanwendungen bieten dem Benutzer entweder eine Schnittstelle zu einer weiteren Anwendung oder erbringen selbst die gewünschte Operation. Zur Erbringung der Leistung beansprucht der Service Ressourcen wie Prozessorzeit, Speicher, Datenbanken o.ä.

Schwachstelle

Die Eingaben des Benutzers werden nicht oder erst sehr spät im Verarbeitungsprozess auf ihre Gültigkeit überprüft.

Angriff

Der Angreifer verbraucht durch unautorisierte oder auch durch autorisierte Anfragen ein Übermaß an Ressourcen. Dabei sendet er entweder sehr viele kleine oder wenige sehr große Anfragen an das angegriffene System.

Bedrohung

Autorisierte Benutzer des Dienstes werden an der Durchführung ihrer berechtigten Anfragen gehindert.

Bedrohte Schutzziele

- Verfügbarkeit

Gegenmaßnahme

- Alle Übergabeparameter hinsichtlich der Einhaltung des Wertebereichs überprüfen.
- Benutzern und Dienstthreads Quotas für Ressourcennutzung zuweisen.
- Alle Anfragen so früh wie möglich im Verarbeitungsprozess auf Authentizität und Autorisierung überprüfen.

Referenzen

[PC04], [BSI05], [TH05], [GJ06]

Bemerkungen

Es existieren sehr viele Varianten und Formen von Denial of Service (DoS) Angriffen. So können z.B. Capture Replay Attacks (2.1.2) und Buffer Overflows (2.2.2) benutzt werden, um einen Server zu stören. Für dieses Dokument sind nur DoS Varianten interessant, die auf der Anwendungsebene verursacht werden.

2.1.2 Capture Replay Attack

Ausgangssituation

Web Services und Webanwendungen erwarten Anfragen in einer spezifischen Form, die meist sehr einfach analysiert und nachgeahmt werden kann.

Schwachstelle

Der Web Service bzw. die Anwendung verfügt über keine Maßnahmen, die sicherstellen, dass eine einkommende Anfrage authentisch und neu ist.

Angriff

Der Angreifer fängt Nachrichten im Netz ab und sendet sie unverändert zu einem späteren Zeitpunkt erneut, unter Umständen mehrfach, an den Server. In einer Variante des Angriffs manipuliert er Teile der Nachricht, wie z.B. Namen von Operationen, Werte von Parametern oder Benutzer-IDs.

Bedrohung

Die wiedereingespielten Anfragen werden erneut ausgeführt und eventuell anfallende Kosten dem Absender der Originalnachricht in Rechnung gestellt. Manipulierte Nachrichten können die Preisgabe vertraulicher Informationen bewirken. Sehr viele Wiederholungen können zu einer Überlastung des Servers führen.

Bedrohte Schutzziele

- Unter Umständen alle, wenn Felder der Nachricht unbemerkt verändert werden können.

Gegenmaßnahme

- Nachrichten mit Timestamps und eindeutigen Identifiern versehen, die mit einer Historie abgeglichen werden.
- Nachrichten gegen unbemerkte Manipulation schützen; z.B. durch digitale Signaturen

Referenzen

[TH05], [GJ06]

Bemerkungen

Capture Replay Attacks bilden die Basis für Man-in-the-Middle Attacks, wie sie in Abschnitt 2.1.3 definiert sind.

2.1.3 Man-in-the-Middle Attack

Ausgangssituation

Web Services und Webanwendungen erwarten Anfragen in einer spezifischen Form, die meist sehr einfach analysiert und nachgeahmt werden kann.

Schwachstelle

Der Web Service verfügt über keine geeigneten Maßnahmen, die sicherstellen, dass eine einkommende Anfrage authentisch und neu ist.

Angriff

Wie beim Capture Replay Angriff (2.1.2) fängt der Angreifer Anfragen vom Dienstanutzer ab und manipuliert diese, bevor er sie an den Server weiterleitet. Zusätzliches Wissen über eventuell vorhandene Lücken in den verwendeten Protokollen und Anwendungen ermöglicht es ihm, sich durch geschicktes Verhalten zwischen zwei Kommunikationspartnern zu schmuggeln. Dabei gaukelt er beiden Seiten die jeweils andere vor.

Bedrohung

Dienstanutzer und Dienstanbieter glauben, einen vertraulichen und authentischen Kanal zu teilen, obwohl alle Nachrichten vom Angreifer gelesen und verändert werden. Der Angreifer kann sich unbemerkt gegenüber beiden Partnern als die jeweils andere Seite ausgeben.

Bedrohte Schutzziele

- alle

Gegenmaßnahme

- Zertifikate oder Schlüsselmaterial über einen anderen, sicheren Kanal austauschen, um eine vertrauliche und authentische Session aufbauen zu können.
- Eine gemeinsame Trusted Third Party (TTP) benutzen, um Zertifikate oder Schlüsselmaterial auszutauschen.
- Nachrichten gegen unbemerkte Manipulation schützen; z.B. durch digitale Signaturen oder Kanalverschlüsselung (SSLv3/TLS).

Referenzen

[TH05], [GJ06]

Bemerkungen

Der Begriff Man-in-the-Middle Attack wird gelegentlich auch als allgemeiner Oberbegriff für Angriffe auf Kommunikationskanäle benutzt, bei denen der Angreifer in irgendeiner Form zwischen den eigentlichen Kommunikationspartnern aktiv wird.

2.2 Standardangriffe auf Webanwendungen

Die folgenden Angriffe beruhen auf Schwachstellen in Software, die durch schlechten Code oder fehleranfällige Technologien verursacht werden. Sie haben insbesondere durch den Einsatz von Webtechnologien, wie z.B. Web Services, an Gefährlichkeit gewonnen. Diese

Angriffe stellen aber genauso für Standalone-Anwendungen, Infrastrukturserver und andere IT-Komponenten eine Bedrohung dar.

2.2.1 SQL Code Injection

Ausgangssituation

Viele Web Services und Webanwendungen benötigen eine Datenbank und bieten dem Benutzer einen indirekten Zugriff auf diese an. So ist es eventuell nötig, dass sich der Benutzer über ein Webformular registriert oder Suchparameter eingibt. Diese Eingaben werden anschließend in SQL Abfragestrings übernommen und der Datenbank zur Ausführung übergeben.

Schwachstelle

Die Übergabeparameter werden ungeprüft in die SQL-Abfragen übernommen und zur Ausführung gebracht.

Angriff

Der Angreifer schleust durch geschickt formulierte Übergabeparameter SQL Code in die Abfragen an die Datenbank des Dienstes.

Bedrohung

Es werden nicht autorisierte Abfragen gegen die Datenbank durchgeführt oder im schlimmsten Fall SQL Kommandos wie das Manipulieren und Löschen von Datenbankeinträgen ausgeführt.

Bedrohte Schutzziele

- Vertraulichkeit
- Integrität

Gegenmaßnahme

- Code Review und statische Codeanalyse durchführen.
- Alle Übergabeparameter hinsichtlich der Einhaltung des Wertebereichs überprüfen (Inputvalidierung).
- SQL-Abfragen mit einem technischen Benutzer durchführen, der nur die minimal benötigten Rechte für die Datenbank besitzt.
- Prepared statements

Referenzen

[OWA1], [OWA2], [WAS], [SEC06], [GJ06]

2.2.2 Buffer Overflows

Ausgangssituation

Viele Web Services und Webanwendungen bieten eine Schnittstelle zu weiteren Anwendungen im Backend an. An diese werden mittels Übergabeparametern Eingaben des Benutzers oder anderer Anwendungen weitergereicht.

Schwachstelle

Die Backend-Anwendung ist in einer Sprache wie C geschrieben, die kein inhärentes Speichermanagement bietet und in der der Entwickler den Speicher selber allokiert und freigeben muss. Die Eingaben des Benutzers werden an einen Eingabebuffer übergeben, ohne zuvor zu prüfen, ob der allokierte Speicher ausreicht, um sie aufzunehmen.

Angriff

Der Angreifer tätigt unerwartete Eingaben, die auch Programmcode enthalten können.

Bedrohung

Die Eingaben, die nicht vom Buffer aufgenommen werden können, überschreiben andere Daten und bringen das Programm zum Absturz oder verursachen Fehlverhalten. Geschickte Eingaben können Programmcode enthalten, der unter Umständen zur Ausführung kommt.

Bedrohte Schutzziele

- Alle, da unter Umständen beliebiger Programmcode mit den Rechten der Anwendung ausgeführt werden kann

Gegenmaßnahme

- Code Review und statische Codeanalyse durchführen.
- Alle Übergabeparameter hinsichtlich der Einhaltung des Wertebereichs überprüfen.
- Die Legacy Anwendung mit einem technischen Benutzer durchführen, der nur die minimal benötigten Rechte besitzt.

Referenzen

[OWA1], [OWA2], [WAS], [SEC06], [GJ06]

Bemerkungen

Ein Buffer Overflow ist auch in einer Web-Anwendung oder einem Web Service selbst möglich, wenn diese in einer Programmiersprache geschrieben sind, die die oben erwähnte Schwäche aufweisen.

2.3 Web Service-spezifische Angriffe

Neben den zuvor beschriebenen Angriffen gibt es bisher wenige Attacks, die wirklich Web Service spezifisch sind, wie z.B. WSDL / Access Scanning (2.3.1) und XML Rewriting (2.3.6). Der Einsatz von Web Services verleiht aber bekannten Bedrohungen, die in der Natur von XML und XML Parsern liegen, mehr Gewicht.

2.3.1 WSDL und Access Scanning

Ausgangssituation

Ein WSDL Dokument beinhaltet Informationen über die von einem Web Service bereitgestellten Operationen, die Ein- und Ausgabeparameter, die die Operationen annehmen beziehungsweise ausgeben, sowie die URI, hinter denen die Operationen verfügbar sind. Diese Informationen werden meist, zusammen mit den Wrapperklassen für die anzubindende Anwendung, automatisch generiert.

Schwachstelle

Wenn Wrapperklassen und WSDL-Dateien für SW-Komponenten automatisch generiert werden, dann geschieht dies für alle Operationen der Komponente – auch für diejenigen, die

nicht öffentlich zugreifbar sein sollen. Auch wenn die URI dieser Operationen aus der WSDL-Datei entfernt werden, sind sie noch über den Web Service erreichbar.

Angriff

Der Angreifer “errät” auf Grund öffentlich zugänglicher Informationen über einen Web Service mögliche weitere URI verfügbarer Operationen. Wenn es eine Operation `getItem` gibt, dann gibt es vielleicht auch `setItem`.

Bedrohung

Der Angreifer kann unautorisiert Operationen ausführen, die nicht für den externen Zugriff gedacht sind. Da Web Services primär dazu gedacht sind, Geschäftsanwendungen anzubinden, erweist sich diese Bedrohung als doppelt gefährlich. Zum einen sind dies die schützenswertesten Anwendungen des Unternehmens, zum anderen handelt es sich oft um Legacy Anwendungen, die nicht für den Zugriff über offene Netzwerke programmiert worden sind. Ist also ein Angreifer um die Zugriffskontrollen der Service Schnittstelle herumgekommen, gibt es praktisch keinen Schutz mehr.

Bedrohte Schutzziele

- Eventuell alle; je nachdem, welche Operationen offen gelegt werden

Gegenmaßnahme

- WSDL und Wrapperklassen manuell von Operationen reinigen, die nur für den internen Gebrauch gedacht sind.
- Tests durchführen, die explizit versuchen, interne Operationen anzusprechen.
- Per Default Policy alle Web Service Zugriffe verbieten und nur die gewünschten Operationen nach entsprechender Zugriffskontrolle erlauben.

Referenzen

[OWA1], [WAS], [GJ06]

2.3.2 External Entity Attacks

Ausgangssituation

XML ist die Sprache der Web Services. SOAP Nachrichten, WSDL Beschreibungen, UDDI Einträge, Systemkonfigurationen, etc. sind XML-Dokumente. Diese können wiederum andere XML-Dokumente an beliebigen Stellen im Netz über URI referenzieren.

Schwachstelle

Die Integrität der URI in WSDL, SOAP oder anderen XML-Dokumenten wird nicht gewährleistet oder die referenzierte Lokation ist nicht schreibgeschützt.

Angriff

Der Angreifer manipuliert die ungeschützte URI, indem er sie auf ein XML-Dokument mit böartigem oder vertraulichen Inhalt verbiegt (s.a. 2.3.3).

Bedrohung

Dies kann zu einem DoS Angriff oder zur Preisgabe von vertraulichen Informationen führen.

Bedrohte Schutzziele

- Vertraulichkeit
- Verfügbarkeit

Gegenmaßnahme

- Generelle Skepsis, beim Referenzieren und Einbinden von XML-Dokumenten von öffentlichen Lokationen.
- Die Integrität von Referenzen mit XML-Signature schützen.

Referenzen

[OWA1], [WAS], [GJ06]

2.3.3 XML Bombs

Ausgangssituation

XML ist die Sprache der Web Services. SOAP Nachrichten, WSDL Beschreibungen, UDDI Einträge, Systemkonfigurationen, etc. sind XML-Dokumente. XML-Dokumente können DTDs enthalten, die neue Entitäten einführen. Diese wiederum können in dem XML-Dokument gleich benutzt werden.

Schwachstelle

Die DTD-Verarbeitung von XML-Parsern kann ausgenutzt werden, um exponentiell wachsende Daten zu generieren.

Angriff

Der Angreifer sendet eine SOAP Nachricht mit einer böartigen DTD-Struktur an den Web Service. Der Aufbau kann folgendermaßen aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE SOAP-ENV:Envelope [
<!ENTITY x0 "BIG">
<!ENTITY x1 "&x0&x0">
<!ENTITY x2 "&x1&x1">
. .
<!ENTITY x99 "&x98&x98">
<!ENTITY x100 "&x99&x99">]>
<SOAP:Envelope entityReference="&x100"
  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
. . .
  </SOAP:Body>
</SOAP:Envelope>
```

Die Entitätsreferenz im Envelope-Tag veranlasst den XML-Parser, &x100 durch &x99&x99 zu ersetzen. Da dies auch wieder Referenzen sind, werden auch diese zu &x98&x98&x98&x98 aufgelöst usw. Der XML-Parser würde also versuchen ein Element bestehend aus 2 hoch 100 Mal der Zeichenkette "BIG" zu erzeugen.

Bedrohung

Der Angreifer kann mit sehr kleinen XML-Dokumenten den kompletten Speicher des Systems beanspruchen und so den Dienst für andere Benutzer lahm legen.

Bedrohte Schutzziele

- Verfügbarkeit

Gegenmaßnahme

- SOAP-Stacks verwenden, die die DTD-Verarbeitung verbieten.
- XML-Parser so konfigurieren, dass die DTD-Verarbeitung unterdrückt wird.

Referenzen

[OWA1], [WAS], [GJ06]

2.3.4 Large Payloads

Ausgangssituation

SOAP Nachrichten müssen komplett geparsed werden, um ihre Wohlgeformtheit zu überprüfen. Dies ist eine sehr rechenintensive Arbeit, insbesondere wenn die Nachrichten in DOM-Objekte überführt werden.

Schwachstelle

Einige SOAP Stacks benutzen DOM-Parser, die einkommende Nachrichten nicht nur parsen, sondern auch in einen Objektbaum überführen. Dabei werden alle für alle Elemente der Nachrichten Objekte erzeugt, sobald sie eingelesen werden. Dies verbraucht CPU und Speicher bevor sichergestellt worden ist, ob das Dokument gültig ist und die Objektrepräsentationen der Nachrichtenelemente bereits zu diesem Zeitpunkt benötigt werden.

Angriff

Der Angreifer sendet eine oder mehrere SOAP Nachrichten von sehr großer Größe an den Web Service.

Bedrohung

Der Angreifer kann mit einfachen SOAP Nachrichten den XML-Parser der Empfängerseite überlasten und so den Dienst für andere Benutzer lahm legen.

Bedrohte Schutzziele

- Verfügbarkeit

Gegenmaßnahme

- Wenn möglich SAX-Parser benutzen.

Referenzen

[OWA1], [WAS], [GJ06]

2.3.5 XPath Injections

Ausgangssituation

XML Path Language (XPath) ist eine Sprache, um schnelle Abfragen über XML-Dokumente zu ermöglichen, ohne das komplette Dokument parsen zu müssen. Webanwendungen erzeugen oft XPath Abfragen aus Eingabeparametern, um Benutzern den Zugriff auf Teile von XML-Dokumenten zu ermöglichen.

Schwachstelle

Die Übergabeparameter werden ungeprüft in die XPath-Abfragen übernommen und zur Ausführung gebracht. Insbesondere werden keine Escape-Zeichen herausgefiltert.

Angriff

Der Angreifer schleust durch geschickt formulierte Übergabeparameter XPath Code in die Abfragen an das XML-Dokument. Folgendes Beispiel erläutert die Schwachstelle und das Vorgehen des Angreifers.

