Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Leroy Metin Yaylacioglu

Business Value einer Web Service Firewall

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Leroy Metin Yaylacioglu

Business Value einer Web Service Firewall

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft
Zweitgutachter : Prof. Dr.Ing. Martin Hübner

Abgegeben am 4. Juni 2008

Leroy Metin Yaylacioglu

Thema der Bachelorarbeit

Business Value einer Web Service Firewall

Stichworte

Web-Service-Firewall, XML-Firewall, XML Bedrohungen, XDoS, Sicherheit, Web-Service, SOAP

Kurzzusammenfassung

Web Services finden, innerhalb serviceorientierter Architekturen oder als allein stehende Dienste, zunehmend Verwendung in der Unternehmensgrenzen übergreifenden Datenkommunikation. Das dabei vorherrschende Datenaustauschprotokoll ist SOAP über HTTP. Gewöhnliche Firewalls auf Port- und Packetebene können Web Services, die HTTP zum Datenaustausch verwenden, nicht von regulärem HTTP unterscheiden - sie passieren die Firewall ungehindert in ein- und ausgehender Richtung. Eine Web-Service-Firewall schließt diese Lücke, indem sie die Nachrichten inhaltlich auswertet und sie gegen ein Regelwerk validiert. Ziel dieser Arbeit ist es den technologischen Mehrwert einer Web-Service-Firewall zu erarbeiten und daraus den geschäftlichen Mehrwert abzuleiten. Dazu werden zielgerichtet manipulierte Nachrichten an Web-Services eines Applikationsservers gerichtet und die Folgen ausgewertet.

Leroy Metin Yaylacioglu

Title of the paper

Business value of a Web Service firewall

Keywords

Web Service Firewall, XML firewall, XML threats, XDoS, security, Web Service, SOAP

Abstract

Web Services are increasingly used for cross-corporate data communication within Service-oriented architectures or as standalone services. Such communication bases on the widely spread data exchange protocoll SOAP over HTTP. Common firewalls based on port and packetlevel filtering do not differ between usual web traffic and Web Service traffic. The traffic passes the firewall unchecked in the incoming and outgoing direction. A Web Service Firewall closes this gap by scanning and validating the content of the messages against contraints and policies while passing through. This paper aims to assses the technical value of a Web Service firewall and derive the business value. For this particular reason specifically compromised messages are beeing send to Web Services of an application server and evaluating the impact subsequently.

Danksagung

Die vorliegende Bachelorarbeit entstand während meines Studiums an der Hochschule für Angewandte Wissenschaften Hamburg im Rahmen meiner Bachloranten Tätigkeit bei der IBM Deutschland GmbH.

Vielen Dank an Dr. Stephan Feil, der mich während der Durchführung der Bachelorarbeit bei der IBM Deutschland GmbH unterstützt und betreut hat.

Ein Großer Dank geht an Prof.Dr. Olaf Zukunft für die umfassende Betreuung vor und während der Ausarbeitung der Bachelorarbeit.

Ich danke meinen Freunden Emmanuell Perrakis und Ilker Gülgün die immer an mich geglaubt haben, mir in schweren Zeiten zur Seite standen und mir das Studium erst ermöglicht haben.

Großer Dank an Dennis Dedaj, der mir ermöglicht hat mein Studium rasch zu absolvieren.

Ein besonderer Dank geht an meine Eltern und meine beiden Geschwister die mir beigestanden haben in den letzen Jahren... sowie meinen beiden Kindern die mir während des gesamten Studiums immer ein reger Ansporn waren =)

Leroy Metin Yaylacioglu, Juni 2008

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

1 Einführung

1.1 Einleitung

Die meisten IT-Systeme entstanden als Insellösungen für einzelne meist technische Probleme oder Herausforderungen. Über die Zeit wurden immer mehr Inseln miteinander verbunden und es entstanden in den Unternehmen bisweilen sehr komplexe Umgebungen. Die Integration einer solchen Anwendungslandschaft stellt einen hohen Anspruch an die zugrundeliegende Integrations-Architektur. Im Rahmen der Integration von innerbetrieblichen[1] sowie Unternehmensgrenzen übergreifenden[2] Anwendungen sind serviceorientierte Architekturen (SOA) in den letzen Jahren äußerst populär geworden und gelten als die zukünftige (und auch schon heutige) Basistechnologie zur Integration von Anwendungen (vgl. [Leser und Naumann, 2007, S.405]).

Die Services einer serviceorientierten Architektur werden heutzutage meist durch Web-Services realisiert. Diese setzen standardisierte Technologien wie XML, SOAP, WSDL und UDDI ein, die umfangreiche Unterstützung durch die Wirtschaft besitzen. Firmen wie IBM, Microsoft, Oracle und Sun[3] sind Mitglieder der Standardisierungsgremien, die für die Spezifikationen rund um Web-Services zuständig sind. Sie treiben die Entwicklungen im Bereich Web-Services stark voran.

Web-Services erlauben eine Interaktion zwischen Maschinen auf standardisiertem Wege. Die Kommunikation zwischen Web-Services findet in der Praxis meist durch SOAP-Nachricht über HTTP statt. Dies erlaubt den Nachrichten herkömmliche Firewalls ungehindert zu passieren. Zunächst galt dies als Vorteil von SOAP über HTTP, wird mittlerweile aber als Risiko eingestuft, da der Verkehr zwar von gewöhnlichem HTTP Verkehr zu unterscheiden ist, aber schadhafte Inhalte nicht von erwünschten Inhalten unterschieden werden können. Entweder passieren alle Nachrichten die Firewall oder keine.

SOAP-Nachrichten bestehen aus Klartext XML-Dokumenten. Die SOAP-Spezifikation selber sieht keine direkten Sicherheitsmechanismen vor, vielmehr delegiert sie die Realisierung von Sicherheitsmerkmalen an die Transport- und Anwendungsschicht.

[1] Sogenannte Enterprise Application Integration (EAI).
[2] Sogenannte Buisness-to-Buisness (B2B) und Buisness-to-Customer (B2C)-Integration.
[3] Die Aufzählung erhebt keinen Anspruch auf Vollständigkeit.

Es besteht ein Bedarf an zusätzlichen Mechanismen, die den Sicherheitsanforderungen einer komplexen Serviceumgebung gerecht werden können. Sie müssen Schutz vor Angriffen auf unzureichend geschützte Programmierung von Web-Service Anwendungen sowie die Web-Service Implementierung des Applikationsservers selbst bieten.

XML-Firewalls bzw. -Gateways sind in der Lage XML-Nachrichten auf invalide und schadhafte Bestandteile zu untersuchen und Nachrichten zu filtern. Solche Systeme können neben der Validierung von XML-Nachrichten die Aufgaben zur Autorisierung, Verschlüsselung und Signierung von XML-Nachrichten und ihren Bestandteilen übernehmen und den Web-Service um diese Aufgaben entlasten.

Die Firma IBM fordert als Fokus der Ausarbeitung die Betrachtung des geschäftlichen Mehrwerts einer XML-Firewall. Dies wird exemplarisch am Beispiel einer WebSphere DataPower Integration Appliance XI50 realisiert, welche die Schutzfunktionen der WebSphere DataPower XML Security Gateway XS40 integriert. Die ermittelten Resultate gelten für beide Systeme.

1.2 Zielsetzung und Abgrenzung

Zielsetzung

Ziel dieser Arbeit ist es, den technologischen Mehrwert einer Web-Service-Firewall, in Bezug auf relevante Schwachstellen, zu erarbeiten und davon den geschäftlichen Mehrwert abzuleiten.

Hierfür ist es notwendig zu zeigen, welche Schwachstellen der WebSphere Application Server in der Version 6.1 bei der Verwendung von SOAP über HTTP aufweist.

Anschließend muss geprüft werden, ob der Einsatz einer Web-Services-Firewall diese Lücken schließt.

Dazu werden gezielt manipulierte Nachrichten an Web-Services eines WebSphere Application Server gerichtet und die Folgen ausgewertet.

Abgrenzung

Im Rahmen dieser Arbeit werden die identifizierten Angriffstechniken definiert, charakterisiert und entsprechende Gegenmaßnahmen vorgestellt. Aufgrund der hohen Komplexität der möglichen Angriffsszenarien und der sich im Laufe der Arbeit ergebenden Gewichtung

durch den Auftraggeber, werden nicht alle identifizierten Angriffe näher behandelt. Angriffe die durch den Einsatz von standardisierten Web-Service Schutzmaßnahmen verhindert werden können, sind nicht Gegenstand der Arbeit.

Aufgrund der Wichtigkeit für den Auftraggeber, werden die Angriffe näher betrachtet, die Lücken in der Implementierung einer Web-Service Spezifikation des Applikationsservers nutzen. Für diese Lücken werden im Rahmen der Arbeit konzeptionelle Angriffe erarbeitet und in der Realisierung an einem Web-Service getestet. Sie stellen die Grundlage für die technische Bewertung und den daraus abgeleiteten geschäftlichen Mehrwert dar.

Es gelten Folgende Namenskonventionen: Das Wort Web-Services wird innerhalb des Textes in der deutschen Schreibweise verwendet. In den Abbildungen und Zitaten kann die Schreibweise abweichen.

1.3 Aufbau der Arbeit

Die Arbeit unterteilt sich in die Kapitel Grundlagen, Konzeption, Realisierung, Interpretation der Ergebnisse, Bewertung der Ergebnisse sowie Fazit und Ausblick.

Das Kapitel Grundlagen erläutert Begriffe und Konzepte die für ein Verständnis der Arbeit notwendig sind. Die Konzepte der serviceorientierten Architektur, Web-Services und ihre für diese Arbeit relevanten Komponenten XML, SOAP und WSDL werden vermittelt.

Der Teilabschnitt Sicherheit führt in Angriffsziele ein, aus denen sich Sicherheitsziele ableiten lassen. Die daraus resultierenden Bedrohungspotentiale werden in allgemeiner und Web-Service spezifischer Ausprägung dargestellt. Im Anschluß werden die Schutzmaßnahmen in Form bestehender Sicherheitsstandards und Web-Service-Firewalls und -Gateways vorgestellt.

Das Verständnis von XML, SOAP, WSDL und der im Teilbereich Sicherheit behandelten Konzepte und Begriffe ist notwendig für das Verständnis des Kapitels Konzeption.

Das Kapitel Konzeption geht auf die ermittelten Angriffstechniken und den daraus abgeleiteten konzeptionellen Angriffen ein. Die Angriffstechniken werden klassifiziert und im Rahmen ihrer Angriffsklassen erläutert. Aus den vorgestellten Angriffstechniken werden die Angriffe für eine weitere Betrachtung ausgewählt, die einen Angriff auf die Implementierung des Applikationsservers einer Web-Service Spezifikation durchführen.

Anhand der ausgewählten Angriffen soll die Verwundbarkeit von Web-Services gezeigt werden.

Der hierfür notwendige Versuchsaufbau unterteilt sich in die Vorstellung der Versuchsumgebung und die Erörterung des geplanten Versuchsablaufs. Der exemplarische Web-Service

wird näher vorgestellt. Neben einer Vorstellung des Quelltextes wird erläutert welche Datentypen verwendet werden und welche Absicht hinter der Auswahl steckt. Im Anschluß werden die Angriffe gegen den Web-Service gemäß ihrer Angriffstechnik konzipiert.

Das Kapitel Realisierung beschäftigt sich mit der praktischen Umsetzung von konzeptionellen Angriffen. Die für die Realisierung notwendigen Anwendungen werden vorgestellt. Die Abweichungen des Versuchsablaufs werden dargestellt.

Zum Ende des Kapitels finden die Versuchsdurchführungen statt. Nach direkten Angriffen auf den Web-Service folgt eine Wiederholung der Angriffe, bei denen der Web-Service durch eine XML-Firewall geschützt wird. Anhand der Resultate der Durchführungen können die Folgen der Angriffe ausgewertet und verglichen werden.

In diesem Abschnitt findet eine Zusammenfassung der Schwachstellen statt, darauf folgt eine Betrachtung des Applikationsserver und der Web-Service-Firewall in Bezug auf die Schwachstellen.

In dem Kapitel Interpretation findet eine Zusammenfassung der Schwachstellen statt. Mögliche Gegenmaßnahmen werden abgeleitet. Es findet eine Betrachtung der Implementierung des Applikationsservers und der Web-Service-Firewall in den untersuchten Punkten statt. Daraus resultiert eine Bewertung der Implementierungen.

Das Kapitel Bewertung der Ergebnisse findet die Bewertung des Mehrwerts statt. Nachdem ein Überblick über die gesamte Schutzfunktionalität der Web-Service-Firewall gegeben wird, folgt eine Betrachtung der Vor- und Nachteile der eingesetzten Web-Service-Firewall in Hinblick auf die untersuchten Aspekte. Sie dienen der Ableitung des qualitativen Mehrwertes.

Das Kapitel endet mit einer Verallgemeinerung der ermittelten Ergebnisse. Es wird gezeigt, welche generellen Aussagen sich für dienstbasierte Technologien anhand der identifizierten Schwachstellen treffen lassen.

Zum Abschluß findet im Kapitel Fazit und Ausblick eine Übersicht der angepeilten Ziele und eine Zusammenfassung der erreichten Ziele statt. Der Abschnitt Ausblick behandelt weiterführende Themen, die im Rahmen der Arbeit nicht bearbeitet werden konnten.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Konzepte und Begriffe erläutert die in der weiteren Arbeit verwendet werden. Zu Beginn wird auf serviceorientierte Architekturen (SOA), den dahinter liegenden Definitionen, dem grundlegenden Modell und die grundlegende Architektur eingegangen. Es folgt eine Erläuterung zu Web-Services und ihrer Definition, den Basiskomponenten, dem Rollenmodell und dem Aufbau der Architektur. Der folgende Abschnitt beschäftigt sich mit der Extensible Markup Language (XML), es wird näher auf die Struktur von XML-Dokumenten, XML-Namespaces sowie den Schemasprachen Document Type Definition (DTD) und XML-Schema eingegangen. Im Anschluß werden die beiden Technologien SOAP und WSDL vertieft. Das Ende des Kapitels widmet sich dem Thema Sicherheit, den allgemeinen und Web-Service spezifischen Ausprägungen, sowie den verfügbaren Sicherheitsstandards.

2.1 Serviceorientierte Architektur (SOA)

Die serviceorientierte Architektur ist in erster Linie ein abstraktes geschäftsgetriebenes Konzept, dass sich an den Geschäftsprozessen[1] eines Unternehmens ausrichtet und eine flexible Anpassung an Veränderungen im Geschäftsumfeld ermöglicht.

Ein Service bildet eine klar abgegrenzte fachliche Aufgabe eines Geschäftsprozesses in Form einer dynamisch nutzbaren Komponente ab. Nach außen werden Services über standardisierte Serviceschnittstellen und deren standardisierten Beschreibungen repräsentiert. Wiederverwendung der Services in unterschiedlichen Kontexten ist aufgrund der Modularität und Abgeschlossenheit der Services vorgesehen. Die daraus resultierende lose Kopplung sorgt für Interoperabilität zwischen Konsumenten und Service-Anbietern, ohne das Details über die spezifische Plattform oder Implementation des Kommunikationspartners vorliegen müssen (vgl. [Werawarana u. a., 2005, S.9]). Die Weiterverwendung bestehender Systeme als integrale Komponente ist eine Eigenschaft einer SOA, die sich von allen anderen Standard-Architekturen unterscheidet. SOA geht davon aus, dass bestehende Systeme existieren und diese als Service in einer SOA eingesetzt werden können [Liebhart, 2007, S.179].

[1] Sie stellen die fachlichen Prozesse dar.

Die standardisierten Serviceschnittstellen sind das zentrale Basiselement einer SOA. Sie garantieren die nahtlose Interaktion von Diensten und somit das Paradigma von SOA, dass Services das fundamentale Element für die Erstellung von Anwendungen sind. Sie verwandeln die Anwendungslandschaft in eine Servicelandschaft. Sajiva Weerawarana sieht in Services den nächsten Schritt der Entwicklung zur gröberen Granularität über das Komponentenmodell hinaus. Services bieten eine Abstraktion spezifischer Komponentenmodelle. Sie erlauben eine Benutzung dieser Komponenten, unabhängig von spezifischen Details des dahinter liegenden Komponentenmodells oder der Implementierung kennen zu müssen (vgl. [Werawarana u. a., 2005, S.15]).

Nach Werawarana u. a. [2005] gehört das Binden, Publizieren und Finden von Services zu den Kerneigenschaften die eine SOA effektiv machen (vgl. [Werawarana u. a., 2005, S.18]). Nach Melzer u. a. [2007] befindet sich das Anbieten, Suchen und Nutzen von Diensten über ein Netzwerk im Zentrum der Architektur (vgl. [Melzer u. a., 2007, S.7]). Konzeptionell herrscht Einigkeit über die Verwendung von Service-Verzeichnissen, sie sind Teil jeder grundlegenden Architekturbeschreibung. Allerdings handelt es sich eher um eine theoretische Einigkeit, in der Wirtschaft spielen Service-Verzeichnisse eine untergeordnete Rolle, sie finden de-facto keine Verwendung.

Geschäftsprozesse werden durch Orchestrierung von Services abgebildet, teilweise geschieht dies unternehmensgrenzenübergreifend. Längerfristig ist zu erwarten, dass sich BPEL (Business Process Execution Language) - ein Sprache zur Modellierung von Geschäftsprozessen - zur Abbildung der Geschäftsprozesse durchsetzt.

SOA wird das frühe Versprechen einer Reduktion der Total Cost of Ownsership (TCO) nicht halten können, es wird IT[2] nicht billiger machen. Neben erheblichen Einführungskosten ist nicht mit einer Reduktion der allgemeinen IT-Kosten zu rechnen. Dennoch ist aufgrund der Tatsache, dass Änderungen schneller umsetzbar werden und die IT auf der Prozessebene flexibler wird, dass Unternehmen in der Lage sein im eigentlichen Kerngeschäft effizienter zu werden und mehr Geld zuverdienen. In den seltensten Fällen dürfte dies die IT sein (vgl. [Melzer u. a., 2007, S.44]).

Die Einführung einer SOA in einem Unternehmen bringt immer einen Änderungsprozess in Gang. Auf der einen Seite widerspricht SOA der in vielen Unternehmen etablierten Strukturierung der IT gemäß den vorhandenen „Anwendungs-Silos“[3], auf der anderen Seite beeinflusst SOA die Art und Weise, wie Anwendungen konstruiert werden (vgl. [Liebhart, 2007, S.167]). Die Stärke von SOA liegt in der Flexibilisierung der Unternehmensprozesse. Da diese aber quer zu den Säulen der Systeme und damit der Organisation laufen, entsteht ein Machtkampf und ein hoher Abstimmungsbedarf. Eine erfolgreiche Umsetzung einer SOA dürfte sich daher unter Beibehaltung der Säulen-Organisation nicht realisieren lassen (vgl.

[2] Informationstechnik: Oberbegriff für die Informations- und Datenverarbeitung.
[3] Isolierte Anwendungen, die in sich geschlossene Teilaufgaben des Unternehmens abbilden.

[Melzer u. a., 2007, S.43]). Wenn eine SOA scheitert, so wird dies in vielen Fällen an nichttechnischem Gründen liegen (vgl. [Melzer u. a., 2007, S.45]).

Treiber eines solchen Konzeptes ist auf langer Sicht die Fähigkeit einer flexiblen Anpassung an geänderte Geschäftsbedingungen. Wiederverwendung sowie Nutzung bestehender externer Services erlaubt kürzere Entwicklungszeiten, sowie die Integration von immer komplexer werdenden Anwendungslandschaften zu Servicelandschaften.

Die Services einer SOA werden in der Regel durch Web-Services realisiert - prinzipiell können sie aber auch durch eine beliebige dienstbasierte Technologie wie CORBA[4], DCOM[5] oder RMI[6] realisiert werden.

2.1.1 Definition und Grundlegendes Modell

Definition
Nach Auffassung von Melzer u. a. [2007] und Liebhart [2007] existiert zurzeit keine allgemeine anerkannte und einheitliche Definition von SOA. Bei allen momentan gebräuchlichen Definitionen bestehen zwar immer wieder Überlappungen, es fehlen allerdings, häufig in einer Definition Aspekte, die von einer anderen Definition als entscheidend angesehen werden ([Melzer u. a., 2007, S.11], vgl. [Liebhart, 2007, S.6fff]).

Definition nach Melzer u. a. [2007]: *Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen Zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachunabhängige Nutzung und Wiederverwendung ermöglichen.* ([Melzer u. a., 2007, S.11])

Die Experton-Group[7] bezeichnet SOA als Architektur, die zum Ziel hat, eine lose Kopplung von interaktiven Software-Agenten zu erreichen, um (große) Anwendungslandschaften zu modularisieren und zu flexibilisieren (vgl. [Liebhart, 2007, S.6]).

Grundlegendes Modell
Alle Hersteller gehen von demselben SOA-Modell aus. Dies ist das erste Mal in der Geschichte der Informatik, dass eine Software Architektur von allen Herstellern unterstützt wird. Jeder Hersteller geht vom Service als standardisierte Grundkomponente aus. Alle Hersteller stellen Instrumente für die Modellierung von Geschäftsprozessen zur Verfügung und sehen eine Integrationsschicht vor. Sie unterscheiden zwischen Service-Ebene, Integration-Ebene,

[4] Common Object Request Broker Architecture
[5] Distributed Component Object Model
[6] Remote Method Invocation
[7] Eine unabhängiges Beratungsunternehmen, http://www.experton-group.de/

Orchestration-Ebene und dem User Interface (Präsentation-Ebene). Die Kombination der Produkte verschiedener Hersteller ist viel einfacher, als dies bislang der Fall war [Liebhart, 2007, S.8].

2.1.2 Grundlegende Architektur

Abbildung 2.1 zeigt das Rollenmodell einer SOA in Anlehnung an Melzer u. a. [2007]. Die Grundfunktionalität besteht aus: **Kommunikation**, **Service-Beschreibung** und einem **Verzeichnisdienst**. Sie sind Grundlage der Interaktion zwischen den drei identifizierbaren Rollen: **Service-Anbieter**, **Service-Konsument** und **Service-Verzeichnis**.

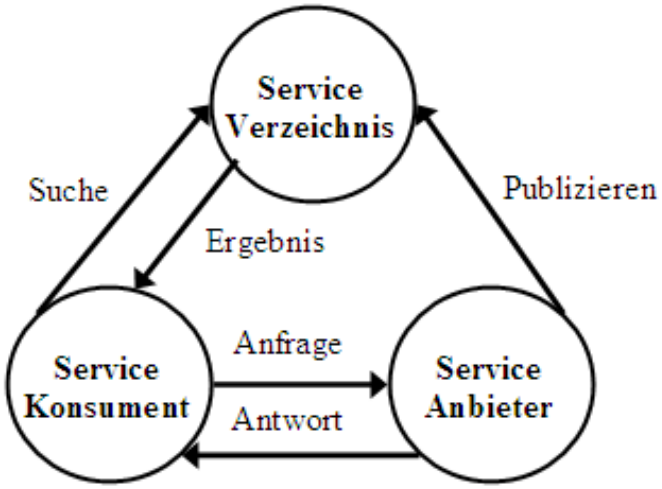

Abbildung 2.1: SOA Rollenmodell
angelehnt an Melzer u. a. [2007]

Um die Implementation eines Services bekannt zu machen, publiziert ein Service-Anbieter die Service-Beschreibung der öffentlichen Schnittstellen des Services in einem Service-Verzeichnis. Der Service-Konsument sucht und findet einen Service (gemäß seiner Anforderungen) in einem Service-Verzeichnis und erhält dessen Service-Beschreibung. Anhand der Service-Beschreibung kann der Service-Konsument die Verbindung dynamisch zum Service-Anbieter herstellen. Die Nutzung des Services erfolgt durch Anfrage und Antwort zwischen Konsumenten und Anbieter (siehe [Melzer u. a., 2007, S.12fff]).

2.2 Web-Services

Web-Services und serviceorientierte Architekturen werden oft in einem gemeinsamen Zusammenhang genannt, dennoch unterscheiden sie sich in grundlegender Art und Weise. Wie im vorherigen Absatz erläutert handelt es sich bei SOA um ein abstraktes Architektur Konzept, während Web-Services eine bedeutende Implementierung einer SOA Realisierung darstellen (siehe [Werawarana u. a., 2005, S.31]).

Bei Web-Services handelt es sich nicht um eine einzelne Technologie, vielmehr setzen sie sich aus einer Sammlung von offenen Spezifikationen und Standards zusammen, die von unterschiedlichen Standardisierungsgremien geführt werden.

2.2.1 Definition

Nach Melzer u. a. [2007] existiert keine verbindliche Definition des Begriffes Web-Services. Beispielhaft für die Vielzahl von Definitionen im Umfeld von Web-Services seien hier repräsentativ die Definitionen der *Web-Services Architecture Working Group*, als Teil des offiziellen Web-Services Standardisierungsgremiums *World Wide Web Consortium* (W3C). Die Definition des W3C wurde mit der Zeit konkretisiert.

Nach W3C (Oktober 2002) *„A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols.“*

Nach W3C (August 2003) *„A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“*

Die Fassung vom August 2003 wurde in der aktuellen Fassung vom Februar 2004 beibehalten.

Ein wesentlicher Aspekt der Definitionen stellt die Gewährleistung der Interoperabilität in der Maschine-Maschine-Kommunikation dar. Während die frühere Fassung noch recht unkonkret bezüglich der eingesetzten Technologien war, so ist die aktuelle Definition des W3C recht konkret und nennt die Technologien und Spezifikation die Verwendung finden. XML, SOAP und WSDL stellen zusammen mit UDDI die Basiskomponenten dar, die im folgenden Abschnitt näher erläutert werden.

2.2.2 Basiskomponenten

In einer Web-Services-Architektur werden diese Komponenten zurzeit mithilfe der folgenden Standards beschrieben:

SOAP [8] *beschreibt das XML-basierte Nachrichtenformat der Kommunikation und dessen Einbettung in ein Transportprotokoll. Die Netzwerkprotokolle SOAP und RPC-XML basieren auf dem Nachrichtenaustausch durch XML Dokumente* ([Melzer u. a., 2007, S.51]). Die SOAP Spezifikation gilt oft als Synonym für Web-Services - sie stellt den wichtigsten Standard der Web-Services dar.

WSDL *ist eine, ebenfalls XML-basierte Beschreibungssprache um Web-Services (Dienste) zu beschreiben* [Melzer u. a., 2007, S.51]. Sie ist die Grundlage der Benutzung der Services durch den Service-Konsumenten.

UDDI beschreibt einen Verzeichnisdienst für Web-Services. *UDDI (Universal Description, Discovery and Integration protocol) spezifiziert eine standardisierte Verzeichnisstruktur für die Verwaltung von Web-Services-Metadaten. Zu den Metadaten zählen allgemeine Anforderungen, Web-Services-Eigenschaften oder die benötigten Informationen zum Auffinden von Web-Services* [Melzer u. a., 2007, S.51]. Ein UDDI Verzeichnis muss mit WSDL-Dokumenten umgehen können und selbst auch als Web-Service verfügbar sein. (vgl. [Melzer u. a., 2007, S.51]).

In den meisten Web-Services Definitionen findet die Komponente UDDI keine Erwähnung. Ein Verzeichnisdienst stellt kein notwendiges Kriterium für den Einsatz von Web-Services dar, sondern vielmehr die Infrastruktur zum Auffinden von geeigneten Web-Services. Solange kein anderer Service oder Service-Anbieter als die bekannten benötigt wird, gibt es keinen Grund einen Verzeichnisdienst aufzurufen.

Nach Melzer u. a. [2007] macht erst eine solche Infrastrukturkomponente die Zusammenstellung der Web-Services Spezifikationen zu einer serviceorientierten Architektur. Aufgrund dessen geht die Mehrzahl von Architekturbeschreibungen davon aus, dass Verzeichnisdienste eine Basiskomponente darstellen (vgl. [Melzer u. a., 2007, S.52]). Nach Liebhart [2007] ist der Standard UDDI in der betrieblichen Realität selten von großem Nutzen, da ein Unternehmen es vermeiden möchte, dass dieselbe Funktionalität durch mehrere Services[9] abgedeckt wird (siehe [Liebhart, 2007, S.14f]). UDDI weist im Bereich Verwaltung erhebliche Mängel auf (vgl. [Liebhart, 2007, S.15]), die fehlende Moderation wirkt sich negativ auf die Qualität der Einträge aus.

[8] Ehemals Akronym für Simple Object Access Protocol, seit Version 1.2 steht SOAP für sich selbst.
[9] Vergleichbare Angebote anderen Firmen.

2.2.3 Rollenmodell

Abbildung 2.2 ist eine konkrete Ausprägung der Abbildung 2.1. Die Grundfunktionalität lässt sich auch hier wieder erkennen: **Kommunikation**, **Service-Beschreibung** und **Verzeichnisdienst** als Grundlage der Interaktion zwischen den drei Rollen: **Service-Anbieter**, **Service-Konsument**, **Service-Verzeichnis**.

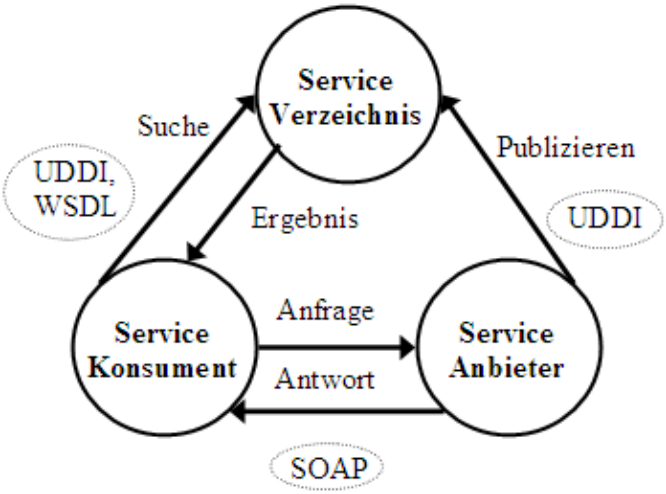

Abbildung 2.2: Web-Services Rollenmodell
angeleht an Melzer u. a. [2007]

Hierbei handelt es sich um das Rollenmodell einer Web-Services-basierten SOA, der in 2.1.3 bereits abstrakt erläuterten Rollen und Grundfunktionen.

Ein Service-Anbieter der einen Dienst in Form eines Web-Service anbieten möchte, erstellt von diesem zunächst eine WSDL-Schnittstellenbeschreibung in Form eines entsprechenden XML-Dokuments. Dieses WSDL-Dokument wird publiziert, in dem es ganz oder in definierten Teilen zu einem UDDI-basierten Verzeichnisdienst transferiert wird. Laut Spezifikationen stellen UDDI-Implementierung eine SOAP-Schnittstelle zur Verfügung, die vom UDDI-Gremium mittels WSDL-Dokumenten beschrieben ist - sie sind als Service nutzbar. Ein Service-Konsument sucht über diese Schnittstelle des Verzeichnisdienstes einen entsprechenden Service und fordert bei erfolgreicher Suche dessen Schnittstellenbeschreibung (das WSDL-Dokument) an. Der Verzeichnisdienst liefert hierzu eine Referenz (URI) auf das WSDL-Dokument, dass der Service-Konsument in einem weiteren Schritt vom Service-Anbieter anfordert. Anschließend werden mithilfe der WSDL-Beschreibung die Programmteile erzeugt, welche die Anwendung des Service-Konsumenten in die Lage versetzt, mit der Anwendung des Service-Anbieters mittels SOAP zu kommunizieren (vgl. [Melzer u. a., 2007, S.52f]).

2.2.4 Architektur

Die folgende Abbildung 2.3 zeigt anhand des Web-Services Stacks der *W3C Web Services Architecture Group* das Zusammenspiel der einzelnen Technolgien durch eine Bottom-Up Sicht der Web-Services-Architektur.

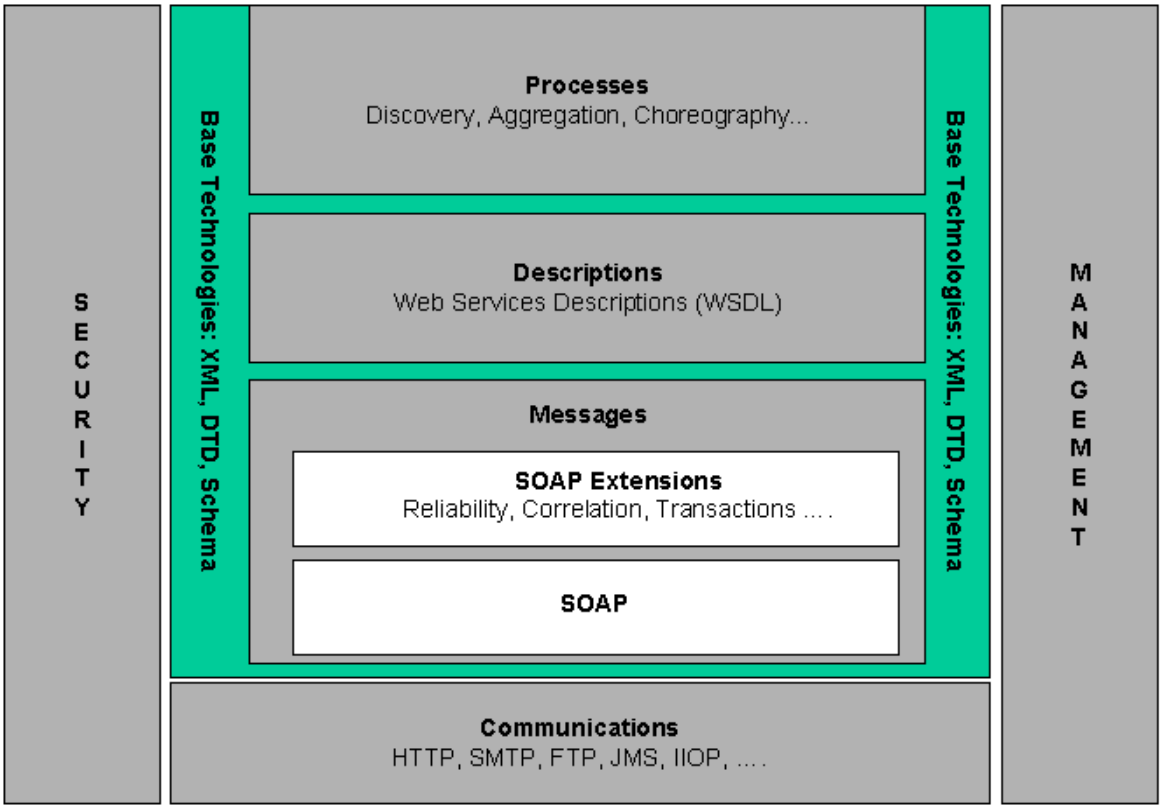

Abbildung 2.3: Web-Services Architektur Stack
Quelle: [WSA, 2004, Kapitel 3.2]

Die unterste Schicht *Communications* ist die Transportschicht. Sie abstrahiert den Transport von Nachrichten über ein Netzwerk für den darüber liegenden Web-Service. Obwohl in der Theorie in dieser Schicht unterschiedliche Transportprotokolle zum Einsatz kommen können, hat sich die Verwendung des Hyper Text Transport Protokoll (HTTP) HTTP [1999] in der Praxis etabliert.

Die darüber liegende Schicht *Base Technologies* umfasst die Kernkomponenten. XML und seine verwandten Technologien, beispielsweise zur Beschreibung von Schemata oder Namensräumen, bilden die Grundlage eines gemeinsamen Datenmodells (siehe [Melzer u. a., 2007, S.53]). SOAP, WSDL und ein Großteil der durch diese Schicht gekapselten Technologien basieren auf XML. Kapitel 2.3 geht detaillierter auf XML ein.

Die Schicht *Messages* unterteilt sich in die Schichten *SOAP* und die darüber liegende Schicht *SOAP Extensions*. SOAP ist ein einfaches Protokoll für den Austausch von strukturierter Information durch XML-Daten über das Web (siehe [Liebhart, 2007, S.15]). Abschnitt 2.4 beschäftigt sich ausführlicher mit SOAP.

Die darüber liegende Schicht *Descriptions besteht aus einem XML-Format zur programmatischen Beschreibung von Web-Services: WSDL beschreibt was ein Service tun kann, wo er zu finden und wie er aufzurufen ist. WSDL stellt den Vertrag zwischen den Services dar, also den zentralen Mechanismus, der eine Anwendung und ihre Komponenten miteinander verbindet* (siehe [Liebhart, 2007, S.16]). In Abschnitt 2.5 wird auf WSDL genauer eingegangen.

Die oberste Schicht der Architektur bildet *Processes*. Hier siedelt sich das Auffinden von Web-Services in einem globalen UDDI Verzeichnisdienst an, dies stellt zumindest in der Theorie, ein Wichtigen Prozess dar. Ein Atomarer oder Zusammengesetzter Web-Service stellt ebenfalls einen Prozess dar. Durch Zusammenspiel mehrerer Web-Services ist es möglich, Prozesse zu modellieren und aus einzelnen Aufgaben einen komplexe Anwendung entstehen zu lassen (vgl. [Melzer u. a., 2007, S.54]). Dies geschieht beispielsweise durch Web-Service Orchestration und Web-Service Choreography.

Neben diesen Kernkomponenten existieren verschiedene, teils Schichten übergreifende Technologien. Die Technologien der *Management* Komponente decken Management Aspekte des Quality of Service (QoS) ab, die eine Verwaltung und Auswertung der Dienstgüte der Web-Service Ressourcen adressieren. Sie bestehen aus Messung (Monitoring), Einhaltung (Controlling) und Auswertung (Reporting) der Dienstgüte. Die Technologien der *Sicherheits* Komponente decken Aspekte des QoS ab, die Zuverlässigkeit und Sicherheit der Web-Service Ressourcen und ihrer Nachrichten adressieren. Zu ihnen Zählen beispielsweise die Technologien: WS-Security, WS-Policy, WS-Reliablitity. Weitere Aspekte des QoS sind Richtlinien für die Koordination zwischen mehreren Partner sowie Standards für verteilte Transaktionen, vertreten durch die Technologien WS-Coordination und WS-Transaction.

Die Security Spezifikationen werden im Kapitel 2.6.5 Sicherheitsstandards genauer behandelt.

Die Technologien der Komponenten *Processes* und *Management* sowie weitere ergänzende Technologien sind für das Verständnis der Arbeit nicht von Bedeutung und werden nicht weiter Betrachtet.

2.2.5 Interoperabilität

Da Web-Services auf XML basieren, wurde mitunter der Eindruck vermittelt, dass damit alle Interoperabilität per se gelöst seien. Allerdings lag das eigentliche Problem nicht darin, dass

ein XML-Dokument auf einem andere System nicht gelesen werden könnte. Vielmehr lag die Schwierigkeit darin begründet, dass Hersteller von Software-Systemen die Struktur einer SOAP-Nachricht oder einer WSDL-Schnittstellenbeschreibung aufgrund einer zukunftsfähigen Spezifikation festlegen müssen [Melzer u. a., 2007, S.64f].

