



Xpert.press

Walter Kriha
Roland Schmitz

Internet-Security aus Software-Sicht

 Springer

Xpert.press

Die Reihe **Xpert.press** vermittelt Professionals
in den Bereichen Softwareentwicklung,
Internettechnologie und IT-Management aktuell
und kompetent relevantes Fachwissen über
Technologien und Produkte zur Entwicklung
und Anwendung moderner Informationstechnologien.

Walter Kriha · Roland Schmitz

Internet-Security aus Software-Sicht

Grundlagen der Software-Erstellung
für sicherheitskritische Bereiche

Walter Kriha
Roland Schmitz
Hochschule der Medien
Nobelstr. 10
70569 Stuttgart
schmitz@hdm-stuttgart.de
www.medieninformatik.hdm-stuttgart.de

ISBN 978-3-540-22223-1 e-ISBN 978-3-54068906-5

DOI 10.1007/978-3-54068906-5

ISSN 1439-5428

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

© 2008 Springer-Verlag Berlin Heidelberg

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland vom 9. September 1965 in der jeweils geltenden Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zuwiderhandlungen unterliegen den Strafbestimmungen des Urheberrechtsgesetzes.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften. Text und Abbildungen wurden mit größter Sorgfalt erarbeitet. Verlag und Autor können jedoch für eventuell verbreitete fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Einbandgestaltung: KünkelLopka Werbeagentur, Heidelberg

Gedruckt auf säurefreiem Papier

9 8 7 6 5 4 3 2 1

springer.com

Danksagung

Dieses Buch wäre nicht möglich gewesen ohne die Hilfe und die Unterstützung vieler Personen. An erster Stelle sind hier unsere Familien zu nennen, die ein großes Maß an Verständnis während der langen Entstehungszeit dieses Buches und des Folgebands aufbringen mussten: Elfi, Beate und Kerstin Kriha, Jutta und Paul Schmitz.

Unsere Freunde und Kollegen aus dem Bereich der IT-Security wurden nicht müde, mit uns wichtige Kernideen der Security zu diskutieren. Sie haben uns die vielleicht wichtigste Idee der Security vermittelt, nämlich bei Unklarheiten ohne Scheu so lange nachzufragen, bis man das Problem oder die Lösung verstanden hat.

Unser Verlag hat einen erstaunlich langen Atem bewiesen, obwohl wir Deadline nach Deadline platzen ließen und aus einem geplanten Buch letztlich zwei wurden. Für dieses Verständnis und die gewährte Unterstützung ein herzliches Dankeschön an unsere Lektoren beim Springer-Verlag.

Last but not least danken wir unseren Studenten und Kollegen im Studiengang Medieninformatik an der Hochschule der Medien Stuttgart für die vielen anregenden Diskussionen zum Thema Security. Wir haben enorm viel daraus gelernt.

Stuttgart,
im Oktober 2007

*Walter Kriha
und Roland Schmitz*

Inhaltsverzeichnis

1	Einführung und Motivation	1
2	Fallstudien	7
2.1	Online-Kartenkauf am Beispiel bahn.de.....	9
2.1.1	Überblick.....	9
2.1.2	Geschäftsmodell.....	11
2.1.3	Kundensicht	12
2.1.4	Geschäftsvorgänge	13
2.1.5	Sicherheitsanforderungen.....	14
2.1.6	Testkonzept	24
2.2	ebay	24
2.2.1	Übersicht und Geschäftsmodell	25
2.2.2	Kundensicht	26
2.2.3	Spezielle Sicherheitsanforderungen	29
3	Sicherheitskonzepte und Analysen	33
3.1	Security Policies	33
3.2	Allgemeine Vorgehensweise	34
3.2.1	Risikoanalyse der Geschäftsvorgänge.....	35
3.2.2	Security Context.....	38
3.2.3	Basisarchitektur.....	40
3.2.4	Bedrohungsmodelle	42
3.3	Bedrohungsmodelle für bahn.de	44
3.3.1	Sicherheit auf Clientseite	45
3.3.2	Die Verantwortung des Servers	49
3.3.3	Kommunikation mit dem Kunden.....	50
3.4	Beispiel einer Sicherheitsanalyse im Embedded Control Bereich	52
3.4.1	Ausgangsszenario.....	53
3.4.2	Analyse des Ausgangsszenarios.....	55
3.4.3	Modifiziertes Szenario	59
3.5	„The People Problem“.....	61

4	Sicherheitsdienste	65
4.1	Authentifikation.....	65
4.1.1	Authentifikation durch Passwörter.....	66
4.1.2	Authentifikation durch Challenge-Response-Verfahren	68
4.1.3	Kontext-Weitergabe, Delegation und Impersonation.....	69
4.2	Vertraulichkeit.....	70
4.3	Integritätsschutz.....	70
4.4	Nicht-Abstrebbarkeit.....	71
4.5	Verfügbarkeit.....	71
4.6	Authorisierung.....	72
4.6.1	DAC – Discretionary Access Control.....	72
4.6.2	MAC – Mandatory Access Control.....	73
4.6.3	RBAC – Role Based Access Control.....	74
4.6.4	Multi-Level Security	75
5	Kryptografische Algorithmen.....	77
5.1	Allgemeines.....	77
5.2	Symmetrische Verschlüsselungsalgorithmen	79
5.2.1	DES und AES	79
5.2.2	Betriebsmodi für Blockchiffren	80
5.3	Hashfunktionen.....	81
5.3.1	Kollisionen.....	81
5.3.2	Schlüsselabhängige Hashfunktionen.....	83
5.3.3	Password-Based Encryption (PBE).....	84
5.4	Asymmetrische Algorithmen.....	85
5.4.1	RSA-Verfahren	85
5.4.2	Diffie-Hellman-Protokoll und diskrete Logarithmen.....	86
5.4.3	Digitale Signaturen	88
5.4.4	Performance-Fragen.....	89
5.5	Zertifikate	89
5.5.1	Zertifikate nach X.509	90
5.5.2	Attributzertifikate.....	92
5.5.3	Zurückziehen von Zertifikaten	93
5.5.4	Certificate Revocation List (CRL)	94
5.5.5	Online Certificate Status Protocol (OCSP)	94
5.6	Authentifikationsprotokolle nach X.509.....	95
5.6.1	One-Pass Authentication.....	95
5.6.2	Three-Pass Authentication	96
5.6.3	Three Pass Mutual Authentication	97
5.7	Zufallswerte	98
5.7.1	Schlüsselerzeugung	98
5.7.2	Challenges.....	99
5.7.3	SessionIDs.....	99

5.8	Kryptografie mit Java	99
5.8.1	Java Cryptography Architecture (JCA).....	100
5.8.2	Symmetrische Verschlüsselung	101
5.8.3	Hashfunktionen und MACs.....	102
5.8.4	Asymmetrische Kryptografie	102
5.8.5	Zertifikate.....	103
5.8.6	Erzeugung von Zufallszahlen.....	103
5.9	Übersicht	104
6	Sicherheit in Verteilten Systemen.....	105
6.1	Lokale versus verteilte Sicherheit.....	106
6.2	Authentisierung in verteilten Systemen.....	107
6.2.1	Authentisierung versus Identifizierung	107
6.2.2	Authentisierung mit Passwörtern	109
6.2.3	Software-Architektur der Authentisierung mit Passwörtern.....	115
6.3	Delegation und Impersonation.....	120
6.3.1	Begriffsklärung	120
6.3.2	Delegation von Aufträgen.....	123
7	Basisprotokolle	127
7.1	http Authentication	127
7.1.1	Basic Authentication	128
7.1.2	Digest Authentication	128
7.2	Kerberos	129
7.2.1	Funktionsweise	129
7.2.2	Principals und Domänen	132
7.2.3	Attacken auf Kerberos	133
7.2.4	Delegation mit Kerberos	134
7.2.5	Cross-Domain-Authentisierung	136
7.2.6	Kerberos Erweiterungen	137
7.2.7	Microsoft Passport	138
7.3	SSL/TLS	139
7.3.1	Aufbau von SSL.....	140
7.3.2	Handshake.....	142
7.3.3	Ciphersuites.....	145
7.3.4	Performanzfragen.....	148
7.3.5	SSL Security	150
7.3.6	Zusammenfassung.....	152
8	Authentication Frameworks	155
8.1	GSS-API.....	155
8.1.1	Überblick.....	156
8.1.2	Ein Beispiel	157

8.1.3	GSS-API Mechanismen	158
8.1.4	Simple and Protected Negotiation Mechanism (SPNEGO)	159
8.1.5	Delegation in GSS-API	160
8.2	SASL	161
8.2.1	Funktionsweise	162
8.2.2	SASL Mechanismen	162
8.3	Zusammenfassung und Bewertung	163
9	Middleware Security	165
9.1	CORBA	165
9.1.1	SECIOP und SSLIOP	166
9.1.2	CSIV2 und SAS	167
9.2	Remote Method Invocation (RMI)	170
9.3	.NET	171
9.3.1	Allgemeines	171
9.3.2	Code Access Security (CAS)	172
9.3.3	Role Based Security (RAS)	174
9.4	Simple Object Access Protocol (SOAP)	175
9.4.1	Allgemeines	175
9.4.2	SOAP Security	176
10	Content-Level Security	179
10.1	Aktuelle Trends	180
10.2	Architektur und Infrastruktur	181
10.2.1	Infrastruktur	182
10.2.2	CMS Sicherheitsarchitektur	185
10.2.3	Abbildung der Berechtigungen auf das Sicherheitsmodell der Firma	188
10.2.4	Rollenmodellierung am Beispiel Portal Access Control	192
10.2.5	Benutzer-Berechtigungs-Systeme (BBS)	196
10.2.6	Geschäftsprozesse: Workflow und Realität	203
10.2.7	Zugriffskontrolle beim Endnutzer	207
10.3	Spezielle Probleme von Content Security	210
10.3.1	Records Management	210
10.3.2	Mobile Dokumente	211
10.3.3	Multi-Level Security (MLS)	213
10.3.4	Enterprise Search Engine Security	216
11	Sicherheit der Infrastruktur	225
11.1	Absicherung der Infrastruktur durch Firewalls und DMZs	226
11.1.1	Firewall-Architekturen	227
11.1.2	Reverse Proxy als Point-of-Contact	231
11.1.3	Beispiel Nevis Web	234

11.1.4	Gegenseitige Authentisierung von Komponenten und Knoten.....	238
11.1.5	Applikationsdesign für zentrale Firewall-Umgebungen	241
11.1.6	Sessionkonzept.....	242
11.2	Verkleinerung der Angriffsfläche.....	248
11.2.1	Maßnahmen.....	248
11.2.2	Ein Beispiel aus der Praxis.....	253
11.3	Single-Sign-On für Portale	261
11.3.1	Vorstellung einer Portallösung mit Weitergabe der Identität.....	261
11.3.2	Aufgaben der Autoritäten.....	263
11.3.3	Heuristiken für Softwareentwickler	265
11.3.4	Das Firmenportal	266
11.3.5	Integration vorhandener Legacy Systeme	267
11.3.6	Ausbau des Security Contexts.....	270
11.3.7	Step-Up Authentication.....	270
11.3.8	Sicherheitsanmerkungen zu Single-Sign-On.....	270
11.4	Mobile Infrastruktur	271
12	Föderative Sicherheit.....	275
12.1	Föderatives Identitätsmanagement	276
12.2	Föderatives Trust Management	278
12.3	Föderative Sicherheit am Beispiel eines Portals.....	281
12.3.1	Physische Architektur	282
12.3.2	Einfacher föderativer Web-SSO zwischen Domänen	283
12.4	Standards für föderatives Identitätsmanagement	290
12.4.1	SAML	291
12.4.2	Liberty Alliance	293
12.4.3	Web Services Federation	294
12.5	Sicherheitsanalyse	296
13	Schlussbetrachtungen	301
Abkürzungsverzeichnis	303	
Literaturverzeichnis	305	
Index	309	

Kapitel 1

Einführung und Motivation

Zwei Großkonzerne erstellten im Jahre 2004 zusammen ein Kundenportal, mit dem Kunden ihre eigenen Stammdaten und die bestellten Services bequem verwalteten konnten. Allerdings gestattete das Portal den Kunden auch die Verwaltung der Daten *anderer* Kunden – durch einfachste Manipulationen an den URLs der herunter geladenen Seiten. Gewieftere Angreifer konnten sich mit wenig Aufwand Administratorrechte verschaffen, und zwar im gesamten internen Netzwerk des Konzerns (Details findet man bei [CCC]). Am Ende musste das Portal für gut zwei Monate abgeschaltet und neu implementiert werden. Die Rede ist vom Online Business Solution Operation Center (OBSOC), dem Portal der Deutschen Telekom zur Verwaltung von Kundendaten. Wie kann es sein, dass wichtige, von großen Konzernen entwickelte Systeme so unsicher sind?

Im Juli 2005 werden von einem amerikanischen Verarbeiter von Visa- und MasterCard-Transaktionen 40 Millionen Kundendaten entwendet – teilweise sogar mit PIN-Nummer. Wie kann es sein, dass auf den Desktops der Mitarbeiter dieser Firma hochbrisante Kundendaten herumliegen und gestohlen werden können? Attacken durch Buffer Overflows, Viren und Trojanische Pferde bedrohen nach wie vor die meisten Maschinen und Benutzer weltweit und verursachen enorme Schäden. Wie kann es sein, dass heutige Betriebssysteme und ihre Besitzer so schnell an Eindringlinge verraten, statt ihnen Hilfestellungen beim Erkennen von Malware zu geben – oder noch besser: das Gefahrenpotential der Malware zu verringern? Warum führt die kleinste Lücke in Applikationen und Servern schon dazu, dass alles verloren ist?

In der Vergangenheit haben sich die Bemühungen der IT-Sicherheitsexperten im Zuge der Erfolgsstory des Internets vor allem auf die Netzwerksicherheit konzentriert. Zwar sind auch auf diesem Gebiet noch längst nicht alle Probleme gelöst – wir verfügen jedoch zumindest über eine Reihe von leistungsfähigen kryptografischen Modulen und Sicherheitsprotokollen, die, wenn richtig implementiert, eine zuverlässige Absicherung der Kommunikation zwischen zwei Rechnern bieten.

Die oben genannten Beispiele zeigen jedoch deutlich, dass nicht die Kommunikation zwischen Client und Server, sondern die *Software* auf Client- und Serverseite das Hauptproblem ist. Unzureichende Validierung der Eingaben der Clients führt dazu, dass böswillige Nutzer sich Rechte auf dem Server verschaffen können,

die ihnen nicht zustehen. Auf der anderen Seite sind die PCs legitimer Nutzer durch Malware gefährdet, die ihre persönlichen Daten ausspäht. Aber wie ist das möglich?

Wieso wird so viel unsichere Software entwickelt? Diese Frage wurde in der Vergangenheit schon in ganz verschiedener Weise beantwortet. Von der mangelnden Haftung der Softwarehersteller ist die Rede sowie vom Zeitdruck und unmöglichen Deadlines bei der Entwicklung. All dies spielt gewiss eine Rolle, aber es ist aus unserer Sicht nicht entscheidend. Im vorliegenden Buch vertreten wir die These, dass die folgenden Gründe für die Existenz der meisten unsicheren Software verantwortlich sind:

- An erster Stelle steht die mangelnde Wahrnehmung von Sicherheitsproblemen auf Seiten der Softwareentwickler. Der Mangel kann dabei auf ganz verschiedenen Ebenen existieren und reicht von Missverständnissen bezüglich der beteiligten Personen und Geschäftsvorgänge bis hin zu einem grundsätzlichen Missverständnis der Kommunikation in verteilten Systemen.
- Hinzu kommt mangelndes Wissen der Entwickler über Sicherheitstechniken und Sicherheitsarchitekturen, vor allem über das Zusammenspiel von Software, Infrastruktur und Benutzern. Verstärkt wird dieses Problem durch ein begriffliches Chaos auf Grund konkurrierender Standards und Techniken gerade im Sicherheitsbereich.
- An dritter Stelle steht ein Mangel an konkreten Mustern (Patterns) zur Lösung von Sicherheitsproblemen.
- An vierter Stelle – jedoch keineswegs weniger wichtig als die anderen – stehen die sicherheitsrelevanten Eigenschaften aktueller Betriebssysteme und Programmiersprachen, die meist gegen bekannte Prinzipien der sicheren Software-Entwicklung verstößen (Fail Gracefully, Principle-of-least-Authority (POLA)).