```
<Kunden>
  <Kunde Benutzername="Max" Passwort="geheim">
    <Kreditkartennr>
      8999723455
    </Kreditkartennr>
    <Adresse>
      <Strasse>
        ...
      </Strasse>
    </Adresse>
  </Kunde>
  <Kunde Benutzer="Max" Passwort="geheim">
    ...
  </Kunde>
</Kunden>
```

Ein Unternehmen hat seine Kundendaten in einer XML Datei gespeichert und bietet seinen Kunden Einsicht in diese Daten mit Hilfe eines Web Services. Dazu senden die Kunden eine verschlüsselte SOAP Nachricht mit ihrem Benutzernamen und Passwort, die vom Service in eine XPath Abfrage übernommen werden. Die zugehörige Zeile Java Code könnte so aussehen:

```
String xpathAbfrage = "/Kunden/Kunde[attribute::Benutzername =" + username
+ "][attribute::Passwort =" pw]/Kreditkartennr";
```

und würde das `Kreditkartennr` Element zurückgeben, wenn die Variablen `username` und `pw` die Werte "Max" und "geheim" halten. Die Kreditkartennummer wird aber auch preisgegeben, wenn ein Angreifer den Kundennamen Max kennt und "" or 1 = 1" als Passwort übergibt. Nach dem Einsetzen der Parameter würde der XPath Abfragestring wie folgt lauten.

```
/Kunden/Kunde[attribute::Benutzername = Max] [attribute::Passwort = '' or 1=1]/Kreditkartennr
```

Dieser Angriff folgt dem selben Muster wie SQL Code Injection (2.2.1).

Bedrohung

Es werden nicht autorisierte und nicht gewünschte Abfragen auf dem XML-Dokument ausgeführt, was zur Preisgabe von vertraulichen Informationen führen kann.

Bedrohte Schutzziele

- Vertraulichkeit

Gegenmaßnahme

- Code Review und statische Codeanalyse durchführen.
- Alle Übergabeparameter hinsichtlich der Einhaltung des Wertebereichs überprüfen (Inputvalidierung).

Referenzen

[OWA1], [WAS], [GJ06]

2.3.6 XML Rewriting

Ausgangssituation

Die Web Service Security Policy Sprache (4.1.4) erlaubt es unter anderem für einen Web Service zu spezifizieren, welche Felder in einer SOAP Anfrage der Service erwartet und welche Felder durch eine digitale Signatur oder Verschlüsselung geschützt werden müssen. Die zu signierenden Felder einer SOAP Nachricht müssen nicht unbedingt aufeinander folgen. Welche Teile einer SOAP Nachricht in die Signatur eingegangen sind, geht aus dem beschreibenden Feld `<SignedInfo>` hervor, das Referenzen auf die signierten Elemente in der XML Nachricht enthält.

Eine signierte MessageID zusammen mit einem signierten Zeitstempel soll den Dienst gegen Replay-Angriffe schützen.

Schwachstelle

Die MessageID wird in der WS-Policy des Dienstes als zu signierendes Element spezifiziert, aber nicht in die Liste der Felder aufgenommen, die obligatorisch sind.

Angriff

Ein Angreifer fängt eine autorisierte SOAP Nachricht für den falsch konfigurierten Dienst ab und manipuliert diese für einen Replay-Angriff wie in den folgenden Schritten beschrieben. Dabei bleibt die Signatur erhalten, aber der Web Service wird das MessageID Feld trotzdem ignorieren. Hier ist ein Auszug aus der fehlerhaften Policy.

```
<Policy Id="MyPolicy">
  <MessagePredicate>
    Body() Header(To) Header(Action)
  </MessagePredicate>
  <Integrity>
    <TokenInfo>
      <SecurityToken> [...] </SecurityToken>
    </TokenInfo>
    <MessageParts>
      Body() Header(To) Header(Action)
      Header(MessageID) Timestamp()
    </MessageParts>
  </Integrity>
</Policy>
```

Das Element `<MessagePredicate>` gibt an, welche Elemente eine SOAP Nachricht enthalten muss, wenn sie von diesem Dienst akzeptiert werden soll. Es wurde ursprünglich in WS-PolicyAssertion [WS-PolAss] als eines der grundsätzlichen Prädikate für den Gebrauch mit WS-Policy eingeführt. Das `<Integrity>` Element enthält die SOAP-Felder, die durch eine Signatur geschützt werden müssen. Das MessageID Feld muss signiert werden, aber es wird nicht als obligatorisches Feld geführt. Auf den ersten Blick scheint dies kein Fehler, aber es gibt einen subtilen Angriff.

Angenommen der Angreifer fängt die folgende SOAP Nachricht ab.

```

<Envelope>
  <Header>
    ...
    <MessageID Id="Id1">uuid:764c...</MessageID>
    <Security mustUnderstand="1">
      [...]
    <Signature>
      <SignedInfo>
        [...]
        <SignatureMethod Algorithm="...#rsa-sha1" />
        <Reference URI="#Id1">
          [...]
          <DigestMethod Algorithm="...#sha1" />
          <DigestValue>f5d7a...=</DigestValue>
        </Reference>
        <Reference URI="#Id3">
          [...]
          <DigestMethod Algorithm="...#sha1" />
          <DigestValue>9gz5K...=</DigestValue>
        </Reference>
      </SignedInfo>
      <SignatureValue>e4EyW...=</SignatureValue>
      <KeyInfo> [...]</KeyInfo>
    </Signature>
  </Security>
</Header>
<Body Id="Id3">
  <MyServiceRequest>...</MyServiceRequest>
</Body>
</Envelope>

```

Diese Nachricht enthält im Wesentlichen drei relevante Teile. Zum einen das Header Feld MessageID, den SOAP Body mit der Anfrage und die Signatur. Das `<SignedInfo>` Element spezifiziert den Umfang der Signatur durch `<Reference>` Elemente, die in diesem Fall die lokale URI der MessageID und des Body sind. Dadurch, dass diese Elemente durch eine Referenz in die Signatur eingebunden werden, können sie innerhalb des XML Dokumentes an beliebigen Stellen stehen ohne die Signatur ungültig zu machen. Dies erlaubt es dem Angreifer ein unbekanntes Element in den Header einzufügen, das die MessageID 'versteckt'. Der folgende XML Auszug zeigt die manipulierte Nachricht mit den rot markierten Ergänzungen.