Die Zukunftsfähigkeit zielt auf Versionskompatibilität zwischen Spezifikationen ab, die selbst bei signifikanten Änderungen nicht zwangsläufig zu Inkompatibilität führt. Erst durch Interpretationsspielräume innerhalb der Spezifikation wird diese Zukunftsfähigkeit ermöglicht. Allerdings haben diese Spielräume zu nicht vollständig interoperablen Web-Services Implementierungen geführt (vgl. [Melzer u. a., 2007, S.65]).

Aus diesem Grund wurde das Gremium WS-Interoperability[10] gegründet, mit dem Ziel Interoperabilität zu erreichen. Vergleicht man den Grad der Interoperabilität von Web-Services-Implementierungen von vor zwei Jahren mit heute, so bleibt nur festzustellen, dass dieser Ansatz zumindest erste Erfolge zeigt (vgl. [Melzer u. a., 2007, S.65]).

2.3 Extensible Markup Language (XML)

In diesem Abschnitt wird der für diese Arbeit relevante Teilausschnitt aus dem Kern der XML Familie behandelt, der als Grundlage zum Verständnis der Dokumentenstruktur der XML-Anwendungen SOAP und WSDL dient. In den darauf folgenden Abschnitten werden SOAP und WSDL im Detail behandelt.

Die *Extensible Markup Language* (XML) wurde vom World Wide Web Consortium (W3C) für den Austausch von hierarchisch strukturierten Daten im Internet entwickelt und ist eine Teilmenge der *Standard Generalized Markup Language* (SGML) (vgl. [Leser und Naumann, 2007, S.23]). Die XML Spezifikation, definiert eine Metasprache zur Definition neuer Sprachen. Die Festlegung der XML-Dokumentenstrukur einer anwendungsspezifischen Sprache erfolgt unter Verwendung einer Schemasprache, welche strukturelle und inhaltliche Einschränkungen der Sprache definiert. Die Definitionen der Schemasprachen *XML-Schema*[11] und *Regular Language Description for XML New Generation* (RELAX NG) werden entsprechend einer *XML-konformen Syntax* erstellt, während Definitionen der funktional eingeschränkteren *Document Type Definition* (DTD) in *Extended Backus Naur Form* (EBNF) verfasst werden. Ein *wohlgeformtes* XML-Dokument, dass sich an die Dokumentenstruktur einer Schemasprache hält, gilt als *valide*. Die Beliebtheit der offenen XML Standards beruht auf der Plattformunabhängigkeit, es existieren Implementation für alle wichtigen Systeme und Sprachen.

[10] http://www.ws-i.org
[11] Die von der W3C empfohlene Schemasprache.

Die XML Spezifikation Version 1.0 ist, abgesehen von nachträglichen Fehlerkorrekturen, seit der Standardisierung 1998 unverändert geblieben. Die wichtigste Neuerung der in 2004 standardisierten Version 1.1 ist die Unterstützung neuerer Unicode Versionen. Trotz minimaler Änderungen in der Spezifikation sind XML-Dokumente der unterschiedlichen Versionen nicht kompatibel. Helmut Vonhoegen folgert daraus, dass die Version 1.1 in den nächsten Jahre in der täglichen Arbeit mit XML-Daten kaum eine Rolle spielen wird (siehe [Vonhoegen, 2007, S.35]).

2.3.1 Struktur eines XML Dokuments

XML-Dokumente sollten mit einer XML-Deklaration beginnen, die die verwendete XML-Version spezifiziert. Ein XML-Dokument besteht aus einem oder mehreren Elementen. Es existiert genau eine Wurzel- oder Dokument-Element, von dem kein Teil im Inhalt eines anderen Elementes enthalten ist. Ein XML-Element besteht aus einem Start-Tag und End-Tag. Zwischen den beiden Tags kann eine Zeichenkette oder ein weiteres XML-Element auftreten. Das Start-Tag kann optional Attribute als Namen-Wert Paare beinhalten. Obwohl ein Empty-Tag ein XML-Element ohne Inhalt ist, kann es auch in sich geschlossen sein. Daneben existieren XML Processing Instructions und Kommentare auf die nicht weiter eingegangen wird. Ein XML-Dokument, dessen Elemente dieser Struktur folgt und demnach korrekt ineinander verschachtelt ist heißt wohlgeformt (vgl. [XML, 2006, Kapitel 3].

Wohlgeformte und valide XML-Dokumente

	Wohlgeformt	Valide
Genau ein Wurzelement	Ja	Ja
Korrespondierender schließender Tag	Ja	ja
Identisch Schreibweise des öffnenden und schließenden Tags	Ja	Ja
Korrekte Reihenfolge bei verschachtelten Elementen	Ja	Ja
Anführungszeichen um Attributwerte	Ja	Ja
Dokumentenstruktur ist vorgegeben (DTD oder XML-Schema)	-	Ja

Tabelle 2.1: Wohlgeformte und valide XML-Dokumente
angelehnt an [XML, 2006, Kapitel 2.8]

Die Tabelle 2.1 veranschaulicht die Regeln nach denen ein Dokument wohlgeformt und valide ist. Zur Vermeidung von Inkompatibilitäten sollten XML-Dokumente die zwischen Anwendungen ausgetauscht werden immer gültig sein.

Quelltext 2.1 zeigt ein nicht wohlgeformtes XML-Dokument. Das schließende Tag </p:namen> stimmt nicht mit dem öffnenden Tag <p:name> überein.

```
<?xml version="1.0"?>
<kunde>
  <p:person>
    <p:name>Musterman</p:namen>
</kunde>
```

Quelltext 2.1: Beispiel eines nicht wohlgeformten XML-Dokument

Quelltext 2.2 zeigt ein Beispiel für ein invalides XML-Dokument in Verbindung mit einem Schema, dessen Definition festlegt, dass ein „person“-Element lediglich ein „name“-Element beinhalten darf. Das XML-Dokument ist invalide, da ein weiteres „vorname“-Element enthalten ist.

```
<?xml version="1.0"?>
<kunde>
  <p:person>
    <p:vorname>Musterman<p:vorname>
    <p:name>Musterman<p:name>
  </p:person>
</kunde>
```

Quelltext 2.2: Beispiel eines invaliden XML-Dokument

XML-Namespaces

Unter Verwendung von Namensräumen können mehrere XML-Sprachen in einem einzelnen Dokument verwendet werden. Namensräume ermöglichen die eindeutige Unterscheidung von mehrfach vorkommenden Elementnamen mit unterschiedlichem Inhalt. Dazu werden Elemente mit einem frei wählbaren *Prefix*, einer Zeichenkette gefolgt von einem Doppelpunkt, gekennzeichnet (siehe XML-NS [2006]). Quelltext 2.3 verdeutlicht die Verwendung von XML-Namespaces.

```
<?xml version="1.0"?>
<kunde>
  <p:person>
    <p:name>Musterman</p:name>
  </p:person>
  <t:telefon>
    <t:nummer>1234568</t:nummer>
  </t:telefon>
</kunde>
```

Quelltext 2.3: Beispiel XML-Namespaces

2.3.2 XML-Schemasprachen

Document Type Definition (DTD)

Eine DTD ist eine Schemasprache die eine Einhaltung einer gemeinsamen Struktur für XML-Dokumente erlaubt. DTD stammt, als Teil der XML Spezifikation, aus einer Zeit in der XML Namensräume nicht existierten. In einer DTD werden Elementtypen und ihre Attribute, Entitäten und ihre Notation als EBNF deklariert (siehe XML [2006]). In der Anfangszeit von XML fand DTD eine weite Verbreitung, die mit erscheinen von XML Schema rückläufig war (siehe [Werawarana u. a., 2005, S.25]).

XML-Schema (XSD)

XML-Schema ist die W3C Empfehlung einer Schemasprache zur Strukturierung von Dokumenten und Spezifikation von Typ Definitionen. Die XML Schemasprache wird auch als XML-Scheme-Defintion (XSD) bezeichnet. Sie erlaubt mittels XML-Syntax die Spezifizierung von XML-Strukturen, ähnlich wie DTD aber in einer viel mächtigeren Art und Weise. Zusätzlich zur Struktur erlaubt XML-Schema die Spezifizierung der Datentypen. Sie definiert eine Menge primitiver Datentypen, die zur Definition eigener Attribut- und Elementwerte sowie rekursiver Typkonstrukte Verwendung finden, aus deren Komposition beliebig komplexe Typ-Strukturen definiert werden können. XML Schema ist Leistungsfähig und hat weitläufig Unterstützung durch die Wirtschaft (siehe [Werawarana u. a., 2005, S.25]).

2.4 SOAP

Die XML protocol Working Group des W3C hat am 27. April 2007 die zweite Fassung der SOAP Spezifikation Version 1.2 als Standard verabschiedet (siehe SOAP12 [2007]). Stand es in früheren Versionen noch als Akronym für Simple Object Access Protocol, so steht es heute für sich selbst.

SOAP Version 1.2 ist ein leichtgewichtiges Protokol für den Austausch strukturierter Informationen in einer dezentralisierten, verteilten Umgebung. Unter Verwendung von XML Technologien definiert es ein erweiterbares Framework für XML-basierte Nachrichtensysteme die ein Nachrichtenkonstrukt anbieten, dass über eine Vielzahl von Protokollen ausgetauscht werden kann. Das Framework wurde frei von Abhängigkeiten zu bestimmten Programmier-Modellen und anderen implementierungsspezifischen Details konzipiert (siehe SOAP12 [2007]). Die Übertragung von SOAP-Nachrichten findet in der Praxis meist per HTTP statt, dies erlaubt den Nachrichten herkömmliche Firewalls ungehindert zu passieren.

Zunächst galt dies als Vorteil von SOAP über HTTP, wird mittlerweile aber als Risiko eingestuft, da der Verkehr praktisch nicht von gewöhnlichem HTTP Verkehr zu unterscheiden ist (vgl. [Melzer u. a., 2007, S.64]).

In der SOAP Spezifikation wurden Sicherheitsaspekte bewusst ausgeklammert. SOAP Nachrichten werden im Klartext übermittelt (vgl. [SOAP12, 2007, Kapitel 7]). Einzelne Aspekte der Sicherheit werden durch Delegation an Anwendungs- und Transportschicht realisiert. SOAP über HTTPS erlaubt eine Punkt-zu-Punkt Verschlüsselung von Nachrichten auf der Transportschicht unter Verwendung von Secure Socket Layer (SSL). Zusätzliche „SOAP Extensions“ ermöglichen den Einsatz von Sicherheitsstandards, die Sicherheitsaspekte auf Ebene des SOAP-Umschlages abdecken - wie z.B. XML Encryption, XML Signature, WS-Security oder SAML.

Auf die Sicherheitsstandard wird in Kapitel 2.8.5 näher eingegangen.

2.4.1 Nachrichtenstruktur

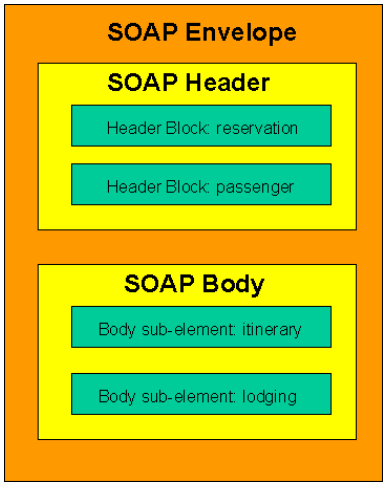

Abbildung 2.4: SOAP Nachrichtenstruktur
Quelle: [SOAPENV, 2007, Kapitel 2.1]

Die grundlegende Spezifikation für den Aufbau einer SOAP-Nachricht ist „SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)“. Die Struktur einer SOAP-Nachricht wird aus Abbildung 2.4 ersichtlich.

Eine im SOAP Format auszutauschende Meldung besteht aus drei Teilen:

SOAP-Envelope ist der Umschlag (Envelope), in den XML-Daten verpackt werden können. Der Envelope bildet das Wurzelelement des XML-Dokuments. In ihm sind die anderen

beiden Teile der SOAP-Nachricht gekapselt. Unter Verwendung eines XML-Schemas, dass dem Envelope als Namensraum dient, wird die konkrete SOAP Spezifikation der SOAP-Nachricht festgelegt.

Beispiel:

```
<env:Envelope xmlns:env="...">
  <!- SOAP Header -->
  <!- SOAP Body -->
<env:Envelope>
```

SOAP-Header ist ein optionaler Bestandteil einer SOAP Nachricht. *Der Inhalt des SOAP-Header-Elements ist nicht in der SOAP Spezifikation definiert. Er enthält Informationen über Sicherheit, Routing und Zusatzinformationen für die korrekte Verarbeitung des SOAP-Bodys. Diese Informationen sind in Elementblöcken strukturiert. Sie werden über eine eindeutige URI (Uniform Ressource Identification) identifiziert und können so den einzelnen Knoten zugeordnet werden. Dieser Mechanismus ist sehr flexibel, weil dadurch zusätzliche Spezifikationen von solchen Blöcken definiert werden können* [Liebhart, 2007, S.15]. Das Attribut „Role“ spezifiziert den Empfänger der das Header-Element verarbeiten darf. Das optionale Attribut „mustUnderstand“ bestimmt, ob der adressierte Knoten diesen Header-Eintrag auswerten können muss. Andernfalls wird die weitere Bearbeitung der SOAP-Nachricht unterbrochen und eine Fehlermeldung wird zurück zum Absender geschickt (vgl. [Melzer u. a., 2007, S.74f]).

Beispiel:

```
<env:Header>
  <ns:authentication
    xmlns:ns="http://anyURL/XML-Schema-Auth"
    env:role="http://anyURL/Role"
    env:mustUnderstand="true">
    <ns:security-token>cryptedToken</ns:security-token>
  </ns:authentication>
</env:Header>
```

SOAP-Body enthält die zu transportierenden XML-Daten. Der XML-Body muss (mit Ausnahme eines vorhanden Prologs) ein wohlgeformtes XML-Dokument darstellen. Der eigentliche Inhalt des SOAP Bodys wird hier nicht näher spezifiziert, da dieser anwendungsbezogen variiert (siehe [Melzer u. a., 2007, S.75]). Zusatzspezifikationen regeln den Versand von Binärdaten als SOAP-Anhänge.

Beispiel:

```
<env:Body>
  <ns:nutzlast xmlns:ns="...">
```

```
    <ns:msg>Hier steht die eigentliche Information</msg>
  </ns:nutzlast>
</env:Body>
```

2.4.2 SOAP-Fehler

Bei einer Kommunikation können immer, das heißt an beliebiger Stelle in der Kommunikationskette, Fehler auftreten. Im Falle eines Fehlers darf ein *Fault Block* als einziges Kindelement des SOAP Body übertragen werden. Die Elemente „Code“ und „Reason“ sind immer Teil eines Fault Blocks und geben Auskunft über die von der SOAP Spezifikation festgelegten Kodierung der Fehlerquelle und einer textuellen Beschreibung des aufgetretenen Fehlers. Die Elemente „Node“, „Role“ und „Detail“ sind optionale Bestandteile des Fault Blocks, die Auskunft geben über die Fehlerstelle der SOAP-Kommunikation, der Rolle des Node bei dem der Fehler aufgetreten ist, sowie frei definierbare Zusatzinformationen zum Fehler (Siehe [SOAP12, 2007, Kapitel 5.4] und [Melzer u. a., 2007, S.76ff]).

2.4.3 SOAP-Knoten und -Rollen

Abbildung 2.5 zeigt, dass ein SOAP-Knoten ein „Sender“, „UltimateReceiver“ oder „Intermediary“ sein können. Die Menge aller Intermediaries bilden gemeinsam mit dem „UltimateReceiver“ den Nachrichtenpfad. Um Teile eines Nachrichtenpfades zu identifizieren, beteiligt sich jeder Knoten in einer oder mehreren Rollen. Die SOAP Spezifikation sieht zwei feste Rollen vor: „Next“ und „UltimateReceiver“. „Next“ ist eine Universalrolle zu der jeder SOAP-Knoten, ausgenommen der Sender, gehört. „UltimateReceiver“ ist die Rolle des Endknotens in einem Nachrichtenpfad, typischerweise ist dies eine Anwendung oder in einigen Fällen eine Infrastrukur, die Aufgaben für die Anwendung durchführt (siehe [Werawarana u. a., 2005, S.38]).

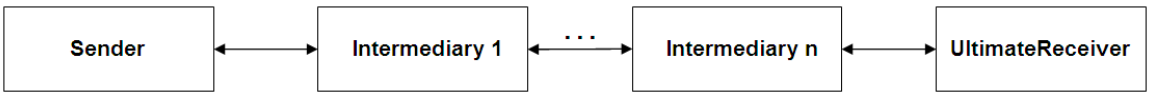

Abbildung 2.5: SOAP-Knoten: Der Weg von Sender über n-Intermediaries zum Ziel

2.5 Web Services Description Language (WSDL)

Die *Web Services Description Language* (WSDL) ist eine XML-Sprache zur Beschreibung von Web-Service-Schnittstellen. Sie unterliegt der Standardisierung durch das W3C. Die Version 1.1 WSDL11 [2001] der WSDL Spezifikation existiert seit 2001 und ist bei allen gebräuchlichen Web-Service-Umgebungen standardmäßig in Verwendung. Der Nachfolger-Standard WSDL 2.0 hat seit Juni 2007 den Status einer W3C Recommendation, allerdings ohne große Verbreitung in realen Systemen (vgl. [Melzer u. a., 2007, S.102] und WSDL20 [2007]).

WSDL beschreibt, welche Funktionalität ein Service bietet, welche Struktur die Daten haben, wie auf den Service zugegriffen wird und wie man ihn aufrufen kann. WSDL stellt den Vertrag zwischen den Services dar. (vgl. [Liebhart, 2007, S.16]).

Sofern nicht anders angegeben, bezieht sich die Beschreibung in dieser Arbeit auf WSDL 1.1.

2.5.1 Aufbau

Wie in Abbildung 2.6 ersichtlich, besteht ein WSDL-Dokument aus einem abstrakten und einem konkreten Teil. Der abstrakte Teil definiert sprach- und plattformunabhängige Teile:„types“, “message“, „operation“ und „portType“. Der konkrete Teil definiert „binding“, „service“ und ihre „ports“.

Abstrakter Teil

types	Ein Container, der die Definition von eigenen Datentypen enthält, die den Nachrichtenaustausch beschreiben.
message	Abstrakte Definition der auszutauschenden Nachrichten, die aus einem oder mehreren Teilen (parts) besteht. Jeder Teil behandelt den Datenaustausch einer einzelnen Übertragung.
operation	Abstrakte Beschreibung einer von einem bestimmten Service unterstützen Aktion.
portType	Abstrakte Menge von Operationen, die von einem oder mehreren Netzwerk-Endpunkten unterstützt werden.

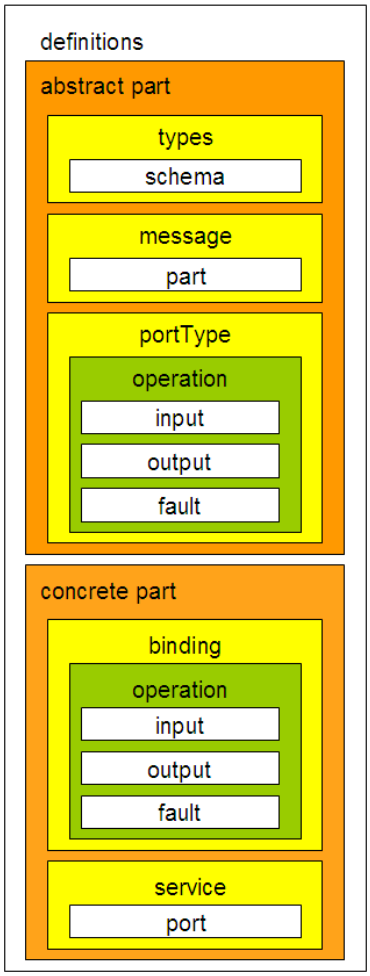

Abbildung 2.6: Struktuer einer WSDL-Servicebeschreibung
in Anlehnung an WSDL11 [2001]

Konkreter Teil

binding	Konkrete Beschreibung des Protokolls und die von einem bestimmten portType unterstützen Aktionen.
port	Ein bestimmter Netzwerk-Endpunkt (Binding und Netzwerk-Adresse).
service	Sammlung zusammenhängender Endpunkte.

2.5.2 Interaktionsmuster der Operationen

Die Interaktionsmuster sind Teil der WSDL Spezifikation und beschreiben die Übertragungsart, die bei der Kommunikation stattfindet. Sie werden im Folgenden in Tabelle 2.2 dargestellt:

Übertragungsart	Aktion des Endpunktes
One-Way	erhält eine Nachricht
Request-Response	erhält eine Nachricht, sendet entsprechende Nachricht
Solicit-Response	sendet eine Nachricht, erhält entsprechende Nachricht
Notification	sendet eine Nachricht

Tabelle 2.2: Interaktionsmuster der Operationen

Die Interaktionensmuster werden durch Kombination von „input/output-messages“ innerhalb einer „operation“ abgebildet.

2.5.3 Unterschiede zu WSDL 2.0

Nach Werawarana u. a. [2005] wurden mit WSDL 2.0 grundlegende Probleme und Einschränkungen von WSDL 1.1 angegangen. Die Grundstruktur von WSDL 2.0 ist, bis auf wenige Unterschiede, gleich geblieben. Das Wurzelelement wurde von „definition“ in „description“ umbenannt. „portType“ wurde durch „interfaces“ ersetzt und das „message“-Element gestrichen. Nachrichten werden innerhalb des „interface“-Elementes beschrieben, statt in einem separaten Element. Ein darin enthaltenes „operation“-Element verwendet das Message Exchange Pattern (MEP) gemäß seines Pattern-Attributes. Interfaces können durch das Konzept von Einfach-/Mehrfachvererbung erweitert werden. Ein „service“ ist nun eine Sammlung von „endpoints“, die ein gemeinsames Interface implementieren. (siehe [Werawarana u. a., 2005, S.122fff]).

2.6 Sicherheit

Dieser Abschnitt soll einen grundlegenden Einblick in Sicherheitsanforderung an Web-Service Dienste geben. Es wird ein kurzer Ausblick auf Angriffsziele und den daraus resultierenden Sicherheitszielen gegeben. Danach folgen allgemeine Bedrohungspotentiale und ihre Web-Service spezifischen Ausprägungen. Abschließend werden die standardisierten Sicherheitsmaßnahmen vorgestellt.

„Gerade im Unternehmensumfeld besteht ein erhöhter Schutzbedarf für den Einsatz von Web-Services, insbesondere wenn diese wichtige Prozesse einer Geschäftslogik nach außen hin öffnen. Die Realisierung eines Geschäftsprozesse über eine Vielzahl lose gekoppelter Services, erfordert beispielsweise mehr Authentisierungsvorgänge, eine jederzeit gewährleistete Vertraulichkeit sowie eine höhere Integrität als in einem monolithischen System. Ein Prozess-Design über Unternehmensgrenzen hinweg und zwischen quasi-anonymen Teilnehmern verstärkt diese Sicherheitsanforderungen an Services und Lösungen“ [BSI, 2008b, S.15].

Durch Verwendung offener Standards, einfacher Lesbarkeit (XML) und Zustandslosigkeit der Kommunikation ergeben sich dementsprechend einfache Möglichkeiten zur Manipulation durch Angreifer. SOAP-Nachrichten eines Konsumenten dringen, ungehindert von konventionellen Firewalls und den damit verbundenen Filtermethoden, unmittelbar bis zum Web-Service vor. Der klassische Ansatz von Sicherheit, die Ränder des Informationssystems zu schützen greift zu kurz, da es keine klar definierten Ränder mehr gibt. Somit sind weitere Maßnahmen erforderlich, um die Sicherheitsziele Vertraulichkeit, Integrität, Authentizität, Nicht-Abstreitbarkeit und Verfügbarkeit sicherzustellen. Als Zielobjekte für mögliche Bedrohungen werden neben den eigentlichen transportierten Nutzdaten eines Serviceaufrufs u.a. auch kommunizierte Sicherheitstoken betrachtet (siehe [BSI, 2008b, S.20]).

2.6.1 Angriffsziele

Die Allgegenwärtigkeit globaler Informationsverarbeitung und Datenvernetzung ermöglicht Angriffe auf öffentlich zugängliche Systeme. Ein besonderes Interesse gilt den Informationen und der Verfügbarkeit militärischer und wirtschaftlicher Ziele.

Das Ziel eines Angriffs kann aus einem oder einer Kombination von Angriffszielen zusammengesetzt sein:

Betrug	Absichtliche Störung durch unerlaubte Informationsgewinnung, Manipulation und Unterschlagung von Informationen.
Spionage	Unerlaubte Informationsgewinnung.
Sabotage	Absichtliche Störung von Abläufen durch Manipulation und Unterschlagung von Informationen oder erreichen eines Dienstausfalles.
Dienstausfall	Gezieltes Ausschalten von Diensten. Ein Dienstausfall kann negative Auswirkungen auf Reputation des Dienstes und seines Anbieters bewirken.
Zerstörungslust	Gezieltes Manipulieren oder Unterschlagen von Informationen oder erreichen eines Dienstausfalles.

Beispielsweise kann ein Angreifer durch Spionage an wirtschaftliche oder militärische Informationen gelangen und dadurch einen Vorteil gegenüber dem Geschädigten erlangen.

2.6.2 Sicherheitsziele

Grundsätzliche Sicherheitsanforderungen nach [BSI, 2008b, Kapitel 3.1]:

„Der Begriff Sicherheit verfügt im allgemeinen Sprachgebrauch über drei Bedeutungen. Er wird sowohl im Sinne von „Schutz“ (vor Gefährdungen) als auch „Gewissheit“ (hinsichtlich einer erhobenen Vermutung) und „Zuverlässigkeit“ (hinsichtlich eines erwünschten oder erwarteten Verhaltens) interpretiert.“ [Melzer u. a., 2007, S.188f]

Bei der Übertragung dieser Teilgebiete auf Computeranwendungen, kristallisieren sich verschiedene Sicherheitsanforderungen heraus:

Authentifizierung identifiziert Subjekte[12] durch Credentials im System. Ein Credential ist ein Identifikations- und Berechtigungsnachweis, der eine Zugriffserlaubnis auf bestimmte Informationen, Ressourcen oder Auktionen nachweist. Häufig besteht ein Credential aus Benutzername und einem dazugehörigen Passwort. Bei erhöhten Sicherheitsanforderungen können alternative Credential, z.B. auf Basis von biometrische Merkmale oder Zertifikate eingesetzt werden.

Autorisierung gewährleistet, dass ein (dem System bekanntes) Subjekt berechtigt ist auf die angeforderten Informationen und Ressourcen zuzugreifen. Typischerweise geschieht die Umsetzung unter Verwendung von Rollen und Gruppen. Bei jedem Zugriff sollte geprüft werden, ob die Berechtigung den Zugriff erlaubt.

[12] Das kann ein Anwender oder ein anderer Service sein.

Integrität bezeichnet die Vertrauenswürdigkeit von Informationen und Ressourcen. Zu unterscheiden sind Datenintegrität und Herkunftsintegrität. Die Datenintegrität soll gewährleisten, dass Daten nicht unbemerkt kompromittiert werden können. Die Herkunftstintegrität soll garantieren, dass der Absender der Information korrekt ist und die Absenderinformationen nicht gefälscht wurden.

Vertraulichkeit gewährleistet Geheimhaltung sensibler Daten durch Verwendung von kryptographischen Verschlüsselungsverfahren. Geschütze Informationen und ihre Daten dürfen nur durch authentifizierte und autorisierte Subjekte gelesen, modifiziert und gelöscht werden.

Verbindlichkeit gewährleiste die Nichtabstreitbarkeit von Aktionen die von einem bestimmten Subjekt ausgeführt werden. Zudem sollte gewährleistet sein, dass das Subjekt eine Bestätigung für das Ausführen der jeweiligen Aktion erhält.

Verfügbarkeit fordert Sicherstellung der Verfügbarkeit benötigter Informationen und Ressourcen. Durch einen Angriff auf die Verfügbarkeit eines Systems kann der Zugriff auf benötigte Informationen nicht mehr gewährleistet werden.

2.6.3 Allgemeine Bedrohungspotentiale

Im Folgenden werden typische Bedrohungen vorgestellt (siehe [BSI, 2008b, Kapitel 3.2.1]:

Schadsoftware sind Computerprogramme, die die Informationssicherheit gefährden. Die potentiellen Risiken reichen von Löschen bzw. Modifizieren einzelner Dokumente bis hin zum gezielten Ausspähen. Die Schadsoftware wird in der Regel ohne Wissen des Anwenders ausgeführt. Grundsätzlich können Schadsoftwareattacken auch in Services „verpackt“ werden.

Buffer Overflows führen bei ungültigen Eingaben von Daten zu unvorhersehbaren Verhalten des Computerprogramms. Die Häufigste Form ist der Absturz des Computerprogramms. Eine weitere Möglichkeit ist das Ausführen von eigenem Programmcode, der in den Daten enthalten ist - unter Umständen kann der Angreifer die komplette Kontrolle über das Computersystem übernehmen. Web-Services unterliegen ebenfalls dieser Gefahr und müssen ebenso vor falscher Parametereingabe geschützt werden wie herkömmliche Anwendungen.

Hintertüren sind direkte Zugangswege zu Computersystemen, die von Herstellern von Computerprogrammen in ihre Produkte integriert werden. Diese Zugangswege entziehen sich der Zugriffskontrolle des Betreibers des Computersystems - im allgemeinen ist ihre Existenz nicht bekannt. Das Bedrohungspotential wächst durch die Vielzahl von Systemen die an einem Geschäftsprozess beteiligt sind. Durch dynamisches Binden

von Services aus einem Service-Repository kann es vorkommen, dass die Herkunft eines angebotenen Services nicht vollständig geklärt ist und somit potentiell gefährliche Services in den Geschäftsprozess integriert werden.

Netzangriffe nutzen schwächen eines Protokolls zur Kommunikation zwischen Computern aus, um Informationen oder Zugangskontrollsysteme zu umgehen. Zu unterscheiden sind „Scanning“, „Sniffing“, „Spoofing“ und „Man-in-the-Middle-Attacken“. Durch Scanning werden Dienste und Schwachstellen auf einem Computersystem identifiziert und ausgenutzt. Sniffing bezeichnet das Belauschen von Datenverkehr in einem Computernetzwerk - es ermöglicht dem Angreifer Zugang zu vertraulichen Daten sowie Berechtigungsinformationen für Zugangskontrollsysteme zu erhalten. Unter Spoofing versteht man das Vortäuschen der Identität eines Subjektes als Absender einer Nachricht, um Zugang zu System oder Daten zu erhalten, für die ansonsten keine Zugangsberechtigung vorliegt. Bei Man-in-the-Middle schaltet sich ein Angreifer zwischen zwei Kommunikationspunkte und täuscht jedem Endpunkt die Identität der ursprünglichen Gegenseite der Kommunikation vor - er kann Nachrichten mitlesen oder auch modifizieren. Im Web-Service Umfeld werden Angriffe, bei denen XML-Nachrichten modifiziert weiterverwendet werden, als „XML-Rewrite“-Angriffe bezeichnet. Da die Web-Services Kommunikation auf einem Klartext XML-Nachrichtenformat aufsetzt, dass herkömmliche Protokolle wie HTTP zum Transport von Daten verwendet, sind dessen zusätzlichen Gefahren ebenfalls zu betrachten.

Denial of Service bezeichnet Angriffe auf die Verfügbarkeit von Computersystem. Ziel ist es eine Überlastung des Computersystems zu erreichen, so dass berechtigte Subjekte ihre Aktivität nicht mehr oder stark eingeschränkt durchführen können. Web-Services verwenden meist HTTP zum Transport von Daten. Hierdurch werden die DoS Angriffe auf das HTTP Protokoll ebenfalls für Web-Services relevant. Die Angriffe auf HTTP Ebene sind nicht Gegenstand der Arbeit. In Kapitel 4 werden Web-Service spezifische Denial of Service Attacken tiefergehend behandelt.

Passwortcracking beschäftigt sich mit Methoden zur Rekonstruktion von Passwörtern für Computerprogramme oder Computersystemen, mit denen der Angreifer Zugang erlangen kann. Zur Erhöhung der Sicherheit werden Passwörter meist nicht im Klartext gespeichert, so das ein Angreifer versucht Passwörter zu raten, gegen Wörter eines Wörterbuches zu prüfen oder einfach alle Zeichenkombinationen systematisch durchzuprobieren. Auch effizientere mathematische Kryptoanalysen sind u.U. möglich. Die Angriffsmöglichkeiten sind vielfältig, muss sich doch jeder Service Aufruf in irgendeiner Weise authentisieren. Werden lediglich Credentials verwendet, die nur aus Benutzername und Passwort bestehen, ist ein Nachrichtenbasierten System an jedem Punkt angreifbar.

Kryptoanalyse bezeichnet Angriffe gegen kryptographische Verfahren. Ziel ist es hierbei, durch Schwächen des Kryptosystems sowohl Verschlüsselungsverfahren zu brechen,

als auch digitale Signaturen fälschen zu können. In einem internationalen Umfeld kann es passieren, dass durch unterschiedliche rechtliche Rahmenbestimmungen nicht immer die optimalen Algorithmen verwendet werden können, was sich negativ auf die Sicherheit der Web-Services auswirkt. Ein Algorithmus der gebrochen wurde, muss schnellstmöglich im gesamten System umgestellt werden, um die Sicherheit des gesamten Prozesses zu gewährleisten.

Neben diesen genannten Angriffen, gibt es auch nicht-technischen Angriffe, wie z.B. Social Hacking, durch die Informationen oder Computersysteme kompromittiert werden könnten. Da es bei Web-Services in erster Line um Maschine-zu-Maschine Kommunikation handelt, soll dies an dieser Stelle vernachlässigt werden.

2.6.4 Web-Service spezifische Bedrohungspotentiale

Die allgemeinen Bedrohungspotentiale gelten ebenfalls für Web-Services. Sie werden um weitere Web-Service spezifischen Bedrohungspotentiale erweitert (siehe [BSI, 2008b, Kapitel 3.2.2])

Replay-Attacks

Web-Services und ihre Nachrichten sind per se zustandslos. Diese Eigenschaft erlaubt einem Angreifer eine legitime Handlung auch dann zu wiederholen, wenn die Nachricht durch Verschlüsselung und Signatur geschützt ist. Die an einem Gesamtprozess beteiligten Service sind häufig zustandslos, sie merken daher nicht welche Nachrichten bereits bearbeitet wurden und würden somit auch eine wiedereingespielte Nachricht bearbeiten. Legitime Nachrichten können, ohne eine Verletzung von einzelnen oder zusammengesetzten Schutzzielen aus Authentifizierung, Autorisierung, Integrität und Vertraulichkeit, wiedereingespielt werden, da sie die benötigten Legitimation bereits beinhalten. Möglich sind Attacken auf die Verbindlichkeit von Handlungen oder Attacken auf die Verfügbarkeit von Services. Aus diesen Angriffen können wirtschaftliche Schäden entstehen, wenn die Verwendung des Services für das Opfer pro Anfrage tarifiert ist. Am Beispiel eines „Versand“-Services könnte ein Angreifer die Nachricht auf dem Weg zur Versandabteilung eines Buchhandels abfangen und diese zu einem späteren Zeitpunkt erneut versenden oder kostenpflichtige externe Services („Kreditkarten Validierung“) mehrfach aufrufen. Unter Umständen ist ein Angriff auf die Verfügbarkeit durch vielfaches Wiedereinspielen von legitimen Nachrichten möglich.

XML-Spezifische Angriffe

Das Parsen eines XML-Dokuments kann aufgrund einer hoher Komplexität der Baumstruktur sehr ressourcenlastig sein. Unterschiedliche Parser eigenen sich daher für unterschiedliche Zugriffsverfahren unterschiedlich gut. Während DOM-Parser den wahlfreien Zugriff auf XML-Elemente effizient implementieren, indem sie den gesamten XML-Dokumentenbaum im Arbeitsspeicher vorhalten, erlauben SAX-Parser einen seriellen Zugriff auf den XML-Datenstrom (vgl. [BSI, 2008b, S.22f] und [Zedlitz, 2006, S.7]).

Durch die Verwendung von SOAP- bzw. XML basierten Nachrichten ergeben sich eine Reihe mögliche Angriffe, die mit den Eigenheiten von XML eng verbunden sind und in erster Line Angriffe auf den XML Parser und seine Implementierung darstellen.

WSDL- und Service-Scannning

Durch UDDI-Verzeichnisdienste sind WSDL-Beschreibungen eines Web-Services öffentlich verfügbar. Durch die WSDL-Beschreibung erfährt ein Angreifer Details und Parameter zum Aufruf eines Web-Services. Auf Basis der Informationen aus der WSDL-Beschreibung kann ein Angreifer diesen Dienst gezielt auf Schwachstellen untersuchen. Er kann anhand der Beschreibung versuchen Service-Namen von nicht veröffentlichten internen Services zu erraten. Sollten die für den internen Gebrauch vorgesehenen Services schwächere Schutzmerkmale aufweisen, dienen sie als möglicher Angriffspunkt für weitere Angriffe. Eine gezielte Manipulation von Parametern kann zu Fehlermeldungen führen, die Auskunft über Interna des Services preisgeben oder den Service in einen undefinierten Fehlerzustand versetzen.

Kompromitieren eines Services

Services die an einem Geschäftsprozess beteiligt sind können oft über Unternehmensgrenzen hinaus verteilt sein. Durch Manipulation eines beteiligten Services oder der Platzierung eines manipulierten Services im Service Repository können andere Services oder ganze Geschäftsprozesse manipuliert oder unterbunden werden. Dieses Risiko lässt sich reduzieren, indem die Authenzität und die Vertrauenswürdigkeit des Service-Anbieters geprüft werden.

Unberechtigte Servicenutzung

Subjekte die einen Service benutzen müssen authentifiziert werden. Die Mechanismen zum Austausch von Subjekten und Authentifizierungsinformationen muss ggf. über Organisationsgrenzen hinaus geschützt werden, so dass ein Angreifer diese nicht manipulieren bzw.

unterbinden kann. Da ein Service einen Bestandteil eines Geschäftsprozesses abbildet, muss die Benutzung durch Subjekte die diesen Service verwenden, geregelt werden.

2.6.5 Web-Service Sicherheitsstandards

Bisherige Sicherheitsmaßnahmen setzten meist auf Transport-Ebene an. Eine Firewall auf Transport-Ebene filtert Pakete anhand der Port Nummer über den eine Nachricht empfangen oder gesendet wird. Dabei kann ein Port entweder offen oder geschlossen sein. Meist findet die Kommunikation bei Web-Services mit SOAP über HTTP statt und ist für die Firewall nicht von wohlgeformten HTML-Dokumenten zu unterscheiden. Die Nachrichten können einen offenen Port ohne Inspektion passieren. Dabei behandelt die Firewall alle eintreffenden Nachrichten identisch, entweder werden alle durchgelassen oder keine (vgl. [BSI, 2008b, S.28f]).