Komplexe Frameworks, wie sie im Falle von OBSOC zum Einsatz kamen, beinhalten umfangreiche Hilfestellungen für die Erstellung von Webapplikationen. Sicherer Code wird hier zu einem guten Teil durch das Framework möglich gemacht. Nur müssen die Entwickler die Problemfelder zumindest erkennen und außerdem die Mechanismen des Frameworks verstehen und einsetzen können.

Der gegenwärtige Trend zu immer mehr Software im „embedded control“-Bereich mit Anwendungen zum Beispiel in der Haustechnik oder der Automobilindustrie wird das Sicherheitsproblem noch verschärfen. Was passiert, wenn die momentane Art der Softwareentwicklung auf diese Systeme übertragen wird? Welche Qualitäts- und Sicherheitsprobleme gefährden die Systeme und ihre Nutzer dann? In diesem Bereich tritt die Zuverlässigkeit als Teilapekt sicherer Software stärker in den Vordergrund als bei Internet-Applikationen. Wo sind hier die Systeme, die Software verschiedenster Hersteller nebeneinander laufen lassen können, ohne dass sich die Komponenten durch die gemeinsame Benutzung von CPU, Speicher oder Geräten gegenseitig beeinflussen könnten? Virtuelles Memory vermag zwar die gröbsten Übergriffe zu verhindern, spielt jedoch innerhalb moderner Application Server und ihrer Virtual Machines eine immer geringere Rolle. Und die ausgeklügelten Virtualisierungstechniken von Mainframes, die auf

Hardware- und Softwareebene eine gute Isolierung garantieren, sind leider bei den kleinen Systemen nicht in dieser Art verfügbar.

In der letzten Zeit hat sich mit dem Spannungsfeld Usability versus Security ein weiterer wichtiger Aspekt sicherer Software bemerkbar gemacht. Angesichts der großen Zahl an semantischen Attacken wie Phishing oder Identity Spoofing, die die Benutzer momentan gefährden, zeigt sich, dass Software nicht mehr nur theoretische Sicherheit gewährleisten, sondern für den Nutzer auch verständlich und beherrschbar bleiben muss. Die Forderungen nach besserer Usability dürfen dabei nicht nur für Endbenutzer gelten, sondern müssen auf die sicherheitsrelevanten Werkzeuge der Entwickler selbst ausgedehnt werden.

Auf eine fundamentale Besserung der Sicherheit von Software zu hoffen, ist zumindest für die nächsten Jahre nicht angebracht. Die Rede von Bill Gates bei der RSA Konferenz 2005 ([Gates]) hat deutlich gemacht, dass sich die Maßnahmen von Microsoft zur Sicherung der Heimrechner auf den Aufbau einer verteilten Infrastruktur konzentrieren – sozusagen der Nachbau der Verwaltungsstrukturen, wie sie in großen Firmen heutzutage üblich sind. Große Data Center überwachen die einzelnen Rechner, erkennen Attacken und installieren Patches automatisch – gegen Gebühr.

Man darf aber nicht verkennen, dass dies eine Maßnahme nach bekannt gewordenen Attacken darstellt, also gegen so genannte Zero-Day Attacken nicht helfen wird. Wer sich nun von Open-Source-Software grundlegend Besserung erhofft, wird enttäuscht: Auch Linux unterscheidet sich hinsichtlich der Sicherheitsarchitektur keineswegs so deutlich von einem Windows basiertem System, dass hier Wunder zu erwarten wären. Die gleiche Beobachtung lässt sich im Vergleich der Open-Source Web-Browser Mozilla bzw. Firefox und dem Microsoft Internet Explorer machen.

Realistisch bleibt deshalb als Lösung auf lange Sicht nur eine auf die Bedürfnisse der Softwareentwickler zugeschnittene Ausbildung im Bereich sicherer Software. Diese Ausbildung muss:

- die Wahrnehmung von Sicherheitsproblemen massiv verbessern. Die Erfahrung hat gezeigt, dass dies am Besten durch häufige Sicherheitsanalysen geschieht, in Verbindung mit der Einführung von Sicherheitsprinzipien, die dann am konkreten Fall erprobt bzw. wieder entdeckt werden. Dieses Buch soll genau diesem Zweck dienen, nämlich Sicherheit als Gesamtsystem begreifbar zu machen und die Verbindung von Policies und Mechanismen zu erläutern.
- das Verständnis von Sicherheitsmechanismen und Techniken vermitteln. Die Erfahrung hat jedoch gezeigt, dass dies am Besten im Kontext eines realen Sicherheitsproblems geschieht und nicht davon losgelöst. Im Anschluss an eine konkrete Anwendung empfiehlt sich dann eine vertiefte Bearbeitung ausgewählter Fragestellungen. Typisches Beispiel ist hier der unterschiedliche Einsatzzweck von Kanal- bzw. Nachrichtenorientierter Kommunikation.
- sicherheitsrelevante Softwarearchitektur im Bereich Authentisierung und Authorisierung am konkreten Beispiel (z. B. Single-Sign-On im Portal) lehren.

- grundlegende Prinzipien sicherer Software wie POLA und deren technische Realisation auf den verschiedensten Ebenen (Betriebssystem, Sprache, Applikationsarchitektur) verstehen und in konkrete Softwaretechnik umsetzen helfen. Design Patterns für sichere Software helfen dabei.
- „weiche Faktoren“, wie die konzeptuellen Modelle von Benutzern, aber auch Entwicklern, als wichtig erkennen. Hier überschreitet man schnell die Grenzen der Informatik hin zu einer ganzheitlichen Betrachtung des Phänomens sichere Software. Dazu gehört beispielsweise die Kenntnis von Usability-Kriterien, aber auch die Prinzipien von Angriffen, die auf der Täuschung der Nutzer basieren, wie sie Kevin Mitnick in [Mit] eindrucksvoll dokumentiert hat.

Kernziel dieses Buches ist deshalb das Aufzeigen der Zusammenhänge zwischen Sicherheitsaspekten in der Infrastruktur, in der Software der Applikationen und Systeme sowie den Benutzern und Entwicklern dieser Systeme. Kryptografische Grundlagen und Protokolle sind für uns hauptsächlich interessant in Bezug auf die Konsequenzen, die ihre Eigenschaften für die Anwendungen nach sich ziehen.

Gleiches gilt für den Bereich der Netzwerksicherheit: Auch sie ist für uns nur interessant in Bezug auf das Zusammenspiel mit der Applikationssoftware, aber nicht als eigenständiges Gebiet. Zudem gibt es für diesen Bereich bereits etliche hervorragende Einführungen ([Bless], [Schä], [Schw]).

Die Schwierigkeiten bei der Wahrnehmung von Sicherheitsproblemen darf jedoch nicht darüber hinwegtäuschen, dass für die Erstellung sicherer Systeme eine gewisse Menge an sicherheitstechnischen Grundlagen nötig ist. Dieses Buch versucht diese Grundlagen auf eine Weise zu erklären, die es Software-Entwicklern erlaubt, sie auch in der Praxis zu erkennen und korrekt anzuwenden.

Der Aufbau des vorliegenden Buches gliedert sich wie folgt:

Zunächst wird in konkreten Fallstudien die Aufmerksamkeit für Sicherheitsprobleme geschärft und die Sicherheitsproblematik eingebettet in das Gesamtkonzept von Applikationen und Geschäftsvorgängen. Zum Beispiel untersuchen wir die Sicherheitsproblematik, die hinter großen Portalen steht und versuchen dabei zu zeigen, dass eine künstliche Einschränkung auf ein einziges Bedrohungsmodell (z. B. das Internet-Bedrohungsmodell) nicht mehr sinnvoll ist. Stattdessen muss der Gesamtkontext bestehend aus Benutzer, Nutzer-PC, Internet, Applikation, Entwickler und Infrastruktur betrachtet werden. Zur Gesamtsicht eines Sicherheitskonzeptes gehört auch seine Auswirkung auf die Betriebsorganisation (Ausbildung von Personal, Hilfe-Center bei Problemen mit der Sicherheit etc.), die meist erst sehr spät in den Blickwinkel von Entwicklern geraten und dadurch unvorhergesehene Kosten oder neue Sicherheitslücken verursachen können.

An die Fallstudien schließt sich ein Grundlagenteil an, der einzelne Aspekte aus den Fallstudien herausgreift und vertieft behandelt. Er beginnt mit einigen Anmerkungen zur Sicherheitsanalyse und ihren Komponenten, wie zum Beispiel Single-Sign-On oder Delegation. So wird die Problematik der Delegation von Requests innerhalb von Infrastrukturen erklärt sowie auf die Unterschiede von kanalbasierter und objektbasierter Sicherheit eingegangen. Die Sicherheitstechni-

ken verteilter Systeme und ihrer Middleware werden ebenfalls in diesem Grundlagenteil diskutiert. Darunter fällt etwa die Frage der Authentisierung in verteilten Systemen. Aus dem Bereich der Kryptographie werden wesentliche Bausteine zum Bau sicherer Protokolle vorgestellt, die Voraussetzung für das Verständnis der Sicherheits-Frameworks in späteren Kapiteln sind. Von besonderer Bedeutung ist dabei der Unterschied zwischen kanalbasierter Sicherheit (z. B. durch die Verwendung von SSL) und der objektbasierten Sicherheit durch die Verwendung von signierten Nachrichten. Das Verständnis objektbasierter Sicherheit ist die Voraussetzung für die Einführung von neuen, flexiblen Geschäftsmodellen durch Föderation.

Das Buch wird abgeschlossen mit drei Kapiteln, in denen wir wichtige Anwendungsfelder betrachten: Die Sicherheit von Content-orientierten Systemen, föderative Systeme und der Aufbau einer sicheren Infrastruktur für Internet-Applikationen. Rollenkonzepte, Benutzerberechtigungssysteme und die Problematik von Stellvertretung werden anhand von Content Management Systemen diskutiert. Als Sicherheitsmechanismus auf Content-Ebene wird der Einsatz von Label-basierter Security (LBAC) untersucht und rollenbasierter Sicherheit (RBAC) gegenübergestellt. Besonderes Interesse verdient LBAC nicht zuletzt deshalb, weil es eine wichtige Rolle innerhalb der User Access Control des neuen Vista Betriebssystems von Microsoft spielt. Es ergänzt hier die rein Permission-bezogene Zugriffskontrolle durch Acess Control Lists (ACLs).

Föderative Systeme beinhalten als wesentlichen Punkt die Übernahme einer bereits erfolgten Authentisierung durch andere. Wir werden sehen, dass sich dadurch nicht nur Ressourcen einsparen lassen, sondern durchaus auch eine Steigerung der Sicherheit erreichbar ist. Außerdem wird die Technik der web-basierten Föderation von Identität unter Wahrung von Privatheit gezeigt.

Im Bereich der Infrastruktur wird ein Konzept zur konsequenten Reduktion von Angriffssoberflächen innerhalb der DMZ sowie des Intranets vorgestellt. Aus der Architektur der DMZ werden Anforderungen für eine damit kompatible Applikationsarchitektur abgeleitet: Was kann/soll ein Application Server tun bezüglich Authentisierung? Nützt ein Proxy in der DMZ und welche Auswirkungen hat er für die Applikationsarchitektur? Wie weit ist die Applikationsarchitektur durch Regeln der DMZ betroffen? Die Reverse Proxy Architektur zur Authentisierung und Authorisierung wird an einem konkreten Produkt erläutert.

Dem aufmerksamen Leser ist wahrscheinlich nicht entgangen, dass die Aufzählung der Themen in diesem Band nur einen Teil der oben von uns erwähnten Problematik bei der Entwicklung sicherer Systeme umfasst. Ein zweiter Band wird sich unter dem Titel „Sichere Systeme“ mit den existierenden Software-Frameworks zur Absicherung von Systemen beschäftigen. Die so genannte End-to-End Security innerhalb von Firmen wird darin ebenso behandelt wie die Absicherung von Servern auf den jeweiligen Plattformen. Anforderungen aus der DMZ Architektur werden in Form von Sicherheitsframeworks in J2EE bzw. Java erneut aufgegriffen und gelöst. Neben diesen softwaretechnischen Grundlagen erweitert der zweite Band das Thema der Software-Sicherheit um zwei wesentliche Dimensionen: Usa-

bility und Security einerseits, und andererseits die Einschränkung von Autorität als Grundprinzip bei der Entwicklung sicherer Systeme.

Der vorliegende Band basiert in Teilen auf Lehrveranstaltungen der Autoren zum Thema Internet-Sicherheit an der Hochschule der Medien in Stuttgart sowie auf eigenen Erfahrungen bei der Entwicklung von Portalen und anderen Applikationen. Es richtet sich vor allem an Softwareentwickler und Studierende der Informatik. Aber auch Spezialisten der Netzwerksicherheit, die eine Brücke zur Software-Security schlagen möchten (was in der letzten Zeit vor allem im Bereich Input Validierung geschieht), werden hoffentlich davon profitieren. Der Folgeband „Sichere Systeme“ geht darauf aufbauend sehr viel tiefer auf die Konstruktionsprinzipien von Software und ihren Auswirkungen für die Sicherheit von Systemen ein und richtet sich in noch stärkerem Maße an die Entwickler von Software. Nun aber zunächst viel Spaß beim Studium der Grundlagen der Internet-Security aus Software-Sicht.

Kapitel 2

Fallstudien

In diesem Kapitel werden wir uns mit zwei großen Internet-Portalen beschäftigen, die sicher jeder Leser schon einmal besucht hat, nämlich bahn.de und ebay.de. Portale bieten aus geschäftlicher Sicht sehr interessante Dienste an: Sie fassen die diversen Informationsquellen und Anwendungen der Unternehmen in einen homogenen Auftritt gegenüber Kunden und Mitarbeitern zusammen und heben dadurch die historisch bedingte Zersplitterung der internen Informationssysteme auf.

Aus IT-Sicht hingegen sind Portale komplexe Gesamtkunstwerke bestehend aus Hardware, Netzwerken und miteinander kommunizierenden Applikationen mit teils subtilen Abhängigkeiten untereinander. Vielen Softwareentwicklern – der Zielgruppe dieses Buches – bereiten gerade diese Abhängigkeiten zwischen der Software und der Netzwerkarchitektur große Probleme. Anders gesagt: Es sind die so genannten „Non-Functional Requirements“ hinter einem Portal – also Performance, Zuverlässigkeit etc., die von der Software zumindest mitbestimmt werden, den Entwicklern aber alles andere als klar sind, da sie sich über verschiedene IT-Bereiche einer Firma erstrecken.

Entsprechend interessant sind auch die Sicherheitsaspekte eines Portals, die sich von der – meist im Internet befindlichen – Kundenseite über die Ankopplung an das eigene Netz bis tief in die eigenen Backend Systeme des Intranets erstrecken. Häufig kommen noch Einbindungen externer Partner dazu.

Es gibt noch eine Reihe weiterer Gründe, warum die Sicherheit in Portalen oft als schwierig empfunden wird:

- Altsysteme müssen in großer Zahl angesteuert werden.
- Entwickler, die bisher interne Applikationen entwickelt haben, müssen sich in der raueren Welt der Internet Applikationen zurechtfinden.
- Inkompatible Systeme für Authentisierung und Authorisierung von Kunden und Mitarbeitern müssen integriert werden.
- Evtl. ist die Zusammenarbeit mit fremden Systemen nötig, um z. B. bereits authentisierte Kunden zu übernehmen oder Aktionen von Kunden an andere Firmen weiterzuleiten.