```

<Envelope>
  <Header>
    ...
    <Wrap-it>
      MessageID Id="Id-1">uuid:764c...</MessageID>
    </Wrap-it>
    <Security mustUnderstand="1">
      [...]
    <Signature>

```

Der Parser des Empfängers erkennt das `<Wrap-it>` Element nicht und ignoriert es daher samt seinem Inhalt, der MessageID. Wegen der weiterhin gültigen Signatur kann der Empfänger diesen Replay nicht erkennen.

Bedrohung

Der Replay-Angriff kann für einen DoS Angriff genutzt werden. Bei kostenpflichtigen Diensten kann er auch genutzt werden, um den Kunden des Dienstbieters zu schaden.

Bedrohte Schutzziele

- Authentizität
- Integrität
- Verbindlichkeit
- Verfügbarkeit

Gegenmaßnahme

- Statische Policyanalyse hinsichtlich dieses Angriffes durchführen.
- Generell die komplette Nachricht signieren.

Referenzen

[Bhar05], [RRS06]

3 Sicherheitsanforderungen

In diesem Kapitel werden die Sicherheitsanforderungen für Web Services bestimmt und erläutert. Um die Sicherheitsanforderungen bestimmen zu können, führen wir zunächst in Abschnitt 3.1 die einzelnen Schritte von der Web Services Entwicklung bis hin zur eigentlichen Nutzung durch eine andere (WS-)Applikation ein. Hierbei nehmen wir Bezug auf das typische WS-Dreieck wie es bereits in Kapitel 1, Abbildung 1 dargestellt wurde. Nachdem die einzelnen Schritte definiert sind, werden in Abschnitt 3.2 die grundlegenden Sicherheitsanforderungen schrittweise erarbeitet. In Abschnitt 3.3 definieren wir ein Referenzmodell und ordnen die zuvor erarbeiteten Sicherheitsanforderungen den durch das Modell definierten Referenzpunkten zu (Abschnitt 3.4).

Wie die in diesem Kapitel identifizierten Sicherheitsanforderungen mittels verfügbarer Sicherheitsmechanismen für Web Services umgesetzt werden können, wird dann im nächsten Kapitel (siehe Abschnitt 4.2) behandelt.

3.1 Phasen der WS-Entwicklung

Wie bereits in Kapitel 1 erwähnt, dienen Web Services primär zur Umsetzung von verteilten Applikationen, welche Dienstaufrufe über heterogene Systeme hinweg ermöglichen. Da hierbei häufig Firmengrenzen überschritten werden und das Internet als Transportmedium genutzt werden kann, wird das Firmennetz neuen bzw. anderen Gefahren ausgesetzt, als wenn WS-Applikationen lediglich Firmen-intern eingesetzt werden.

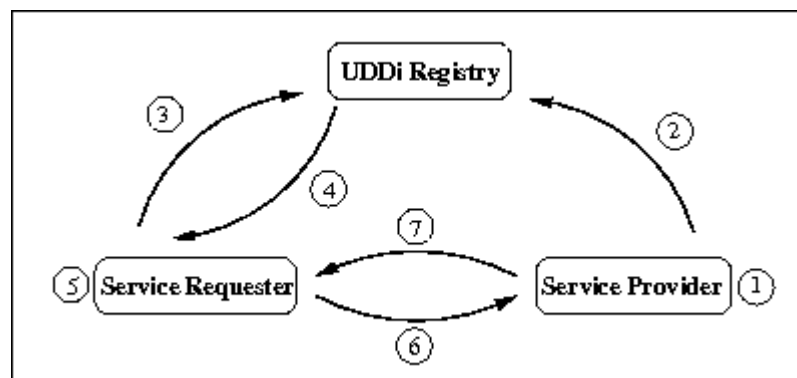


Abbildung 2: Phasen der WS-Entwicklung (in Anlehnung an [Har03])

Um die einzelnen Phasen bei der WS-Entwicklung untersuchen zu können, gehen wir zunächst näher auf das bereits in Kapitel 1 eingeführte WS-Dreieck ein. In Abbildung 2 sind die einzelnen Schritte von der Entwicklung eines Web Services durch einen Service Provider bis hin zur Einbindung und Nutzung in einer (WS-)Applikation durch einen Service Subscriber / Requester dargestellt. Die einzelnen Schritte lassen sich auf abstrakter Ebene wie folgt beschreiben (in Anlehnung an [Har03]):

1. Entwicklung und Dokumentation eines Web Services

Der zu entwickelnde Web Service, den eine Organisation, das heißt der Service Provider, anbieten möchte, muss nicht zwangsweise ein neuer Dienst sein. Häufig werden nur so genannte Web Service-Wrapper für Legacy-Anwendungen entwickelt. Der neue Web Service stellt hierbei lediglich eine vorhandene Applikation über ein

neues Web Service-Front-End zur Verfügung. In beiden Fällen sollten aber die Schnittstellen zum Web Service dokumentiert werden.

2. *Veröffentlichung des Web Services*

In diesem Schritt erfolgt die Beschreibung der Serviceschnittstellen, im Idealfall mittels WSDL, und die Registrierung des neuen Web Services in einer UDDI-Registry (siehe Kapitel 1.1.4 bzgl. Details). Es ist allerdings nicht zwingend nötig, einen Web Service in einer UDDI-Registry zu publizieren, falls dieser nur einem eingeschränkten Nutzerkreis zugänglich gemacht werden soll. Wurde also ein Web Service nur für den internen Gebrauch bzw. für einen festen Partner entwickelt, so fallen die Schritte 2 – 4 weg. Ein Szenario, bei dem dies der Fall sein könnte, wäre zum Beispiel ein Szenario aus dem Bankenumfeld. Weitere Möglichkeiten einen neuen Web Service bekannt zu machen, können auch Publikationspapiere oder die Verbreitung durch einen anderen Web Service sein.

3. *Suche nach Service-Anbietern*

In der nächsten Phase suchen potentielle Service-Nutzer nach einem Anbieter für einen gewünschten Dienst, den Service Provider.

4. *Erhalt einer Liste mit Service-Anbietern*

Als Resultat von Schritt 3 erhält der potentielle Service Subscriber / Requester von der UDDI-Registry eine Liste der möglichen Service Provider, die einen derartigen Web Service anbieten. Die Liste beinhaltet nicht die eigentlichen Service-Beschreibungen, sondern lediglich Referenzen auf die jeweiligen Servicebeschreibungen der Dienste (siehe Kapitel 1.1.4 bzgl. Details).

5. *Entwicklung einer (Web Services-)Applikation*

Nun kann der Service Subscriber / Requester eine eigene (WS-)Applikation entwickeln, welche einen (evtl. auch mehrere) der erfragten Dienste aus der UDDI-Registry anhand der über die Referenz beim Service-Provider erhaltenen Schnittstellenbeschreibung und Nutzungsbedingungen in Anspruch nimmt.

6. *Web Service-Anfrage*

Die Endphase beginnt mit Schritt 6, in welchem die eigentliche Service-Anfrage (Service Request) vom Service Subscriber / Requester (bzw. von dessen Applikation) an den Service Provider gestellt wird. In dieser Phase werden WSDL und die UDDI-Registry nicht mehr benötigt.

7. *Web Service-Antwort*

In Schritt 7 erfolgt dann die Service-Antwort (Service Response) durch den Service Provider an den Service Subscriber / Requester (bzw. die anfragende Applikation).

Nachdem die einzelnen Phasen bei der WS-Entwicklung näher betrachtet wurden, werden wir im nächsten Abschnitt die grundlegenden Sicherheitsanforderungen bestimmen.

3.2 Bestimmung der grundlegenden Sicherheitsanforderungen

Bevor wir im nächsten Abschnitt die Sicherheitsanforderungen (SA) jeweils zwischen zwei der am WS-Dreieck beteiligten Kommunikationspartner bestimmen, gehen wir in diesem Kapitel zunächst allgemein auf die grundlegenden Sicherheitsanforderungen ein.

SA1: Gewährleistung der Informationsvertraulichkeit (Confidentiality)

SA2: Gewährleistung der Datenintegrität (Integrity)

Betrachten wir zuerst die Kommunikationskanäle zwischen den einzelnen Entitäten. Diese erstrecken sich über das Internet, dies bedeutet auch über öffentliche Netze. Die unter

Umständen vertraulichen Daten sollten auf den Kommunikationskanälen daher vor Abhören, Verändern oder Löschen von Informationen geschützt werden. Gefordert werden somit die Gewährleistung der *Informationsvertraulichkeit* und die *Datenintegrität* der übertragenen Nachrichten (Daten) auf dem Kommunikationskanal.

In komplexeren Szenarien werden eventuell die Daten auch von Zwischenstationen auf dem Weg zum finalen Empfänger weiterverarbeitet bzw. weitergeleitet. Daher sollte die *Ende-zu-Ende-Sicherheit* gefordert werden, sodass vertrauliche Daten nicht auf Vermittlerentitäten, wenn auch meistens nur zeitweise, ungeschützt vorliegen. Ende-zu-Ende-Sicherheit bedeutet in diesem Zusammenhang, dass die Sicherheit der Nachrichten (Daten) auf dem zurückgelegten Pfad der Requests und Replies zwischen WS-Applikationskomponenten zu jeder Zeit und an jedem Punkt im Netz ohne Unterbrechungen gewährleistet ist.

Da sich die Kommunikation bei WS-Applikationen in der Regel über mehrere Anfragen und Antworten erstreckt, sollte ein *Sicherheitskontext* (Security Context), das heißt eine Kombination aus einer Identität und den sicherheitsrelevanten Attributen eines Kommunikationspartners, erzeugt und somit die *Sicherheitssitzung* (Security Session) bewahrt werden, damit nicht jede Anfrage erneut authentifiziert und autorisiert werden muss. Bei diesem Ansatz etablieren und teilen sich somit die Kommunikationspartner einen Sicherheitskontext zwecks eines längeren Nachrichten- bzw. Datenaustausch. Wie oben bereits erwähnt, können auch mehrere Verarbeitungsschritte an Zwischenstationen erfolgen. Daher sollte ein über alle beteiligten Entitäten bzw. (WS-)Applikationen verteilter Sicherheitskontext etabliert werden.

SA3: Gewährleistung der Authentizität (Authenticity)

Die Absicherung des Kommunikationskanals, ohne sicher zu sein, wer mit wem kommuniziert, reicht nicht aus. Um einen sicheren Web Service bzw. eine WS-Applikation anbieten bzw. betreiben zu können, müssen die Kommunikationspartner auf beiden Seiten identifiziert und deren Echtheit und Glaubwürdigkeit überprüft werden. Das heißt das Schutzziel *Authentizität* muss gewährleistet werden. Hierzu sollte eine gegenseitige Authentifizierung durchgeführt werden, sodass auf beiden Seiten sichergestellt ist, dass der Kommunikationspartner auch wirklich der ist, für den er sich ausgibt. Wird zum Beispiel bei der Kommunikation zwischen Service Subscriber / Requester und Service Provider nur eine Authentifizierung des Service Subscribers / Requesters durchgeführt, so kann dieser nicht sicher sein, dass vertrauliche Daten an einen Angreifer gesendet werden, anstatt an den gewünschten Service Provider.

Es könnte auch ein *Single Sign-On-Konzept* (SSO) realisiert werden. Hierbei muss sich ein Benutzer oder Dienst nur einmal gegenüber einer Instanz aus einer Gruppe von gemeinsam administrierten Diensten oder Applikationen authentifizieren. SSO bedeutet, dass die Instanzen dieser Gruppe untereinander die jeweiligen Authentifizierungsdaten akzeptieren. Der Zugang zu den Diensten wird dann automatisch, ohne Interaktion mit dem Nutzer, authentifiziert. Es ist allerdings zu beachten, dass ein erfolgreicher Angriff auf ein SSO-System einen großen Wirkungsbereich haben kann (s.a. Kap 4.2.2).

Um beim Einsatz von mehreren Web Services die Vielzahl an Identitäten, dies bedeutet Nutzerkennungen und personenbezogene Informationen, die für den Zugriff auf unterschiedliche (WS-)Applikationen bzw. deren Ressourcen nötig sind, auf eine sichere Weise konsistent, ständig verfügbar und verlässlich bereithalten zu können, muss ein sicheres *Identitätsmanagement* (Identity Management) umgesetzt werden. Hierbei sollte ein Identitätsmanagementsystem sämtliche Identitäten eines Nutzers durch eine

Zusammenführung der unterschiedlichen Kennungen auf eine Identität reduzieren, wodurch eine einfachere und sicherere Benutzerverwaltung möglich wird.

SA4: Gewährleistung der Autorisation (Authorization)

Nachdem die Kommunikationspartner authentifiziert sind, müssen ihnen Attribute, wie zum Beispiel Identität, Rollen, Gruppen oder Freigaben durch die Vergabe von Credentials (*Autorisation*), als Resultat der Authentifizierung, zugeordnet werden. Die Credentials dienen als Ausweise. Sie bescheinigen, wozu ein Kommunikationspartner berechtigt ist. Dies bedeutet, dass ein Mechanismus, nämlich eine *Zugriffskontrolle*, gewährleisten sollte, dass einem Kommunikationspartner nur Zugriff auf einen Web Service bzw. dessen Ressourcen gewährt wird, falls die Credentials des anfragenden Kommunikationspartners - die zugeordneten Attribute - dies erlauben. Ist dies der Fall, so ist der Kommunikationspartner autorisiert (berechtigt) den gewünschten Zugriff durchzuführen.

SA5: Gewährleistung der Verfügbarkeit (Availability)

Des Weiteren muss eine Entität, die einen Web Services, eine (WS-)Applikation oder einen sonstigen Dienst anbietet auch immer verfügbar sein. Dies bedeutet authentifizierte und autorisierte Kommunikationspartner sollten in der Wahrnehmung ihrer Berechtigungen nicht beeinträchtigt werden können. Somit wird die Gewährleistung der *Verfügbarkeit* eines Dienstes gefordert. Dies erfordert einerseits infrastrukturelle Maßnahmen, die hier nicht beschrieben werden sollen und andererseits Schutz gegen DoS-Angriffe auf der Anwendungsebene. Insbesondere muss der Web Service gegen Replay-Angriffe geschützt werden.

SA6: Gewährleistung der Verbindlichkeit (Non-Repudiation)

Die Nutzung eines Web Services ist in vielen Fällen kostenpflichtig. Wird also ein Dienst durch einen anderen Dienst bzw. eine andere Applikation aufgerufen, so muss dies verbindlich geschehen, dies bedeutet, die Nutzung darf nicht im Nachhinein abstreitbar sein. Für die Abrechenbarkeit (Accountability) eines Dienstes ist daher die Gewährleistung der *Verbindlichkeit* zu fordern. Mit dieser Forderung ist auch die Forderung nach der Rückverfolgbarkeit (Traceability) eng verbunden. Hierzu zählen somit auch die Protokollierung (Logging) und die Überwachung (Auditing) von relevanten Ereignissen. Hierunter fallen zum Beispiel Authentifizierungsversuche, sowohl erfolgreich als auch nicht erfolgreich durchgeführte Zugriffe auf (sensitive) Daten bzw. Ressourcen, Verletzungen einer Sicherheitsrichtlinie etc. Bezüglich Protokollierung und Überwachung sind allerdings auch datenschutzrechtliche Bestimmungen einzuhalten.

SA7: Gewährleistung der Anonymität (Privacy)

Zugriffe auf einen Web Service bzw. das Speichern von persönlichen Daten sollten anonym erfolgen bzw. anonymisiert werden, um die Privatsphäre der Kommunikationspartner zu schützen. Dies bedeutet, dass Angaben über persönliche oder sachliche Verhältnisse nicht ohne unverhältnismäßig großen Aufwand einem bestimmten Kommunikationspartner zugeordnet werden können. Aus diesem Grund fordern wir die Gewährleistung der *Anonymität*. Es ist allerdings anzumerken, dass sich diese Forderung und die Forderung nach Verbindlichkeit bzw. Rückverfolgbarkeit (Traceability) gegenseitig ausschließen. Je nach Anwendungsszenario liegt die Priorität somit mal bei der Anforderung nach Anonymität oder bei der Anforderung nach Rückverfolgbarkeit. Eine Abschwächung der Anonymisierung stellt die Pseudomisierung dar. Bei der Pseudomisierung werden zum Beispiel Pseudonyme

eingesetzt, sodass nur mit Kenntnis bzw. Anwendung einer Zuordnungsvorschrift die Identität eines bestimmten Kommunikationspartners zugeordnet werden kann. Echte Anonymität schließt Verbindlichkeit und Accountability aus; in normalen Anwendungsszenarien wird gewöhnlich auch nur eine der beiden Anforderungen gewünscht sein.

Sicherheitsrichtlinien

Um die Gewährleistung der meisten angestrebten Sicherheitsanforderungen erreichen zu können, sollte eine *Sicherheitsstrategie* bzw. *Sicherheitsrichtlinie* (Security Policy) festgelegt werden, welche eine Menge von technischen und organisatorischen Regeln, Verhaltensrichtlinien, Verantwortlichkeiten und Rollen sowie Maßnahmen vorgibt. Im weiteren Verlauf dieses Dokumentes beschränken wir uns allerdings auf die maschinenverarbeitbaren Anteile einer Sicherheitsrichtlinie. Zur Erstellung, Durchsetzung und Pflege sollte ein Security-Policy-Management-System etabliert werden. Dabei muss die Einhaltung (Compliance) der Sicherheitsrichtlinien gewährleistet werden.

Ein bisher nicht betrachtetes Thema stellt das sichere Speichern von Informationen bzw. Daten dar. In dieses Gebiet fällt zum Beispiel die Datenbanksicherheit (DB-Security). Auf dieses Themengebiet wird allerdings in diesem Dokument nicht weiter eingegangen.

In Tabelle 2 sind die identifizierten Sicherheitsanforderungen noch einmal zusammenfassend dargestellt.

Tabelle 2: Sicherheitsanforderungen

#	Sicherheitsanforderung (SA)	Kurzbeschreibung
SA1	Gewährleistung der <i>Informationsvertraulichkeit</i> (Confidentiality)	Vertraulicher Kommunikationskanal, d.h. kein Abhören von Nachrichten möglich
SA2	Gewährleistung der <i>Datenintegrität</i> (Integrity)	Integrier Kommunikationskanal, d.h. keine unbemerkte Änderung von Nachrichten möglich
SA3	Gewährleistung der <i>Authentizität</i> (Authenticity)	Echtheit und Glaubwürdigkeit der jeweils an einer Kommunikation beteiligten Entitäten
SA4	Gewährleistung der <i>Autorisation</i> (Authorization)	Vergabe von Rechten bzgl. Service- bzw. Ressourcenzugriffen an Benutzer direkt oder über Rollen
SA5	Gewährleistung der <i>Verfügbarkeit</i> (Availability)	Sowohl der durch Service Provider zur Verfügung gestellten Web Services als auch der (WS-)Applikation selbst
SA6	Gewährleistung der <i>Verbindlichkeit</i> (Non-Repudiation)	Betrifft Abrechenbarkeit (Accountability) und Rückverfolgbarkeit (Traceability)
SA7	Gewährleistung der <i>Anonymität</i> (Privacy)	Schutz der Privatsphäre durch Anonymisierung bzw. Pseudomisierung

3.3 Referenzmodell

Um nun die Sicherheitsanforderungen für die einzelnen physikalischen bzw. logischen Verbindungen zu bestimmen, legen wir zunächst Referenzpunkte zwischen den einzelnen Entitäten fest.

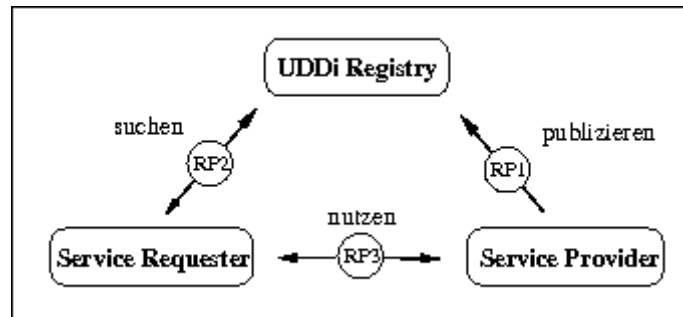


Abbildung 3: Referenzpunkte im WS-Dreieck

Diese werden, wie in Abbildung 3 dargestellt, folgendermaßen definiert:

- **Referenzpunkt 1 (RP1)**