Durch Einsatz von SSL und seinem Nachfolger TLS kann ein Sicherheitskontext zwischen einzelnen Punkten hergestellt werden (siehe Abbildung 2.7). Die Abbildung 2.8 zeigt einen Sicherheitskontext der echte Ende-zu-Ende Sicherheit bietet, wie sie bei einer Komposition von Services benötigt wird. Sobald die Nachricht über einen oder mehr Intermediary-Knoten zu seinem Empfänger gesendet wird, ist der Einsatz von SSL nicht ausreichend um eine Ende-zu-Ende Sicherheit zu bieten. Ein Schutz der Nachricht findet nur auf dem Transportweg statt und endet mit Eingang im nächsten Intermediary-Knoten oder Zielsystem. Bei n-Intermediary-Knoten ergeben sich bei einer Punkt-zu-Punkt Sicherheit n+1 Sicherheitskontexte, während bei einer Ende-zu-Ende Sicherheit ein durchgehender Sicherheitskontext gegeben ist.

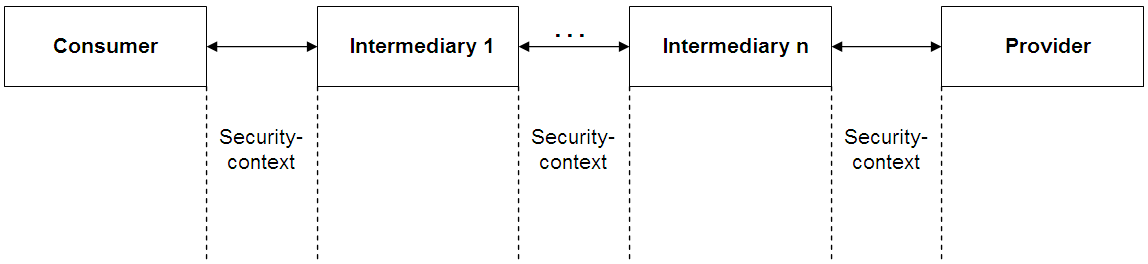

Abbildung 2.7: SSL Punkt-zu-Punkt Sicherheit

Der SSL-Schutz erstreckt sich immer auf die gesamte Nachricht, währen der Schutz auf Nachrichten-Ebene ermöglicht auch einzelne Teile oder Element der Nachricht zu schützen.

Es wird deutlich, dass die Sicherheit auf Transport-Ebene nicht ausreichend ist, um komplexe Serviceumgebungen hinreichend zu schützen. Die schwächen der herkömmlichen

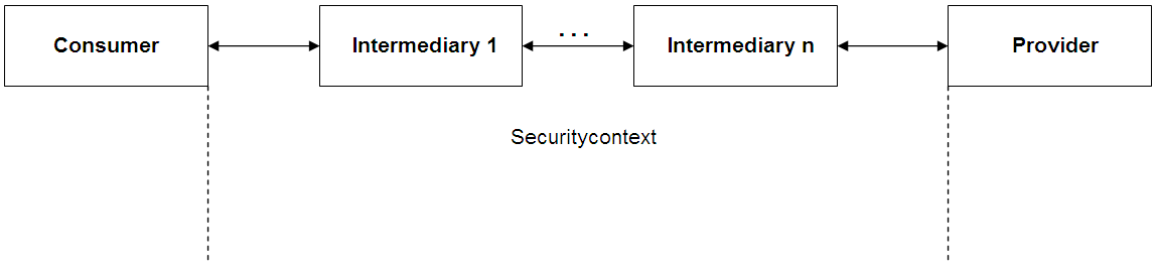

Abbildung 2.8: Web-Services Ende-zu-Ende Sicherheit

Sicherheitsmechanismen können mit dem Einsatz von Nachrichtenbasierten Web-Service Sicherheitsstandards behoben werden.

Tabelle 2.3 zeigt nach [BSI, 2008b, S.31] die wichtigsten Sicherheitsstandards und die zuständigen Gremien für die Spezifikation.

W3C	OASIS
XML Encryption	SAML
XML Signature	WS-Security
XML Key Management Specification	- WS-Trust
	- WS-Policy
	- WS-Federation

Tabelle 2.3: Übersicht der wichtigsten Web-Service Sicherheitsstandards

Das W3C kümmert sich um die Standardisierung XML spezifischer Sicherheitsstandards, während die *Organization for the Advancement of Structured Information Standards*[13] (OASIS) die weiterreichenden Sicherheitsstandards spezifiziert. Die Sicherheitsstandards definieren Metadaten zur Sicherung von ganzen Nachrichten oder einzelner Bestandteile.

Im dem folgendenen Abschnitt sollen die verbreiteten und etablierten Sicherheitsstandards die für die Arbeit relevant sind kurz vorgestellt werden (vgl. [BSI, 2008b, Kapitel 4.3]). Die Standards XML-Key Management specification, Security Assertion Markup Language, WS-Policy und WS-Federation sind für das Verständnis der Arbeit nicht zwingend notwendig und werden daher an dieser Stelle nicht weiter erwähnt. Sie werden im Anhang B vorgestellt.

XML-Encryption

Diese Spezifikation legt fest, wie und in welcher Form XML-Dokumente innerhalb der XML-Syntax verschlüsselt werden. Die XML-Encryption gewährleistet Vertraulichkeit. Die Ver-

[13] http://www.oasis-open.org

schlüsselung kann angewandt werden auf das gesamte XML-Dokument, auf einzelne Elemente (und ihren Unterelemente) und auf Inhalte eines XML-Elements. Verschiedene Teile eines Dokuments können mit unterschiedlichen Schlüsseln verschlüsselt werden, so das die Vertraulichkeit gewährleistet ist, dass nur bestimmte Empfänger den für sie vorgesehenen Abschnitt lesen können. Die verschlüsselten Informationen bleiben nach der Transformation wohlgeformte XML-Dokumente, die allerdings nur im Falle einer Verschlüsselung von Elementeninhalten weiterhin valide gemäß des verwendeten XML-Schemas bleiben.(vgl. BSI [2008b] und [Melzer u. a., 2007, S.204])

XML-Signature

Die XML-Digital-Signature Spezifikation legt die Regeln und die Syntax fest, mit denen XML-Signaturen erzeugt werden. XML-Signature gewährleistet Integrität und Verbindlichkeit [14]. Analog zu XML-Encryption erlaubt sie das Signieren ganzer XML-Dokumente oder einzelner Teile. Durch das Signieren auf Element-Ebene, können Teile der Nachricht signiert werden, während andere Teile variabel bleiben. Die Signatur behält weiterhin ihre Gültigkeit, wenn nicht Signierte Teile der Nachricht geändert werden. Einzelne Teile des XML-Dokuments können mit unterschiedlichen (Signatur-) Schlüsseln signiert werden. Die digitalen Signaturen werden persistent in das XML-Dokument integriert und sind somit dauerhaft verifizierbar. XML-Signature adressiert die Integrität und Verbindlichkeit von XML-Daten. (vgl. BSI [2008b] und [Melzer u. a., 2007, S.204])

WS-Security

WS-Security ist eine OASIS Spezifikation die in Zusammenarbeit von IBM, Microsoft und Verisign entstanden ist. WS-Security bildet ein Rahmenwerk, dass sowohl auf vorhandene Basis-Security-Technolgien, wie XML-Encryption, XML-Signature und XML-Key Management aufsetzt, als auch Raum für Erweiterungen bietet, die spezielle Anforderungen abdecken. Unter Verwendung der Basis-Security-Technolgie gewährleistet WS-Security Integration, Vertraulichkeit und Verteilung der sicherheitsrelevanten Informationen. Sie beschreibt wie diese Security-Technologien mit dem Web-Service Standardformat SOAP integriert werden können. Dabei werden Regeln und Standards definiert, wie und in welcher Form entsprechende Angaben in den Header von SOAP-Nachrichten eingebettet werden (vgl. BSI [2008b] und [Melzer u. a., 2007, S.209]).

[14] Authenzität der verbindlichen, glaubwürdigen Urheberschaft von Nachrichten.

WS-Trust

WS-Trust ist Eine Erweiterung des WS-Security Standards, der SOAP-basierte Mechanismen für das Ausstellen, Erneuern und Bestätigen von Sicherheitstoken ermöglicht. Mit WS-Trust kann die angefragte Stelle zudem Anforderungen für die Ausgabe von Sicherheitstoken (beispielsweise X.590 Zertifikate, Kerberos-Tickets oder BenutzerID) stellen. Voraussetzung ist daher, dass das Subjekt nicht gänzlich unbekannt ist. WS-Trust dienst in erster Linie dazu, sichere Beziehungen zu etablieren, zu vermitteln und zu beurteilen. Es dient ebenso dem Mapping von verschiedenen Credentials für unterschiedliche Applikationen und Zonen. WS-Trust kann beispielweise bei Erhalt eines X.509 Zertifikats einen Kerberos-Token desselben Benutzers liefern. Ziel ist das Sicherstellung der Interoperabilität bei der Nutzung von Sicherheitsrelevanten Daten (vgl. [Melzer u. a., 2007, S.211]).

WS-SecureConversation

WS-SecureConverstation gibt einen Rahmen auf der SOAP-Schicht vor, der das Erzeugen eines Sicherheitskontext nach erfolgreichen Authentifizierung ermöglicht. Teile von WS-Trust werden erweitert, um Sicherheitstokens erzeugen und verteilen zu können. Das Sicherheitstoken enthält einen Sitzungsschlüssel mit dem die Nachrichten innerhalb des Sicherheitskontexts gemäß der XML-Encryption Spezifikation verschlüsselt werden. WS-SecureConverstation adressiert die Vertraulichkeit und Verbindlichkeit von Nachrichten. Nach Beendigung des Sicherheitskontext, lassen sich abgefangene oder abgehörte Nachrichten zu einem späteren Zeitpunkt nicht wiedereinspielen (vgl. [Melzer u. a., 2007, S.211]).

2.6.6 Web-Service-Firewalls und -Gateways

Die Web-Service Sicherheitsstandards bieten eine Grundlage zur Durchsetzung der Schutzziele auf Nachrichten-Ebene. Darüber hinaus gibt es weitere nicht standardisierte Sicherheitsmaßnahmen, die einen zusätzlichen Schutz für unzureichende geschützte Programmierung von Web-Service Anwendungen sowie die Web-Service Implementierung des Applikationsservers selbst bieten. Zu dieser Gattung gehören Web-Service- bzw. XML-Firewalls und -Gateways. Sie bilden die Funktionalität einer gängigen Firewall auf den XML-Nachrichtenaustausch ab, indem sie stellvertretend (als Proxy) für den Web-Service die Nachrichten entgegennehmen und inhaltsbasiert gegen ihr Regelwerk validieren. Die unschädlichen oder gefilterten Nachrichten werden dann an den Web-Service weitergeleitet. Diese Systeme befinden sich im Normalfall zwischen der Unternehmens-Firewall und den

internen Systemen. Während XML-Firewalls auf das Abblocken von invaliden oder schädlichen XML-Dokumenten konzentriert sind, können XML-Gateways weitere Mehrwerte bieten, wie z.B. Transformationen von Nachrichten. Unter anderem bieten diese Systeme Schutz gegen SQL-Injections, Brute-Force und Denial-of-Service Angriffen (vgl. [BSI, 2008b, S.29]).

Durch eine vorgelagerte Sicherheitslogik wird die Entwicklung von Services vereinfacht, da die Logik zur Authentifizierung, Verschlüsselung und Signierung der XML-Nachrichten sowie das durchsetzen von Policies nicht mehr durch den Service selbst durchgeführt werden müssen. Durch diese Trennung der Sicherheitslogik hat der Web-Service mehr freie Ressourcen für die Bearbeitung der Anwendungslogik zur Verfügung. Eine weitere Entlastung stellt das validieren der XML-Nachrichten, insbesondere bei großen Nachrichten, durch den XML-Parser der XML-Firewall dar. Fehlerhafte XML-Nachrichten erreichen den Web-Service nicht (vgl. [Smith und Franke, 2003, S.11]).

Die XML-Firewalls und -Gateways können als zentrale Komponente im System bereitgestellt werden, welche alle eingehenden Nachrichten überprüft. Die andere Möglichkeit ist, vor jeden Web-Service des Unternehmens eine eigene XML-Firewall zu stellen. Beim ersten Ansatz ist zu beachten, dass die Skalierbarkeit den Anforderungen des Unternehmen gewachsen sein muss, um nicht zu einem Flaschenhals zu werden. Bei einem solchen Konzept würde der Sicherheitskontext der Ende-zu-Ende Sicherheit bei der XML-Firewall enden - interne Kommunikation wäre ungeschützt. Der zweite Ansatz wäre sehr kostenintensiv und bei Fehlen einer zentralen Administration der Systeme aufwendiger zu Verwalten (vgl. [BSI, 2008b, S.29]).

Die Sicherheitsfunktionen der verwendeten XML-Firewall wird in Anhang C vorgestellt.

2.7 Zusammenfassung

Web-Services stellen die verbreitetste Realisierung einer Serviceorientierten Architektur dar. Zu ihrer Verbreitung haben neben der losen Kopplung und Interoperabilität die weitreichende Akzeptanz der Web-Service Technologien, die auf offenen Standards und Spezifikationen beruhen, beigetragen. Services bilden in sich geschlossene Dienste an, die über ihre WSDL-Schnittstellenbeschreibung angesprochen werden können. Der Datenaustausch der XML-Nachrichten findet in der Regel per SOAP über HTTP statt und ist daher von gewöhnlichen Firewalls nicht von reinem HTTP-Nachrichten zu unterscheiden. Entweder lässt die Firewall alle Nachrichten auf einem Port durch oder keine.

Mit Hilfe von UDDI und WSDL können Web-Services und eine Beschreibung ihrer Verwendung öffentlich gefunden werden. Die Informationen aus der WSDL-Beschreibung stellen einen ersten Einstiegspunkt für Angriffe auf den Web-Service dar. Durch Service-Aufrufe mit manipulierten Parametern können Fehlermeldungen provoziert werden, die unter Umständen Auskunft über Interna des Web-Services und lokaler Ressourcen liefern können, die als Basis für weitere Angriffe genutzt werden können.

SOAP-Nachrichten bestehen aus Klartext XML-Dokumenten. Die SOAP Spezifikation selber sieht keine direkten Sicherheitsmechanismen vor, vielmehr delegiert sie die Realisierung von Sicherheitsmerkmalen an die Transport- und Anwendungsschicht.

Unter Verwendung von SSL (oder seinem Nachfolger TLS) ist eine Punkt-zu-Punkt Verschlüsselung der SOAP-Nachrichten über HTTP auf der Transportschicht realisierbar. Sie gewährleisten einen Sicherheitskontext während der Datenübermittlung der mit Eintreffen der Nachricht beim Empfänger endet. Der Schutz auf der Transportschicht ist unzureichend, sobald Nachrichten über Intermediary-Knoten laufen. Ein geschlossener Sicherheitskontext, der seine Gültigkeit zwischen Sender, Intermediary und Empfänger behält, kann durch SSL nicht abgebildet werden. Ein Sicherheitskontext, der eine Ende-zu-Ende Sicherheit gewährleistet, kann unter Verwendung von Sicherheitsstandards auf Nachrichten-Ebene realisiert werden. Die Standards adressieren Authentifizierung und Autorisierung, Vertraulichkeit unter Einsatz von Verschlüsselung sowie Verbindlichkeit von Handlungen unter Einsatz von Signaturen. Die Integrität kann unter Einsatz von Verschlüsselung oder Signaturen gewährleistet werden.

Um die Sicherheitsanforderungen einer komplexen Serviceumgebung zu realisieren, werden zusätzliche Mechanismen benötigt, die einen Schutz vor Angriffen auf unzureichend geschützte Programmierung von Web-Service Anwendungen sowie die Web-Service Implementierung des Applikationsservers selbst bieten. XML-Firewalls bzw. -Gateways sind in der Lage XML-Nachrichten auf invalide und schadhafte Bestandteile zu Untersuchen und Nachrichten zu verwerfen oder zu filtern. Solche Systeme können neben der Validierung von XML-Nachrichten die Aufgaben zur Autorisierung, Verschlüsselung und Signierung von

XML-Nachrichten und ihren Bestandteilen übernehmen und den Web-Service um diese Aufgaben entlasten.

Das folgende Kapitel beschäftigt sich mit der Konzeption von Angriffen auf Web-Services. Die Angriffe erfolgen durch manipulierte SOAP-Nachrichten. Neben einer Unterteilung der Angriffe in klare Angriffskategorien werden Definition und Auswirkung der Angriffe beschrieben. Die Konzeption und Realisierung von Angriffen beschränkt sich auf Angriffe, die einen direkten Angriff auf die Web-Service Implementation des Applikationsservers und den beteiligten Spezifikationen darstellen.

3 Konzeption

In diesem Kapitel werden Angriffe auf Web-Services behandelt. Die identifizierten Angriffstechniken werden definiert, charakterisiert, ihre Auswirkung dargestellt und Gegenmaßnahmen vorgestellt.

Die Angriffstechniken werden in Klassen gemäß ihrer Bedrohungspotentiale unterteilt. Angriffstechniken die implementationsspezifische Eigenschaften des Applikationsservers nutzen - gemäß der Implementierung eines Web-Service Standards - werden gesondert behandelt und stehen im Mittelpunkt dieser Arbeit. Sie bilden die Basis für die konzeptionellen Angriffe.

Der Versuchsaufbau geht näher auf die Versuchsumgebung und die Durchführung ein. Im Anschluss wird der exemplarische Web-Service vorgestellt, der als Angriffsfläche für die konzipierten Angriffe dient. Anschließend werden konzeptionelle Angriffe behandelt.

3.1 Abgrenzung

Web-Service Spezifikationen enthalten Definitionen, die aufgrund von Zukunftssicherheit oder Unterspezifizierung Interpretationsspielräume aufweisen. Eine andere Möglichkeit stellen fehlerhafte Bestandteile einer Definition dar, beispielsweise erlaubt der Web-Service-Security Header eine unbegrenzte Anzahl unspezifizierter Elemente zu verwenden. Demnach kann ein System sicherheitskritische Schwachstellen enthalten, selbst wenn das System frei von Implementierungsfehlern ist (vgl. [Zedlitz, 2006, S.5]).

Nach Zedlitz [2006] können die Angriffspunkte auf Web-Services in diese vier Kategorien unterteilt werden:

- HTTP,
- XML,
- SOAP und
- Applikations-Ebene.

Die folgenden Angriffstechniken beschränken sich auf die höheren Schichten des Web-Service Layers: XML, SOAP und die darüber liegende Applikations-Ebene. Angriffe auf Protokolle die sich auf Ebene des Web-Service Transport-Layers (HTTP) und darunter befinden, sind nicht Gegenstand der Arbeit.

3.1.1 Angriffsklassen

In der Literatur findet sich des öfteren eine recht feine Klassifizierung der Angriffstechniken. Im Kontext der Angriffe auf Web-Services lassen sich einzelne Angriffstechniken nicht immer klar einer einzelnen granularen Klasse zuordnen. In dieser Arbeit werden die Angriffsklassen ihrer Angriffsziele entsprechend unterteilt. Angriffstechniken die auf Daten-Integrität und Vertraulichkeit abzielen, werden aufgrund ähnlicher Vorgehensweisen zu einer Klasse zusammengefasst.

In dieser Arbeit werden die folgenden Angriffsklassen unterschieden:

Berechtigung	Angriffe bei denen die Identität eines legitimen Subjektes missbraucht wird.
Daten-Integrität und Vertraulichkeit	Angriffe bei denen unbefugte Informationsgewinnung und Modifikation im Mittelpunkt stehen.
Systemumgebung	Angriffe auf das Verhalten der Systemumgebung.
Verfügbarkeit	Angriffe auf die Erreichbarkeit eines Web-Services. Kompromittierung des Applikationsservers oder der Anwendungslogik durch XML-spezifische Eigenschaften von XML-Nachrichten: XDoS.

Die Angriffsklasse „Systemumgebung“ beinhaltet Angriffstechniken die das Ziel verfolgen Schäden an der Systemumgebung zu verursachen. Die übrigen Angriffsklassen beinhalten Angriffstechniken die das Ziel verfolgen die Daten und Informationen des Web-Services und seinen Backend-Systemen sowie den Web-Service selbst zu kompromittieren.

Eine Unterscheidung zwischen Angriffen aus dem internen oder externen Netzwerk findet im Rahmen der Arbeit nicht statt. Vielmehr ist Entscheidend, ob sich der Web-Service bei dem Angriff hinter eine XML-Firewall befindet, oder nicht.

Die im Rahmen der Arbeit identifizierten Angriffstechniken erheben keinen Anspruch auf Vollständigkeit. Die Spezifikation SOAP, WSDL, WS-Security, XML-Signature, XML-Encryption werden näher untersucht. Es kann nicht ausgeschlossen werden, dass andere Web-Service Spezifikationen zusätzliche Angriffsmöglichkeiten bieten.

Ein Großteil der Literatur zu Angriffstechniken stammt aus dem Englischen. Zum Zweck der Eindeutigkeit werden in der Arbeit die englischen Namen übernommen.

3.2 Angriffe auf die Berechtigung

Ziel dieser Angriffe ist es eine unautorisierte Berechtigung zu einem Web-Service zu erhalten. Die Palette an Angriffen reicht vom einfachen ausprobieren von möglichen Passwörtern, über unautorisierten Übernahmen von Sessions, bis hin zum wiederverwenden von gestohlenen Credentials. Unter Verwendung der Berechtigungen des Opfers lassen sich weitere Angriffe innerhalb des kompromittierten Systems ausführen.

3.2.1 Dictionary Attack

Bei diesem Angriff versucht der Angreifer durch systematisches ausprobieren möglicher Kombinationen von Benutzername und Passwörtern die Authentifizierungsmechanismen eines Computersystems zu umgehen.

Ist der Benutzername beispielsweise durch Informationen des *UserNameToken* eines WS-Security Headers bekannt, wird der Angriff effektiver - die Suche beschränkt sich auf das ausprobieren aller möglichen Passwörter. Bei einer Wörterbuch-Attacke werden die Wörter des Wörterbuches (die mit hoher Wahrscheinlichkeit zu einem Erfolg führen) der Reihe nach getestet, bis das richtige Passwort gefunden wurde (vgl. [OASIS, 2006a, Kapitel 5], [Actional, 2004, S.14] , [Moradian und Hakansson, 2006, S.159] und [BSI, 2008b, S.22]).

Der Angriff lässt sich in abgewandelter Form als Brute-Force Angriff durch systematisches erzeugen aller möglichen Zeichenkombinationen anstelle von Wörtern durchführen. Bei Brute-Force-Angriffen ist der Ereignisraum der möglichen Zeichenkombinationen wesentlich komplexer und ist daher bei langen Passwörtern nicht in angemessener Zeit durchführbar (vgl. [OASIS, 2006a, Kapitel 5] und [BSI, 2008b, S.22]).

Nach Moradian und Hakansson [2006] sind die meisten Passwort-basierten Authentifikations-Algorithmen anfällig für Wörterbuch-Attacken. Erfolgreiche Angriffe basieren auf der Verwendung schwacher Passwörter, z.B. durch ein Passwort mit weniger als 8 Zeichen länge oder die Verwendung von einzelnen Wörtern die in Wörterbüchern zu finden sind, oder die leicht hervorsagbare Variante bei der eine Zahl vor oder hinter das Wort gehangen wird (vgl. [Moradian und Hakansson, 2006, S.159] und [Actional, 2004, S.14]).

Der WS-Security Standard selber sieht keine Begrenzung der Authentifizierungsversuche vor. Es liegt in der Verantwortung der Anwendungslogik.

Der Angreifer versucht Berechtigungen zu erlangen, um unautorisiert privilegierte Aktionen mit den Rechten des Benutzers im System auszuführen.

Gegenmaßnahme: Das Problem kann behoben werden durch Einsatz von digitalen Zertifikaten zur Authentifizierung. Wird eine Passwort-Authentifizierung benötigt, führt eine Beschränkung der Authentifizierungsversuche zu einer Abmilderung des Problems.

3.2.2 Session-Hijacking

Bei dieser Angriffstechnik findet eine unautorisiert Übernahme einer aktiven Session zwischen Konsument und Service-Anbieter statt. Voraussetzung für ein Session-Hijacking ist, dass ein Service-Anbieter im Zuge einer Session, die Berechtigungen eines Subjekts serverseitig speichert und diese durch einen SessionToken identifiziert. Bei einem Angriff übernimmt der Angreifer die Rollen des ursprünglich beteiligten Subjekts.

Der Angreifer verwendet einen vorher mitgeschnittenen und identifizierten *SessionToken* innerhalb einer neuen XML-Nachricht, der schadhafte Bestandteile enthält. War das Opfer vor der Übernahme der Session im Zielsystem bereits Authentifiziert, bleiben dessen Autorisierung im Rahmen des Session-Hijackings bestehen und erlaubt dem Angreifer alle Aktionen am Zielsystem auszuführen zu denen das Opfer autorisiert ist (vgl. [Moradian und Hakansson, 2006, S.159]).

Ziel dieses Angriffes ist das Erlangen von Berechtigung, um unautorisiert privilegierte Aktionen mit den Rechten des Benutzers im System auszuführen.

Gegenmaßnahme: Eine Kombination von SessionToken und Mechanismen zur Gewährleistung der Verbindlichkeit des Subjektes lösen das Problem.

3.2.3 Replay Attack

Für eine Replay Attacke schneidet der Angreifer legitime XML-Nachrichten zwischen einem Konsumenten und einem Web-Service mit, um diese zu einem späteren Zeitpunkt wiederholt einzuspielen und die darin enthaltenen legitimen Handlungen zu wiederholen.

Die lose Kopplung zwischen Web-Service und Konsument hat zur Folge, dass ein Web-Service wiederholt eintreffende XML-Nachrichten nicht von neuen XML-Nachrichten unterscheiden kann. Beide XML-Nachrichten werden wie eine neue Nachricht behandelt.

Eine XML-Nachricht die signierte oder verschlüsselte Bestandteile auf Nachrichten-Ebene enthält, kann ebenfalls wiederholt werden, ohne dabei die Schutzziele der Berechtigung, Vertraulichkeit oder Daten-Integrität zu verletzen, solange das enthaltene Credential seine Gültigkeit behält.

Der Ressourcenverbrauch steigt bei Nachrichten mit verschlüsselten oder signierten Elementen enorm. Stellt das Entschlüsseln oder Signieren doch einen aufwendigen Prozess dar, der bei jedem wiederholten Eintreffen der Nachricht erneut stattfindet. Nach [O'Neill, 2005, S.132] ist ein Replay Angriff ausgefeilter als ein reiner Flooding-Angriff, da schwächen des Authentifikationsmechanismusses des Zielsystems genutzt werden. In Kombination mit einem Flooding Angriff erhöht sich der Schaden.

Ein Angriff kann zu wirtschaftlichem Schaden führen, wenn die Verwendung des Services für das Opfer pro Anfrage kosten erzeugt. Am Beispiel eines „Versand"-Services könnte ein Angreifer die Nachricht auf dem Weg zur Versandabteilung des Buchhandels abfangen und diese zu einem späteren Zeitpunkt erneut versenden oder kostenpflichtige externe Services („Kreditkarten Validierung") mehrfach aufrufen.

Gegenmaßnahme: Auf Nachrichten-Ebene sieht WS-Security vor, den SecurityToken durch eindeutige Identifikationsmerkmale beispielsweise Timestamps zu schützen. WS-SecureConversation und WS-Trust verwenden einen Sitzungsschlüssel um den SecurityToken zu verschlüsseln. Bei reiner Punkt-zu-Punkt Kommunikation kann auch die Verwendung von SSL/TLS auf Transport-Ebene Schutz gegen Wiedereinspielung bieten (siehe [O'Neill, 2005, S.132]).

Nach OASIS [2006a] kennt das *UserNameToken* zwei Mechanismen zur Abwehr von Replay-Attacken. Die Spezifikation sieht ein Nonce Element *<wsse:Nonce>* vor, dass einen eindeutigen *Identifier* spezifiziert bzw. das optionale Element *<wsu:Timestamp/wsu:Created>* das die Erstellung einer *Timestamp* spezifiziert. Der Standard empfiehlt die Verwendung beider Sicherheits-Elemente. Dabei ist das „Alter" der Timestamp zu beachten, als auch ein limitierten Cache für die Nonce-Werte und Timestamps zu verwenden (siehe [OASIS, 2006b, Kapitel 3.1] und [OASIS, 2006a, Kapitel 13]).

Der Angriff ist auch bekannt unter dem Namen Capture-Replay Attack.

3.2.4 Übersicht

Tabelle 3.1 zeigt eine Übersicht der Angriffe auf die Berechtigung.

Angriffstechnik	Ziel	Ort	Schwachstelle
Dictionary Attack	Berechtigung	Anwendungslogik, Benutzer	Schwaches Passwort gewählt. Fehlende Beschränkung der Passworteingabe. Keine Authentifizierung durch digitale Zertifikate.
Session-Hijacking	Berechtigung	Anwendungslogik	Keine oder unzureichende Verwendung von WS-Security bzw. SSL/TLS.
Replay Attack	Verbindlichkeit	Anwendungslogik	Keine oder unzureichende Verwendung von WS-Security bzw. SSL/TLS.

Tabelle 3.1: Übersicht der Angriffe auf die Berechtigung

3.3 Angriffe auf Daten-Integrität und Vertraulichkeit

Die Angriffe dieser Angriffsklasse zielen primär auf die Manipulation von Daten der XML-Nachricht ab, bzw. auf die unautorisierte Gewinnung von vertraulichen Daten. Die Vorgehensweise ist dabei vielfältig.

3.3.1 WSDL Scanning

UDDI-Verzeichnisdienst und WSDL-Beschreibungen eines Web-Services machen Details und Parameter zum Aufruf eines Web-Services und der darunter liegenden Technologien öffentlich verfügbar[1]. Auf Basis dieser Informationen kann ein Angreifer den Dienst gezielt auf Schwachstellen untersuchen.

[1] Oft befindet sich die WSDL unter der URL des ServiceEndpoint mit angehängtem „?“.

Er kann versuchen anhand der Namensgebung der Service-Methoden weitere nicht veröffentlichte interne Service-Methoden zu erraten. Sollten die für den internen Gebrauch vorgesehenen Services schwächere Schutzmerkmale aufweisen, dienen sie als möglicher Angriffspunkt für weitere Angriffe (vgl. [Actional, 2004, S.14] und [Lindstrom, 2004, S.5]).

Gegenmaßnahme: Die öffentliche WSDL-Beschreibung sollte keine unnötige Information enthalten. Das Verwenden von erratenen, nicht veröffentlichten Methodenaufrufen, kann durch eine Prüfung gegen die Methoden der öffentlichen WSDL-Beschreibung geschehen verhindert werden.

Der Angriff ist auch bekannt unter den Namen Service-Scanning und WSDL Enumeration.

3.3.2 Message Sniffing

Diese Art der Man-in-the-Middle-Attacke stellt einen direkten Angriff auf die Vertraulichkeit dar. Der Angreifer schneidet XML-Nachrichten während des Transports auf Netzwerk-Ebene oder auf einem zwischenspeichernden Knoten mit.

Verschlüsselt übertragene XML-Nachrichten, die im Klartext zwischengespeichert werden, schneidet der Angreifer im zwischenspeichernden Knoten mit. XML-Nachrichten die verschlüsselt übertragen und gespeichert werden, können nur durch einen gestohlenen Schlüssel oder Techniken der Kryptoanalyse entschlüsselt werden. Der Angreifer gelangt an die vertrauliche Information, indem er die Elemente einer Klartext-XML-Nachricht auswertet.

Der Angriff ist auch bekannt unter dem Namen Message Snooping.

3.3.3 Routing Detour

Eine Form der Man-in-the-Middle-Attacke, bei der ein Angreifer einen manipulierten Intermediary-Knoten veranlasst XML-Nachrichten mit vertraulichem Inhalt an einen feindlichen Intermediary-Knoten weiterzuleiten. Im Zuge einer Message Sniffing Attacke stiehlt der Angreifer die vertraulichen Daten.

Der feindliche Intermediary-Knoten entfernt die zusätzlichen Routingspuren innerhalb des SOAP- und HTTP-Headers, bevor er die Nachricht an den ursprünglichen Empfänger weiterleitet. Der Empfänger bemerkt den Angriff nicht (vgl. [Actional, 2004, S.15] und [Lindstrom, 2004, S.5]).

Nach Zedlitz hat SOAP-Routing zurzeit keine praktische Relevanz. Daher ist davon auszugehen, dass die entsprechenden Implementierungen noch nicht gut erprobt sind und dort Schwachstellen vorhanden sind (vgl. [Zedlitz, 2006, S.9]).

3.3.4 Malicious Morphing

Bei dieser Form der Man-in-the-Middle-Attacke werden nicht signierte Daten einer XML-Nachricht zur Laufzeit des Transports auf Netzwerk-Ebene manipuliert.

Beispielsweise könnte ein Angreifer die Routing-Informationen, die in einer SOAP-Nachricht eingebettet sind, um feindliche Intermediary-Knoten erweitern (siehe Routing Detour). Ebenfalls möglich wäre es bei einer Bestellung die Empfänger-Adresse zu verändern (vgl. [Actional, 2004, S.15]).

Gegenmaßnahme: Bereits die Verwendung der WS-Security-Basismechanismen XML-Signature und XML-Encryption verhindern diese Art von Angriff.

3.3.5 Message Tampering

Beim Message Tampering werden XML-Nachrichten im Rahmen einer Man-in-the-Middle-Attacke zur Laufzeit einer Anfrage bzw. Antwort während des Transports im Netzwerk geändert, bzw. manipuliert. Ein Angreifer reichert XML-Nachrichten - auf dem Weg vom Konsumenten zum Web-Service - um schädliche Bestandteile an.

Werden bei einer Manipulation weder Credential noch signierte oder verschlüsselte Bestandteile einer Nachricht verändert, bemerkt der empfangende Web-Service die Manipulation nicht zwangsläufig, autorisiert das Credential und verarbeitet dann die manipulierte Nutzlast.

Im Gegensatz zu Malicious Morphing zielt Message Tampering auf mehr als die Veränderung von Werten ab. Je nach Manipulation der Nutzlast kann der eintretende Schaden unterschiedlich sein:

- Manipulation der Werte stellt einen Angriff auf die Datenkonsistenz dar.
- Manipulation der verwendeten Aktionen erlauben dem Angreifer unautorisiert Privilegierte Aktionen auszuführen.
- Manipulation der Elemente oder der Nachrichtenstruktur erlauben XDoS Attacken auszuführen.

Der Angriff ist auch bekannt unter den Namen Content Tampering, Message Alteration, Data Tampering und Falsified Message.

3.3.6 Parameter Tampering

Ein Angreifer manipuliert Werte einer SOAP-Anfrage, um Verarbeitungsfehler innerhalb der Anwendungslogik zu provozieren. Ein präparierter Wert, der ungeprüft als Parameter innerhalb der Anwendungslogik Verwendung findet, kann zu einer Fehlermeldung führen, die Auskunft über Interna des Services preisgibt. Eine unzureichende Fehlerbehandlung kann zum Ausfall des Services führen.

Beispielweise kann ein Parameter einen Wert enthalten, der den Wertebereich des verwendeten Datentypen verletzt z.B. ein Wert 2^{32} bei einem Integer-Datentypen der einen Wertebereich bis 2^{31} besitzt. Die Auswirkung kann die Rückgabe einer Fehlermeldung sein, aber auch ein DoS.

Enthält ein Schema in der WSDL-Beschreibung die Angabe maxOccours="unbounded", kann ein Angreifer einen XDoS Angriff durchführen. Der Angreifer kann eine beliebige Anzahl des Elementes senden um die Systemressourcen auszulasten. Ein solcher SOAP-Request wäre weiterhin valide gegenüber dem Schema (vgl. [Zedlitz, 2006, S.10] und [Lindstrom, 2004, S.5]).

Wird der Wert innerhalb der Anwendungslogik ungeprüft als Parameter eines auszuführenden Kommandos verwandt, sind unter Umständen Injection-Code Attacken möglich (siehe SQL-Injection, XPath-Injection, Command-Injection und Buffer Overflow).

Gegenmaßnahme: Eine Prüfung der Parameter gegen WSDL-Beschreibung und XML-Schemas die aus verlässlichen Quellen stammen bietet einen Schutz gegen falsche Datentypen. Ein Schema sollte Vorgaben bezüglich der maximalen Häufigkeit von Elementen anstelle der Angabe „unbounded“ enthalten. Zedlitz [2006] empfiehlt bei der Erstellung des XML-Schemas für die übertragenen Typen darauf zu achten, den Datentyp möglichst „streng“ zu wählen.

Der Angriff ist auch bekannt unter dem Namen WSDL Parameter Tampering und XML Parameter Tampering.

3.3.7 SQL-Injection

Ein Angreifer manipuliert in XML-Nachrichten eingebettete Teile eines SQL-Statements. Vorbedingungen für einen erfolgreichen Angriff sind, dass die Anwendungslogik unzureichende Prüfung von Metazeichen des Auszuführenden SQL-Statements durchführt und die ausführende Komponente[2] ausreichende Berechtigung zur Ausführung des Angriffes besitzt (vgl. WASC [2005] und [Actional, 2004, S.12])

[2] Häufig existiert ein dedizierter Datenbankbenutzer innerhalb der Anwendungslogik.

Durch fehlerhafte SQL-Statements lassen sich Fehlermeldungen provozieren. Wenn der Web-Service diese Fehlermeldung weiterleitet, kann der Angreifer diese Informationen nutzen um systematisch den Aufbau der Datenbank- und Tabellenstrukturen zu erkunden. Nach Zedlitz erlaubt die Standardeinstellung vieler Webserver die Ausgaben der Fehlermeldung zu lesen - dies gilt gleichermaßen für Webapplikationen und Web-Services (vgl. [Zedlitz, 2006, S.10]). Blind-SQL-Injections kommen zum Einsatz, wenn die Auslieferung der Fehlermeldung unterdrückt wird bzw. generisch ist. Durch eine gezielte Manipulation, die das SQL-Statement immer Wahr oder Falsch werden lässt, kann herausgefunden werden, ob eine SQL-Injection dennoch möglich ist (siehe WASC [2005] und [Actional, 2004, S.13]).

Ein Angreifer der Kenntnisse über Datenbank- und Tabellenstrukturen besitzt, ist er in der Lage durch manipulierte SQL-Statements, gezielt Daten in Tabellen zu verändern oder zusätzliche Daten aus der Datenbank zu beziehen. SQL-Statements außerhalb der regulären Datenbankanfrage führen dazu, dass vertrauliche Informationen offengelegt oder Daten manipuliert werden. Enthält die XML-Nachricht ein komplettes SQL-Statement, kann dieses beliebig verändert werden. Sollten nur Teile des Statements in der XML-Nachricht enthalten sein, werden zusätzliche SQL-Statements in das bestehende Statement geschleust (injiziert).

Ein Beispiel, bei dem dieses SQL-Statement in der Anwendungslogik sitzt:
SQL-Statement - `SELECT USER_NAME FROM USERS WHERE USER_ID = '?1'`

In der XML-Nachricht ist der Teil des SQL-Statements, der den Parameter ?1 für die WHERE Klausel USER_ID=?1 beinhaltet:

Original: `Smith01`
Manipulation 1: `Smith01' OR '1'='1'`
Manipulation 2: `Smith01' ;DELETE FROM USERS`

Manipulation 1 hebt die Einschränkung der WHERE Klausel auf.
Ziel: Verletzung der Vertraulichkeit.

