Fachhochschule
Bonn-Rhein-Sieg

University
of Applied Sciences

Fachbereich
Angewandte Informatik

Sicherheitsrisiken und Sicherheitskonzepte von XML Web-Services, dargestellt an einem E-Business-Szenario.

Diplomarbeit

von

Michael Hildebrandt

(info@michael-hildebrandt.de)

Bearbeitungszeitraum: 24.01.2003 - 24.04.2003

Betreuer: Prof. Dr. Martin Leischner
Fachhochschule Bonn-Rhein-Sieg
Prof. Dr. Hartmut Pohl
Fachhochschule Bonn-Rhein-Sieg

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Bonn, den 24.04.2003

.....
(Michael Hildebrandt)

Inhaltsverzeichnis

1 Einleitung

XML–Web–Services werden die Informationstechnologie ähnlich revolutionieren wie in den vergangen zehn Jahren die Ansätze des Client–Server–Computing oder die der webbasierten Anwendungen. Doch während sich die großen Unternehmen wie z.B. Microsoft, IBM oder Sun fortlaufend zu Web–Services bekennen, und diese bereits zu einer Kerntechnologie Ihrer Produkte gemacht haben, zögert die Masse der Unternehmen noch beim Einsatz.

Die Gründe sind in den noch ungeklärten Fragen nach praktischer Sicherheit von Web–Services zu suchen. In den Spezifikationen von SOAP, WSDL und UDDI, den Basistechnologien heutiger Web–Services, ist auf die Implementierung von Sicherheit verzichtet worden. Die benötigten Sicherheitsmechanismen mussten daher von den Standardisierungsorganisationen in separaten Standards untergebracht werden. Die für einen sicheren Einsatz von Web–Services relativ hohe Anzahl benötigter Standards und deren sehr flexible Einsatzmöglichkeiten, haben zunächst einen abschreckenden Charakter.

Ziel dieser Arbeit ist die Behandlung der Sicherheit von Web–Services zum heutigen Zeitpunkt, durch die exemplarische Absicherung eines typischen E–Business–Szenarios. Zu diesem Zweck werden zunächst die Grundlagen der Web–Services Technologie vermittelt. Es folgt die Einführung eines Web–Services Szenarios, auf das in den nachfolgenden Kapiteln bei Bedarf zurückgegriffen wird. Die Beschreibung der bestehenden Sicherheitskonzepte und -risiken bildet den gewichtigsten Teil dieser Arbeit. Sie finden in der abschließenden Absicherung des bereits eingeführten E–Business–Szenarios Verwendung.

2 Grundlagen der Web–Services Technologie

Unter dem Begriff der Web–Services wird die lose Kopplung heterogener Systeme verstanden. Die assoziierte Technologie führt zu diesem Zweck eine neue Abstraktionsebene ein. Sie ermöglicht sowohl das Anbieten als auch die Verwendung von Diensten unabhängig von der verwendeten Plattform oder Programmiersprache. Die Komplexität der bestehenden Technologien wird dabei durch die Verwendung von einfach zu verstehenden Nachrichtenformaten, Schnittstellenbeschreibungen und gängigen Transportprotokollen (http) gekapselt. Web–Services vereinheitlichen so die Kommunikation verschiedenartiger Systeme, die bei B2B[1], B2C[2], B2BI[3] und EAI[4] bisher für Komplikationen gesorgt haben. Ein weiterer Aspekt von Web-Services ist die Möglichkeit, Dienste in einem Verzeichnis zu publizieren. Diese Dienste können dann von potentiellen Dienstnehmern gesucht, gefunden und ggf. genutzt werden. Die zugrundeliegende Architektur erinnert dabei stark an die in Abbildung 1 dargestellte Funktionsweise von Branchenbüchern. Im Folgenden werden die fünf Schritte des Branchenbuch–Szenarios stichpunktartig beschrieben:

1. Dienstleister publiziert seine Dienstleistung in einem Branchenbuch.
2. Kunde sucht in dem Branchenbuch nach einer bestimmten Dienstleistung (z.B. Maurer).
3. Dienstleister (hier Maurer) wird im Branchenbuch gefunden und der Kunde erhält Informationen zur Nutzung der Dienstleistung (Name, Adresse, usw.).
4. Kunde setzt sich unter zu Hilfenahme der erhaltenen Informationen mit dem Dienstleister (Mauer) in Verbindung und vergibt den Auftrag.
5. Dienstleister (Mauer) erbringt die angeforderte Dienstleistung (baut eine Mauer).

[1] Business to Business
[2] Business to Consumer
[3] Business to Business Integration
[4] Enterprise Application Integration

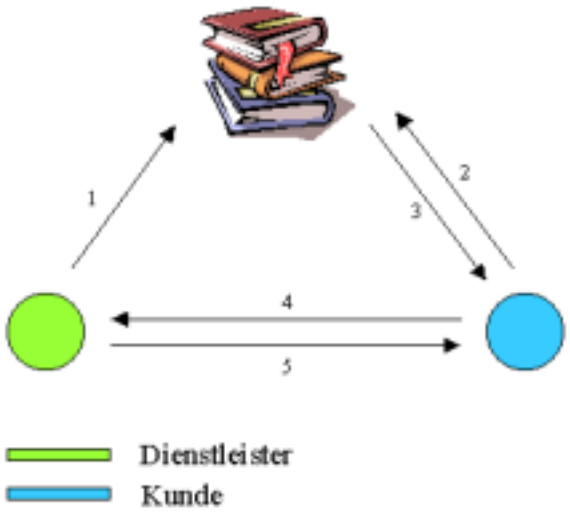

Abbildung 1: Branchenbuch Szenario

Um die Analogien zu verdeutlichen, wird mit Abbildung 2 die Web–Services Architektur vorgestellt. Der UDDI Verzeichnisdienst übernimmt dabei die Aufgabe des Branchenbuches. Fasst man die Schritte (2,3) und (4,5) jeweils zu einem Schritt zusammen, läßt sich sagen, dass die zugrundeliegende Architektur aus dem

1. publizieren eines Dienstes (einmalig)
2. entdecken eines Dienstes (beliebig oft)
3. nutzen eines Dienstes (beliebig oft)

besteht. Bei allen Schritten findet die Kommunikation über das SOAP[5] Protokoll statt. Während das Publizieren und Entdecken eines Dienstes über den Austausch von UDDI Nachrichten[6] erfolgt, kommt bei der Nutzung eines Dienstes ein vom Dienstleister vorgegebenes Nachrichtenformat zum Einsatz. Alle hierbei verwendeten Nachrichten, Protokolle und Adressinformationen werden vom Dienstleister in Form einer WSDL[7] Datei beschrieben.

[5]Ein Protokoll zum Austausch XML basierter Nachrichten

[6]Ein von der UDDI Spezifikation vorgegebenes Nachrichtenformat, zur Kommunikation mit dem Verzeichnisdienst.

[7]Eine Schnittstellenbeschreibungssprache für Web–Services.

Die Grundlage von SOAP, UDDI und WSDL ist XML. Gemeinsam bilden diese den Web–Services Technologiekern.

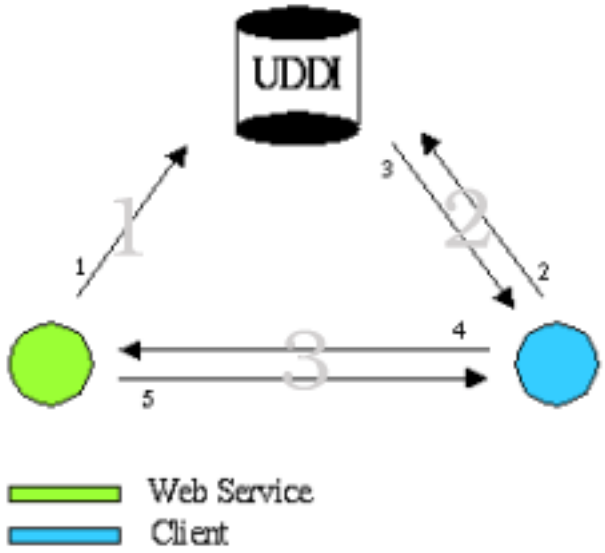

Abbildung 2: Web–Services Szenario

2.1 Definition: Web–Services

Bei der Suche nach einer formalen Definition von Web–Services in Büchern oder dem Internet, wird man mit einer Fülle verschiedener Auslegungen konfrontiert. In dieser Arbeit wird auf die Definition des W3C[8] zurückgegriffen, da sich die meisten der Web–Services Kerntechnologien (XML, SOAP und WSDL) unter dessen Obhut befinden. Nach [WS-ARCH] wird ein Web-Service wie folgt definiert:

„[A Web service is a software system identified by a URI [RFC 2396], 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.]“

Obwohl sich für den Einsatz von Web-Services XML, SOAP, WSDL und

[8]World Wide Web Consortium

UDDI inzwischen etabliert haben, ist auffällig, dass in der Definition formal nur auf XML als Nachrichtenformat bestanden wird. Die Begründung des [WS-ARCH] hierfür ist etwas widersprüchlich. Zu Beginn dieser wird noch auf die mögliche Verdrängung von SOAP oder WSDL durch andere Technologien hingewiesen, derer Entwicklung man nicht im Wege stehen möchte. Einige Sätze später besteht man jedoch auf den Einsatz eben dieser Technologien, falls dies geeignet sei. Das von der OASIS[9] standardisierte UDDI wird in diesem Zusammenhang nicht erwähnt.

2.2 XML

Bei der e**X**tensible **M**arkup **L**anguage handelt es sich, wie aus der formalen Definition des vorangegangenen Abschnitts ersichtlich, um die Basis von Web-Services. XML stellt formal eine Untermenge von SGML[10] dar, welches wegen seiner Komplexität bisher kaum Anwendung fand. In diesem Kapitel werden nur die wichtigsten Eigenschaften von XML erläutert. Für einen schnellen und leichten Einstieg in die Thematik sei [Born 01] empfohlen. Weiterführende Informationen zu diesem Thema liefert [Goldfarb 99].

2.2.1 XML Grundlagen

XML ist eine Metasprache, die bei der Erstellung strukturierter Dokumente Verwendung findet. Die eigentlichen Informationen dieser Dokumente werden dabei von Auszeichnungselementen umgeben. Abbildung 3 stellt ein einfaches Beispiel für den Aufbau eines XML Dokumentes vor. XML definiert, im Gegensatz zu Auszeichnungssprachen wie z.B. HTML, nur Regeln für die Verwendung von Auszeichnungselementen und nicht die Elemente an sich. Um Dokumentstrukturen[11] festzulegen, die das Erstellen XML basierter Sprachen wie z.B. XHTML oder die Definition fester Datenformate ermöglichen, bietet XML zwei Mechanismen:

[9] Organisation for the Advancement of Structured Information Standards

[10] Standard Generalized Markup Language (ISO 8879)

[11] Bei festgelegten Dokumentstrukturen wird in Zusammenhang mit XML auch von Sprachen gesprochen

```
<?xml version='1.0' encoding='utf-8'?>
  <product>
    <name farbe='rot'>Kamm</name>
    <preis>1,49</preis>
  </product>
```

Abbildung 3: XML Dokumentstruktur

1. DTD (Document Type Definition)
2. XML Schema

Sowohl die DTD als auch das XML Schema ermöglichen es XML Datenstrukturen zu definieren. Während das XML Schema zu diesem Zweck auf eine XML Syntax zurückgreift, nutzt die DTD dafür eine eigene Sprache.

Zur syntaktischen Kontrolle eines XML Dokuments existieren Parser, die XML Dokumente auf

- Wohlgeformtheit
- Gültigkeit

überprüfen können. Dabei ist ein XML Dokument, das nach den Regeln der XML Spezifikation korrekt formuliert ist, als wohlgeformt anzusehen. Gültig hingegen ist ein XML Dokument nur, wenn es über die Wohlgeformtheit hinaus, in Bezug auf eine existierende DTD oder XML Schema Beschreibung korrekt ist.

2.2.2 Vorteile von XML

XML weist im Gegensatz zu anderen Datenformaten erhebliche Vorteile auf. Im Folgenden sollen nur die Wichtigsten davon genannt werden.

- Für Menschen lesbar
- Leicht zu erlernen

- Plattform- und Programmiersprachen unabhängig (Unicode [12])
- Ein offener Standard
- Flexibel und erweiterbar
- Einfache Prüfung der syntaktischen Korrektheit von Daten (Parser)
- Breite Unterstützung nahezu aller „Global Player“ im IT Sektor

2.2.3 XML Familie

Ist von der Arbeit mit XML die Rede, so ist häufig die mit der XML-Familie gemeint. Erst das Zusammenwirken einiger Familienmitglieder machen oft die Arbeit mit XML sinnvoll. Abbildung 4 stellt die XML Familie vor.

2.3 SOAP

Das **S**imple **O**bject **A**ccess **P**rotocol ist ein XML basiertes Protokoll, das den plattformübergreifenden Austausch von XML Nachrichten auf einem standardisierten Weg beschreibt. Dabei beschreibt die Spezifikation im Wesentlichen folgende Hauptteile:

- **Envelope**: Element das die SOAP Nachricht umschließt.
- **Header**: Element mit Verarbeitungsinformationen. Diese Informationen können bestimmten Knotenpunkten auf einem Nachrichtenpfad zugewiesen werden.
- **Body**: Element, das die eigentliche Nachricht enthält. Mit Nachricht ist hier ein beliebiges wohlgeformtes XML Dokument gemeint.
- **Message path**: Ermöglicht die Verwendung von Nachrichtenpfaden, um Nachrichten einen Mehrwert zu geben.
- **RPC**: Definiert einen Mechanismus um RPC Aufrufe unter Verwendung von SOAP zu realisieren.

[12] Plattform, Programm- und Sprachenunabhängiger Zeichensatz. Mehr Informationen unter: http://www.unicode.org/

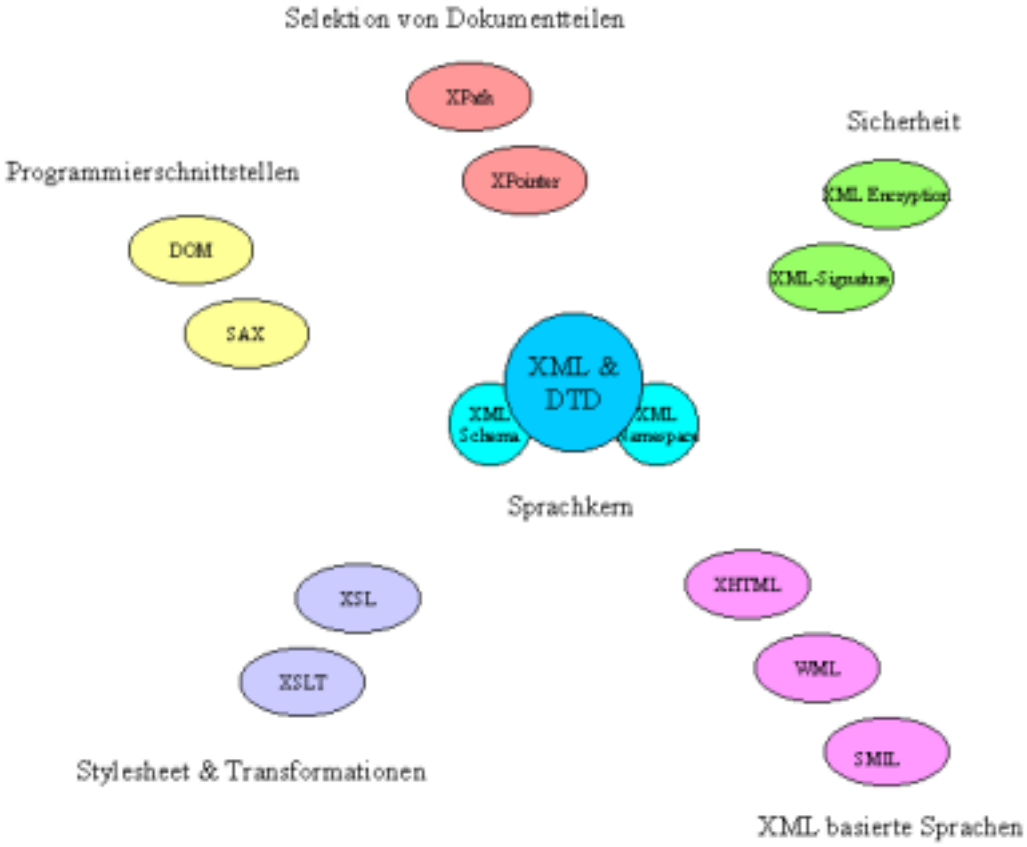

Abbildung 4: Die XML Familie

- **Encoding**: Spezifiziert die Darstellung einfacher und komplexer Datentypen.
- **Faults**: Beschreibt das Handhaben von Fehlern bei der Verarbeitung.

Der Einsatz von Envelope, Header und Body wird in Abschnitt 2.3.1 vorgestellt. Den anderen Punkten wird mit 2.3.2, 2.3.3, 2.3.4 und 2.3.5 jeweils ein eigener Abschnitt zuteil.

2.3.1 SOAP Nachrichtenaufbau

Der Aufbau einer SOAP Nachricht besteht aus den in Abbildung 5 (s.u.) dargestellten drei Teilen.

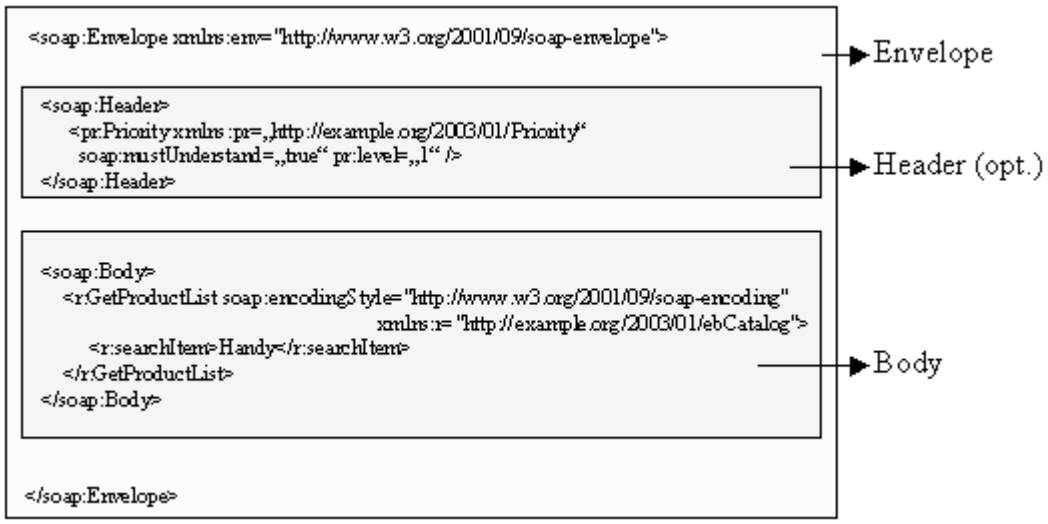

Abbildung 5: SOAP Nachrichtenaufbau

1. Mit dem <Envelope> Element wird eine SOAP Nachricht umschlossen. Jede SOAP Nachricht muss genau ein, die Nachricht umschließendes, <Envelope> Element enthalten.
2. Der <Header> Bereich ist optional und soll Verarbeitungsanweisungen beinhalten. Sinnvolle Einsatzgebiete von Headern in SOAP Nachrichten

werden von [Graham 02] folgendermaßen zusammengefasst. „[...] authentication and authorization, transaction managment, payment processing, tracing and auditing, and so on.“ Die einzelnen Verarbeitungsanweisungen werden auch als Headerblöcke bezeichnet, und können mit dem URI[13] Mechanismus einem Verarbeiter zugeordnet werden.

3. Die eigentliche Nachricht wird im <Body> Element platziert. Unter einer Nachricht versteht sich hier ein beliebiges, wohlgeformtes XML Dokument.