- Organisatorische Trennungen in Internet- und Intranetabteilungen erschweren den Wissensaustausch und schaffen Unklarheiten bezüglich der Sicherheitsstufe von Informationen.
- Komplexe Frameworks wie J2EE oder .NET unterstützen Entwickler zwar, indem sie den zu schreibenden Sicherheitscode reduzieren, allerdings um den Preis dass Entwickler die Abstraktionen dahinter verstehen müssen.

Somit sehen sich die Entwickler im Portalbau mit einer ganzen Reihe sicherheitsrelevanter Fragestellungen konfrontiert:

- Welche Eigenschaften muss ein Softwareprodukt besitzen, damit es in die existierende Sicherheitsarchitektur integriert werden kann? Dies ist wichtig bei Fremdprodukten, aber auch für die Entwicklung von Inhouse Software. Genügt es zum Beispiel, einen LDAP Anschluss anzubieten zur Authentisierung über ein Fremdsystem?
- An welcher Stelle in der Infrastruktur müssen die Portalrechner und Komponenten platziert werden damit sie aus dem Internet erreichbar sind und die interne Infrastruktur nutzen aber nicht gefährden können? Welche Anforderungen stellt eine solche „De-Militarized Zone“ (DMZ) an meine Software? Welche Ports werden normalerweise geöffnet? Welche Protokolle sind erlaubt, welche vorgeschrieben? Wie bekommt man die nötige Software in die DMZ? Wie erfolgt die Wartung und Administration?
- Wie muss eine solche Applikation abgesichert werden, das heißt sowohl netzwerktechnisch als auch in der internen Programmierung? Wie werden Server abgesichert? Was darf/soll die Applikation sicherheitstechnisch tun? Welche Services von Frameworks müssen genutzt werden?
- Mit welchen Attacken von intern oder extern muss gerechnet werden? Wie weit geht die Verantwortung der Applikation gegenüber dem Kunden?
- Wie müssen Rollen und Gruppen als Mittel der Zugriffskontrolle implementiert werden, damit sie skalieren, aber auch bei organisatorischen Änderungen leicht angepasst werden können?
- Welche Anforderungen an die Verfügbarkeit – ein wichtiger Sicherheitsdienst und wesentlicher Teil der Security eines Portals – werden gestellt und wie sind sie zu erfüllen?
- Mit welchen Identitäten bzw. Rechten müssen Infrastrukturkomponenten wie etwa Web Server laufen? Wie erfolgt dementsprechend der Zugriff auf Backend-Systeme und wie breit/offen ist dieser Zugriff?

Diese und weitere Fragen wollen wir auf den folgenden Seiten beantworten – zunächst im Kontext realer Anwendungen und dann anhand spezieller Modellkonfigurationen. Gerade der Kontext eines realen Portals ist wichtig für das erste Ziel dieses Buches, nämlich die Wahrnehmung von Sicherheitsproblemen bei Softwareentwicklern zu schärfen. Viele der Probleme und Begriffe, die wir in späteren Kapiteln dieses Buches genauer analysieren, werden in unseren Fallstudien eingeführt.

Als erstes konkretes Beispiel für ein komplexes Portal soll uns der Internet-Auftritt der Deutschen Bahn AG – bahn.de – dienen. Anhand der Geschäftsvorgänge für dieses Portal werden wir zentrale Sicherheitsanforderungen an das Portal ableiten. Als zweites betrachten wir die Internet-Auktionsplattform ebay.de, die in der Vergangenheit durch einige Angriffe in die Schlagzeilen geraten ist.

2.1 Online-Kartenkauf am Beispiel bahn.de

Die Sicherheit einer Applikation oder Infrastruktur beginnt nicht mit Protokollen oder Algorithmen wie SSL oder SHA-1, sondern mit dem Geschäftsmodell und den Geschäftsvorgängen, die durch die Applikation lediglich umgesetzt werden. Die Risiken hinter den Geschäftsvorgängen müssen bestimmt und in ihren Auswirkungen geschätzt werden, dann erst kann über Sicherheitstechniken nachgedacht werden, um die Risiken zu begrenzen. Und es muss genau überprüft werden, ob die gewählten Sicherheitstechniken den intendierten Geschäftszweck tatsächlich unterstützen, oder im Extremfall sogar neue Unsicherheiten mit sich bringen.

Damit soll nicht geleugnet werden, dass auch Sicherheitstechniken wie SSL unabhängig von ihrer Anwendung gewisse Risiken in sich bergen können: Der automatische Rückfall auf kryptografische Verfahren geringerer Qualität oder die Gefahren bei der Wiederaufnahme von Sessions sind Beispiele dafür, die wir weiter unten aufgreifen werden. Für Entwickler ist es aber zunächst am wichtigsten zu begreifen, dass es die Anforderungen aus den Geschäftsvorgängen sind, die die Sicherheitstechnik bestimmen.

Sicherheit ist zudem ein Prozess – wie der bekannte Security-Spezialist Bruce Schneier (siehe etwa [Schn01], S. 264) nicht müde wird zu betonen – und das heißt, dass alle Beteiligten – Manager wie Entwickler – auch über ein Betriebskonzept des Portals verfügen müssen. In diesem Sinne stellt die Entwicklung eines Sicherheitskonzeptes zwangsläufig einen Top-Down Prozess dar, der mit einem Überblick des Portals und seiner Funktionalität beginnt und den ganzen Lebenszyklus der Lösung umfasst.

2.1.1 Überblick

Eine Website zum Nachsehen von Fahrterminen und Preisen gibt es bei der Bahn schon lange. Im Sommer 2002 jedoch wurde die Funktionalität drastisch erweitert durch die Einführung des Online-Ticket-Verkaufs. Kunden können seitdem bequem von zuhause aus Tickets und Reservierungen bestellen und das Ticket komplett am Drucker daheim ausdrucken.

Diese Umstellung brachte einen großen Aufwand auch in der so genannten Betriebsorganisation mit sich: Schaffner mussten geschult werden und in der Lage sein, Online-Tickets direkt und unmittelbar zu validieren. Neue Geräte mussten

angeschafft oder alte angepasst werden, mit denen die Validierung durchgeführt werden konnte. Dies ist ein Aspekt, den gerade Entwickler gerne übersehen: Die eingesetzte Sicherheitstechnologie besitzt Auswirkungen auf die gesamte Arbeitsweise einer Firma und beeinflusst die Betriebsorganisation beträchtlich.

Gleichzeitig entwickelte sich das Portal immer mehr zu einer übergreifenden Plattform. Fremdwerbung tauchte auf, Partnerfirmen mit erweiterten Angeboten bis hin zur Kreditvergabe und Reiseplanung wurden eingebettet (s. Abb. 2.1). Aus dem Kundenportal der Bahn wurde damit ein so genanntes *föderatives Portal* – ähnlich der Entwicklung, wie man sie an den realen Bahnhöfen verfolgen kann, in denen die Bahn selbst häufig nur noch die Verkaufsplattform sowie den Kundenkontakt zur Verfügung stellt, aber die eigentlichen Verkaufsaktivitäten (Bäcker, Cafés, Bücher, Lebensmittelläden etc.) von Fremdfirmen erledigt werden.

In der Anfangsphase des Portals traten häufig Probleme im Bereich Performance auf. Wer am Sonntagabend einen Zug für Montagmorgen buchen wollte, sah sehr häufig die Meldung „Server nicht verfügbar“, speziell wenn es ans virtuelle Bezahlen ging. Falsche oder abgelaufene Zertifikate der Bahn selbst oder von Partnerfirmen trugen zusätzlich zur Verwirrung der Kunden bei.

Keineswegs einfach war auch die Umsetzung der Kontrollen des Online-Tickets – das sich übrigens im Laufe der Zeit kaum gewandelt hat – durch die Betriebsorganisation. Bei Reklamationen beispielsweise durfte der Online-Kunde sich nicht mehr an die überall verfügbaren Bahnschalter wenden. Kunden, die



Abb. 2.1 Online-Portal der Deutschen Bahn AG

einen Anschlusszug auf Grund von Verzögerungen des Zubringers nicht erreichen, müssen ihre verfallenen Reservierungen an einer bestimmten Stelle vorlegen. Offensichtlich sind also Online-Ticket und Normalticket in der IT-Infrastruktur in ganz verschiedenen Systemen untergebracht.

Sehr interessant war die Reaktion der beteiligten Schaffner die – ausgerüstet mit ihren mobilen Rechnern – die Online-Tickets prüfen mussten. In der Anfangszeit herrschte große Verwirrung über die Frage, wer wann welche Tickets zu prüfen hatte. Sollte nach jedem Umsteigen geprüft werden? Wie sollte das Ticket „entwertet“, das heißt, wie kann die Mehrfachnutzung eines Tickets verhindert werden? Hier galt es für die Bahn ein hinreichend flexibles System zu finden, das einerseits umfassende Kontrollen erlaubt und andererseits den Betrieb nicht zu sehr aufhält (und vielleicht auch die Kunden nicht über Gebühr belastet). Schon diese anfänglichen Überlegungen zeigen, dass das Herstellen von „Sicherheit“ aus geschäftlicher Sicht nicht unbedingt ausschließlich Risikovermeidung bedeutet, sondern häufig vielmehr den betriebswirtschaftlich sinnvollen Umgang mit Risiken beinhaltet.

Die Bedeutung von bahn.de für die zukünftigen Services der Bahn kann kaum überschätzt werden. So erzeugt das Portal bereits im Jahr nach der Einführung 1,6 Millionen Buchungen, 2004 wurden 3,8 Millionen und 2005 insgesamt 6,1 Millionen Tickets auf diesem Wege erworben. In 2006 wurden bereits 17% aller Buchungen online durchgeführt [heise83744].

2.1.2 Geschäftsmodell

Das Geschäftsmodell des Portals besteht aus dem Verkauf von Tickets und dem Anbieten weiterer Services, entweder durch die Bahn selbst oder vermittelt an Partnerfirmen. Die Besonderheit liegt in der hauptsächlich elektronischen Abwicklung aller Vorgänge, die dem Kunden einerseits Wege und Kosten spart und andererseits der Bahn Personalkosten in Verkaufsstellen erspart. Ähnlich den elektronischen Vorräumen von Banken (e-banking, ATMs) besteht auch hier der Zwang, dass sämtliche Vorgänge automatisch und fehlerfrei ablaufen müssen – anderenfalls lassen sich die geplanten Einsparungen wegen der dann nötigen Mehrausgaben für den Kundendienst (zum Beispiel für Call Center) nicht erzielen. Eine besondere Rolle spielt dabei die Art und Weise, wie mit dem Kunden kommuniziert wird (über das Portal, via E-Mail, über eine Hotline etc.).

Kiosk-Systeme in Supermärkten und anderen öffentlichen Plätzen erlauben den Erwerb von Tickets oder die Servicenutzung auch ohne Heim-Computer. Es kann erwartet werden, dass Kunden in Zukunft vermehrt PDAs mit Telefoniefunktion bzw. Smartphones für den Zugriff auf das Portal verwenden werden.

Einen weiteren interessanten Aspekt stellt die Möglichkeit der Profilbildung durch die Daten aus Kundentransaktionen dar – entweder auf individueller Ebene oder als Aggregatdaten. Durch die Vielfalt der angebotenen Services entstehen natürlich auch sehr aussagekräftige Daten, die in sich bereits einen hohen Wert

darstellen. Dazu passt, dass das Rabattsystem der Bahn im Privatkundengeschäft auf Einzelpersonen bezogen ist, das heißt, eine Bahncard 50 wird speziell für einen Kunden ausgestellt. Wenn nun bei Bezahl- oder Kontrollvorgängen die Bahncard verwendet wird, entsteht automatisch eine lückenlose Buchführung zum Reiseverhalten des Kunden – ein drastischer Unterschied zum anonym bezahlten und „abgeknipsten“ Ticket alter Schule.

Es bleibt noch das Problem des gemischten Kaufverhaltens (teils online, teils traditionell) am Schalter oder Automaten zu lösen. Kaufvorgänge, die offline durchgeführt werden, können mangels Identifizierung dem Kundenprofil nicht zugeordnet werden. Hier hilft die Einführung eines Bonuspunkte-Systems, wie es auch Supermärkte oder Kaufhäuser bereits kennen. Solche Systeme sollen unter anderem den Kunden dazu bringen, bei allen Kaufvorgängen seine Identität preiszugeben. Bei der Bahn findet die Datenkonzentration über die Bahncard des Kunden statt. So wird der Kunde seither auch am Schalter gefragt, ob er Punkte sammeln möchte. Dazu muss er natürlich identifiziert werden und dies geschieht durch die elektronisch lesbare Bahncard (die ansonsten beim Kauf eines herkömmlichen, nicht-elektronischen Tickets nicht nötig ist). Die teils beträchtlichen Rabatte, die bei solchen Bonuspunktessystemen eingeräumt werden, sind ein klarer Hinweis auf den großen Wert, den derartig erstellte Kundenprofile für die Firmen haben.

Wesentlich für das Geschäftsmodell ist, dass der Kunde das Portal als Anlaufstelle für eine ganze Reihe von geschäftlichen Vorgängen ansieht und diese durch einen einheitlichen Zugang nutzen kann. Dies betrifft vor allem die Frage der *Authentisierung*, also dem Nachweis einer behaupteten Identität, gegenüber weiteren Diensten.

Somit kann sich das Bahnportal seinerseits gegenüber den verschiedenen Geschäftsstellen der Bahn sowie Fremd- und Partnerfirmen als zentrale Stelle für die Verwaltung von Kundendaten und die Abwicklung von Finanztransaktionen anbieten. Voraussetzung dieses Geschäftsmodells ist natürlich, dass die Kunden Tickets und Services sicher und einfach bestellen können und die Abrechnung der Leistungen korrekt erfolgt.

2.1.3 Kundensicht

Die Kundensicht bestimmt die Funktionsweise eines Portals zu einem großen Teil. Das Portal bahn.de konzentriert sich auf den Einzelkunden. In der Vergangenheit konnten Privatpersonen unter der Voraussetzung, dass eine Bahncard vorhanden war, Online-Tickets bestellen (mittlerweile hat sich dies geändert). Natürlich stellte sich schnell die Frage, was mit Firmenkunden passieren würde: In größeren Firmen werden Tickets oft zentral bestellt und abgerechnet, das heißt, es besteht in diesem Fall keine 1:1 Beziehung zwischen Kunde und Portal. Dieses Problem wurde über einen separaten Zugang für Firmenkunden gelöst, der von dem hier betrachteten Portal unabhängig ist.

Ebenfalls zur Kundensicht gehört die Frage, was mit den Mitarbeitern der Bahn selbst geschehen soll. Wenn der Besitz einer Bahncard plötzlich Voraussetzung für Online-Tickets ist, wie kommen dann beispielsweise Mitarbeiter der Bahn, die bisher nur einen speziellen Ausweis besaßen, zu Tickets? Wenn ein Mitarbeiter über das Portal etwas bestellt – handelt er dann als Mitarbeiter oder Kunde? Ein ähnliches Problem betrifft die externen Partnerfirmen und deren Mitarbeiter: Welche Rolle spielen sie und wie werden sie identifiziert? Die Administratoren eines solchen Portals sowie die Mitarbeiter des Call Centers (falls es eines gibt) gehören ebenfalls zur Gruppe derer, die letztlich Zugriff auf das Portal erhalten müssen. Kann diese Administration dann „outgesourced“ werden? Welche Konsequenzen hätte dies für die Sicherheit des Zugangs?

Die Modellierung des Personenkreises, der Zugriff auf das Portal hat, ist eine der schwierigsten Aufgaben. Aus technischer Sicht tauchen die Probleme vor allem in den Gebieten *Authentisierung* (der sicheren Verifikation einer Kunden- oder Mitarbeiteridentität) und *Authorisierung* (der Zuweisung einer Rechtemenge an einen authentisierte Nutzer) auf. Ein klassisches Beispiel für diese Problematik stellen temporäre Kunden dar, die vielleicht im Rahmen einer Werbeaktion für begrenzte Zeit kostenlosen Zugriff auf bestimmte Services erhalten. Sie brauchen eine echte Identität im System, die jedoch automatisch nach festgelegter Zeit verfallen muss – oder der Kunde wird ein „fester“ Kunde: Können seine bisherigen Einstellungen und die Historie seines Verhaltens migriert werden oder beginnt er als registrierter Kunde mit einer neuen Identität?