Manipulation 2 löscht alle Benutzer.
Ziel: Verletzung der Datenkonsistenz; Transitives Ziel: Verletzung der Berechtigung.

Dieser Angriff hat das Hauptziel Vertraulichkeit und Daten-Integrität zu verletzen.

Ein Verletzung der Daten-Integrität von datenbankbasierten Berechtigungssystemen, kann als transitives Ziel zu einer Verletzung der Berechtigung führen. Rechte können beliebig verändert oder neue Benutzer angelegt werden. Diese Spezialisierung der SQL-Injection hat den Namen Authenticiation Bypass with SQL-Injection.

Gegenmaßnahmen: Validieren des Parameters gegen einen Regulären Ausdruck bzw. das enthalten von Metazeichen vermeiden das Problem.

3.3.8 XPath-Injection

Bei dieser Angriffstechnik manipuliert ein Angreifer, analog zur SQL-Injection, die in XML-Nachrichten eingebetteten Teile einer XPath-Suche. XPath wird verwendet um auf einzelne Elemente eines XML Dokuments zuzugreifen. Standards wie z.B. XSLT und XQuery verwenden XPath. Unter der Voraussetzung, dass die Anwendungslogik eine unzureichende Prüfung der XPath-Suche durchführt, ist der Angreifer in der Lage manipulierte Statements außerhalb der regulären XPath-Suche zu verwenden. Im Gegensatz zu regulären Datenbanken haben XML-Datenbanken (XML-Dokumente) keine Berechtigungsmechanismen - der Angreifer erhält uneingeschränkten Zugriff (vgl. Klein [2005] und Dwibedi [2005]).

Eine erweiterte Form der XPath-Injection stellt die Blind-XPath-Injection dar. Die XPath-Suche wird so erweitert, dass sie immer einen festen Boolschen Wert zurück gibt. Eine aufwendigere Methode ist das XML-Crawling, bei der Iterativ alle Elemente der XML-Datenbank durchgegangen werden. Blind-XPath-Injection wird näher beschrieben unter Klein [2005].

Ziel einer manipulierten XPath-Suche ist, Informationen aus der XML-Datenbank zu gewinnen.

Ein Beispiel, bei dem dieses XPath-Suche in der Anwendungslogik sitzt:
XPath-Anfrage - `string(//user[name/text()='?1'and password/text()='?2']/account/text())`

Die Rückgabewert entspricht dem Account, wenn bei Benutzername und Passwort übereinstimmen.

In der XML-Nachricht sind die Teile der XPath-Suche enthalten, die den Parameter ?1 für den Benutzernamen und ?2 für das Passwort beinhalten:

Original: ?1: `Smith01` ?2: `passwd`
Manipulation: ?1: `' OR '1'='1'` ?2: `'passwd'`

Die Parameter der XPath-Suche werden so verändert, dass die Bedingung innerhalb des Ausdruckes immer Wahr wird. Die XPath-Suche gibt somit den Account-Wert des ersten User-Elements zurück. Das in ?2 übergebene Passwort hat dadurch keine Bedeutung mehr.

Der Standard WS-Security sieht die Verwendung von XPath zur Ansteuerung seiner Elemente innerhalb der SOAP-Nachricht vor. Die Anwendungslogik entspricht dabei der Applikationsserver Implementierung des WS-Security Standards. Eine Untersuchung der WS-Security Implementierung des Applikationsservers würde aufgrund der Komplexität des

Standards den Rahmen der Arbeit übersteigen und findet daher keine weitere Betrachtung.

Gegenmaßnahmen: Validieren des Parameters gegen einen Regulären Ausdruck bzw. das enthalten von Metazeichen vermeiden das Problem.

3.3.9 XML-Injection

Bei einer XML-Injection manipuliert eine Angreifer die in XML-Nachrichten eingebetteten Parameter eines schreibenden XML-Ausdrucks. Verwendet die Anwendungslogik einen unzureichend geprüften Parameter bei einer schreibenden XML-Anweisung ist es möglich, dass ein Element in das XML-Dokument eingefügt wird, das Metazeichen enthält die für die eigentliche Struktur des XML-Dokuments reserviert sind.

Werden Daten außerhalb der vorgesehen XML-Anwendung gespeichert, beispielsweise durch einfügen eines nicht vorgesehen Kindknotens, hat der Angreifer die Daten-Integrität kompromittiert. Wird die Struktur des XML-Dokuments durch einfügen schadhaft formatierter Metazeichen zerstört, resultiert die Verletzung der Daten-Integrität unter Umständen in einer DoS-Attacke, da die Anwendungslogik nicht mehr fehlerfrei auf das XML-Dokument zugreifen kann. Nach BSI [2008a] ist im Zuge einer XML-Injection die Struktur der XML-Datei im Regelfall zerstört.

Gegenmaßnahmen: Validieren des Parameters gegen einen Regulären Ausdruck bzw. das enthalten von Metazeichen vermeiden das Problem.

3.3.10 Übersicht

Tabelle 3.2 zeigt eine Übersicht der Angriffe auf die Daten-Integrität und Vertraulichkeit.

Angriffstechnik	**Ziel**	**Ort**	**Schwachstelle**
WSDL Scanning	Vertraulichkeit, Berechtigung	UDDI, WSDL, Applikationsserver	WSDL direkt oder über UDDI öffentlich zugänglich. Unzureichende Prüfung von Methodenaufrufen gegen die WSDL.
Message Sniffing	Vertraulichkeit	Netzwerk, Intermediary, Applikationsserver	Unzureichende Verwendung von WS-Security bzw. SSL/TLS.
Routing Detour	Vertraulichkeit	Manipulierter Intermediary	Unzureichende Verwendung von WS-Security bzw. SSL/TLS.
Malicous Morphing	Daten-Integrität	Netzwerk, Manipulierter Intermediary	Fehlender Einsatz von digitalen Signaturen.
Message Tampering	Daten-Integrität, Verbindlichkeit, Verfügbarkeit	Netzwerk	Unzureichende Verwendung von WS-Security bzw. SSL/TLS. Unzureichende Parameterprüfung innerhalb der Anwendungslogik.
Parameter Tampering	Daten-Integrität, Verfügbarkeit	Anwendungslogik	Unzureichende Parameterprüfung gegen die WSDL bzw. innerhalb der Anwendungslogik.
SQL-Injection	Vertraulichkeit, Daten-Integrität	Anwendungslogik	Unzureichende Parameterprüfung innerhalb eines SQL-Statements.
XPath-Injection	Vertraulichkeit	Anwendungslogik	Unzureichende Parameterprüfung innerhalb einer XPath-Suche.
XML-Injection	Daten-Integrität	Anwendungslogik	Unzureichende Parameterprüfung innerhalb einer XML-Anweisung.

Tabelle 3.2: Übersicht der Angriffe auf die Daten-Integrität und Vertraulichkeit

3.4 Angriffe auf die Systemumgebung

Das Ziel der Angriffe dieser Angriffsklasse ist Manipulationen am Verhalten der Systemumgebung zu erreichen. Meist werden sie durch eine unzureichende Prüfung der zu verarbeitenden Daten innerhalb der Anwendungslogik möglich. Legacy-Anwendungen, die durch eine Service gekapselt werden, bieten die geeignete Angriffsfläche für Angriffe dieser Angriffsklasse. In der Regel sind die Legacy-Anwendungen und die Programmiersprachen in denen sie entwickelt wurden auf XML-Bedrohungen nicht vorbereitet.

3.4.1 Command Injection

Bei dieser Angriffstechnik manipuliert ein Angreifer die in XML-Nachrichten eingebetteten Parameter eines Kommandos. Unter der Voraussetzung, dass die Anwendungslogik eine unzureichende Prüfung der Parameter durchführt, ist der Angreifer in der Lage Befehle außerhalb der regulären Nutzung zu verwenden. Unter Umständen ist eine Ausführung weiterer Befehle möglich.

Findet die Verarbeitung des Kommandos innerhalb des Applikationsservers statt, kann der Angreifer die Rechte des Web-Services nutzen um weitere Funktionen aufzurufen. Wird die Kommandoverarbeitung an das Betriebssystems weitergereicht, hat der Angreifer die Möglichkeit Systembefehle mit den Rechten des Hostprozesses auszuführen.

Gegenmaßnahme: Validieren des Parameters gegen einen Regulären Ausdruck bzw. das enthalten von Metazeichen vermeiden das Problem.

3.4.2 XML-Encapsulation

Bei der XML-Encapsulation wird boshafter Code in CDATA Blöcken oder Parametern einer XML-Nachricht eingebettet. Normalerweise wird der Inhalt von CDATA Blöcken nicht geparsed. Eine unzureichende Prüfung des Parameters bzw. CDATA Blocks kann bewirken, dass boshafter Code in der Anwendungslogik verarbeitet wird. Enthält die Antwort an einen Portal-Client boshaften JavaScript Code, wird dieser unmittelbar beim Client ausgeführt (vgl. [Actional, 2004, S.16]).

Ein Beispiel für einen CDATA Block:

```
<attack>
<![CDATA[<]]>SCRIPT<![CDATA[>]]>
alert('XSS');
<![CDATA[<]]>/SCRIPT<![CDATA[>]]>
</attack>
```

Der Angriff ist auch bekannt unter dem Namen Code-Injection und Cross-Site-Scripting.

3.4.3 XML-Virus

Ein Angreifer sendet SOAP-Attachements die in der Anwendungslogik ungeprüft weiter verarbeitet werden. Es ist möglich, dass ein Virus in Form einer ausführbaren Datei oder einer manipulierten Nutzlast-Datei, beispielsweise eine manipulierte Bild-Datei, im Anhang enthalten sind. Ein Web-Service der SOAP-Attachements empfängt, obwohl sie per se im Web-Service nicht vorgesehen sind, verbraucht unnötige Ressourcen (vgl. [O'Neill, 2005, S.134]).

Gegenmaßnahme: Wenn SOAP-Attachements innerhalb des Web-Service vorgesehen sind, bringt eine Überprüfung des Attachements durch eine geeignete Antivirus Anwendung sowie eine Filterung der Attachements nach MIME-Type einen Grundschutz. Sind SOAP-Attachements nicht vorgesehen, sollten die Attachements explizit verboten werden.

3.4.4 Malicious Include

Bei diesem Angriff enthält die XML-Nachricht eine Entity-Referenz die eine lokale Datei definiert. Anhand der Fehlermeldungen kann der Angreifer Rückschlüsse über die Existenz einer Datei ziehen. Ist der Inhalt der aufgelösten „Entity“ in der Fehlermeldung des Web-Services enthalten, so erhält der Angreifer möglicherweise eine Kopie der Datei. Der Angriff Zielt auf die Verletzung der Vertraulichkeit ab.

Ein Beispiel bei dem versucht wird die Passwort-Datei eines Linux/Unix Systems auszulesen:

```
<!DOCTYPE show [
<!ENTITY filecontent SYSTEM "file:///etc/passwd">]>
<show>&filecontent</show>
```

Ist die Entity Teil der Antwort, erhält der Angreifer eine Kopie des Dateiinhaltes.

Bei einem Unix/Linux-System kann das Referenzieren von beispielweise /dev/random[3] zu einem Denial of Service führen.

Gegenmaßnahme: Nach SOAP12 [2007] darf eine SOAP-Nachricht keine DTD enthalten.

The XML infoset of a SOAP message MUST NOT contain a document type declaration information item [SOAP12, 2007, Kapitel 5].

WS-I Basic konforme Web-Service-Implementierungen sind immun gegen diese Angriff. Sie Verbieten, dass Vorhandensein von DTD innerhalb der XML-Nachricht (siehe [Zedlitz, 2006, S.8]).

An ENVELOPE MUST NOT contain a Document Type Declaration [WSI-B12, 2007, R1008].

Die Spezifikationen legt nicht fest, wie ein XML-Parser eine SOAP-Nachricht mit DTD verarbeiten soll. Der Einsatz eines XML-Parsers, der DTD innerhalb der SOAP-Nachricht verbietet, behebt das Problem.

Der Angriff ist auch bekannt unter dem Namen XML External Entity (XXE) Attack.

3.4.5 Übersicht

Tabelle 3.3 zeigt eine Übersicht der Angriffe auf die Systemumgebung.

Angriffstechnik	**Ziel**	**Ort**	**Schwachstelle**
Command-Injection	Berechtigung, Systemumgebung	Anwendungslogik	Unzureichende Parameterprüfung innerhalb eines Befehls.
XML-Encapsulation	Berechtigung, Systemressourcen, Systemumgebung	Anwendungslogik	Unzureichende Prüfung von Elementen vor der Ausführung.
XML-Virus	Berechtigung, Systemressourcen, Systemumgebung	Anwendungslogik	Unzureichende Prüfung von Attachements vor der Ausführung.
Malicious Include	Vertraulichkeit	XML-Parser	XML-Parser verarbeitet DTD.

Tabelle 3.3: Übersicht der Angriffe auf die Systemumgebung.

[3] Das Gerät liefert bei Zugriff einen endlosen Strom an Zufallswerten.

3.5 Angriffe auf die Verfügbarkeit

Diese Angriffstechniken nutzen manipulierte XML-Nachrichten um die Verfügbarkeit von Web-Services zu attackieren. Sie stellen eine Adaption der klassischen Denial-of-Sevice Angriffe auf Web-Services dar. Der Begriff XML Denial-of-Service (XDoS) ist zum Synonym für diese Angriffsklasse geworden. Das Ziel von XDoS Attacken ist den Web-Service für legitime Subjekte unbenutzbar zu machen. Unbenutzbar bedeutet in diesem Fall, dass der Web-Service nur noch in der Lage ist stark verzögert oder überhaupt nicht zu antwortet.

In den folgenden Unterkapiteln werden diese Angriffstechniken näher erläutert.

3.5.1 Buffer Overflow

Dieser Angriff beruht auf einer unzureichenden Fehlerbehandlung bei überlangen Eingaben seitens des empfangenden Services oder der unzureichenden Überprüfung der dahinter liegenden Anwendungslogik. Ein Buffer Overflow kann durch die Verarbeitung eines einzelne Elemente, gruppierter Elemente oder der gesamten XML-Nachricht entstehen.

Ist das empfangende System nicht in der Lage Elemente oder Nachrichten mit einer unerwarteten Größe zu verarbeiten, besteht unter Umständen die Möglichkeit beliebigen Code mit Rechten des Host-Prozesses auszuführen oder die System Verfügbarkeit zu kompromittieren (vgl [Moradian und Hakansson, 2006, S.158] und [S.11]Acti04).

Parameter Tampering (siehe Kapitel 3.3.6) und beinahe alle Angriffe auf den XML-Parser enden in einem Buffer Overflow.

Gegenmaßnahme: Durch eine verbesserte Fehlerbehandlung innerhalb des Applikationsservers, bzw. eine strengere Überprüfung der Eingabe seitens der Anwendungslogik ist das Problem reduzierbar.

Der Angriff ist auch bekannt unter dem Namen Buffer Overrun.

3.5.2 Schema Poisoning

Ein Angreifer kompromittiert ein XML-Schema an seinem gespeicherten Ort. Dabei wird das XML-Schema durch ein ähnliches, modifiziertes Schema ersetzt, bei dem veränderte oder fehlende Restriktionen bezüglich der Struktur und Datentypen vorliegen.

Beispielsweise führt eine XML-Nachricht, die ein manipuliertes XML-Schema referenziert unter Umständen zu einem Denial-of-Service, wenn arithmetische Operationen Zahlenwerte

anstelle des gelieferten Strings erwarten (vgl. [Lindstrom, 2004, S.6] und [Moradian und Hakansson, 2006, S.163f]).

Gemäß dem modifizierten XML-Schemas ist der Web-Service ungeschützt gegen die meisten XDoS Angrffe, wie z.B. Coercive Parsing oder Buffer Overflow.

Gegenmaßnahme: Eine Prüfung der Schemas die in der XML-Nachricht enthalten sind gegen WSDL-Dokumente und XML-Schemas die aus verlässlichen Quellen stammen bietet einen ausreichende Schutz gegen diese Angriffsart.

3.5.3 Ressource Hijacking

Diese Angriffstechnik zielt auf das Blockieren bzw. Auslasten von Ressourcen des Zielsystem. Eine XML-Nachricht die eine Transaktion auslöst, die niemals abschließt, sorgt unter Umständen für eine Sperrung von Ressourcen.

Datenbank-Anbindungen werden innerhalb der Anwendungslogik meist unter Verwendung von Connection Pools[4] mit begrenzter Anzahl von Connections hergestellt. Findet innerhalb der Anwendungslogik eine Datenbankabfrage statt, beispielsweise im Zuge der Authentifizierung, wird eine Connection aus dem Connection Pool verwendet. Wird solch eine Anfrage bei einer Flooding Attacke verwendet, kann unter Umständen der gesamte Connection Pool blockiert werden.

Gegenmaßnahme: Eine Beschränkung der Laufzeit einer Transaktion vermindert das Problem der Ressourcensperrung bei Transaktionen. Weitere Gegenmaßnahmen sind der Flooding Attack zu entnehmen.

3.5.4 Publik Key DoS Attack

Bei diesem Angriff befinden sich eine große Anzahl Encryption- bzw. Signatur-Elemente im Header. Die zu Entschlüsselnde bzw. Signierende Nutzlast befindet sich eingebettet innerhalb des Dokuments oder als Referenz auf eine externe URI.

Der Einsatz kryptografischer Funktionen in Verbindung mit Public Key Verfahren ist stark ressourcenlastig. Sie führen zu einer starken Prozessorlast.

Die Verwendung einer Referenz auf eine externe URL stellt eine Spezialisierung dieses Angriffs dar. Sie werden unter Signature Redirect und Encryption näher erläutert.

[4] Das Vorhalten von Datenbankverbindungsobjekten verhindert eine teuere Objekterzeugung.

3.5.5 Signature Redirect

In einer XML-Nachricht befindet sich eine XML-Signatur, die eine Referenzierung auf signierte Daten auf einer externen URL verwendet. Zur Validierung der Signatur muss eine verarbeitende Anwendung diese URL dereferenzieren, um die Daten herunterzuladen und anschließend ihre Signatur zu berechnen.

Der Angriff zielt auf einen Bandbreiten-Verbrauch während des Herunterladens und Ressourcen-Verbrauch des Systems während der Signatur-Berechnung ab.

Die XML-Signature Spezifikation sieht vor, dass implementierende Anwendungen eine URI Syntax parsen können müssen. Weiterhin empfiehlt der Standard, dass URI's im HTTP Schema dereferenzierbar sein sollten (siehe [XML-DSIG, 2008, Kapitel 4.3.3.1]). In [XML-DSIG, 2008, Kapitel 3.2] wird darauf hingewiesen, dass Anwendungen gültige Signaturen unter bestimmten Bedingungen möglicherweise nicht validieren können, beispielsweise wenn die Dereferenzierung von spezifischen URI's (aufgrund von unerwünschten Seiteneffekten des URI Schemas) nicht gestattet ist. (vgl. [O'Neill, 2005, S.134f])

Zum Beispiel: Die Referenzierung <ds:Reference URI ="http://ardownload.adobe.com/pub/adobe/reader/win/8.x/8.1.2/enu/AdbeRdr812_en_US.exe"> beinhaltet eine 22MB große Datei. Das Herunterladen verbraucht die Bandbreite des Systems, während die Verarbeitung Systemressourcen verbraucht. Der Angriff wird effektiver, wenn die referenzierte Datei größer ist und die Nachricht per Flooding mehrfach verarbeitet werden muss.

Signature Redirect ist ein spezialisierter Public Key DoS Angriff.

Gegenmaßnahme: Der Problem kann verhindert werden, indem die XML-Signature Anwendung die Derefenzierung von URL's nicht erlaubt. Eine systemweite Beschränkung der gleichzeitig erlaubten URL Dereferenzierungen kann das Problem mildern.

3.5.6 Encryption Redirect

Analog zu dem Signature Redirect ist es auch bei einem Encryption Redirect möglich eine externe URL Referenz als Ort der verschlüsselten Daten zu verwenden. Die XML-Encryption Spezifikation verwendet die URI-Definition der XML-Signature Spezifikation.

URIs MUST abide by the type definition and the [XML-DSIG, 4.3.3.1 The URI Attribute] specification [XML-ENC, 2002, Kapitel 1.2]

Das Beispiel aus Signature Redirect funktioniert ebenso für Encryption Redirect, lediglich das Reference-Tag heisst nun <enc:DataReference >.

Encryption Redirect ist ein spezialisierter Public Key DoS Angriff.

Gegenmaßnahme: Wird die Derefenzierung auf eine URL nicht benötigt, löst das verbieten der URL Dereferenzierung das Problem. Eine systemweite Beschränkung der gleichzeitig erlaubten URL Dereferenzierungen kann das Problem mildern.

3.5.7 Extra-long Names

Eine explizite Begrenzung der Länge von Element- und Attribut-Namen sowie Namensraum-Präfixen ist in den Spezifikationen XML [2006] und XML-NS [2006] nicht enthalten. Hat der XML-Parser keine interne Maximallänge für diese Namen vorgegeben, können beliebig lange Namen gesendet werden, bis dem parsenden Rechner der Hauptspeicher ausgeht (vgl. [Zedlitz, 2006, S.8]).

Der Extra-long Names Angriff ist eine Spezialisierung des Buffer Overflow Angriffes.

Gegenmaßnahme: Eine Vorgabe für die Maximallänge lokaler Namen innerhalb des XML-Parsers behebt das Problem.

Der Angriff ist auch bekannt unter den Namen MegaTags und Jumbo Tag Names.

3.5.8 Namespace Prefix Attack

Bevor ein Namensraum-Präfix deklariert werden kann, müssen alle Attribute eines Elementes gelesen werden, da sich am Ende des öffnenden Tags noch eine neue Definition für das Namensraum-Präfix befinden könnte.

Der Typ eines Attributes kann durch das XML-Schema erst bestimmt werden, nachdem der gesamte Inhalt des Attributes gelesen wurde und ein lokaler Name und Namensraum bekannt sind. Fehlt eine Grenze für den Attribut-Inhalt, können beliebig große Daten gesendet werden. In der Praxis sind bereits XML-Dokumente mit Attribut-Inhalten von mehreren Kilobyte Größe aufgetaucht (vgl. [Zedlitz, 2006, S.8]).

Ein Beispiel:

```
<soapenv:Envelope xlmns:soapenv="http://schemas.xmlsoap.org">
   <soapenv:Body xlmns="http://example.com/">
     <ns:Prefix Attack="xxx .... x">
     </ns:Prefix>
   </soapenv:Body>
</soapenv:Envelope>
```

Mit der Absicht, den XML-Parser bei der Verarbeitung des Attribut-Inhaltes zu überlasten enthält das Attribut in dem Beispiel einen überlangen String.

Der Namespace Prefix Attack ist eine Spezialisierung des Buffer Overflow Angriffes.

Gegenmaßnahme: Eine Vorgabe für die Maximallänge der Attribut-Inhalte sowie eine Begrenzung der maximal Anzahl der Attribute.

3.5.9 Coercive Parsing

Bei einem Coercive Parsing Angriff empfängt der Web-Service eine XML-Nachricht mit komplexen Verschachtelungen, welche im Rahmen des verwendeten Schemas valide sind. Das Ziel ist den XML-Parser während des parsens einer solchen Nachricht zu überlasten.

Die XML-Nachricht enthält eine hohe Anzahl von Geschwister-Knoten auf einer Ebene (XML Document Width Attack) bzw. eine hohe Anzahl von Kindern in die Tiefe (XML Document Depth Attack). Meist findet eine Kombination aus beiden statt.

Die SOAP Spezifikation sieht keine Beschränkung für das zusätzlich Auftreten von Elementen innerhalb des SOAP:Header, SOAP:Body, bzw. SOAP:Fault vor, solange diese nicht aus dem SOAP-Namensraum stammen. Aus [SOAP12, 2007, Kapitel 5.1] geht hervor, dass ein SOAP:Envelope ein oder zwei Kind-Elemente hat: den optionalen Header und Body. Nach [Zedlitz, 2006, S.9] lassen sich im SOAP:Envelope beliebige XML-Elemente transportieren, die im Normalfall nicht verarbeitet werden, da nur SOAP:Header, SOAP:Body und SOAP:Fault ausgewertet werden. Ist für ein SOAP:Header Element das Attribut SOAP:mustUnderstand="0" gesetzt, stört es die weiter Kommunikation nicht.

Der Coercive Parsing Angriff ist ein spezialisierter Buffer Overflow Angriff.

Gegenmaßnahme: Die erlaubte Verschachtelungstiefe und Breite sollte auf sinnvolle Vorgaben beschränkt werden. Der XML-Parser inspiziert die XML-Nachricht und setzte die Vorgaben bezüglich der maximalen Tiefe und Breite der Knoten durch.

Unter Verwendung von WSI-Basic 1.2 lassen sich zusätzliche Kind-Elemente innerhalb des SOAP:Envelope und SOAP:Fault unterbinden:

A MESSAGE MUST NOT have any element children of soap:Envelope following the soap:Body element [WSI-B12, 2007, R1011].

When a MESSAGE contains a soap:Fault element, that element MUST NOT have element children other than faultcode, faultstring, faultfacrtor and detail [WSI-B12, 2007, R1000].

Der Angriff ist auch bekannt unter den Namen Recursive Payload und Nested Data Attack

3.5.10 Flooding Attack

Bei diesem Angriff werden legitime XML-Nachrichten in hoher Frequenz an den Web-Service gerichtet. Ziel des Angriffes ist die Ressourcen des Web-Service zu überlasten, so das Anfragen legitimer Subjekte verzögert oder überhaupt nicht bearbeitet werden können. Der Angriff kann mit einer Replay Attack, einem Signature Redirect Angriff oder anderen XDoS Angriffen kombiniert werden um die Auswirkung zu erhöhen.

Gegenmaßname: Durch Vorgabe einer Maximalanzahl von Anfragen, die der Service von einem Konsument in einem gegebenen Zeitfenster empfangen darf, kann der Angriff abgemildert werden.

Der Angriff ist auch bekannt unter dem Namen XML Flood.

3.5.11 XML Document Size Attack

Die XML Document Size Attack ist ein XDoS Angriff, bei dem der XML-Parser eine sehr große XML-Nachricht verarbeiten muss. Ist der XML-Parser während des Empfangs nicht in der Lage den XML-Datenstrom der Nachricht zu verarbeiten, kommt es zu einem Buffer Overflow, der bei unzureichender Fehlerbehandlung in einem XDoS endet.

Bei diesem Angriff kann eine Kombination von Angriffstechniken gegen den XML-Parser Verwendung finden, mit dem Ziel eine Möglichst große Nutzlast zu erzeugen. Überlange Tag-Namen, Attribut-Name und Werte der Elemente führen schnell zu großen XML-Dokumenten. Gleiches gilt für eingebettete XML-Attachements in die XML-Nachricht (vgl. Zedlitz [2006] und Lindstrom [2004]).

Unter Verwendung eines Dom-Parsers wird der gesamte XML-Dokumentenbaum einer XML-Nachricht im Hauptspeicher aufgebaut. Während dem Aufbau des Dokumentenbaums verbraucht der Dom-Parser ein Vielfaches mehr an Hauptspeicher und Ressourcen als bei der Erstellung nötig waren, bzw. als das XML-Dokument groß ist. Wenn der Applikationsserver einen eventbasierten XML-Parser einsetzt, wie beispielsweise SAX oder StAX, wird das Problem abgeschwächt (vgl. Zedlitz [2006]).

Gegenmaßnahme: Der XML-Parser muss eine Beschränkung des XML-Datenstroms vornehmen, sowie Länge und Anzahl von Elementen und ihren Eigenschaften sinnvoll begrenzen.

Der Angriff ist auch bekannt unter den Namen Jumbo Payload und Oversized Payload.

3.5.12 XML Entity Expansion

Der Angreifer verwendet eine hohe Anzahl von Referenzierungen auf eine Entity-Deklaration mit überlangem Inhalt innerhalb einer XML-Nachricht. Hat der XML-Parser keine Beschränkung bezüglich der Referenzierungshäufigkeit, verbraucht der XML-Parser während der wiederholten Entity-Auflösung den zur Verfügung stehen Hauptspeicher.

Während der Entity-Auflösung ersetzt der XML-Parser die Referenzen dynamisch mit den dereferenzierten Daten.

Gegenmaßnahme: Nach SOAP12 [2007] darf eine SOAP-Nachricht keine DTD enthalten.

The XML infoset of a SOAP message MUST NOT contain a document type declaration information item [SOAP12, 2007, Kapitel 5].

WS-I Basic konforme Webservice-Implementierungen sind immun gegen diese Angriff. Sie Verbieten, dass Vorhandensein von DTD innerhalb der XML-Nachricht.

An ENVELOPE MUST NOT contain a Document Type Declaration [WSI-B12, 2007, R1008].

Demnach sollte ein SOAP-XML-Parser keine DTD verarbeiten dürfen.

Der Angriff ist auch bekannt unter dem Namen Quadrativ Blowup DoS Attack.

3.5.13 XML Recursive Entity Expansion

Bei der Rekursiven Variante der XML Entity Expansion verwendet der Angreifer eine rekursive Entity-Deklaration innerhalb einer XML-Nachricht. Erkennt der verarbeitende XML-Parser die rekursive Entity Deklaration nicht, verbraucht der XML-Parser während der Entity-Auflösung den zur Verfügung stehenden Hauptspeicher.

Eventuell vorhandene Rekursionen werden in beliebiger Rekursionstiefe aufgelöst. Erst wenn der XML-Parser die Datentypen zusammengesetzt hat, kann die XML-Nachricht validiert und weiterverarbeitet werden.

Gegenmaßnahme: siehe XML Entity Expansion

Der Angriff ist auch bekannt unter den Namen Recursive Entity Expansion und XML-Bomb.

3.5.14 XML Remote Entity Expansion

Im Unterschied zu der XML Recursive Entity Expansion finden bei dieser Angriffstechnik externe Entity-Deklarationen Verwendung. Sie verweisen auf die DTD eines entfernten Systems, deren Entity-Deklaration wiederum auch auf eine entfernte DTD verweist, usw.

Der XML-Parser muss sämtliche Entity-Auflösungen aller rekursiv benötigten XML-Dokumente vervollständigen, bevor die XML-Nachricht validiert und weiterverarbeitet werden kann.

Bei Intensiver Nutzung des Web-Services, entsteht so auch last der anderen beteiligten Rechner. Bereits der Ausfall eines entfernten Systems das an der Entity-Auflösung beteiligt ist, bedeutet eine Kompromittierung des Web-Services.

Gegenmaßnahme: siehe XML Entity Expansion

3.5.15 SOAP Array Attack

Diese Angriffstechnik verwendet die Möglichkeit die Anzahl der Elemente eines SOAP Arrays zu definieren bzw. die Position eines Elementes innerhalb des Arrays festzulegen. Im Zuge der Objekterzeugung reserviert das *XML to Java Object Mapping* den Hauptspeicher für das Array gemäß des Speicherbedarfs des Datentypen und der Anzahl der Elemente.

Ziel ist es den verfügbaren Hauptspeicher auszuschöpfen und dadurch das System zu kompromittieren.

Bei einem Sparse Arrays kann ein Offset für das Erste belegte Indize angegeben werden bzw. die Position eines Elementes angegeben werde. Die nicht belegten Elemente sind ebenfalls Teil des Arrays.

Ein Beispiel:

```
<soapenv:Envelope xmlns:soapenv="..." xmlns: soapenc:"...">
  <soapenv:Body>
    <ns1:FunctionWithArrayInput xmlns:ns1="...">
      <DataSet xsi:type="soapenc:Array"
       soapenc:arrayType="xsd:string[1000000]">
        <item xsi:type="xsd:string">Data1</item>
        <item xsi:type="xsd:string">Data2</item>
        <item xsi:type="xsd:string">Data3</item>
    </DataSet>
    </ns1:FunctionWithArrayInput>
  </soapenv:Body>
</soapenv:Envelope>
```

In diesem Beispiel wird ein String-Array mit 1000000 String-Objekte im Hauptspeicher angelegt.

Gegenmaßnahme: Ein Vergleich der Anzahl von vorhandenen Array Elementen mit der Anzahl durch das Attribut „soapenv:arrayType“ festgelegten Elemente sollte das Problem beschränken.

3.5.16 Übersicht

Tabelle 3.4 zeigt eine Übersicht der Angriffe auf die Verfügbarkeit.

Angriffstechnik	**Ziel**	**Ort**	**Schwachstelle**
Buffer Overflow	Verfügbarkeit, Systemumgebung	SOAP-Engine, Anwendungslogik	Unzureichende Fehlerbehandlung und Parameterprüfung durch das System.
Schema Poisoning	Verfügbarkeit	SOAP-Engine	Unzureichende Validierung externer XML-Schemas.
Ressource Hijacking	Verfügbarkeit, Ressourcen	Anwendungslogik	Unzureichender Schutz vor Ressourcensperrung.
Public Key Dos Atack	Verfügbarkeit	WS-Security	Fehlender Schutz vor hoher Anzahl signierter bzw. verschlüsselter Elemente.
Signature Redirect	Verfügbarkeit	WS-Security	Fehlender Schutz vor URL-Derefenzierung von signierten Daten.
Encryption Redirect	Verfügbarkeit	WS-Security	Fehlender Schutz vor URL-Derefenzierung von verschlüsselten Daten.
Extra-long Names	Verfügbarkeit	XML-Parser	Fehlende Beschränkung der Namenslänge.
Namensraum-Präfixe	Verfügbarkeit	XML-Parser	Fehlende Beschränkung der Attribut-Namen und Inhalten.
Coercive Parsing	Verfügbarkeit	XML-Parser	Fehlende Beschränkung der Element Verschachtelung.
Flooding Attack	Verfügbarkeit	SOAP-Engine, XML-Parser	Fehlende Beschränkung der Anzahl von Anfragen.
XML Doc. Size Attack	Verfügbarkeit	XML-Parser	Fehlende Beschränkung der Dokumentgröße.
XML Entity Expansion und Varianten	Verfügbarkeit	XML-Parser	XML-Parser verarbeitet DTD.
SOAP Array Attack	Verfügbarkeit	XML-Parser	Fehlerhafte Erkennung der Arraygröße.

Tabelle 3.4: Übersicht der Angriffe auf die Verfügbarkeit

3.6 Versuchsaufbau

In diesem Kapitel wird der Versuchsaufbau erläutert. Nach einer Betrachtung der Auswahlkriterien folgt eine Beschreibung des Vorgehens, dass die Rahmenbedingungen des Versuchsaufbaus festlegt. Abgeschlossen wird das Kapitel mit der Konzeption von Angriffen.

3.6.1 Auswahlkriterien

In den vorherigen Kapiteln wurde eine Vielzahl von Angriffstechniken unterschiedlicher Angriffsklassen dargestellt. Enthalten waren Angriffe auf die Schutzziele Identität, Verbindlichkeit, Vertraulichkeit, Daten-Integrität und Verfügbarkeit.

Der Einsatz von Basis-Schutz-Mechanismen der Web-Service-Security Spezifikation gewährleistet unter Verwendung von digitalen Zertifikaten das Einhalten der Schutzziele auf Nachrichten-Ebene in den Bereichen Identität, Verbindlich, Vertraulichkeit und Daten-Integrität. Unter Verwendung von Web-Service-Security Erweiterung lassen sich auch komplexere zuständigkeitsdomänenübergreifend Schutzmaßnahmen realisieren, wie sie in einem sicherheitskritischen Umfeld benötigt werden, beispielsweise Single Sign-On zwischen Geschäftspartnern.

Das Schutzziel Verfügbarkeit wird nur indirekt von WS-Security adressiert. WS-Security wirkt unterstützend bei der Einhaltung dieses Schutzziels, beispielsweise durch die Anwendung digitaler Signaturen auf XML-Schemas, deren Daten-Integrität und Verbindlichkeit somit gewährleistet wird.

Gerade dieses Schutzziel stellt einen grundlegenden Aspekt für den Betrieb eines Web-Service dar. Sind die Merkmale Erreichbarkeit und Reaktionszeit Teil eines Service Level Agreements, kann ein Denial of Service zu hohen Vertragsstrafen führen. Bei einem kostenpflichtigen Service gehen Probleme mit der Verfügbarkeit einher mit Verdienstausfall. Ein Ausfall eines Services, der Teil eines geschäftskritischen Prozesses darstellt, bewirkt weitreichendere Schäden, beispielsweise wenn der Einkaufs/Verkaufs-Service eines Börsenmaklers nicht Verfügbar ist, bedeutet dies Ausfall in einem geschäftskritischen Bereich, der neben Verdienstausfall auch einen Verlust der Reputation bedeutet.

Diese Arbeit beschränkt sich bei der Konzeption und Realisierung von Angriffen auf die Betrachtung der Angriffstechniken, die auf die Verfügbarkeit von Web-Services abzielen, die implementationsspezifische Eigenschaften des Applikationsservers für die Realisierung eines Angriffes bedingen.

Die Angriffstechniken lassen sich in folgende Bereiche unterteilen:

- XML-Inhalt
  - Buffer Overflow
  - Extra-long Names
  - Namepspace Prefix Attack
  - Coercive Parsing
  - Flooding Attack
  - XML Document Size Attack
- XML-Struktur
  - XML Entity Expansion
  - XML Recursive Entity Expansion
  - XML Remote Entity Expansion
  - SOAP Array Attack
- WS-Security-Threats
  - Public Key DoS
  - Signature Redirect
  - Encryption Redirect
- Andere
  - Schema Poisoning
  - Ressource Hijacking

Die Angriffstechniken der Bereiche XML-Inhalt und XML-Struktur haben das gemeinsame Ziel den XML-Parser zu überlasten und dadurch die Verfügbarkeit zu kompromittieren.

Die XML-Inhalt Angriffe geschehen unter Verwendung von wohlgeformten, validen XML-Nachrichten die den XML-Parser während des parsens überlasten. Unter Umständen kann die Nachricht mehrere hundert Megabyte Größe besitzen.

Die XML-Struktur Angriffe machen sich die Eigenschaft von XML zu nutze, Elemente dynamisch erzeugen zu können. Durch vielfache oder rekursive Dereferenzierung von Elementen erzeugt der XML-Parser die XML-Struktur während des parsens im verfügbaren Hauptspeicher. Bei einer SOAP Array Attack wird der Speicher durch den XML-Parser reserviert und dabei ebenfalls Verbraucht. Die Nachrichten sind meist einige Kilobyte Groß.

Die Verwendung von WS-Security ermöglicht neue Anriffspunkte. Der Public Key DoS Angriff ist ein Beispiel dafür, dass einen beliebige Anzahl von unnötigen kryptografischen Operationen in einer XML-Nachricht enthalten sein kann. Ebenso gibt es keine Beschränkung für die Anzahl der enthaltenen SecurityToken.

XML Remote Entity Expansion wird nicht näher betrachtet - sie unterscheidet sich von der XML Entity Expansion in dem Detail, das zur Entity-Auflösung rekursive Dokumente Type Declarationen von entfernten Systemen benötigt werden.