2.3.2 SOAP Nachrichtenpfade

SOAP bietet die Möglichkeit der Nutzung von Nachrichtenpfaden. Bei diesem Verfahren wird eine Nachricht über einen oder mehrere Vermittler (engl.: intermediaries) zum Zielknoten geleitet. Abbildung 6 zeigt einen solchen Nachrichtenpfad. Den Vermittlern kommt dabei die Aufgabe zu, den Nachrichten

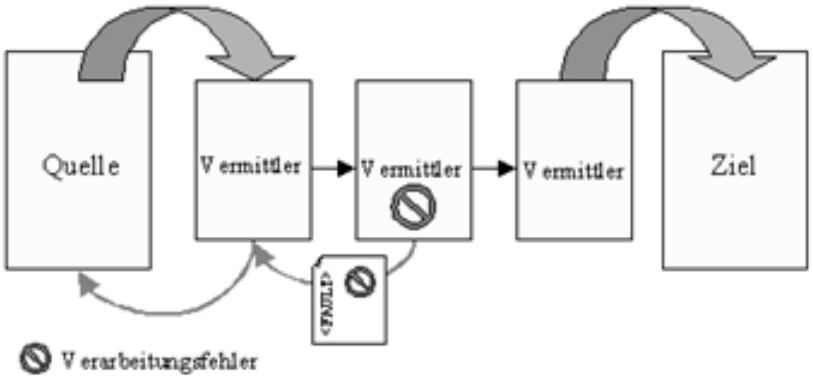

Abbildung 6: SOAP Nachrichtenpfad

einen Mehrwert in Form von zusätzlichen Informationen, einer Verschlüsselung o.ä. zu geben. Benötigte Verarbeitungsanweisungen entnehmen die Vermittler dabei dem Header der Nachricht. Die Zuordnung der einzelnen Headerblöcke zu den jeweiligen Vermittlern erfolgt über das von SOAP spezifizierte actor Attribut. Diesem Attribut wird eine URI zugewiesen, die den verantwortlichen Vermittler eindeutig bezeichnet. Um die korrekte Verarbeitung

[13] Uniform Resource Identifier [RFC 2396]

durch einen Vermittler zu gewährleisten, kann auf das ebenfalls von SOAP spezifizierte mustUnderstand Attribut zurückgegriffen werden. Ist dieses Attribut gesetzt, muss der Vermittler in der Lage sein, die an ihn gerichtete Anfrage korrekt zu bearbeiten, oder anderenfalls einen Fehler (engl. fault) melden. Der Nachrichtenpfad wäre in diesem Fall unterbrochen (siehe Abbildung 6).

2.3.3 SOAP, HTTP und RPC

Eine durchaus wichtige Eigenschaft von SOAP liegt in dem zur Verfügung gestellten RPC[14] Mechanismus. Aufgrund des universellen Nachrichtenformats (XML) ist es nun möglich, Funktionen nahezu jeder Programmiersprache und Plattform als Web-Service zur Verfügung zu stellen oder zu nutzen. Dass SOAP fast immer auf http als Transportprotokoll aufsetzt, gilt insbesondere für den RPC Mechanismus. Zu begründen ist dies mit dem gleichen Kommunikationsverhalten, da beiden Verfahren eine Anfrage / Antwort Architektur zugrunde liegt. Abbildung 7 veranschaulicht einen SOAP–RPC über http Aufruf.

Im Gegensatz zu üblichen Anfragen, weist der http Header bei dem Versand von SOAP Nachrichten folgende Unterschiede auf:

1. Content-Type: text/xml; charset=„utf-8“
2. SOAPAction: „Eine-URI“ wird an den http Header angehangen

Der erste Punkt teilt dem Empfänger mit, dass es sich bei den transportierten Daten innerhalb der Anfrage um XML in dem angegebenen Unicode Format (hier ASCII kompatibel) handelt. Bei einer gewöhnlichen http Anfrage erschiene hier 'text/html' als Content-Type. Punkt 2 benennt, durch die Verwendung einer URI, eine für die Verarbeitung der Anfrage verantwortliche Anwendung.

Der Aufbau von RPC Anfrage- und Antwortnachrichten folgt einem einfachen Schema. Bei der Anfrage (Funktionsaufruf) sind Funktionsname und

[14] **R**emote **P**rocedure **C**all bezeichnet den Rechnerübergreifenden Aufruf von Programmcode über ein Netzwerk.

Abbildung 7: SOAP RPC Aufbau

Parameter als XML Elemente darzustellen. Die zugehörige Antwortnachricht ist durch den Funktionsnamen mit einem angehangenen „Response“ gekennzeichnet, wobei eine solche Antwort mehr als einen Rückgabeparameter enthalten darf. Abbildung 8 zeigt ein nach diesem Muster aufgebautes Anfrage- und Antwortpaar. Bei den dargestellten Nachrichten handelt es sich jeweils um den Inhalt des <Body> Elements einer entsprechenden SOAP Nachricht.

Anfrage

<Funktionsname>
  <Parameter_1>parameter</Parameter_1>
  <Parameter_n>parameter</Parameter_n>
</Funktionsname>

Antwort

<Funktionsname**Response**>
  <Rückgabeparameter_**1**>rückgabewert</Rückgabeparameter_1>
  <Rückgabeparameter_**n**>rückgabewert</Rückgabeparameter_n>
</Funktionsname**Response**>

Abbildung 8: Aufbau eines SOAP-RPC Aufrufs

2.3.4 SOAP Encoding

Bei der Verwendung unterschiedlicher Datentypen in SOAP Nachrichten, ist die Anwendung von Regeln zur einheitlichen Darstellung dieser erforderlich. SOAP Encoding stellt ein solches Regelwerk in Form eines XML Schemas[15] zur Verfügung. Dabei verwendet dieses, die von der XML Schema Spezifikation [SCHEMA-2] vordefinierten Datentypen. Bei Bedarf kann auch auf eigene Regelwerke zur Darstellung komplexer Datentypen zurückgegriffen werden. Das angewandte Regelwerk ist, wie in Abbildung 9 dargestellt, unter Verwendung des 'encodingStyle' Attributs zu setzen.

[15] zu finden unter: http://www.w3.org/2002/12/soap-encoding

```
<soap:Envelope xmlns:soap='http://www.w3.org/2001/12/soap-envelope'
      xmlns:m='http://example.org/headers'
      soap:encodingStyle='http://www.w3.org/2001/12/soap-encoding'>
```

Abbildung 9: Setzen des 'encodingStyle' Attributs

2.3.5 SOAP Fehlerbehandlung

Tritt bei der Verarbeitung einer SOAP Nachricht ein Fehler auf, besteht die SOAP Spezifikation auf das Erstellen einer Fehlermeldung, die dann in einer SOAP Nachricht an den Versender der fehlerhaften Nachricht geschickt wird. Die Fehlermeldung in einem <Fault> Element eingeschlossen, das genau ein Mal und als einziges Element im <Body> einer SOAP Nachricht vorkommen darf. Die Spezifikation definiert fünf Grundtypen von Fehlermeldungen:

- **VersionMismatch**: Der Verarbeiter der Nachricht ist auf einen für ihn ungültigen Namensraum bei der Deklaration des <Envelope> Elements gestoßen.
- **MustUnderstand**: Die Anweisungen eines Headerblocks konnten durch den verantwortlichen Prozeß nicht verstanden werden, obwohl dieser mit einem mustUnderstand Attributwert von '1' gekennzeichnet war.
- **DataEncodingUnknown**: Ein Kindelement des SOAP <Header> oder <Body> Elements verwendet Datentypen, die von dem zur Verarbeitung adressierten Knoten nicht unterstützt werden.
- **Client**: Das Benutzen der Client–Fehlerklasse weist darauf hin, dass die Nachricht den Empfänger in einem ungültigen Format erreicht hat oder nicht die zur Verarbeitung notwendigen Informationen enthielt.
- **Server**: Die Server–Fehlerklasse findet Verwendung, wenn die Nachricht, aus Gründen die nichts mit dem eigentlichen Inhalt dieser zu tun haben, nicht verarbeitet werden konnte.

SOAP spezifiziert zur detaillierten Beschreibung eines Fehlers einige Unterelemente (siehe [SOAP]).

2.4 WSDL

Die **W**eb **S**ervices **D**escription **L**anguage wurde erarbeitet, um einen standardisierten Weg zur Beschreibung von Web-Services Schnittstellen zu schaffen. Aufgrund des öffentlichen Wesens von Web-Services, kommt der Beschreibung ihrer Schnittstellen eine große Bedeutung zu. Das Nutzen von WSDL ermöglicht einem Client:

- die definierte Nutzung eines Web-Services.
- das Auffinden eines Dienstes in einer UDDI, der über eine bestimmte, evtl. bereits implementierte, Schnittstelle verfügt.

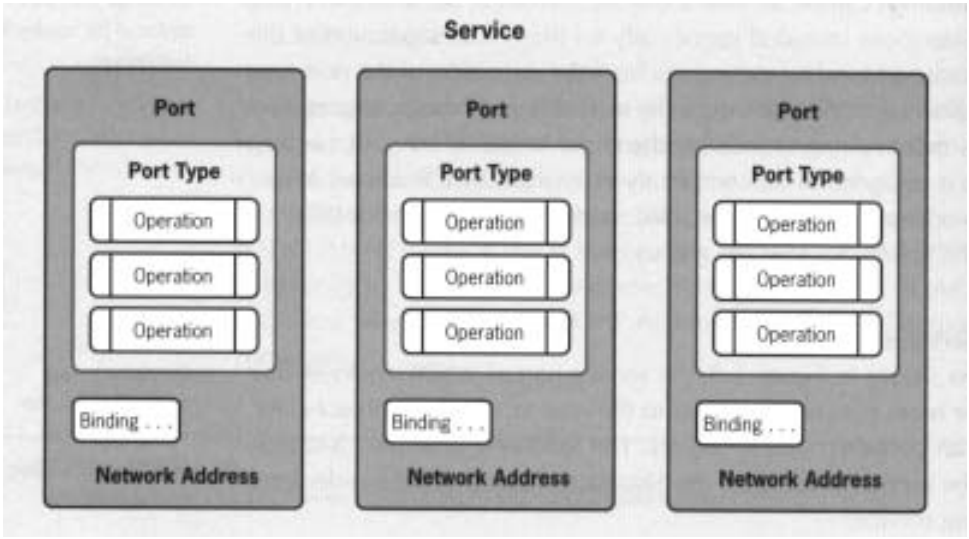

Abbildung 10: [Newcomer 02] WSDL Architektur

Abbildung 10 zeigt den Aufbau einer mit WSDL beschriebenen Schnittstelle. Die in der Grafik verwendeten Begriffe werden im Folgenden erklärt.

- **Operation**: Sie bilden Funktionsaufrufe ab. Operations beziehen sich dabei auf konkrete Nachrichten (engl. messages), die den Funktionsaufruf und evtl. vorhandene Rückgabewerte definieren. Ob Rückgabewerte vorgesehen sind, hängt vom Typ der Operation ab. WSDL unterscheidet dabei nach [Newcomer 02] zwischen den folgenden vier Typen:

  - One-way: Eine vom Sender (Client) verschickte Nachricht erwartet keine Antwort.
  - Request / response: Der Client erwartet als Reaktion auf den Versand einer Nachricht eine Antwortnachricht. Dieser Typ findet bei der Implementierung des RPC Mechanismus Verwendung.
  - Solicit response (noch nicht definiert): Eine Nachricht wird an den Client verschickt, ohne dass dieser vorher eine konkrete Anfragenachricht verschickt hat. Der Versand der Nachricht könnte z.B. von einem 'ping' ausgelöst werden.
  - Notification (noch nicht definiert): Eine Nachricht wird an einen Client geschickt, ohne dass für den Erhalt dieser eine Anfrage existiert. Ein Dienst, der diese Operation unterstützt, könnte z.B. mit vorher registrierten Empfängern zusammen arbeiten. Das Kommunikationsverhalten ist mit dem Versand von Newslettern[16] vergleichbar.

- **Port Type**: Definiert durch das Zusammenfassen mehrerer Operationen eine abstrakte Funktionsbibliothek.
- **Binding**: Bindet die Operationen an ein konkretes Nachrichtenformat und Transportprotokoll.
- **Port**: Kombiniert operations und bindings mit einer Netzwerkadresse.
- **Service**: Ist ein Container für einen oder mehrere Ports.

WSDL Dateien werden sowohl Client- als auch Serverseitig meistens automatisch verarbeitet. Dazu existieren eine Vielzahl an Tools, die imstande sind, entweder aus vorhandenem Quellcode eine WSDL Datei, oder aus einer vorhandenen WSDL Datei ein Klassen- oder Funktionsgerüst zu erstellen. Die Nutzung von WSDL Editoren [17] erleichtert die manuelle Erstellung von WSDL Dateien erheblich.

[16] Eine Email die in regelmäßigen Abständen oder sporadisch an ein vorher registriertes Publikum verschickt wird.

[17] z.B. der CapeClear Editor (http://www.capeclear.com)

2.5 UDDI

Universal **D**escription, **D**iscovery and **I**ntegration definiert sowohl die Möglichkeit eigene Web-Services im Internet zu publizieren, als auch das Auffinden dieser auf schnelle und dynamische Art. Diese Funktionalität wird von einer UDDI durch den Austausch XML basierter Nachrichten über SOAP realisiert. Der UDDI Verzeichnisdienst darf demnach bereits als Web–Service bezeichnet werden. Um eine gezielte menschliche oder maschinelle Suche nach benötigten Diensten zu ermöglichen, werden die publizierten Informationen in drei Rubriken aufgeteilt, den White-, Yellow- und Green Pages:

1. **White Pages** speichern unternehmensspezifische Informationen. Gemeint sind hier etwa der Firmenname, eine multilinguale Beschreibung des Unternehmens, Kontaktinformationen (Ansprechpartner, Telefonnummern, usw.), sowie bekannte Identifikationskürzel des Unternehmens.

2. **Yellow Pages** ordnen das Unternehmen in verschiedene Kategorien ein. Unter Kategorien versteht sich hier Industriesparte, angebotene Produkte oder Dienstleistungen, sowie geographischer Standort.

3. **Green Pages** enthalten Informationen über Geschäftsprozesse, Schnittstellenbeschreibungen (z.B. WSDL) und Verbindungsinformationen.

Während Informationen aus den White- und Yellow Pages für die Beurteilung durch Menschen gedacht sind, können die der Green Pages zur Implementierung von Automatismen verwendet werden. Eine Anwendung, die hier bereits implementierte Schnittstellen findet, kann die gefundenen Dienste sofort nutzen.
UDDI Verzeichnisse können öffentlich oder auch unternehmensintern Verwendung finden. Besonders in großen Unternehmen, bei denen eine Vielfalt unterschiedlicher Anwendungen existieren, kann der Einsatz einer UDDI im Rahmen von EAI sehr sinnvoll sein.
Ausführlichere Informationen können aus [UDDI] bezogen werden. UDDI befindet sich bei OASIS in der Standardisierung und ist z.Zt. in der Version 3.0 verfügbar.

3 XML–Web–Services Szenario

Das in diesem Kapitel vorgestellte XML–Web–Services Szenario wird in den folgenden Kapiteln zur Betrachtung der Sicherheitsaspekte herangezogen. Es versteht sich außerdem als praxisorientiertes Beispiel einer funktionierenden losen Kopplung.

3.1 Ausgangssituation

Die für das Fallbeispiel frei erfundene Firma EasyBuy, beschließt ihre Produkte zukünftig auch über das Internet zu vermarkten. Um eine möglichst zukunftssichere Investition zu tätigen, setzt sie bei der Umsetzung auf die Web–Services Technologie. Dem Kunden soll es so ermöglicht werden, gezielt nach Produkten von EasyBuy zu suchen und diese ggf. zu bestellen.

3.2 Standards

Standards spielen nicht nur bei der Web–Services Architektur (XML, SOAP, UDDI und WSDL) eine große Rolle. Um der Idee der losen Kopplung gerecht zu werden ist es ebenfalls nötig, sich auf Web-Services Schnittstellen zu einigen. EasyBuy beschließt daher auf die (**fiktiven**) Standards ebCatalog und ebOrder zurückzugreifen. Diese vereinheitlichen Suchanfragen, Ergebnismengen (ebCatalog) und Bestellungen (ebOrder), und stellen einen de–facto–Standard auf dem Gebiet der Produktvermarktung dar. Die beiden Standards werden bereits von vielen Marktplätzen und Anwendungen genutzt, so dass EasyBuy mit einer hohen Zahl potentieller Neukunden rechnen darf.

3.2.1 ebCatalog

ebCatalog bietet Unternehmen eine Schnittstelle, um Suchanfragen an Produktdatenbanken entgegenzunehmen. Der Web–Service wird von einer WSDL Datei beschrieben und muss (so sagt es der Standard) in der Lage sein, SOAP Nachrichten über http zu verarbeiten. Die Nachrichten sind, zusätzlich zur WSDL Beschreibung, als XML Schema beschrieben, was eine Überprüfung (Parser) des Nachrichtenformats erleichtert. Als Namensraum für ebCatalog

basierende Nachrichten wurde „http://example.org/2003/ebCatalog“ festgelegt.

Abbildung 11 und 12 zeigen exemplarisch den Nachrichtenaufbau gemäß dem ebCatalog Standard.

```
<?xml version=„1.0 “?>
<env:Envelope xmlns:env=„http://www.w3.org/2001/09/soap-envelope“>
   <env:Body>
      <r:getProduktListe xmlns:r=„ http://example.org/2003/ebCatalog “>
         <r:SuchKriterium>Handy</r:SuchKriterium>
      </r:getProduktListe>
   </env:Body>
</env:Envelope>
```

Abbildung 11: ebCatalog Produktanfrage

```
<?xml version=„1.0 “?>
<env:Envelope xmlns:env=„http://www.w3.org/2001/09/soap-envelope“>
   <env:Body>
      <r:getProduktListeResponse xmlns:r=„http://example.org/2003/ebCatalog“>
         <r:Produkt>
            <Name>Siemens C25</Name> <Preis>25 EUR</Preis>
            <ArtikelNummer>12345</ArtikelNummer>
         </r:Produkt>
         <r:Produkt>
            <Name>Nokia 6150</Name> <Preis>10 EUR</Preis>
            <ArtikelNummer>54321</ArtikelNummer>
      </r:Produkt>
      </r:getProduktListResponse>
      </env:Body>
</env:Envelope>
```

Abbildung 12: ebCatalog Ergebnismenge

3.2.2 ebOrder

ebOrder beschreibt eine Schnittstelle zur elektronischen Bestellannahme. Ähnlich dem ebCatalog Standard, gibt es eine WSDL Schnittstellenbeschreibung, eine XML Schema Datei, sowie die Reglementierung, SOAP über http Anfragen verarbeiten zu können. Als Namensraum für den Standard wurde „http://example.org/2003/ebOrder“ festgelegt. Das Nachrichtenformat ist bei ebOrder den Bedürfnissen von Bestellungen angepasst. Für eine Bestellanfrage werden so Daten wie z.B. Adressinformationen, Kreditkartennummer, uvm. spezifiziert. Die Antwort enthält mindestens Auskunft über den Erfolg oder das Misslingen einer Bestellung. Auf ein konkretes Beispiel, wie es für ebCatalog in Abbildung 11 und 12 vorgestellt wurde, wird hier verzichtet.

3.3 Das Szenario

EasyBuy beschließt, die beiden Standards ebCatalog und ebOrder in das bestehende System zu integrieren. Der Einfachheit halber finden Zahlungen in diesem Szenario nur über Kreditkarten statt. EasyBuy beauftragt zum Geldeinzug einen Payment Provider, der die zum hierfür benötigten Informationen ebenfalls über den Aufruf eines Web–Services erhält. Der Einsatz der Standards und das Publizieren der Dienste in einer UDDI ermöglichen das automatisierte Auffinden und Nutzen der Dienste. Dies kann z.B. durch die (**fiktive**) Software CheaperProducts geschehen. CheaperProducts durchsucht dazu UDDI Verzeichnisse nach Diensten, die ebCatalog und ebOrder implementiert haben. Produktsuchanfragen des Benutzers werden an alle in Frage kommenden Dienste geschickt, und die zurückgelieferten Ergebnismengen werden dann z.B. nach Kosten- oder Lieferaspekten aufgelistet. Unter anderem bekommt der Benutzer jetzt auch das Produkt von EasyBuy aufgelistet, was bei einer normalen Produktrecherche im Internet eher unwahrscheinlich gewesen wäre. Hat sich der Benutzer für ein Produkt entschieden, erteilt er bequem per Software den Kaufauftrag. Die von ebOrder benötigen Daten (Lieferanschrift, usw.) sind bereits bei der Installation der Software erfasst und müssen daher nicht für jede Bestellung erneut angegeben werden. Von der Möglichkeit, auch die Kreditkartendaten in dem sogenannten Benut-

zerprofil der Software speichern zu lassen, sollte man aus Sicherheitsgründen absehen.

Der Ablauf einer Bestellung in dem beschriebenen Szenario könnte etwa wie in Abbildung 13 dargestellt aussehen.

1. EasyBuy registriert seinen Web-Service in einer UDDI (einmalig).
2. Die CheaperProducts Software sucht in der UDDI nach ebCatalog und ebOrder unterstützenden Web–Services.
3. Die UDDI liefert als Ergebnis der Suchanfrage die gefundenen Web–Services an die CheaperProducts Software (CPS).
4. Produktsuchanfragen des Benutzers werden von der CPS an alle gefundenen Web–Services geschickt (u.a. dem von EasyBuy).
5. Ergebnisse der Anfragen werden von der CPS empfangen und dem Benutzer, nach bestimmten Sortierkriterien aufgelistet, mitgeteilt.
6. Der Benutzer gibt dem Produkt von EasyBuy den Zuschlag und die Software schickt die Bestellung an den Web–Service des Unternehmens. Fehlen im Benutzerprofil relevante Daten (z.B. Kreditkartennummer), wird der Benutzer vorher von der CPS zur Vervollständigung dieser aufgefordert.
7. Nachdem die Zahlungsfähigkeit des Kunden durch den Payment Provider überprüft und die Anfrage durch die Warenwirtschaft bearbeitet ist, wird
   (a) die erfolgreiche Bestellung der CPS (dem Kunden) unverzüglich bestätigt.
   (b) der Abbruch des Bestellvorgangs und die Gründe hierfür der CPS (dem Kunden) mitgeteilt.