Eng mit dem Konzept Kunde bzw. Partner ist auch die Frage der Identität im System verknüpft: Darf der Kunde sich einen eigenen Namen/UserID aussuchen? Gibt es eine vorgeschriebene Syntax? Legt das Portal hier Regeln fest?

Schließlich ist ein letzter Punkt von wesentlicher Bedeutung für das Geschäftsmodell: Tritt bahn.de als selbständige Authentisierungsstelle auf, das heißt identifiziert das Portal die Benutzer selbständig (was auch die initiale Registrierung bisher unbekannter Kunden einschließt) oder schließt es sich einem Verbund mehrerer Firmen an, in dem die Authentisierung von einer gemeinsamen, zentralen Instanz übernommen wird? Hier hat sich die Bahn zunächst für die Selbständigkeit entschieden – wohl nicht zuletzt, um die wertvolle direkte Kundenbeziehung nicht zu gefährden. Der Preis dafür ist die Notwendigkeit, eine eigene Infrastruktur für das Identitätsmanagement aufzubauen zu müssen, wie sie weiter unten beschrieben wird. Damit bildet bahn.de eine eigene so genannte Sicherheitsdomain oder *Realm*.

2.1.4 Geschäftsvorgänge

Welche Geschäftsvorgänge müssen beim Aufbau eines Sicherheitskonzepts für das Portal betrachtet werden? Hier eine (sicher unvollständige) Liste:

- Ein neuer Kunde registriert sich und erhält so genannte *Credentials* (das heißt Username und Passwort) für zukünftige Anmeldungen.
- Ein unbekannter Kunde informiert sich anhand der öffentlichen Seiten des Portals.
- Eine Firma bestellt ein Ticket für eine Mitarbeiterin (ausgelagert in Firmenportal).
- Ein Kunde storniert einen Auftrag und muss eine Vergütung erhalten.
- Ein Kunde meldet sich an, kauft ein Ticket und benutzt Services und Angebote von Partnerfirmen.
- Ein Kunde verliert oder vergisst seine Credentials und fordert neue an.
- Ein Sachbearbeiter vollzieht einen problematischen Vorgang anhand von Vorgangsdaten nach.
- Mitarbeiter von Partnerfirmen beantragen Zugriff auf Stammdaten der Kunden.
- Ein Kaufvorgang wird abgerechnet, z. B. über Kreditkartenfirmen.
- Ein Kunde bestellt kurz vor Abfahrt des Zuges ein Ticket.
- Ein Kunde bestellt eine Reservierung.
- Ein Kunde will seine Stammdaten ändern (andere Karten, Adressen etc.).

Es ist sicher sinnvoll, auch negative Geschäftsvorgänge aufzunehmen:

- Ein Kunde verwendet eine gestohlen oder verloren gemeldete Bahncard.
- Zwei Kunden versuchen, das gleiche Online-Ticket zu verwenden.
- Ein Kunde hat sein Online-Ticket vergessen, hat jedoch eine elektronische Kopie auf dem Laptop dabei.
- Ein Kunde streitet einen Ticket- oder Servicekauf ab.
- Ein Kunde kann Tickets oder Services nicht bezahlen.

Die reguläre Bearbeitung von Vorgängen, aber auch administrative Tätigkeiten erfolgen schließlich durch das interne Komponentenmodell. Datenarten, Datenhaltung etc. werden hier modelliert und für Bearbeiter zugreifbar.

2.1.5 Sicherheitsanforderungen

Aus den Geschäftsvorgängen lässt sich dann unter Berücksichtigung verschiedener Bedrohungsmodelle der so genannte *Security Context* erstellen, in dem eine ganze Reihe von Sicherheitsanforderungen an das Portal festgehalten werden. Auf diese Anforderungen gehen wir im Folgenden näher ein.

2.1.5.1 Schutz der öffentlichen Bereiche

Das Portal soll frei zugängliche Informationen anbieten, und zwar über Fahrziele und Fahrpläne, Preise, Sonderangebote und sonstige Dienstleistungen. Diese Informationen sollen die korrekte Antwort auf entsprechende Anfragen der Kunden dar-

stellen. Ebenso soll der Kunde sicher sein können, dass die in seinem Browser angezeigten Informationen korrekt sind und auch tatsächlich von der Bahn stammen.

Diese recht einfach klingenden Anforderungen beinhalten bereits eine Fülle von Sicherheitsproblemen und offenen Fragen, die sich auf drei Bereiche konzentrieren:

- Identifikation des Portals
- Produktion der Information
- Auslieferung der Information

Zunächst stellt sich für die Bahn das Problem, wie der Kunde überhaupt das Bahnportal findet und erkennt. Firmen meinen oft, das Problem der Identifikation durch die Verwendung einprägsamer Domain-Namen und eines durchgängig gestylten „Look&Feel“ lösen zu können. Um darüber hinaus die Kunden vor Angreifern zu schützen, die ihr Look&Feel kopieren und leicht abgeänderte Domain-Namen verwenden, verwenden die Firmen meist das Sicherheitsprotokoll SSL (siehe Kap. 7). Dabei kommen so genannte *Zertifikate* zum Einsatz, das sind elektronische Bestätigungen durch eine vertrauenswürdige Instanz, dass der Server mit dem Namen www.bahn.de der Deutschen Bahn AG gehört und darüber hinaus einen bestimmten öffentlichen Schlüssel besitzt. Auf die damit verbundenen Mechanismen der Identifikation und Authentisierung wird weiter unten noch genauer eingegangen, hier nur eine etwas pessimistische Vorbemerkung dazu: Es gibt kein rein technisches Mittel, mit dem man sicherstellen könnte, dass die Intention des Kunden über die Identität seines Kommunikationspartners mit der Realität übereinstimmt, denn das Look&Feel des originalen Bahn-Servers lässt sich fälschen, und welcher Kunde würde schon auf den Unterschied zwischen der „Deutsche Bahn AG“ und einer (fiktiven) „Deutsche Bahn GmbH“ achten, die im Server-Zertifikat als Inhaber genannt werden?

Letztlich ist dies auch das Problem, das dem in der letzten Zeit so populär gewordenen *Phishing* (ein Kunstwort aus „Password“ und „Fishing“) zugrunde liegt, bei dem Kunden durch eine authentisch aussehende E-Mail und einen ähnlich lautenden Domänennamen auf eine identisch aussehende Website gelockt wird. Wenn der Kunde seine Credentials (UserID, Passwort) dort eingibt, um sich zu authentisieren, übergibt er sie tatsächlich an einen Angreifer, der sie sofort missbraucht.

Dieses grundsätzliche Identifikationsproblem im Internet lässt sich nur mit Hilfe aufwändiger Maßnahmen wie etwa vorinstallierten Applikationen, so genanntem Key Continuity Management oder der separaten Signierung jeder einzelnen Transaktion durch den Kunden in den Griff bekommen (so genannte *objektbasierte Sicherheit*), die alle auch mit gewissen Kosten verbunden sind.

Die Produktion der korrekten Informationen und ihre Veröffentlichung im Internet ist eine interne Angelegenheit des Portals. Da sich die publizierte Information nicht vollständig formal und automatisch prüfen lässt, sichern die meisten Firmen ihre Informationen durch eine Mischung aus technischen Prozessen und organisatorischen Maßnahmen. Dies dient sowohl der Qualitätssicherung als auch

dem Schutz vor Klagen auf Grund falscher Informationen. Zu den Anforderungen an einen sicheren Veröffentlichungs-Prozesses gehören:

- Abklärung, welche Daten öffentlich sind (Datenklassifizierung)
- Authentisierung aller Redakteure
- Zuteilung und Verwaltung von Gebieten und Zuständigkeiten (Authorisierung) durch Rollen
- Versionskontrolle aller Änderungen
- Archivierung einmal gezeigter Information
- Absicherung dynamischer Inhalte durch rollenbasierte Zugriffsmodelle auf Datenbanken und die Verwaltung von Templates durch einen Software-Deploymentprozess
- Auditing aller Zugriffe von extern wie intern
- Monitoring der Verfügbarkeit der Systeme

Portale, die außerdem extrem kritische Daten beinhalten, können als Ergänzung zu diesen organisatorischen und technischen Maßnahmen auch noch ein so genanntes *Mandatory Access Control* (MAC) System einsetzen, das durch *Daten-Labeling*, also ein Einteilen der Dokumente in unterschiedliche Vertraulichkeitsstufen, verhindert, dass geheime Daten auf diese Weise nach draußen gelangen können.

Schließlich soll der Kunde sicher sein, dass die Informationen von der korrekten Quelle ausgeliefert werden und auf korrekte, unverfälschte Weise bei ihm ankommen. Was aber bedeutet „korrekt“ in diesem Zusammenhang? Soll der Kunde beispielsweise die Informationen in einer Form erhalten, die er später in einer rechtlichen Auseinandersetzung verwerten kann? Soll der Kunde also im Nachhinein beweisen können, dass bestimmte Informationen so und nicht anders auf www.bahn.de zu sehen waren? Dann müssen die Inhalte durch das Portal digital signiert werden. Oder genügt es bahn.de sicherzustellen, dass die Informationen auf dem Weg zum Kunden nicht verfälscht werden können? Dann muss das Portal lediglich für einen sicheren Kanal zum Kunden sorgen. Oder sollte das Portal davon ausgehen, dass die von ihm angezeigten öffentlichen Informationen ohnehin nicht gefälscht werden und überträgt sie deshalb ohne Absicherung? Dies spart auf jeden Fall Infrastrukturkosten auf Seiten des Portals. In diesem Zusammenhang muss geklärt werden, ob bereits die Aufforderung zur Authentisierung über eine sichere Verbindung erfolgen soll, oder ob diese erst bei der Übertragung der Daten aufgebaut wird. Selbst Bankportale (z. B. das der Citibank) haben sich in der Vergangenheit hier falsch entschieden.

Sollte sich das Portal für eine sichere Übertragung oder gar für signierte Dokumente entscheiden, dann muss es auch die Verantwortung dafür übernehmen, das heißt, es ist Aufgabe des Portals, sicherzustellen, dass die dazu nötigen Verfahren zur Absicherung mit ausreichender Sicherheit durchgeführt werden (was Konsequenzen für die Einstellungen der Server, die verwendeten Schlüssel etc. mit sich bringt).

Manche Portale versuchen ihre Seiten und Links so zu gestalten, dass es möglich ist festzustellen, ob eine Seite von einem bestimmten Portal erzeugt und aus-

geliefert wurde. Dies hilft, gewisse Attacken (so genanntes *Cross-Site-Request-Forging*) zu vermeiden. Hier muss zwischen den Gütern „Performance durch Caching“ und „Sicherheit“ abgewogen werden. Öffentliche read-only Informationen können beispielsweise gut in einem Cache gespeichert werden – gegebenenfalls am so genannten „Rand“ des eigenen Netzwerks (edge caching) – und damit häufige und teure Zugriffe auf die Backendsysteme vermieden werden.

Der Zugriff auf öffentliche Informationen durch die Kunden birgt für das Portal jedoch noch eine weitere Gefahr: Dadurch ist es *jedem* Client erlaubt, Requests an das Portal zu stellen. Somit können also auch automatische Scripts, die auf Hunderten von Rechnern laufen, gleichzeitig solche Requests stellen – eine so genannte *Distributed Denial of Service* (DDoS) Attacke, bei der eine große Zahl von Rechnern unter der Kontrolle von „Bots“ eine automatische Attacke gegen einen bestimmten Host fahren.

Wie sich das Portal gegen solche Attacken wehren kann, wird weiter unten im Kapitel „Sicherheit der Infrastruktur“ beschrieben. Hier wollen wir einstweilen nur festhalten, dass öffentliche Zugriffe keine Gefahr für interne Netzwerke des Portals darstellen dürfen und dass der Aufbau einer Sicherheitszone (DMZ) für das Portal und seine öffentlichen Inhalte Pflicht ist.

Eine weitere wichtige Sicherheitsfrage im Zusammenhang mit öffentlichen Inhalten ist, ob und welche Schreibvorgänge erlaubt sein sollen. Sollen zum Beispiel auch nicht-registrierte Kunden ein Feedback Formular ausfüllen oder eine Nachricht senden dürfen? Der technische Hintergrund hierfür besteht darin, dass das Halten von State serverseitig besonders kritisch ist da es Ressourcen verbraucht. Dies kann unter Umständen ebenfalls zu DoS-Attacken führen.

2.1.5.2 Gesicherte, personalisierte Bereiche

Bestimmte Dienste wie die Bestellung von Tickets haben Vertragscharakter und benötigen dazu die Identität der Vertragspartner. Bestimmte Daten wie die Tickethistorie eines Kunden sind privater Natur und dürfen nach außen nur für den Kunden zugänglich sein.

Voraussetzung für die Nutzung dieser Dienste und Daten ist daher eine vorhergehende Authentisierung des Kunden. Jeder Zugriff ohne vorherige Authentisierung muss vom Portal abgewiesen werden. Insbesondere darf es einem Kunden nicht möglich sein, die Daten anderer Kunden zu sehen oder zu verändern (wie bei OBSOC geschehen). Das System hat also geeignete Vorkehrungen zu treffen, dass Kunden voneinander getrennt behandelt werden. Dies ist ein wichtiges Detail für die spätere serverseitige Implementation.

Diese 1:1 Beziehung zwischen Portal und Kunde wird in dem Moment aufgeweicht, in dem z. B. kollaborative Services wie Nutzerforen oder Wikis angeboten werden. Dann treten Kunden direkt miteinander in Kontakt und es entstehen neue Angriffsmöglichkeiten. Die Einführung solcher Services ist daher sowohl aus Business-Sicht als auch aus Sicht der Security gut zu überlegen.

2.1.5.3 Sichere Administration

Die Systemverwaltung stellt einen sehr kritischen Dienst dar. Die Dienste der Systemverwaltung des Portals sollten deshalb nur aus dem Intranet der Firma zur Verfügung stehen und nur einem ausgewählten Personenkreis. Aufgrund der Sensitivität dieses Dienstes gelten hier besonders hohe Sicherheitsanforderungen:

- Für die Anmeldung in einer Administratorrolle ist eine besondere Güte der Authentisierung vorauszusetzen. Das bedeutet insbesondere, dass Passwörter von hoher Qualität (zufällig gewählt, ausreichend lang) für die Administratorrolle zum Einsatz kommen und regelmäßig gewechselt werden. Oder noch besser: Es wird grundsätzlich auf stärkere Mechanismen der Authentisierung als Passwörter gesetzt.
- Die Administratorfunktion muss in mehrere Rollen aufgeteilt sein, das heißt, es darf keinen allmächtigen Systemverwalter geben wie das bei Unix der „Root“ oder bei Windows der „Administrator“ ist. Vielmehr ist darauf zu achten, dass jede Rolle nur die für die Erfüllung ihrer Pflichten nötigen Rechte erhält (*Need-to-know/Need-to-do Prinzip*).
- Des Weiteren müssen einzelne Abteilungen die Möglichkeit haben, eigene Subadministratoren zu benennen, die für Teilbereiche verantwortlich sind. Innerhalb dieser Teilbereiche können sie Rollen und Rechte von Sachbearbeitern festlegen (*Service Management Delegation*).
- Jede administrative Tätigkeit muss ausführlich protokolliert werden. Die Protokolle werden auf einem write-once Medium gespeichert (*Mandatory Audit*).

2.1.5.4 Sichere Sachbearbeitung

Innerhalb der Betriebsorganisation des Portals und der anschließenden Dienste der Bahn erhalten Mitarbeiter Zugriff auf Daten von Kunden zur weiteren Verarbeitung. Dazu gehören neben den Stammdaten natürlich auch Kreditkartenzahlen, Kartenprüfzahlen, Bankverbindungen und weitere persönliche Daten. Als Teilnehmer am Kreditkartensystem unterliegt das Portal ohnehin den Regeln der Kreditkartenunternehmen, die die Einhaltung dieser Regeln durch so genannte *Audits* (online und lokal bei der jeweiligen Firma) prüfen. VISA z. B. hat eine Liste von Anforderungen veröffentlicht, die von beteiligten Firmen erfüllt werden müssen.