Schema Poisoning wird nicht gesondert betrachtet. Dieser Angriff erlaubt durch ein aufheben von Schema-Restriktionen den Einsatz der oben genanten XML-Inhalt und XML-Struktur Angriffe.

Die WS-Security-Threats würden aufgrund der Komplexität des benötigten Szenarios den Rahmen der Arbeit übersteigen und finden daher keine weitere Betrachtung.

Die Ressource Hijacking Angriffstechnik wird aufgrund der Komplexität der zusätzlichen benötigten Infrastruktur nicht weiter Betrachtet. Der Einsatz von Datenbanken und Transaktionen ist im Rahmen der Arbeit nicht vorgesehen.

3.6.2 Versuchsumgebung

Ein exemplarischer Web-Service wird erstellt und als EAR-Datei auf einem *WebSphere Application Server 6.1* (WAS) mit Standard-Konfiguration eingespielt.

Der exemplarische Web-Service besteht aus drei Arten von Diensten:

- Stringverarbeitung
- Zahlenarithmetik
- Arrayverarbeitung

Konfiguration und Versionierung

Folgende Geräte-Konfigurationen und Softwareversionen finden im Rahmen der Versuche Verwendung:

Applikationsserver	
Gerätehersteller	Lenovo
Gerätebezeichnung	Thinkpad T42p
Gerätetyp	Notebook
Prozessor	Pentium-M
Prozesssortakt	2,1 GHz
Arbeitsspeicher	2048MB
Netzwerkgeschwindigkeit	1 Gigabit
Betriebssystem	IBM OpenClient Linux
Java-Version	IBM-Java
Applikationsserver	Websphere Application Server 6.1

Angreifer	
Gerätehersteller	Lenovo
Gerätebezeichnung	Thinkpad X60s
Gerätetyp	Notebook
Prozessor	Core Duo L2400
Prozesssortakt	1,66 GHz
Arbeitsspeicher	1024MB
Netzwerkgeschwindigkeit	1 Gigabit
Betriebssystem	Microsoft Windows XP SP2
Java-Version	Sun-Java

Web-Services Firewall	
Gerätehersteller	IBM
Gerätebezeichnung	WebSphere DataPower Integration Appliance XI50
Firmware Revision	XI50.3.6.1.3
Build	153445
Netzwerkgeschwindigkeit	100 Megabit
Gerätetyp	Appliance, 19“ 1 Höheneinheit

Die *IBM WebSphere DataPower Integration Appliance XI50*[5] ist eine in sich geschlossene Appliance. Die DataPower XI50 bietet die Funktionalität eines Enterprise Service Bus, Nachrichten können in verschieden Formate Transformiert werden. Dabei ist unerheblich, ob das Nachrichtenformat ein Binär-, XML- oder Legacy-Format darstellt. Zusätzlich bietet die IX50 Routing- und Sicherheitsfunktionalität des WebSphere DataPower XML Security Gateway XS40. Die Funktionalität im Bereich Sicherheit umfasst: WS-Security, XML-Encryption, XML/SOAP-Firewall Filtering, XML-Signature, XML-Schema validation, Zwei-Wege-SSL, XML Berechtigungskontrolle, XPath und detailliertes logging.

Der Zugriff auf die XI50 geschieht durch ein webbasiertes Management-Frontend.

In dem Kapitel Realisierung wird die Software zum Erstellen und Durchführen der Angriffe näher Erläutert. Siehe Kapitel 6.1

Netzwerk

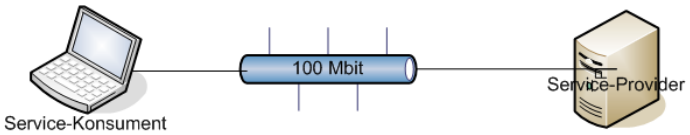

Abbildung 3.1: Versuchsaufbau ohne Web-Service Firewall

Abbildung 3.2: Versuchsaufbau mit Web-Service Firewall

Die Abbildung 3.1 zeigt den Versuchsaufbau bei dem der Angreifer direkt mit dem Web-Service kommuniziert. Der Versuchsaufbau wird entsprechend der Abbildung 3.2 verändert, so dass der Angreifer über die XML-Firewall mit dem Web-Service kommuniziert. Die XML-Firewall stellt eine Service-Virtualisierung bereit, über die der Web-Service stellvertretend angesprochen wird.

[5] http://www-306.ibm.com/software/integration/datapower/

3.6.3 Vorgehen

Der exemplarische Web-Service wird erstellt und in Form eines *Enterprise Application Archive* (EAR) auf dem *WebSphere Application Server 6.1* installiert. Mehr dazu in Kapitel 3.7.2.

Im Kapitel 3.7 „Konzeption von Angriffen“ werden die Angriffe gegen den exemplarischen Web-Service Konzipiert.

In dem Kapitel 4 „Realisierung“ finden die Versuchsdurchführungen, gemäß den im folgenden Absatz dargestellten Vorgaben, statt. Die Metriken der Versuchsläufe werde protokolliert und ausgewertet.

Vorgaben des Versuchsablaufes

Ein Angriffsversuch besteht aus folgenden Abläufen:

- Testumgebung neu initialisieren.
- SOAP-Requests gemäß der Angriffstechnik erstellen.
- SOAP-Requests an den Web-Service-Endpunkt senden.
- SOAP-Reply des Web-Service auswerten.

Um Seiteneffekte durch vorherige Versuche auszuschließen, wird der Applikationsserver vor jedem Versuchslauf neu gestartet.

Metriken

Während des Angriffes werden folgende Metriken aufgezeichnet:

- SOAP-Nachricht
  - Typ und Wert der Parameter
  - Nachrichtengröße in Byte
- Systemumgebung
  - Prozessorauslastung in Prozent
  - Hauptspeicherauslastung in MByte
- Applikationsserver
  - Verarbeitungszeit in Millisekunden
  - Reaktionsfähigkeit

Sowie die Verfügbarkeit des Applikationsservers nach einem Angriff.

3.7 Konzeption der Angriffe

An dieser Stelle werden konkrete Angriffe für die Angriffstechniken dargestellt. Die Anwendungslogik des exemplarischen Web-Services wird, gefolgt von dem exemplarischen SOAP-Request und seine Methodenaufrufen, vorgestellt. Danach wird die Notation, die bei der Darstellung der konzeptionellen Angriffe Verwendung findet gezeigt. Das Kapitel schließt ab, mit der Konzeption von Angriffen durch den XML-Inhalt und die XML-Struktur.

3.7.1 Quelltext der Anwendungslogik

Der Java-Quelltext der Anwendungslogik des examplarischen Web-Services wird im folgend gezeigten Quelltext 3.1 abgebildet:

```
package myPackageNamespace;

public class MyServices {

    public String reverse(String s)
    {
        StringBuffer sb = new StringBuffer();
        for (int i = s.length()-1; i >= 0; i--){
            sb.append(s.charAt(i));
        }
        return sb.toString();
    }
    public int fract(int a, int b)
    {
    return a/b;
    }
    public long countStringArray(String[] s)
    {
        return s.length;
    }
}
```

Quelltext 3.1: Anwendungslogik der exemplarischen Web-Services

Die Methode *reverse* verwendet als Parameter und Rückgabewert ein String-Objekt. In Java kennt dieses Objekt, nur eine Längenbeschränkung durch Methoden, die Zeichen innerhalb des Strings durch int primitive adressieren und die Größe des verfügbaren Heapspeichers. Das Objekt wurde gewählt um mögliche Pufferüberläufe in dem XML-Parser der *Web-Service Engine* provozieren zu können.

Die Methode *fract* verwendet den primitiven Datentypen *int* als Parameter und Rückgabewert. Der primitive Datentyp *int* hat die Länge von 4 Byte und deckt den Wertebereich von -2^{31} bis $2^{31} - 1$ ab ($-2147483648 \ldots 2147483647$). Der Datentyp *int* wurde gewählt um eine Verletzungen durch Wertebereichsüberlaufe testen zu können.

Die Methode *countStringArray* verwendet als Parameter ein Low-Level String-Array. Der Rückgabewert zeigt die Größe des String-Arrays, dass im Zuge der Objekterzeugung gemäß dem *XML to Java Object Mappings* erzeugt wurde.

3.7.2 Generierung des Web-Services

Zur Generierung des Web-Service wurde der Wizard des *IBM Rational Application Developer* (RAD) *7.5 beta* verwendet. Der RAD setzt auf das *Eclipse Framework*[6] auf. Der „Web-Service“-Wizard ist in einer funktionsreduzierten Form Bestandteil der *Eclipse Web Tools Platform*[7] (WTP).

Aus der Java-Klasse wurde im Bottom-Up Verfahren die WSDL-Beschreibung generiert. Der Wizard erzeugt das benötigte EAR-Paket und liefert es auf den *WebSphere Application Server* aus. Der Web-Service steht ab diesem Zeitpunkt unter der URL des WebServiceEndpoints: „http://localhost:9081/myWS/services/MyServices“ zur Verfügung.

Es ist darauf Hinzuweisen, dass Klassennamen mit einem Großbuchstaben beginnen müssen, um vom Wizard verarbeitet zu werden. Ferner muss die Klasse Instanzmethoden besitzen, da nur solche Methoden als Service-Methoden Verwendung finden können.

3.7.3 SOAP-Nachricht

Quelltext 3.2 zeigt den exemplarischen SOAP-Request. Um den Web-Service zu verwenden, muss dieser SOAP-Request über HTTP an den WebServiceEndpoint gesendet werden.

[6] http://www.eclipse.org/
[7] http://www.eclipse.org/webtools/

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:ns1="http://myPackageNamespace"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header/>
  <soapenv:Body>
    ...
  </soapenv:Body>
</soapenv:Envelope>
```

Quelltext 3.2: Exemplarischer SOAP-Request

Der Methodenaufruf einer Service-Methode wird in den soapenv:Body des Quelltext 3.2 eingebettet und kann für die entsprechende Methode anstelle der ... eingesetzt werden.

Im folgenden werden die Methodenaufrufe in den Quelltexten 3.3, 3.4 und 3.5 exemplarisch dargestellt:

```
    <ns1:reverse>
      <s>any string</s>
    </ns1:reverse>
```

Quelltext 3.3: Methodenaufruf: reverse

```
    <ns1:fract>
      <a>10</a>
      <b>20</b>
    </ns1:fract>
```

Quelltext 3.4: Methodenaufruf: fract

```
    <ns1:countStringArray>
      <soapenc:Array soapenc:arrayType="xsd:String[1]">
       <ns1:s xsi:type="xsd:string">any string</ns1:s>
      </soapenc:Array>
     </ns1:countStringArray>
```

Quelltext 3.5: Methodenaufruf: countStringArray

3.7.4 Notation der Darstellung

Die Darstellung der konzeptionellen Angriffe enthält Platzhalter, die dynamische Inhalte beschreiben. Sie entsprechen folgender Konvention:

- (X) - Eine Zahl, Stellvertretend für einen Wert oder eine Anzahl.
- {x}(X) - Das angegebene Zeichen (in diesem Fall 'x') wird X fach wiederholt.
- {*}(X) - Ein, durch geschweifte Klammern, gruppierter Inhalt wird X fach wiederholt.

Um die Parameter von den XML-Anweisungen abzuheben, wurden geschweifte und runde Klammern gewählt. Sie gehören nicht zu den Metazeichen der XML-Auszeichnungssprache. Die Zusammengehörigkeit der Klammern ist angelehnt an die Programmiersprachen Java und C.

Anwendungs-Beispiele

Der Quelltext 3.6 stellt die Beschreibung eines dynamischen Inhaltes dar.

```
{<s>{x}(X)<s>}(Y)
```

Quelltext 3.6: Verwendete Notation: Beispiel 1

Das Zeichen 'x' wird X fach wiederholt und entspricht damit einem String der Länge X. Während Y die Wiederholungen der Gruppierung <s>{x}(X)</s> darstellt und dadurch die Anzahl der Elemente ausdrückt.

Quelltext 3.7 zeigt die Ausgabe des Quelltext 3.6 bei der Verwendung von X=3 und Y=2.

```
<s>xxx</s>
<s>xxx</s>
```

Quelltext 3.7: Verwendete Notation: Ergebnis Beispiel 1

Das nächste Beispiel aus Quelltext 3.8 zeigt X als Parameter für die Wiederholungen der Gruppierung als auch als Laufvariable innerhalb einer Wiederholung.

```
{<s>any string(X)</s>\}(X)
```

Quelltext 3.8: Verwendete Notation: Beispiel 2

Der Quelltext 3.9 zeigt die Ausgabe des Quelltext 3.8 bei der Verwendung von X=3.

```
<s>any string1</s>
<s>any string2</s>
<s>any string3</s>
```

Quelltext 3.9: Verwendete Notation: Ergebnis Beispiel 2

Die Verwendung der jeweiligen Platzhalter wird in der zugehörigen Tabelle zum Quelltext eines Angriffes näher erläutert.

3.7.5 Angriffe durch den XML-Inhalt

Bei diesen Angriffstechniken wird die SOAP-Nachricht um schadhafte Bestandteile angereichert, bzw. vorhandene Bestandteile werden schadhaft parametrisiert. Als Grundlage der Angriffe gilt immer der exemplarische SOAP-Request. Er wird um die charakteristischen Bestandteile des jeweiligen Angriffes erweitert.

Buffer Overflow

Bei dieser Angriffstechnik wird beabsichtigt den vorgesehenen Puffer des XML-Parsers für den übergebenen Parameter zu überlasten.

Quelltext 3.10 zeigt die konzeptionelle Darstellung eines Buffer Flow Angriffes. Die Tabelle 3.5 zeigt die entsprechende Verwendung des Platzhalters.

Der Angriff soll mögliche Pufferüberlauf durch große Element-Inhalte ermitteln.

```
...
  <soapenv:Body>
    <ns1:reverse>
      <s>{x}(X)</s>
    </ns1:reverse>
  </soapenv:Body>
...
```

Quelltext 3.10: Konzeptioneller Angriff: Buffer Overflow

Variable	Art	Verwendung	Wertebereich	Empfehlung
{x}(X)	String	Beliebig langer Inhalt	≥ 1	≥ 1

Tabelle 3.5: Parameter des konzeptionellen Angriffes: Buffer Overflow

Datatype Range Breach

Dieser Angriff soll zeigen, wie der XML-Parser auf Werte außerhalb des Wertebereichs eines Datentypen reagiert. Das Ziel ist eine Wertebereichsverletzung des verwendeten Datentypen int ($>2^{31}$) zu provozieren.

Quelltext 3.11 zeigt die konzeptionelle Darstellung eines Datatype Range Breach Angriffe. Die Tabelle 3.9 zeigt die entsprechende Verwendung der Platzhalter.

Der Angriff soll die Behandlung von Wertebereichsverletzungen zeigen.

```
...
  <soapenv:Body>
    <ns1:fract>
      <a>(X)</a>
      <b>(Y)</b>
    </ns1:fract>
  </soapenv:Body>
...
```

Quelltext 3.11: Konzeptioneller Angriff: Wertebereichsverletzung

Variable	Art	Verwendung	Wertebereich	Empfehlung
(X)	Wert	Wertebereichsverletzung	≥ 1	$\geq 2^{31}$
(Y)	Wert	Wertebereichsverletzung	≥ 1	$\geq 2^{31}$

Tabelle 3.6: Parameter des konzeptionellen Angriffes: Datatype Range Breach

Extra-long Names

Bei dieser Angriffstechnik wird beabsichtigt den Puffer des XML-Parsers für die Tag-Namen, Attribut-Namen und Namensräume zu überlasten.

Extra-Long Tagnames

Quelltext 3.12 zeigt die konzeptionelle Darstellung eines Extra-Long Tagnames Angriffes. Die Tabelle 3.7 zeigt die entsprechende Verwendung des Platzhalters.

Der Angriff soll einen möglichen Pufferüberlauf durch einen überlangen Tag-Namen ermittelt.

```
...
  <soapenv:Body>
    <ns1:reverse>
      <{x}(X)>content</{x}(X)>
    </ns1:reverse>
  </soapenv:Body>
...
```

Quelltext 3.12: Konzeptioneller Angriff: Extra-long Tagnames

Variable	Art	Verwendung	Wertebereich	Empfehlung
{x}(X)	String	Beliebig langer Tag-Name	≥ 1	≥ 1

Tabelle 3.7: Parameter des konzeptionellen Angriffes: Extra-long Tagnames

Extra-long Attributnames

Quelltext 3.13 zeigt die konzeptionelle Darstellung eines Extra-long Attributnames Angriffes. Die Tabelle 3.8 zeigt die entsprechende Verwendung des Platzhalters.

Der Angriff soll einen möglichen Pufferüberlauf durch einen überlangen Attribut-Namen ermittelt.

```
...
  <soapenv:Body>
    <ns1:reverse {x}(X)="any value">
      <s>content</s>
    </ns1:reverse>
  </soapenv:Body>
...
```

Quelltext 3.13: Konzeptioneller Angriff: Extra-long Attributenames

Variable	Art	Verwendung	Wertebereich	Empfehlung
{x}(X)	String	Beliebig langer Attribut-Name	≥ 1	≥ 1

Tabelle 3.8: Parameter des konzeptionellen Angriffes: Extra-long Attributenames

Extra-long Namespaces

Quelltext 3.14 zeigt die konzeptionelle Darstellung eines Extra-long Namespaces Angriffes. Die Tabelle 3.9 zeigt die entsprechende Verwendung des Platzhalters.

Der Angriff soll einen möglichen Pufferüberlauf durch einen überlangen Namensraum ermittelt.

```
...
<soapenv:Envelope  xmlns:{x}(X)="http://myPackageNamespace">
  <soapenv:Body>
    <{x}(X):reverse>
      <s>content</s>
    </{x}(X):reverse>
  </soapenv:Body>
...
```

Quelltext 3.14: Konzeptioneller Angriff: Extra-long Namespaces

Variable	Art	Verwendung	Wertebereich	Empfehlung
{x}(X)	String	Beliebig langer Namespace	≥ 1	≥ 1

Tabelle 3.9: Parameter des konzeptionellen Angriffes: Extra-long Namespaces

Namepspace Prefix Attack

Eine Namespace Prefix Attack beabsichtigt den Puffer des XML-Parsers bei der Verarbeitung von Attributen zu überlasten und dadurch die Zuweisung des Namensraums zu verzögern oder verhindern.

Attributename and -content

Quelltext 3.15 zeigt die konzeptionelle Darstellung einer Namespace Prefix Attack, realisiert durch eine Vielzahl von langen Attribut-Namen und -Inhalten. Die Tabelle 3.10 zeigt die entsprechende Verwendung der Platzhalter.

```
...
  <soapenv:Body>
    <ns1:reverse { {x}(X)="{y}(Y)" }(Z) >
      <s>any string</s>
    </ns1:reverse>
  </soapenv:Body>
...
```

Quelltext 3.15: Konzeptioneller Angriff: Namespace Prefix Attack (Attributename and -content)

Variable	Art	Verwendung	Wertebereich	Empfehlung
{x}(X)	String	Beliebig langer Attribut-Name	≥ 1	≥ 1
{y}(Y)	String	Beliebig langer Attributinhalt	≥ 1	≥ 1
{*}(Z)	Element	Attribut Wiederholungen	≥ 1	≥ 1

Tabelle 3.10: Parameter des konzeptionellen Angriffes: Namespace Prefix Attack (Attributename and -conten)

Namespace Declarations

Quelltext 3.16 zeigt den konzeptionellen Angriff einer Namespace Prefix Attack durch eine Vielzahl von Namespace Deklarationen. Die Tabelle 3.11 zeigt die entsprechende Verwendung des Platzhalters.

Der Angriff soll einen Pufferüberlauf im XML-Parser erzeugen.

```
...
  <soapenv:Body>
    <ns1:reverse>
      <s>content</s>
    </ns1:reverse>
    <attack
       {xmlns:x(X)]="http://myPackageNamespace/x(X)"}(X)
    ></attack>
  </soapenv:Body>
...
```

Quelltext 3.16: Konzeptionller Angriff: Namespace Prefix Attack (Namespace Declarations)

Variable	Art	Verwendung	Wertebereich	Empfehlung
(X)	Laufvariable	Aktueller Wert	≥ 1	≥ 1
{*}(X)	Element	Namespace Deklarationen	≥ 1	≥ 1

Tabelle 3.11: Parameter des konzeptionellen Angriffes: Namespace Prefix Attack (Namespace Declarations)

Coercive Parsing

Bei diesem Angriff enthält der SOAP-Request stark verschachtelte Strukturen, die darauf abzielen schwer parsbar zu sein. Ziel des Angriffes ist es den zur Verfügung stehenden Speicher des XML-Parser zu überlasten.

Quelltext 3.17 zeigt die konzeptionelle Darstellung eines Coercive Parsing Angriffes. Die Tabelle 3.12 zeigt die entsprechende Verwendung der Platzhalter.

Im Falle einer Rekursion wird „z(Z)“ entsprechend der Knotentiefe „{*}(T)“ durch „<y(Y)>z(Z)</y(Y)>“ ersetzt.

```
<soapenv:Envelope  xmlns:{x}(X)="http://myPackageNamespace" >
  ...
  <soapenv:Body>
    <{x}(X):reverse { {v}(V)="{w}(W)" }(U)>
      {<{y}(Y)>{z}(Z)</{y}(Y)>}(S)
    </{x}(X):reverse>
  </soapenv:Body>
...
```

Quelltext 3.17: Konzeptioneller Angriff: Coercive Parsing

Die Wahl der Werte für Knotentiefe (S) und Knotenbreite (T) sollte mit bedacht gewählt werden. Die Anzahl der „reverse“-Elemente steigt exponentiell um $(S)^{(T)}$ an.

Variable	Art	Verwendung	Wertebereich	Empfehlung
{*}(S)	Element	Knote auf einer Ebene	≥ 1	≥ 1
{*}(T)	Element	Knoten in die Tiefe	≥ 1	≥ 1
{*}(U)	Element	Attribut Wiederholungen	≥ 1	≥ 1
{v}(V)	String	Beliebig langer Attribut-Name	≥ 1	≥ 1
{w}(W)	String	Beliebig langer Attribut-Inhalt	≥ 1	≥ 1
{x}(X)	String	Beliebig langer Namespace	≥ 1	≥ 1
{y}(Y)	String	Beliebig langer Tag-Name	≥ 1	≥ 1
{z}(Z)	String	Beliebig langer Inhalt	≥ 1	≥ 1

Tabelle 3.12: Parameter des konzeptionellen Angriffes: Coercive Parsing

Flooding Attack

Bei diesem Angriff werden mehrere SOAP-Requests parallel an den Service gesendet. Das Ziel ist festzustellen, ob und ab welcher Anzahl von gleichzeitigen Anfragen die Verfügbarkeit des Web-Service kompromittiert wird. An dieser Stelle wird kein eigener Quelltext dargestellt, da diese Angriffstechnik die Angriffe der anderen Angriffstechniken verwendet.

XML Document Size Attack

Dieser Angriff ist durch die oberen Angriffe bereits abgedeckt. Unter Verwendung der Angriffe können beliebig große Dokumente erzeugt werden. Eine weitere Möglichkeit stellt das einbetten großer Anhänge mit mehreren hundert Megabyte Größe in den SOAP-Request dar.

3.7.6 Angriffe durch die XML-Struktur

Die Angriffe dieses Bereiches verwenden dynamische Elemente, die innerhalb der SOAP-Nachricht definiert sind. Sie zielen auf eine Auslastung des zur Verfügung stehenden Hauptspeichers und Prozessors.

Als Grundlage der Angriffe gilt auch hier immer der exemplarische SOAP-Request. Die Varianten der Entity Expansion Angriffe werden vollständig abgebildet, die relevanten Teile sind über die gesamte Nachricht verteilt. Während die Varianten der SOAP Array Attack, jeweils nur die charakterisierenden Bestandteile des Angriffes enthalten.

XML Entity Expansion

Der XML-Parser ersetzt während der Entity-Auflösung eine hohe Anzahl Referenzen auf einen überlangen Entity-Inhalt.

Während des Parse-Vorgangs löst der XML-Parser Entity-Referenzen auf. Dabei ersetzt er dynamisch die Entity-Referenzen durch den konkreten Wert der Entity (siehe [XML, 2006, Kapitel 4.5]).

Quelltext 3.18 zeigt die konzeptionelle Darstellung eines XML Entity Expansion Angriffes. Die Tabelle 3.13 zeigt die entsprechende Verwendung des Platzhalters.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE s [<!ENTITY "{x}(X)">]>
<soapenv:Envelope xmlns:ns1="http://myPackageNamespace"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:reverse>
            <s>{&x;}(Y)</s>
        </ns1:reverse>
    </soapenv:Body>
</soapenv:Envelope>
```

Quelltext 3.18: Konzeptioneller Angriff: XML Entity Expansion

Variable	Art	Verwendung	Wertebereich	Empfehlung
{x}(X)	String	Beliebig langer String	≥ 1	$\geq 10^5$
{*}(Y)	Element	Referenz-Wiederholungen	≥ 1	$\geq 3 * 10^4$

Tabelle 3.13: Parameter des konzeptionellen Angriffes: XML Entity Expansion

XML Recursive Entity Expansion

Der XML-Parser ersetzt während der Entity-Auflösung tief verschachtelte Referenzen auf eine Entity und verbraucht den zur Verfügung stehenden Hauptspeicher.

Quelltext 3.19 zeigt die konzeptionelle Darstellung eines XML Recursive Entity Expansion Angriffes. Die Tabelle 3.14 zeigt die entsprechende Verwendung der Platzhalter.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE s[
<!ENTITY x0 "ha!">
```

```
{<!ENTITY x(X) "&x(X-1);&x(X-1)">}(X)
]>
<soapenv:Envelope xmlns:soapenv="..." xmlns:ns1="...">
    <soapenv:Header>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:reverse>
            <s>&x(X);</s>
        </ns1:reverse>
    </soapenv:Body>
</soapenv:Envelope>
```

Quelltext 3.19: Konzeptioneller Angriff: XML Recursive Entity Expansion

Variable	Art	Verwendung	Wertebereich	Empfehlung
(X)	Laufvariable	Aktueller Wert	≥ 1	≥ 100
{*}(X)	Element	Entity Rekursion	≥ 0	≥ 100

Tabelle 3.14: Parameter des konzeptionellen Angriffes: XML Recursive Entity Expansion

Die Anzahl der Rekursionen entspricht $2^{(X)}$ Wiederholungen des Strings x0.

SOAP Array Attack

Die Low-level Arrays der Anwendungslogik werden in SOAP-Nachrichten durch SOAP-Arrays repräsentiert. Während der Objekterzeugung wird, entsprechend der Größe des SOAP-Arrays, Hauptspeicher für die Daten des Arrays belegt. Bei der Verwendung eines SOAP-Arrays muss das Attribut „arrayType“ angegeben werden, dass Datentyp und Anzahl der enthaltenen Elemente wiederspiegelt.

SOAP arrays MUST contain a „SOAP-ENC:arrayType“ attribute whose value specifies the type of the contained elements as well as the dimension(s) of the array [SOAP11, 2000, Kapitel 5.1].

Findet die Bestimmung der Arraygröße lediglich unter Verwendung des „arrayType“-Attributes statt, wird bei der Objekterzeugung ein entsprechend großes Array angelegt - unabhängig von der tatsächliche Anzahl übermittelter Elemente. So kann es beispielsweise möglich sein, dass bei der Objekterzeugung im Zuge des *XML to Java Objekt Mappings* Speicherplatz für 100000 Elemente reserviert und belegt wird, obwohl tatsächlich nur 1 Element übermittelt wurde.

Quelltext 3.20 zeigt die konzeptionelle Darstellung einer SOAP Array Attack. Die Tabelle 3.15 zeigt die entsprechende Verwendung der Platzhalter.

```
...
   <soapenv:Body>
      <ns1:countStringArray>
         <soapenc:Array soapenc:arrayType="xsd:String[(Y)]">
            {<ns1:s xsi:type="xsd:string">any string</ns1:s>}(X)
         </soapenc:Array>
      </myp:countStringArray>
   </soapenv:Body>
...
```

Quelltext 3.20: Konzeptioneller Angriff: SOAP Array Attack

Variable	Art	Verwendung	Wertebereich	Empfehlung
{*}(X)	Element	Array Element	≥ 1	≥ 1
(Y)	Wert	Array-Größe	≥ 0	$\geq 10^7$

Tabelle 3.15: Parameter des konzeptionellen Angriffes: SOAP Array Attack

Wird die Anzahl des „arrayType“-Attributes ausgewertet, wird für (Y)-Elemente Hauptspeicher reserviert, wenn (Y)>(X). Ansonsten, entsprechend der Anzahl übermittelter Array-Elemente, für (X)-Elemente.

Sparse Arrays

Eine Besonderheit stellen „Sparse Arrays“ dar. Ihre Elemente können innerhalb eines SOAP Arrays ab einem festgelegten Indize beginnen oder eine feste Position innerhalb des Arrays erhalten. Die Angabe eines „SOAP-ENC:offset“ legt fest, ab welchem Indize das erste Element des Arrays beginnt. Die Position eines Elementes kann mit „SOAP-ENC:position“ innerhalb des Arrays zugewiesen werden.

A SOAP array member MAY contain a „SOAP-ENC:offset“ attribute indicating the offset position of that item in the enclosing array. [SOAP11, 2000, Kapitel 5.1]

Likewise, an array member MAY contain a „SOAP-ENC:position“ attribute indicating the position of that item in the enclosing array. [SOAP11, 2000, Kapitel 5.1]

Die leeren Elemente eines Sparse Arrays werden in der SOAP-Nachricht nicht übermittelt. Erst im Zuge der Objekterzeugung des Arrays wird das Offset und die Position ausgewertet. Das angelegt Array hat eine Größe entsprechend dem Offset bzw. der Position und den übertragenen Elementen.

Sparse Arrays with Offset

Quelltext 3.21 zeigt die konzeptionelle Darstellung einer SOAP Array Attack durch ein Sparse Array, welche ein Offset für das Array verwendet. Die Tabelle 3.16 zeigt die entsprechende Verwendung der Platzhalter.

```
...
   <soapenv:Body>
      <ns1:countStringArray>
         <soapenc:Array
          soapenc:offset="[(Z)]"
          soapenc:arrayType="xsd:String[(Y)]"
         >
           {<xsd:string>any string</xsd:string>}(X)
         </soapenc:Array>
      </ns1:countStringArray>
    </soapenv:Body>
...
```

Quelltext 3.21: Konzeptioneller Angriff: SOAP Array Attack (Sparse Array with Offset)

Variable	Art	Verwendung	Wertebereich	Empfehlung
{*}(X)	Element	Array Element	≥ 1	≥ 1
(Y)	Wert	Array-Größe	≥ 0	$\geq 10^7$
(Z)	Wert	Position des ersten Elementes	≥ 0	$\geq 10^7$

Tabelle 3.16: Parameter des konzeptionellen Angriffes: SOAP Array Attack (Sparse Array with Offset)

Wird die Anzahl des „arrayType“-Attributes ausgewertet, wird für (Y)-Elemente Hauptspeicher reserviert, wenn (Y)>(X)+(Z). Ansonsten für (X)+(Z)-Elemente, die der Anzahl vorhandener Array-Elemente ab dem angegebenen Offset entspricht.

Sparse Arrays wit Position

Quelltext 3.22 zeigt den konzeptionellen Angriff der SOAP Array Attack durch ein Sparse Array, bei dem das erste Element die angegebene Position erhält. Die Tabelle 3.17 zeigt die entsprechende Verwendung der Platzhalter.

```
...
  <soapenv:Body>
    <myp:countStringArray>
      <soapenc:Array soapenc:arrayType="xsd:String[(Y)]">
        <soapenc:string soapenc:position="[(Z)]">
           any string
        </soapenc:string>
```