Durch die Web–Services Schnittstelle findet auf einfache Weise eine Kapselung der hausintern verwendeten Technologie statt. Zur Realisierung kann

daher auf bereits bestehende Plattformen, Programmiersprachen und hausintern verwendete Protokolle zurückgegriffen werden. Bindend ist nur die nach aussen präsentierte Schnittstelle (z.B. ebOrder).

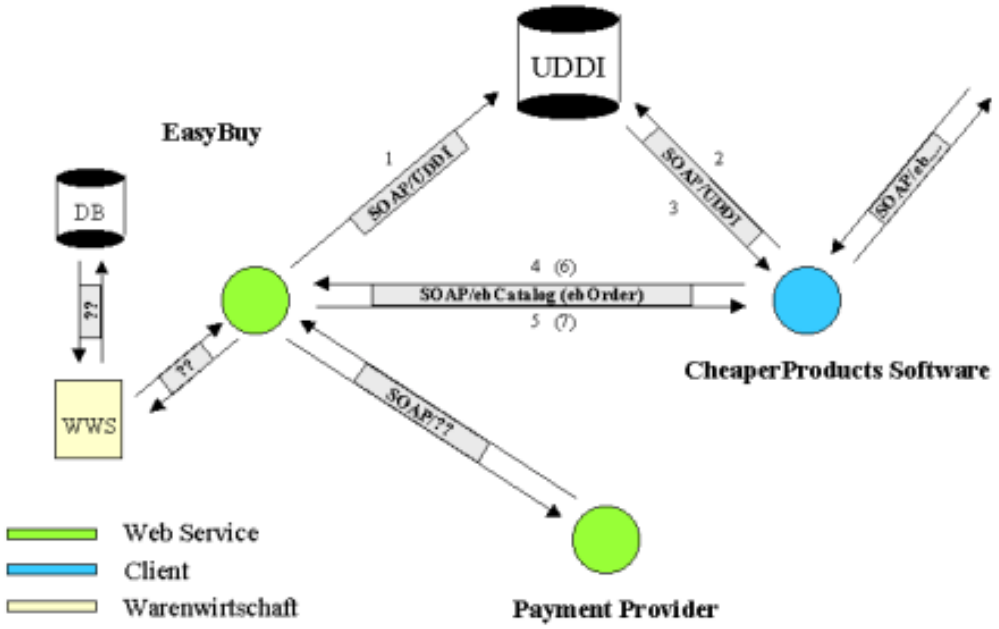

Abbildung 13: XML–Web–Services Szenario

3.4 Sicherheitsbedarf

Die Öffnung eines Systems gegenüber unsicheren Netzen, wie es z.B. mit der Einführung von Web–Services in dem gegebenen Szenario geschehen ist, birgt spezifische Sicherheitsrisiken. Dass diese Sicherheitsrisiken bei dem Entwurf neuer Technologien aus den unterschiedlichsten Gründen vernachlässigt werden, ist auch der maßgebliche Grund für die noch spärliche Verbreitung von Web–Services. Obwohl es mittlerweile verschiedene Ansätze zur Sicherung von Web–Services gibt, hat EasyBuy sich - unwissend - auf die bereits existente Firewall verlassen. Nach den ersten unangenehmen Erfahrungen mit Angreifern, sieht sich EasyBuy gezwungen die neuen Dienste vorerst einzustellen.

Die Kapitel 4 und 5 beleuchten sowohl die Sicherheitskonzepte, als auch die speziellen Risiken, die durch den Einsatz von Web–Services entstehen. In Kapitel 7 werden die gewonnenen Erkenntnisse dann eingesetzt, um die benötigeten Sicherheitsmerkmale in das Szenario zu integrieren.

4 Sicherheitskonzepte und Stand der Normung

Bei dem Versuch, die durch den Einsatz von Web–Services entstehenden Sicherheitsrisiken einzudämmen, sind die Bemühungen der offenen Standardisierungsgremien von entscheidender Bedeutung. Nach [Fontana 02] arbeiten IETF, OASIS und W3C an insgesamt 13 Sicherheitskonzepten, die beim Einsatz von Web-Services Verwendung finden können (siehe Abbildung 14). Diese Arbeit konzentriert sich auf die Betrachtung der Kerntechnologien, zu denen XML Signature, XML Encryption, WS-Security und XKMS gehören.

Standard (proposed or final)	Standards body / Status	Description
SAML Security Assertion Markup Language	OASIS/Final vote	Exchanges user and machine authentication and authorization information.
XACML Extensible Access Control Markup Language	OASIS/Committee review	Expresses policies for information access.
SPML Service Provisioning Markup Language	OASIS/In committee	Defines how to exchange user, resource and service provisioning information.
WS-Security	OASIS/Forming technical committee	Extends Simple Object Access Protocol with XML security protocols.
XrML Extensible Rights Management Language	OASIS/Gathering requirements	Manages copyrights of digital content.
XCBF XML Common Biometric Format	OASIS/Defining scope of work	Defines XML codings for Common Biometric Exchange File Format.
XML Digital Signature	W3C/Completed	Provides integrity, signature assurance and non-repudiation.
XML Encryption	W3C/Final vote in committee	Encrypts and decrypts digital content.
XKMS XML Key Management Specification	W3C/Working draft	Provides a method for obtaining cryptographic keys.
Transport Layer Security/ Secure Sockets Layer	IETF/RFC 2246	Secures Internet traffic between two points.
SASL Simple Authentication and Security Layer	IETF/RFC 2222	Adds authentication to connection-based protocols.
Kerberos	IETF/RFC 1510	Provides tickets for authenticating users.
BEEP Blocks Extensible Exchange Protocol	IETF/RFC 3080	Helps establish quality of service over the Internet.

Abbildung 14: [Fontana 02] Auflistung Sicherheitsprotokolle

4.1 Standardisierungsorganisationen

Offene Standards spielen bei der Evolution des Internet eine entscheidende Rolle. Bei Web–Services, die als ein Evolutionsschritt des Internet angesehen werden dürfen, tragen diese Standards zu breiter Unterstützung bei. Die Gründe hierfür lassen sich mit der Struktur und Finanzierung der Standardisierungsorganisationen erklären. Aus der Tatsache, dass bei offenen Standards keine Zahlungen von Lizenznehmern erfolgen, läßt sich ableiten, dass die Zahlungen für die Verwaltung der Organisatinen von anderer Stelle getragen werden. Diese Finanzierung erfolgt durch Unternehmen, die als Mitglieder einer bestimmten Organisation ebenfalls maßgeblich an der Entwicklung der Standards beteiligt sind. Die Mitarbeit von Individuen ist dabei allerdings nicht ausgeschlossen, beim IETF sogar ausdrücklich erwünscht. Ist die Notwendigkeit zur Erstellung eines neuen Standards gegeben, wird für diesen eine Arbeitsgruppe ins Leben gerufen, die aus interessierten Unternehmen und ggf. Individuen besteht. Durch diese kooperative Vorgehensweise wird in der Regel eine schnelle und hohe Akzeptanz der erarbeiteten Standards erreicht.

In diesem Abschnitt werden die direkt mit Web–Services in Bezug stehenden Standardisierungsorganisationen, bei denen es sich um W3C, OASIS und IETF handelt, kurz vorgestellt. Alle in Abbildung 14 bereits aufgelisteten Sicherheitskonzepte sind hier übersichtlich den jeweils verantwortlichen Organisationen zugeordnet.

4.1.1 W3C

Das **W**orld **W**ide **W**eb **C**onsortium (W3C) wurde im Oktober 1994 gegründet und zielt darauf ab, insbesondere Protokolle zu standardisieren, die die Evoulution des Internet und dessen Interoperabilität sichern. So stammen, um nur die bekanntesten zu nennen, HTTP, HTML, und XML vom W3C.

Sicherheitskonzepte XML Signature (IETF), XML Encryption und XKMS.

4.1.2 OASIS

Die **O**rganisation for the **A**dvancement of **S**tructured **I**nformation **S**tandards wurde 1993 unter dem Namen SGML Open von einer Reihe Hersteller und Benutzer gegründet um Richtlinien für die Zusammenarbeit SGML unterstützender Produkte zu schaffen. 1998 änderte die SGML Open ihren Namen in OASIS, um den erweiterten Arbeitsbereich zu verdeutlichen. Die seither entworfenen Standards basieren größtenteils auf XML.

Sicherheitskonzepte SAML, XACML, SPML, WS–Security, XrML und XCBF.

4.1.3 IETF

Die **I**nternet **E**ngineering **T**ask **F**orce besteht seit 1992. Sie sollte eine internationale Gemeinschaft aus Netzwerkdesignern, Anwendern, Herstellern und Marktforschern aufbauen, um die Evolution der Architektur und Arbeitsweise des Internet voranzutreiben.

Sicherheitskonzepte XML Signature (W3C), SSL/TLS, SASL, Kerberos und BEEP.

4.2 Verbindungsorientierte Sicherheit

Verbindungsorientierte Sicherheit schützt, wie es der Name bereits sagt, Verbindungen. Sicherheitsmerkmale, wie Authentifikation, Vertraulichkeit oder Integrität werden hier an die Verbindung und nicht an die eigentlichen Daten gebunden. Verbindungsorientierte Sicherheit erreicht man im Internet üblicherweise durch den Einsatz von HTTPS (SSL / TLS) oder IPSec. Mit dem Einsatz dieser in Web–Services Szenarien müssten jedoch auch einige Schwachstellen in Kauf genommen werden:

1. Die Sicherheit der Daten kann zwar während der Übertragung, nicht aber nach der Ankunft dieser am Bestimmungsort gewährleistet werden.

2. Wird eine Nachricht von einem Quellknoten A über einen Vermittlerknoten B zu einem Zielknoten C geroutet, wäre Knoten B in der Lage unbemerkt alle Daten zu lesen oder zu modifizieren.

3. Können nur bei der Verwendung verbindungsorientierter Transportprotokolle eingesetzt werden.

Während der erste Punkt allgemeiner Natur ist, stehen die letzteren der Architektur und dem Grundgedanken von Web–Services entgegen. Diese propagieren nämlich u.a. den Einsatz von

- Nachrichtenpfaden.
- verschiedenartigen, auch verbindungslosen Transportprotokollen.

Von einer tiefergehenden Betrachtung verbindungsorienterter Sicherheitsprotokolle wird daher in dieser Arbeit abgesehen. Aufgrund der hohen Verbreitung und der relativ einfachen Einbindung derer in bestehende Architekturen ist jedoch davon auszugehen, dass ihr Einsatz auch in Web–Services Szenarien Verwendung finden wird. In einigen Fällen mag dies sogar sinnvoll erscheinen.

Eine gute Beschreibung von Aufbau und Einsatzmöglichkeiten der im Internet am weitest verbreiteten verbindungsorientierten Sicherheitsprotokolle SSL / TLS liefert [Rescorla 00].

4.3 Datengebundene Sicherheit

Im Gegensatz zu der verbindungsorientierten Sicherheit sorgt die datengebundene Sicherheit für eine unabhängig vom Aufenthaltsort existierende Sicherheit der Daten. Um den besonderen Eigenschaften von XML Dokumenten gerecht zu werden, wurden zwei Standards entworfen, die das Signieren (XML Signature) oder Verschlüsseln (XML Encryption) von XML Dokumentteilen (Elementen) spezifizieren. Den einheitlichen Einsatz dieser Technologien in einer Web–Services Umgebung beschreibt die WS-Security Spezifikation.

4.3.1 Exclusive XML Canonicalization

Organisation:	W3C und IETF
Normierung:	Recommendation 18.07.2002
Spezifikation:	http://www.w3.org/TR/xml-exc-c14n/
Namensraum:	

Nach [XML-ENC] ist XML Canonicalization „[...] a method of consistently serializing XML into an octet stream as is necessary prior to encrypting XML.“. Sowohl XML Signature als auch XML Encryption greifen zu diesem Zweck auf XML Canonicalisation zurück.

Die Notwendigkeit für den Einsatz von XML Canonicalisation besteht darin, dass zwei vom Inhalt identische XML Dokumente aufgrund syntaktischer Unterschiede nicht immer die gleiche Bytefolge aufweisen müssen. So würde eine digitale Signatur bereits dann brechen, wenn zur besseren Lesbarkeit des Dokumentes ein Element nachträglich eingerückt würde, obwohl dies weder die Beeinträchtigung des signiereten Inhalts noch die der Gültigkeit des Dokumentes zur Folge hätte. Ursache dafür ist, dass Signaturen nicht über Inhalte, sondern über konkrete Bytefolgen von XML Dokumenten oder Teilen dieser gebildet werden.

XML Canonicalization beschreibt einen Mechanismus, der zur syntaktischen Vereinheitlichung von XML Dokumenten verwendet wird. Es schafft damit die Möglichkeit, Inhalte von XML Dokumenten zu signieren oder zu verschlüsseln, ohne das Fehler wegen syntaktischer Spielräume auftreten.

4.3.2 XML Signature

Organisation:	W3C und IETF
Normierung:	Recommendation 12.02.2002
Spezifikation:	http://www.w3.org/TR/xmldsig-core
Namensraum:	xmlns:ds='http://www.w3.org/2000/09/xmldsig'

XML Signature führt keine neuen Signaturalgorithmen ein, sondern spezifiziert die Art und Weise in der

- XML Dokumente
- Teile dieser
- externe Ressourcen

mit bestehenden Algorithmen signiert werden. Die Spezifikation beschreibt einige Regeln zur Vorgehensweise bei der Signierung und eine XML Struktur zur Unterbringung der Sicherheitsmetadaten. Die Signatur eines XML Elementes kann dabei die in Abbildung 15 dargestellten Varianten annehmen. Besonderer Erklärung bedarf hier nur die XML Signature Variante 'enve-

Abbildung 15: Erscheinungsformen XML Signature

loped', bei der sich die Signatur innerhalb des zu signierenden Elementes befindet. Folglich müsste sich die Signatur selber signieren, was unmöglich ist. [XML-SIG] schreibt in diesem Fall vor das gesamte <Signature> Element, in dem sich alle Informationen über die Signatur befinden, sowohl bei der Signierung als auch bei der Validierung unberücksicht zu lassen.

Die Spezifikation ermöglicht das Signieren mehrerer Ressourcen innerhalb einer XML Signature. Erreicht wird dies durch die Möglichkeit innerhalb einer Signatur auf mehrere Ressourcen zu verweisen. Die Referenzierung findet dabei über den URI Mechanismus statt, der entweder auf das Identifikationsmerkmal (ID) des zu signierenden Elementes, oder auf eine externe Ressource verweist.

Bevor näher auf die Funktionsweise einer XML Signature eingegangen wird, stellt Abbildung 16 den Elementbaum von XML Signature vor. Um

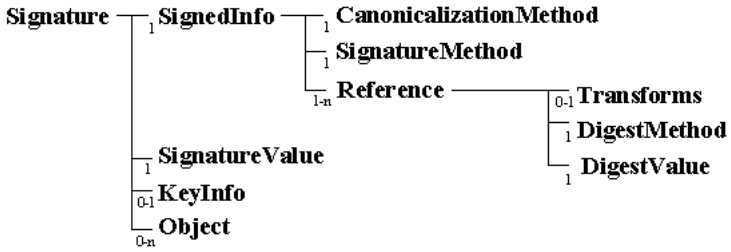

Abbildung 16: XML Signature Elementbaum

eine XML Signature konforme Signatur zu erhalten werden nicht die Referenzierten Ressourcen signiert, sondern die digitalen Fingerabdrücke dieser. Um diese zu erhalten, kann z.B. auf den SHA oder beliebige andere Algorithmen zurückgegriffen werden. Die eigentliche Signatur ist dann über das <SignedInfo> Element zu bilden, das

- den verwendeten Algorithmus zur Canonicalization (<CanonicalizationMethod>)
- den verwendeten Signaturalgorithmus (<SignatureMethod>)
- die Referenzen (URI) der zu signierenden Ressourcen (<Reference>) und der zugehörigen digitalen Fingerabdrücke (<DigestValue>)
- den zur Erstellung der Fingerabdrücke verwendeten Algorithmus (<DigestMethod>)
- ggf. vor der Validierung auszuführende Transformationen (<Transforms>)

beinhaltet. Abbildung 17 zeigt den Aufbau einer XML Signature. Das <KeyInfo> Element ist, wie aus dem Elementbaum (siehe Abbildung 16) ersichtlich, optional und liefert Schlüsselinformationen zur Validierung der Signatur.

Erzielte Sicherheit *Integrität.* Beim Einsatz zusätzlicher Mechanismen (Zeitstempel, Sequenznummern) kann auch die *Verbindlichkeit* von signierten Nachrichten und die *Authentifikation* von Benutzern erreicht werden.

```
<Signature Id='SigID' xmlns=„http://www.w3.org/2000/09/xmldsig# “>
   <SignedInfo>
      <CanonicalizationMethod
      Algorithm='http://www.w3.org/TR/2001/REC-xml-c14n-20010315'/>
      <SignatureMethod
      Algorithm=' http://www.w3.org/2000/09/xmldsig#dsa-sha1' />
      <Reference URI='http://www.michael-hildebrandt.de/' >
         <Transforms>
            <Transform
               Algorithm='http://www.w3.org/TR/2001/REC-xml
                -c14n-20010315' />
         </Transforms>
         <DigestMethod Algorithm='http://www.w3.org/2000/09/xmldsig#sha1'/>
         <DigestValue>k3kLj5Spdka3PkkKLSSkssie</DigestValue>
      </Reference>
   </SignedInfo>
      <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
      <KeyInfo>
      <KeyValue>
         <DSAKeyValue>
            <P>...</P> <Q>...</Q> <G>...</G> <Y>...</Y>
         </DSAKeyValue>
      </KeyValue>
   </KeyInfo>
</Signature>
```

Abbildung 17: Beispiel XML Signature

Sicherheitskritische Betrachtung Bei der Signatur von Verträgen o.ä. sollten, um ein rechtliches Fundament zu schaffen, nicht nur das XML Dokument, sondern ggf. auch alle für die Darstellung wichtigen Ressourcen signiert werden. Zu solchen Ressourcen zählen z.B. Stylesheets oder Transformationen. Nur so kann bei Streitigkeiten der Beweis erbracht werden, dass ein bestimmtes Dokument, bei dem sowohl Daten als auch Layout eine Rolle spielen, unterzeichnet worden ist.
Außerdem ist sowohl bei der Auswahl des zur Signierung verwendeten Algorithmus als auch bei dem Einsatz der jeweiligen Implementierung des Algo-

rithmus Vorsicht geboten. Als Beispiel für eine fehlerhafte Implementierung von Sicherheitsalgorithmen sei die Verschlüsselung vom Netscape Browser in der Version 1.1 genannt. Der 128–Bit–Schlüssel hatte durch einen fehlerhaften Zufallsgenerator nur eine maximale Entropie von etwa 20 Bit. Der Algorithmus war also nicht besser, als hätte er einen 20–Bit–Schlüssel gehabt [Schneier 00]. In [XML-SIG] wird ausdrücklich vor dem Einsatz des MD5 Algorithmus zum Erstellen digitaler Fingerabdrücke gewarnt, da dieser Algorithmus als schwach anzusehen ist.

4.3.3 XML Encryption

Organisation:	W3C
Normierung:	Recommendation 10.12.2002
Spezifikation:	http://www.w3.org/TR/xmlenc-core
Namensraum:	xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'

XML Encryption stellt einen Mechanismus zur Verschlüsselung von

- ganzen XML Dokumenten
- einzelnen Elementen
- Elementinhalten
- externen Ressourcen

zur Verfügung. Die Spezifikation enthält Regeln zur Vorgehensweise bei der Verschlüsselung, und beschreibt die in Abbildung 18 dargestellte XML Struktur als Container für die Sicherheitsmetadaten. XML Encryption legt sich nicht auf die Benutzung bestimmter Verschlüsselungsalgorithmen fest, schreibt jedoch einige, z.B. AES 128 [AES], zwingend zur Implementierung vor. Zum besseren Verständnis führt Abbildung 19 ein Beispiel für den Einsatz von XML Encryption ein. Es erfüllt lediglich die von der Spezifikation vorgeschriebene Minimalstruktur (siehe Elementbaum), zeigt jedoch bereits die wesentlichen Vorteile von XML Encryption. Der Standard ermöglicht die

Abbildung 18: XML Encryption Elementbaum

partielle Verschlüsselung von XML Dokumenten. Die daraus resultierenden Vorteile sind im Wesentlichen:

- Die Möglichkeit gezielt sensible Daten eines Dokumentes zu verschlüsseln (das Element <Kreditkarte> im vorgestellten Beispiel).
- Die Verschlüsselung mehrerer Dokumentteile unter zu Hilfenahme verschiedener Schlüssel. Dies ermöglicht die sichere Verwendung von Nachrichtenpfaden in Web–Services Szenarien, weil dadurch sichergestellt werden kann, dass jeder Knoten nur die für Ihn bestimmten Teile einer Nachricht lesen kann.