Neben allgemeinen Sicherheitsmaßnahmen z. B. durch Firewalls und Virenscanner setzt eine sichere interne Verarbeitung die Einführung von Authentisierung und Autorisierung auf Basis von Rollenkonzepten voraus. Konkrete Aktionen von Mitarbeitern sind anschließend zu protokollieren.

Dies ist ohne Zweifel einer der schwierigsten Aspekte sicherer Software und dennoch leider unumgänglich, wie Fälle wie der von Elliot Castro zeigen: Castro missbrauchte jahrelang fremde Kreditkarteninformationen, an die er z. B. über Tätigkeiten in Call Centern oder Firmen gelangte (s. [Castro]). Allerdings darf nicht verschwiegen werden, dass es häufig die mangelnde Betriebsorganisation

der beteiligten (Karten)-Firmen war bzw. der Wunsch nach mehr verkauften Karten, der die Betrügereien wesentlich erleichterte. Einen sehr interessanten Einblick in die Online-Unterwelt, in der Kreditkartendaten gesammelt, angeboten und weiter verkauft werden, gibt Tobias Knecht aus dem 1und1 Abuse Center in seinem Vortrag [Knecht].

2.1.5.5 Sichere Anmeldemöglichkeiten

Damit eine Bestellung einem bestimmten Kunden zugeordnet und der Bezahlvorgang eingeleitet werden kann, ist eine sichere Anmeldung der Kunden beim Portal scheinbar unabdingbar. Dazu werden üblicherweise einmalig die Stammdaten des Kunden abgefragt und dann ein Login mit zugehörigem Geheimnis erzeugt. Der Kunde ist dann gezwungen, bei jeder Buchung diesen Login zu verwenden.

Wie Erfahrungen mit Online-Shops zeigen, möchten jedoch viele Kunden gerne einfach mit ihrer Kreditkarte jeden Kauf bezahlen und dabei auf einen Login verzichten – wer will sich schon gerne für jeden Shop ein Passwort merken? Für diese Fälle hat bahn.de die Bezahlung mit Kreditkarte eingeführt – und zwar ohne zusätzlichen Login. Der Verzicht auf einen Login zeigt klar die Grenzen von Authentisierung im Geschäftssinn auf: Karten, Tickets, Zertifikate etc. ermöglichen Geschäftsvorgänge auch ohne vorherige genaue serverseitige Authentisierung. In Zukunft ist damit zu rechnen, dass sich diese Form von Absicherung von Geschäftsvorgängen weiter ausdehnen wird. Dafür ist die so genannte objektbasierte Sicherheit nötig, auf die wir später zurück kommen werden. Im Fall der direkten Buchungen mit Kreditkarte ist z. B. die Bestellung durch jemand anderen als den Kartenbesitzer kein Problem – im guten wie im schlechten Sinne. Hier gilt es im Geschäftsmodell die Vorteile durch die höhere Bequemlichkeit für den Kunden zu vergleichen mit den besseren Betrugsmöglichkeiten durch gestohlene Karteninformationen, und dann zu einer quantitativen Einschätzung des Risikos zu kommen.

Gehen wir nun davon aus, dass sich die Kunden tatsächlich bei bahn.de einloggen und authentisieren. Dann sind die wesentlichen Anforderungen des Portals bzgl. der Authentisierung:

- dass sich Benutzer selbständig online auf sichere (das heißt insbesondere vertrauliche und unverfälschte) Weise registrieren können;
- dass Benutzer ihre Stammdaten selbständig online verwalten können (z. B. Updates der Kreditkarteninformationen oder der Adresse), um Kosten zu sparen;
- dass die Authentisierung genügend sicher ist, um Tickets verkaufen zu können. Insbesondere darf es für einen Angreifer nicht möglich sein, sich unter einem falschen Namen beim Portal anzumelden;
- dass der Kunde eindeutig als rechtlich eigenständige Person identifizierbar sein muss, das heißt ein bloßes Pseudonym als Identität – wie es um Beispiel in Internet-Foren völlig ausreicht – ist hier nicht möglich. Anders ausgedrückt: der

Kunde muss unter einer Adresse erreichbar sein, unter der notfalls Forderungen eingetrieben werden können – eine bloße E-Mail Adresse reicht hierfür normalerweise nicht aus.

Ein interessantes Problem besteht in der Frage, ab wann ein neu registrierter Kunde kostenpflichtige Services nutzen kann. Darf nämlich jemand sofort nach der Registrierung beliebige Tickets bestellen, so kann das für Denial-Of-Service-Attacken ausgenutzt werden, sei es manuell oder durch Scripts, die automatisch neue Benutzer anlegen und Bestellungen erzeugen.

Wie bereits erwähnt, war lange Zeit der Besitz einer BahnCard Voraussetzung für die Nutzung des bahn.de Portals zur Bestellung von Online-Tickets. Dadurch konnte bereits bei der Registrierung ein Missbrauch eingeschränkt werden. Dies ist ein Beispiel für die geschickte Nutzung bereits bestehender Authentisierungen bei der Registrierung von Neukunden.

2.1.5.6 Single-Sign-On

Single-Sign-On (SSO) bedeutet, dass sich ein Kunde nur ein einziges Mal mit Username und Passwort authentisieren muss, um in der Folge mehrere unterschiedliche Dienste nutzen zu können. Dies betrifft:

- die Verwendung mehrerer portalinterner Dienste
- die indirekte – das heißt über Portaldienste vermittelte – Nutzung externer Dienste
- die Weiterleitung des Kunden an externe Dienste

Für ein Portal wie bahn.de, dessen Geschäftsmodell unter anderem in der Vermittlung der Kunden an externe Dienstleister besteht, ist Single-Sign-On sicher eine wünschenswerte Systemeigenschaft. Natürlich gelten für die initiale Anmeldung am Portal die gleichen Sicherheitsanforderungen wie oben besprochen. Single-Sign-On beinhaltet aber zusätzlich das sichere „Weiterreichen“ der authentisierten Kunden an weitere, evtl. auch externe Dienste. Die hierfür eingesetzten Techniken und die damit verbundenen Problematiken werden uns im späteren Verlauf noch mehrere Male begegnen.

In dem Maße, wie das das Portal „fremde“ Authentisierungen annehmen möchte – es muss sich hier nicht um „Sessions“ handeln, sondern Kunden könnten auch Gutscheine anderer Organisationen zur Bezahlung etc. einreichen – wird die Frage fremder Autoritäten eine größere Rolle spielen. Das gleiche gilt für Firmen, die sich dem Bahnhof angeschließen wollen und dessen Authentisierungen bzw. Autorisierungen übernehmen möchten. Hier existieren sehr viele verschiedene Möglichkeiten, die von der Einrichtung vieler unabhängiger Instanzen zur Authentisierung über föderative Strukturen bis hin zu zentralen Systemen mit Identity Provisioning für beteiligte Firmen reichen. Wir werden diese Möglichkeiten im Kapitel zu föderativer Sicherheit besprechen.

2.1.5.7 Geschützte Kommunikation Kunde – Portal

Das primäre Kommunikationsmedium für die Kommunikation mit dem Kunden ist natürlich das Portal selbst. Die hiermit verbundenen Probleme der korrekten Identifikation und korrekten Informationsübermittlung haben wir bereits angeprochen. Von Zeit zu Zeit wird es aber auch notwendig sein, einzelne Kunden direkt anzusprechen, um ihnen einen neuen Punktstand, veränderte Abläufe etc. mitzuteilen. Auch diese direkte Kommunikation hat auf sichere und zuverlässige Weise zu geschehen, da nicht auszuschließen ist, dass sensitive Daten in diesen Nachrichten versandt werden müssen.

Das offensichtlichste (und auch billigste) Medium E-Mail weist hier große Probleme auf: Die Inhalte sind weder vertraulich noch vor Änderungen geschützt, es gibt keine Garantie, dass eine E-Mail auch ankommt, und schließlich stellt sich auch hier wieder die Frage, wie ein Kunde entscheiden soll, ob eine Mail tatsächlich von bahn.de stammt oder von einem Angreifer, der die Anmeldedaten des Kunden „phishen“ will. Die Alternativen zur E-Mail lauten Briefpost, der Betrieb eines Call-Centers oder der Aufbau einer eigenen Messaging Lösung auf Basis einer Datenbank.

Grundsätzlich gilt in diesem Bereich: Je sicherer die gewählte Lösung, desto aufwändiger und teurer ist sie auch. Hier sind sorgfältige Abwägungen von Aufwand und Nutzen gefragt. Der Aufbau eines sicheren Messaging auf Datenbank-Basis besitzt in jedem Fall einige entscheidende Vorteile gegenüber E-Mail:

- Eine Nachricht geht nicht verloren.
- Die Integrität und Vertraulichkeit der Nachricht kann zumindest teilweise gesichert werden.
- Es kann überprüft werden, ob eine Nachricht gelesen wurde.
- An eine Nachricht des Kunden kann ein firmeninterner Workflow gebunden werden, d.h. es gibt klare Regelungen bezüglich des Weiterreichens im Falle von Urlaub, Stellvertretungen etc.
- Die Bearbeitungszeit von Nachrichten kann festgestellt werden.
- Der Absender kann klar identifiziert werden. Dies ist wichtig bei Nachrichten mit großen finanziellen Konsequenzen.
- Nachrichten des Kunden werden strukturiert empfangen und können daher leichter automatisch ausgewertet werden.

Im Grunde fallen diese Punkte bereits unter den Oberbegriff Enterprise Feedback Management. Hier ist das Ziel, alle Kommunikation von und zum Kunden strukturiert zu sammeln und über Message Bus Systeme an Applikationen zur Auswertung weiter zu transportieren. In diesem Zusammenhang ist die Kommunikation zwischen Kunde und Firma über E-Mail auch aus Sicht der Geschäftsorganisation mangelhaft.

Entscheidet sich die Applikation dennoch für E-Mail als Kommunikationsmedium, so sollten zumindest die Nachrichten des Portals durch das Portal digital signiert werden und auf diese Weise auch der Schlüssel des Portals zum Kunden transportiert werden. Es gibt zunehmend einfachere Plug-ins für E-Mail-Systeme, die eine recht einfache Validierung erlauben.

2.1.5.8 Schutz der Tickets gegen Missbrauch und Fälschung

Welche Anforderungen bestehen nun hinsichtlich des Online-Tickets, das dem Kunden in Form einer pdf-Datei übermittelt wird (siehe Abb. 2.2)?

Da der Ticketkauf an eine bestimmte Kundenidentität (und damit auch an gewisse Rabatte zum Beispiel für Besitzer einer BahnCard) gebunden ist, sollte die Identität des Käufers auch auf dem Ticket erscheinen. Des Weiteren hängt der Ticketpreis natürlich von einer bestimmten, ausgewählten Strecke ab. Beide Informationen, Inhaber des Tickets und Fahrstrecke, müssen in also einer Weise auf dem Ticket erscheinen, die es unmöglich macht, sie vom Rest des Tickets abzutrennen und durch andere Daten zu ersetzen. Die Bahn setzt zu diesem Zweck so genannte kryptografische Hashfunktionen (siehe Kap. 5) ein.

Natürlich darf es für Unbefugte auch nicht möglich sein, selber Tickets herzustellen. Deshalb muss in die Hashwertbildung auch ein geheimer Schlüssel eingehen. Das Ergebnis der Hashwertbildung findet sich auf dem Ticket in Form des „Zertifikats“ und kann vom Schaffner schnell und unkompliziert validiert werden. Schließlich kann, da das Ticket in elektronischer Form vorliegt, dieses theoretisch mehrmals ausgedruckt und für mehrmalige Fahrten genutzt werden. Für diesen Missbrauchsfall wird Sorge getragen, indem die Validierungsvorgänge für jedes Zertifikat vom Schaffner in einen zentralen Rechner übertragen werden. So kann bei mehrfacher Nutzung eines Tickets das Konto des betreffenden Kunden mehrfach belastet werden.

Es hat sich erst kürzlich gezeigt, dass das Online-Ticket für die Bahn sogar sicherer ist als herkömmliche Fahrkarten. So wurden im Zusammenhang mit Wochenendtickets aus Fahrkartautomaten Fälschungen sichergestellt, bei denen Aus-

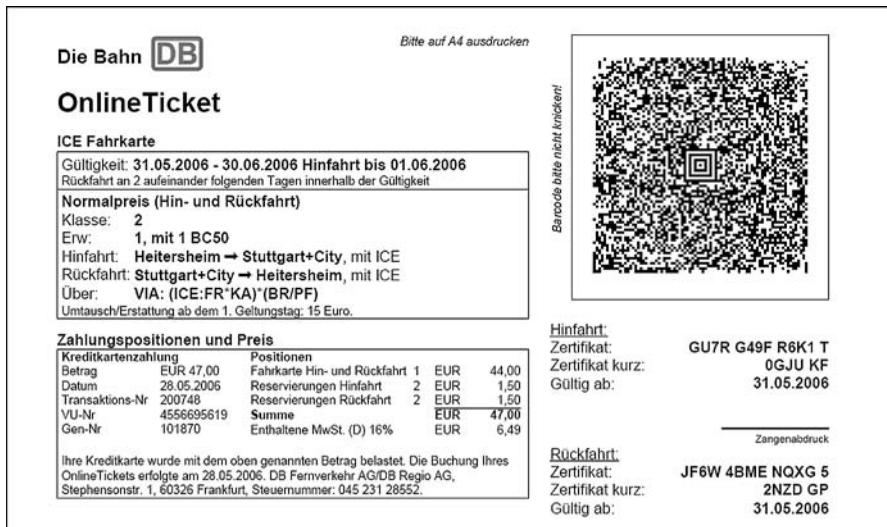


Abb. 2.2 Online-Ticket der Deutschen Bahn AG

kunftsbelege – die den echten Tickets optisch recht ähnlich sind – durch Haarspray gelöscht und neu bedruckt wurden. Diese Tickets wurden anschließend an Reisende in Bahnhöfen zu einem günstigeren Preis als dem Originalpreis verkauft. Dies erforderte letztlich sogar eine Umrüstaktion der Automaten (s. [heise83744]).

2.1.5.9 Sicherheit des Kunden (Clientbereich)

Die Sicherheit der Kunden, der von ihnen benutzten Plattform und der eventuell darauf gespeicherten Daten und Credentials war lange Zeit kein Thema für die Betreiber von Internet-Sites. Das Phänomen Phishing sowie eine Unzahl verwandter Attacken wie Spoofing, Cross-Site-Scripting etc. in Verbindung mit dem ökonomischen Problem, dass gefälschte Käufe nicht gerne beglichen werden, führen hier langsam zu einem Umdenken.

Deshalb werden die folgenden Sicherheitsanforderungen aus Kundensicht an die Portalentwicklung gestellt:

- Die Kundenplattform muss in die Sicherheitsanalyse des Portals mit einbezogen werden. Insbesondere muss von Seiten des Portals die maximal mögliche Sicherheit für die Clientseite gefördert werden.
- Kundenbeziehungen zu anderen Sites/Services müssen berücksichtigt werden.
- Die lokale Situation des Kunden (Heim-PC/Kiosk/Internet-Cafe) muss berücksichtigt werden.
- Von einer grundsätzlichen Gefährdung des Kundenrechners durch Malware ist auszugehen. Dies betrifft auch die auf Kundenrechnern gespeicherten Anmelddaten (Credentials) wie Passwörter, UserIDs, etc.
- Die Risiken für die Vertraulichkeit der Kundencredentials müssen abgeschätzt werden. Das Risiko für den Kunden ist zu minimieren.
- Die Bedienung der sicherheitsrelevanten Dialoge des Portals muss einfach und verständlich sein.