Als RP1 definieren wir die Schnittstelle zwischen dem Service Provider und der UDDI-Registry. Über diese Schnittstelle läuft die Veröffentlichungsphase eines Web Services ab (siehe Schritt 2 bzgl. der oben beschriebenen WS-Entwicklungsphasen, Abbildung 2).
- **Referenzpunkt 2 (RP2)**
Zwischen Service Subscriber / Requester und der UDDI-Registry definieren wir RP2. Über diese Schnittstelle werden die Suchanfrage eines Service Subscribers / Requesters nach einem gewünschten Web Service bei der UDDI-Registry und die Übermittlung der entsprechenden Antwortliste durch diese durchgeführt (siehe Schritte 3 und 4, Abbildung 2).
- **Referenzpunkt 3 (RP3)**
Als RP3 definieren wir die Schnittstelle zwischen Service Subscriber / Requester und Service Provider. Über diese Schnittstelle werden die Service-Anfrage eines Service Subscribers / Requesters nach einem gewünschten Web Service beim Service Provider und entsprechend eine Service-Antwort übermittelt. Dies bedeutet, dass über RP3 die eigentliche Service-Nutzung abläuft (siehe Schritte 6 und 7, Abbildung 2).

3.4 Sicherheitsanforderungen an den Referenzpunkten

Anhand der Referenzpunkte bestimmen wir nun die Sicherheitsanforderungen bzgl. der jeweils beteiligten Kommunikationspartner.

Sicherheitsanforderungen RP1

Bei der Registrierung eines Web Services in der UDDI-Registry muss gewährleistet sein, dass der Kommunikationskanal sowohl vertraulich (SA1) als auch integer (SA2) ist, da sonst Nachrichten abgefangen, manipuliert und/oder wieder eingespielt werden könnten, mit dem Ziel eine Servicebeschreibung bereits bei der Registrierung zu fälschen oder Informationen

über den Web Service unautorisiert zu erlangen. Bei einer für jeden zugänglichen UDDI-Registry wäre SA1 nicht zwangsweise nötig, da die Informationen über den Dienst sowieso öffentlich wären. In beiden Fällen muss allerdings die Authentizität, sowohl des Service Providers als auch der UDDI-Registry, sichergestellt werden (SA3) und anhand einer Zugriffskontrolle überprüft werden, wozu der anfragende Service Provider autorisiert ist, bevor ein neuer Web Service registriert, eine Aktualisierung oder eine Löschung einer WS-Beschreibung durchgeführt werden kann (SA4). Damit ein Web Service jederzeit registriert werden kann, muss die UDDI-Registry immer verfügbar sein, damit einem authentifizierten und autorisierten Service Provider eine Interaktion mit der UDDI-Registry jederzeit möglich ist (SA5).

Sicherheitsanforderungen RP2

Dieser Referenzpunkt bezieht die beiden Entitäten Service Subscriber / Requester und UDDI-Registry ein. Dass der Kommunikationskanal bei einer Suchabfrage vertraulich sein muss (SA1), ist nicht zwangsweise notwendig. An diesem Referenzpunkt muss allerdings gewährleistet sein, dass der Kommunikationskanal integer (SA2) ist, um eine unautorisierte Informationsveränderung, das heißt Manipulation von Suchanfragen und Antwortlisten, zu verhindern (gilt für öffentliche UDDI-Registry). Falls keine öffentliche UDDI-Registry genutzt wird, müssen sich Service Subscriber / Requester und UDDI-Registry für diese beiden Schritte zusätzlich gegenseitig authentifizieren (SA3) und die UDDI-Registry muss überprüfen, ob der Service Subscriber / Requester autorisiert ist, (die gewünschten) Services anzufragen (SA4). Des Weiteren muss eine UDDI-Registry ständig verfügbar sein, damit ein authentifizierter und autorisierter Service Subscribers / Requesters jederzeit eine erneute Suchanfrage stellen kann (SA5). Dies ist besonders wichtig für ein dynamisches WS-Szenario, in welchem Web Services während der Laufzeit einer (WS-)Applikation zum Beispiel durch neu gesuchte Services ergänzt bzw. ersetzt und sofort in Anspruch genommen werden sollen.

Sicherheitsanforderungen RP3

Dieser Referenzpunkt liegt zwischen den beiden Entitäten Service Subscriber / Requester und Service Provider. Der Kommunikationskanal für die Service-Nutzung muss sowohl vertraulich (SA1) als auch integer (SA2) sein, damit eine unautorisierte Informationsgewinnung bzw. Informationsveränderung von Service-Anfragen bzw. Service-Antworten nicht möglich ist. Ebenso müssen sich Service Subscriber / Requester und Service Provider gegenseitig authentifizieren (SA3) und der Service Provider muss überprüfen, ob der Service Subscriber / Requester autorisiert ist, den angefragten Web Service zu nutzen (SA4). Um einem authentifizierten und autorisierten Service Subscribers / Requesters jederzeit die Service-Nutzung zu ermöglichen, muss der angebotene Web Service ständig verfügbar sein (SA5). Für den Service Subscriber / Requester hingegen muss die Nutzung des angefragten Web Services verbindlich sein (SA6). Um die Privatsphäre des Service Subscribers / Requesters zu schützen sollte die Service-Nutzung anonym bzw. zumindest pseudomisiert erfolgen (SA7).

Eine Zusammenfassung der zuvor an den einzelnen Referenzpunkten diskutierten Sicherheitsanforderungen ist in Tabelle 3 dargestellt.

Tabelle 3: Sicherheitsanforderungen an den Referenzpunkten

Referenzpunkt	Sicherheitsanforderungen
RP1	SA1 – SA5
RP2	(SA1), SA2 – SA5
RP3	SA1 – SA7

4 Lösungen zur Umsetzung der Sicherheitsanforderungen

Die in Kapitel 3 bestimmten Sicherheitsanforderungen können fast alle durch geeignete WS-Sicherheitsstandards erfüllt werden. Dieses wird im folgenden Kapitel beschrieben. Bis auf den WS-Security-Standard (siehe Abschnitt 4.1.1) sind die meisten Spezifikationen allerdings noch nicht bei öffentlichen Standardisierungsgremien eingereicht worden. WS* steht im Folgenden für sämtliche für Web Services sicherheitsrelevanten Spezifikationen. Kapitel 1.2 gibt einen Überblick der Standardisierungsaktivitäten der Industrie und definiert, wie die Begriffe Standard und Spezifikation in diesem Dokument gehandhabt werden.

Dieses Kapitel ist wie folgt strukturiert: In Abschnitt 4.1 geben wir zunächst einen kurzen Überblick über die WS-Sicherheitsspezifikationen, bevor wir dann in Abschnitt 4.2 auf die eigentliche Umsetzung der grundlegenden Sicherheitsanforderungen durch die Sicherheitsspezifikationen eingehen. Abschließend stellen wir in Abschnitt 4.3 wieder den Bezug zu den in Kapitel 3.3 definierten Referenzpunkten her.

4.1 Überblick über Web Service Sicherheitsspezifikationen

Im April 2002 veröffentlichten Microsoft und IBM ein Dokument mit dem Titel „Security in a Web Services World“ [WS-Roadmap02]. In diesem Dokument stellten die Firmen ihre Roadmap zur Entwicklung eines Sicherheits-Frameworks für Web Services vor. Dieses Framework beschreibt insbesondere die Kern-Spezifikation WS-Security, sowie die zusätzlichen Erweiterungen WS-Policy, WS-Trust, WS-Privacy, WS-SecureConversation, WS-Federation und WS-Authorization. Des Weiteren beschreibt das Dokument Szenarien, wie diese Spezifikationen zusammen genutzt werden könnten.

Der Entwurf für WS-Security wurde noch im gleichen Jahr beim Standardisierungsgremium OASIS eingereicht, das WS-Security unter Beteiligung weiterer Firmen wie Sun und VeriSign zu einem Standard ausgearbeitet hat. Später wurden Entwürfe für WS-Trust, WS-SecurityPolicy sowie WS-SecureConversations nachgereicht, die aber bis zum Zeitpunkt der Erstellung dieses Dokuments noch nicht von OASIS als Standard verabschiedet wurden. Für WS-Federation wurde ebenfalls ein Entwurf von Microsoft und IBM erstellt, der aber bisher nicht zur Standardisierung bei OASIS eingereicht ist. Inhalte von WS-Privacy und WS-Authorization finden sich in der Microsoft-eigenen Web Services Entwicklungsumgebung, deren Standardisierung aber nicht weiter vorangetrieben wurde.

Zusätzlich zu den von Microsoft und IBM initiierten WS*-Spezifikationen wird bei OASIS noch an weiteren Standards gearbeitet, die Sicherheit von Web Services zum Ziel haben. Hier sind vor allem SAML und XACML zu nennen, die im folgenden Abschnitt zusätzlich zu den WS*-Spezifikationen vorgestellt werden. Dabei gehen wir nur kurz auf deren Aufgaben und Ziele ein. Für weitere Details verweisen wir auf das jeweilig referenzierte Original-Dokument der Spezifikationen.

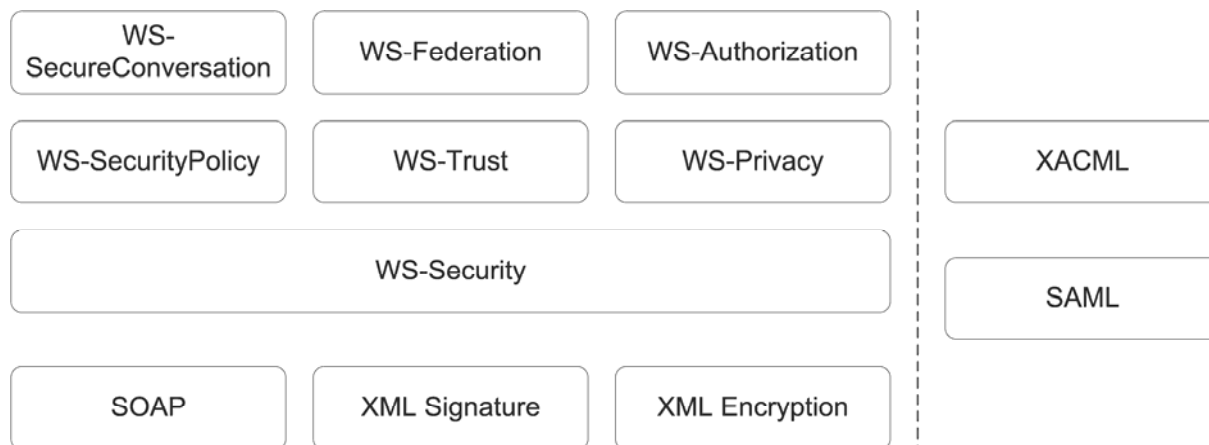


Abbildung 4: Web Service Sicherheitsspezifikationen im Überblick: WS-* Standards und Spezifikationen, grundlegende Standards SOAP, XML Signature und XML Encryption sowie die weiteren OASIS Standards XACML und SAML

4.1.1 XML Encryption

XML Encryption [XMLEnc02] ist ein W3C Standard und beschreibt die XML-Datenstrukturen und den Prozess für die Ver- und Entschlüsselung von XML-Strukturen und ihre Repräsentation in XML. Das Vorgehen kann nicht nur auf XML, sondern auf jede Form von Daten, die in ein XML Dokument eingebettet sind, angewendet werden. Dieser Standard definiert keine eigenen Verschlüsselungsalgorithmen, sondern bedient sich wohlbekannter, standardisierter Verfahren. Die Methoden zum Austausch und zur Verwaltung geheimer Schlüssel ist ebenfalls nicht Gegenstand von XML Encryption.

4.1.2 XML-Signature

Der W3C Standard XML-Signature [XMLSig02] beschreibt, wie XML-Dokumente oder beliebige Daten signiert und die Signaturen in XML repräsentiert werden. Wie XML Encryption (Verweis) definiert XML Signature keine eigenen Signaturalgorithmen, sondern verwendet Standardverfahren. Bevor ein XML-Dokument signiert wird, muss es in ein kanonisches Format gebracht werden, damit für verschiedene Darstellungen desselben Dokumentes dieselbe Signatur erzeugt wird. Die Verfahren zur Vereinheitlichung werden von der W3C in einem anderen Standard definiert.

4.1.3 WS-Security

WS-Security (WSS) [WSS06] stellt den Grundbaustein der hier behandelten WS-Sicherheitsspezifikationen dar. Ziel des WSS-Standards ist es, WS-Applikationen einen sicheren Austausch von SOAP Nachrichten zu ermöglichen. Dazu werden lediglich einige grundlegende Mechanismen definiert, die Vertraulichkeit und einen Integritätsschutz von SOAP-Nachrichten sowie den Transport spezieller Sicherheits-Token ermöglichen. WS-Security ist damit an sich noch kein Sicherheitsprotokoll, sondern liefert lediglich die Elemente zum Entwurf eines an die Anforderungen der Applikation angepassten Sicherheitsprotokolls. Die Mechanismen zum Signieren und Verschlüsseln entstammen ihrerseits den Standards XML Digital Signature [XMLSig02] und XML Encryption [XMLEnc02], die das Signieren und Verschlüsseln von XML-Dokumenten im Allgemeinen spezifizieren, und werden in WS-Security auf SOAP Nachrichten angewendet.

4.1.4 WS-SecurityPolicy

Der ursprüngliche Entwurf von WS-Policy [WS-Pol06] hat sich zu einem allgemeinen Framework entwickelt, mit dem Web Services Bedingungen und Einschränkungen ihrer Nutzung definieren können. Diese Bedingungen (Requirements) und Einschränkungen (Constraints) werden in sogenannten Policy Assertions ausgedrückt. Innerhalb der späteren Spezialisierung WS-SecurityPolicy [WS-SecPol05] werden Policy Assertions definiert, mit denen sicherheitsrelevante Richtlinien für den WSS-Standard und die Spezifikationen WS-Trust und WS-SecureConversations ausgedrückt werden können. Mit Hilfe von WS-SecurityPolicy können Web Services ihrem Kommunikationspartner mitteilen, welche Art der Sicherung der zukünftig auszutauschenden SOAP-Nachrichten sie erwarten. WS-SecurityPolicy ist sehr flexibel gestaltet, um eine Beschreibung der verschiedenen Token-Typen, kryptographischen Algorithmen oder den verwendeten Mechanismen zu ermöglichen.

4.1.5 WS-Trust

WS-Trust [WS-Trust06] ist eine Erweiterung des WS-Security Standards, mit dem Ziel, die Bildung von Vertrauensbeziehungen zwischen Web Service Komponenten zu ermöglichen. Dazu werden zusätzliche Mechanismen definiert, mit denen Authentifikationsdaten inklusive kryptografischer Schlüssel ausgetauscht werden können. Der Austausch kann direkt oder auch über Dritte erfolgen. Mit Hilfe der Funktionen von WS-Trust können Vertrauensbeziehungen neu initiiert, bestehende erneuert und validiert sowie an andere Web Services weiter vermittelt werden.

4.1.6 WS-Privacy

WS-Privacy [WS-Priv] nutzt eine Kombination von WS-Policy, WS-Security und WS-Trust um Vertraulichkeitsbestimmungen für die Nutzung eines Web Services zu definieren. Zum Beispiel kann Clients die Nutzung eines Web Services nur unter der Auflage gestattet werden, die Vertraulichkeitsbestimmungen zu akzeptieren und die ausgetauschten Daten unter Beachtung der Bestimmungen zu verwenden.

4.1.7 WS-SecureConversation

Während sich die Mechanismen von WS-Security zunächst auf die Sicherheit auf Nachrichtenebene beschränken, dient WS-SecureConversation [WS-SC06] dem Etablieren und Teilen eines Sicherheitskontextes für einen längeren Datenaustausch. Aus einem mit Hilfe von WS-Trust etablierten Sicherheitskontext können Sitzungsschlüssel für einzelne Nachrichten abgeleitet werden, um eine effizientere Nachrichtenbearbeitung mit symmetrischen Verschlüsselungsverfahren zu ermöglichen. Die verwendeten Mechanismen sind ähnlich den Mechanismen von SSL auf TCP/IP Ebene. So können zunächst asymmetrische Verschlüsselungstechniken und Signaturverfahren verwendet werden, die den Aufbau eines sicheren Kanals zum Austausch der Sitzungsschlüssel für die weitere Nachrichtenverarbeitung ermöglichen.

4.1.8 WS-Federation

WS-Federation [WS-Fed03] dient der Bildung virtueller Vereinigungen von Web Services in unterschiedlichen Sicherheitsdomänen. In komplexen Web Service Szenarien, in denen sich Services in Domänen mit unterschiedlichen Sicherheitsrichtlinien befinden, müsste ein Client für jede Domäne eine individuelle Authentifizierung durchführen. Um den Clients einen nahtlosen Übergang zwischen den Domänen zu ermöglichen, kann mit WS-Federation ein

gemeinsames Identitätsmanagement etabliert werden, das den Sicherheitsanforderungen der einzelnen Domänen genügt.

4.1.9 WS-Authorization

Mit Hilfe von WS-Authorization [WS-Auth] können Zugriffsbestimmungen für Web Services definiert und verwaltet werden. Durch einfache Zugriffskontrolllisten oder rollenbasierte Zugriffsmodell können Berechtigungen für Funktionen der Web Services bestimmt werden. Die Mechanismen sind flexibel gestaltet in Bezug auf Format und Sprache der Autorisation.

4.1.10 SAML

Die Security Assertion Markup Language (SAML) [SAML05] ist eine Spezifikation des OASIS-Konsortiums, aber nicht Teil der WS*-Spezifikationen aus der ursprünglichen Roadmap von IBM und Microsoft. Dennoch wird in WS-Security der Transport von SAML-Token explizit vorgesehen.

Es handelt sich um einen XML-Standard, um sicherheitsrelevante Authentifizierungs-, Berechtigungs- und Attributinformationen zwischen verteilten WS-Komponenten austauschen zu können. Diese Daten werden in so genannten Assertions ausgedrückt und können mit einem im SAML-Standard definierten Protokoll zwischen verteilten Komponenten abgefragt werden. Wie die Daten in andere Transportprotokolle integriert werden, wird in den Bindings bestimmt. Um in konkreten Szenarien trotz der enormen Flexibilität von SAML eine Kompatibilität zwischen verschiedenen Anwendungen zu ermöglichen, können in Profiles Einschränkungen und konkrete Vorgaben gemacht werden.

Mit Hilfe von SAML lassen sich Funktionen wie SSO, verteilte Transaktionen und Autorisierungsdienste implementieren. Es überschneidet sich somit in der Funktionalität mit den Zielen der WS*-Spezifikationen; zum Beispiel mit WS-Federation und WS-Trust. SAML wird vielfach auch von Projekten aufgegriffen, die konkrete Frameworks für Funktionen wie Single Sign On definieren, darunter die Spezifikationen der Liberty Alliance oder Shibboleth des Internet2-Konsortiums.

4.1.11 XACML

Die eXtensible Access Control Markup Language (XACML) [XACML05] ist eine XML-basierte Sprache, mit der Zugriffsregeln ausgedrückt und abgefragt werden können. Der OASIS Standard liegt in Version 2.0 vor und überschneidet sich in seiner Funktionalität mit WS-Authorization, ist diesem aber durch seine jahrelange Weiterentwicklung als offener Standard überlegen. XACML entstammt ebenso wie SAML nicht den ursprünglich von IBM und Microsoft vorgeschlagenen WS*-Spezifikationen, lässt sich aber mit diesen kombinieren.

XACML erlaubt die Definition von komplexen Policies, mit denen ausgedrückt werden kann, welche Subjekte unter welchen Umständen Zugriff auf bestimmte Ressourcen erhalten dürfen. Es können auch Bedingungen für die Definition neuer Policies ausgedrückt werden, ebenso wie für die Kombination von Policies und die Art der Auswertung dieser. Somit eignet sich XACML als Sprache um Applikationsübergreifend Authentifikations- und Autorisationsdaten in XML auszudrücken.

4.2 Umsetzung der Sicherheitsanforderungen durch WS*-Spezifikationen