Komplexere Beispiele zu XML Encryption können bei Interesse in [Eastlake 02] gefunden werden.

Erzielte Sicherheit *Vertraulichkeit.*

Sicherheitskritische Betrachtung Aufgrund der Tatsache, dass bei Web–Services das Nachrichtenformat durch die Veröffentlichung einer WSDL Beschreibung bekannt ist, ist bei der Verschlüsselung eines Elementes auch ein gewisser Teil Klartext (Elementname) bekannt. Die Kenntnis von Klartextanteilen kann dabei nach [Schneier 96] die Schwächung der Verschlüsselung zur Folge haben, weil Known–Plaintext Angriffe ermöglicht werden.
Bei der Verwendung von Namensräumen oder Entitäten, sollte Exclusive XML Canonicalization auf die entsprechenden Dokumentteile angewendet werden. Dies stellt sicher, dass sowohl die Präfixe der Namensräume als auch die Entitäten expandiert werden. Geschieht dies nicht, kann durch eine

Klartext

```
<?xml version='1.0' ?>
<ZahlungsInformation xmlns='http://example.org/Zahlung' >
   <Name>Michael Hildebrandt</Name>
   <Kreditkarte>
      <Nummer>7082 1512 0121 6853</Nummer>
      <Gueltig>04/04</Gueltig>
   </Kreditkarte>
</ZahlungsInformation>
```

XML Encrypted

```
<?xml version='1.0'?>
<ZahlungsInformation xmlns='http://example.org/Zahlung'>
   <Name>Michael Hildebrandt</Name>
   <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
     xmlns='http://www.w3.org/2001/04/xmlenc#'>
      <CipherData>
         <CipherValue>B56X33Lloe3</CipherValue>
      </CipherData>
   </EncryptedData>
</ZahlungsInformation>
```

Abbildung 19: Beispiel XML Encryption

nachträgliche Veränderung der Namensraumdeklaration oder der Entitäten die Aussage des Dokuments verfälscht werden.
Auf den Einsatz starker und fehlerfrei implementierter Algorithmen ist zu achten.

4.3.4 WS-Security

Organisation:	OASIS
Normierung:	Draft 9 26.01.2003
Spezifikation:	http://www.oasis-open.org/committees/wss/documents/ WSS-Core-09-0126.pdf
Namensraum:	xmlns:wsse='http://schemas.xmlsoap.org/ws/2002/07/secext' xmlns:wsu='http://schemas.xmlsoap.org/ws/2002/07/utility'

Die **W**eb–**S**ervices–**S**ecurity Spezifikation definiert einen Standard zum sicheren Austausch SOAP basierter Nachrichten. Der Schwerpunkt der Spezi-

fikation liegt darauf, zu beschreiben, wie die bereits bestehenden Sicherheitsmechanismen XML Signature und XML Encryption in SOAP zu integrieren sind. [Seely 02] beschreibt den Mehrwert zu den bereits existierenden Spezifikationen als „[...] a framework to embed these mechanisms into a SOAP message“. Der einheitliche Einsatz der Sicherheitsmechanismen vereinfacht dabei die automatische Verarbeitung verschlüsselter oder signierter Nachrichten.

Um dieses Ziel zu erreichen, definiert die Spezifikation ein <Security> Element, das im Header einer SOAP Nachricht zu platzieren ist, und grundsätzlich jede sicherheitsrelevante Information aufnehmen soll. Die Vorgehensweise zur Beschreibung dieses Elementes ähnelt der Definition des Header und Body Bereichs in der SOAP Spezifikation. Dabei wird zunächst nur mit Worten beschrieben welche Funktion die entsprechenden Elemente zu erfüllen haben. Dadurch ergeben sich, im Gegensatz zu einem Korsett aus fest vorgegebenen Elementen, Spielräume für Erweiterungen. Wie bereits im Abriss des Standards ersichtlich, definiert dieser Standard zwei Namensräume. Davon wird einer für die Identifikation der Basiselemente verwendet, während sich der andere auf die Struktur für einen Zeitstempel (engl. timestamp) bezieht. Diese Aufteilung ermöglicht Anwendungen oder Spezifikationen den bereits spezifizierten Zeitstempel zu verwenden, ohne die komplette WS-Security Spezifikation implementieren zu müssen.

Der gewichtigste Teil der Spezifikation beschäftigt sich mit der Einführung von Sicherheitstokens. Solche Token sind als Informationseinheit zur Authentifikation zu verstehen. Die Spezifikation unterscheidet hier zwischen zwei Typen:

1. **UsernameToken**
   Authentifikation findet über Benutzername / Passwort Verfahren statt. Damit das Passwort nicht im Klartext übertragen werden muss, führt man hier einen Mechanismus zur Erstellung eines sicheren Passwort–Hash ein.

2. **BinarySecurityToken**
   Authentifikation findet über ein binäres Sicherheitstoken statt. Bei-

spielhaft erwähnt werden in diesem Zusammenhang X.509 Zertifikate oder Kerberos Tickets.

Mit der Einführung von Sicherheitstokens und den Regelungen für den Einsatz von XML Signature und XML Encryption liefert WS-Security ein Sicherheitsframework, das es Anwendungen ermöglicht grundlegende Sicherheit in SOAP Nachrichten einheitlich zu implementieren. Abbildung 20 zeigt die wichtigsten Charakteristika von WS-Security. Zu Begin des SOAP Headers ist ein Zeitstempel mit einem eindeutigen Identifikationswert (ID) zu setzen. Weil das Element signiert werden soll, ist diese ID in der Signatur des Beispiels als Referenz erforderlich. Im Anschluss daran wird das <Security> Element der Spezifikation eingeführt. Dieses Element beinhaltet in dem vorgegebenen Beispiel drei sicherheitsrelevante Teile.

1. Die Einführung eines X.509 Zertifikats (<BinarySecurityToken>).
2. Der Verweis auf verschlüsselte Ressourcen (<ReferenceList>).
3. Platzierung einer Signatur für den Zeitstempel (<Signature>)

Die in der Nachricht enthaltene Sicherheit konzentriert sich jetzt auf das <Security> Element. Eine Anwendung muss nur noch an dieser definierten Stelle nachsehen, um zu erfahren was für Sicherheitsmechanismen in Kraft gesetzt werden müssen, damit die korrekte Verarbeitung der Nachricht gewährleistet ist. Ebenfalls möglich ist die Verwendung mehrerer <Security> Elemente. Diese müssen dann über das soap:actor Attribut einen verantwortlichen Verarbeiter (URI) zugewiesen bekommen. Auf diese Weise kann man für jeden Zwischenknoten auf einem SOAP Nachrichtenpfad ein eigenes Sicherheitspaket definieren.

Erzielte Sicherheit *Vertraulichkeit*, *Integrität* und *Nicht-Abstreitbarkeit*.

Sicherheitskritische Betrachtung Sowohl [XML-ENC] als auch [WS-SEC] warnen vor einer möglichen Verwundbarkeit der Kryptographie bei der Verwendung von Signatur und Verschlüsselung über das gleiche Element, wenn

```
<?xml version='1.0' encoding='utf-8'?>
<soap:Envelope xmlns:soap=...>
   <soap:Header>
      <wsu:Timestamp wsu:id='Stamp-000001'>
         <wsu:Created wsu:Id='Id-time-crea-000001'>
            2002-08-22T00:26:15Z
         </wsu:Created>
      </wsu:Timestamp>
      <wsse:Security soap:mustUnderstand='1' >
         <wsse:BinarySecurityToken ValueType='wsse:X509v3'
          EncodingType='wsse:Base64Binary' wsu:Id='SecTok-000001'>
            MIIHdjCCB...
         </wsse:BinarySecurityToken>
         <xenc:ReferenceList>
            <xenc:DataReference URI='#EncryptedContent' />
         </xenc:ReferenceList>
         <ds:Signature xmlns:ds='...'>
            <ds:SignedInfo>
               <ds:Reference URI:'#Stamp-000001'>
                  ...
               </ds:Reference>
            </ds:SignedInfo>
            ...
            <ds:KeyInfo>
               <wsse:SecurityTokenReference>
                  <wsse:Reference URI='#SecTok-000001'>
               </wsse:SecurityTokenReference>
            </ds:KeyInfo>
         </ds:Signature>
      </wsse:Security>
   </soap:Header>
   <soap:Body>
      <xenc:EncryptedData Id='EncryptedContent'
         ...
      </xenc:EncryptedData>
   </soap:Body>
</soap:Envelope>
```

Abbildung 20: Beispiel WS-Security

die Signatur unverschlüsselt im Dokument verbleibt. Nach [WS-SEC] könnten damit „plain text guessing attacks“ ermöglicht werden.

4.4 Authentifikation, Autorisation und Schlüsselmanagement

Für die Implementierung der bereits vorgestellten Sicherheitsmechanismen ist die Frage des Schlüsselmanagements von entscheidender Bedeutung. Weil sich die Verwendung proprietärer Protokolle beim Zugriff auf eine PKI oft als nicht triviale Aufgabe herausstellt, wurde zur Vereinfachung der in Abschnitt 4.4.1 vorgestellte XKMS Standard entwickelt.

Eine weitere Herausforderung stellt die verteilte Umgebung von Web–Services an die Authentifikation und Autorisation von Benutzern. Dieses Problem wird von der SAML Spezifikation angesprochen, auf die in Abschnitt 4.4.2 eingegangen wird.

4.4.1 XKMS

Organisation:	W3C
Normierung:	Working Draft 18.03.2002
Spezifikation:	http://www.w3.org/TR/2002/WD-xkms2-20020318/
Namensraum:	xmlns:xkms='http://www.w3.org/2002/03/xkms'

Die **X**ML **K**ey **M**anagement **S**pecification beschreibt nach [XKMS] „[...]protocols for distributing and registering public keys, suitable for use in conjunction with the proposed standard for XML Signature [XML-SIG] developed by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) and an anticipated companion standard for XML encryption“. Dabei werden die komplizierten proprietären Schnittstellen einer PKI durch die Verwendung XML basierter Nachrichten über Standardprotokolle ersetzt. Der XKMS Service stellt, wie in Abbildung 21 dargestellt, eine Art Middelware zwischen Anwendung und PKI dar. Ein XKMS Dienst ist

wegen der XML basierten Kommunikation über SOAP als Web–Service zu betrachten. Die wichtigsten Argumente für den Einsatz von XKMS sind:

- Einfache Implementierung. Zur Einbindung von Sicherheit reichen XML Kenntnisse.
- Plattform-, Hersteller- und Transportprotokollunabhängig.
- Zukunftssicher, da Erweiterungen die Abwärtskompatibilität nicht gefährden.
- Geeignet für die Verwendung in speicherarmen Mobilgeräten.

Innerhalb der XKMS Spezifikation werden die Aufgaben den Substandards X-KISS und X-KRSS zugewiesen.

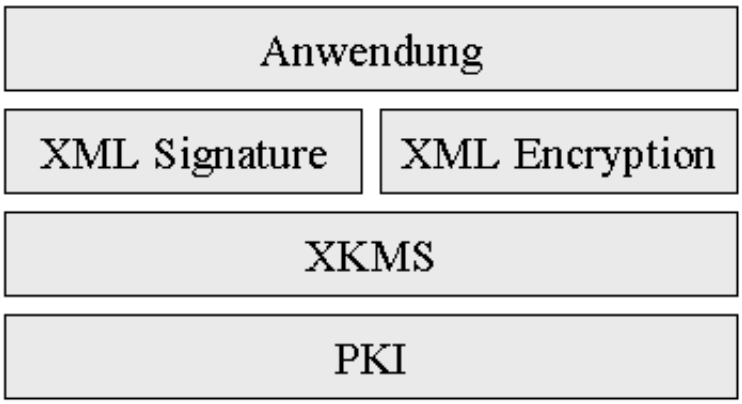

Abbildung 21: Der XKMS Technologiestack

X-KISS (**X**ML - **K**ey **I**nformation **S**ervice **S**pecification) beschreibt einen Web–Service zur Suche und Überprüfung von Schlüsselmaterial. Der Dienst ist in ein Drei–Schichten–Modell aufgeteilt. Ein X-KISS Service kann nach diesem Modell die Spezifkation bis zu einer bestimmten Schicht (engl. tier) implementieren und den Rest außen vor lassen. Solche Modelle lassen aufgrund der geringeren Entwicklungszeiten und -kosten, bei der Implementierung unterer Schichten auf eine schnelle Verbreitung hoffen. Ein weiterer

Vorteil ist, dass nur der tatsächliche Bedarf an Funktionalität implementiert werden muss. Die Aussage über die Implementierung einer bestimmten Schicht beinhaltet, dass die darunter liegenden Schichten auch unterstützt werden. Die drei Schichten des X-KISS Protokolls setzten sich wie folgt zusammen:

1. Tier 0: <ds:RetrievalMethod> Processing
2. Tier 1: Locate Service
3. Tier 2: Validate Service

(Tier 0) beschreibt die Unterstützung des aus der XML Signature Spezifikation stammenden <RetrievalMethod> Elements. Innerhalb eines <KeyInfo> Elements kann so z.B. auf Zertifikate verwiesen werden. Bei mehrfacher Verwendung eines Zertifikates innerhalb eines XML Dokumentes erspart dies die redundante Einbindung dessen. Ob die Beschreibung eines solchen Verhaltens eine eigene Schicht rechtfertigt, darf bezweifelt werden. „An XKMS service does not actually provide this level, so why the XKMS document lists this ability isn't completely clear." [Eastlake 02].
(Tier 1) beschreibt wie Schlüsselsuchanfragen entgegenzunehmen und zu bearbeiten sind. Als Basiselement wird sowohl für die Suchanfrage, als auch für die Antwort auf diese, auf das <KeyInfo> Element aus der XML Signature Spezifikation zurückgegriffen. Wird ein Schlüssel gefunden, beinhaltet das zurückgegebene <KeyInfo> Element diesen. Der Service ist außerdem in der Lage, digitale Zertifikate zu parsen und die extrahierten Informationen in einem <KeyInfo> Element zurückzugeben.
(Tier 2) überprüft Schlüssel auf Gültigkeit. So kann z.B. überprüft werden ob ein Schlüssel zu dem vermeintlich zugehörigen Namen passt oder ob Zertifikate und Zertifikatketten Gültigkeit besitzen. Abbildung 22 stellt eine auf Tier 2 stattfindende Anfrage dar.

X-KRSS (**X**ML - **K**ey **R**egistration **S**ervice **S**pecification) beschreibt einen Web–Service zur Registrierung und Verwaltung von öffentlichen Schlüsseln,

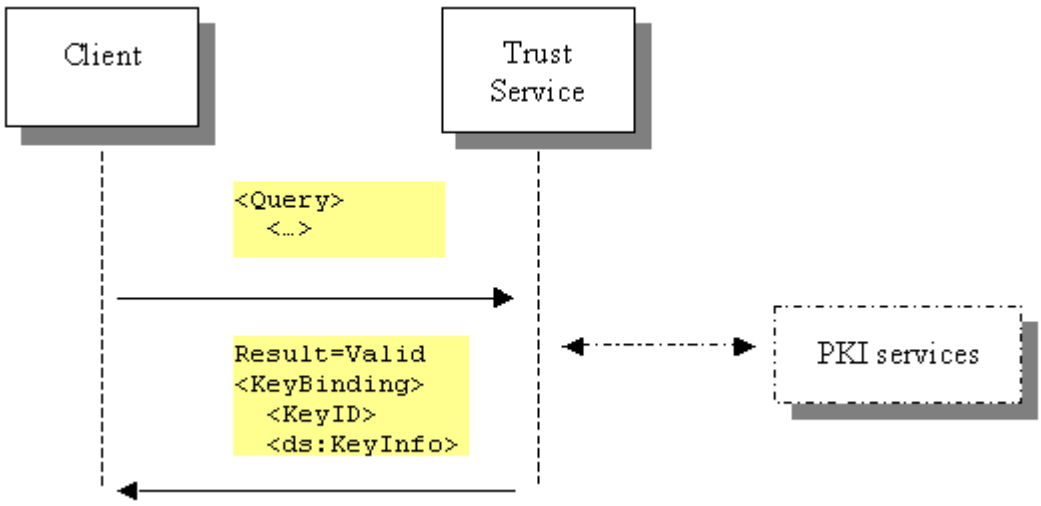

Abbildung 22: [XKMS] Tier 2 Key Validation Service

die dann z.B. von einem X-KISS Dienst abgerufen werden können. Ein X-KRSS Dienst unterstützt folgende Funktionen:

- Generierung eines Schlüsselpaares durch den Dienst.
- Publizieren eines eigenständig generierten öffentlichen Schlüssels.
- Wiederherstellung eines vorher registrierten privaten Schlüssels (key recovery).
- Schlüssel für ungültig erklären (key revokation).

Erzielte Sicherheit Dieser Dienst adressiert die Vereinfachung von Schlüsselmanagementfunktionen in einem Web–Services Szenario.

Sicherheitskritische Betrachtung In den „Security Considerations“ aus [XKMS] wird auf Sicherheitsrisiken durch

- Replay Attacken
- DoS Attacken

- Key Recovery Policy
- Security of Limited Use Shared Secret

hingewiesen. Die Risiken betreffen alle die Implementierung des XKMS Dienstes und fallen daher aus dem Betrachtungsrahmen dieser Arbeit, der sich auf die Sicherheit des Web–Services konzentriert.

4.4.2 SAML

Organisation:	OASIS
Normierung:	Committee Specification 31.05.2002
Spezifikation:	http://www.oasis-open.org/committees/ documents.php?wg_abbrev=security
Namensraum:	xmlns:'urn:oasis:names:tc:SAML:1.0:assertion' xmlns:'urn:oasis:names:tc:SAML:1.0:protocol'

Die **S**ecurity **A**ssertion **M**arkup **L**anguage führt „single sign on“ auf der Basis von Web–Services Technologien ein. Mit SAML wird ein Verfahren eingeführt, um die Zentralisierung von Authentifikation und Autorisation in verteilten Web–Services Umgebungen zu ermöglichen. Die unpraktikable Lösung bisher existenter, dezentraler Authentifikations- und Autorisationsmechanismen, bei der sich ein Benutzer zunächst an jedem verwendeten Service registrieren muss, entfällt damit gänzlich. Anstelle dessen treten, wie Abbildung 23 veranschaulicht, zentrale Authentifikationsstellen, die nach einmaliger Anmeldung zur Benutzung diverser Web–Services autorisieren. SAML fürht zu diesem Zweck Assertions ein. Es handelt sich dabei um XML basierte Berechtigungsscheine, die einen Client zur Nutzung verschiedener Web–Services autorisiert. Es ist wichtig zu verstehen, dass sich die beteiligten Instanzen nicht fremd sind. Ein Vertrauen auf der Basis von Verträgen oder anderen Kriterien herrscht zwischen den Partien. Nur so kann zum einen die SAML Instanz wissen wozu registrierte Benutzer berechtigt sind und zum anderen der Web–Service darauf vertrauen dem Besitzer einer Assertion berechtigten Zugriff gewährt zu haben.

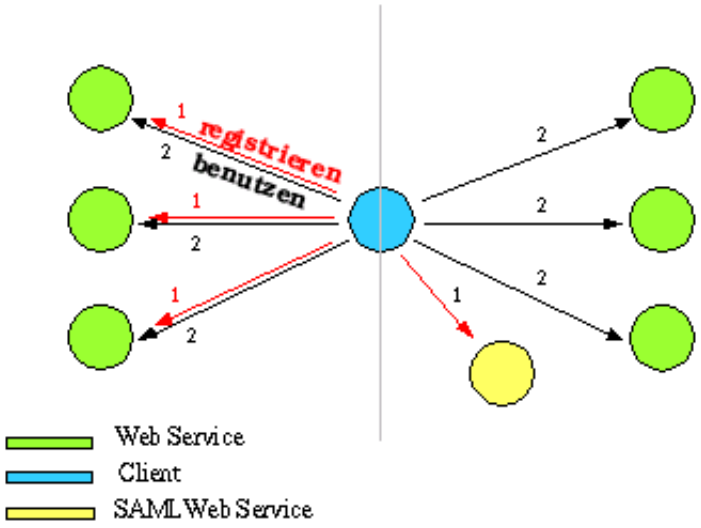

Abbildung 23: Dezentrale Authentifikation u. Autorisation vs. SAML

Der Aufbau einer Assertion besteht aus zwei logisch zu trennenden Teilen:

1. Allgemeine Informationen wie z.B.
   - SAML Version
   - Identifikationsnummer des Tickets
   - Identifikation der Ticket ausstellenden Instanz
   - die Gültigikeitsbedingungen des Tickets (u.a. Zeitspanne)
   - Signatur des Tickets durch die ausstellende Instanz(XML Signature)
2. Authentifikations- und Autorisationsinformationen
   - **Authentication assertion**: Enthält Benutzerinformationen.
   - **Autorisation assertion**: Trifft Aussagen über die Berechtigung zur Verwendung bestimmter Dienste durch den Benutzer. Entsprechende Berechtigungsaussagen müssen vorher zwischen der ausstellenden Instanz und dem Dienstanbieter abgesprochen sein.