Das Gefährdungspotential auf der Clientseite richtig zu erkennen und soweit möglich Abwehrmaßnahmen zu treffen, ist für die Sicherheit der Geschäftstransaktionen – und darum geht es beim Portal ja letztlich – essentiell. Praktisch bedeutet dies, dass die Bedrohungsmodelle, die der Applikation zu Grunde liegen, erweitert werden müssen um die Nutzer-Plattform und das User Conceptual Model.

Im Rahmen des Geschäftsmodells von bahn.de muss geklärt werden, welche Risiken auf der Clientseite bestehen und welche ökonomischen Folgen daraus resultieren könnten. Besonders interessant ist hier der Fall der föderativen Nutzung des Client-Logins durch andere Dienste und Firmen.

2.1.5.10 Absicherung des Infrastrukturbereichs

Der zum Portal gehörige Infrastrukturbereich muss gegen Attacken und Missbrauch bzw. den Verlust der darin verarbeiteten Daten geschützt werden. Die

Gefahr, die dem Portal durch Denial-of-Service Attacken droht, haben wir bereits angesprochen. Ihr kann durch Maßnahmen auf Netzwerkebene wie Firewalls und eine geeignete Netztopologie begegnet werden.

Auf Software-Ebene ergeben sich weitere potenzielle Angriffsmöglichkeiten aus der Tatsache, dass das Portal vom Kunden bereit gestellte Daten entgegen nehmen und an die nachgeordneten Systeme weiterreichen muss. Ein geschickter Angreifer könnte diese Daten so gestalten, dass sie von den nachgeordneten Systemen als Befehle interpretiert werden (*SQL-Injection, Buffer-Overflow-Attacks*). Durch eine so genannte *Input Validation* muss deshalb sichergestellt werden, dass an Hintergrundsysteme weitergereichte Kundendaten keine Schadwirkung entfalten können.

Eine besondere Schwierigkeit liegt beim Portalbau in der Vielzahl der Backendsysteme, die üblicherweise daran angeschlossen werden müssen. Daraus ergibt sich eine Vielzahl komplexer Fragestellungen, die wir im zweiten Band genauer untersuchen werden: Mit welchen Protokollen sollen bzw. müssen diese Systeme angeschlossen werden? Wie fließt die Kundenidentität von System zu System? Wo wird authentisiert? Unter welchen Identitäten arbeiten Server innerhalb der Infrastruktur?

2.1.6 Testkonzept

Abschließend sind die Erfüllung der oben genannten Anforderungen bezüglich der Sicherheit des Portals und der benötigten Backendsysteme durch empirische Testverfahren zu prüfen. Dazu gehören im Besonderen:

- Versuche, als Kunde Daten anderer Kunden zu manipulieren;
- Versuche, als Kunde Dienste unberechtigterweise zu nutzen;
- Versuche, die Verfügbarkeit des Portals zu beeinträchtigen;
- Versuche, Administrationsfunktionen durch interne Mitarbeiter zu missbrauchen;
- Versuche, die Anmeldungsverfahren zu unterlaufen und
- Versuche, die Clientseite zu attackieren, um dort Credentials zu stehlen.

Ein besonderes Problem stellt das Testen von semantischen Attacken dar: Wie leicht können Kunden auf falsche Sites gelockt werden? Wie leicht ist die Kommunikation mit den Kunden durch Fremde zu unterlaufen bzw. nachzuahmen?

2.2 ebay

Als weiteres Beispiel für ein weithin genutztes Portal betrachten wir in diesem Abschnitt ebay, den weltweit führenden Anbieter von Internet-Auktionen. ebay ist für uns vor allem interessant als Beispiel für ein ausgedehntes Peer-to-Peer-Netzwerk, welches Käufer und Verkäufer unter Zuhilfenahme der ebay-Plattform



Abb. 2.3 Einstiegsseite des ebay-Portals

bilden. ebay übernimmt hierbei keine Verantwortung für Qualität, Sicherheit oder Rechtmäßigkeit der vermittelten Transaktionen (allerdings gibt es in letzter Zeit Versuche auf Seiten von ebay eine gewisse Garantie bei Transaktionen zu gewähren, jedoch nur bis zu einer gewissen Grenze). Bei ebay interessiert uns weniger die hinter dem Portal liegende Architektur, vielmehr werden wir uns dem Portal sehr stark aus Kundensicht nähern, die mit dem Portal und seinem Geschäftsmodell verbundenen Sicherheitsproblematiken herausarbeiten und die Lösungen, die ebay dafür anbietet, analysieren.

2.2.1 Übersicht und Geschäftsmodell

ebay fungiert als reine Verkaufsplattform für seine Mitglieder und bietet selbst keine Artikel an. ebay wird auch selbst nicht Vertragspartner der ausschließlich zwischen den Mitgliedern geschlossenen Verträge, wie sie bei einer erfolgreichen Auktion zustande kommen. Die Erfüllung dieser über die ebay-Website geschlossenen Verträge erfolgt ausschließlich zwischen den Mitgliedern. Die Mitgliedschaft bei ebay ist kostenlos, das Geschäftsmodell von ebay besteht im Erheben einer Angebotsgebühr für eingestellte Artikel und einer Verkaufsgebühr, falls der Artikel erfolgreich verkauft wurde. Zudem fungiert ebay mittlerweile ähnlich wie

bahn.de als Vermittler zu zusätzlichen Anbietern, wie einem Dienstleister für Verpackung und Versand oder einem Kreditunternehmen.

2.2.2 *Kundensicht*

Versetzen wir uns einmal in die Lage eines Kunden, der zum ersten Mal den ebay-Server besucht und vollziehen wir die einzelnen Schritte nach, die zum Bieten, Kaufen und Versteigern von Artikeln nötig sind.

2.2.2.1 *Registrierung*

Wenngleich ebay, anders als bahn.de, zumindest mit den nur als Bieter in Erscheinung tretenden Kunden keinerlei vertragliche Bindungen eingeht, so fordert doch das Geschäftsmodell eine verlässliche und sichere Registrierungs- und Verifikationsprozedur. Würden sich die Fälle falscher Identitäten von Käufern oder Verkäufern häufen, so verlören die Kunden das Vertrauen in die Plattform und würden sich anderen Anbietern zuwenden.

Deshalb steht am Anfang ein Registrierungsprozess, der analog zu der bei bahn.de durchgeführten verläuft: Über eine über vom Sicherheitsprotokoll SSL hergestellten sicheren Tunnel zwischen ebay-Server und Kundenbrowser werden postalische Adresse und Telefonnummer sowie die E-Mail-Adresse des Neukunden abgefragt. Zudem wählt der User selbst einen Mitgliedsnamen und ein Passwort. ebay empfiehlt eine Mindestlänge des Passworts von sechs Zeichen, erzwingt dies aber nicht bei der Registrierung. Erhielt man früher nach der Registrierung lediglich eine E-Mail mit einer URL, über die das Mitgliedskonto frei geschaltet wurde, so erfolgt heutzutage zumindest eine Verifikation der postalischen Adresse über einen Brief, der ein Anfangspasswort enthält. Allerdings entfällt diese Verifikation per Brief, wenn die angegebene E-Mail-Adresse von einem größeren Provider wie t-online, aol oder msn stammt. In diesen Fällen verlässt sich ebay auf die voran gegangene Verifikation der Anschrift durch den E-Mail-Provider.

2.2.2.2 *Anmeldung*

Ein registrierter ebay-Nutzer meldet sich bei dem Server signin.ebay.de über seinen Username und sein Passwort an. Die Anmeldung ist sowohl für das Abgeben von Geboten als auch für die Nutzung personalisierter Sichten („Mein ebay“) auf das ebay-Angebot erforderlich. Die Anmeldung erfolgt standardmäßig über eine mit SSL gesicherte Verbindung – allerdings erst seit 2005.

Über die Gründe dafür, dass ebay die sichere Anmeldung via SSL, die bei anderen Portalen längst obligatorisch ist, erst so spät als Standard angeboten hat, kann nur

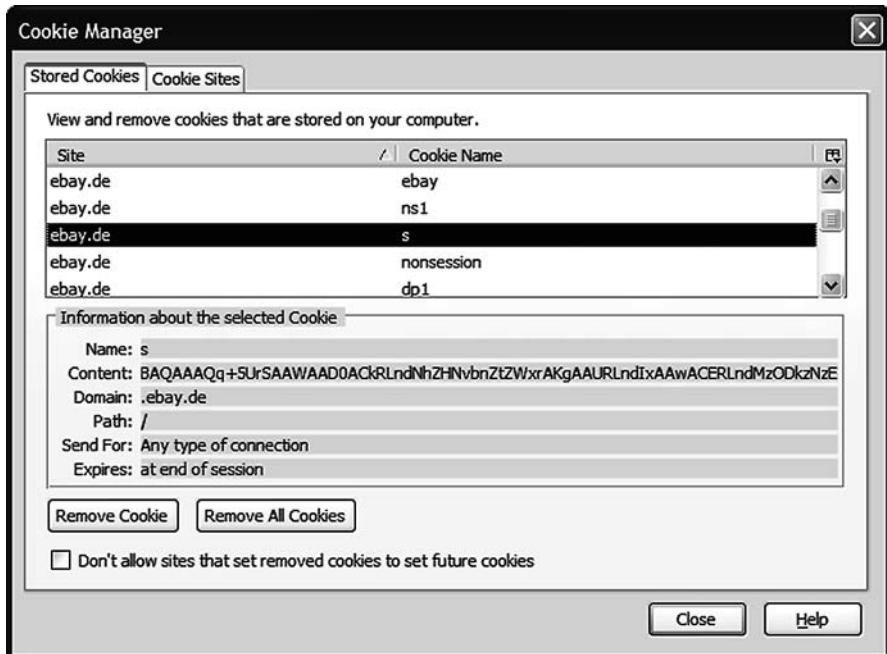


Abb. 2.4 Session Cookie zur Wiedererkennung authentisierter Nutzer bei ebay.de

spekuliert werden. Vermutlich scheute der Betreiber die hohe Rechenlast, die die massenhaften Anmeldungen über SSL für seine Server bedeutet, und die im Zusammenhang mit Load-Balancing-Schemata und SSL auftretenden Probleme¹. Nach der Anmeldung mit Username und Passwort überträgt der für die Einwahl zuständige Server signin.ebay.de an den Browser des Users eine große Anzahl von Cookies. Wie einfacher Test mit dem Cookie Manager des Mozilla-Browsers zeigt, erfolgt danach die Wiedererkennung des Users über eine durch einen Session-Cookie mit dem Namen „s“ übermittelte temporäre Identität (siehe Abb. 2.4), die bei Beenden des Browsers wieder vergessen wird (das heißt, der Session Cookie wird gelöscht).

Nach Beendigung der Browser-Session muss sich also der User erneut mit Username und Passwort authentifizieren. Wählt der User jedoch beim Einloggen die Option „Auf diesem Computer eingeloggt bleiben“, werden zur Authentifikation des Users permanente Cookies verwendet, die eine Gültigkeitsdauer von einem

¹ Dass es ebay überdies auch schon früher bei der SSL-Konfiguration seines Servers nicht allzu genau nahm, zeigt ein Vorfall aus dem September 2004, als auf dem Login-Server signin.ebay.de ein SSL-Zertifikat installiert wurde, das auf den Namen signin.ebay.com.au ausgestellt war. Der Browser reagierte darauf mit einer entsprechenden Fehlermeldung („Domain Name Mismatch“), die ein User, der ebay tatsächlich nutzen wollte, in diesem Fall ignorieren musste. Dazu zitieren wir den Heise-Newsletter: „Gewöhnen sich die Mitglieder an diese Warnung, fallen sie leichter den zahlreichen Versendern von Phishing-Mails zum Opfer, die sie auf gefälschte Anmeldeseiten im ebay-Design leiten, um ihnen dort Benutzernamen und Passwort zu entlocken.“ ([heise50569])

Jahr besitzen. Kann sich also ein Angreifer in den Besitz der permanenten Identität wie sie etwa im „nonsession“ Cookie zu finden ist, bringen, erhält er so lange Zugang auf die personalisierten Seiten von ebay, bis sich der rechtmäßige User explizit ausloggt. Die „eingeloggt bleiben“ Option stellt deshalb in öffentlichen Client-Umgebungen wie Kiosken oder Internet-Cafés ein großes Sicherheitsproblem dar.

Eine Alternative bei der Anmeldung bei ebay bot bis vor kurzem der zentrale Web-basierte Authentikations-Dienst Microsoft Passport. Dabei handelt es sich um einen Authentikations-Dienst, bei dem ein User über seine E-Mail Adresse und ein Passwort eine eindeutige Kennung erzeugt, über die er sich bei allen an Passport teilnehmenden Websites und Diensten authentifizieren kann. Mittlerweile wird MS Passport aber nicht mehr von ebay unterstützt (vergleiche hierzu auch das Kapitel „Föderative Sicherheit“).

2.2.2.3 Bieten

Bietet ein bereits angemeldeter Nutzer auf einen Artikel, müssen zur Bestätigung des Gebots nicht erneut Username und Passwort eingegeben werden – ein einfacher Mausklick genügt zur Bestätigung des Gebots. Ist der Nutzer noch nicht angemeldet, so wird er vorab auf die Login-Seite geleitet.

Mit dem Abgeben eines Gebots ist implizit eine Willenserklärung des Users verbunden, den Artikel zu dem genannten Preis auch tatsächlich erwerben zu wollen. Ist das Gebot erfolgreich, so muss der Verkäufer davon ausgehen, dass er den entsprechenden Betrag auch erhält – ebenso wie der erfolgreiche Bieter den Artikel für diesen Betrag auch erhalten möchte. Gefragt ist also eine beiderseitige Nichtabstreitbarkeit (*Nonrepudiation*) – auf Verkäuferseite des Angebots, auf Bieterseite des Gebots. Zumindest auf Seiten des Käufers kann aber von einer Nichtabstreitbarkeit des Gebots keine Rede sein – nicht umsonst macht in der ebay-Community der Begriff des „Spaßbieters“ die Runde. Das Problem liegt nicht allein in der zweifelhaften Verifikation der Identität der Bieter, sondern auch in der mangelhaften, nämlich abstreitbaren Bestätigung des Gebots über ein Passwort. Leicht zu merkende (das heißt in der Regel auch leicht zu erratende) Passwörter bilden nicht nur eine Gefahr für den User, sondern können im Ernstfall auch als Ausrede für ein nicht ernst gemeintes Gebot genutzt werden.

2.2.2.4 Verkaufen

Für die Registrierung als Verkäufer ist eine eigene Prozedur erforderlich, bei der die Adress- und Kontodaten des angehenden Verkäufers über eine SSL-Verbindung übermittelt werden. Danach wird ein Datenabgleich mit der SCHUFA zur Verifizierung der Identität durchgeführt. Zudem müssen Kontodaten für den Verkäufer hinterlegt werden.

Als zusätzliche „vertrauensbildende Maßnahme“ können Verkäufer freiwillig am so genannten PostIdent-Verfahren teilnehmen, bei dem ihre Identität durch die

Deutsche Post anhand der Ausweispapiere verifiziert wird. Solcherart ausgewiesene Mitglieder dürfen sich „geprüftes Mitglied“ nennen und auf einen Vertrauensvorschuss bei den potenziellen Käufern hoffen.

2.2.3 Spezielle Sicherheitsanforderungen

Über die in diesem Abschnitt und in der Diskussion des bahn.de Portals bereits diskutierten allgemeinen Sicherheitsanforderungen an ein Portal hinaus bringt das Geschäftsmodell von ebay noch einige weitere, für das Kaufen und Verkaufen im Peer-to-Peer-Umfeld spezielle Sicherheitsanforderungen mit sich, die wir nun diskutieren wollen.

2.2.3.1 Vertrauensaufbau

Das Kaufen und Verkaufen von Waren erfordert ein gewisses Vertrauensverhältnis: Wird der Kunde tatsächlich zahlen? Hält die angebotene Ware das, was der Verkäufer verspricht? Wird man sie wirklich zu dem angebotenen Preis erwerben können? Im täglichen Leben lässt sich das Vertrauensproblem häufig (aber nicht immer) durch persönlichen Kontakt lösen: Man kennt sich bereits und hat schon früher Geschäfte zur beiderseitigen Zufriedenheit abgewickelt. Oder man hat einen vertrauenswürdigen Bekannten, der diesen oder jenen Händler empfehlen kann. Schließlich und endlich entscheidet oft auch der persönliche Eindruck darüber, ob man sich auf ein Geschäft einlässt.