```
        {<soapenc:string>any string</soapenc:string>}(X)
      </soapenc:Array>
    </myp:countStringArray>
  </soapenv:Body>
...
```

Quelltext 3.22: Konzeptioneller Angriff: SOAP Array Attack (Sparse Array with Position)

Variable	Art	Verwendung	Wertebereich	Empfehlung
{*}(X)	Element	Array Element	≥ 1	≥ 1
(Y)	Wert	Array-Größe	≥ 0	$\geq 10^7$
(Z)	Wert	Position des ersten Elementes	≥ 0	$\geq 10^7$

Tabelle 3.17: Parameter des konzeptionellen Angriffes: SOAP Array Attack (Sparse Array with Position)

Wird die Anzahl des „arrayType“-Attributes ausgewertet, wird für (Y)-Elemente Hauptspeicher reserviert, wenn (Y)>(X)+(Z). Ansonsten für (X)+(Z)-Elemente, die der Anzahl vorhandener Array-Elemente ab der angegebenen Position entspricht.

In der Version 1.2 der SOAP Spezifikation ist die Unterstützung für parse-Arrays nicht vorgesehen.

The support provided in SOAP 1.1 for partially transmitted and sparse arrays is not available in SOAP 1.2. [SOAP12, 2007, Kapitel 6]

3.8 Zusammenfassung

Die Angriffe auf Web-Services finden durch manipulierte SOAP-Nachrichten statt. Mittels Angriff auf die die Berechtigung versucht ein Angreifer unautorisierte Berechtigung zu einem Web-Service zu erhalten. Angriffe auf die Daten-Integrität und Vertraulichkeit zielen primär auf die Manipulation von Daten der XML-Nachricht ab bzw. auf die unautorisierter Gewinnung von vertraulichen Daten. Angriffe auf die Systemumgebung versuchen Manipulationen am Verhalten selbiger zu erreichen. Der Einsatz der Basis-Schutz-Mechanismen der Web-Service-Security Spezifikation deckt diese Schutzziele ab.

Das Schutzziel Verfügbarkeit wird nicht direkt von WS-Security adressiert. Dennoch wirkt WS-Security unterstützend bei der Einhaltung dieses Schutzziels, beispielsweise durch die Anwendung digitaler Signaturen auf XML-Schemas, deren Daten-Integrität und Verbindlichkeit somit gewährleistet wird.

Gerade dieses Schutzziel stellt einen grundlegenden Aspekt beim Betrieb von Web-Services dar. Sind Erreichbarkeit und Reaktionszeit Teil eines Service Level Agreements, kann ein Denial of Service zu hohen Vertragsstrafen führen. Bei einem kostenpflichtigen Service gehen Probleme mit der Verfügbarkeit einher mit Verdienstausfall.

Diese Arbeit beschränkt sich bei der Konzeption und Realisierung von Angriffen auf die Betrachtung der Angriffstechniken, die auf die Verfügbarkeit von Web-Services abzielen, die implementationsspezifische Eigenschaften des Applikationsservers für die Realisierung eines Angriffes bedingen.

Der Versuchsaufbau zeigt die Versuchsumgebung und beschreibt das geplante Vorgehen bei den Versuchen. Im Anschluß wird die Konzeption der Angriffe dargestellt. Hierfür wird zunächst der Quelltext des Verwendeten Web-Services gezeigt und in die Notation eingeführt, die bei der Beschreibung der Konzeptionellen Angriffe verwendet wird. Abschließen werden die konzipierten Angriffe dargestellt.

Die konzeptionellen Angriffe dienen als Grundlage für die Realisierung. Im nächsten Kapitel finden ihre konkreten Ausprägungen Verwendung bei der Durchführung der Angriffe.

4 Realisierung

Dieses Kapitel beschäftigt sich mit der Durchführung der konzipierten Angriffe. Zunächst kommt eine kurze Vorstellung der verwendeten Software. Im Anschluß findet die Durchführung und Messung der Angriffsversuche statt. Beginnend mit den direkten Angriffen auf den Web-Service, folgen dann die Angriffe auf den Web-Service hinter der XML-Firewall.

4.1 Verwendete Software

An dieser Stelle werden die Programme zum Erstellen und Durchführen der Angriffe vorgestellt. Zuerst wird auf bereits verfügbare Software eingegangen und dann die Eigenentwicklung erläutert.

4.1.1 Verfügbare Software

Eviware soapUI

Zum Einsatz kommt die freie Open Source Anwendung *soapUI 2.0.2*[1] der Firma *Eviware Software AB*[2]. Die Anwendung unterstützt bei Inspektion, Aufruf und Entwicklung von Web-Services. Daneben bietet *soapUI* die Möglichkeit sowohl Funktionale-, als auch Last-Tests von Web-Services durchzuführen und diese auszuwerten. Nach dem Einbinden von digitalen Zertifikaten stehen Funktionen zum Signieren und Verschlüsseln durch Web-Service-Security zur Verfügung. Die Anwendung bietet einen konsistenten Umgang mit WSDL-Beschreibungen. Sie generiert Beispiel-Requests, übernimmt die URL des WebServiceEndpoint und bestimmt die SOAP-Action anhand dieser. Die Anwendung soapUI ist für Windows und Linux Systeme erhältlich.

soapUI steht als Plugin für die Entwicklungsumgebungen Eclipse, IntelliJ IDEA und NetBeans zur Verfügung.

[1] http://www.soapui.org/
[2] http://www.eviware.com

Nach Eviware: *It is mainly aimed at developers and testers providing or consuming Web Services (Java, .net, etc). Functional and Load Testing can be done both interactively in soapUI or within an automated build or integration process using the soapUI command line tools.*

Mit dieser Software wurden die SOAP-Requests entwickelt und auf Validität geprüft. In Zuge der Entwicklungsphase konnte die Akzeptanz der SOAP-Requests durch den Applikationsserver getestet werden.

Vordel SOAPbox

Daneben wurde die Evaluationsversion des Produktes *SOAPbox* der Firma *Vordel*[3] getestet. Die Anwendung unterstützt ebenfalls die Inspektion, den Aufruf und die Entwicklung von Web-Services. Funktionale- und Last-Tests sind nicht vorgesehen. Die Versionen einzelner Web-Service Spezifikationen können durch die Oberfläche verändert werden. *SOAPbox* unterstützt ebenfalls das Signieren und Verschlüsseln gemäß Web-Service-Security, allerdings gelang das Einbinden von Zertifikaten nicht. Eine automatische Request-Generierung anhand der WSDL-Beschreibung ist nicht vorgesehen. Die Evaluationsversion ist auf das absenden von 5 Requests, je Programmstart, beschränkt. Der Downloadlink wird erst nach kostenloser Registrierung per Email zugestellt. *SOAPbox* ist für Windows und Linux Systeme erhältlich.

Apache JMeter

Eine weitere Alternative ist die Open Source Anwendung *Apache JMeter 2.3.1* des *Apache Jakarta Project*. Die Anwendung ist auf Last-Tests für HTTP und FTP Server spezialisiert. Die Multithreading-Funktion ist für Flooding-Angriffe verwendbar. Das „WebService(SOAP) Request“ Template wertet ebenfalls WSDL-Beschreibungen aus und übernimmt die Daten des WebServiceEndpoint und der SOAP-Action. Die Verwendung von *JMeter* kann nicht als intuitiv bezeichnet werden. Eine Funktion zum generieren von SOAP-Requests gemäß der WSDL-Beschreibung gibt es nicht. Die Software ist für Windows, Linux und weitere Systeme erhältlich.

[3] http://www.vordel.com

Einschränkungen

Diese Anwendungen sind auf den Versand von statischen SOAP-Requests beschränkt. Eine Möglichkeit Parameter bezüglich der Häufigkeit von Elementen oder ihren Werten festzulegen ist nicht gegeben.

soapUI bietet im Kontext seiner Test-Läufe die Möglichkeit Einfluss auf Request und Response durch die eingebettete Scriptsprache *Groovy*[4] zu nehmen. In der Testphase ist es gelungen dynamisch Änderungen an Requests vorzunehmen. Allerdings konnte im Rahmen der Evaluation nicht identifiziert werden, wie Parameter aus der Benutzeroberfläche an die Methoden der Scriptsprache übergeben werden können. Die Parameter müssen unmittelbar im Quelltext des Scripts geändert werden.

4.1.2 Eigenentwicklung: SOAP-Generator

Zusätzlich zu den oben genannten Anwendungen wurde ein Java-Werkzeug für die Kommandozeile entwickelt, dass die Erzeugung von dynamischen SOAP-Requests erlaubt. Es ist auf den exemplarischen Web-Service zugeschnitten und funktioniert auch nur mit diesem. Eine dynamische Unterstützung weiterer Web-Services wäre durch Auswertung von WSDL-Beschreibungen möglich. Sie ist im Rahmen der Arbeit aber nicht notwendig.

Im Folgenden wird die Anwendung *SOAP-Generator* genannt.

Aus Performance Gründen verzichtet der *SOAP-Generator* auf den Einsatz von komplexen Frameworks oder APIs. Die SOAP-Requests und der darin enthaltene XML-Baum wird durch einfache Java-Klassen abgebildet. Die Kommunikation mit dem Applikationsserver geschieht durch eine HTTP-Komponenten, die zum Lieferumfang der Java-Umgebung gehört.

Verwendet wird die *Java Runtime Environment (JRE) 1.6.0_05*. Die Anwendung sollte ebenfalls unter Java 1.5 arbeiten.

Die Erstellung der SOAP-Nachrichten kann parametrisiert werden, um den benötigten Inhalt dynamisch zu erzeugen. Eine SOAP-Nachricht kann bei Bedarf eine DTD Definition enthalten. Angaben zu der SOAP-Action sind in dem HTTP-Header enthalten.

Für den SOAP-Request stehen folgende Metriken zur Verfügung:

- SOAP-Request Größe in Byte
- SOAP-Request Versand in ms
- Typ und Wert von Parametern

[4] http://groovy.codehaus.org/

Für den SOAP-Reply stehen folgende Metriken zur Verfügung:

- SOAP-Replay Empfang in ms
- Zeit-Delta zwischen SOAP-Request und SOAP-Reply in ms

Eine Alternative Implmentierung wäre unter Verwendung der *SOAP with Attachments API (SAAJ)*[5] möglich. *SAAJ* unterstützt den Aufbau des SOAP-Requests und die Kommunikation mit dem Applikationsserver. Der Name der API deutet lediglich darauf hin, dass Attachments unterstützt werden. Sie müssen in einem SOAP-Request nicht enthalten sein.

Im Rahmen der Evaluation musste der SOAP-Request aus nicht nachvollziehbaren Gründen vor dem Versand mindestens einmal auf dem Bildschirm oder in Datei ausgegeben werden. Ansonsten fehlten die SOAP-Action Angaben im HTTP-Header. Diese werden von dem Applikationsserver benötigt. Die Transformation der Daten, der Versand und der Empfang der SOAP Nachrichten mangelt es an benötigter Transparenz. Ein weiteres Problem stellt der erhöhte Speicherverbrauch während nebenläufiger Anfragen dar. Jeder Prozess verwendet seine eigene Kopie des SOAP-Request. Die Verwendung von SAAJ wurde aufgrund dessen ausgeschlossen.

4.2 Angriffsdetails

In den folgenden Abschnitten werden die Angriffe Durchgeführt, die Messwerte die dabei entstehen werden Tabellarisch dargestellt.

Eine Interpretation der Schwachstellen die zu einem Angriff geführt haben wird in Kapitel 5 behandelt. Auf Versuche bei denen der WAS aufgrund der Implementierung einer Web-Service Spezifikation mit einer Fehlermeldung reagiert werden an Ort und Stelle eingegangen.

Eine Auswahl an konkreten SOAP-Requests befindet sich im Anhang auf der CD-ROM. Sie dienen der Veranschaulichung. Eine Kopie der SOAP-Request mit abweichender Parametrisierung kann mit dem *SOAP-Generator* jederzeit erzeugt und gespeichert werden.

4.2.1 Änderungen am Versuchsablauf

Im Rahmen der Realisierung finden folgende Abweichungen im Versuchsablauf statt:

- Abweichung der serverseitigen Metriken

[5] Enthalten in der JRE 1.6.0_05 und im Java Web-Services Developer Pack

  - Messung der Prozessorauslastung in Prozent
  - Messung des Hauptspeicherauslastung in MByte
- Neustartregelung

Eine Erläuterung der Abweichungen findet in den nächsten Unterabschnitten statt.

Abweichung der serverseitigen Metriken

Die „Admin Console“ des *WebSphere Application Servers* beinhaltet das Monitoring-Werkzeug *Tivoli Performance Monitor (TVP)*. Mittels TVP lassen sich die Metriken einzelner Komponenten des Applikationsservers in festen Intervallen messen. Ereignisgesteuerte Messungen sind nicht möglich. Um Messwerte mit möglichst hoher Genauigkeit zu ermitteln, wurde die kleinste Intervalleinheit von 5 Sekunden ausgewählt.

Prozessorauslastung: Eine Auswertung der TVP Messpunkte, die nach einem Request gemessen werden, geben eine ausreichend präzise Auskunft über die Prozessorauslastung. Verwendet wird jeweils der höhere Wert zweier aufeinander folgender Messpunkte nach Eingang eines Requests.

Hauptspeicherauslastung: Ein Begrenzung des Hauptspeicher stellt bei aktuellen Systemen in der Regel nicht das Problem dar. Vielmehr liegt das Problem in einer Beschränkung des Heapspeichers[6] begründet, den die Java Virtuell Maschine für die Ausführung ihrer Anwendungen zur Verfügung hat. Die automatische Speicherbereinigung[7] arbeitet permanent im Hintergrund und hat starken Einfluss auf den Heapspeicher und die Prozessorauslastung.

Die Messungen der Heapspeicherauslastung in 5 Sekunden Intervallen lassen in Verbindung mit der automatischen Speicherauslastung keine verlässliche Aussage über den Speicherverbrauch des Web-Services innerhalb der Java Virtual Maschine zu. Auf die Messung des Hauptspeichers bzw. Heapspeichers wird aus diesem Grund verzichtet.

Eine höhere Genauigkeit der Messresultate ist bei einer ereignisgesteuerten Messung zu erwarten, bei der die Messpunkte vor, während und nach der Reqest-Verarbeitung durch XML-Parser und Objekterzeugung, sowie vor der Response-Verarbeitung liegen. Die Einschränkungen der automatischen Speicherverwaltung gelten weiterhin, sollten aber im Rahmen der ereignisgesteuerten Messung aussagekräftiger sein. Aus Komplexitätsgründen findet eine ereignisgesteuerte Messung im Rahmen der Arbeit nicht statt.

[6] Auch bekannt unter dem Namen Heap-Space.
[7] Auch bekannt unter dem Namen Garbage Collector.

Neustartregelung

Eine weitere Abweichung stellt die Neustartregelung dar. Ein manueller Neustart des Applikationsservers wird ausgeführt, sobald kein SOAP-Reply innerhalb eines Zeitfensters von 120 Sekunden nach Versand des SOAP-Requests eingeht.

Bei ausbleibendem SOAP-Reply wird die Erreichbarkeit des Web-Services ermittelt, indem die URL des WebServiceEnpoints in einem Browser aufgerufen wird. Erhält der Browser keine Antwort des Web-Services, gilt der Web-Service als nicht verfügbar. Ist der Service erreichbar, wird durch Aufruf der „Admin Console“ festgestellt, ob der Web-Service ein reguläres Verhalten bezüglich Reply-Zeiten und Prozessorauslastung aufweist. Sollte das System träge reagieren, wird es ebenfalls neu gestartet, aber als verfügbar gewertet.

4.2.2 Auswertungsschwierigkeiten

Eine Auswertung der, durch den SOAP-Generator erzeugten Ausgabedatei ist aufgrund des nicht durchgehend einheitlichen Formates, nur manuell möglich. Eine einheitliche Speicherung in einem CSV-Format, bei dem die Werte[8] durch ein Semikolon getrennt werden, hätte eine automatische Datenübernahme in andere Anwendungen wie z.B. Excel erlaubt.

Beispiel für ein entsprechendes CVS-Format:
Zeit;Angriff;bytesRequest;bytesReply;timeSend;timeReceive;timeDelta;Param1;...;Param8
10:00:00;Buffer Overflow;1442;1439;0;0;125;16;1000;;;;;;;;

Eine Anpassung der Ausgabedatei findet im Rahmen der Arbeit nicht statt. Der SOAP-Generator stellt lediglich ein Werkzeug dar, dass Angriffe durchführen und Metriken protokollieren soll. Er ist nicht Kernziel der Arbeit.

Es findet eine manuelle Datenübernahme statt, da eine Anpassung des SOAP-Generators und Neumessungen zeitlich aufwendiger sind.

4.2.3 Verwendete Metriken in der Realisierung

Die signifikanten Kenngrößen innerhalb eines Versuches sind Prozessorlast und Reply-Zeit. Die Prozessorauslastung wird im Folgenden wie oben beschrieben mit den Mitteln des TVP

[8] Dies gilt auch für nicht vorhandene Werte.

durchgeführt. Die Reply-Zeit wird durch den SOAP-Generator protokolliert. Die Werte werden durch Mittelwertbildung aus Messwerten mindestens dreier Messungen gebildet. Messdurchläufe mit stark abweichenden Werten, werden der automatischen Speicherverwaltung zugeschrieben und findet erneut statt.

Während des Angriffs werden folgende Metriken aufgezeichnet:

- SOAP-Nachricht
  - Typ und Wert der Parameter
  - Nachrichtengröße in Byte
- Systemumgebung
  - Prozessor Auslastung in Prozent
- Applikationsserver
  - Verarbeitungszeit in Millisekunden
  - Verfügbarkeit

4.3 Direkte Angriffe auf Web-Services

Bei dieser Angriffsreihe steht der Angreifer in direkter Kommunkation mit dem Web-Service. Der SOAP-Generator sendet die SOAP-Requests unmittelbar an den Service des WebSphere Aplication Servers.

4.3.1 Angriffe durch den XML-Inhalt

Buffer Overflow

Tabelle 4.1 zeigt die ermittelten Messwerte der ausgeführten Buffer Overflow Angriffe gemäß Quellcode 3.10 unter der Verwendung des Platzhalters nach Tabelle 3.5.

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{x}(X)
1	10.442	1	21	ja	10^4
2	100.442	1	68	ja	10^5
3	1.000.442	8	411	ja	10^6
4	10.000.442	46	4016	ja	10^7
5	50.000.442	58	9364	ja	$5 * 10^7$
6	80.000.442	43	9797	ja	$8 * 10^7$

Tabelle 4.1: Versuchsresultate: Buffer Overflow

Die Requests 5 und 6 erhalten als Antwort diesen SOAP-Fault: Server.generalException, unexpected exception (siehe Quelltext G.1)

Ein Begrenzung des Heapspeicher des Angreifer-Computers erlaubt keine Inhalte mit der Länge von 10^8 zu erzeugen.

Aus der Tabelle 4.1 ist ersichtlich, dass eine Kompromittierung der Erreichbarkeit, selbst bei großen SOAP-Nachrichten, nicht gegeben ist. Der Web-Service ist stets verfügbar und der Ressourcenverbrauch normalisiert sich.

Datatype Space Breach

Der Wertebereichsverletzung wird unmittelbar erkannt und mit einem SOAP-Fault beantwortet.

Extra-long Names

Im folgenden werden die Versuchsresultate der drei Varianten dargestellt:

Extra-Long Namespaces

Tabelle 4.2 zeigt die ermittelten Messwerte der ausgeführten Extra-long Namespaces Angriffe gemäß Quellcode 3.14 unter der Verwendung des Platzhalters nach Tabelle 3.9.

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{x}(X)
1	30.450	1	16	ja	10^4
2	300.450	3	86	ja	10^5
3	3.000.450	5	375	ja	10^6
4	6.000.450	100	2003	ja	$2 * 10^6$
5	9.000.450	100	-	nein	$3 * 10^6$

Tabelle 4.2: Versuchsresultate: Extra-long Namespaces

Request 4 führte in einem der Durchläufe zu einem Ausfall der Verfügbarkeit.

Aus der Tabelle 4.2 ist ersichtlich, dass die Verfügbarkeit bei Werten ab $3*10^6$ kompromittiert wird. Der Web-Service erholt sich von diesem Angriff nicht.

Extra-long Attributenames

Tabelle 4.3 zeigt die ermittelten Messwerte der ausgeführten Extra-long Attributenames Angriffe gemäß Quellcode 3.13 unter der Verwendung des Platzhalters nach Tabelle 3.8.

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{x}(X)
1	10.466	1	16	ja	10^4
2	100.466	1	26	ja	10^5
3	1.000.466	4	120	ja	10^6
4	4.000.466	74	383	ja	$4 * 10^6$
5	5.000.466	100	-	nein	$5 * 10^6$

Tabelle 4.3: Versuchsresultate: Extra-long Attributenames

Aus der Tabelle 4.3 ist ersichtlich, dass die Verfügbarkeit bei Werten ab $5*10^6$ kompromittiert wird. Der Web-Service erholt sich von diesem Angriff nicht.

Extra-long Tagnames
Tabelle 4.4 zeigt die ermittelten Messwerte der ausgeführten Extra-long Tagnames Angriffe gemäß Quellcode 3.12 unter der Verwendung des Platzhalters nach Tabelle 3.7.

#	Request(Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{x}(X)
1	20.450	2	16	ja	10^4
2	200.450	3	46	ja	10^5
3	2.000.450	6	84	ja	10^6
4	8.000.450	76	20784	ja	$4 * 10^6$
5	10.000.450	100	-	nein	$5 * 10^6$

Tabelle 4.4: Versuchsresultate: Extra-long Attributenames

Request 4 führte in einem der Durchläufe zu einem Ausfall der Verfügbarkeit, dass System reagiert anschließend träge oder war nicht erreichbar.

Aus der Tabelle 4.4 ist ersichtlich, dass die Verfügbarkeit bei Werten ab $5*10^6$ kompromittiert wird. Der Web-Service erholt sich von diesem Angriff nicht.

Namepspace Prefix Attack

Atrtributname and -content
Tabelle 4.5 zeigt die ermittelten Messwerte der Namespace Prefix Attacks mit einer hohen Anzahl von Attributen mit beliebig langen Attributnamen und -inhalten gemäß Quellcode 3.15 unter der Verwendung der Platzhalter nach Tabelle 3.10.

Bei unterschiedlichen Versuchsläufen erzeugte Request 2 diese SOAP-Faults:

- Server.generalException - unexpected exception (siehe Quelltext G.1)
- Server.generalException - SaxParseException (well-formed) (siehe Quelltext G.2)

Namespace Declarations
Aus der Tabelle 4.5 ist ersichtlich, dass eine Kompromittierung der Verfügbarkeit nicht zwangsläufig in Zusammenhang mit der Größe der SOAP-Nachricht steht. Lange Attributnamen und -inhalte in Verbindung mit einer ausreichenden Anzahl Wiederholungen führen, wie in Request 12 ersichtlich, zu einer Kompromittierung des Web-Services. Der Web-Service erholt sich von diesem Angriff nicht.

Tabelle 4.6 zeigt die ermittelten Messwerte der Namespace Prefix Attack mit einer hohen Anzahl von Namespace Deklarationen gemäß Quellcode 3.16 unter der Verwendung des Platzhalters nach Tabelle 3.11.

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{x}(X)	{y}(Y)	{*}(Z)
1	11.889.349	69	3226	ja	1	1	10^6
2	50.889.349	69	5000	ja	1	1	$4 * 10^6$
3	20.889.349	15	723	ja	10^2	10^2	10^5
4	83.889.349	80	11895	ja	10^2	10^2	$4 * 10^5$
5	100.439.349	20	9104	ja	10^3	10^3	$5 * 10^4$
6	20.007.349	8	531	ja	10^4	10^4	10^3
7	100.039.349	68	2908	ja	10^4	10^4	$5 * 10^3$
8	20.001.049	15	844	ja	10^5	10^5	10^2
9	100.003.849	45	9198	ja	10^5	10^5	10^2
10	20.000.509	12	1959	ja	10^6	10^6	10
11	100.000.749	38	8880	ja	10^6	10^6	50
12	60.000.509	100	-	nein	$4 * 10^6$	$4 * 10^6$	10

Tabelle 4.5: Versuchsresultate: Namespace Prefix Attack (Attributname and -content)

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{x}(X)
1	438.239	40	6302	ja	10^4
2	4.578.239	79	-	nein	10^5
3	47.778.239	80	18047	ja	10^6

Tabelle 4.6: Versuchsresultate: Namespace Prefix Attack (Namespace Declarations)

Eine SOAP-Nachricht mit einer Anzahl von $2 * 10^6$ Namespace Deklarationen konnte aufgrund einer Beschränkung des Heapspeichers des Clients nicht realisiert werden.

Request 2 erzeugt eine Server.generalException - unexpected exception (siehe Quelltext G.1). Das System reagiert nach dem Request träge oder ist nicht erreichbar.

Request 3 erzeugt eine Server.generalException - SaxParseException (well-formed) (siehe Quelltext G.2). Aufgrund des frühzeitig erkannten Fehlers, führt dieser Request zu weniger Ressourcenauslastung als Request 2.

Aus der Tabelle 4.3 ist zu entnehmen, dass die Verfügbarkeit bei Werten um 10^5 kompromittierbar sein kann.

Coercive Parsing

Tabelle 4.7 zeigt die ermittelten Messwerte der ausgeführten Coercive Parsing Angriffe gemäß Quellcode 3.17 unter der Verwendung der Platzhalter nach Tabelle 3.12.

Anhand der Messungen wurde ersichtlich, dass die Werte für Knotenbreite und Knotentiefe die ausschlaggebenden Werte für die Größe des SOAP-Request sind. Die anderen Werte werden der Übersichtlichkeit halber in der Tabelle vernachlässigt. Die Parameter {x}(X), {y}(Y), {z}(Z), {v}(V), {w}(W) haben bei den Messungen jeweils den Wert 1.

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{*}(T)	{*}(S)
1	1.015.961	1	31	ja	1	500
2	4.031.461	3	133	ja	1	1000
3	23.462.311	37	2547	ja	2	16
4	23.462.311	27	1636	ja	4	8
5	49.545.639	43	6052	ja	2	17
6	5.378.931	14	1104	ja	6	6
7	36.078.579	54	6597	ja	6	7

Tabelle 4.7: Versuchsresultate: Coercive Parsing Attack

Der Client-Computer war aufgrund der Beschränkung des Heapspeichers nicht in der Lage größere Nachrichten zu erzeugen.

Aus der Tabelle 4.7 wird ersichtlich, dass eine Kompromittierung der Verfügbarkeit, selbst bei großen SOAP-Nachrichten, nicht gegeben ist. Der Web-Service ist stets verfügbar und der Ressourcenverbrauch normalisiert sich.

Der Algorithmus zum erstellen der Coercive Parsing Angriffe hat sich als nicht optimal erwiesen. Eine zufällige Verteilung der Elemente innerhalb des Teilbaumes könnte unter Umständen effektiveren Schaden bei kleinerer Nutzlast erzeugen.

Flooding Attack

Bei dieser Versuchsreihe wurden parallele Angriffe durch nebenläufige Prozesse durchgeführt. Die einzelnen Requests sind der Nummer entsprechend aus der jeweiligen Tabelle des verwendeten Angriffes zu entnehmen.

Um eine optimale Darstellung der Resultate zu gewährleisten, finden sich in der Tabelle 4.8 Angaben über die Anzahl der parallelen Prozesse, die Zeit in der eine 100% Prozessorzauslastung und die Verfügbarkeit nach dem Angriff.

Um die Angriffe möglichst effizient zu halten, wurden Request ausgewählt, deren Nachrichtengröße 10^7 Byte nicht übersteigt. Die Ausnahme bildet der Buffer Overflow Request 4, er ist um 442 Byte größer und wurde der Vollständigkeit halber hinzugenommen.

Angriffsname	Request	Prozesse	CPU-Zeit (ms)	verfügbar
Buffer Overflow	2	1000	18203	ja
Buffer Overflow	3	50	124969	ja
Buffer Overflow	4	20	160047	ja
Extra-long Namespaces	2	50	-	nein
Extra-long Attributenames	2	200	9625	ja
Extra-long Attributenames	3	10	-	nein
Extra-long Tagnames	2	50	5062	ja
Extra-long Tagnames	2	200	-	nein
Extra-long Tagnames	3	10	-	nein
Namespace Prefix Attack	2	10	-	nein
Coersive Parsing	6	40	59140	ja
SOAP Array Attack	9	50	214640	ja
SOAP Array Attack	9	100	300000	ja

Tabelle 4.8: Versuchsresultate: Flooding Attack

Aus der Tabelle 4.8 wird ersichtlich, dass einzelne Angriffe die bereits unter Verwendung hoher Werte einen Denial of Service verursachen konnten, bei der Verwendung als Flooding Attack bereits mit kleineren Werten zu einem erfolgreichen Angriff führen. Entgegen der Erwartung führten die SOAP Array Attacks auch bei hoher Frequenz zu keinem Ausfall der Erreichbarkeit. Ein Denial of Service ist beim Buffer Overflow, Coercive Parsing und SOAP Array Attack dennoch gegeben, der Web-Service ist für einen legitimen Benutzer in der Zeit nicht oder nur stark verzögert nutzbar.

XML Document Size Attack

Dieser Angriff ist durch die bisherigen Versuchsreihen abgedeckt. An dieser Stelle sind keine zusätzlichen Messungen vorgesehen. Es ist zu beobachten, dass erfolgreiche Angriffe meist nicht durch die Größe des Dokuments zu einem Erfolg führen, sondern durch den spezifischen Schwachpunkt, den sie angreifen.

Im Rahmen der Buffer Overflow, Coervice Parsing und Namespace Prefix Attacks sind Angriffe mit Nachrichtengrößen zwischen 20 und 100MB verwendet worden. Aus diesen Angriffen führte lediglich Request 12 der Namespace Prefix Attack zu einem Erfolg. Dieser ist in einem spezifischen Schwachpunkt des XML-Parsers begründet und nicht in der Nachrichtengröße.

Bleibt zu erwähnen, dass die Standardeinstellung des Applikationsserver Anhänge erlaubt, obwohl diese innerhalb des Web-Services keine Verwendung finden.

Versuche mit Anhängen von mehreren hundert Megabyte haben keine nennenswerte Last auf dem Server erzeugt. Diese Aussage trifft auch auf Verteilte-Angriffe zu.

4.3.2 Angriffe durch die XML-Struktur

XML Entity Expansion

Der XML-Parser des *WebSphere Application Server 6.1* verarbeitet keine DTD-Inhalte innerhalb einer SOAP-Nachricht.

Tabelle 4.9 zeigt die ermittelten Messwerte der ausgeführten XML Entity Expansion Angriffe gemäß Quellcode 3.18 unter der Verwendung der Platzhalter nach Tabelle 3.13.

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{x}(X)	{y}(Y)
1	400.473	9	742	ja	10^5	10^5
2	1.300.473	28	3140	ja	10^6	10^5
3	3.100.473	43	11929	ja	10^5	10^6
4	4.000.473	53	14234	ja	10^6	10^6
5	13.000.473	50	31828	ja	10^7	10^6
6	31.000.473	59	6239	ja	10^6	10^7

Tabelle 4.9: Versuchsresultate: XML Entity Expansion

Bei Anfragen die kleinere Werte als Request 1 verwenden, weist der Applikationsserver mit folgendem SOAP-Fault darauf hin, dass keine DTDs verarbeitet werden: Server.generalException, unsupportedSaxEvent (siehe Quelltext G.3)

Bei der Antwort auf Request 2 enthält der SOAP-Fault zusätzlich eine Kopie des Requests.

Die Requests 3, 4 und 5 erzeugen jeweils den HTTP-Fehler „Error 500:“ ohne weitere Angaben.

Der Request 6 erzeugt diesen SOAP-Fault: Server.generalException, unexpected exception (siehe Quelltext G.1)

XML Recursive Entity Expansion

Tabelle 4.10 zeigt die ermittelten Messwerte der ausgeführten XML Recursive Entity Expansion Angriffe gemäß Quellcode 3.19 unter der Verwendung der Platzhalter nach Tabelle 3.14.

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	(X)
1	327.159	9	375	ja	10^4
2	3.567.161	50	12114	ja	10^5
3	38.667.163	54	9479	ja	10^6

Tabelle 4.10: Versuchsresultate: XML Recursive Entity Expansion

Eine Anzahl von 10^6 Rekursionen liefert einen SOAP-Fault zurück: Server.generalException.

Bei einer Anzahl von 10^5 Rekursionen erhält der Client lediglich die Anwort „500:“ oder ebenfalls eine Server generalException einschließlich einer Request-Kopie.

SOAP Array Attack

Tabelle 4.11 zeigt die ermittelten Messwerte der ausgeführten SOAP Array Attacks gemäß Quellcode 3.20 unter der Verwendung der Platzhalter nach Tabelle 3.15.

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{*}(X)	(Y)
1	959.455	3	156	ja	10^4	10^4
2	9.689.456	14	1172	ja	10^5	10^5
3	48.889.456	18	6148	ja	$5 * 10^5$	$5 * 10^5$
4	658	1	26	ja	1	10^4
5	659	3	78	ja	1	10^5
6	660	15	807	ja	1	10^6
7	661	77	10562	ja	1	10^7
8	661	77	29145	ja	1	$2 * 10^7$
9	661	77	76254	ja	1	$2,3 * 10^7$
10	661	77	49350	ja	1	$2,4 * 10^7$
11	662	19	770	ja	1	10^8

Tabelle 4.11: Versuchsresultate: SOAP Array Attack

Die ersten drei Requests verwenden die Anzahl übertragener Array-Elemente als Wert des „arrayType“-Attributes. Bei einer Array-Größe von $2,4 * 10^7$ bleibt die Prozessorauslastung des Web-Services nach Ende der Anfrage bei ~77% bestehen. Der Web-Service reagiert träge. Bei größeren Werten reagiert der Applikationsserver mit einen SOAP-Fault. Allerdings erzeugt er dabei weniger Prozessorlast.

Die Requests 10 und 11 erhalten eine Antwort mit diesem SOAP-Fault: Server.generalException, unexpected exception (siehe Quelltext G.1)

Auffällig ist die signifikant kürzere Reaktionszeit der Fehlermeldung bei einem Wert des Attributes „arrayType“ von 10^8. Der Applikationsserver reagiert unmittelbar auf den Fehler ohne starke Last zu erzeugen.

Sparse Arrays

Tabelle 4.12 zeigt die ermittelten Messwerte der ausgeführten SOAP Array Attack (Sparse Array with Offset) und (Sparse Array with Position) gemäß Quellcode 3.21 und Quellcode 3.22 unter der Verwendung der Platzhalter nach Tabelle 3.16 und Tabelle 3.17 .

Die Resultate der Versuche mit „offset“ und „position“ wurden in dieser Tabelle zusammengefasst. Um einen Vergleich zwischen den SOAP-Array Parametrisierungen zu ermöglichen, wurden die Werte des Request 7 aus der vorherigen Tabelle übertragen.

#	Request (Bytes)	CPU (~%)	Reply (~ms)	verfügbar	{*}(X)	(Y)	(Z)
1	682	77	9703	ja	1	0	10^7
2	705	77	9234	ja	1	0	10^7
3	661	77	10562	ja	1	10^7	-

Tabelle 4.12: Versuchsresultate: SOAP Array Attack (Sparse Array with Offset/Position)

Request 1 verwendet ein „offset“ Attribut. Request 2 verwendet ein „position“ Attribut, um die Position des ersten Elementes innerhalb des Arrays festzulegen.

Es ist zu erkennen, dass die Antwortzeiten unter Verwendung eines „offset“ oder „position“ Attributes nur geringfügig von der des „arrayType“ abweichen. Eine Kombination der Parameter verändert das Bild nicht.

4.4 Angriffe auf Web-Services hinter einer XML-Firewall

In diesem Szenario befindet sich eine IBM DataPower IX50 zwischen dem Web-Service und dem Angreifer. Aus organisatorischen Gründen befindet sich die DataPower IX50 nicht in dem lokalen Netzwerk. Das Gerät ist innerhalb des Intranets der Firma zu erreichen. Aufgrund dessen sind die Messwerte der Reply-Zeit nicht direkt mit den vorherigen Messungen vergleichbar. Sie stellen in diesem Teilabschnitt auch nicht das entscheidende Kriterium dar. Gezeigt werden soll die Schutzwirkung durch die DataPower IX50 Appliance.

Ein virtueller Web-Service wurde in Form eines XML-Firewall-Services auf der DataPower IX50 eingerichtet. Ein Wizard führt unterstützend durch die Schritte der Erstellung eines XML-Firewall-Services. Hierzu wird eine WSDL-Beschreibung benötigt, diese kann z.B. über eine URL referenziert werden oder als Datei hochgeladen werden.

Die WSDL-Beschreibung dient als Vorlage für die XML-Schema-Validierung der SOAP-Requests. An diesem Punkt lassen sich bereits einzelne Service-Methoden filtern. Die DataPower lässt nur registrierte Service-Methoden passieren. WSDL-Scanning Angriffe sind durch diese Maßnahme nicht mehr möglich.

Die XML-Firewall wird mit den Standardeeinstellungen verwendet. Es werden lediglich Einstellungen vorgenommen, die einen Zähler für die Anzahl von Zugriffen eines Konsumenten respektive auf den gesamten Web-Service überwachen. Hierdurch lässt sich einen Beschränkung der Nachrichtenfrequenz in einem Intervall festlegen. Ein Denial of Service durch einen Flooding Angriff wird massiv erschwert. Gleiches gilt für einen verteilten Denial of Service Angriff.

Die Parameter der SOAP-Requests werden mit Werten belegt, die einen Ressourcenverbrauch auf dem Applikationsserver ausgelöst haben. Kleine Werte werden nur getestet, wenn eine Beschränkung durch die XML-Firewall keine höheren Werte zulässt. Da die Prozessorauslastung des Web-Services den Werten aus den vorherigen Versuchen mit denselben Parametern entspricht, wird sie an dieser Stelle nicht mehr angegeben.

Durch die zusätzliche Strecke von Konsument zur XML-Firewall und von der XML-Firewall zum Web-Service und zurück, wird die Reply-Zeit erwartungsgemäß höher ausfallen.

Je nach aktivierter Filter-Regel antwortet die XML-Firewall mit unterschiedlichen Fehlermeldungen oder verwirft die Anfrage unkommentiert. Eine genau Aufschlüsselung der SOAP-Faults ist den Einträgen aus den Syslogs der XML-Firewall zu entnehmen.

Anhand der Größe des SOAP-Replys lässt sich erkennen, ob der Reply von der XML-Firewall oder vom Applikationsserver stammt. Werte Zwischen 0 und 256 Byte sind Firewallbedingt.

Der volle Funktionsumfang der Sicherheitsfunktionen wird in Anhang C vorgestellt.

4.4.1 Angriffe durch den XML-Inhalt

Buffer Overflow

Tabelle 4.13 zeigt die ermittelten Messwerte der ausgeführten Buffer Overflow Angriffe gemäß Quellcode 3.10 unter der Verwendung der Platzhalter nach Tabelle 3.5.

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{x}(X)
1	1.000.449	1.000.477	3656	nein	10^6
2	4.000.449	4.000.477	3656	nein	$4 * 10^6$
3	10.000.449	0	7219	ja	10^7

Tabelle 4.13: Versuchsresultate mit XML-FW: Buffer Overflow

Der Request 3 mit einer Nutzlast von 10^7 Zeichen wird von der XML-Firewall unkommentiert verworfen. Aus den Syslogs der XML-Firewall ist folgendes zu entnehmen:

XMLFIREWALL (MYSERVICESSERVICE): DOCUMENT SIZE LIMIT OF 4194304 BYTES EXCEEDED, ABORTING

Datatype Space Breach

Dokumente, die eine Größe von 4194304 Bytes überschreiten, werden gemäß der Standarteinstellung der XML-Firewall nicht akzeptiert.

Eine Wertebereichsverletzung des Datentyps int wird erfolgreich erkannt und verhindert:

CVC-SIMPLE-TYPE 1: ELEMENT A VALUE ' 2400000000 ' IS NOT A VALID INSTANCE OF TYPE {HTTP://WWW.W3.ORG/2001/XMLSCHEMA}INT

Extra-long Names

Im folgenden werden die Versuchsresultate der drei Varianten dargestellt:

Extra-long Namespaces

Tabelle 4.14 zeigt die ermittelten Messwerte der ausgeführten Extra-long Namespaces Angrife gemäß Quellcode 3.14 unter der Verwendung der Platzhalter nach Tabelle 3.9.

Der Request 2 erhält folgenden SOAP-Fault: Malformed content (from client) (siehe G.4)

Den Syslogs der XML-Firewall ist dazu folgendes zu entnehmen:

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{x}(X)
1	3.450	487	203	nein	10^3
2	30.450	256	78	ja	10^4
3	300.450	0	412	ja	10^5
4	3.000.450	0	1890	ja	10^6

Tabelle 4.14: Versuchsresultate mit XML-FW: Extra-long Namespaces

QNAME PREFIX TABLE OVERFLOW AT LINE 5 OF HTTP://9.155.139.37:9081/MYWS/SERVICES/MYSERVICES

Der Grund liegt in einer Vorgabe der XML-Firewall für die Länge der Namespaces von 3503 Zeichen.

Die Requests 3 und 4 werden durch die XML-Firewall kommentarlos verworfen.

Extra-long Attributenames

Tabelle 4.15 zeigt die ermittelten Messwerte der ausgeführten Extra-long Attributenames Angriffe gemäß Quellcode 3.13 unter der Verwendung der Platzhalter nach Tabelle 3.8.

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{x}(X)
1	1.473	253	141	ja	10^3
2	10.473	256	150	ja	10^4
3	100.473	256	312	ja	10^5
4	1.000.473	256	1875	ja	10^6

Tabelle 4.15: Versuchsresultate mit XML-FW: Extra-long Attributenames

Der Fehlermeldung des 1 Request gilt ebenso für Namespacelänge von $1 - 10^3$. Das XML-Schema der WSDL-Beschreibung sieht kein Attribut an dieser Stelle des Body vor.

Der zugehörige Auszug aus den Syslogs der XML-Firewall:

HTTP://9.155.139.37:9081/MYWS/SERVICES/MYSERVICES:6:
CVC-COMPLEX-TYPE 3: ELEMENT {HTTP://MYPACKAGENAMESPACE}REVERSE WITH ANONYMOUS TYPE HAD UNDEFINED ATTRIBUTE X

Um diese Einschränkung zu umgehen, wird der Angriff so verändert, dass die Elemente im Header des SOAP-Reqests untergebracht werden. Die Verwendung eines WS-Security Headerblocks erlaubt eine Erweiterung durch beliebigen Inhalt.

Die Zeichenlänge für Attributnamen ist durch eine Vorgabe der XML-Firewall auf 3503 Zeichen begrenzt.

Auf eine Darstellung der zusätzlichen Messwerte wird verzichtet, da Requests mit Attributnamen mit mehr als 3503 Zeichen weiterhin von der XML-Firewall gefiltert werden. Zeitunterschiede sind zwischen den gefilterten Angriffen im Header und Body nicht feststellbar.

Lediglich die Fehlermeldung in der XML-Firewall unterscheidet sich:

QNAME LOCAL-PART TABLE OVERFLOW AT LINE 5 OF
HTTP://9.155.139.37:9081/MYWS/SERVICES/MYSERVICES

Extra-long Tagnames

Tabelle 4.16 zeigt die ermittelten Messwerte der ausgeführten Extra-long Tagnames Angriffe gemäß Quellcode 3.12 unter der Verwendung der Platzhalter nach Tabelle 3.7.

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{x}(X)
1	2.457	253	125	ja	10^3
2	20.457	256	218	ja	10^4
3	200.457	0	312	ja	10^5
4	2.000.457	256	1875	ja	10^6

Tabelle 4.16: Versuchsresultate mit XML-FW: Extra-long Tagnames

Die XML-Firewall erwartet, gemäß dem XML-Schema der WSDL, einen fest definierten Tagnamen für das verwendete Element. Abweichende Tagnamen sind nicht erlaubt. Die Firewall blockt die Versuche mit dieser Fehlermeldung:

CVC-PARTICLE 3.1: IN ELEMENT HTTP://MYPACKAGENAMESPACEREVERSE WITH ANONYMOUS TYPE, FOUND <X> (IN DEFAULT NAMESPACE), BUT NEXT ITEM SHOULD BE S

Wird der Angriff in einem WS-Security Headerblock eingebettet, erzeugt er ab einer Tagnamenlänge von 3503 diese Fehlermeldung:

QNAME LOCAL-PART TABLE OVERFLOW AT LINE 6 OF
HTTP://9.155.139.37:9081/MYWS/SERVICES/MYSERVICES

Namepspace Prefix Attack

Attributnames and -content

Tabelle 4.17 zeigt die ermittelten Messwerte der ausgeführten Namespace Prefix Attacks mit einer hohen Anzahl von Attributen mit beliebig langen Attributnamen und -inhalten gemäß Quellcode 3.15 unter der Verwendung der Platzhalter nach Tabelle 3.10.

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{x}(X)	{y}(Y)	{*}(Z)
1	466	253	78	ja	1	1	1
2	2.007.349	0	797	ja	10^3	10^3	10^3
3	466	487	78	nein	1	1	1
4	21.263	0	797	ja	10^3	10^3	10^3

Tabelle 4.17: Versuchsresultate mit XML-FW: Namepspace Prefix Attack (Attributname and -content)

Aus Request 1 lässt sich erkennen, dass die XML-Schema-Validierung der XML-Firewall greift:

ELEMENT {HTTP://MYPACKAGENAMESPACE}REVERSE WITH ANONYMOUS TYPE HAD UNDEFINED ATTRIBUTE X0

Ein Attribut, dass in dem XML-Schema nicht definiert ist, kann nicht verwendet werden.

Daher wird die Namespace Prefix Attack bei Request 3 und 4 in einem WS-Security Headerblock untergebracht. Der Applikationsserver verarbeitet Request 3 ohne besondere Vorkommnisse.

Die Requests 2 und 4 werden durch die XML-Firewall mit folgendem Fehler verworfen:

ATTRIBUTE LIMIT OF 128 PER ELEMENT EXCEEDED, ABORTING AT LINE 5 OF HTTP://9.155.139.37:9081/MYWS/SERVICES/MYSERVICES

Namespace Declarations

Tabelle 4.18 zeigt die ermittelten Messwerte der Namespace Prefix Attack mit einer hohen Anzahl von Namespace Deklarationen gemäß Quellcode 3.15 unter der Verwendung der Platzhalter nach Tabelle 3.10.

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{x}(X)
1	42.239	487	140	nein	10^3
2	438.239	256	63	ja	10^4
3	42.453	487	156	nein	10^3
4	438.453	256	63	ja	10^4

Tabelle 4.18: Versuchsresultate mit XML-FW: Namepspace Prefix Attack (Namespace Declarations)

Auch hier sind die Requests 3 und 4 wieder im Header untergebracht. Die Fehlermeldung entspricht der oberen mit dem Limit von 128 Attributen pro Konten.

Coercive Parsing

Tabelle 4.19 zeigt die ermittelten Messwerte der ausgeführten Coercive Parsing Angriffe gemäß Quellcode 3.17 unter der Verwendung der Platzhalter nach Tabelle 3.12.

Wie bereits geschildert, wurde die Tabelle der Übersicht halber auf die Werte der Knotenbreite und Knotentiefe beschränkt. Die Werte {x}(X), {y}(Y), {z}(Z), {v}(V), {w}(W) haben bei diesen Versuchen alle den Wert 1.

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{*}(T)	{*}(S)
1	1.020.136	487	2144	nein	1	500
2	51.111	487	63	nein	2	8
2	24.773.186	0	15380	ja	2	16
3	10.952.380	0	15234	ja	4	8
4	6.013.614	0	10062	ja	6	6
5	39.885.870	0	36812	ja	6	7

Tabelle 4.19: Versuchsresultate mit XML-FW: Coercive Parsing

Aufgrund der XML-Schema-Validierung der XML-Firewall lassen sich keine zusätzlichen Elemente im Body unterbringen. Um diese Einschränkung zu umgehen, wurden die „Coercive Parsing“-Elemente innerhalb eines Web-Service-Security Headerblocks verwendet.

Request 1 ist als erfolgreiche Anfrage durchgegangen. Alle übrigen Requests wurden durch die XML-Firewall verworfen.

Die entsprechende Fehlermeldung aus den Syslogs der Firewall lautet:

DOCUMENT SIZE LIMIT OF 4194304 BYTES EXCEEDED, ABORTING

Die maximale Knotentiefe ist durch die XML-Firewall auf eine Tiefe von 512 Ebenen beschränkt. Zusätzlich gibt es einen Begrenzung für die maximale Knotengröße in Bytes. Die Standardeinstellung der Firewall erlaubt keine Angriffe auf diesem Wege.

Flooding Attack

Diese Art von Angriff wird nicht durch die Standardeinstellungen der XML-Firewall abgedeckt. Die XML-Firewall bietet die Möglichkeit Vorgaben für die Anzahl der Verbindungen eines Host in einem Intervall zu definieren. Die Gleiche Möglichkeit besteht auch für den Service als solchen (siehe Anhang D.2).

Auf Messungen wird an dieser Stelle verzichtet, da die Anzahl der Requests die bei dem Web-Service ankommen durch einen Filter festgelegt werden. Jeder zusätzliche Request wird verworfen. Ein Angriff ist somit nicht mehr möglich.

Aus den Syslogs der XML-Firewall ist folgendes zu entnehmen: REJECTED BY MONITOR FILTER

Eine Abbildung von Metriken bringt keine zusätzlichen Erkenntnise.

XML Document Size Attack

Diese Form des Angriffs wird durch die Standardeinstellungen der XML-Firewall verhindert. Der Inhalt eines Elements ist auf eine Größe von maximal 4194304 Bytes beschränkt

Durch die Standard-Vorgaben der XML-Firewall werden SOAP-Attachements entfernt. Zusätzlich gilt die Vorgabe einer Größenbeschränkung auf $2 * 10^9$ Bytes.

Die Einschränkungen lassen sich bei Bedarf anpassen.

4.4.2 Angriffe durch die XML-Struktur

XML Entity Expansion

Die XML-Firewall verbietet, gemäß der SOAP Spezifikation, eine Verwendung von „Document Type Definitions“ innerhalb SOAP-Nachrichten.

Tabelle 4.20 zeigt die ermittelten Messwerte der ausgeführten XML Entity Expansion Angriffe gemäß Quellcode 3.18 unter der Verwendung der Platzhalter nach Tabelle 3.13.

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{x}(X)	{y}(Y)
1	4.480	256	47	ja	10^4	10^4
2	40.480	256	78	ja	10^5	10^5
3	400.480	0	203	ja	10^6	10^6
4	49.900.480	0	203	ja	10^7	10^7

Tabelle 4.20: Versuchsresultate mit XML-FW: XML Entity Expansion

In den Syslogs der XML-Firewall steht zu diesen Angriffen:

XMLFIREWALL (MYSERVICESSERVICE): DTD ENCOUNTERED IN SOAP CONTEXT AT LINE 2 OF HTTP://9.155.139.37:9081/MYWS/SERVICES/MYSERVICES

Die Requests 1 und 2 erhalten einen „Malformed content“ SOAP-Fault durch die XML-Firewall. Die Requests 3 und 4 werden kommentarlos verworfen.

XML Recursive Entity Expansion

Tabelle 4.21 zeigt die ermittelten Messwerte der ausgeführten XML Entity Expansion Angriffe gemäß Quellcode 3.19 unter der Verwendung der Platzhalter nach Tabelle 3.14.

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{x}(X)
1	30.164	256	47	ja	10^3
2	327.166	0	219	ja	10^4
3	3.567.168	0	329	ja	10^5

Tabelle 4.21: Versuchsresultate mit XML-FW: XML Recursive Entity Expansion

wi Request 1 erhält einen „Malformed content“ SOAP-Fault durch die XML-Firewall, während die Requests 2 und 3 kommentarlos verworfen werden. Der SOAP-Fault ist identisch zu dem der XML Entity Expansion.

SOAP Array Attack

Tabelle 4.22 zeigt die ermittelten Messwerte der ausgeführten SOAP Array Attacks gemäß Quellcode 3.20 unter der Verwendung der Platzhalter nach Tabelle 3.15.

#	Request (Bytes)	Reply (Bytes)	Reply (~ms)	gefiltert	{*}(X)	{*}(Y)
2	89.461	253	47	ja	10^3	10^3
1	899.462	0	186	ja	10^4	10^4

Tabelle 4.22: Versuchsresultate mit XML-FW: SOAP Array Attack

Bei diesem Angriff greift die XML-Schema-Validierung der Firewall:

CVC-PARTICLE 3.1: IN ELEMENT {HTTP://MYPACKAGENAMESPACE}COUNTSTRINGARRAY WITH ANONYMOUS TYPE, FOUND <SOAPENC:ARRAY> (IN NAMESPACE HTTP://SCHEMAS.XMLSOAP.ORG/SOAP/ENCODING/), BUT NEXT ITEM SHOULD BE S

Das Verwendete XML-Schema der WSDL-Beschreibung erlaubt keine zusätzlichen Attribute für die Array-Elemente innerhalb des Bodies. Ein Angriff ist nur in Verbindung mit einem XML-Schema möglich, dass die Attribute für „arrayType“, „offset“ und „position“ erlaubt. Demzufolge sind auch „Sparse Arrays“ nicht erlaubt.

Wird das SOAP-Array in einem WS-Security Header untergebracht, werden die Element geparsed, aber nicht weiter verarbeitet. Die Werte für die Attribute „arrayType“, „offset“ und „position“ haben keine Auswirkung.

4.5 Zusammenfassung

Nachdem die Anwendungen zum Erstellen und Versenden der SOAP-Nachrichten identifiziert wurden, konnte die Akzeptanz der Nachrichten durch den *WebSphere Application Server* geprüft werden.

Das ausgearbeitete Vorgehen musste bei der Realisierung an die Gegebenheiten angepasst werden. Die serverseitigen Metriken konnten nicht mit der erwarteten Genauigkeit erfasst werden. Anstatt ereignisgesteuerter Messungen die an den Request/Repyl Zyklus gebunden sind, wurden feste Messintervalle von 5 Sekunden verwendet.

Die Messung des Hauptspeicherauslastung konnte aufgrund der automatischen Speicherbereinigung in Verbindung mit den Intervallmessdaten nicht zuverlässig ausgewertet werden. Die Hauptspeicherauslastung unterlag bereits ohne Eingang eines Requests starken Schwankungen. Eine zuverlässige Aussage zu treffen war so nicht möglich.

Während der Versuchsdurchführung hat sich gezeigt, dass der *WebSphere Application Server 6.1* verwundbar gegen Angriffe auf die Verfügbarkeit ist. Meist handelt es sich um Buffer Overflow Angriffe auf den XML-Parser, die zu einer Kompromittierung der Verfügbarkeit des Web-Services geführt haben. In der Regel lag die Prozessorauslastung während einer Kompromittierung zwischen 50%-100%. Es hat sich gezeigt, dass es drei Arten der Kompromittierung gibt.

Ein Request kann zu folgenden Zuständen des Applikationsservers führen:

1. Eingeschränkte Verfügbarkeit während der Verarbeitung.
2. Eingeschränkte Verfügbarkeit während und nach der Verarbeitung.
3. Sofortiger Verlust der Verfügbarkeit während der Verarbeitung.

Im dritten Fall erholt sich der Web-Service nicht mehr von der Anfrage und verbleibt in einem nicht benutzbaren Zustand.

Bei den Versuchsdurchführungen mit dem *WebSphere Application Server 6.1* hinter der *WebSphere DataPower XI50* hat sich gezeigt, dass die XML-Firewall die Angriffe filtert. Es war nicht möglich die Angriffe in der vorherigen Form zu wiederholen. Daher musste einige Angriffe so modifiziert werden, dass der schadhafte Inhalt in einem „freizügigen„ Namensraum untergebracht wird, der durch die XML-Schema Validierung nicht entdeckt werden kann. Dies führte ebenfalls zu keinem erfolgreichen Angriff.

Bereits mit den Standardeinstellungen hat die XML-Firewall alle Angriffe gefiltert, ausgenommen den Flooding Attacks. Für einen Schutz gegen Flooding Attacks mussten zusätzliche Einstellungen vorgenommen werden, die eine Begrenzung der Anfragen eines Host in einem festen Intervall erlauben. Im Anschluß wurden die Flooding Attacks erwartungsgemäß gefiltert.

Im anschließenden Kapitel findet eine Interpretation der Ergebnisse statt. Hierzu werden die erkannten Schwachstellen näher Betrachtung. Sie dienen als Ausgangspunkt für die technische Bewertung des Applikationsservers und der Web-Service-Firewall.

5 Interpretation der Ergebnisse

In diesem Abschnitt findet eine Zusammenfassung der Schwachstellen statt, darauf folgt eine Betrachtung des Applikationsserver und der Web-Service-Firewall in Bezug auf die Schwachstellen. Sie bilden die Grundlage für die technische Bewertung. Die Interpretation bezieht sich ausschließlich auf Angriffe, die in Kapitel 3.7 konzeptionell erarbeitet wurden und in Kapitel 4.3 und 4.4 im Rahmen der Realisierung durchgeführt wurden.

5.1 Zusammenfassen der Schwachstellen (technisch)

Die Messwerte in Kapitel 4.3 haben verdeutlicht, dass Web-Services auf einem WebSphere Application Server 6.1 verwundbar gegen Angriffe auf die Verfügbarkeit sind. Ein Web-Service kann bereits durch eine Nachricht mit einer Größe von 5-10 Megabyte, ohne großen Aufwand, in der Verfügbarkeit kompromittiert werden. Die Messwerte in Kapitel 4.4 belegen, dass die Angriffe durch den Einsatz der Web-Service-Firewall verhindert werden.

Folgende Komponenten sind Ziel der Angriffe auf die Verfügbarkeit:

- XML-Schema der WSDL
- XML-Parser

Die Schwachstellen, werden in den anschließenden Unterkapiteln näher erläutert. Dazu wird das Problem dargestellt, die Ursache erklärt und eine mögliche Gegenmaßnahme aufgezeigt.

5.1.1 XML-Schema der WSDL

Die Verwendung von unzureichend spezifizierten XML-Schemas innerhalb der WSDL, die keine einschränkende Struktur für die XML-Sprache und die darin verwendeten Datentypen vorsehen, kann Angriffe begünstigen. In der Praxis findet eine Validierung der SOAP-Nachrichten gegen die XML-Schemas der WSDL aus Performance-Gründen nicht immer statt. Ein ungenügender Umgang mit den XML-Schemas der WSDL kann dazu führen, dass eine valide SOAP-Nachricht mit schadhaftem Inhalt verarbeitet wird.

Gegenmaßnahme

Um hinreichenden Schutz zu bieten muss ein XML-Schema gehärtet sein. Ein XML-Schema gilt als gehärtet, wenn die Vorgaben bezüglich der Dokumentenstruktur und der eingesetzten Datentypen mitsamt ihrere einschränkenden Fassetten „streng“ abgebildet werden.

Probleme der Gegenmaßnahme

Dennoch kann durch die Verwendung von gehärteten XML-Schemas nicht gewährleistet werden, dass innerhalb der SOAP-Nachricht keine weiteren Namensräume Verwendung finden, die keine gehärteten XML-Schemas verwenden. Eine derartige SOAP-Nachricht ist für den XML-Parser weiterhin gegenüber dem Schema valide.

Beispielweise erlaubt der WS-Security Namensraum beliebige Kind-Elemente innerhalb des `<wsse:Security>`-Headerblocks. Eine Validierung dieser Kind-Elemente findet durch das Schema der WSDL nicht statt. Sie erlauben beliebigen Inhalt zu transportieren. In Kapitel 4.4 wird dieses Vorgehen zum Teil angewendet.

Bewertung

Die Verwendung von gehärteten XML-Schemas in Verbindung mit einem beschränkenden XML-Parser bewirkt eine Einhaltung der Struktur und der erwarteten Datentypen innerhalb der SOAP-Nachrichten.

Prinzipiell könnten die Restriktionen des XML-Schemas den XML-Parser vor Buffer Overflow Attacken schützen. Allerdings muss ein XML-Parser eine SOAP-Nachricht erst vollständig parsen, bevor eine Validierung gegen das XML-Schema stattfinden kann. Nutzt ein Angriff eine Schwachstelle des XML-Parsers, findet dieser vor der „schützenden“ Validierung statt.

Es lässt sich erkennen, dass ein Schutz durch Validierung der SOAP-Nachrichten gegen ein XML-Schema alleine keinen ausreichenden Schutz darstellt. Ein anfälliger XML-Parser kann durch ein XML-Schema nicht geschützt werden. Schadhafte Elemente können innerhalb „freizügiger“ Namensräume mit einem weniger restriktiven Schema untergebracht werden.