- **Attribute assertion**: Es werden Attribute gesetzt, die den Benutzer z.B. einer Benutzerklasse zuordnen. Anhand dieser Informationen kann der Angesprochene Dienst über die Berechtigungen des Benutzers entscheiden.

Während die allgemeinen Informationen durch SAML Elemente spezfiziert werden, müssen sich bei der Angabe von Authentifikations- und Autorisationsinformationen der Web–Service Betreiber und die SAML Instanz auf die Verwendung bestimmter Nachrichten einigen. SAML beschreibt lediglich die drei Elemente in denen diese Informationen innerhalb einer Assertion unterzubringen sind.

Erzielte Sicherheit *Authentifikation* und *Autorisation*.

Sicherheitskritische Betrachtung Der Einsatz von SAML hat aus sicherheitskritischer Sicht zu viele Facetten, als das auf diese im Rahmen dieser Arbeit eingegangen werden könnte. An dieser Stelle wird daher auf die Abhandlung von [SAML-SEC] zu diesem Thema verwiesen.

5 Sicherheitsrisiken beim Einsatz von Web–Services

Die meisten Sicherheitsereignisse können nach [Schneier 00] auf eine oder mehrere der folgenden Systemeigenschaften zurückgeführt werden:

- Komplex
- Interaktiv
- Emergent[18]
- Fehlerbehaftet

Auf Web–Services treffen diese Systemeigenschaften in einem hohen Maß zu. Folgerichtig lässt sich daraus ableiten, dass der Einsatz von Web–Services ein erhebliches Maß an Sicherheitsrisiken in sich birgt. Dieses Kapitel beleuchtet diese und bringt sie, falls sinnvoll, in Bezug mit dem in Kapitel 3 vorgestellten E–Business–Szenario. Desweiteren werden Ansätze zur Risikominimierung vorgestellt. Abschliessend werden, in einer reflektiven Betrachtung, alle besprochenen Sicherheitsrisken übersichtlich dargestellt.

5.1 Strukturelle Schwächen

Dieser Abschnitt erläutert Sicherheitsrisiken aufgrund struktureller Besonderheiten von Web–Services. Diese Risken stellen an sich keinen Angriff dar, sondern nur Schwachstellen, die einen gezielten Angriff erleichtern oder erst ermöglichen.

5.1.1 Offene Ports 80 / 443

Anwendungen verwenden Ports zur Kommunikation mit anderen Programmen im Internet. Ein solcher Port wird durch eine Nummer zwischen 0 und 65535 dargestellt. Die von Browsern verwendeten Transportprotokolle http

[18] „>>Emergenz<< ist ein Begriff aus der neueren englischen Philosophie, wonach höhere Seinstufen durch neu auftauchende Qualitäten aus niederen entstehen“ [Schneier 00]

und https nutzen zu diesem Zweck die Portnummern 80 und 443. Um die Erreichbarkeit eines Web–Servers sicherzustellen, muss in Abhängigkeit der Protokolle mindestens einer dieser Ports 'offen' sein, d.h. dass der Verkehr von einer existierenden Paket–Firewall[19] nicht blockiert, sondern an den entsprechenden Rechner des Unternehmens weitergeleitet wird. Dass Web–Services Aufrufe, so wie herkömmliche Anfragen an das Informationsangebot[20] eines Web–Servers über

- http
- Port 80 / 443

stattfinden, macht eine Unterscheidung dieser durch eine Paket–Firewall unmöglich.

Risikoszenario

Angenommen, EasyBuy hätte die Funktionalität von ebCatalog und ebOrder vorher bereits mit einem proprietären System - z.B. RMI auf Port 1099 - im Einsatz gehabt. Aufgrund der eigenen Port Nummer hatte die bestehende Firewall die Möglichkeit, Zugriffe auf bestimmte IP Adressen oder Adressräume zu beschränken. Der Zugriff auf die neu implementierten Web–Services erfolgt jedoch über http auf Port 80. Würde der Kommunikationsverkehr auf diesem Port blockiert werden, wäre die Folge, dass Interessenten mit Ihren Browsern keinen Zugriff mehr auf das Informationsangebot des Web–Servers bekämen. Dieser soll aber in der Lage sein, jedem Anfragenden das Informationsangebot zu präsentieren. Es bleibt EasyBuy also nichts anderes übrig als die Ports 'offen' zu lassen, was zu Folge hat, dass alle an die Web–Services gerichteten Anfragen die Firewall passieren.

Risikominimierung

Der Einsatz einer Application–Level–Firewall ermöglicht das Filtern von Anfragen anhand von Informationen, die über der IP Schicht liegen. Dazu

[19] Eine Paket Firewall filtert Pakete anhand von Kriterien wie z.B. IP-Adressen, auf IP basierenden Protokollen (TCP, UDP, ICMP) oder Ports.

[20] HTML, PHP o.ä. Anfragen an einen Web–Server

gehört die Möglichkeit der Analyse höherwertiger Protokolle oder anwendungsabhängiger Daten. Diese Firewall ermöglicht die Unterscheidung von 'normalen' http Anfragen und dem Aufruf eines Web–Services anhand z.B.

- des HTTP Content-Type Attributes (Bei Web–Services text/xml).
- des HTTP SOAPAction Attributes (bei SOAP Version 1.2 das action Attribut in der SOAP Nachricht).

Eine Application–Level–Firewall kann dann, anhand vorgegebener Regelwerke entscheiden, wie solche Anfragen an das System zu handhaben sind. Im Vergleich zu paketorientierten Firewalls, benötigen Application–Level–Firewalls ein höheres Maß an Rechenleistung.

5.1.2 SOAP Nachrichtenpfade

Kritisch betrachtet, stellt die in Abschnitt 2.3.2 beschriebene Möglichkeit zur Nutzung von SOAP Nachrichtenpfaden ein Risiko dar. In diesen Szenarien wird eine Nachricht von mehreren Vermittlern bearbeitet, bevor diese beim Zielknoten ankommt. Jeder Vermittler ist dabei zunächst in der Lage die gesamte Nachricht einzusehen oder zu modifizieren, ohne dass dies auffallen müsste.

Risikominimierung

Die Minimierung dieses Risikos findet durch den Einsatz von XML Signature und XML Encryption statt. Die entsprechenden Nachrichtenteile sind dabei für den adressierten Empfänger, je nach benötigter Sicherheit, zu signieren oder zu verschlüsseln. Durch diese Verfahrensweise ist es jedem Knotenpunkt lediglich möglich, die für ihn bestimmten Daten einzusehen oder zu verändern.

5.1.3 Publizierte WSDL Dateien

Publizierte WSDL Dateien liefern detaillierte Beschreibungen über die Schnittstelle eines Web–Services. Ein potentieller Angreifer ist so bereits vor dem Angriff über

- Nachrichtenformat
- Protokollbindungen
- Adresse des Dienstes

informiert. Das ermöglicht dem Angreifer nicht nur das Erstellen syntaktisch korrekter Nachrichten, sondern lässt ggf. sogar Rückschlüsse auf die intern verwendeten Systeme zu. So können Namen und Struktur von RPC Aufrufen Aufschluss über verwendete Programmiersprachen (Namenskonventionen) oder Anwendungssoftware (Schnittstelle ähnelt z.B. in großen Teilen der von bekannten Anwendungen wie z.B. SAP) geben. Diese Informationen ermöglichen gezielte Angriffe auf ein System, die aufgrund des bekannten Nachrichtenformats nur mit einem verhältnismäßig hohen Rechenaufwand als solche erkennbar sind.

Riskoszenario

Nehmen wir an, EasyBuy setzt die Warenwirtschaft des Unternehmens PotentialLeak ein. Diese bietet bereits den Export von Funktionalität als Web–Service an. Die Publikation dieser Schnittstelle würde EasyBuy die Programmier- und Konfigurationsarbeiten zur Erfüllung der Standards ebCatalog und ebOrder ersparen. Ein Angreifer könnte jetzt jedoch anhand der publizierten WSDL Datei Rückschlüsse auf das verwendete System ziehen. Mit diesem Wissen wäre es ihm möglich, gezielt Angriffe auf bekannte Schwachstellen des Systems von PotentialLeak durchzuführen.

Risikominimierung

Die Möglichkeit zur Minimierung des Risikos kann mit folgenden Punkten beschrieben werden:

1. WSDL Dateien nur dann publizieren, wenn sich daraus ein Nutzen ergibt. Es gibt z.B. Szenarien in denen ein Web–Service nur einem bestimmten Personenkreis zugänglich gemacht werden soll. Das Publizieren des Dienstes wäre dann ein unnötiges Risiko.

2. Nur die Funktionalität exportieren, bei der dies sinnvoll erscheint.

3. Schnittstellen bekannter Softwarehersteller nicht 1:1 publizieren, sondern mit leicht modifizierten Schnittstellen darstellen. Bei dem Einsatz einer modifizierten Schnittstelle muss eine Konvertierung (mapping) in das von der Software unterstützte Format erfolgen (siehe Abbildung 24).

4. Eingesetzte Namenskonventionen der RPC Aufrufe sollten keine Rückschlüsse auf die zur Implementierung verwendeten Programmiersprachen zulassen.

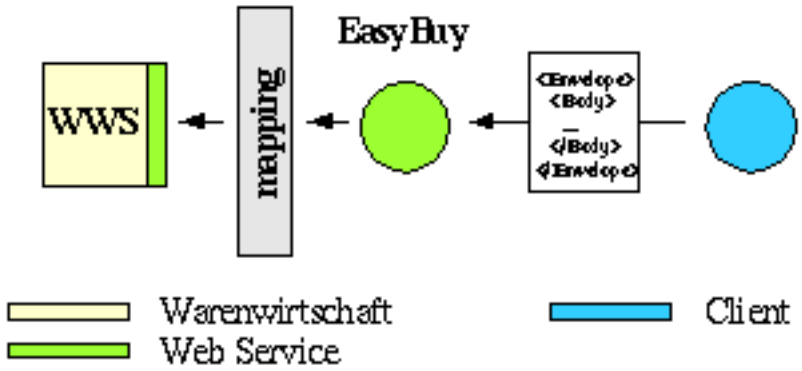

Abbildung 24: Schnittstellen–Mapping

5.1.4 Toolkits

Zur Implementierung von Sicherheitsmechanismen wie z.B. XML Encryption, XML Signature o.a. können Toolkits eingesetzt werden. Ein Toolkit stellt dabei eine Funktionsbibliothek dar, die es einem Programmierer erleichtert die gewünschten Sicherheitsmechanismen zu implementieren. Weil bei neuen Technologien die schnelle Entwicklung von Toolkits Priorität hat, bleibt für das ausgiebige Testen oft keine Zeit. Die Aktualität der Standards führt auch dazu, dass man zum Einsatz nicht Evaluierter Toolkits gezwungen wird. Der Einsatz eines solchen Toolkits könnte so z.B. zur Folge haben, dass

1. vertrauliche Daten von unbefugten trotz Verschlüsselung gelesen werden können (fehlerhafte Implementierung von Algorithmen).

2. Instabilitäten im System auftreten (schlechte Programmierung).

3. das System teilweise unerklärliches Verhalten aufweist (schlechte Programmierung).

Außerdem ist es denkbar, dass beim Zusammenwirken mehrerer funktionierender Toolkits ein unerwünschtes emergentes Verhalten auftritt.

Risikoszenario

Die um Sicherheitsmerkmale erweiterten Standards ebCatalog und ebOrder verlangen, dass XML Signature und XML Encryption von spezifikationskonformen Web–Services implementiert werden müssen. Zur technischen Realisierung dieser Sicherheitsmerkmale beschließt EasyBuy auf ein bestehendes Toolkit[21] zurückzugreifen. Der Implementierungsaufwand für die Sicherheitsmerkmale fällt aufgrund des Toolkits erwartungsgemäß niedrig aus. Der intensive Einsatz der Verschlüsselungsfunktionen lässt den Server aber sporadisch abstürzen.

Risikominimierung

Beim Einsatz von Toolkits muss jemand damit beauftragt werden, sich in regelmäßigen Intervallen über mögliche Schwachstellen und Sicherheitsupdates zu informieren. Bei neuen Toolkits ist wegen des höheren Fehlerpotentials, diese Recherche in häufigeren Intervallen zu vollziehen als bei länger etablierten. Da der damit verbundene Aufwand für kleinere Firmen kaum zu bewerkstelligen ist, empfiehlt es sich diese Aufgabe an einen externen Sicherheitsberater weiterzugeben. Zum heutigen Zeitpunkt ist der wohl beste Ansatz die Sicherheitsmerkmale nicht selber mit Toolkits zu realisieren, sondern auf eine XML–Firewall zurückzugreifen, die diese Funktionalität zur Verfügung stellt. Enthält die XML–Firewall eine Sicherheitslücke, so kann

[21] z.B. WSTK 3.2.2 von IBM

das Problem von dem entsprechenden Unternehmen mit einem Sicherheitsupdate schnell behoben werden.

5.1.5 Emergenz

Nach [Schneier 00] birgt emergentes Verhalten das Risiko von unerwartetem und zum Teil sogar unerklärlichem Verhalten. Das Kombinieren mehrerer feingranularer Web–Services zu größeren Kompositionen birgt also immer auch das Risiko unerwünschtes Verhalten zur Folge zu haben.

Risikoszenario

EasyBuy beschliesst seinen Web–Service zu erweitern. Zukünftig sollen Anfragen, die auf von EasyBuy nicht geführte Produkte abzielen, an Partnerunternehmen weitergeleitet werden. Dies soll für den Benutzer der CheaperProducts Software jedoch unbemerkt geschehen. Zu diesem Zweck hat EasyBuy mit den entsprechenden Partnerunternehmen Verträge abgeschlossen, die besagen, dass

- eine Provision für die erfolgreiche Vermittlung eines Kaufauftrages an EasyBuy zu zahlen ist.
- Verpackung und Rechnung müssen dem Kunden den Eindruck vermitteln, die Ware von EasyBuy bezogen zu haben.

Das System wird getestet und die Funktionalität sowohl von EasyBuy als auch den Partnerunternehmen abgenommen. Im Praxiseinsatz werden jedoch aus unerklärlichen Gründen weitergeleitete Anfragen von einem Partnerunternehmen sporadisch nicht beantwortet. Das System wurde durch die Erweiterung in eine höhere Seinsstufe (erweiterte Funktionalität) gebracht und produziert nicht vorhergesehene Ergebnisse. Die Ursache für den Fehler ist eine Verkettung mehrerer Umstände.

- Das Partnerunternehmen hat seinen Web–Service ebenfalls in einer UDDI publiziert.
- Die Clientsoftware schickt die Anfrage an EasyBuy und an das Partnerunternehmen.

- EasyBuy hat das gesuchte Produkt nicht und leitet die Anfrage an das Partnerunternehmen weiter.
- Das Partnerunternehmen hat bereits die Anfrage des Benutzers erhalten und das Sicherheitssystem des Unternehmens wertet die von EasyBuy weitergeleitete Anfrage als Angriffsversuch.

Obwohl die neu implementierte Funktionalität für sich gesehen einwandfrei funktionierte, traten beim Einsatz Unregelmäßigkeiten auf. Emergentes Verhalten ist also in großen Teilen darauf zurückzuführen, dass Komponenten nicht atomar arbeiten, sondern als ein Gesamtwerk.

Risikominimierung

Um das Risko eines ungewollten Verhaltens bei Erweiterungen zu minimieren, hilft eine gute Kenntnis des Gesamtsystems verbunden mit ausgiebigen Tests. An dieser Stelle sei erwähnt, dass emergentes Verhalten auch oft seinen Ursprung in unsauberer Programmierung hat.

5.1.6 Komplexität

Komplexe Gebilde sind schwer zu durchschauen und damit auch empfänglicher für Sicherheitslücken. Web–Services sind aufgrund Ihrer Charakteristika äußerst komplexe Gebilde. [Yang 02] beschreibt diese in fünf Punkten.

1. Dezentral in Architektur und Administration.
2. Heterogen in der Wahl der Implementierungstechnologien.
3. Abteilungs- und Unternehmensübergreifende Verbindungen.
4. Peer–basierte Architektur.
5. Über das öffentliche Internet zugänglich.

Das zugrundeliegende Problem beim Einsatz komplexer Systeme wird von [Schneier 00] damit schlicht begründet, dass „[...] unzählig viele Dinge schief laufen können.“.

Risikominimierung

Komplexitätsbedingte Risiken sind schwer zu erfassen. [Schneier 00] kommt bei der Minimierung der Risiken auf folgenden Punkt: „Es gibt keinen Ersatz für Einfachheit“. Stehen bei der Realisierung von Web–Services also mehrere Ansätze zur Verwirklichung zur Verfügung, sollte die einfachste Lösung bevorzugt werden.

5.1.7 Fehlermeldungen

Fehlermeldungen informieren den Benutzer eines Systems über das Misslingen einer Operation. Das von Web–Services eingesetzte SOAP Protokoll verlangt bei Verarbeitungsfehlern den Versand eines SOAP Faults. Details zu dem aufgetretenen Fehler werden dabei üblicherweise innerhalb einer der fünf vordefinierten Fehlerklassen (siehe Abschnitt 2.3.5) untergebracht. Die Beschreibung des Fehlers erfolgt dort in menschenverständlicher Art und Weise, was zu einer ausführlichen Fehlerbeschreibung verleitet. Je nach Detailgrad der Informationen sind diese nicht nur zum Verständnis des Fehlers nützlich, sondern auch für die Vorbereitung eines Angriffs.

Risikominimierung

Fehlermeldungen sollten spartanisch gehalten werden. Fehlermeldungen dürfen nach Möglichkeit keine Implementationsdetails oder Rückschlüsse auf verwendete Systeme zulassen. Bei dem Ausfall eines Datenbankservers würde eine Auskunft in der Art „Datenbankserver z.Zt. nicht erreichbar - Probieren Sie es.....“ vollkommen reichen. Zu erwähnen, dass es sich um ein Datenbanksystem von Oracle handelt und der Zugriff auf die Datenbank „Produkte“ nicht erfolgreich verlaufen sei, wäre dagegen nicht nur überflüssig sondern auch ein Sicherheitsrisiko.

5.1.8 Transaktionen

Transaktionen stellen in einer Web–Services Umgebung ein noch nicht vollständig gelöstes Problem dar. Im Bereich der Datenbanken werden mit Transaktionen

eine bestimmte Menge an Operationen nach dem ACID[22] Prinzip entweder vollständig oder gar nicht ausgeführt. Dieses Verhalten wird üblicherweise mit „two-phase commit“ Protokollen realisiert. Diese basieren, im Gegensatz zu denen von Web–Services, auf verbindungsorientierten Protokollen. IBM arbeitet z.Zt. an einer Spezifikation namens WS-Transaktion[23], die sich mit der Problematik beschäftigt. Transkationen in Web–Services Umgebungen stellen komplexe Gebilde dar, deren Handhabung eine anspruchsvolle Aufgabe darstellt. Weil Transaktionen nicht im Betrachtungsrahmen dieser Arbeit liegen, wird an dieser Stelle lediglich auf die Möglichkeit neuer Sicherheitsrisiken durch den Einsatz dieser hingewiesen.

5.2 Gezielte Angriffe

Die in diesem Abschnitt aufgeführten Sicherheitsrisken beziehen sich auf greifbare Angriffe gegen Web–Services. Die Betrachtung beschränkt sich dabei auf Angriffe die auf der Anwendungsschicht stattfinden. Es sein an dieser Stelle erwähnt, dass Angriffe auf den niedrigeren Schichten des OSI Modells die Sicherheit und Verfügbarkeit eines Web–Services ebenfalls betreffen können. Informationen zum Schutz vor derartigen Angriffen werden von [Siyan 95] beschrieben.

5.2.1 Data Injection

Data Injections werden möglich, wenn die Parameter eines Funktionsaufrufs darin Verwendung finden, interpretiert oder ausgeführt zu werden. Der Angreifer versucht dabei den Parameter so zu formulieren, dass bei der Verarbeitung z.B.

- vertrauliche Daten zurück gegeben werden.
- die unberechtigte Modifikation oder Löschung von Datenbeständen erfolgt.
- unberechtigt Systemressourcen genutzt werden können

[22] **A**tomicity, **C**onsistency, **I**solation and Durability.
[23] http://www-106.ibm.com/developerworks/library/ws-transpec/

Die Gefahr der übergebenen Daten geht dabei direkt von der Interpretierenden Anwendung aus. Der Umstand, dass für jede Anwendung andere Datenkonstrukte gefährlich sein können, macht die Verwendung eines universellen Datenscanners unmöglich.