Im Internet entfällt dieser persönliche Kontakt. Es ist noch nicht einmal klar, mit wem man es überhaupt zu tun hat. Große E-Commerce-Anbieter lösen, wie bereits am Beispiel bahn.de diskutiert, zumindest das Problem der Verifikation der Identität des Servers über ein Zertifikat, das von einer *Certification Authority* (abgekürzt CA) ausgestellt wird. Auch der Server ebay.de identifiziert sich mit Hilfe eines Zertifikats, das im Rahmen des SSL-Protokolls an den Browser übertragen wird. Trotzdem liegt bei ebay die Problematik etwas anders: ebay fungiert ja lediglich als Plattform, die Käufer und Verkäufer zusammenbringen möchte. Das eigentliche Vertrauensverhältnis muss aber zwischen Käufern und Verkäufern etabliert werden, beide gleichberechtigte Nutzer (Peers) des Systems. Selbst wenn jeder dieser User selbst über ein Zertifikat verfügen würde, würde das noch nicht das Vertrauensproblem lösen, da das Zertifikat nur die Identität verifiziert (die ja auch in einem Pseudonym bestehen kann), jedoch nichts über die Vertrauenswürdigkeit der dazu gehörigen Person aussagt. ebay löst das Problem über so genannte Bewertungsprofile. Hierbei kann jedes Mitglied nach abgewickelter Transaktion von dem Geschäftspartner positiv, neutral oder negativ bewertet werden. Bei jedem angebotenen Artikel werden zusammen mit dem Username des Verkäufers die Differenz zwischen der Anzahl von positiven und negativen Bewertungen sowie der prozentuale Anteil positiver Bewertungen angezeigt. Analoge Bewer-

tungen gibt es auch beim Kauf von Artikeln. Auf diese Weise erzeugt die ebay-Gemeinschaft im Idealfall ihre eigene Reputationshierarchie.

Leider hat diese Lösung einige Schwächen. Offensichtlich ist die Möglichkeit, durch mehrfache Registrierung unter unterschiedlichen Pseudonymen und Scheinkäufe für sich selbst positive Bewertungen zu kreieren. Des Weiteren können Freunde und Bekannte zu positiven Bewertungen bei evtl. gar nicht stattgefundenen Transaktionen animiert werden. Eine andere Strategie zur Erlangung einer hohen Reputation besteht darin, zunächst eine große Anzahl von Artikeln mit relativ geringem Wert zum Kauf anzubieten und diese Transaktionen ordnungsgemäß durchzuführen. Ist die gewünschte Reputation erreicht, bietet man ein hochwertiges Objekt zu einem günstigen Preis gegen Vorkasse an, ohne dann tatsächlich zu liefern. Das Problem besteht hier darin, dass sich der Bewertungsmaßstab nicht nach der Höhe der Transaktionen, sondern lediglich nach deren Anzahl bemisst. Aber auch ohne diese Schwäche bleibt das eigentliche Problem, nämlich die Etablierung von Vertrauen zwischen Unbekannten ohne eine zentrale, vertrauenswürdige Instanz, ungelöst.

2.2.3.2 Sichere Zahlungsmethoden

Obwohl ebay selbst nicht an den Zahlungsvorgängen der Mitglieder untereinander beteiligt ist, sollte auch in diesem Bereich ein genuines Interesse des Portals an einer sicheren Zahlungsabwicklung bestehen. Typischerweise nehmen nach erfolgter Auktion Käufer und Verkäufer per E-Mail miteinander Kontakt auf und handeln die Zahlungsmodalitäten selbst aus. Die entsprechenden Adressen bekommen die Beteiligten von ebay mitgeteilt. Optional kann auch ein Verkäufer seine Kontodaten bei ebay hinterlegen und den erfolgreichen Bieter dort zugänglich machen. In jedem Fall ist aber davon auszugehen, dass während des Bezahlvorgangs hoch sensitive Kontoinformationen auf ungeschützten Kanälen über das Internet übertragen werden. Aus eventuell auftretenden Streitigkeiten hält sich ebay weitestgehend heraus und verweist auf die Eigenverantwortung der Mitglieder. Handelt es sich um höhere Beträge, können Treuhand-Dienste in Anspruch genommen werden, wobei der Kaufpreis zunächst auf ein Treuhandkonto überwiesen wird und erst bei erfolgter Warenlieferung frei gegeben wird.

Um selbst von den bei der Nutzung von Treuhandservices fälligen Gebühren profitieren zu können, unterhält ebay ein eigenes Subunternehmen namens PayPal, welches sich selbst als „Anbieter für elektronischen Zahlungsverkehr“ bezeichnet. Die Käufer unterhalten ein virtuelles Konto bei PayPal, von dem sie bestimmte Beträge für die Anbieter ersteigerter Waren freigeben. Daraufhin wird der freigegebene Betrag vom angegebenen Konto abgebucht und an den Verkäufer weiter gegeben (s. Abb. 2.5).

PayPal fungiert also lediglich als eine für beide Seiten vertrauenswürdige Instanz (eine so genannte *Trusted Third Party* (TTP)), die die jeweiligen Kontoinformationen vor dem Geschäftspartner verbirgt.

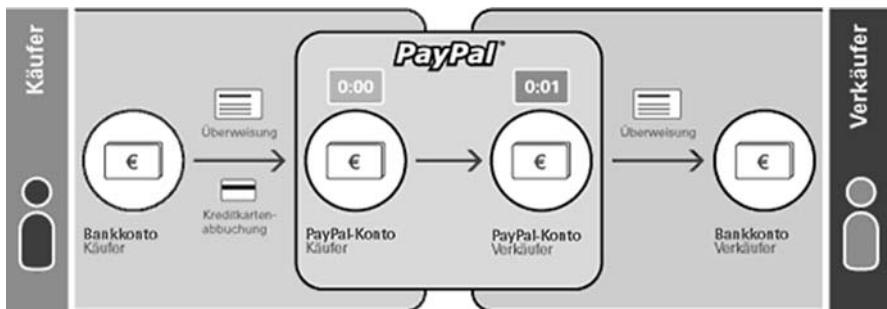


Abb. 2.5 Schematischer Ablauf einer PayPal-Transaktion (Quelle: paypal.de)

2.2.3.3 Sicherheit der Kunden-Credentials

Eine Besonderheit des ebay-Portals stellt die Tatsache dar, dass die Login-Namen der Nutzer im Klartext für alle Besucher des Portals sichtbar sind. Somit liegt für einen Angreifer quasi bereits eine Hälfte der Credentials offen. Die andere Hälfte – also das Passwort – lässt sich dann bei schlechter Qualität durch Raten oder durch eine so genannte *Wörterbuch-Attacke* (Dictionary Attack) ermitteln. ebay bietet jedoch (optional) an, die Güte eines selbst gewählten Passworts online zu prüfen.

Da ebay den Anbietern erlaubt, im Rahmen der Artikelbeschreibungen und „Shop-Seiten“ eigene Inhalte auf den ebay-Servern zu platzieren, muss besondere Sorge getragen werden, dass diese Inhalte die Sicherheit der anderen Nutzer (und der ebay-Infrastruktur) nicht gefährden. Leider schien dies lange Zeit nicht zu geschehen. Zwar verbietet ebay in seinem „Überarbeiteten Grundsatz zur Verwendung von Skriptsprachen beim Anbieten von Artikeln“ (s. [ebay]) unter anderem die Verwendung von Skripten, die Cookies setzen bzw. auslesen oder den User zu anderen Servern weiterleiten können, darüber hinaus schien jedoch keine tiefer gehende inhaltliche Analyse der auf den ebay-Seiten platzierten Inhalte zu erfolgen. Dies führte im Jahr 2004 zu einer Reihe von so genannten *Cross-Site-Scripting* Angriffen, bei denen nichtsahnenden Nutzern ein scheinbar authentischer ebay-Login-Bereich präsentiert wurde, der sich in Wirklichkeit auf dem Server eines Angreifers befand ([Koss]). Mittlerweile schreibt ebay seinen Usern die Nutzung vorgefertigter JavaScript-Bausteine vor.

2.2.3.4 Geschützte Kommunikation Kunde – Portal

ebay verkehrt mit seinen Kunden ausschließlich per E-Mail, zumeist zu Werbezwecken. Seit langem machen sich dies Phisher zu Nutze und generieren massenweise eigene, scheinbar authentische Mails wie die in Abb. 2.6, in denen zum Login bei einer scheinbar authentischen Webseite aufgefordert wird, die aber in Wirklichkeit zum Server eines Angreifers führt.

Subject: ***SPAM*** Official Information To All eBay Clients [Fri, 30 Dec 2005 19:14:06 +0500]
From: eBay Inc <support_num_448211@ebay.com>
Date: 15:17
To: professoren@hdm-stuttgart.de



Dear eBay Member,

We regret to inform you that your eBay account could be suspended if you don't re-update your account information.

To resolve this problem please visit link below and re-enter your account information:

https://signin.ebay.com/ws/eBayISAPI.dll?SignIn&sid=verify&co_partnerId=2&siteid=0

If your problems could not be resolved your account will be suspended for a period of 24 hours, after this period your account will be terminated.

For the User Agreement, Section 9, we may immediately issue a warning, temporarily suspend, indefinitely suspend or terminate your membership and refuse to provide our services to you if we believe that your actions may cause financial loss or legal liability for you, our users or us. We may also take these actions if we are unable to verify or authenticate any information you provide to us.

Due to the suspension of this account, please be advised you are prohibited from using eBay in any way. This includes the registering of a new account. Please note that this suspension does not relieve you of your agreed-upon obligation to pay any fees you may owe to eBay.

Regards,
Safeharbor Department eBay, Inc
The eBay team

Abb. 2.6 Phishing-Mail, die von ebay zu kommen scheint

Ebay reagiert auf diese Bedrohung, indem es in eigenen Mails niemals vertrauliche Daten der Nutzer einfordert, und – wie viele andere Portale auch, etwa im Bankenumfeld – mit dem Versuch, die Nutzer über Phishing-Mails und ihre Gefahren aufzuklären.

Kapitel 3

Sicherheitskonzepte und Analysen

In diesem Kapitel stehen Methoden im Vordergrund, mit denen Risiken geschätzt und der Sicherheitskontext der Anwendung erstellt werden kann. Dazu gehören eine Risikoanalyse der Geschäftsmodelle sowie die Erfassung der Applikationskomponenten und der Infrastruktur (d. h. ihre Vernetzung mit Internet und Intranet Services). Die Risikobetrachtung umfasst also sowohl eine eher funktionale Sicht (die wirtschaftliche und rechtliche Rolle von Sicherheit in der geplanten Applikation) als auch eine eher nicht-funktionale Betrachtungsweise der Sicherheit im Kontext des technischen Umfelds der Applikation, also die klassische IT-Security.

Im Idealfall besitzt die betreibende Firma bereits eine detaillierte und schriftlich fixierte Sicherheitsrichtlinie, in der die Vorgaben für den Umgang mit Daten und Diensten festgelegt sind. Aus dieser werden dann konkrete technische Maßnahmen abgeleitet und gerechtfertigt. Zur Erstellung einer solchen Policy sowie dem grundsätzlichen Vorgehen dabei gibt es bereits ausführliche Literatur (z. B. das BSI-Grundschutzhandbuch), deren Problem für Softwareentwickler wohl eher darin liegt, viel zu umfangreich zu sein als dass sie für die eigene Arbeit direkt als nützlich empfunden würde. Die folgenden Seiten beschränken sich daher auf das, was wir als das Allernötigste für den Entwickler erachten.

3.1 Security Policies

Als erster Schritt – also vor der Klassifikation der Daten, dem Erstellen von Bedrohungsmodellen und der Risikoabschätzung – erfolgt das Erstellen einer Sicherheitsrichtlinie (*Security Policy*) für das Portal bzw. eine ganze Firma. Formal gesprochen, definiert eine Security Policy, welche Zustände des Gesamtsystems erlaubt und welche verboten sind. Eine Verletzung der Sicherheitsrichtlinie liegt vor, wenn das System einen verbotenen Zustand annimmt (zum Beispiel ein Kunde kann die Daten anderer Kunden einsehen).

Für real existierende, komplexe Systeme ist diese Definition aber deutlich zu abstrakt, um tatsächlich brauchbar zu sein. Etwas weniger formal gesprochen, beschreibt eine Security Policy die Sicherheitsanforderungen an das System (etwa in

der Form, wie in unserer Diskussion von bahn.de), ohne jedoch konkret vorzugeben, wie diese umgesetzt werden sollten. Dies ist auch sinnvoll, um nicht bei jedem Technologiewechsel oder neuem veröffentlichten Angriff auf ein konkretes Sicherheitsprotokoll eine neue Policy erstellen zu müssen. Zudem ist die Security Policy für ein Portal gerade in großen Firmen nur als Teil einer übergeordneten Policy für die gesamte Firma zu sehen, die auch physikalische Sicherheitsanforderungen wie zum Beispiel Zutrittskontrollmaßnahmen zu Serverräumen umfassen kann. Es ist wichtig, dass Entwickler hier nicht im luftleeren Raum arbeiten, sondern die Gesamtpolicy ihrer Firma kennen und mit den Verantwortlichen in Kontakt stehen.

Für die konzeptuelle Arbeit unumgänglich sind folgende Richtlinien einer Security Policy:

- *Mandatory Data Ownership*: Das Prinzip, dass alle Daten und Dienste einen Eigentümer haben, der die Vergabe von Zugriffsrechten und -arten kontrolliert.
- *Mandatory Data Classification*: Die Klassifikation der Daten und Dienste in Sicherheitskategorien, an die Regeln für den Umgang mit den darin enthaltenen Daten und Diensten geknüpft sind.
- Die Verpflichtung aller Zugriffe auf Authentisierung, Autorisierung und Nachvollziehbarkeit.
- Die Verpflichtung zur sicherheitstechnischen Abnahme der Software-Architektur und anschließend auch der Implementation.

Diese Regeln geben den Entwicklern einen Sicherheitsrahmen vor, der während der Entwicklung einzuhalten ist. Sie regeln auch, wer für die Backend-Zugriffe des Portals zuständig ist und wie vertraulich bestimmte Daten zu behandeln sind.

3.2 Allgemeine Vorgehensweise

Für die Erstellung von Sicherheitsanalysen bzw. Sicherheitskonzepten im Umfeld eines Portals sind drei Aufstellungen unabdingbar:

- die wichtigsten Geschäftsvorgänge (interne wie externe),
- der Portalkontext (das heißt, der Bezug des Portals zu externen und internen Systemen),
- und das Modell der internen Software-Komponenten, über die die internen Geschäftsvorgänge abgewickelt werden.

Alle drei Aufstellungen sind normalerweise ohnehin Teil eines ordentlichen System-Engineerings und werden hier nur für die Abschätzung der Sicherheitsrisiken verwendet. Auf den Kontext und das Komponentenmodell gehen wir weiter unten gesondert ein.

Für den Entwickler ist es jedoch häufig schwierig, Sicherheitsprobleme in einem komplexen System zu entdecken und einzuschätzen. Die nachfolgende allgemeine Vorgehensweise hat sich in der Praxis bewährt:

- Risikoanalysen und Abschätzungen;
- Erstellung eines Security Contexts;
- Erstellung einer Basisarchitektur, die die zentralen Datenflüsse sowie die beteiligten Komponenten und Systeme zeigt;
- Aufstellen von Bedrohungsmodellen für die Seiten Client, Transport und Server

In diesem Kapitel werden wir diese allgemeine Vorgehensweise nachvollziehen und versuchen, sie transparent zu machen.