In diesem Abschnitt wird überprüft, inwieweit sich die im vorherigen Abschnitt vorgestellten WS*-Spezifikationen zur Gewährleistung der in Kapitel 3.1 bestimmten Sicherheitsanforderungen eignen. Inwieweit zukünftige Implementierungen der noch in Arbeit befindlichen Spezifikationen wirklich die Sicherheitsanforderungen erfüllen werden, kann zum Zeitpunkt der Erstellung dieses Dokumentes nicht beurteilt werden. Die in diesem Abschnitt betrachteten Umsetzungsmöglichkeiten der Sicherheitsanforderungen sind daher lediglich theoretischer Natur, da die WS*-Spezifikationen per se noch keine Sicherheit gewährleisten können. In Kapitel 5 werden dann die bereits implementierten und somit einsatzfähigen Sicherheitseigenschaften bzw. -mechanismen der aktuell am weitesten verbreiteten Frameworks zur Erstellung von (sicheren) Web Services dargestellt.

4.2.1 Gewährleistung der Vertraulichkeit und Integrität, sowie Sicherheit der Kommunikation über mehrere Anfragen hinweg (SA1 & SA2)

Um nun die Eignung der einzelnen WS*-Spezifikationen zur Gewährleistung der Sicherheitsanforderungen zu bestimmen, betrachten wir zunächst wieder den Kommunikationskanal.

Vertraulichkeit und Integrität

Als Sicherheitsanforderungen wurden die Informations-Vertraulichkeit und die Daten-Integrität identifiziert. Um nun die auf dem Kommunikationskanal übermittelten SOAP-Nachrichten vor Mitlesen und Veränderungen zu schützen, kann der OASIS Standard WS-Security eingesetzt werden. Dieser gewährleistet die Informations-Vertraulichkeit und die Daten-Integrität, indem er definiert, wie die XML Security Standards XML Encryption und XML Digital Signature auf SOAP-Nachrichten anzuwenden sind.

Genauer gesagt nutzt der WSS-Standard den XML Encryption Standard zum Verschlüsseln bestimmter Bereiche einer SOAP-Nachricht. Um die Integrität auf der Nachrichtenebene zu gewährleisten, nutzt der WSS-Standard den XML Digital Signature Standard, um SOAP-Nachrichten bzw. Teile dieser, wie zum Beispiel der SOAP-Body, die Benutzer-Credentials und die Zeitstempel, digital signiert zu übertragen.

Hierzu beschreibt der WSS-Standard, wie XML Digital Signature Daten in SOAP-Nachrichten eingebettet werden können. Der XML Digital Signature Standard erklärt, wie bestimmte Teile eines XML-Dokuments digital signiert und wie digitale Signaturen beliebiger Daten in XML ausgedrückt werden können. Welche Bereiche einer SOAP-Nachricht letztendlich geschützt werden sollen, ist allerdings nicht durch den WSS-Standard definiert.

Ende-zu-Ende-Sicherheit

Im Gegensatz zu SSL, das ein übertragungsorientiertes Sicherheitsprotokoll darstellt, welches Punkt-zu-Punkt-Verbindungen schützt, ermöglicht WSS nachrichtenorientierte Sicherheit. Informationen bzw. Daten können aber auch in einem komplexeren WS-Szenario (verteilte SOA) über mehrere Zwischenstationen hinweg verarbeitet werden. Für diesen Fall kann die Ende-zu-Ende-Sicherheit mit Hilfe des WSS-Standards zum Beispiel durch mehrfaches, evtl. überlappendes Verschlüsseln bzw. Signieren realisiert werden, sodass nur die für den jeweiligen Zwischenknoten bestimmten Informationen bzw. Daten einsehbar bzw. änderbar sind. Für die korrekte Entschlüsselung bzw. Überprüfung der jeweiligen Signatur an den Zwischenstationen ist anzumerken, dass es hierbei wichtig ist, die richtige Reihenfolge der Operationen anzuwenden.

Sicherheitskontext

Zum Etablieren und Teilen eines Sicherheitskontextes für einen längeren Datenaustausch während einer Kommunikation über mehrere Anfragen hinweg eignet sich die WS-SecureConversation-Spezifikation als Ergänzung zum WSS-Standard. Der so genannte Sicherheitskontext-Token wird durch WS-SecureConversation als neuer Token-Typ, ergänzend zu den im WSS-Standard bereits definierten Token, eingeführt. Diese neue Token-Variante wird unter anderem mit Hilfe einer Bindung zur WS-Trust-Spezifikation erzielt. Die WS-SecureConversation-Spezifikation beschreibt hierzu drei Möglichkeiten, wie ein Sicherheitskontext zwischen den Kommunikationspartnern unter Einbezug der Mechanismen von WS-Trust etabliert werden kann:

Erzeugung eines Sicherheitskontextes durch

1. einen Sicherheits-Token-Service,
2. einen der Kommunikationspartner oder
3. Aushandeln bzw. Austauschen der benötigten Inhalte wie zum Beispiel ein gemeinsames Geheimnis.

WS-SecureConversation beschreibt auch, wie sich ein Sitzungsschlüssel für einzelne Nachrichten aus dem etablierten Sicherheitskontext ableiten lässt. Symmetrische Sitzungsschlüssel haben den Vorteil, dass symmetrische Verschlüsselungsalgorithmen wesentlich leistungsfähiger sind als Public-Key-Verfahren, welche bei einer reinen Nutzung des WSS-Standard ohne WS-SecureConversation eingesetzt werden, da - anders als zum Beispiel bei SSL – kein Handshake beim Versenden von SOAP-Nachrichten erfolgt.

4.2.2 Gewährleistung der Authentizität, sowie Vertrauensbeziehungen und Vereinigungen (SA3)

Betrachten wir nun die Eignung der WS*-Spezifikationen bzgl. der Gewährleistung der Authentizität. In diesen Bereich fallen auch das Herstellen und Vermitteln von Vertrauensbeziehungen und virtuelle Vereinigungen von Web Services.

Authentizität

Zur Durchführung der Identifikation und Gewährleistung der Authentizität eines Kommunikationspartners kann als Basis ebenfalls der WSS-Standard genutzt werden, da dieser die grundlegenden Mechanismen zum Transport von so genannten Sicherheits-Token bzw. Credentials in SOAP-Nachrichten definiert.

Dies können Token bestehend aus Benutzername (optional: Passwort bzw. Passwort-Hash), Sicherheits-Token in Binärform wie zum Beispiel X.509-Zertifikate oder XML Token wie zum Beispiel SAML-Assertions sein. Anhand dieser Token lässt sich die Identität ermitteln und durch eine Überprüfung der Credentials bzw. des Sicherheits-Tokens die Echtheit und Glaubwürdigkeit gewährleisten; Integrität dieser Informationen vorausgesetzt (s.o.).

Die eigentliche Authentifizierung findet somit lediglich gegenüber dem End-Service bzw. dem Service-Vermittler statt. Der Empfänger einer SOAP-Nachricht führt nur noch einen Überprüfungsmechanismus durch, um die so genannten Behauptungen (Claims) des Senders zu validieren.

Bei den Claims handelt es sich um die oben genannten Varianten der Sicherheits-Token in integrierter oder referenzierter Form. Die Frische der Tokens, das heißt der Schutz vor Replay-Attacken (Details zu Attacken siehe Kap. 2) wird durch den Einsatz von Zeitstempeln

und MessageIDs gewährleistet.

Da der WSS-Standard von Anfang an als Toolkit auf Nachrichtenebene geplant war, hat er auch nur die Aufgabe, die SOAP-Nachrichten sicher an Protokolle höherer Ebenen auszuliefern. Verarbeitet bzw. erzeugt werden somit die Sicherheits-Token nicht direkt auf der Nachrichtenebene, sondern durch Protokolle basierend auf Spezifikationen wie zum Beispiel WS-Trust.

Vertrauensbeziehungen

Durch die Umsetzung von WS-Trust können die in dieser Spezifikation definierten Methoden, um Authentifikationsdaten in Sicherheits-Token zu integrieren, zu erneuern oder zu validieren, für die Authentifizierung herangezogen werden. Da die Spezifikation ebenfalls Methoden definiert, um Vertrauensbeziehungen zu etablieren und zu vermitteln, kann zum Beispiel ein fehlendes Token durch entsprechende Vertrauensmechanismen angefordert und durch den jeweiligen WS-Trust-unterstützenden Service ausgeliefert werden.

Dies kann entweder direkt oder über einen Broker erfolgen. Im Brokerfall könnte ein Trust Proxy anhand einer Sicherheitsrichtlinie den richtigen Tokentyp bestimmen und beim entsprechenden Sicherheits-Token-Aussteller anfordern. Auf Basis der Mechanismen von WS-Security, WS-SecurityPolicy und WS-Trust kann somit ein sicherer, vertrauenswürdiger Token- bzw. Credentia austausch zur Authentifizierung einer bestimmten Identität implementiert werden.

Vereinigung

Auf WS-Trust aufbauend kann WS-Federation implementiert werden, um Mechanismen zur Vermittlung und Vereinigung (Federation) von Vertrauen, Identität und Behauptungen (Claims) nutzen zu können.

WS-Federation kann hierbei eingesetzt werden, um virtuelle Vereinigungen von Web Services in unterschiedlichen Sicherheitsdomänen - das heißt in Domänen mit unterschiedlichen Sicherheitsrichtlinien - zu bilden. Durch WS-Federation kann die Notwendigkeit, in jeder Domäne eine individuelle Authentifizierung durchzuführen, beseitigt werden. Somit kann ein gemeinsames Identitätsmanagement bzw. ein SSO-Konzept durch das automatische Zuordnen von mehreren Identitäten etabliert werden, das die jeweiligen Sicherheitsrichtlinien der einzelnen Domänen erfüllt.

Mit Hilfe der WS-SecureConversation-Spezifikation kann die Notwendigkeit einer erneuten Authentifizierung von SOAP-Nachrichten an möglichen Zwischenstationen eliminiert werden.

4.2.3 Gewährleistung der Autorisation (SA4)

Um nun überprüfen zu können, ob ein Kommunikationspartner autorisiert ist, einen Web Service in Anspruch zu nehmen, kann die WS-Authorization-Spezifikation genutzt werden. WS-Authorization beschreibt hierzu, wie Zugriffsrichtlinien (Access Policy) für Web Services definiert und verwaltet werden können. Unterstützt werden sowohl Zugriffskontrolllisten als auch rollenbasierte Zugriffsmodelle.

4.2.4 Gewährleistung der Verfügbarkeit (SA5)

Die Gewährleistung der Verfügbarkeit kann nicht direkt durch eine WS*-Spezifikation realisiert werden. Hierzu müssen übliche Verfahren wie zum Beispiel Lastverteilung (Load

Balancing), Fallback- bzw. Failover-Systeme, Clusterbildung etc. genutzt werden, um zum Beispiel Überlastungsversuche durch DoS-Attacken abzufangen.

4.2.5 Gewährleistung der Verbindlichkeit (SA6)

Die Gewährleistung der Verbindlichkeit ist zwar kein Ziel des WSS-Standards, kann aber indirekt durch diesen erreicht werden. Der WSS-Standard setzt, wie bereits weiter oben beschrieben, digitale Signaturen ein, um die Integrität von SOAP-Nachrichten zu gewährleisten. Wird nun eine signierte SOAP-Nachricht zum Beispiel durch einen Auditing-Mechanismus in eine Log-Datei geschrieben, so kann diese - zumindest theoretisch - zuverlässig auf den Besitzer des Signaturschlüssels zurückgeführt werden.

4.2.6 Gewährleistung der Anonymität (SA7)

Um die Anonymität bzw. die Privatsphäre zu gewährleisten, kann die WS-Privacy-Spezifikation eingesetzt werden. Wie bereits in Abschnitt 4.1.6 erwähnt wurde, nutzt WS-Privacy eine Kombination aus den Spezifikationen WS-Policy bzw. WS-SecurityPolicy, WS-Security und WS-Trust, um eine so genannte Privacy-Richtlinie (Privacy Policy) festzulegen, zu übermitteln und auf Konformität zu dieser festgelegten Privacy-Richtlinie zu überprüfen.

WS-Privacy beschreibt hierzu ein Model wie eine Privacy-Sprache (Privacy Language) in WS-Policy-Beschreibungen eingebettet werden könnte. Des Weiteren wird beschrieben, wie WS-Security genutzt werden könnte, um Privacy-Behauptungen (Privacy Claims) an eine Nachricht zu binden.

WS-Trust kann dann genutzt werden, um diese Privacy-Behauptungen zu evaluieren; sowohl bzgl. Benutzereinstellungen als auch bzgl. der Organisationspraxis. In der Praxis sieht der Ablauf wie folgt aus:

Die Privacy-Richtlinien müssen vom WS-Client vor der Nutzung des jeweiligen Web Services akzeptiert werden. Die sichere Übertragung erfolgt mit Hilfe des WSS-Standards, indem die Zustimmung in Form eines Claims in einem Sicherheits-Token übermittelt und verifiziert wird. Die mit dem Web Service ausgetauschten Daten dürfen dann nur unter Beachtung dieser im Vorfeld festgelegten Richtlinien verwendet werden.

Wird die WS-Federation-Spezifikation zur Bildung von virtuellen Vereinigungen von Web Services in unterschiedlichen Sicherheitsdomänen eingesetzt, so gibt es die Möglichkeit, einen so genannten Pseudonym Service zu nutzen, um einerseits SSO durch das automatische Zuordnen von mehreren Identitäten zu unterstützen, aber auch andererseits eine Pseudomisierung der Identität einer Person zu ermöglichen. Die Pseudomisierung der Identität erfolgt durch den Einsatz von Aliassen für den Zugriff auf verschiedene Services bzw. Ressourcen in unterschiedlichen Domänen. Der Principal kann optional zwischen einem Aliaswechsel per Service oder per Login wählen.

4.2.7 Sicherheitsrichtlinien

Sicherheitsrichtlinien (Security Policy), wie zum Beispiel welches Authentifizierungsprotokoll bzw. welche Algorithmen-Suite vorausgesetzt wird, um bestimmte kryptografische Operationen durchführen zu können, lassen sich durch WS-SecurityPolicy realisieren. Die Kommunikationspartner sind mit Hilfe dieser Spezifikation in der Lage, anhand einer zu veröffentlichenden Sicherheitsrichtlinie festzulegen, welche Sicherheitsanforderungen zur Nutzung eines Web Services erfüllt sein müssen. Dies bedeutet, welche geeigneten Sicherheitsprotokolle bzw. -mechanismen zur Erzeugung von SOAP-Nachrichten dementsprechend einzusetzen sind. Mittels WS-SecurityPolicy kann somit zum Beispiel

festgelegt werden, dass keine X.509-Zertifikate, sondern ausschließlich Kerberos-Tickets akzeptiert werden.

Wie bereits im Zusammenhang mit dem WSS-Standard erwähnt wurde, kommt es auf die Reihenfolge der Durchführung von Signatur- und Verschlüsselungsprozessen an. Des Weiteren wurde durch den WSS-Standard nicht festgelegt, welche Bereiche einer SOAP-Nachricht signiert und/oder verschlüsselt werden sollten. Dies alles kann mit Hilfe der WS-SecurityPolicy-Spezifikation durch Sicherheitsrichtlinien festgelegt werden.

Primäres Ziel ist es, ein initiales Set an Mustern (Patterns) bzw. Aussagen (Assertions) zu beschreiben, das die gängigen Möglichkeiten zur Sicherung des Kommunikationspfades repräsentiert. Dadurch wird einerseits die Flexibilität bzgl. eingesetzter Token, Kryptografie und Mechanismen gewahrt und andererseits Interoperabilität gewährleistet. Wie die (Sicherheits-)Richtlinien erkannt und an einen Web Service gebunden werden, definiert WS-SecurityPolicy allerdings nicht.

4.2.8 Zusammenfassung

Tabelle 4 zeigt noch einmal zusammenfassend die Zuordnungen der WS*-Spezifikationen zu den jeweilig durch den Standard bzw. die Spezifikation(en) erfüllten Sicherheitsanforderungen basierend auf den Ergebnissen aus den vorherigen Abschnitten 4.2.1 bis 4.2.7. Die Spezifikation WS-SecurityPolicy wird, abgesehen von SA5, jeder Sicherheitsanforderung zugeordnet, da der Einsatz von Sicherheitsrichtlinien zur Gewährleistung dieser Sicherheitsanforderungen beitragen kann, wie bereits in Abschnitt 3.2 erwähnt wurde.

Tabelle 4: Umsetzung der Sicherheitsanforderungen durch WS*-Spezifikationen

#	Sicherheitsanforderung (SA)	Mögliche Umsetzung durch WS*-Spezifikationen
SA1	Gewährleistung der <i>Informationsvertraulichkeit</i> (Confidentiality)	<ul style="list-style-type: none"> • WS-Security • WS-SecureConversation (Sicherheitskontext) • WS-SecurityPolicy (Sicherheitsrichtlinien)
SA2	Gewährleistung der <i>Datenintegrität</i> (Integrity)	<ul style="list-style-type: none"> • WS-Security • WS-SecureConversation (Sicherheitskontext) • WS-SecurityPolicy (Sicherheitsrichtlinien)
SA3	Gewährleistung der <i>Authentizität</i> (Authenticity)	<ul style="list-style-type: none"> • WS-Security (Sicherheits-Token-Transport) • WS-Trust (Vertrauensbeziehungen) • WS-Federation (Vereinigung) • WS-SecureConversation • WS-SecurityPolicy (Sicherheitsrichtlinien)
SA4	Gewährleistung der <i>Autorisation</i> (Authorization)	<ul style="list-style-type: none"> • WS-Authorization • WS-SecurityPolicy (Sicherheitsrichtlinien)

SA5	Gewährleistung der <i>Verfügbarkeit</i> (Availability)	-
SA6	Gewährleistung der <i>Verbindlichkeit</i> (Non-Repudiation)	<ul style="list-style-type: none"> • WS-Security (indirekt, s.o.) • WS-SecurityPolicy (Sicherheitsrichtlinien)
SA7	Gewährleistung der <i>Anonymität</i> (Privacy)	<ul style="list-style-type: none"> • WS-Privacy • WS-Federation (Vereinigung) • WS-SecurityPolicy (Sicherheitsrichtlinien)

4.3 Zuordnung der WS*-Standards zu den Referenzpunkten

Abschließend ordnen wir die möglichen Lösungen zur Erfüllung der grundlegenden Sicherheitsanforderungen den in Kapitel 3.3 definierten Referenzpunkten zu. Basierend auf den Ergebnissen aus Kapitel 3.4 und aus Abschnitt 4.2.7 ergibt sich die in Tabelle 5 dargestellte Zuordnung.

Tabelle 5: Zuordnung der WS*-Spezifikationen zu den Referenzpunkten

Referenzpunkt	Einsatz folgender WS*-Spezifikationen
RP1	<ul style="list-style-type: none"> • WS-Security • WS-SecureConversation • WS-Trust • WS-Federation • WS-Authorization • WS-SecurityPolicy
RP2	<ul style="list-style-type: none"> • WS-Security • WS-SecureConversation • WS-Trust • WS-Federation • WS-Authorization • WS-SecurityPolicy
RP3	<ul style="list-style-type: none"> • WS-Security • WS-SecureConversation • WS-Trust • WS-Federation • WS-Authorization • WS-SecurityPolicy • WS-Privacy

5 Frameworks zur Erstellung von sicheren Web Services