Risikoszenario

EasyBuy könnte sich aus Gründen der Performance dazu entschließen, Produktanfragen direkt an die Datenbank zu stellen. Das modifizierte Szenario für ebCatalog Anfragen sähe dann wie in Abbildung 26 dargestellt aus. Aus technischer Sicht wird dabei das Suchkriterium einer ebCatalog Anfrage in ein SQL Grundgerüst eingefügt. Dieses Grundgerüst sieht für das gegebene Szenario wie in Abbilung 25 dargestellt aus, wobei das <Suchkriterium> den variablen Teil einnimmt. Das aus einer Anfrage generierte SQL State-

```
SELECT * FROM Produkte WHERE produktname = <Suchkriterium>
```

Abbildung 25: SQL Grundgerüst

ment wird zur Verarbeitung direkt an die Datenbank weitergereicht, ohne dass die Warenwirtschaft in den Vorgang einbezogen wird. Die resultierende Ergebnismenge (ResultSet) wird von dem Web–Service gemäß der ebCatalog Spezifikation aufgearbeitet und dem Client zugeschickt. Die eigentliche

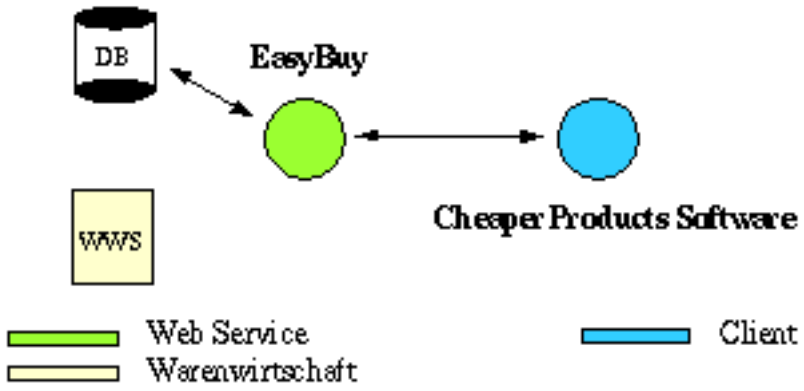

Abbildung 26: Modifiziertes ebCatalog Szenario

Gefahr besteht darin, dass die vom Client zugesandten Daten von der SQL

Engine interpretiert werden. Ein Angreifer würde sich dies zunutze machen indem er versucht SQL Befehle als Parameter zu übergeben. Das Suchkriterium `; drop table Produkte` könnte so - eine besonders schlechte Rechtekonfiguration der Datenbank vorausgesetzt - die Löschung der Produktdatenbank zur Folge haben. Der hier beschrieben Angriff ist bereits unter dem Namen „SQL Injection“ bekannt. Mit der Verbreitung von Web–Services und der damit verbundenen Öffnung einer Vielzahl von Anwendungen gegenüber dem Internet, werden noch zahlreiche andere Formen der Data Injection Bekanntheit erlangen.

Risikominimierung

[Anley 02] beischreibt zwei komplementäre Ansäzte zur Abwehr von SQL Injection Attacken.

1. **Input Validation**: Dabei werden die empfangenen Daten auf das Vorhandensein unerwünschter Zeichen oder Zeichenketten untersucht. Das getarnte Einschleusen unerwünschter SQL Befehle wird damit verhindert.

2. **SQL Server Lockdown**: Hier handelt es sich um die Absicherung des SQL Servers. Insbesondere wird hier Wert auf das Einspielen von Sicherheitspatches und eine kritische Rechteverwaltung gelegt. Eine gute Checkliste zur Absicherung eines Microsoft SQL Servers ist unter [SQL-SEC] zu finden.

Beide Ansätze können, nach einer allgemeineren Betrachtung der SQL spezifischen Anteile, auch als Strategie zur Minimierung von Data Injections verwendet werden.

5.2.2 DoS und verteilte DoS

Jede Anfrage an ein bestehendes System verbraucht Ressourcen. Das ist der Ansatzpunkt für **D**enail **o**f **S**ervice Attacken. Ein Angreifer versucht dabei mit einem oder mehreren Rechnern in kurzer Zeit so viele Anfragen an ein System zu richten, bis dieses schließlich aufgrund der hohen Last seine

Verfügbarkeit verliert. Diese Art der Angriffe stellen für Web–Services eine besondere Gefahr dar. Im Gegensatz zu den relativ einfach zu handhabenden Zugriffen auf Internetseiten, löst der Aufruf eines Web–Services komplexe Vorgänge aus.

- Bilden des DOM Baums
- Prüfung auf Wohlgeformtheit und Gültigkeit
- Prüfung eines Zertifikates
- Prüfung oder Erstellung einer Signatur
- Ver- oder Entschlüsselung von Nachrichtenteilen
- Parsen der Nachricht
- Ausführen von Programmcode

Nach [Yang 02] ist die Besonderheit von DoS Angriffen in einem Web–Services Umfeld auf die heterogenen Schnittstellen dieser zurückzuführen. So könnten bei einem rechenintensiven Web–Service 10 Anfragen in der Stunde bereits einen DoS Angriff darstellen, während für einen anderen 1000 Anfragen in der Sekunde ohne weiteres zu bewältigen sind.

Risikoszenario

EasyBuy bekommt eine Fülle von ebCatalog Anfragen von einem Angreifer. Der Angreifer könnte mit dem Versand der Nachrichten zwei Ziele verfolgen:

1. Das Überlasten des Web–Services Rechners. Im Gegensatz zum Versender der Nachrichten, muss dieser für jede ankommende Nachricht mehrere der weiter oben aufgeführten Schritte ausführen. Bei entsprechend vielen Anfragen führt dies zum Verlust der Verfügbarkeit für den Web–Services Rechner.
2. Das Überlasten des Warenwirtschaftssystems. Der Angreifer geht aufgrund des Charakters von ebCatalog folgerichtig davon aus, dass die

Anfragen von einem Warenwirtschaftsystem verarbeitet werden. Der Angreifer formuliert dazu Produktanfragen so vage, dass die Masse gefundener Produkte die Warenwirtschaft bereits mit wenigen Anfragen überlastet.

Während bei dem ersten Angriff 'nur' die Verfügbarkeit des Web–Services betroffen ist, würde mit dem Zweiten auch die hausinterne Verfügbarkeit des Systems in Mitleidenschaft gezogen werden. Produktanfragen oder Bestellungen könnten dann z.B. nicht einmal mehr von einem Callcenter bearbeitet werden.

Risikominimierung

Ein Angriff auf die Verfügbarkeit des Web–Services Rechners kann durch eine XML–Firewall[24], verhindet werden. Diese könnte nach dem Erhalt einer bestimmten Menge von Anfragen gleichen Ursprungs innerhalb einer vorgegebenen Zeitspanne eine packetorientierte Firewall anweisen, Anfragen der entsprechenden IP Adresse zu sperren. Weitere Angriffsversuche würden dann ressourcenschonend auf der Netzwerkebene abgelehnt werden.
Um den Angriff auf Legacy Applications zu vereiteln, muss eine solche Firewall Möglichkeiten zur Prüfung kritischer Parameter zur Verfügung stellen. Dabei ist nicht nur die Prüfung der Nachricht auf Wohlgeformtheit und Gültigkeit von Bedeutung, sondern auch die Möglichkeit bestimmte Parameter einer Nachricht nach besonderen Regelwerken zu überprüfen. Eine ebCatalog Anfrage mit dem Suchkriterium '*' mag nach den Gesichtspunkten der ebCatalog Schnittstellenbeschreibung durchaus korrekt erscheinen. Führt dieses jedoch in der Warenwirtschaft dazu, dass der Rückgabewert den gesamten Produktkatalog darstellt, muss dies durch entsprechende Prüfungen verhindert werden (siehe dazu auch Data Injection aus Abschnitt 5.2.1).

[24]Eine solche XML–Firewall wird z.B. von Westbridge Technologies vertrieben (www.westbridgetech.com).

5.2.3 Böswillige Modifikation

Daten werden auf dem Weg zum Zielknoten von einem unbefugten Dritten unbemerkt modifiziert.

Risikoszenario

Entschließt sich ein Benutzer der CheaperProducts Software zu einer Bestellung, kann diese in allen möglichen Varianten modifiziert werden, ohne dass dies von EasyBuy entdeckt werden könnte. Im harmlosesten Fall würde die Nachricht nach der Modifikation ihre Gültigkeit verlieren, schlimmstenfalls wären jedoch Mengen- und Preisangaben oder Artikelnummern von Veränderungen betroffen.

Risikominimierung

Die Integrität der Daten kann durch den Einsatz von XML Signature (siehe Abschnitt 4.3.2) sichergestellt werden. Der Einsatz solcher Signaturen ermöglicht neben der Integritätsprüfung auch die Identifikation des Versenders. In dem gegebenen Szenario besteht bei der Implementierung von XML Signature ein Hindernis. EasyBuy muss bei der Implementierung seiner Web–Services konform zu den gegebenen Standards bleiben. Ist in diesen der Einsatz von Sicherheitsmechanismen nicht vorgesehen, wird EasyBuy, um spezifikationskonform zu bleiben, auf die Implementation solcher verzichten müssen.

5.2.4 Spionage

Daten werden von unbefugten Dritten eingesehen.

Risikoszenario

Ein Unternehmen interessiert sich für das Kaufverhalten bestimmter Zielgruppen. Dazu werden alle an EasyBuy versandten Nachrichten (ebCatalog / ebOrder) abgehört und gespeichert. Diese personenbezogen Daten könnten zukünftig dazu genutzt werden, den betreffenden Personen eine Fülle an

Werbesendungen zukommen zu lassen, die auf das entsprechende Kaufverhalten abgestimmt sind. Eine andere Motivation könnte auch sein, die Daten gezielt zur Erpressung oder Verunglimpfung dieser Personen einzusetzen.

Risikominimierung

XML Encryption (siehe Abschnitt 4.3.3) muss zur Verschlüsselung aller vertraulichen Daten eingesetzt werden. Analog zu dem Einsatz von XML Signature, muss auf die Konformität mit evtl. eingesetzten Standards geachtet werden.

5.2.5 Replay Attacke

Anfragen an einen Web–Service können von unbefugten Dritten aufgezeichnet und dann erneut eingespielt werden. Dies kann auch mit Nachrichten geschehen, die durch die Verwendung von XML Signature und XML Encryption, als vertraulich und integer einzustufen sind. Die Ursache für diesen Angriff liegt in der fehlenden Verbindlichkeit der Kommunikation. Beim Versand inhaltlich identischer Nachrichten kann nicht festgestellt werden, ob beide Nachrichten vom Versender unterzeichnet worden sind, oder ob es sich um das erneute Einspielen einer vorher abgehörten Nachricht durch einen Angreifer handelt.

Risikoszenario

In dem gegebenen Szenario wird eine Bestellung gemäß der ebOrder Spezifikation aufgegeben. Um Sicherheitslücken zu schließen, wurde die Spezifikation zwischenzeitlich um den Einsatz der Standards XML Signature und XML Encrpytion erweitert. Die Nachricht wird von einem Angreifer abgefangen und zu einem späteren Zeitpunkt erneut an den Web–Service von EasyBuy übertragen. Obwohl der Angreifer in diesem Fall weder den Inhalt kennt, noch imstande ist diesen unbemerkt zu verändern, stellt das erneute Einspielen der Nachricht einen erheblichen Sicherheitsmangel dar. Das System von EasyBuy erhält eine syntaktisch korrekte Nachricht, die gemäß der Spezifikation sowohl korrekt verschlüsselt als auch signiert ist. Die Bestellung

würde vom System als korrekt erkannt und erneut ausgeführt werden. Wird der Angriff nicht bemerkt, kann dies einem Unternehmen einen erheblichen Finanz- und Imageschaden zufügen.

Risikominimierung

Die Verbindlichkeit der Kommunikation muss sichergestellt werden. Nachrichten müssen dabei um zusätzliche Merkmale erweitert werden, die z.B. bei identischen Bestellungen eine Unterscheidung ermöglichen. Dies kann durch den Einsatz von Zeitstempeln oder Sequenznummern erreicht werden. Ein Format für solche Zeitstempel wird von der WS-Security Spezifikation (siehe Abschnitt 4.3.4) vorgestellt. Die alleinige Verwendung solcher Zeitstempel genügt dabei nicht. Die interne Verwaltung und Kontrolle dieser bringt erst die gewünschte Sicherheit mit sich. Der Einsatz von Zeitstempeln o.ä. kann in dem gegebenen Szenario ebenfalls erst nach der entsprechenden Erweiterung der Spezifikationen erfolgen.

5.2.6 Buffer Overflow

Ein Buffer Overflow tritt auf, wenn von einem Prozess eingelesene Daten den dafür reservierten Speicherbereich überschreiten. Ein solcher Angriff kann es ermöglichen Maschinencode mit den Rechten des jeweiligen Prozesses ausführen zu lassen. Die Ursache dafür ist unsaubere Programmierung. Von Programmiersprachen wie z.B. C oder C++ wird die Reservierung des Speichers für Variablen dem Programmierer überlassen. Schreibt dieser in seinem Programm mehr Daten in den Speicher als im Voraus dafür reserviert, werden andere Speicherbereiche überschrieben. Eine sehr gute Beschreibung der Thematik bietet [Bauer].

Mit diesem Angriff wird der Grundstock für ein breites Spektrum an Fehlverhalten gelegt. Eine Buffer Overflow Attacke kann damit die Systemintegrität erheblich gefährden. Web–Services sind wegen der Möglichkeit, die Funktionalität beinahe jeder Anwendung exportieren zu können, besonders gefährdet. Ältere Anwendungen (Legacy Applications) sind häufig nicht mit Blick auf die Möglichkeit eines missbräuchlichen Einsatzes entwickelt

worden und sind so besonders verwundbar für derartige Angriffe.

Risikoszenario

Die Warenwirtschaft von EasyBuy stellt zur Produktsuche eine Funktion zur Verfügung. Diese Funktion reserviert für die Entgegennahme eines Parameters Speicher für 10 Zeichen. Da es sich um ein proprietäres System handelt, für das der Hersteller sauber arbeitende Clients programmiert hat, findet in der eigentlichen Suchfunktion der Warenwirtschaft keine Überprüfung der Parameterlänge statt. Die XML Schema Datei des ebCatalog Standards definiert ein Suchkriterium mit der Länge von 15 Zeichen. Eine Suchanfrage mit der Länge von 14 Zeichen würde demnach beim Parsen der entsprechenden Nachricht als korrekt erkannt. In der Warenwirtschaft würden die vier überschüssigen Zeichen zu einem Buffer Overflow führen.

Risikominimierung

Die Vorgehensweise ist ähnlich der in Abschnitt 5.2.2 vorgestellten Methode zur Vermeidung von DoS Attacken. Eine XML Firewall sollte hier ein zusätliches Regelwerk zur Überprüfung und Modifikation bestimmter Datentypen bereitstellen. Ein 14 Zeichen langer Parameter kann so problemlos erkannt werden, bevor der eigentliche Funktionsaufruf erfolgt. Ob eine solche Zeichenkette abgelehnt oder nach einer entsprechenden Kürzung weiterverarbeitet werden soll, ist dabei von einer Sicherheitsrichtlinie festzulegen.

5.2.7 WSDL Spoofing

Von Spoofing spricht man bei der Vortäuschung einer falschen Identität. Nach [Russel 01] ist die besondere Gefahr des Spoofing, dass „[...] die brutalsten Spoofing-Angriffe nicht nur die Identität des Angreifers vertuschen, sondern die Tatsache, dass ein Angriff überhaupt stattgefunden hat.“. Beim WSDL Spoofing wird die falsche Identität durch die Mainpulation einer publizierten WSDL Datei erreicht. Die URL an die der Dienst eines Unternehmens gebunden wurde, wird hier durch eine vom Angreifer kontrollierte ersetzt. Bei einer automatischen Bindung durch eine Clientsoftware würde dies - im

Gegensatz zu der Eingabe in einem Browser - nicht unbedingt auffallen. Der Angreifer kann durch die bekannte Schnittstelle vortäuschen der angeforderte Dienst zu sein und so an vertrauliche Informationen gelangen, oder gezielt Fehlinformationen verbreiten.

Risikominimierung

Das Signieren von WSDL Dateien ermöglicht es dem Client eine Integritätskontrolle durchzuführen. Ist diesem allerdings der öffentliche Schlüssel des Unternehmens nicht im Voraus bekannt, reicht eine Signatur nicht.
Angenommen ein Angreifer tauscht die Signatur in der WSDL Datei durch seine eigene und ersetzt einige der publizierten Unternehmensdaten in der UDDI. Eine XKMS Anfrage mit den manipulierten Unternehmensdaten würde jetzt den öffentlichen Schlüssel des Angreifers zurückliefern. Die Signatur der WSDL Datei wäre Gültig! Abhilfe können in solchen Szenarien nur Vertrauensinfrastrukturen schaffen, in denen nur Schlüssel eines bestimmten XKMS Dienstes (TTP) als vertrauenswürdig eingestuft werden.

5.3 Reflektive Betrachtung

Konzeptionelle Sicherheit bereichert Web–Services Nachrichten zwar an Sicherheitsmerkmalen wie Authentifikation, Autorisiation, Vertraulichkeit, Integrität und Verbindlichkeit, führt jedoch alleine nicht zu sicheren Systemen. Eine hohe Annäherung an dieses Ziel wird erst durch die ganzheitliche Betrachtung aller Sicherheitsrisiken erreicht. Bei dieser Betrachtung lassen sich die Eindämmungsstrategien in drei Kategorien aufteilen.

1. **Konzeptionell**: Unter diese Kategorie fallen alle Risiken, die durch den Einsatz von Sicherheitsstandards eliminiert werden.

2. **Architektonisch**: Hierunter fallen Risiken, die durch das Einspielen von Sicherheitsupdates, die Architektonische Anordnung von Systemen, oder durch die vernünftige Abwehr von erkannten Angriffen eingedämmt werden können.

3. **Datenorientiert**: Risiken, die dieser Kategorie untergeordnet werden, entstehen durch die Nichterkennung bösartiger Daten.

Die Tabelle in Abbildung 27 stellt die Zuordnung der betrachteten Sicherheitsrisiken zu jeweils einer der Kategorien her. Diese Tabelle soll bei der Planung von Web–Services als Gedankenstütze Verwendung finden. Bei der Eindämmung Architektonischer Risiken spielt die XML–Firewall eine entscheidende Rolle. Diese Firewalls arbeiten auf der Anwendungsschicht und sind daher viel komplexer als herkömmliche Firewalls. In [Firewalls] werden die Aufgaben einer XML–Firewall der einer gewöhnlichen Paket–Firewall gegenübergestellt. Solche Firewalls stellen aufgrund ihrer Komplexität eigenständige Sicherheitssysteme dar.

Sicherheitsrisiko	**Typ**	**Lösung**
SOAP Nachrichtenpfade	Konzeptionell	XML Encryption und XML Signature für jeden Vermittler
Spionage	Konzeptionell	XML Encryption
WSDL Spoofing	Konzeptionell	WSDL signieren
Replay Attacke	Konzeptionell	Timestamp oder Sequenznummern
Offene Ports	Architektonisch	Application–Firewall / XML–Firewall
Toolkits	Architektonisch	XML–Firewall
Emergenz	Architektonisch	- Kenntnis des Gesamtsystems - Saubere Programmierung - Realistische Tests
Publiziertes WSDL	Architektonisch	WSDL nur bei Mehrwert exportieren
Komplexität	Architektonisch	Systeme so einfach wie möglich halten
Fehlermeldungen	Architektonisch	Fehlermeldungen spartanisch halten
DoS und verteilte DoS	Architektonisch	XML–Firewall muss eine Ablehnung auf IP Basis unterstützen
Buffer Overflow	Datenorientiert	Regeln zur Datenprüfung müssen erstellt und angewandt werden
Data Injection	Datenorientiert	Regeln zur Datenprüfung müssen erstellt und angewandt werden

Abbildung 27: Sicherheitsrisiken Web–Services

6 Datenorientierte Sicherheit

Ein wesentlicher Bestandteil von Web–Services ist die Verwendung XML basierter Nachrichten. Dieses Kapitel beschäftigt sich mit den Gefahren, die von diesen Daten ausgehen und stellt ferner einen Weg vor, diese Einzudämmen. Der hier vorgestellte Ansatz versteht sich nicht als vollständige Lösung, vielmehr als Basis einer solchen.

6.1 Kritische Daten

XML Nachrichten stellen, wie alle textbasierten Nachrichten, atomar gesehen kein Sicherheitsrisiko dar. Die Gefahr geht, wie in den Abschnitten 5.2.1 und 5.2.6 beschrieben, von der Interpretation der Daten durch eine Anwendung aus. Die Tatsache, ob die Zeichenkette `format c: /u` von einer MSDOS Shell[25] oder einer Suchmaschine interpretiert wird, ist also von entscheidender Bedeutung. Die Frage nach der von Daten ausgehenden Gefahr kann also nur im Kontext der Anwendung gestellt werden, die für die Interpretation zuständig ist.

6.2 Wichtige Fragen

Um ein Profil ungefährlicher Daten für einen bestimmten Web–Service zu erhalten, müssen zunächst einige Fragen an das verarbeitende System gestellt werden. Im Kern muss durch die Beantwortung dieser festgehalten werden, welche Daten erlaubt sind und welche dem System Schaden zufügen können. Um ein Höchstmaß an Sicherheit zu erzielen, sollten die Fragen aus zwei Blickwinkeln gestellt werden.