5.1.2 XML-Parser

Ein XML-Parser muss einzelne Element einer Nachricht vollständig parsen, bevor er sie validieren kann. Ein Angriff kann aber, wie die Messwerte in Kapitel 4.3 belegen, bereits während des parsens stattfinden. Beispielweise müssen lange Attributnamen erst vollständig eingelesen werden, bevor sie gegen das Schema geprüft werden können.

Wertet ein XML-Parser eine DTD innerhalb einer SOAP-Nachricht aus, werden unnötige Angriffe ermöglicht, bei denen der XML-Parser die SOAP-Nachricht dynamisch um schadhafte Elemente erweitert.

Bestimmt ein XML-Parser die Größe eines SOAP-Arrays lediglich anhand des Wertes des „arrayType“-Attributes, wird im Zuge des „XML to Java-Object Mappings“ und der anschließenden Objekt-Generierung ein entsprechend großes Array für die Verarbeitung in der Anwendungslogik angelegt. Das Ergebnis ist eine Belegung von unnötig viel Hauptspeicher. Bei einem Sparse Array ist der XML-Parser nicht in der Lage eine Reservierung von ungenutztem Speicherplatz zu unterbinden. Die Rückgabewerte der Service-Methode *countStringArray* belegen die Aussage.

Die Angriffe werden ermöglicht durch einen Buffer Overflow oder einer Begrenzung des Heapspeichers.

Gegenmaßnahme

Es muss ein XML-Parser verwendet werden, der Beschränkungen des Datenstroms während des Einlesens von Elementen, Attributen, Inhalten und Namensräume durchsetzt. Die maximale Anzahl der Elemente in der Breite und Tiefe des Dokumentenbaums sollte festlegbar sein. Er muss in der Lage sein die Verarbeitung bei der Verletzung einer Vorgabe unmittelbar abzubrechen und diese bei Bedarf mit einem SOAP-Fault zu quittieren.

Nach SOAP12 [2007] darf eine SOAP-Nachricht keine DTD enthalten, daher sollte der XML-Parser diese nicht erlauben.

Der Wert des „arrayType“-Attributes sollte mit der Anzahl der enthaltenen Array-Elemente des SOAP-Arrays übereinstimmen. Der XML-Parser sollte das Attribut ignorieren, um nur die tatsächlich übermittelten Array-Elemente im Rahmen des Objekt-Mappings zur verarbeiten. Eine Alternative wäre das verbieten des Attributes „arrayType“ durch Einschränkungen eines gehärteten XML-Schemas. Auf gleichem Wege lässt sich die Bedrohung durch Sparse Arrays eindämmen. Die Attribute „position“ und „offset“ sollten durch das Schema verboten werden.

Probleme der Gegenmaßnahme

Ist die Validerung von SOAP-Nachrichten gegen das XML-Schema der WSDL abgeschaltet, findet lediglich eine Beschränkung gemäß der Vorgaben des XML-Parsers statt. Dadurch bleibt die Struktur der SOAP-Nachricht ungeprüft und kann beliebigen Inhalt enthalten, der den Vorgaben des beschränkenden XML-Parsers genügt.

Bewertung

Ein einschränkender XML-Parser verhindert Buffer Overflow Angriffe auf den XML-Parser während des parsens einzelner Elemente. Er erlaubt zuverlässiges parsen und validieren von SOAP-Nachrichten gegen ein XML-Schema. Der Schutz durch den XML-Parser ist auch ohne Schema-Validierung gegeben.

Der Lebenszyklus (und damit die Dauer der Speicherbelegung) eines Arrays innerhalb der Anwendungslogik, kann je nach Lebensdauer des Objektes beliebig lang sein. Die Bedrohung durch unnötige Speicherplatzbelegung durch SOAP-Arrays lässt sich in Verbindung mit einem XML-Schema verhindern.

5.1.3 Zusammenfassung der Schwachstellen

Die erläuterten Schutzmaßnahmen sind prinzipiell in der Lage einen ausreichenden Grundschutz gegen Angriffe auf die Verfügbarkeit zu gewährleisten. Das gehärtete XML-Schema der WSDL gewährleistet die Einhaltung der SOAP-Nachrichtenstruktur, während ein einschränkender XML-Parser garantiert, dass schadhafte Bestandteile der Nachricht nicht bereits vor der Validierung zu einem erfolgreichen Angriff führen.

Ein Grundschutz gegen die Kompromittierung der Verfügbarkeit eines Web-Services ist möglich, wenn die oben genannten Punkte beachtet werden.

Findet innerhalb des Applikationsservers keine Validierung der SOAP-Nachricht gegen das XML-Schema der WSDL statt, bzw. verfügt dieser nicht über einen einschränkenden XML-Parser, ist ein Web-Service grundsätzlich Verwundbar.

Selbst wenn die Validierung der SOAP-Nachricht gegen ein gehärtetes XML-Schema der WSDL stattfindet, ist der wunde Punkt weiterhin der XML-Parser. Ein Service-Anbieter kann auf die Funktionsweise dieser Komponente des Applikationsservers keinen Einfluss nehmen.

Die Anpassung eines bestehenden XML-Parsers kann je nach Verfügbarkeit und Komplexität des Sourcecode aufwendig werden. Von einer Eigenentwicklung eines XML-Parsern wird aufgrund der Komplexität abgeraten.

5.2 Technische Zusammenfassung

An dieser Stelle soll gezeigt werden, in welchem Ausmaß der Applikationsserver und die Web-Service-Firewall von den vorgestellten Schwachstellen betroffen sind.

5.2.1 Applikationsserver

Der Websphere Application Server 6.1 verwendet in der Standardeinstellung keine Schema-Validierung gegen die WSDL-Beschreibung. Es findet lediglich eine Validierung gegen die Schemata der Namensräume aus den SOAP-Nachrichten statt. Diese Aussage lässt sich anhand durchgeführter Angriffe belegen, die nicht valide gegenüber den XML-Schemas der WSDL sind, beispielsweise der Extra-long Tagname Angriff: obwohl das Schema einen bestimmten Tagnamen erwartet, verarbeitet der Applikationsserver auch abweichende Tagnamen.

Aus den SOAP-Faults in Kapitel 4.3.3 lässt sich erkennen, dass der XML-Parser ein SAX[1]-Parser ist, der die SOAP-Nachricht ereignisbasiert verarbeitet. Wird bei der Verarbeitung der Nachricht ein Element erkannt, wird ein Ereignis ausgelöst.

Der verwendete SAX-Parser kennt keine Beschränkung für die Länge des Datenstroms die ein Element haben kann - es können beliebig lange Datenströme verwendet werden. Die Angriffe durch den XML-Inhalt und die XML-Struktur machen sich die genannten Eigenschaften zu nutzen.

Die Resultate der Flooding Angriffe in Kapitel 4.3.1 zeigen, dass der Applikationsserver eine beliebige Anzahl von Anfragen eines Clients entgegen nimmt. Geeignete Maßnahmen gegen Flooding Angriffe sind nicht gegeben.

5.2.2 Web-Service-Firewall

Die WebSphere DataPower Integration Appliance XI50 verwendet in der Standardeinstellung eine Schema-Validierung gegen die WSDL-Beschreibung. Es findet eine Validierung der Strukturen und Datentypen einschließlich ihrer Fassetten statt.

[1] Simple API for XML

Der enthaltene XML-Parser arbeitet einschränkend. Er erlaubt Vorgaben für Nachrichtengröße, Attribut-Anzahl, Datenstromlänge, Knotentiefe, Knotengröße, Anhanggröße und Schutz vor rekursiven Elementen einer SOAP-Nachricht festzulegen. Die Standardeinstellung der XML-Firewall verhindert die ausgearbeiteten Angriffe durch den XML-Inhalt und die XML-Struktur. Ein Schutz gegen Flooding Angriffe benötigt zusätzliche Einstellungen der Requestdauer und der Beschränkung der Nachrichtenfrequenz innerhalb eines Intervalls für Hosts und die gesamte XML-Firewall.

Im Anhang befinden sich entsprechende Abbildungen der Einstellungen und der zugehörigen Hilfetexte aus der Management-Oberfläche der DataPower XI50.

5.3 Technische Bewertung

Die Resultate der Realisierung aus Kapitel 4.3 verdeutlichen, dass der WebSphere Aplication Server 6.1, in den untersuchten Punkten, anfällig für Angriffe gegen die Verfügbarkeit ist. Insbesondere stellen Buffer Overflow Angriffe auf den XML-Parser ein akutes Problem dar. Die Verfügbarkeit kann durch Nachrichten mit einer Größe von wenigen Megabyte gestört werden.

Es ist davon auszugehen, dass sich die Schwachstellen der Implementierungen nur unter hohem Aufwand beseitigen lassen. Erst durch Austausch der fehlerhaften Komponenten bzw. der Erweiterung der Implementierung kann das Problem gelöst werden.

Die WebSphere DataPower Integration Appliance XI50 verwendet, wie in Kapitel 4.4 gezeigt, über Implementierungen die nicht anfällig für die dargestellten Angriffe auf die Verfügbarkeit sind. Durch die Verwendung eines beschränkenden XML-Parsers und der konsequenten Validierung gegen das XML-Schema der WSDL werden die Angriffe erfolgreich abgewehrt. Durch Zusatzeinstellungen lassen sich ebenso Flooding Angriffe zuverlässig verhindern.

Die Erstellung eines XML-Firewall-Services wird durch einen Wizard begleitet. Der Firewall-Service verfügt unmittelbar über einen Grundschutz, welcher über den Schutz der Verfügbarkeit hinausgeht.

Die DataPower Appliance erzielt einen umfassenden Schutz für die Verfügbarkeit der Web-Services des WebSphere Application Servers.

6 Bewertung der Ergebnisse

In diesem Abschnitt findet eine Bewertung des geschäftlichen Mehrwertes statt. Die technische Bewertung aus dem vorherigen Kapitel stellt die Grundlage für die Ableitung des geschäftlichen Mehrwertes dar.

6.1 Betrachtung des geschäftlichen Mehrwertes

Eine Betrachtung des geschäftlichen Mehrwertes kann anhand quantitativer oder qualitativer Kriterien stattfinden. Eine quantitative Bewertung kann nach reinen Kostenaspekten oder Kosten-Nutzenaspekten geschehen.

Bei einer qualitativen Betrachtung findet eine Bewertung der funktionalen Aspekte der Web-Services-Firewall in Bezug auf die Problemdomäne statt.

Steht der Bedarf eines bestimmtes Schutzziels fest[1] ist eine Bewertung nach Kostenaspekten vorzuziehen. Diese basiert auf dem Vergleich der Total Cost of Ownership (TCO) aller möglichen Lösungen. Eine TCO umfasst die Summe der entstehenden Kosten über die Lebensdauer der Lösung hinweg, typischerweise setzen sich diese aus aus Kosten für Anschaffung, Installation und Betriebskosten zusammen. Das Ziel dieser Methode ist die Lösung mit der geringsten TCO zu ermitteln, die dem angeforderten Schutzziel entspricht.

Eine Bewertung nach Kosten-Nutzenaspekten basiert auf dem Return on Investment. Sie kommt zum Einsatz, wenn der Bedarf einer Schutzmaßnahme ermittelt werden soll. Sie bildet die Summe der Ersparnisse oder Mehreinnahmen, die nach Abzug der Kosten für die Schutzmaßnahme erzielt werden. Bei dieser Betrachtung ist eine rasche Amortisierung von Vorteil für eine Lösung. Der Nutzen wird, im Gegensatz zu den Kosten, aufgrund von Erwartungen abgeschätzt. Im Bereich der IT-Sicherheit wird der Return on Security Investment (RoSI) verwendet.

[1] Beispielweise durch gesetzliche Vorgaben.

Aufgrund der fehlenden Datengrundlage ist im Rahmen dieser Ausarbeitung eine quantitaive Mehrwertbetrachtung nicht möglich. Im Folgenden findet in dieser Arbeit eine qualitative Bewertung des geschäftlichen Mehrwertes statt. Für weiterführende Literatur zu quantitativer Mehrwertbetrachtung siehe Pohlmann [2006].

6.1.1 Schutzmerkmale der Web-Service-Firewall

Die WebSphere DataPower Integration Appliance XI50 bietet einen umfassenden Schutz für Web-Services. Sie deckt die Schutzziele Authentifizierung, Autorisierung, Integrität, Verbindlichkeit, Vertraulichkeit und Verfügbarkeit ab.

An dieser Stelle folgt eine Zusammenfassung der Schutzmerkmale der Web-Service-Firewall. Schutzmerkmale, die über die untersuchten Aspekt der Verfügbarkeit hinausgehen, werden hier kurz vorgestellt, finden aber im Rahmen der Bewertung keine weitere Betrachtung.

Der Schutz vor schadhaften XML-Inhalten erhöht die Verfügbarkeit des Web-Services und bietet Schutz gegen Kompromittierung der Anwendungslogik und den dahinter liegenden Back-End-Systemen. Die Unterstützung von WS-Security erlaubt standardisierte Schutzmaßnahmen unter Verwendung digitaler Zertifikate in den Bereichen Berechtigung, Integrität, Vertraulichkeit und Verbindlichkeit. Die Berechtigungsprüfung bietet einen Schutz der Authentifizierung und Autorisierung. Kommerzielle als auch auf offenen Standards basierende Technologien wie WS-Security, SAML und LDAP können in die Berechtigungsprüfung integriert werden. Ein Umschreiben der SOAP-Faults garantiert, dass diese keine sensiblen Informationen über Interna der Web-Services preisgegeben. Die Verwendung von nicht freigegebenen Service-Methoden ist nicht möglich.

6.1.2 Initialinvestition für den Betrieb einer Web-Service-Firewall

Im folgenden wird die Investition für die Beschaffung und den Betrieb eines WebSphere DataPower Systems abgeschätzt. Die Preise der DataPower Systeme sind Listenpreise mit Stand vom 04.06.2008.

Der Gesamtaufwand lässt sich unterteilen in:

- Anschaffungskosten des WebSphere DataPower Systems.
- Inbetriebnahme der DataPower und Mitarbeiterschulung.
- Jährliche Weiterbildung der Mitarbeiter.

Aufwand	Kosten
DataPower XS40	62.500 Euro
Inbetriebnahme und Schulung	40.000 Euro
Summe	102.500 Euro

Tabelle 6.1: Initalaufwand DataPower XS40

Aufwand	Kosten
DataPower XI50	72.500 Euro
Inbetriebnahme und Schulung	40.000 Euro
Summe	112.500 Euro

Tabelle 6.2: Initalaufwand DataPower XI50

Der Aufwand für die Inbetriebnahme und die Mitarbeiterschulung von 2 Mitarbeitern wird mit 40 Personentagen abgeschätzt. Der Aufwand der jährlichen Weiterbildung wird pro Mitarbeit mit 5 Personentagen abgeschätzt. Der Aufwand eines Personentages wird mit 1.000 Euro abgeschätzt.

Tabelle 6.1 zeigt den Initalaufwand für den Betrieb einer DataPower XS40. Tabelle 6.2 zeigt den Initalaufwand für den Betrieb einer DataPower XI50. Zu diesen Kosten addieren sich jährliche Weiterbildungskosten der administrierenden Mitarbeiter mit je 5000 Euro.

Die Kosten beziehen sich auf den gesamten Funktionsumfang der Sicherheitsmerkmale.

6.1.3 Qualitativer Mehrwert der Web-Service-Firewall

Im vorherigen Abschnitt wurde ein kurzer Überblick über die Schutzmerkmale der DataPower XI50 gegeben. Die Mehrwertbetrachtung der Web-Service-Firewall konzentriert sich auf die untersuchten Angriffe gegen die Verfügbarkeit von Web-Services. Weiterführende Schutzmaßnahmen sind bei der Mehrwertbetrachtung nicht mit eingeschlossen.

Durch die Resultate aus Kapitel 4.3 und 4.4 wird belegt, dass Web-Services auf einem WebSphere Application Server 6.1 bei direkten Anfragen verwundbar gegen Angriffe auf die Verfügbarkeit sind und dass die XML-Firewall der DataPower XI50 die Web-Services vor diesen Angriffen schützt.

Die Datapower XI50 bietet folgende Vor- und Nachteile:

Vorteile:

- Die untersuchte XML-Firewall bietet einen Zusätzlichen Schutz bei Angriffen gegen die Verfügbarkeit.
- Die Schutzmaßnahmen können an individuell angepasst werden (siehe Anhang D).
- Die Applikationsserver können durch eine vorgezogene Überprüfung der SOAP-Nachricht gegen Wohlgeformtheit und Validität entlastet und geschützt werden.
- Höhere Effektivität des Sicherheitsmanagement durch Konsolidierung der Schutzmaßnahmen. Gebündeltes Management an einem Punkt verringert die Kosten, indem es die Wartbarkeit verbessert und die Fehleranfälligkeit verringert.
- Als Infrastrukturkomponente gewährleistet die Web-Service-Firewall Schutz für alle angeschlossenen Web-Services - Unabhängig vom eingesetzten Applikationsserver oder Legacy-System.
- Benötigt die Implementierung der Schutzmaßnahmen eine Erweiterung oder Fehlerbereinigung, ist dies aufgrund der reduzierten Komplexität der DataPower XI50 im Vergleich zu einem Applikationsserver oder einem Legacy-System einfacher zu implementieren, weniger Fehleranfällig, erzeugt weniger Seiteneffekte und verursacht dadurch weniger Kosten.

Nachteile:

- Durch ein zentrales Sicherheitsmanagement können Fehlkonfigurationen unter Umständen Auswirkungen auf alle zu schützenden Web-Services haben.
- Mit wachsenden Anforderungen an die Performance kann der Einsatz einer XML-Firewall zu einem Ressourcen-Engpass führen.
- Es entstehen Anschaffungskosten und zusätzliche Schulungskosten des Wartungspersonals für ein fachgerechtes Sicherheitsmanagement im Umgang mit dem System.

Darüber hinaus verfügt das Gerät über zusätzliche Mehrwerte, die nicht Teil der Bewertung sind. Diese können dem Anhang E entnommen werden.

Daraus lässt sich folgender qualitative Mehrwert Ableiten:

Die DataPower XI50 bietet einen zuverlässigen Schutz der Verfügbarkeit und ist an individuelle Bedürfnisse anpassbar. Eine Entlastung der Server ist durch vorgezogene Prüfung der Wohlgeformtheit und Validität möglich. Das Sicherheitsmanagement wird durch Konsolidierung effektiver. Das Resultate sind Kostenreduktion durch verbesserte Wartbarkeit und verminderte Fehleranfälligkeit. Sie gewährleistet als Infrastrukturkomponente Schutz für alle angeschlossenen Systeme, unabhängig von der Implementation des Applikationsservers oder Legacy-Systems. Erweiterungen und Fehlerbereinigungen sind einfacher implementiert, weniger fehleranfällig, erzeugen weniger Seiteneffekte und verursachen dadurch weniger Kosten. Demgegenüber stehen Initialkosten für Anschaffung und notwendige Schulungen des Wartungspersonals. Die Schulungen sollen den fachgerechten Umgang mit dem

System vermitteln und dadurch mögliche Fehlkonfigurationen vermeiden. Die XML-Firewall sollte den Anforderungen an die benötigte Performance entsprechen, um nicht zu einem Ressourcen-Engpass zu führen.

6.2 Übertragbarkeit

Anhand der untersuchten Angriffe konnten Schwachpunkte im Einsatz von Web-Services identifiziert werden, die durch gegebene Schutzmaßnahmen nicht abgedeckt werden.

Diese Schwachpunkte lassen sich identifizieren:

- Das Kommunikationsprotokoll muss einen Schutz für Berechtigung, Integrität, Verbindlichkeit, Vertraulichkeit, Zuverlässigkeit bieten.
- Die Deserieliserungs[2]- bzw. Unmarshalling[3]-Komponente muss vor schadhaften Inhalten schützen und einen geeigneten Buffer Overflow Schutz bieten.

6.2.1 Bedeutung für Web-Services

SOAP verfügt selbst über keine Sicherheitsmerkmale. Erst in Verbindung mit den SOAP-Erweiterungen WS-Security und WS-Reliable-Messaging können SOAP-Nachrichten auf der Nachrichten-Ebene geschützt werden. WS-Security deckt die Schutzziele Berechtigung, Integrität, Verbindlichkeit, Vertraulichkeit ab, während WS-Reliable-Messaging einen Schutz der Zuverlässigkeit der Kommunikation adressiert.

Ein Schutz der Verfügbarkeit kann durch die Verwendung von SOAP-Erweiterungen alleine nicht abgedeckt werden.

Eine Deseriealiserung der SOAP-Nachrichten findet unter Verwendung eines XML-Parsers und einem Mapping der XML-Elemente in die implementationsspezifische Representation statt. Beispielsweise fand im Rahmen der Untersuchung ein „XML to Java Object Mapping“ statt, aus dem die entsprechenden Java-Objekte erzeugt wurden. Ein Schutz der Verfügbarkeit ist an dieser Stelle nur gewährleistet, wenn alle Teile die an der Deserialisierung beteiligt sind einen Schutz vor Buffer Overflow Angriffen und schadhaften Inhalten bieten.

[2] Umwandlung eines Bytestroms in eine Objekt

[3] Objekte aus dem Adressraum eines anderen Computers Lokal Verfügbar machen

6.2.2 Verallgemeinerung

Aus den Schwachpunkten lässt sich ableiten, dass die geforderten Schutzmerkmale für den Betrieb einer dienstbasierten Technologie, wie zum Beispiel CORBA, DCOM und RMI, notwendig sind.

Ein Kommunikationsprotokoll mit den benötigten Schutzmerkmalen ist hier ebenso entscheidend wie eine hinreichend geschützte Unmarshalling-Komponente. Kann die Unmarshalling-Komponente schadhafte Inhalte nicht filtern, oder nimmt keine Begrenzung des zu verarbeitenden Datenstroms vor, ist auch dort eine Anfälligkeit für Angriffe auf die Verfügbarkeit gegeben.

Die Schwachstellen finden sich auch bei der Verwendung einer anderen dienstbasierten Technologie an den bekannten Stellen. Der Einsatz einer geeigneten *Application Layer Firewall*[4] kann inhaltsbasierte Filterungen vornehmen und die dienstbasierte Technologie auf diese Weise vor direkten Angriffen schützen.

Verallgemeinert lässt sich sagen, dass auch dienstbasierte Technologien mit umfangreichen Sicherheitsmerkmalen Schwachstellen innerhalb des Schutzes von Kommunikationsprotokoll oder Unmarshalling-Komponente aufweisen können, die zu einer Kompromittierung der Verfügbarkeit führen.

[4] Ein Schutz auf der Anwendungsschicht, der 7. Schicht des ISO/OSI-Schichtenmodels.

7 Fazit und Ausblick

Zum Abschluß dieser Arbeit wird das Erreichen der festgelegten Ziele Anhand der Ergebnisse dokumentiert. Danach erfolgt ein Ausblick auf sinnvolle Anschlußarbeiten.

7.1 Zusammenfassung

Am Ende der Arbeit wurden folgende Ziele erreicht:

Die erfolgreich identifizierten Angriffstechniken der Angriffspunkte XML, SOAP und Applikationsebene wurden gemäß ihrer Angriffsziele in Angriffsklassen unterteilt:

- Angriffe auf die Berechtigung,
- Angriffe auf die Daten-Integrität und Vertraulichkeit,
- Angriffe auf die Systemumgebung und
- Angriffe auf die Verfügbarkeit.

Für die entsprechenden Angriffstechniken wurden konzeptionelle Angriffen in Kapitel 3.7 erarbeitet. Im Zuge der Realisierung wurde das Kommandozeilen-Werkzeug „SOAP-Generator“ zur Durchführung der Angriffe entwickelt. In Kapitel 4.3 wurden die manipulierten Nachrichten bei direkten Angriffen gegen die Web-Services des WebSphere Application Servers eingesetzt. Es konnten erfolgreiche Angriffe gegen die Verfügbarkeit des Web-Services durchgeführt werden.

Bei einer Wiederholung der Angriffe in Kapitel 4.4 konnte ermittelt werden, dass der Web-Service hinter einer Web-Service-Firewall geschützt war. Die zuvor ermittelten Lücken des WebSphere Application Server wurden abgedeckt, die Angriffe waren nicht mehr möglich.

Die Schwachstellen die zu den Angriffen führen wurden in Kapitel 5 erkannt:

- Validierung gegen unzureichend gehärtete XML-Schemas
- XML-Parser die keine Beschränkung des Datenstroms vornehmen

Ein unzureichend gehärtetes XML-Schema kann unnötige „Freiräume“ innerhalb einer XML-Nachricht erlauben. Dies erlaubt XML-Nachrichten mit schadhaften Bestandteilen, die dennoch valide gegenüber dem Schema sind. Verwendet der Web-Service keine Schema Validierung darf die XML-Nachricht beliebigen Inhalt besitzen.

Allerdings muss ein XML-Parser eine XML-Nachricht erst komplett parsen, bevor er sie gegen ein XML-Schema validieren kann. Ein Angriff auf den XML-Parser findet bereits beim parsen statt. Das Problem lässt sich beheben durch die Verwendung eines Parsers der Beschränkungen des Datenstroms vornehmen kann.

Die technische Bewertung des Applikationsservers hat gezeigt, dass dessen Implementierung anfällig für die ermittelten Schwachstellen sind. Die Implementierung der Web-Service-Firewall weist diese Schwachstellen nicht auf.

Basierend auf der technischen Bewertung aus Kapitel 5.3 wurde der qualitative Mehrwert der Web-Service-Firewall in Kapitel 6.1.2 im untersuchten Bereich abgeleitet.

Die DataPower XI50 bietet einen zusätzlichen Schutz der Verfügbarkeit. Durch vorgezogene Schutzmaßnahmen erlaubt sie die Server um diese Aufgaben zu entlasten. Ein zentralisiertes Sicherheitsmanagement führt zu einer Kostenreduktion durch eine verbesserte Wartbarkeit und Verringerung der Fehleranfälligkeit. Der Schutz ist unabhängig vom verwendeten Applikationsserver oder Legacy-System verfügbar. Eine Erweiterung oder Fehlerbereinigung der DataPower Schutzfunktionen ist einfacher implementiert, weniger fehleranfällig, erzeugt weniger Seiteneffekte und verursacht dadurch weniger Kosten. Demgegenüber stehen Initialkosten für die Anschaffung und notwendige Schulungen des Wartungspersonals. Die Schulungen sollen den fachgerechten Umgang mit dem System vermitteln und dadurch mögliche Fehlkonfigurationen vermeiden. Die XML-Firewall sollte den Anforderungen an die benötigte Performance entsprechen, um nicht zu einem Ressourcen-Engpass zu führen.

7.2 Ausblick

Im Rahmen der Arbeit konnte nur ein kleiner Teilausschnitt der Web-Service Spezifikationen Betrachtung finden. Durch jede zusätzliche Web-Service Spezifikation bleibt nicht auszuschließen, dass neben neuer Funktionalität auch potentielle Schwachstellen hinzukommen können. Es bietet sich an entsprechende Spezifikationen systematisch zu sondieren und dadurch mögliche Schwachstellen zu identifizieren, die auf einer Anforderung nach Zukunftsfähigkeit oder grundlegenden Entwurfsschwächen beruhen.

Aufgrund der Komplexität und Vielzahl der Spezifikationen wurden in dieser Arbeit die Angriffe, die durch den Einsatz standardisierter Schutzmaßnahmen abgedeckt sind nicht weiter

betrachtet. Auch bei den Standardisierten Schutzmaßnahmen besteht ein Bedarf an automatisierten Testmethoden zum Auffinden möglicher Schwachstellen. An dieser Stelle wäre ein Vergleich möglich, der Unterschiede in den einzelnen Implementierungen aufzeigt, die zu Schwachstellen führen.

Das entwickelte Kommandozeilen-Werkzeug SOAP-Generator könnte zu einem automatisierten Stresstest von Web-Service Implementationen ausgebaut werden. Durch die Einbeziehung der Informationen einer WSLD-Beschreibung könnte das Werkzeug generisch eingesetz werden. Für einen vollständig automatisierten Testablauf benötigt das Werkzeug eine Zusätzliche Komponenten auf dem System des Applikationsserver, dass ihm Daten über Ressourcenauslastung mitteilt und bei bedarf den Applikationsserver neu startet. Das Ausgabeformat bedarf ebenfalls einer Überarbeitung, um besser auswertbar zu sein. Eine automatisierte Auswertung der Resultate ist erstrebenswert.

Durch eventbasierte Messungen ist mit genaueren Messwerten zu rechnen, insbesondere in Hinblick auf die Messung der Speicherauslastung. Die Ausarbeitung einer eventbasierten Messmethode hätte den Zeitlichen Rahmen der Arbeit überstiegen.

Literaturverzeichnis

[Actional 2004] ACTIONAL: *The Web Services Security Threat White Paper*. Whitepaper. Oktober 2004. – URL http://www.actional.com/products/docs/white_paper_web_service_security_threat.pdf. – Zugriffsdatum: 2008.04.25

[BSI 2008a] BSI, Bundesamt für Sicherheit in der Informationstechnik : *Sicherheit der Java-Plattform. Bedrohungen und Risiken.* Februar 2008. – URL http://www.bsi.de/fachthem/java/sicherheit_bedrohungen.htm. – Zugriffsdatum: 2008.03.15

[BSI 2008b] BSI, Bundesamt für Sicherheit in der Informationstechnik: *SOA-Security-Kompendium*. Januar 2008. – URL http://www.bsi.de/literat/studien/soa/SOA-Security-Kompendium.pdf

[Dwibedi 2005] DWIBEDI, Runa: *XPath injection in XML databases*. Juli 2005. – URL http://palisade.plynt.com/issues/2005Jul/xpath-injection/. – Zugriffsdatum: 2008.03.24

[HTTP 1999] HTTP, The Internet Engineering Task F.: *Hypertext Transfer Protocol – HTTP/1.1*. 1999. – URL http://www.ietf.org/rfc/rfc2616.txt. – Zugriffsdatum: 2008.02.12

[Klein 2005] KLEIN, Amit: *Blind XPath Injection - A whitepaper from Watchfire*. 2005

[Leser und Naumann 2007] LESER, Ulf ; NAUMANN, Felix: *Informationsintegration. Architekturen und Methoden zur Integration Verteilter heterogener Datenquellen*. dpunkt.verlag, 2007. – ISBN 3-89864-400-6

[Liebhart 2007] LIEBHART, Daniel: *SOA goes real*. Hanser, 2007. – ISBN 978-3-446-41088-6

[Lindstrom 2004] LINDSTROM, Pete: *Attacking and Defending Web Services*. Januar 2004. – URL http://www.forumsystems.com/papers/Attacking_and_Defending_WS.pdf

[Melzer u. a. 2007] MELZER, Ingo u. a.: *Service-orientierte Architekturen mit Web Services. Konzepte - Standards - Praxis*. 2. Elsevir, 2007. – ISBN 978-3-8274-1885-2

[Moradian und Hakansson 2006] MORADIAN, Esmiralda ; HAKANSSON, Anne: Possible attacks on XML Web Services. In: *IJCSNS International Journal of Computer Science*

and Network Security 6 (2006), January, Nr. 1B, S. 154–170. – URL http://paper.ijcsns.org/07_book/200601/200601B48.pdf. – Zugriffsdatum: 2008.03.24

[OASIS 2006a] OASIS, Organization for the Advancement of Structured Information Standards: *Web Services Security SOAP Message Security 1.1*. Februar 2006. – URL http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf. – Zugriffsdatum: 2008.03.17

[OASIS 2006b] OASIS, Organization for the Advancement of Structured Information Standards: *Web Services Security UsernameToken Profile 1.1*. Februar 2006. – URL http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf. – Zugriffsdatum: 2008.03.17

[O'Neill 2005] O'NEILL, Mark: *Hardening Network Security*. Kap. Chapter 5 - Hardening Web Services, S. 113–140, McGraw-Hill Osborne Media, Januar 2005. – URL http://www.vordel.com/downloads/hardening_network_security_chapter.pdf. – Zugriffsdatum: 2008.03.23

[Pohlmann 2006] POHLMANN, Prof. Dr. N.: *Wirtschaftlichkeitsbetrachtung von IT-Sicherheitsmechanismen*. 2006. – URL http://www.internet-sicherheit.de/fileadmin/docs/publikationen/Wirtschaftlichkeit_ITsec_06_03_04.pdf

[Smith und Franke 2003] SMITH, Randy F. ; FRANKE, Susanne: Kontrollposten im Vorfeld. In: *Windows IT Pro* 11 (2003). – URL http://www.netigator.de/netigator/live/fachartikelarchiv/ha_artikel/powerslave,id,10062176,obj,WM,np,archiv,ng,,thes,.html. – Zugriffsdatum: 2008.04.12

[SOAP11 2000] SOAP11, World Wide Web C.: *Simple Object Access Protocol (SOAP) 1.1*. Mai 2000. – URL http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. – Zugriffsdatum: 2008.02.11

[SOAP12 2007] SOAP12, World Wide Web C.: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. April 2007. – URL http://www.w3.org/TR/2007/REC-soap12-part1-20070427/. – Zugriffsdatum: 2008.02.14

[SOAPENV 2007] SOAPENV, World Wide Web C.: *SOAP Version 1.2 Part 0: Primer (Second Edition)*. April 2007. – URL http://www.w3.org/TR/2007/REC-soap12-part0-20070427/. – Zugriffsdatum: 2008.02.14

[Vonhoegen 2007] VONHOEGEN, Helmut: *Einstieg in XML - Grundlagen, Praxis, Referenzen*. 4. Galileo Computing, 2007. – ISBN 978-3-8362-1074-4

[WASC 2005] WASC, Web Application Security C.: *Threat Classification - SQL Injection*.

2005. – URL http://www.webappsec.org/projects/threat/classes/sql_injection.shtml. – Zugriffsdatum: 2008.04.24

[Werawarana u. a. 2005] WERAWARANA, Sanjiva ; CURBERA, Francisco ; LEYMANN, Frank ; STOREY, Tony ; FERGUSON, Donald F.: *Web Services Platform Architecture. SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging , and More*. Person Education, 2005. – ISBN 0-13-148874-0

[WSA 2004] WSA, World Wide Web C.: *Web Services Architecture*. Februar 2004. – URL http://www.w3.org/TR/2001/NOTE-wsdl-20010315. – Zugriffsdatum: 2008.04.01

[WSDL11 2001] WSDL11, World Wide Web C.: *Web Services Description Language (WSDL) 1.1*. März 2001. – URL http://www.w3.org/TR/2001/NOTE-wsdl-20010315. – Zugriffsdatum: 2008.02.20

[WSDL20 2007] WSDL20, World Wide Web C.: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. Juni 2007. – URL http://www.w3.org/TR/2007/REC-wsdl20-20070626/. – Zugriffsdatum: 2008.02.17

[WSI-B12 2007] WSI-B12, Web Services Interoperability O.: *Basic Profile Version 1.2*. Oktober 2007. – URL http://www.ws-i.org/Profiles/BasicProfile-1_2(WGAD).html. – Zugriffsdatum: 2008.03.16

[XML 2006] XML, World Wide Web C.: *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. August 2006. – URL http://www.w3.org/TR/2006/REC-xml-20060816. – Zugriffsdatum: 2008.02.10

[XML-DSIG 2008] XML-DSIG, World Wide Web C.: *XML Signature Syntax and Processing (Second Edition)*. März 2008. – URL http://www.w3.org/TR/2008/PER-xmldsig-core-20080326/. – Zugriffsdatum: 2008.03.13

[XML-ENC 2002] XML-ENC, World Wide Web C.: *XML Encryption Syntax and Processing*. Dezember 2002. – URL http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/. – Zugriffsdatum: 2008.03.13

[XML-NS 2006] XML-NS, World Wide Web C.: *Namespaces in XML 1.0 (Second Edition)*. August 2006. – URL http://www.w3.org/TR/2006/REC-xml-names-20060816. – Zugriffsdatum: 2008.02.10

[Zedlitz 2006] ZEDLITZ, Jesper: *Webservice-Firewall. Sicherheit durch Validierte SOAP-Nachrichten*. VDM Verlag Dr. Müller, 2006. – ISBN 3-8364-0001-4

A Inhalt der beiliegenden CD-ROM

Auf der beiliegenden CD-ROM befindet sich folgender Inhalt:

- /ws/ - Quelltext des exemplarischen Web-Services. Enthält eine kurze Anleitung zur Generierung des Web-Services unter Verwendung eines Wizards.
- /wsdl/ - Die WSDL-Beschreibung des exemplarischen Web-Services.
- /soapgen/ - Ausführbare Version und Quelltext des SOAP-Generators. Inklusive einer Kurzanleitung.
- /reqex/ - Die exemplarischen SOAP-Requests.
- /misc/ - Sammlung keiner Bash-Hilfsskripte.
- /Bachelorarbeit.pdf - Eine digitale Kopie der Bachelorarbeit im PDF-Format.
- /content.txt - Eine Auflistung des CD-ROM Inhalts.

B Zusätzliche Sicherheitsstandards

Die im folgenden vorgestellten Sicherheitsstandards erweitern den Schutz für Web-Services. Im Mittelpunkt stehen Spezifikationen für einen standardisierten Umgang mit Credentials (auch über Unternehmensgrenzen hinaus) und Formulierung von Sicherheitsanforderungen und Richtlinien an die Kommunikation.

B.1 XML-Key Management specification (XKMS)

XMKS standardisiert den Zugriff auf eine PKI für den Austausch und die Überprüfung von digitalen Zertifikaten. XMKS unterstützt XML-Signature und XML-Encryption bei der Gewährleistung von Integrität, Verbindlichkeit und Vertraulichkeit. Dabei kapselt XKMS die hohe Komplexität einer PKI, indem es eine Service-Fassade bietet. XMMS besteht aus zwei Komponenten:

XML Key Information Service Specifiation (X-KISS) ermöglicht die Authentifizierung der Zertifikate mittels Abfrage bei einem entsprechenden PKI-Dienstleiter. Dazu zählen Bereitstellung, Lokalisierung und Validierung eines Zertifikats.

XML Key Regisitraion Service Specification (X-KRSS) sorgt für das Management der Zertifikate: Ausstellung, Wiederrufung und Wiederherstellung von Zertifikatsschlüsseln.

B.2 Security Assertion Markup Language (SAML)

SAML ist ein OASIS Standard, der den Austausch von authentisierungs- und autorisierungsbezogene Informationen unter Web-Services festlegt. Er ermöglicht die Einbettung erweiterter SAML Sicherheitsinformationen in WS-Security Elemente eines SOAP-Dokuments. Diese Informationen beziehen sich auf ein beliebiges Subjekt (z.B einen Service-Konsumenten) der einen Web-Service aufruft und werden in Form von Assertions eingebunden. Assertions sind festgelegte, zusätzliche Einträge in SOAP-Nachrichten.

Bisher sind drei Arten (Statements) standardisiert:

Attribute Statement erlaubt vertrauenswürdige Aussagen über beliebige Attribute eines Subjects.

Authentication Decision Statement enthält eingebette Information aufgrund dieser die Entscheidung getroffen wird, ob ein Subject autorisiert ist, auf eine Ressource zuzugeifen.

Authentication Statement dient der Einbindung von erweiterten Authentifizierungsinformationen in eine SOAP-Nachricht gemäß dem WS-Security Standard. Ein Beispiel für eine solche Information ist ein Benutzer-Zertifikat.

So lassen sich auf standardisierte Weise speziellere Szenarien umsetzen, z.B. lassen sich Assertions erzeugen mit denen ein „Single Sign-On“ (SSO)-Szenario realisierbar ist. Dadurch können auch Web-Services die nicht direkt in Verbindung mit dem Benutzer stehen, Zugriff auf die gesamten Authentisierungsdaten des Benutzers erhalten.

B.3 WS-Policy

WS-Policy ermöglicht den Service-Anbieter und Service-Konsumenten die Formulierung von Sicherheitsanforderungen und Richtlinien an den Service-Request (bzw. Response) zu stellen. Der Service-Anbieter teilt dem Service-Konsumenten die Anforderungen und Kriterien mit, unter denen der Service-Request akzeptiert wird. WS-Policy ermöglicht Bekanntmachung und Durchsetzung von rudimentären Security-Policies auf Service-Ebene (vgl. [Melzer u. a., 2007, S.209]).

B.4 WS-Federation

WS-Federation ermöglicht die Definition und den Zusammenschluss von Vertrauenszonen. Der Standard setzt u.a. auf den Standards WS-Security, WS-Trust, WS-Policy und WS-SecureConversation auf. WS-Federation benötigt *Identity Provider*, die ein Mapping unterschiedlicher User Credentials aus unterschiedlichen Quellen und über Zonen- und Domänengrenzen hinweg erlauben. Im Wesentlichen geht es bei WS-Federation um eine Möglichkeit für Unternehmen, die gegenseitige Nutzung von Web-Services zu ermöglichen, obwohl die Beteiligten verschiedene Standards wie zum Beispiel Sicherheitstoken verwenden. Unter Verwendung von WS-Federation kann eine Single Sign-On Umgebung geschaffen werden (vgl. [Melzer u. a., 2007, S.212f]).

C Sicherheitsfunktionen der WebSphere DataPower XI50

Die WebSphere DataPower Integration Appliance XI50[1] gehört zu der Kategorie der XML-Gateways. Sie fungiert als Web-Service-Proxy und enthält zusätzliche Funktionen, die über die Merkmale einer XML-Firewall hinausgehen. Im folgenden finden Merkmale, die über Sicherheitsspezifische Aspekte einer XML-Firewall hinausgehen keine Betrachtung.

Die XML-Firewall bietet umfassende Schutzmaßnahmen für die Schutzziele:
Berechtigung, Integrität, Verbindlichkeit, Vertraulichkeit und Verfügbarkeit

C.1 Basis-Schutz

Bereits die Standardeinstellung eines XML-Firewall-Services bietet einem Web-Service Schutz gegen Kompromittierung der Verfügbarkeit.

Die XML-Firewall legt eine lokale Kopie der WSDL-Beschreibung ab. Der beschränkende XML-Parser validiert die SOAP-Nachrichten gegen das XML-Schema der WSDL-Beschreibung. Die Einstellungen des XML-Parsers können für jeden XML-Firewall-Service unter dem Menüpunkt XML-Threat-Protection individuell festgelegt werden.

Diese Angriffe werden zu diesem Zeitpunkt bereits durch die XML-Firewall gefiltert: XML Entity Expansion and Recursion Attacks, XML Wellformedness-based Parser Attacks, Memory Space Breach and Buffer Overflow Attacks, Public Key Dos Attacks, Resource Hijack Attacks, XPath Injection Attacks, Schema Poisoning Attacks und WSDL Scanning[2].

Die Wellformedness-based Parser Attacks gehören zu Angriffen durch XML-Inhalt (siehe Kapitel 3.7) mit schadhaften Bestandteilen. Die Memory Space Breach Attacks zielen auf den Verbrauch des Verfügbaren Hauptspeichers ab. Zu den übrigen Angriffstechnik finden sich in den Kapitel 3.1 bis 3.4 nähere Beschreibungen.

---

[1] http://www-306.ibm.com/software/integration/datapower/

[2] Die Bezeichnungen sind der Administrationsoberfläche der DataPower XI50 entnommen und können von den Bezeichnungen der Arbeit abweichen.

C.2 XML-Threat-Protection

Diese Schutzmaßnahmen erlauben eine Anpassung an individuelle Bedürfnisse:

- Single Message XML Denial of Service (XDoS) Protection
- Multiple Message XML Denial of Service (MMXDoS) Protection
- Message Tampering Protection
- SQL Injection Protection
- Protocol Threat Protection
- XML Virus (X-Virus) Protection
- Dictionary Attack Protection

Single Message XML Denial of Service (XDoS) Protection:
An dieser Stelle lassen sich Beschränkungen für das Verhalten des XML-Parsers festlegen. Die Einstellmöglichkeit umfasst Höchstwerte für Nachrichtengröße, Attribut-Anzahl, Datenstromlänge, Knotentiefe, Knotengröße und Anhangsgröße.

Multiple Message XML Denial of Service (MMXDoS) Protection:
An dieser Stelle lassen sich Beschränkungen für die Anzahl von erlaubten Nachrichten eines Hosts in einem Zeitintervall festlegen. Ebenso kann eine Anzahl von erlaubten Nachrichten an die XML-Firewall in einem Zeitintervall festgelegt werden. Zusätzlich kann die maximale Dauer des Requests festgelegt werden.

Abbildungen der Denial of Service Einstellmöglichkeiten befinden sich in Anhang Kapitel D.

C.3 Erweitere Schutzmaßnahmen

Unter Verwendung von digitalen Zertifikaten bietet die XML-Firewall folgende Schutzfunktionalität:

- Vollständige WS-Security Unterstützung
- Berechtigungsprüfung für Web-Services
- Transparente Signaturprüfung und Entschlüsselung von eingehenden Nachrichten
- Transparente Signierung und Verschlüsselung von ausgehenden Nachrichten

Die Web-Service-Firewall bietet den vollen Umfang von WS-Security und unterstützt die Sicherheitsziele Verbindlichkeit, Vertraulichkeit, Integrität, Authentifizierung und Autorisierung.

Die Berechtigungsprüfung bietet Authentifizierung und Autorisierung von Subjekten, damit diese sicher auf Web-Service-basierte Anwendungen zugreifen können. Sowohl kommerzielle als auch auf offenen Standards basierende Technologien wie WS-Security, SAML und LDAP können in die Zugriffskontrolle integriert werden.

Das Gerät kann aufwendige kryptografische Operationen auf eingehende oder ausgehende Nachrichten entsprechend festgelegter Policies anwenden. Übernimmt die Web-Service-Firewall diese ressourcenintensiven Schutzfunktionen, wird der Applikationsserver entlastet. Die zusätzlichen Systemressourcen des Applikationsservers stehen für die eigentliche Aufgabe des Web-Services zur Verfügung: Verarbeitung der Anwendungslogik.

D DataPower XI50 XML-Threat-Protection Einstellungen

An dieser Stelle werden die im Zuge der Arbeit verwendeten Einstellungen der XML-Threat-Protection durch Abbildungen der Bedienoberfläche und Auszügen der Hiletexte dargestellt. Die englischen Texte sind direkt übernommen worden.

D.1 Single Message XML Denial of Service (XDoS) Protection

Die Abbildung D.1 zeigt einen Screenshot der Single Message XML Denial of Service (XDoS) Protection. Die Tabele D.1 zeigt die zugehörigen Hilfetexte an.

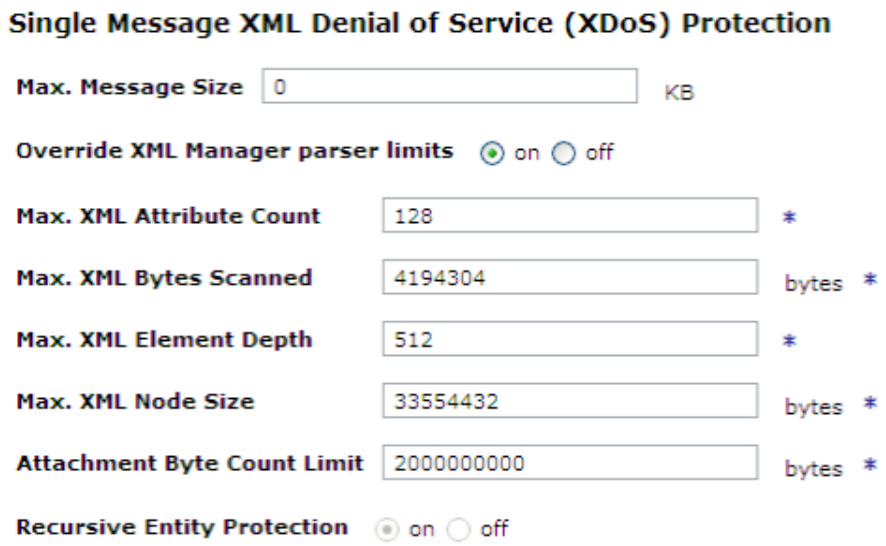

Abbildung D.1: Single Message XML Denial of Service (XDoS) Protection

Max. Message Size	Specifies the maximum size of a SOAP or XML message in kilobytes. If this value is 0, no limit is enforced.
Override XML Manager parser limits	Use the firewall parser limits instead of the parser limits in the XML Manager for this firewall. Firewall limits override XML Manager limits.
Max. XML Attribute Count	Defines the maximum number of attributes of a given element. If any of the parser limits are set in the XML Firewall, they will override those on the XML Manager.
Max. XML Bytes Scanned	Defines the maximum number of bytes scanned by the XML parser. This applies to any XML document that is parsed. If any of the parser limits are set in the XML Firewall, they will override those on the XML Manager. If this value is 0, no limit is enforced.
Max. XML Element Depth	Defines the maximum depth of element nesting in XML parser. If any of the parser limits are set in the XML Firewall, they will override those on the XML Manager.
Max. XML Node Size	Defines the maximum size any one node may consume. The default is 32 MB. Sizes which are powers of two result in the best performance. If any of the parser limits are set in the XML Firewall, they will override those on the XML Manager.
Attachment Byte Count Limit	Defines the maximum number of bytes allowed in any single attachments. Attachments that exceed this size will result in a failure of the whole transaction. If this value is 0, no limit is enforced.
Recursive Entity Protection	Protection against recursive entity attacks is always enabled.

Tabelle D.1: Hilfetexte: Single Message XML Denial of Service (XDoS) Protection

D.2 Multi Message XML Denial of Service (XDoS) Protection

Die Abbildung D.2 zeigt einen Screenshot der Multi Message XML Denial of Service (XDoS) Protection. Die Tabele D.2 zeigt die zugehörigen Hilfetexte an.

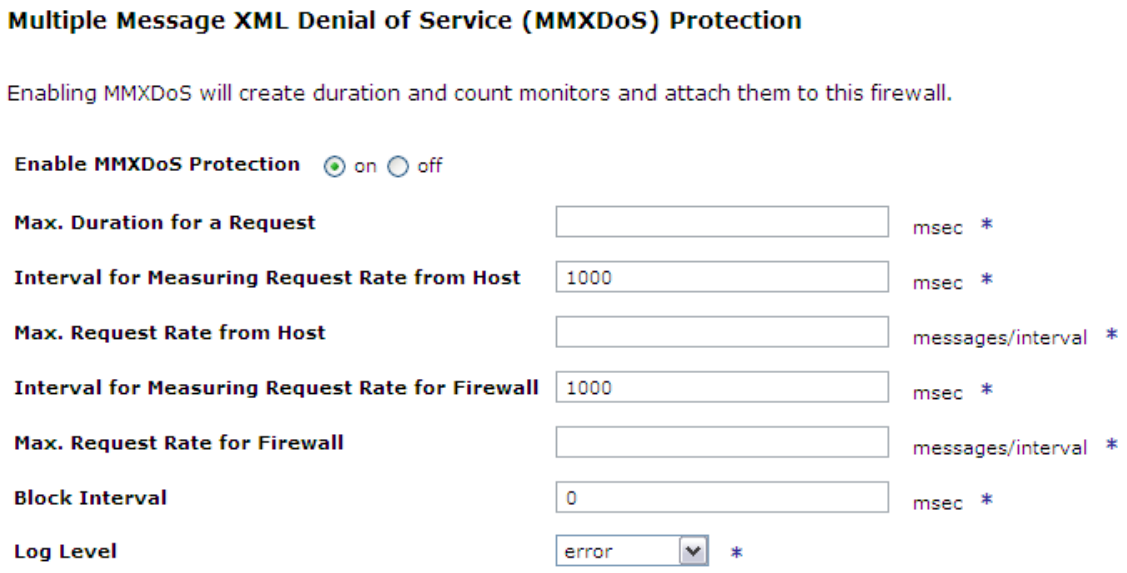

Abbildung D.2: Multi Message Denial of Service (MMXDoS) Protection

Enable MMXDoS Protection	Enable protection against multiple message denial of service attacks.
Max. Duration for a Request	The maximum amount of time to allow a request to take.
Interval for Measuring Request Rate from Host	The interval for measuring the rate at which requests come in from a single IP address.
Max. Request Rate from Host	The maximum rate at which requests should be accepted from a single IP address.
Interval for Measuring Request Rate for Firewall	The interval for measuring the rate at which requests come in from all IP addresses.
Max. Request Rate for Firewall	The maximum rate at which requests should be accepted from all IP addresses.
Block Interval	Specify an optional blackout period during which an over-threshold message type is denied service. The default value (0) indicates that while over-threshold messages are dropped, no service denial penalty is imposed.
Log Level	The severity of the log message when an over-threshold event happens. • emergency • alert • critical • error • warning • notice • information • debug Request HTTP Version

Tabelle D.2: Hilfetexte: Multi Message XML Denial of Service (XDoS) Protection

E Zusätzliche Mehrwerte der DataPower XI50

Im Rahmen der Arbeit konnte nur ein Teil des gesamten Schutzumfangs der DataPower XI50 (bzw. DataPower XS40) behandelt werden. Im folgenden werden die Vor- und Nachteile dargestellt, die durch den Einsatz der untersuchten Web-Service-Firewall entstehen.

Die Auflistung ist nicht formal und erhebt keinen Anspruch auf Vollständigkeit.

Vorteile:

- Die untersuchte XML-Firewall bietet einen Zusätzlichen Schutz bei Angriffen gegen die Verfügbarkeit.
- Die Schutzmaßnahmen können an individuell angepasst werden (siehe Anhang B1).
- Die Applikationsserver können durch eine vorgezogene Überprüfung der SOAP-Nachricht gegen Wohlgeformtheit und Validität entlastet und geschützt werden.
- Höhere Effektivität des Sicherheitsmanagement durch Konsolidierung der Schutzmaßnahmen. Gebündeltes Management an einem Punkt verringert die Kosten, indem es die Wartbarkeit verbessert und die Fehleranfälligkeit verringert.
- Als Infrastrukturkomponente gewährleistet die Web-Service-Firewall Schutz für alle angeschlossenen Web-Services - Unabhängig vom eingesetzten Applikationsserver oder Legacy-System.
- Benötigt die Implementierung der Schutzmaßnahmen eine Erweiterung oder Fehlerbereinigung, ist dies aufgrund der reduzierten Komplexität der DataPower XI50 im Vergleich zu einem Applikationsserver oder einem Legacy-System einfacher zu implementiere , weniger Fehleranfällig, erzeugt weniger Seiteneffekte und verursacht dadurch weniger Kosten.

Nachteile:

- Durch ein zentrales Sicherheitsmanagement können Fehlkonfigurationen unter Umständen Auswirkungen auf alle zu schützenden Web-Services haben.
- Mit wachsenden Anforderungen an die Performance kann der Einsatz einer XML-Firewall zu einem Ressourcen-Engpass führen.

- Es entstehen Anschaffungskosten und zusätzliche Schulungskosten des Wartungspersonals für ein fachgerechtes Sicherheitsmanagement im Umgang mit dem System.

Vorteile:

- Bietet Schutz für den Betrieb von Applikationsservern.
- Bietet „out of the box“ Schutz für die Verfügbarkeit von Web-Services .
- Bietet Schutz vor Datenmanipulation der angebundenen Backend-Systeme.
- Bietet Schutz vor unbefugter Benutzung von Web-Services.
- Bietet Schutz vor Manipulation und Vertraulichkeitsverletzung von Nachrichten.
- Enlastung des Servers durch vorgezogene XML-Validerung, Kryptografische Funktionen und Berechtigungsprüfung.
- Bietet Schutz unabhängig vom verwendeten Applikationsserver oder Legacy-System.
- Vereinfacht Entwicklung von Web-Services.
  - Keine komplexe Programmierung oder Administration von Schutzfunktionalität auf dem Applikationsserver notwendig.
  - Web-Services können im nachhinein mit Sicherheitsmerkmalen versehen werden.
  - Gilt auch für Lagcy-Systeme.
- Spart Kosten durch den Einsatz als Infrastruktur-Komponente.
  - Vereinfachte Administration durch Konsolidierung der betriebsweiten Sicherheit.
  - Reduktion von Konfiguration und Wartung sowie Verringerung von Fehlkonfiguration, 1 XML-Firewall vs. n Applikationsserver.
  - Ermöglicht einheitliche Schutzmaßnahmen und Policies.
  - Geringere Anzahl qualifizierter Mitarbeiter notwendig.
- Geringere Komplexität der DataPower Appliance im Vergleich zu einem Applikationsserver.
  - Erweiterung des Funktionsumfangs kostengünstiger als beim Applikationsserver.
  - Updates sind weniger Aufwendig, erlaubt kürzere Entwicklungszeiten
  - Updates sind weniger Fehleranfällig, als bei Applikationsservern.
  - Updates bewirken weniger Seiteneffekte, als bei Applikationsservern.

Nachteile:

- Kein vollständiger Ersatz für Sicherheit als Teil der Web-Service Entwicklung.
- Durch ein zentrales Sicherheitsmanagement können Fehlkonfigurationen unter Umständen Auswirkungen auf alle zu schützenden Web-Services haben.

- Mit wachsenden Anforderungen an die Performance kann der Einsatz einer XML-Firewall zu einem Ressourcen-Engpass führen.
- Es entstehen Anschaffungskosten und zusätzliche Schulungskosten des Wartungspersonals für ein fachgerechtes Sicherheitsmanagement im Umgang mit dem System.
- Mangelnde Flexibilität bei Sonderwünschen im Funktionsumfang, Abhängigkeit vom Hersteller.

F Tomcat 6.0 mit Axis2 im Vergleich

Der exemplarische Web-Service wurde unter Verwendung des RAD-Wizards auf einem *Apache Tomcat 6.0*[1] mit der SOAP-Engine *Apache Axis2*[2] installiert. Der Web-Service konnte ohne Quelltextänderungen, unter Zuhilfenahme des Web-Service-Wizards generiert und verwendet werden. Der Tomcat 6.0 Server und die Axis2 SOAP-Engine werden in der Standardeinstellung getestet.

Bei einer Durchführung der Angriffe aus Kapitel 3.7 konnte ermittelt werden, dass die Axis2 SOAP-Engine ebenso anfällig für Angriffe gegen die Verfügbarkeit ist. Da dem Lieferumfang kein geeignetes Monitoring-Werkzeug gehört, wird die Messung der Prozessorauslastung mit den Mitteln des Betriebssystems durchgeführt.

Die SOAP-Engine konnte durch einzelne Nachrichten nicht in der Erreichbarkeit kompromittiert werden. Der Web-Service war immer erreichbar, selbst wenn die Prozessorauslastung für längere Zeit bei 100% lag.

Tabelle F.1 zeigt einen Auszug der Angriffe, die den Web-Service für längere Zeit voll ausgelastet haben.

Angriffsname	**CPU-Zeit (s)**	Wert
Extra-long Namespaces	238	10^7
Extra-long Attributenames	106	10^7
Extra-long Tagnames	200	10^7
SOAP Array Attack	78	offset mit $2,3 * 10^7$

Tabelle F.1: Auszug der Axis2 Versuchsresultate

Die Standardeinstellung der Axis2 SOAP-Engine verwendet keine Schema-Validierung gegen das XML-Schema der WSDL-Beschreibung. Sie akzeptiert wohlgeformte Nachrichten, auch wenn diese nicht dem Schema entsprechen. Das Verhalten entspricht dem WAS.

Der XML-Parser der SOAP-Engine arbeitet, analog zu der des WebSphere Application Servers, ebenfalls ohne Beschränkungen. Axis2 verwendet wie der WAS einen SAX-Parser.

[1] http://tomcat.apache.org/

[2] Apache eXtensible Interaction System 2, http://ws.apache.org/axis2/

Der XML-Parser ist durch einzelne Nachrichten nicht in der Erreichbarkeit kompromittierbar. SOAP-Nachrichten mit eingebetten Document Type Declarations werden zuverlässig erkannt, die Verarbeitung wird unmittelbar abgebrochen.

Erst der Einsatz von Flooding Attacks konnte zu einer Kompromittierung der Erreichbarkeit führen. Bei den Angriffen war eine Prozessorauslastung für 300 Sekunden und mehr keine Seltenheit. Beispielsweise hat die Verwendung von 50 parallelen SOAP-Array Attacks mit einem Offset von $2,3 * 10^7$ zu einer erfolgreichen Kompromittierung geführt. Der Web-Service hat nicht mehr reagiert.

Abschließend lässt sich sagen, dass das Gespann aus Tomcat 6.0 und Axis2 auch anfällig für Angriffe auf die Verfügbarkeit ist. Das Resultat der Angriffe kann neben einer starken Verzögerung der Antworten des Web-Services auch ein Totalausfall sein, sobald Flooding Attacks eingesetzt werden.

G Auszug der SOAP-Faults

Es folgen die SOAP-Faults die durch den WebSphere Application Server 6.1 bzw. der WebSphere DataPower XI50 erzeugt wurden.

G.1 Direkte Angriffe

Im folgenden werden die drei SOAP-Faults gezeigt, die der *Websphere Application Server* im Rahmen eines „fehlerhaften“ SOAP-Requests als Reply versendet.

Der Quellcode G.1 zeigt einen SOAP-Fault der durch einen nicht bestimmten Fehler ausgelöst wurde. Es handelt sich um eine generische Fehlermeldung.