In Kapitel 4 wurden Standards und Spezifikationen zur sicheren Implementierung von Web Services vorgestellt. In Kapitel 5 soll nun gezeigt werden, wie diese Standards und Spezifikationen in der Praxis genutzt werden können. Dazu werden Frameworks zur Implementierung von Web Services vorgestellt und beispielhaft gezeigt, wie in diesen Sicherheitsfunktionen umgesetzt werden können. Näher erläutert werden SAP NetWeaver sowie zwei verbreitete Open Source Frameworks, Projekt GlassFish und Apache Axis.

5.1 Überblick

Die nachfolgende Tabelle bietet zunächst einen Überblick über die Eigenschaften der Open Source Frameworks sowie SAP NetWeaver.

Tabelle 6: Überblick Web Services Frameworks

	SAP NetWeaver	GlassFish	Axis 1.4	Axis 2
Grundfunktionalitäten				
SOAP Version	1.1	1.1, 1.2	1.1,1.2	1.1,1.2
XML-Parser	StAX Impl.	SJSXP (StAX Impl.)	SAX	AXIOM (StAX)
WSDL Version	1.1	1.1	1.1	1.1, 2.0 in Planung
WS*-Standards				
WS-Security	X (siehe Text)	X (WSIT)	X (WSS4J)	X (Rampart)
Token-Profile	Username-Token, X.509, In der Entwicklung: SAML	Username-Token, X.509, SAML, WS-Trust 3rd Party	Username-Token, X.509, SAML	Username-Token, X.509, SAML
WS-SecureConversation	In der Entwicklung	X (WSIT)		In der Entwicklung (Rampart)
WS-Policy		X (WSIT)		X
WS-Trust		X (WSIT)		In der Entwicklung (Rahas)

5.2 SAP NetWeaver

SAP NetWeaver ist eine offene Integrations- und Anwendungsplattform für Service-orientierte Unternehmensarchitekturen. Neben einigen anderen Komponenten sind in NetWeaver zwei verschiedene Applikationsserver integriert, nämlich

- der ABAP Server für die SAP eigene ABAP Programmiersprache
- sowie ein J2EE konformer Java Applikationsserver.

Beide Servertypen unterstützen Web Services. Im Folgenden richten wir den Fokus auf den Java Applikationsserver.

Die Entwicklung von Java-basierten Web Services wird durch das ebenfalls zu NetWeaver gehörende Developer Studio unterstützt. Mittels Softwareassistenten ist es möglich, durch wenige Bedienschritte Web Services aus Java Anwendungen zu generieren, mit zusätzlichen Sicherheitsfunktionen zu ergänzen und in den Applikationsserver zu integrieren.

Im Detail werden für die Web Service Sicherheit folgende Technologien unterstützt:

- Verschlüsselung und Authentifikation auf Transportebene durch die Verwendung von SSL,
- Verschlüsselung der Authentifikationstoken im SOAP Header,
- sowie die Authentifikation auf Nachrichtenebene durch WS-Security.

Es werden die Tokenprofile für Benutzername und Passwort sowie für die Authentifikation mittels X.509 Zertifikaten angeboten. Die Unterstützung von WS-Security ist im gegenwärtigen SAP Web Application Server Java Version 7 noch nicht vollständig ausgearbeitet, so dass noch keine komplette SOAP Nachrichtenverschlüsselung nach WS-Security möglich ist. Es wird lediglich eine Verschlüsselung der im Header befindlichen Authentifikationsdaten angeboten. Nachfolgende Versionen versprechen eine weitergehende Unterstützung für WS-Security und weitere WS*-Spezifikationen.

Im folgenden Abschnitt wird ein kurzer Einblick in die Generierung von Web Services sowie der Auswahl sicherheitsrelevanter Parameter aus dem NetWeaver Developer Studio gegeben. Für eine detaillierte Vorgehensweise seien folgende Artikel aus dem SAP Insider Magazin empfohlen: [SAPIN05] und [SAPIN06]. Weitergehende Informationen befinden sich auf den Seiten des SAP Developer Network [SAPDN].

5.2.1 Web Service Sicherheit in SAP NetWeaver

Beim Anlegen eines neuen Web Services kann eine von drei Standardkonfigurationen gewählt werden. Zur Auswahl stehen (siehe Abbildung 5)

- Simple SOAP,
- Basic Auth SOAP und
- Secure SOAP

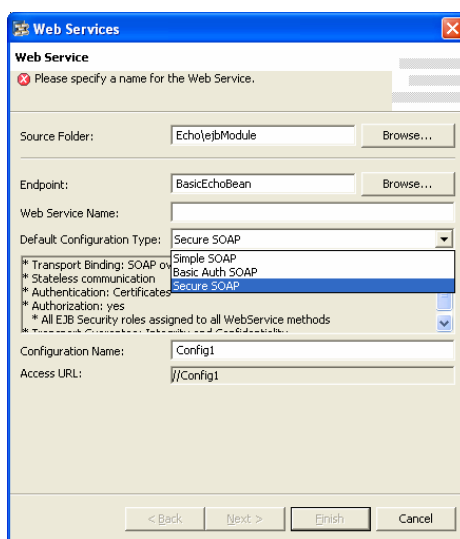


Abbildung 5: NetWeaver Sicherheitseinstellungen

In der Konfiguration “Simple SOAP” werden keinerlei Sicherheitsfunktionen aktiviert. Die Kommunikation mit dem Client findet ohne Authentifizierung und ohne Verschlüsselung mittels SOAP über HTTP statt. Wird „Basic Auth SOAP“ gewählt, wird eine Authentifizierung mit Benutzername und Passwort auf Basis von HTTP durchgeführt. Die Konfiguration „Secure SOAP“ sorgt für eine Verschlüsselung der Daten durch Verwendung von HTTPS (SSL-basiert). Die Authentifizierung findet auf Nachrichtenebene via Zertifikat statt, es wird also ein X.509-Token im Sinne des WS-Security Standards verwendet. Zusätzlich wird die SOAP-Nachricht signiert, um die Integrität zusätzlich auf Nachrichtenebene sicherzustellen. Die nachfolgende Tabelle bietet einen Überblick der auswählbaren Konfigurationen.

Tabelle 7: Sicherheitskonfigurationen NetWeaver

	Simple SOAP	Basic Auth SOAP	Secure SOAP
Transport Binding	SOAP über HTTP	SOAP über HTTP	SOAP über HTTPS
Authentifikation	keine	HTTP Basic	WS-Security X.509-Token
Nachrichtenverschlüsselung	keine	keine	SSL (HTTPS)
Integritätsschutz	keine	keine	WS-Security Signature

Ist der Web Service fertig angelegt, kann die vorgewählte Konfiguration noch angepasst werden. Nach Öffnen der Web Service Konfiguration im NetWeaver Developer Studio bietet der Editor an, die Sicherheitsfunktionen einzeln auszuwählen (siehe Abbildung 6). Auch nach der Installation können die Sicherheitsparameter ohne erneutes Deployment geändert werden.

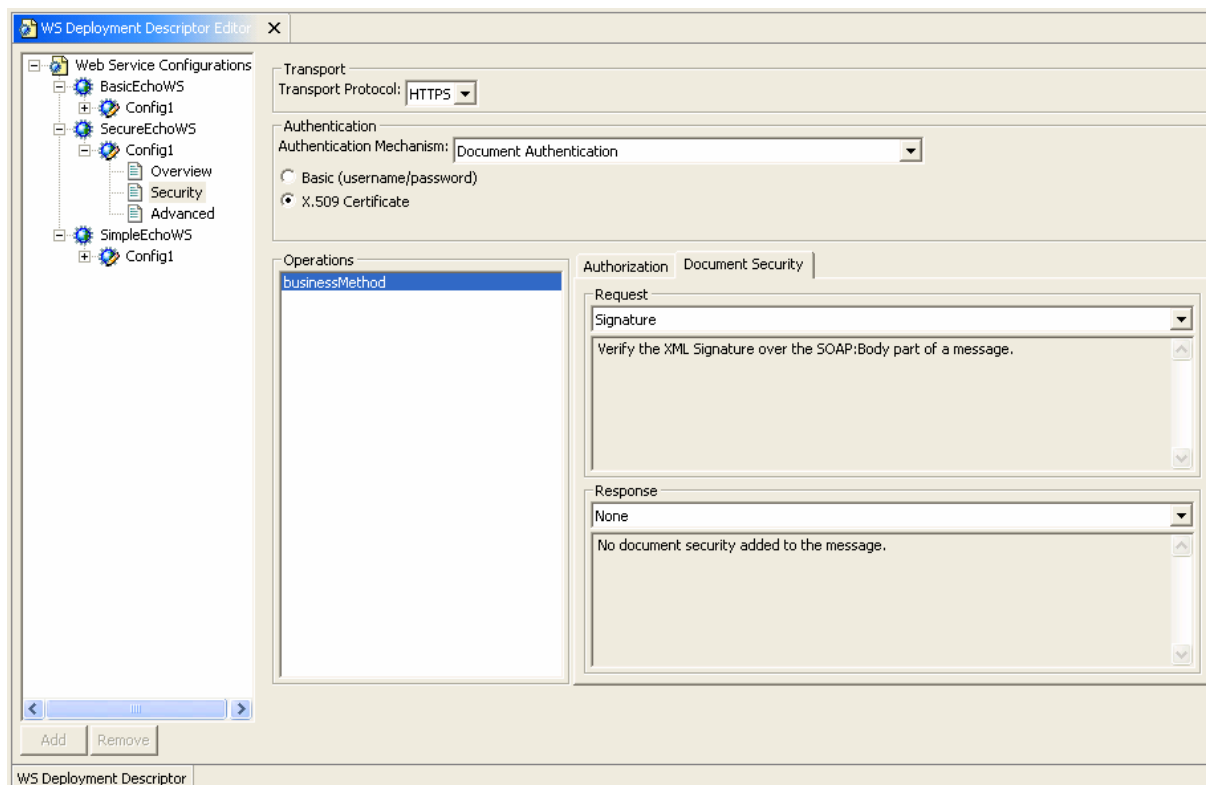


Abbildung 6: NetWeaver Web Service Konfiguration

5.3 Open Source Frameworks

In diesem Abschnitt wird nun beschrieben, wie sich Sicherheitsfunktionen in den beiden Open Source Frameworks GlassFish und Apache Axis umsetzen lassen.

5.3.1 GlassFish

Das Projekt GlassFish ist ein von Sun Microsystems und Oracle initiiertes Open Source Projekt zur Entwicklung eines Java EE5 konformen Application Servers. Obwohl dieses Projekt jung und die Gemeinde der Entwickler noch relativ klein ist, ist es sehr weit fortgeschritten, da es auf der Codebasis des Sun Java System Application Server PE9 aufbaut.

5.3.1.1 Web Service Unterstützung in GlassFish

Es gibt innerhalb von Projekt *GlassFish* [Gla] und Suns java.net Entwickler Gemeinde einige Subprojekte, die die Entwicklung und das Deployment von Web Services unterstützen und die entsprechenden Java WS APIs implementieren. Diese Projekte haben sich genauso wie die verschiedenen Standards und Spezifikationen rund um Web Services über einige Jahre entwickelt und ergänzen bzw. überlappen sich teilweise. Zurzeit setzt eine Konsolidierung dieser Projekte und Implementierungen ein. In diesem Abschnitt soll eine kurze Orientierung gegeben werden, welche Projekte und API Implementierungen für die Entwicklung sicherer Web Services mit Hilfe von GlassFish relevant sind. Es ist zu erwarten, dass die hier aufgeführten Technologien und Projekte auch weiterhin in Bewegung bleiben. Daher ist dieser Abschnitt als eine Momentaufnahme zu verstehen.

Das *Java Web Services Developer Pack (JWSDP)* stellt die Kernbibliotheken für die Web Service und XML Verarbeitung mit Java bereit. Dieses Library Paket existiert zwar noch eigenständig, es ist aber bereits komplett in Projekt GlassFish integriert und wird dort unter einer weniger restriktiven Lizenz vertrieben. JWSDP 2.0 enthält den neuen, integrierten Web Service Stack, der aus den Implementierungen der Standard APIs JAX-WS 2.0, JAXB 2.0 und SAAJ 1.3. Diese Akronyme stehen für die *Java API for XML Web Services*, die *Java Architecture for XML Binding* und die *SOAP with Attachments API for Java*, letztlich also die Programmierschnittstellen, die die SOAP-Kommunikation und die Abbildung von Java nach XML vor dem Benutzer kapselt.

Die Implementierung der JAX-WS API wird in einem gleichnamigen Open Source Projekt [Jax] durchgeführt, das ebenfalls unter dem Dach des GlassFish Projektes lebt. Neben dem Java API Standard unterstützt die Implementierung die Web Service Interoperabilitätsstandards *WS-I Basic Profile 1.1*, *WS-I Attachments Profile 1.0* und *WS-I Simple SOAP Binding Profile 1.0*. Mit dem Projekt *Tango* [Tan] beherbergt JAX-WS ein Subprojekt, das ergänzende Web Service Standards (teilweise) implementiert. Dies sind bisher WS-Policy, WS-Security, WS-ReliableMessaging, WS-SecureConversation und WS-MetadataExchange. Die Implementierung dieser Standards wird auch unter dem Namen *Web Services Interoperability Technologies (WSIT)* geführt.

5.3.1.2 Security in GlassFish

Dieser Abschnitt gibt einen kurzen Überblick über die Sicherheitsfunktionen von GlassFish und Tango. Die Konfiguration, so wie sie hier beschrieben wird, setzt voraus, dass GlassFish in der Version 2 Build 15 oder höher, die freie Entwicklungsumgebung NetBeans 5.5 [Net] von Sun und die WSIT Unterstützung eingesetzt wird. Zum Zeitpunkt der Erstellung dieses Dokumentes war dies JAX-WS 2.0.1 mit WSIT Milestone 2 und das Modul für NetBeans 5.5 und WSIT M2. Die beschriebenen Features können auch ohne die Netbeans IDE genutzt werden.

Netbeans bietet vielfältige Möglichkeiten, um Security-Einstellungen für Web Services und Web Service Clients einzustellen. Abbildung 7 zeigt das Konfigurationsmenü für die Security Einstellungen der Services bzw. Clients. Einige ausgewählte Optionen werden anschließend kurz erläutert.

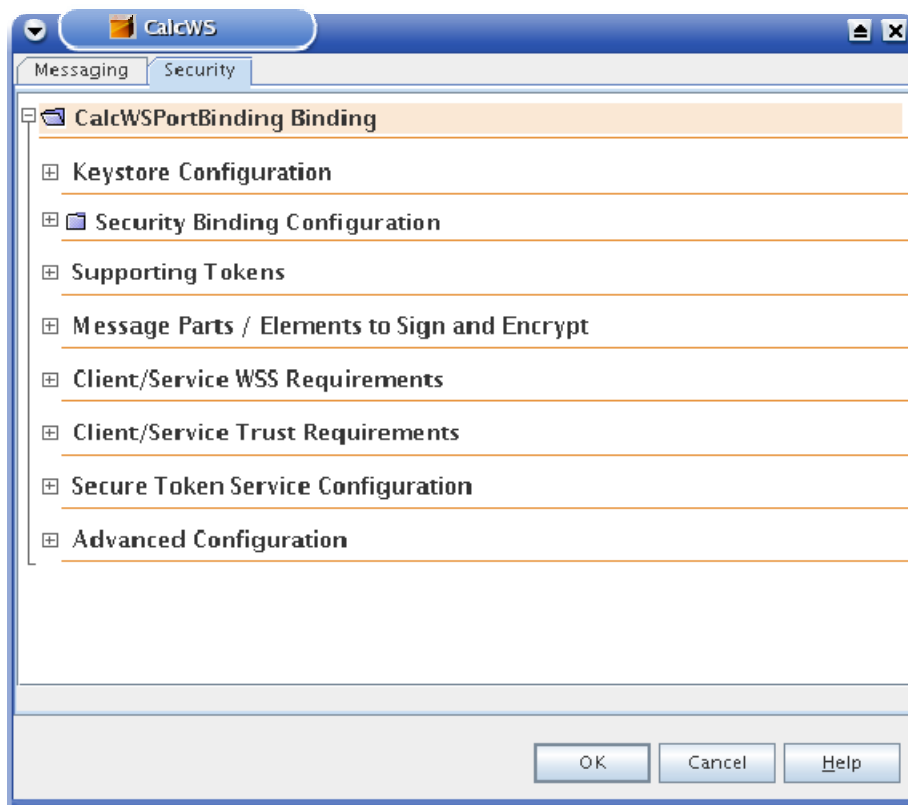


Abbildung 7: NetBeans Konfigurationsmenü für Web Service Security

Keystore Configuration

Hier werden zwei Konfigurationsdateien für die Authentifizierung eingestellt. Die Keystore Datei enthält die Public und Private Keys für die Authentifizierung des Services bzw. Clients. Die Truststore Datei enthält die Zertifikate von bekannten und vertrauenswürdigen Kommunikationspartnern und Certificate Authorities (CAs).

Security Binding Configuration

Unter diesem Menüpunkt können die Security Einstellungen vorgenommen werden, die dieser Dienst von seinen Clients verlangt. Dies ist zum Beispiel die Art der Authentifizierungstoken (X.509, SAML, etc.), die geforderten Security Header Elemente und deren Reihenfolge in der Nachricht, Optionen zur Festlegung der zu verwendenden kryptographischen Algorithmen und einiges mehr.

Message Parts / Elements to Sign and Encrypt

Hier kann der Entwickler sowohl für Web Services als auch Clients einstellen, welche Nachrichtenelemente verschlüsselt und/oder signiert werden sollen.

5.3.2 Apache Axis

Apache Axis [Axis] ist eine Open Source Implementierung des Web Service Standards SOAP. Axis bietet ein Framework um SOAP Nachrichten zu verarbeiten und kann in Server-, Client- oder auch Gateway-Anwendungen integriert werden. Die Implementierung basiert auf Java. Parallel wird aber ebenfalls an einer Implementierung in C++ gearbeitet.

Zusätzlich zur so genannten SOAP-Engine bietet das Projekt noch eine Server-Komponente, die eigenständig oder als Servlet integriert in einen Servlet-Container wie Apache Tomcat das Hosting von Web Services übernehmen kann. Ferner sind in das Projekt einige Hilfsprogramme für die Entwicklung integriert. Darunter befinden sich Programme zur Unterstützung bei Erstellung von WSDL-Beschreibungen, zur Erzeugung von Java-Klassen aus WSDL-Dokumenten sowie ein Tool zum Überwachen der ein- und ausgehenden SOAP-Nachrichten.

Apache Axis enthält keine direkten Möglichkeiten, SOAP-Nachrichten mit Sicherheitsfunktionen zu erweitern. Diese Funktionalität wird in eigenen Projekten erarbeitet. Das Apache Projekt WSS4J bietet eine Implementierung des WS-Security Standards, die in Apache Axis eingebunden werden kann. Dies wird näher im Abschnitt 5.3.2.2 beschrieben.

5.3.2.1 Axis Architektur

Das Projekt Axis der Apache Software Foundation stellt die zweite Generation einer Apache SOAP Engine dar. Das Projekt ist aus der Apache SOAP Implementierung hervorgegangen, welche ihren Ursprung im IBM Projekt SOAP4J hatte. Im Vergleich zu den Vorgängern wurde die Architektur von Axis grundlegend überarbeitet. Axis verwendet die Simple API for XML (SAX) zur Verarbeitung der XML-Dokumente, die deutliche Geschwindigkeitsvorteile im Vergleich zum Document Object Model (DOM) der Vorgänger bietet.

Die DOM-Architektur sah vor, dass die Inhalte von XML zunächst komplett in den Speicher eingelesen werden und dort repräsentiert durch ein Objekt im Sinne der objektorientierten Programmierung bearbeitet werden können. Im Gegensatz zu DOM erlaubt SAX keinen freien Zugriff auf die Inhalte eines XML-Dokuments. Das Dokument wird sequentiell eingelesen und SAX reagiert auf die Inhalte mit zuvor definierten Ereignissen. Entsprechend erfolgt die Bearbeitung von Dokumenten in Axis durch so genannte Handler, die auf Inhalte der eingehenden Nachrichten reagieren und in Filterketten angeordnet werden. Die Filterketten lassen sich über Konfigurationsdateien festlegen. Die Nachrichtenverarbeitung kann durch eigene und vordefinierte Handler gestaltet werden. Typische Beispiele für Handler sind Logging oder auch das Überprüfen und Einfügen von Sicherheitsinformationen.

Um Web Services anzubieten wird ein Server benötigt, der eingehende Anfragen bearbeitet. Axis bietet die Möglichkeit als eigenständiger Server zu laufen, es ist aber auch möglich, die Dienste von Axis als Servlet in einen Servlet-Container unterzubringen. Als Servlet-Container bietet sich Apache Tomcat an. Die Installation in Tomcat ist entsprechend der Anleitung einfach durch Kopieren der Axis Dateien in den Tomcat Pfad für Web-Applikationen möglich. Über Tomcat sind nach kurzer Konfiguration drei Servlets von Axis erreichbar:

- Das eigentliche Axis Servlet, auch SOAP Router genannt, mit einigen vorkonfigurierten Web Services,
- ein AdminServlet zur weiteren Konfiguration und
- ein SOAP Monitor Servlet, das zum Überwachen von SOAP Nachrichten verwendet werden kann.

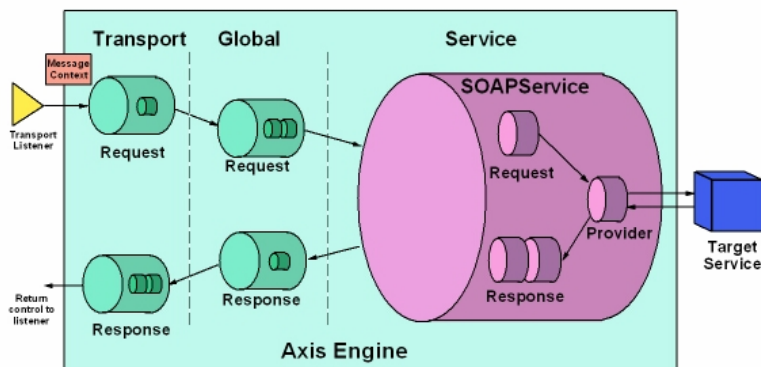


Abbildung 8: Axis Server Architektur [Axis]

Um weitere Web Services hinzuzufügen, müssen diese im SOAP Router installiert werden. Dieser Vorgang wird als Deployment bezeichnet. Ein neuer Webservice besteht im Wesentlichen aus den kompilierten Java Dateien des Services und einem Deployment Descriptor, in dem Eigenschaften des Web Service definiert werden. Das Deployment erfolgt über ein Kommandozeilen-Programm. Zum einfachen Einstieg eignen sich die mit Axis gelieferten Beispiele. Die WSDL-Beschreibung des Web Services wird von Axis automatisch generiert und kann über einen Browser abgerufen werden.

5.3.2.2 WSS4J

WSS4J [WSS4J] ist ein eigenständiges Web Services Projekt von Apache. Es handelt sich um eine Implementierung des WS-Security Standards der OASIS Web Services Security TC (siehe Kapitel 4.1.1). Im Kern ist WSS4J eine Java-Bibliothek, mit der SOAP-Nachrichten nach dem WS-Security Standard signiert, verifiziert, verschlüsselt und entschlüsselt werden können. Zusätzlich werden die Token Profile für Username Token, SAML Token und X.509 Token unterstützt.

Die Erweiterung von Axis um die Sicherheitsfunktionen von WSS4J ist einfach möglich. Die Klassenbibliothek von WSS4J muss zusammen mit anderen erforderlichen Bibliotheken, die aber in einem Paket von Apache erhältlich sind, in das Bibliotheksverzeichnis von Axis abgelegt werden. Anschließend können Web Services über den Deployment Descriptor so konfiguriert werden, dass bei ein- und ausgehende Nachrichten die Handler von WSS4J sicherheitsrelevante Modifikationen an den Nachrichten durchführen. Ein Web Service Client muss bei der Programmierung so angepasst werden, dass er die erwarteten Sicherheitsinformationen anfügt beziehungsweise eine Verschlüsselung von Inhalten vornimmt. Auch hierzu bietet die WSS4J-API Funktionen.

Das folgende Beispiel fasst kurz zusammen, wie in einen bestehenden Web Service die Abfrage eines Username Token zur Benutzerauthentifikation eingeführt werden kann. Anhand von diesem Beispiel soll gezeigt werden, wie WSS4j in Axis integriert wird und wie Entwickler die Funktionalität von WSS4J nutzen können. Eine detaillierte Dokumentation der weiteren Funktionen von WSS4J lässt sich auf den Internetseiten des Projekts sowie innerhalb des Installationspakets finden.

1. Im ersten Schritt muss zunächst WSS4J in Axis integriert werden. Ausgehend von der WSS4J Version 1.5 müssen von der Apache WSS4J Projektseite die Dateien wss4j-bin-1.5.0.zip und wss4j-otherjars-1.5.0.zip herunter geladen werden. Die Datei ZIP-Datei ww4j-bin-1.5.0.zip enthält die WSS4J-Bibliothek ww4j-bin-1.5.0.jar, die zusammen mit den in der Datei wss4j-otherjars-1.5.0.zip enthaltenen .jar Dateien in das „axis/WEB-INF/lib“-verzeichnis der Axis-Installation kopiert werden müssen.
2. Im nächsten Schritt muss der Deployment-Descriptor des Web Service erweitert werden. Unterhalb des Service-Elements wird ein neuer Handler für den RequestFlow definiert. Im folgendem Ausschnitt eines Descriptors wurden die neuen Elemente fett markiert:

```
<service name="BeispielWS" provider="java:RPC" style="document"
use="literal">
  <requestFlow>
    <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
      <parameter name="passwordCallbackClass" value="PWCallback"/>
      <parameter name="action" value="UsernameToken"/>
    </handler>
  </requestFlow>
  <parameter name="className" value="test.BeiispielWS"/>
  <parameter name="allowedMethods" value="ping"/>
  <parameter name="scope" value="application"/>
</service>
```

Über den Handler `WSDoAllReceiver` kann auf die Funktionalität von WSS4J für alle eingehenden SOAP-Nachrichten zugegriffen werden. Die genau durchzuführenden Aktionen werden durch die Parameter definiert. In diesem Beispiel wird zunächst ein `CallbackHandler` `PWCallback` definiert, der die Überprüfung der Passworte vornimmt. Näheres dazu findet sich unter Punkt 3. Der Parameter `action` wird eine Aktion definiert, in diesem Beispiel die Auswertung des Username Token.

3. Die Erwartung eines Username Token macht nur dann Sinn, wenn die im Username Token übergebenen Benutzernamen und Passwörter überprüft werden. Diese Funktion übernimmt ein zu implementierender `CallbackHandler`, der dem Web Service hinzugefügt werden muss. Die verwendete Callback Technik orientiert sich am *Java Authentication and Authorization Service (JAAS)*. Anhand des gegebenen Benutzernamens, dem so genannten Identifier, muss das zugehörige Passwort zur Verifizierung der Token Inhalte übergeben werden. Der folgende Code zeigt eine beispielhafte Implementierung:

```
public class PWSecurity implements CallbackHandler {
    public void handle(Callback[] callbacks) {
        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof WSPasswordCallback) {
                WSPasswordCallback pc = (WSPasswordCallback)callbacks[i];
                String identifier = pc.getIdentifier();
                byte[] key = lookForPassword(identifier);
                pc.setPassword(key);
            }
        }
    }

    private byte[] lookForPassword(String id) {
        ...
    }
}
```

5.3.2.3 Axis 2

Mit Axis 2 [Axis2] ist ein Nachfolgeprojekt von Axis in der Entstehung, das eine weiterentwickelte Architektur und weitere Geschwindigkeitsvorteile verspricht. Durch die Ablösung der SAX-API durch Apache Axiom (AXIS Object Model) kann die Verarbeitungsgeschwindigkeit von SOAP-Nachrichten gesteigert werden, gleichzeitig verringert sich der Speicherbedarf. Die Architektur wurde durch die Einführung von Modulen und Phasen erweitert, die aufbauend von den Handlern aus Axis die Nachrichtenverarbeitung in logische Abschnitte unterteilen. Die Module erlauben es einfacher, Funktionalitäten nachträglich einzuführen. So soll sich in Zukunft eine Unterstützung der verschiedenen WS*-Standards nachinstallieren lassen. Im Bereich Security ist bereits eine Umsetzung von WSS4J als Modul für Axis 2 verfügbar. Der Projektname lautet Rampart.

Zum Zeitpunkt der Erstellung dieses Dokuments war noch keine offiziell als „stabil“ bezeichnete Version von Axis 2 erhältlich. Deshalb wurde hier auf eine detailliertere Betrachtung von Axis 2 verzichtet. Zukünftig verspricht Axis 2 eine umfangreichere Unterstützung der WS*-Standards und wird mit hoher Wahrscheinlichkeit Axis im produktiven Einsatz ablösen.

5.4 Infrastrukturelle Maßnahmen

In den vorangegangenen Abschnitten wurden Methoden gezeigt, wie Web Services mit Hilfe von Frameworks sicher gestaltet werden können. Um Web Services sicher betreiben zu können, ist aber zusätzlich zur sicheren Implementierung der Web Services auch eine Absicherung der Infrastruktur notwendig. Dazu müssen generelle Maßnahmen getroffen werden, wie die Absicherung vor Angriffen aus dem Internet durch Firewalls, Intrusion Prevention Systemen oder andere Maßnahmen auf Netzwerkebene. Beim Betrieb der Web Service Server aus den vorgestellten Frameworks sind zusätzlich einige spezielle Maßnahmen zu beachten, die nachfolgend erläutert werden sollen.

5.4.1 Sicherer Betrieb des NetWeaver Application Servers Java

Ausführliche Informationen zum sicheren Betrieb des NetWeaver Application Servers finden sich in der Online-Dokumentation von SAP. Diese ist frei zugänglich und zum überwiegenden Teil in deutscher Sprache abrufbar. An dieser Stelle sei besonders der Sicherheitsleitfaden zum Betrieb des Application Servers empfohlen, der Empfehlungen und wichtige Hinweise zu Sicherheitsaspekten gibt. Er ist von der Startseite der SAP NetWeaver Dokumentation [SAPNWH] über die Links “Deutsch” und “Sicherheitsleitfaden” zu erreichen.

5.4.2 Sicherer Betrieb eines GlassFish Application Servers

In diesem Abschnitt sollen kurz einige der wesentlichen Konfigurationsschritte beispielhaft für den sicheren Betrieb eines GlassFish Application Servers beschrieben werden.

Admin Webkonsole

In einer Produktivumgebung sollte die Admin Webkonsole Anwendung entweder entfernt werden oder nur aus dem Intranet des Unternehmens erreichbar sein.

Admin Benutzeraccount

Der Admin Benutzeraccount ist in der Standardinstallation immer die Benutzer-ID admin mit dem Passwort adminadmin. Dies muss als erstes auf ein sicheres Passwort geändert werden. Konfigurationsdateien, die das Admin Passwort enthalten müssen, dürfen nur von dem Adminbenutzer, der den Server betreibt lesbar sein. Hierbei sollte auch beachtet werden, dass das Adminpasswort unter Umständen an unerwarteten Stellen steht. So kann es während der Entwicklung von Web Services in der Server-Konfiguration der NetBeans Umgebung angegeben worden sein.

Beispielapplikationen

Beispielapplikationen sollten generell aus der Produktivumgebung entfernt werden. Diese verbrauchen nicht nur unnötig Ressourcen, sondern können einem potentiellen Angreifer Informationen über den installierten Server-Typ, dessen Versionsnummer und andere Dinge liefern.

5.4.3 Sicherer Betrieb von Axis

Bevor Axis in Betrieb genommen wird, sollten zunächst alle nicht mehr benötigten vorinstallierten Web Services deaktiviert werden. Dazu zählt insbesondere der AdminService, der in der Regel, sobald alle benötigten Web Services installiert und konfiguriert sind, nicht mehr benötigt wird. Ebenso kann der ListingService zur Anzeige der installierten Services sowie die HappyAxis Seite zur Anzeige von Installationsinformationen deaktiviert werden. Die benannten Web Services sind nach der Standardinstallation an für einen potentiellen Angreifer bekannten Orten zugreifbar und können so einfach zur Informationssammlung ausgenutzt werden. Sollten die Funktionen weiterhin unbedingt benötigt werden, sollten sie zumindest umbenannt und die Zugriffsschnittstellen verändert werden. Das Verstecken und Verändern der Schnittstellen bietet aber generell keinen ausreichenden Schutz.

In den Axis Standardeinstellungen werden im Falle eines Fehlers keine Fehlermeldungen an einen Web Service Client gesendet. Sollte diese Funktion während der Testphase aktiviert worden sein, muss diese für den Betrieb dringend wieder deaktiviert werden. Ein Angreifer könnte diese Informationen ausnutzen, um Schwachstellen zu finden.

Eine weitere Funktion, die in einigen Fällen im Betrieb deaktiviert werden kann, ist die automatische Generierung von WSDL Beschreibungen für installierte Web Services. In Anwendungsszenarien, in denen der Web Service lediglich einer begrenzten Anzahl von Clients zur Verfügung stehen soll, kann diesen die WSDL-Beschreibung über andere Kommunikationswege wie zum Beispiel E-Mail bekannt gemacht werden und muss nicht öffentlich zugänglich gemacht werden.

Um Axis im Betrieb überwachen zu können, sollte das Schreiben von Log-Dateien auf die eigenen Bedürfnisse angepasst und regelmäßig ausgewertet werden. Eine Archivierung der Dateien ermöglicht auch nach längerer Zeit die Suche nach Fehlern oder versuchten Angriffen. Das zu ausführliche Schreiben von Log-Dateien kann aber auch wiederum für einen DoS-Angriff ausgenutzt werden.

Generell sollte Axis und der ausführende Server immer mit minimalen Systemrechten laufen. So kann im Falle eines erfolgreichen Angriffs der Schaden minimal gehalten werden, da eventuell in Axis ausgeführter schadhafter Code keinen Zugriff auf andere Anwendungen oder Systemkomponenten hat. Entsprechend sollten das Dateisystem und die Rechte der Java-Umgebung so konfiguriert werden, dass nur die notwendigen Dateizugriffe erlaubt sind.

Auch wenn Axis bereits seit Jahren weiterentwickelt wird, kann nicht ausgeschlossen werden, dass Schwachstellen durch Fehler in der Implementierung bestehen. Um im Fall eines neu entdeckten Fehlers schnell reagieren zu können, sollten sich Administratoren über die zur Verfügung stehenden Mailinglisten regelmäßig über sicherheitsrelevante Entwicklungen informieren lassen.

6 Fazit

Web Services sind eine wichtige Technologie für die Vernetzung von Unternehmens-Anwendungen. Sie ermöglichen eine neue Form der Automatisierung und Flexibilisierung von Geschäftsprozessen über Unternehmens- und Technologiegrenzen hinweg. Die sichere Umsetzung von Web Services ist ein entscheidendes Kriterium für deren erfolgreichen Einsatz. Wie in dem Dokument vorgestellt gibt es bereits eine Vielzahl von Frameworks, die die Entwicklung von sicheren Web Services unterstützen.

Die Konzeption von sicheren Web Services kann dennoch im konkreten Fall zu einer Herausforderung werden.

Die Vielzahl von Spezifikationen und Standards, deren Standardisierungsprozesse oft nicht abgeschlossen sind und einer permanenten Weiterentwicklung unterliegen verlangen eine Dynamik in der Entwicklung und Anpassung des jeweiligen Frameworks. Dabei zeigt sich, dass die beschriebenen Open Source Frameworks schnell auf die Arbeit der Standardisierungsgremien reagieren und eine umfangreiche Unterstützung der Web Service Standards bieten. Allerdings hält die Dokumentation und die Erstellung von Entwicklerwerkzeugen nicht immer mit der Umsetzung der Standards Schritt.

Die im SAP NetWeaver Framework verfügbaren Funktionen zur sicheren Implementierung von Web Services sind demgegenüber wesentlich leichter in die Anwendungen zu integrieren. Die in Kürze erscheinende Nachfolgeversion bietet zudem ebenfalls eine weitergehende Unterstützung der WS-* Standards.

Referenzen

- [Alo03] G. Alonso, F. Casati, H. Kuno, V. Machiraju; Web Services – Concepts, Architectures and Applications, Springer Verlag 2004.
- [Axis] Apache Web Services Project – Axis, Projekt-Homepage, <http://ws.apache.org/axis>.
- [Axis2] Apache Axis2/Java Version 1.1, Projekt-Homepage, <http://ws.apache.org/axis2>.
- [Bhar05] Karthikeyan Bhargavan, Cedric Fournet, Andrew D. Gordon, Greg O’Shea: "An Advisor for Web Services Security Policies", SWS '05, ACM Press, 2005.
- [BSI05] Bundesamt für Sicherheit in der Informationstechnik "IT-Grundschutzhandbuch", 2005
<http://www.bsi.de/gshb/index.htm>.
- [GJ06] Rix Groenboom, Rami Jaamour: "Securing Web Services", OWASP Europe Conference, 2006.
- [Gla] Projekt GlassFish, <https://glassfish.dev.java.net/>.
- [Har03] B. Hartman et al.: Mastering Web Services Security. 1. Auflage. Indianapolis: Wiley 2003.
- [Jax] JAX-WS Projekt, <https://jax-ws.dev.java.net/>.
- [Net] NetBeans IDE, <http://www.netbeans.org/index.html>.
- [OWA1] The Open Web Application Security Project (OWASP), <http://www.owasp.org>.
- [OWA2] OWASP Top Ten Project, http://www.owasp.org/index.php/OWASP_Top_Ten_Project.
- [PC04] Peikari, Cyrus, Chuvakin, Anton: "Kenne deinen Feind", 2004, O’Reilly.
- [RRS06] Mohammad Ashiqur Rahaman, Maarten Rits and Andreas Schaad: "An Inline Approach for Secure SOAP Requests and Early Validation", OWASP Europe Conference, 2006.
- [SAML05] Security Assertion Markup Language (SAML), V2.0 OASIS Standard specification set, March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip>
OASIS Security Services (SAML) TC: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.
- [SAPDN] SAP Developer Network <http://www.sdn.sap.com>.
- [SAPNWH] Online-Dokumentation zu SAP NetWeaver <http://help.sap.com/content/documentation/netweaver/index.htm>
- [SAPIN05] Gerlinde Zibulski, Peter McNulty: "A Code-Free, Wizard-Driven Approach to Securing Your Web Services“, SAPinsider, 2005.
- [SAPIN06] Gerlinde Zibulski, Peter McNulty: "Securely Consume Web Services – With No Coding“, SAPinsider, 2006.
- [SEC06] Secologic, Java Web Application Security - Best Practice Guide, 2006.
- [SOAP03] SOAP Version 1.2, W3C Recommendation, Juni 2003, <http://www.w3.org/TR/>.

- [Tan] Projekt Tango,
<https://wsit.dev.java.net/>.
- [TH05] van Tilborg, Henk C.A. (Ed.): "Encyclopedia of Cryptography and Security", 2005, Springer.
- [UDDI05] Universal Description, Discovery, and Integration (UDDI), Version 3.0.2, OASIS Standard, Februar 2005,
<http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3>
 OASIS UDDI Specification TC:
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec.
- [WAS] The Web Application Security Consortium (WASC),
<http://www.webappsec.org/>.
- [WS-Auth] WS-Authorization, siehe [WS-Roadmap02].
- [WS-Fed03] WS-Federation, Web Services Federation Language, 1.0, Draft, Juli 2003,
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-federation.asp>.
- [WS-Pol06] Web Services Policy 1.5 – Framework (WS-Policy), W3C,
<http://www.w3.org/TR/ws-policy/>.
- [WS-PolAss] WS-PolicyAssertionWeb Services Policy Assertions Language (WS-PolicyAssertions), this specification is deprecated, Version 1.1, May 2003,
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policyassertions.asp>.
- [WS-Priv] WS-Privacy, siehe [WS-Roadmap02].
- [WS-Roadmap02] Security in a Web Services World: A Proposed Architecture and Roadmap, A joint security whitepaper from IBM Corporation and Microsoft Corporation. April 2002, Version 1.0,
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/securitywhitepaper.asp>.
- [WS-SC06] WS-SecureConversation 1.3, Committee Draft 01, OASIS, September 2006,
<http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-spec-cd-01.pdf>
 OASIS Web Services Secure Exchange (WS-SX) TC:
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-sx.
- [WS-SecPol05] Web Services Security Policy Language (WS-SecurityPolicy). Version 1.1. July 2005,
<http://xml.coverpages.org/WS-SecurityPolicyV11-200507.pdf>.
- [WS-Trust06] WS-Trust 1.3, Committee Draft 01, OASIS, September 2006,
<http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-spec-cd-01.pdf>
 OASIS Web Services Secure Exchange (WS-SX) TC:
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-sx.
- [WSDL06] Web Services Description Language (WSDL) Version 2.0, W3C Draft, März 2006,
<http://www.w3.org/TR/>.

- [WSS06] Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), OASIS Standard Specification, February 2006,
<http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
OASIS Web Services Security (WSS) TC:
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.
- [WSS4J] Apache Web Services Project – WSS4J, Projekt-Homepage,
<http://ws.apache.org/wss4j>.
- [XACML05] eXtensible Access Control Markup Language (XACML) Version 2.0, OASIS Standard, Feb 2005,
http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
OASIS eXtensible Access Control Markup Language (XACML) TC:
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
- [XML06] Extensible Markup Language (XML) 1.1 (Second Edition), W3C Recommendation, August 2006,
<http://www.w3.org/TR/2006/REC-xml11-20060816/>.
- [XMLSig02] World Wide Web Consortium (W3C), „XML-Signature Syntax and Processing“,
<http://www.w3.org/TR/xmlsig-core/>.
- [XMLEnc02] World Wide Web Consortium (W3C), „XML Encryption Syntax and Processing“,
<http://www.w3.org/TR/xmlenc-core/>.

Abkürzungsverzeichnis

API	Application Programming Interface
AXIOM	AXis Object Model
BDSG	Bundesdatenschutzgesetz
CORBA	Common Object Request Broker Architecture
DB	Datenbank
DIN	Deutsches Institut für Normung
DOM	Document Object Model
DoS	Denial of Service
DTD	Document Type Definition
ETSI	European Telecommunications Standards Institute
HTTP	Hypertext Transfer Protocol
ID	Identifikationsnummer
IDL	Interface Description Languages
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Informationstechnik / -technologie
JAAS	Java Authentication and Authorization Service
MS	Microsoft
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
RPC	Remote Procedure Call
RSA	Rivest Shamir Adleman
SA	Sicherheitsanforderung
SAML	Security Assertion Markup Language
SAX	Simple API for XML
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Sockets Layer
SSO	Single Sign-On
SW	Software
TC	Technical Committee
TCP	Transmission Control Protocol
TDDSG	Teledienstschutzgesetz
TLS	Transport Layer Security
TTP	Trusted Third Party
UDDI	Universal Description, Discovery, and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
W3C	World Wide Web Consortium
WS	Web Service(s)
WSDL	Web Services Description Language
WSS	Web Services Security, WS-Security



XML Extensible Markup Language
XMLDSig XML Digital Signature
XMLEnc XML Encryption
XPath XML Path Language