1. Was wird benötigt?
2. Was ist verboten?

Die bewusste Einschränkung des Erlaubten, soll das Risiko von Fehlverhalten durch vergessene Verbote minimieren. Im Folgenden wird ein auf diesem

[25] Befehlsschnittstelle zum Betriebssystem

Prinzip beruhender Fragenkatalog vorgestellt, der zur Kontrolle von Parametern auf der Basis von Zeichenketten dienen soll.

- Welche Länge darf die Zeichenkette haben?
- Werden Buchstaben benötigt?
- Werden Sonderzeichen benötigt?
- Werden Ziffern benötigt?
- Gibt es Zeichen die verboten werden müssen (z.B. Steuerzeichen)?
- Gibt es Zeichenketten die verboten werden müssen (z.B. SQL oder Shell Befehle)?

6.3 Ablaufdiagramm

Jede der im vorangegangenen Abschnitt vorgestellten Fragen trägt zur Erstellung eines Regelwerks bei. Obwohl die Prüfung der Daten erst nach der vollständigen Beantwortung aller Fragen erfolgen kann, stellt jede Frage bereits Anforderungen an die Implementierung. Das in Abbildung 28 vorgestellte Diagramm veranschaulicht sowohl die Fragen als auch Hinweise auf Implementationsdetails. Es handelt sich dabei nicht um einen Programmablauf, sondern vielmehr um eine Übersicht des aufgestellten Regelwerks mit Implementationshinweisen. Auf der Internetseite http://www.michael-hildebrandt.de/diplomarbeit/checkliste.html existiert ein Demonstrator zur Validierung von Eingabedaten anhand des vorgegebenen Regelwerkes. Dabei wurde das Ablaufdiagramm sowohl bei der Implementation der Funktionalität als auch zur Erstellung der in Abbildung 29 vorgestellten Checkliste eingesetzt. Der Aufbau der Checkliste fand auch als Vorlage für die Regeleingabe auf der Internetseite Verwendung.

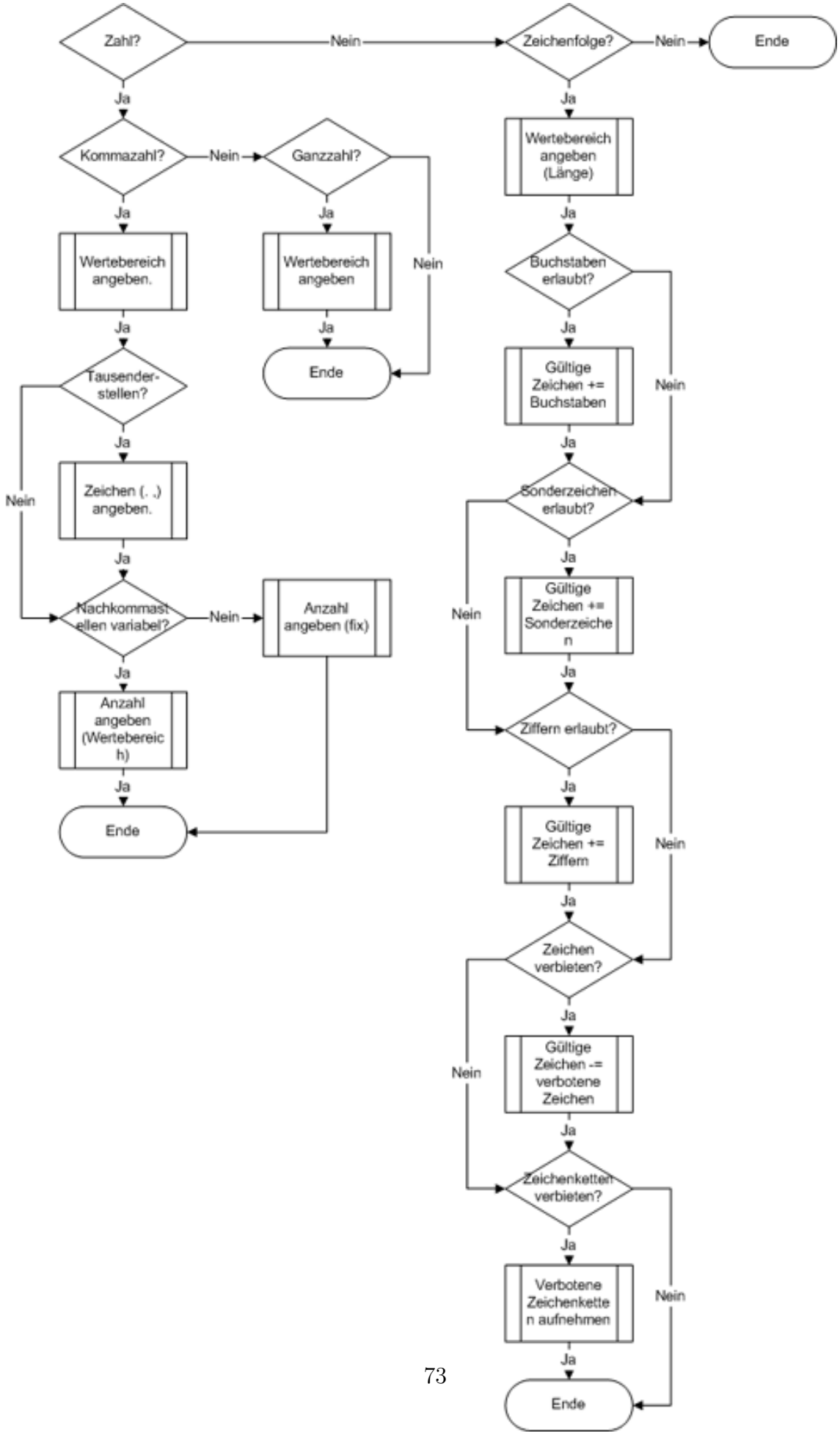

Abbildung 28: Regelerstellungsdiagramm

6.4 Checkliste

Bei der Dokumentation gültiger Daten kann die Verwendung einer Checkliste hilfreich sein. Dabei ist eine Checkliste für jeden Parameter eines Web–Services auszufüllen. Bei gleichartigen Web–Services kann eine ausgefüllte Checkliste als Profil für eine ganze Gruppe dienen. Die in Abbildung 29 vorgestellte Checkliste wurde anhand des Ablaufdiagramms erstellt. Sie beschränkt sich auf die Betrachtung von Zeichenfolgen.

[Checkliste Parameter Zeichenfolge]

Web--Service: _____

Parameter: ____ _____
(Nr.) (Name)

Anwendung: _____

Funktion: _____

Variable Zeichenfolgen

Länge Zeichenkette: min: _____ max:____

Buchstaben erlaubt: ja: ☐ nein: ☐

Sonderzeichen erlaubt: ja: ☐ nein: ☐

Ziffern erlaubt: ja: ☐ nein: ☐

Case sensitive: ja: ☐ nein: ☐

Verbotene Zeichen: (1)_____

(2)_____

Anlage: _____

Verbotene Zeichenketten:(1)_____

(SQL Befehle, Escape Sequenzen, usw.) (2)_____

Anlage: _____

Abbildung 29: Parameter Checkliste für Web–Services

7 Angewandte Sicherheit in einem Web–Services Szenario

Dieses Kapitel beschäftigt sich mit der exemplarischen Absicherung des in Kapitel 3 vorgestellten Web–Services Szenarios. Ziel ist es, den praktischen Einsatz der in Kapitel 4 vorgestellten Sicherheitskonzepte aufzuzeigen und Ideen zur Vorgehensweise bei der Absicherung eines Web–Services Szenarios zu geben.

7.1 Ausgangssituation

EasyBuy hat seinen Web–Service auf Basis der Standards ebCatalog und ebOrder implementiert. Aufgrund der entstandenen Sicherheitsrisken beschließt die Unternehmensleitung die eingeführten Web–Services mit adäquaten Mitteln absichern zu lassen.

7.2 Sicherheitskritische Betrachtung

Um das benötigte Maß an Sicherheit festzustellen, werden in diesem Abschnitt die wichtigsten Akteure des Szenarios punktuell beleuchtet. Dabei handelt es sich um

- die Datenbank.
- die Warenwirtschaft.
- den Web–Service.
- den UDDI Dienst.
- den Client.
- den Payment Provider

Diese Betrachtung, bildet die Grundlage zur Erstellung einer sicheren Web–Service Umgebung.

7.2.1 Die Datenbank

Der Zugriff auf die Datenbank erfolgt nicht unmittelbar durch den Web–Service, sondern über das Warenwirtschaftssystem. Um uneingeschränkt mit den Datenbeständen arbeiten zu können, erhält dieses Administrationsrechte auf der Datenbank. Ob einzelne Benutzer bestimmte Datensätze sehen, verändern oder löschen dürfen, unterliegt also der Entscheidung der Warenwirtschaft. Zu diesem Zweck verwaltet sie Benutzer unterschiedlicher Berechtigungsstufen. Ein erfolgreicher Buffer Overflow Angriff auf die Warenwirtschaft könnte diese Berechtigungsstufen umgehen und dem Angreifer den uneingeschränkten Zugriff auf die Datenbestände der Warenwirtschaft verschaffen. Weil die Programmierung der Warenwirtschaft keine andere Rechtekonfiguration auf der Datenbank zulässt, kann die Gefahr an dieser Stelle nur zur Kenntnis genommen werden.

7.2.2 Die Warenwirtschaft

Die Web–Services exportieren in dem gegebenen Szenario die Funktionalität der Warenwirtschaft. Daten die in den vorgegebenen XML Nachrichten an den Service geschickt werden, müssen demnach zur Verarbeitung an die Warenwirtschaft weitergeleitet werden. Bisher findet der Zugriff auf die Warenwirtschaft nur über die unternehmensintern eingesetzte Clientsoftware des Herstellers statt. Zum Umfang der Warenwirtschaft gehört jedoch auch eine API[26], mit der der Zugriff auf die Funktionalität der Warenwirtschaft auch aus eigenen Programmen heraus erfolgen kann. Die benötigte Funktionalität der Web–Services wird von zwei Funktionen der API zur Verfügung gestellt:

1. getProductList (ebCatalog)
2. doOrder (ebOrder)

Die fehlerfreie Funktionalität der API setzt jedoch die Verwendung gültiger Eingabedaten voraus. Zu diesem Zweck muß zunächst die Dokumentation gültiger Eingabedaten erfolgen, die z.B. anhand der in Kapitel 6 vorgestellten Checkliste erfolgen kann. Abbildung 30 zeigt eine beispielhaft ausgefüllte

[26] **A**pplicatin **P**rogramming **I**nterface

Checkliste für die Daten des ebCatalog Web–Service. Um an die Informationen zum Erstellen solcher Dokumentationen zu kommen, können unterschiedliche Bezugsquellen in Betracht kommen. Zu diesen zählen z.B.:

- Die Schnittstellenbeschreibung der verwendeten API.
- Eine Anfage beim Hersteller der verwendeten Software.
- Eigene Versuchsreihen in einer Testumgebung.

In dem Szenario muss - z.B. durch den Einsatz einer XML–Firewall - sichergestellt werden, dass die Regeln der Dokumentation eingehalten werden, bevor die Daten an entsprechende Funktionen der API weitergegeben werden. Geschieht dies nicht, sind Buffer Overflow oder Data Injection Angriffe mögliche Folgen. Die Administrationsrechte der Warenwirtschaft auf der Datenbank, würden in diesem Fall von einem theoretischen Risiko zu einer praktischen Gefahr werden.

7.2.3 Der Web–Service

Der Web–Service stellt die Schnittstelle zu einer bestimmten Funktionalität dar. Weil jeder Dienst verschiedene Anforderungen an die Informationssicherheit stellt, müssen zunächst für jeden die Fragen nach

- eingesetzten Standards
- Authentifikation
- Autorisation
- Vertaulichkeit
- Verbindlichkeit
- Integrität
- Verfügbarkeit

beantwortet werden.

[Checkliste Parameter Zeichenfolge]

Web--Service: **ebCatalog**

Parameter: **1** (Nr.) **<Suchkriterium>** (Name)

Anwendung: **Warenwirtschaftsystem**

Funktion: **getProductList (cKriterium[12])**

Variable Zeichenfolgen

Länge Zeichenkette:	min: **3**		max: **12**	
Buchstaben erlaubt:	ja:	☒	nein:	☐
Sonderzeichen erlaubt:	ja:	☐	nein:	☒
Ziffern erlaubt:	ja:	☒	nein:	☐
Case sensitive:	ja:	☐	nein:	☒

Verbotene Zeichen: (1) *

(2) _____

Anlage: _____

Verbotene Zeichenketten: (1) _____

(SQL Befehle, Escape Sequenzen, usw.) (2) _____

Anlage: _____

Abbildung 30: Checkliste ebCatalog

Standards vereinfachen in der Web–Services Welt vieles. Durch den Einsatz der Standards ebCatalog und ebOrder steht das Angebot von EasyBuy z.B. schlagartig auf vielen elektronischen Marktplätzen zur Verfügung. Der Vorteil wird daraus gezogen, dass sich die beteiligten Instanzen an vorgegebene Regeln halten. Für die Sicherheit von Web–Services Umgebungen bedeutet dies allerdings auch, dass benötigte Sicherheitsmechanismen von den eingesetzten Standards bedacht werden müssen. Bei den folgenden Betrachtungen wird von dem Einsatz der Standards in der Version 2.0 ausgegangen. Diese unterstützen den Einsatz von XML Signature, XML Encryption, WS-Security, XKMS und SAML.

Authentifikation und Autorisation wird von den Spezfikationen (ebCatalog, ebOrder) durch die Nutzung eines vorgegebenen SAML Dienstes vorgesehen. Bei diesem Dienst handelt es sich um eine Trusted Third Party (TTP) die Assertions zur Verwendung beliebiger ebCatalog und ebOrder Dienste vergibt. In dem gegebenen Szenario enthält eine solche Assertion

- den Benutzernamen.
- das X.509 Zertifikat des Benutzers.
- Autorisationsanweisungen (ist berechtigt ebCatalog und ebOrder Dienste zu nutzen).
- den richtigen Namen und die Anschrift des Benutzers.
- das Kreditlimit.
- usw..

Für die Richtigkeit der vom Benutzer angegebenen Daten verbürgt sich die SAML Instanz. Die Kosten der notwendigen Überprüfungen werden von den profitierenden Unternehmen getragen, um potentielle Kunden zu ködern. Der SAML Web–Service signiert alle ausgestellten Assertions um deren Echtheit zu bestätigen. Die vertraulichern Daten innerhalb der Assertion werden verschlüsselt an den Kunden (Client) übertragen. Um die Assertion nutzen zu

können, muss der Kunde diese zunächst entschlüsseln und dann für den jeweils adressierten Empfänger (Web–Service) erneut verschlüsseln.

Vertraulichkeit und Integrität werden durch die Verwendung von XML Signature und XML Encryption erreicht. Der Einsatz dieser Sicherheitsmechanismen innerhalb der übertragenen SOAP Nachrichten, findet unter Verwendung der WS-Security Spezifikation statt. Die vertrauliche und integere Kommunikation zwischen Web–Service und Client findet wie in Abbildung 31 dargestellt statt. Vorausgesetzt wird dabei, dass sich der Client bereits in dem Besitz einer berechtigenden Assertion befindet. Der Client (z.B. Chea-

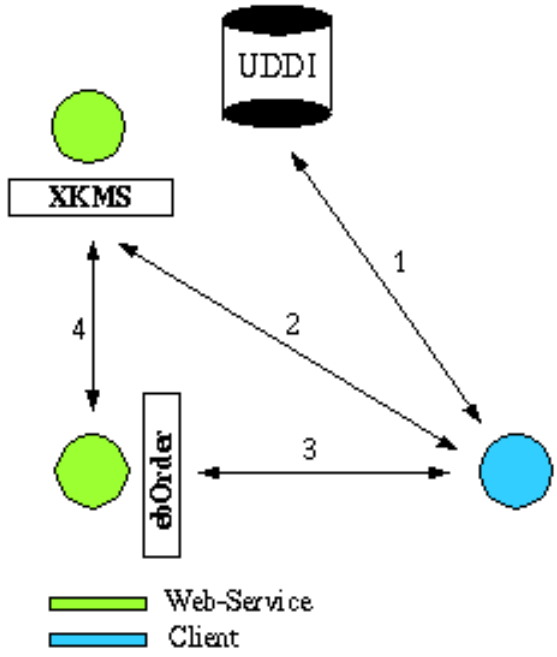

Abbildung 31: Vertraulichkeit und Integrität

perProducts Software) sucht in einer UDDI nach Diensten, die ebCatalog und ebOrder unterstützen. Die von der UDDI zurückgelieferten Informationen der entsprechenden Unternehmen nutzt der Client, um sich von einem vorgegebenen XKMS Service den öffentlichen Schlüssel EasyBuys geben zu lassen. Mit diesem verschlüsselt er alle vertraulichen Informationen der Nachricht und übermittelt diese an den Web–Service, der die Nachricht entschlüsselt

und von dem XKMS Dienst den öffentlichen Schlüssel des Clients erfragt, der zur Überprüfung der signierten Teile der Nachricht dient.

Verbindlichkeit Die Spezifikation schreibt vor, alle Nachrichten mit Zeitstempeln (WS-Security) zu versehen, um die Verbindlichkeit bei der Kommunikation zu gewährleisten. Desweiteren sind identische Nachrichten (Replay Attacke) zu erkennen und abzulehnen. Die Speicherung aller empfangener Nachrichten zur Entlastung des jeweiligen Händlers (Kommunikationsnachweis) hat auf unbestimmte Zeit zu erfolgen. Nachrichten die einen Zeitstempel aufweisen der älter als 10 Minuten ist, sind abzulehnen.

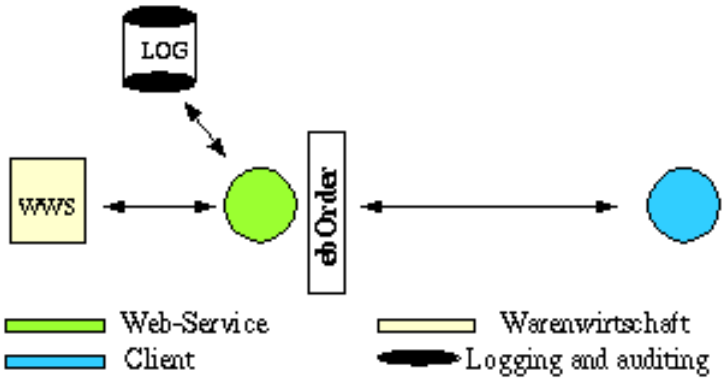

Abbildung 32: Verbindlichkeit

Verfügbarkeit Um die Verfügbarkeit eines Dienstes sicherzustellen, müssen Regeln bezüglich der Nutzung des Dienstes erstellt werden. Diese klären

- wer
- zu welchem Zeitpunkt
- wie viele Anfragen
- an welchen Dienst

stellen darf. Werden diese Regeln gebrochen handelt es sich um einen Angriff auf die Verfügbarkeit des Systems und entsprechende Gegenmaßnahmen

müssen ergriffen werden (z.B. Ressourcen schonende Sperrung eines Clients auf Paket–Ebene). Die Erkennung und Reaktion auf derartige Angriffe ist eine typische Aufgabe einer XML–Firewall.

Natürliche Lastspitzen können die Verfügbarkeit des Systems (Warenwirtschaft, Web–Service) ebenfalls gefährden.

7.2.4 Der UDDI Dienst

Der UDDI Dienst stellt Informationen über EasyBuy und die publizierten Dienste des Unternehmens zur Verfügung. Alle publizierten Daten von EasyBuy müssen signiert werden, um die Integrität der Informationen sicherzustellen. Wärend die Integration einer XML Signature in einer WSDL Datei immer möglich ist, können Unternehmensdaten nur signiert werden, wenn der Verzeichnisdienst den UDDI Standard ab der Version 3.0 unterstützt.

7.2.5 Der Client

Sicherheitsmechanismen, die den Client betreffen, müssen bereits von dem Web–Service gefordert werden. Dem Client weitere Sicherheitsregeln aufzuerlegen ist nicht möglich, weil es sich bei einem Client auch immer um einen potentiellen Angreifer handelt, der sich an solche Reglen nicht halten würde.

Um an dieser Stelle auch auf die Sicherheitsbedürfnisse des Clients einzugehen, sei erwähnt, dass diesem die Einsicht einiger Daten durch EasyBuy mißfallen könnte. Aus Sicht des Clients ist es z.B. nicht notwendig, dass EasyBuy die Kreditkartennummer lesen kann. Web–Services Technologien ermöglichen es z.B. die Kreditkartennummer bereits von dem Client für den Payment Provider verschlüsseln zu lassen. EasyBuy würde die entsprechenden Nachrichtenteile dann an den Payment Provider weiterleiten, ohne in der Lage zu sein, diese zu lesen. Dies ist jedoch in dem gegebenen Szenario nicht vorgesehen.