```
...
<soapenv:Body>
  <soapenv:Fault>
    <faultcode>soapenv:Server.generalException</faultcode>
    <faultstring>
      <![CDATA[WSWS3400I: Info: unexpected exception.]]>
    </faultstring>
  </soapenv:Fault>
</soapenv:Body>
...
```

Quelltext G.1: SOAP-Fault: Server.generalException, unexpected exception

Der Quellcode G.2 zeigt einen SOAP-Fault der durch eine SAXParserException ausgelöst wurde. Der Fehler tritt meist in Verbindung mit sehr hohen Werten auf.

```
...
<soapenv:Body>
  <soapenv:Fault>
    <faultcode>soapenv:Server.generalException</faultcode>
      <faultstring>
        <![CDATA
         [org.xml.sax.SAXParseException: The root element
         is required in a well-formed document. Message
         being parsed
         :]
        ]>
     </faultstring>
 </soapenv:Fault>
</soapenv:Body>
...
```

Quelltext G.2: SOAP-Fault: Server.generalException, SAXParseException

Der Quellcode G.3 zeigt einen SOAP-Fault der bei der Verwendung von DTDs innerhalb des SOAP-Requests durch einen unsupportedSaxEvent ausgelöst wurde. Die Fehlermeldung besagt, dass sich für den Event keine registrierte Aktion finden lässt.

```
...
<soapenv:Body>
  <soapenv:Fault>
    <faultcode>soapenv:Server.generalException</faultcode>
      <faultstring>
        <![CDATA[org.xml.sax.SAXException: Can't find resource
        for bundle com.ibm.ws.webservices.engine.resources.
        ProjectResourceBundle, key unsupportedSAXEvent Message
        being parsed:
        ]]>
    </faultstring>
  </soapenv:Fault>
</soapenv:Body>
...
```

Quelltext G.3: SOAP-Fault: Server.generalException, unsupportedSaxEvent

G.2 Angriffe hinter der XML-Firewall

Die XML-Firewall reagiert im Falle eines Verstoßes gegen die definierten Regeln mit einem der folgenden SOAP-Faults oder verwirft die Anfrage vollständig:

```
..
  <env:Body>
    <env:Fault>
      <faultcode>env:Client</faultcode>
      <faultstring>Malformed content (from client)</faultstring>
    </env:Fault>
  </env:Body>
...
```

Quelltext G.4: SOAP-Fault: Malformed content (from client)

```
...
  <env:Body>
    <env:Fault>
      <faultcode>env:Client</faultcode>
      <faultstring>Internal Error (from client)</faultstring>
    </env:Fault>
  </env:Body>
...
```

Quelltext G.5: SOAP-Fault: Internal Error (from client)

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 4. Juni 2008	
Ort, Datum	Unterschrift