7.2.6 Der Payment Provider

In dem gegebenen Szenario ist davon auszugehen, dass der Payment Provider den Einsatz angemessener Sicherheitmechanismen vorschreibt.

7.3 Zentrales Sicherheitsmanagement

Der Einsatz konzeptioneller Sicherheit, ist in Web–Services Umgebungen an Nachrichten gebunden. Ist eine solche mit Sicherheitsmerkmalen (z.B. Verschlüsselung, Signatur, o.ä.) ausgestattet, müssen diese von dem Web–Service erkannt und verarbeitet werden können. Dieser dezentrale Ansatz macht die Verwaltung der Sicherheit mit steigender Anzahl verwendeter Web–Services schwierig. Die Kosten für die Programmierung der Sicherheit steigen linear zu der Anzahl neu erstellter Web–Services und eine nachträgliche Skalierung der Sicherheit erweist sich als schwierige Aufgabe.

Abhilfe schaffen XML–Firewalls, die im Gegensatz zu den üblichen Paket–Firewalls, ein zentrales Sicherheitsmanagementsystem darstellen. Konzeptionelle Sicherheit wird von diesen an zentraler Stelle implementiert und verwaltet (siehe Abbildung 33). Dies senkt Implementierungskosten, vereinfacht die Skalierung von Sicherheit erheblich und ermöglicht es dem Web–Service sich auf die Implementierung der eigentlichen Funktionalität zu konzentrieren. Eine intensive Betrachtung dieser Systeme kann im Rahmen dieser Arbeit nicht erfolgen, die wesentlichen Vor- und Nachteile sollen jedoch stichpunktartig aufgezählt werden.

Vorteile:

- Zentrale Verwaltung der Sicherheit.
- Programmierung zur Implementierung konzeptioneller Sicherheit entfällt.
- Neue Standards können leicht in bestehende Web–Services Landschaften implementiert werden.
- Protokollierung der Kommunikation (logging)
- Erkennung von Angriffen und Abwehr dieser.
- Vereinfacht den Einsatz von Sicherheit in Web–Services Szenarien.

Nachteile:

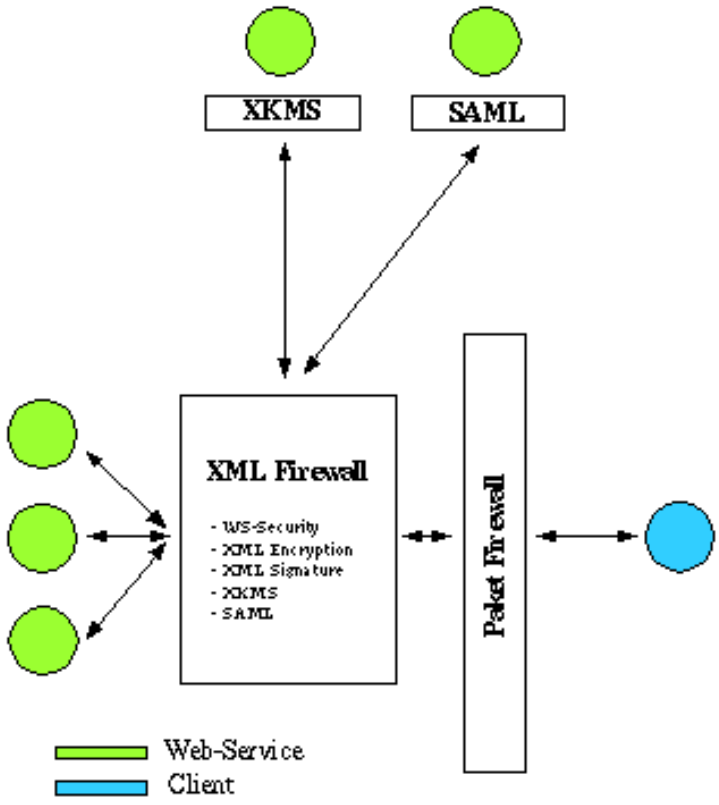

Abbildung 33: Einsatz einer XML Firewall

- Sind nicht standardisiert. Es muss bei der Anschaffung darauf geachtet werden, dass die jetzt und evtl. zukünftig benötigten Standards oder Algorithmen unterstützt werden.
- Werden oft von jungen Unternehmen angeboten (höheres Konkursrisiko).

Der Einsatz von XML–Firewalls in Web–Services Szenarien scheint aufgrund der gravierenden Vorteile zwingend. [Gralla 02] gibt Unternemen diesbezüglich folgenden Rat: „Even if you don't need an XML firewall now, you will at some time in the future, so you'd do well to start investigating them today."

7.4 Das abgesicherte Szenario

Abbildung 34 führt das abgesicherte Szenario ein. Dabei wurde davon ausgegangen, dass EasyBuy seine Unternehmensdaten und die WSDL Schnittstellenbeschreibung bereits unter der Verwendung von Signaturen in einer UDDI publiziert hat. Desweitern haben sowohl EasyBuy, als auch der Client, Schlüsselmaterial bei dem XKMS Service hinterlegt, und der Client darüber hinaus eine Registrierung bei dem SAML Dienst vollzogen. EasyBuy hat sich auf den Einsatz einer XML–Firewall festgelegt und nutzt die in der sicherheitskritischen Betrachtung gesammelten Informationen um diese zu konfigurieren. Der Aufruf des ebOrder Web–Service von EasyBuy durch einen Client würde in dem dargestellten Szenario in etwa wie folgt aussehen:

1. Der Client fordert von der SAML Instanz die Autorisation zur Nutzung von ebOrder Diensten an und erhält eine dazu berechtigende Assertion.
2. Anschließend sucht der Client in einem UDDI Verzeichnis nach ebOrder Web–Services und findet unter anderem den Dienst von EasyBuy.
3. Anhand der aus der UDDI Anfrage bekannten Unternehmensdaten, fordert der Client von der XKMS Instanz den öffentlichen Schlüssel von EasyBuy an. Mit diesem können Nachrichten für den Dienst verschlüsselt und von dem Unternehmen ausgestellte Signaturen überprüft werden.

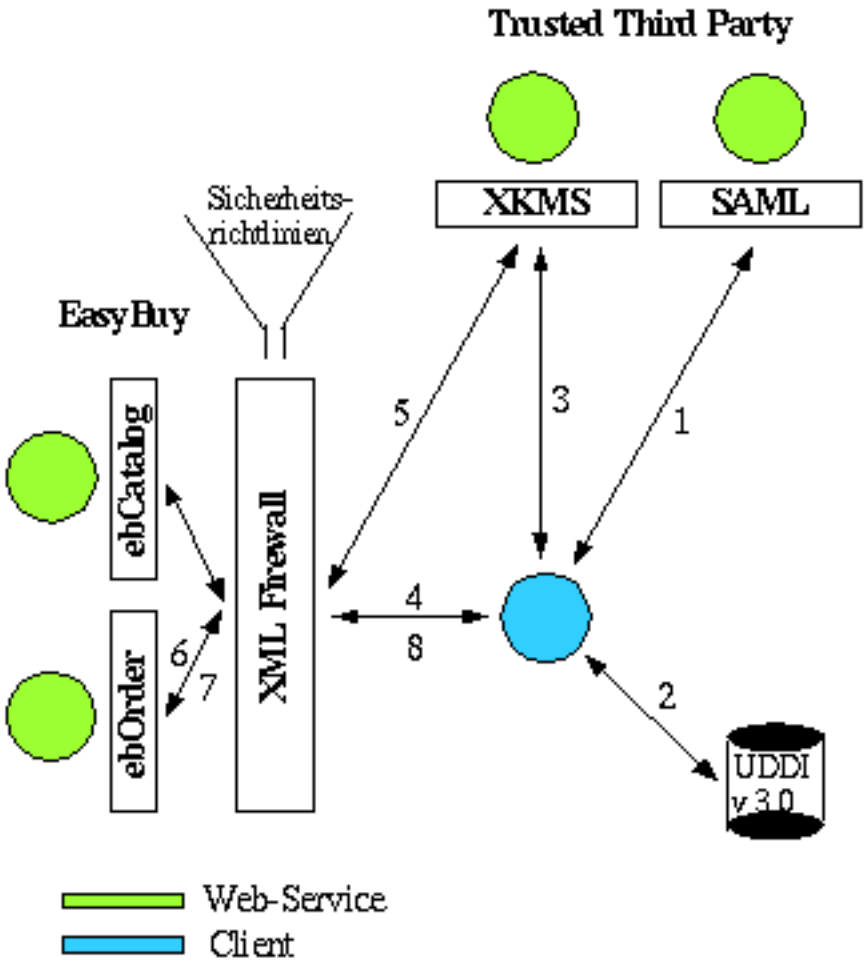

Abbildung 34: Abgesichertes Szenario

4. Der Client erstellt eine ebOrder Anfrage für den Dienst von EasyBuy und nutzt dabei das entsprechende Schlüsselmaterial um die verlangten Sicherheitsmerkmale (Signatur, Verschlüsselung) in die Nachricht zu integrieren. Im Anschluss daran, wird die Nachricht an den Dienst geschickt.

5. Die XML–Firewall fängt die Nachricht ab, entschlüsselt die entsprechenden Teile der Nachricht und prüft anhand der Assertion ob der Client zur Nutzung des Dienstes berechtigt ist. Ist dies der Fall, schickt sie das in der Assertion mitgelieferte Zertifikat des Clients an die XKMS Instanz, um es prüfen und bei Gültigkeit den öffentlichen Schlüssel daraus extrahieren zu lassen. Mit dem öffentlichen Schlüssel des Clients werden im Anschluß die Signaturen der Nachricht überprüft. Außerdem werden die Filterregeln zur Erkennung bösartiger Daten jetzt angewandt.

6. Die XML Firewall schickt die von Sicherheitsmerkmalen bereinigte Nachricht zur Verarbeitung an den ebOrder Web–Service.

7. Dieser führt die Verarbeitung aus und schickt die Ergebnisse zurück an den Client.

8. Auch diese Nachricht wird von der XML Firewall abgefangen. Sie stattet die Nachricht mit den benötigten Sicherheitsmerkmalen aus und leitet diese dann an den Client weiter.

Die XML–Firewall fungiert in dem Szenario als ein Proxie, den jede Nachricht von oder an firmeninterne Web–Services durchläuft. Durch eine entsprechende Konfiguration ist dieser bekannt, welche Sicherheitsmerkmale in den entsprechenden Nachrichten zu verarbeiten sind.

8 Fazit

Die Standardisierung gundlegender Mechanismen um Web–Services Szenarien mit Sicherheitsmerkmalen wie Authentifikation, Autorisation, Vertraulichkeit, Integrität und Verbindlichkeit auszustatten sind weitestgehend abgeschlossen. XML Signature, XML Encryption, WS-Security, XKMS und SAML wurden mit Hinblick auf die architektonischen Besonderheiten von Web–Services entwickelt und bieten so einen Schutz, der an die Merkmale von verteilten Umgebungen und XML als Nachrichtenformat angepasst ist.

Die Betrachtung der Sicherheitsrisiken macht allerdings klar, dass der Einsatz konzeptioneller Sicherheit nicht zu umfassender Systemsicherheit führt. Zu viele Risiken sind durch diese alleine nicht auszuräumen. Insbesondere der hohe Rechenaufwand bei der Prüfung mit konzeptioneller Sicherheit ausgestatteter XML Nachrichten und die Möglichkeit Legacy Applications durch bösartige Daten zu einem Fehlverhalten zu bewegen, müssen berücksichtigt werden. Um dem entgegen zu wirken, muss vor der Realisierung eines Web–Services Projekts, eine umfassende Analyse möglicher Schwachstellen aller beteiligten Systeme erfolgen.

Eine wichtige Erkenntnis dieser Arbeit ist die Notwendigkeit für den Einsatz von XML–Firewalls. Diese bieten weit mehr an Funktionalität als man dies von bestehenden Firewalls gewohnt ist. Dabei geht es längst nicht nur um das Erkennen und Blockieren bösartiger Nachrichten. Sie zentralisieren konzeptionelle Sicherheit und verlagern diese so aus den Web–Services heraus in eine administrierbare Umgebung. Sicherheitsrichtlinien werden von dieser umgesetzt und bösartige Angriffe erkannt, protokolliert und abgewehrt. Der zentrale Ansatz nimmt dem Einsatz konzeptioneller Sicherheit die Komplexität und trägt so erheblich zur Vereinfachung von Sicherheit in Web–Services Umgebungen bei. Auf den Einsatz eines solchen Systems werden Unternehmen bei der Arbeit mit Web–Services nicht verzichten können.

Bei der Auswahl einer XML–Firewall ist Vorsicht geboten, da es keine allgemeingültige Leistungsbeschreibung derartiger Systeme gibt. Ob die hausin-

tern eingesetzten Standards oder Verschlüsselungsalgorithmen unterstützt werden, muss also von Fall zu Fall geprüft werden. Auf die Möglichkeit zur Integration XML Schema erweiternder Datenprüfungen, wie sie in dieser Arbeit für Notwendig gehalten werden, sollte ebenfalls geachtet werden. Ein Aspekt der innerhalb dieser Arbeit nicht intensiv beleuchtet wurde, ist die relativ hohe Rechenleistung bei der Verarbeitung XML basierter Nachrichten und dem Einsatz konzeptioneller Sicherheit innerhalb dieser. Eine XML–Firewall muss in der Lage sein, Aufgaben über so genannte „load Balancer“ zu verteilen, um die Umgebung einer wachsenden Anzahl von Zugriffen anzupassen und dadurch die Verfügbarkeit zu gewährleisten. Die Unterstützung von Kryptohardware ist ebenfalls wünschenswert.

Wie schnell sich Web–Services bei neuen B2B, B2BI, B2C oder EAI Projekten durchsetzen werden ist noch unklar. Auf die unzureichende Untersützung von Sicherheit, die langezeit den ernsthaften Einsatz der Web–Services Technologie verhindert hat, können sich Unternehmen jedoch nicht mehr berufen.

Abbildungsverzeichnis

Abkürzungsverzeichnis

AES	Advanced Encryption Standard
ACID	Atomicity, Consistency, Isolation and Durability
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BEEP	Blocks Extensible Exchange Protocol
B2B	Business to Business
B2BI	Business to Business Integration
B2C	Business to Consumer
CPS	CheaperProducts Software
DB	Datenbank
DOM	Document Object Model
DoS	Denial of Service
DTD	Document Type Definition
EAI	Enterprise Application Integration
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
HTTPS	Hypertext Transport Protocol Secure
ID	Identifikationsmermal
IETF	Internet Engineering Task Force
IMCP	Internet Message Control Protocol
IPSec	Internet Protocol Security
MD5	Message Digest 5
MSDOS	Microsoft Disk Operating System
OASIS	Organisation for the Advancement of Structured Information Standards
PHP	PHP Hypertext Preprocessor
PKI	Public Key Infrastructure
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language

SASL	Simple Authentication and Security Layer
SGML	Standard Generalized Markup Language
SHA	Secure Hash Algorithm
SOAP	Simple Object Access Protocol
SPML	Service Provisioning Markup Language
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTP	Trusted Third Party
UDDI	Universal Description Discovery and Integration
UDP	User Datagram Protocol
URI	Uniform Ressource Identifier
URL	Uniform Ressource Locator
WSDL	Web Services Description Language
WWS	Warenwirtschaftssystem
W3C	World Wide Web Consortium
XACML	eXtensible Access Control Markup Language
XCBF	XML Common Biometric Format
XHTML	eXtensible Hypertext Markup Language
XKMS	XML Key Management Specification
XML	eXtensible Markup Language
XrML	Extensible Rights Management Language
X-KISS	XML - Key Information Service Specification
X-KRSS	XML - Key Registration Service Specification

Literatur

[AES] *Advanced Encryption Standard.* Federal Information Processing Standards Publication (FIPS) 197. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. 26 November 2001.

[Anley 02] C. Anley: *Advanced SQL Injections In SQL Server Applications.* NGSSoftware Insight Security Research (NISR) Publication. http://www.ngssoftware.com. 2002.

[Bauer] J. Bauer: *Buffer Overflow Bugs.* http://www.theo2.physik.uni-stuttgart.de/jtb/overflow/tutorial.html.

[Born 01] G. Born: *Jetzt lerne ich XML.* Markt und Technik Verlag. 2001.

[Eastlake 02] D. Eastlake, K. Niles: *Secure XML.* Addison-Wesley. 2002.

[Firewalls] www.westbridgetech.com: *XML Firewall.* http://www.westbridgetech.com/appfirewall.html.

[Fontana 02] J. Fontana: *Securing Web Services.* Network World. http://www.nwfusion.com/buzz/2002/websec.html. 23 September 2002.

[Gralla 02] P. Gralla: *XML Firewalls.* The Web Services Advisor. http://searchwebservices.techtarget.com/tip/1,289483,sid26_gci855052,00.html. Oktober 2002.

[Graham 02] S. Graham, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, R. Neyama, S. Simeonov: *Building Web Services with Java: SOAP.* http://www.informit.com. 10 Mai 2002.

[Goldfarb 99] C. F. Goldfarb, P. Prescod: *XML Handbuch.* Markt und Technik Verlag. 1999.

[Newcomer 02] E. Newcomer: *Understanding Web Services.* Addison Wesley. 2002.

[Rescorla 00] E. Rescorla: *SSL and TLS.* Addison Wesley. 2000.

[RFC 2396] T. Berners-Lee, R. Fielding, L. Masinter: *Uniform Ressource Identifiers (URI): Generic Syntax.* IETF. 1998.

[Russel 01] R. Russel, S. Cunningham: *Maximum Protection.* MITP Verlag. 2001.

[SAML] P. Hallam-Baker, E. Maler: *Security Assertion Markup Language (SAML). OASIS Committee Specification.* http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security. 31. Mai 2002.

[SAML-SEC] J. Hodges, C. McLaren, P. Mishra, B. Morgan, T. Moses, E. Prodromou, M. Erdos: *Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML).* http://www.oasis-open.org/committees/security/ docs/draft-sstc-sec-consider-04.pdf. 15. Januar 2002.

[SCHEMA-2] P. V. Biron, A. Malhotra: *XML Schema Part 2: Datatypes.* W3C Recommendation. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/. 02 Mai 2001.

[Schneier 00] B. Schneier: *Secrets & Lies.* Wiley VCH–Verlag. 2000.

[Schneier 96] B. Schneier: *Angewandte Kryptographie.* Addison-Wesley. 1996.

[Seely 02] S. Seely: *Understanding WS-Security.* Microsoft Corporation. http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/understw.asp. Oktober 2002.

[Siyan 95] K. Siyan, C. Hare: *Internet Firewalls and Network Security.* New Riders Publishing. 1995.

[SOAP] M. Gudgin, M. Hadley, N. Mendelsohn, J.Moreau, H. F. Nielsen: *SOAP Version 1.2 Part 1: Messaging Framework* W3C Candidate Recommendation. http://www.w3.org/TR/2002/CR-soap12-part1-20021219. 19 Dezember 2002.

[SQL-SEC] sqlsecurity.com: *SQLSecurity Checklist* http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=3&tabid=4. Letzter Zugriff: 27.03.2003.

[UDDI] T. Bellwood, L. Clément, D. Ethnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, C. Riegen: *Universal Description, Discovery and Integration Version 3.0.* http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm. 19 Juli 2002.

[WSDL] R. Chinninci, M. Gudgin, J.J. Moreau, S. Weeravarana: *Web Services Description Language Version 1.2.* W3C Working Draft. http://www.w3.org/TR/2003/WD-wsdl12-20030124. 24 Januar 2003.

[WS-ARCH] M. Champion, C. Ferris, E. Newcomer, D.Orchard: *Web Services Architecture.* W3C Working Draft. http://www.w3.org/TR/2002/WD-ws-arch/20021114/. 14 November 2002.

[WS-SEC] P. H. Baker, C. Kaler, R. Monzillo, A. Nadalin: *Web Services Security Core Specification.* OASIS Working Draft. http://www.oasis-open.org/committees/wss/documents/WSS-Core-09-0126.pdf. 26 Januar 2003.

[XKMS] P. Hallam-Baker: *XML Key Management Specification (XKMS 2.0).* W3C Working Draft. http://www.w3.org/TR/2002/WD-xkms2-20020318/. 18 März 2002.

[XML] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler: *Extensible Markup Language (XML) 1.0 (Second Edition).* W3C Recommendation. http://www.w3.org/TR/REC-xml. 6 Oktober 2000.

[XML-ENC] D. Eastlake, J. Reagle, T. Imamura, B. Dillaway, E. Simon: *XML Encryption Syntax and Processing.* W3C Recommendation. http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/. 10 Dezember 2002.

[XML-SIG] D. Eastlake, J. Reagle, D. Solo, M. Bartel, J. Boyer, B. Fox, B. LaMacchia, E. Simon: *XML-Signature Syntax and Processing.* W3C Recommendation. http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/. 12 Februar 2002.

[Yang 02] A. Yang: *Web Services Security.* eAI Journal. September 2002.