3.2.1 Risikoanalyse der Geschäftsvorgänge

Eine Risikoabschätzung beruht in ihrer einfachsten Form auf drei Merkmalen pro Risiko: Art, Häufigkeit und Konsequenz. Aus den quantitativen Merkmalen Häufigkeit (Wahrscheinlichkeit) und Konsequenz (finanzialer Schaden) lässt sich dann die finanzielle Dimension des Risikos bestimmen. Ein solches Vorgehen ist Standard in größeren Projekten und dient nicht nur der Abschätzung von Sicherheitsrisiken, sondern lässt auch die Risiken bei der Entwicklung von Komponenten, bei der Zulieferung von Teilen etc. sichtbar werden.

Das Ergebnis ist ein einfaches Diagramm mit vier Quadranten, die für vier unterschiedliche Typen von Risiken stehen (s. Abb. 3.1). Der erste Quadrant unten links beinhaltet seltene und harmlose Risiken und stellt daher kein Problem dar. Ähnliches gilt für den Quadranten links oben, der zwar häufigere aber dennoch harmlose Risiken darstellt. Auch der Quadrant rechts oben ist recht einfach einzuschätzen: Er steht für häufig auftretende und gleichzeitig fatale Risiken, die deshalb nicht akzeptabel sind. Der letzte Quadrant rechts unten ist ebenfalls leicht einzuschätzen: Es handelt sich hier um häufige und relativ harmlose Risiken, die ebenfalls nicht akzeptabel sind.

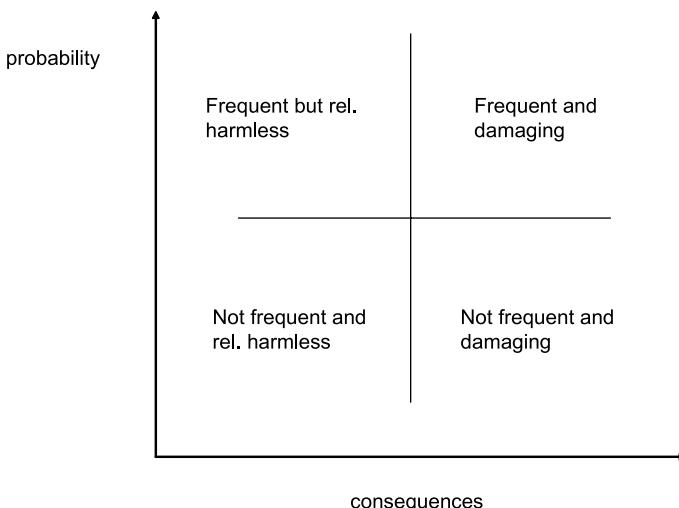


Abb. 3.1 Die vier grundsätzlichen Risikotypen

Abb. 3.2 FCRA für einige der Risiken beim Portal bahn.de

Threat	Frequency	Damage	Result
Forged tickets	Rare	small	Accept risc
Stolen Cards	More often	Small	Accept risc
DOS Attack on Service	Frequent	Small	Accept risc

tabel sind. Der Quadrant unten rechts zeigt seltene, aber fatale Risiken, deren Einschätzung deshalb besonders schwer ist. Mehr dazu weiter unten.

Diese erste Analyse, bei der die identifizierten Risiken in einen der vier Quadranten eingeordnet werden, wird auch als *First Cut Risk Analysis* (FCRA) bezeichnet. Die FCRA dient dazu, die Risiken für das Geschäft zu bewerten und eventuell untragbare Projektrisiken gleich am Anfang des Entwicklungsprozesses sichtbar zu machen. Dabei sollten sowohl Angriffe als auch eventuelle Probleme mit der Verfügbarkeit des Portals abgeschätzt werden. Theoretisch sollte diese Analyse von den Business-Spezialisten durchgeführt werden, doch hat sich gezeigt, dass eine enge Zusammenarbeit mit den Entwicklern hier sehr wichtig ist. Das Ergebnis lässt sich in einer einfachen Tabelle festhalten und macht letztlich allen Beteiligten klar, welche Risiken existieren und ob sie akzeptiert werden – natürlich nach entsprechenden Maßnahmen zur Eindämmung der Häufigkeit oder Konsequenzen (s. Abb. 3.2). Dadurch wird das unvermeidliche Restrisiko sichtbar und es ist eine Entscheidung des Business, dieses zu tragen oder abzulehnen. Bei Ablehnung fällt natürlich das Projekt.

Da es hier nur um eine geschäftliche Abschätzung des Gesamtrisikos des Portalprojekts geht, reichen diese drei Merkmale durchaus aus. Später – wenn die technische Basis klar ist – kann jede Zeile durch weitere Merkmale verfeinert werden, zum Beispiel welche Fähigkeiten für einen Angriff nötig wären, wie groß das Risiko des Angreifers, ertappt zu werden wäre, sein potenzieller Gewinn bei der Aktion etc. Dadurch lässt sich meist die Wahrscheinlichkeit einer speziellen Gefahr besser abschätzen. Diese genauere Analyse dient dann häufig dazu, die Abwehrmaßnahmen nach Priorität zu sortieren, wie Abb. 3.3 zeigt.

Diese Tabellen sind relativ einfach zu erstellen und decken doch einen wichtigen Teil der Beurteilung eines Systems ab. Sie können beliebig verfeinert werden, z. B. in Form der von Bruce Schneier eingeführten „Attack Trees“ (s. [Schn99]). Dort werden die definierten Attacken in ihre Teilschritte zerlegt und deren Aufwand, Gefahr und Wahrscheinlichkeit bewertet pro Schritt. Außerdem besteht die Möglichkeit die Attacken von links nach rechts zeitlich so anzutragen, dass sie den Lebenszyklus des Projektes wiedergeben. So können Attacken auf die Sicherheit einer Smartcard, auf der geheime Schlüssel gespeichert sind, bereits bei der

Abb. 3.3 Priorisierung von Abwehrmaßnahmen am Beispiel bahn.de

Attack	Attacker	Difficulty	Gain	Risc	Priority
Forged ticket	anybody	easy	Small	high	low
DOS	Script kiddy	Easy	None/ fame	High	high
Stolen ID	Professional	Medium	Small	medium	medium

Produktion der Hardware (CPU, RAM, etc.) beginnen. Sie schreiten fort mit dem Erzeugen und Laden der Schlüssel, mit evtl. Updates von Software und enden schließlich mit der sicheren Entsorgung der Geräte. Es ist meistens die fehlende Betrachtung des Lebenszyklusses einer Lösung, die am Anfang sowie am Ende grobe Sicherheitslücken aufreißt. Beispiele dafür sind PCs, die am Ende ihrer Nützlichkeit zerlegt und in Teilen verkauft werden – inklusive der Festplatten, die immer noch vertrauliche Daten enthalten!

Unser Diagramm in Abb. 3.4 zeigt einen rudimentären Attack Tree für die Portal Security von bahn.de.

Derartige Analysen beziehen sich auf die Gesamtsicherheit einer Lösung, aber meist nur zu einem geringen Teil auf die Software-Sicherheit des Systems. Sie werden meist von IT-Security Spezialisten erstellt. Für die Entwickler der Software – und diese sind ja im Blickpunkt dieses Buches – sind sie meist zu weit von der konkreten Architektur einer Lösung entfernt. Entwickler benötigen eher eine

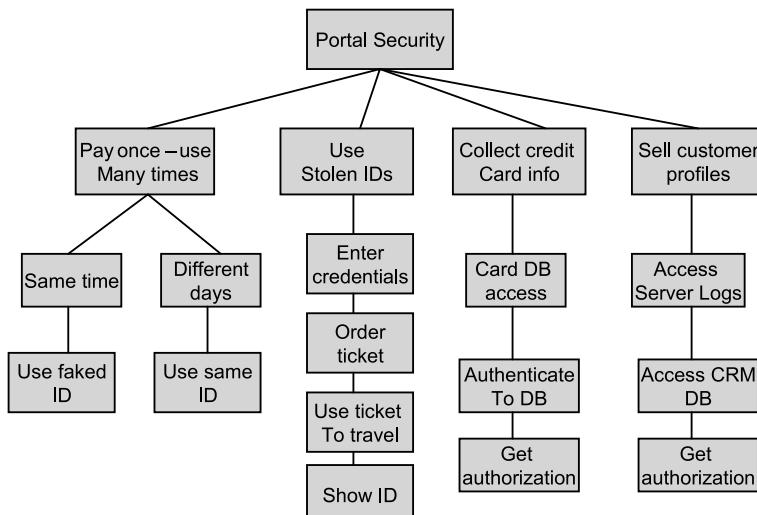


Abb. 3.4 Attack Tree für bahn.de

Aufstellung von Sicherheitskontexten, damit sie Risiken besser erkennen und einschätzen können.

3.2.2 Security Context

Im Laufe der Architekturplanung und der Implementation einer Applikation geht der Fokus der Sicherheitsanalyse langsam über von den Risiken der Geschäftsvorgänge auf die Risiken die in der technischen Infrastruktur (Vernetzung, Rechner, Software) vorhanden sind. Auch werden die konkreten Interaktionen aller Teilnehmer wichtiger.

Für das Verständnis eines solchen Gesamt-Systems ist oft das so genannte *System Context Diagram* sehr wichtig. Es zeigt die Verbindungen des Systems zu seiner Umwelt auf und spielt in den meisten Vorgehensmodellen für die Entwicklung von Software-Lösungen eine zentrale Rolle (Die Terminologie hier orientiert sich grob an der IBM Global Services Method (GSM), die auch in der Informatik-Ausbildung an der HdM eingesetzt wird). Im Fall des Portals wird ein solches Diagramm die hauptsächlichen Partner und Benutzer aufweisen. Typischerweise wird dabei die innere Struktur der Lösung nicht beachtet.

Im Portalbeispiel zeigt der Kontext alle beteiligten Systeme, mit denen das Portal interagiert. Aus diesem Diagramm lässt sich nun ein Security Context Diagram entwickeln, indem man sicherheitsrelevante Annotationen einfügt und es eventuell

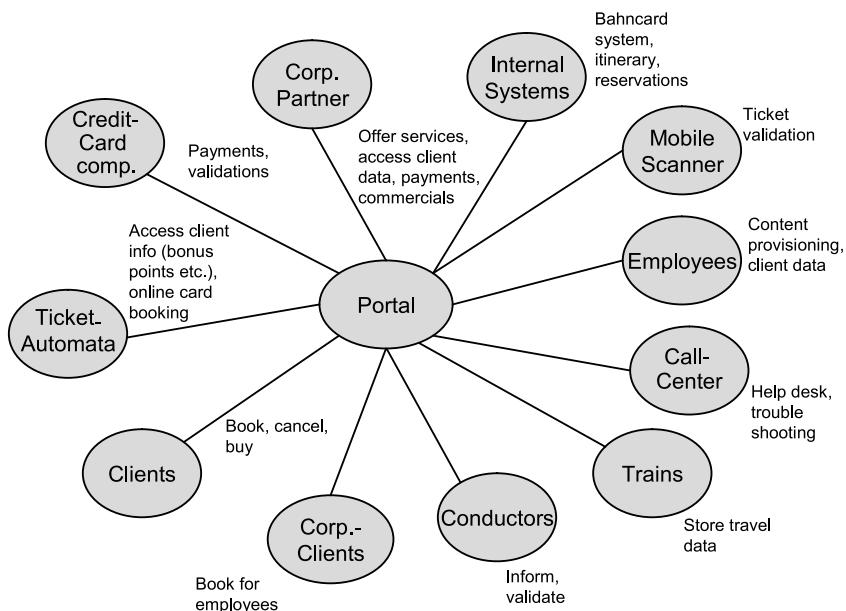


Abb. 3.5 System Context Diagram für bahn.de

auch um besonders wichtige Komponenten der Lösung anreichert. Ziel des Security Context Diagrams ist es, sich einen Überblick über die möglichen Sicherheitsprobleme zu verschaffen, ohne diese jedoch detailliert zu beschreiben. Auch lassen sich daran später die nötigen Bedrohungsmodele festmachen (siehe unten).

Entscheidend für die Sicherheitsanalyse ist nun die Frage, in welcher Tiefe diese Systeme einbezogen werden. Wird zum Beispiel nur der Browser des Kunden als Systempartner gesehen oder auch der Kunde selbst? In die Entscheidung darüber fließt sicher ein, ob jemand den Weg Kunde-Browser bereits kritisch sieht (etwa auf Grund eines möglichen Befalls des Kunden-PCs mit Malware) oder ob der Brower grundsätzlich als vertrauenswürdiger Stellvertreter des Kunden gesehen wird. Das heißt letzten Endes, welches *Bedrohungsmodell* wird der Analyse zu Grunde gelegt? Traditionell wurde für Web-Applikationen gerne das sog. Internet-Bedrohungsmodell verwendet, bei dem der Schwerpunkt der Sicherheitsprobleme im Übertragungsweg der Daten gesehen wird – also ohne Betrachtung der Plattform-Sicherheit oder der beteiligten Personen und nachgelagerten Prozesse. Bei diesem sehr eingeschränkten Bedrohungsmodell entgehen den Entwicklern sehr viele andere Formen von Sicherheitsproblemen, die erst durch den Gesamtkontext der Applikation sichtbar werden. Zunächst aber benötigen wir eine Übersicht über die Gesamtarchitektur des Systems.

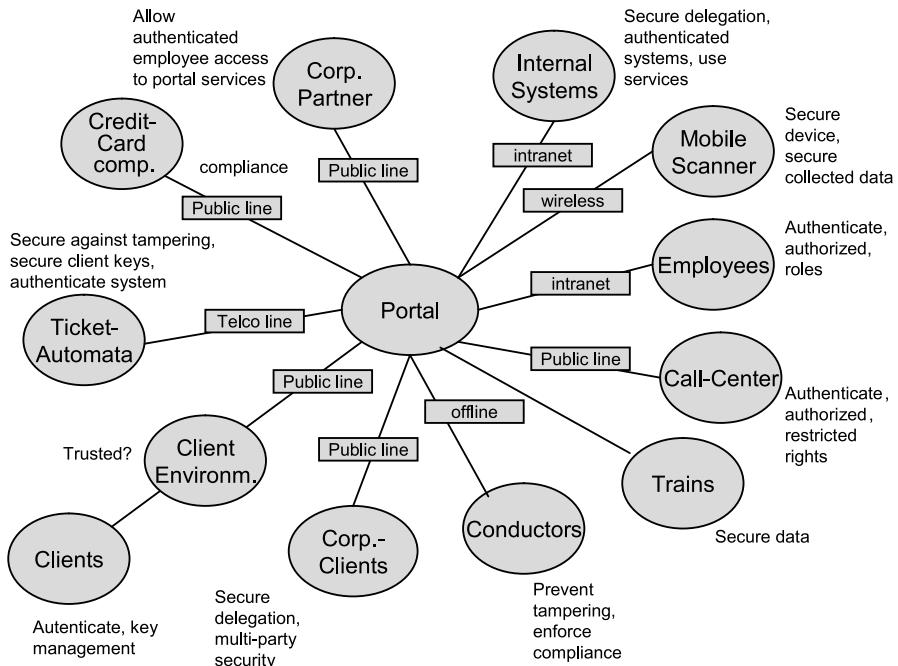


Abb. 3.6 Security Context Diagram für bahn.de

3.2.3 Basisarchitektur

Sinn einer Basisarchitektur ist es, den Blick auf das Wesentliche einer Systemarchitektur zu lenken. Konkrete Details wie spezielle Softwareversionen stören hier nur. Die wesentlichen Elemente einer Basisarchitektur sind:

- ein Diagramm der Knoten und Verbindungen unter Einschluss ihrer Funktion,
- ein Diagramm der Aufteilung der Knoten und Verbindungen in unterschiedliche Schutzzonen,
- Software-Komponenten und Services.

3.2.3.1 Knoten und Verbindungen

Ein solcher Graph gestattet es, sofort grundsätzliche Problemzonen zu erkennen, die sich zum Beispiel aus der Verteilung der beteiligten Systeme ergeben. Die Verteilung eines Systems beeinflusst seine Sicherheit auf praktisch allen Ebenen:

- Angriffspunkte entstehen, die es bei lokaler Kommunikation gar nicht geben würde.
- Auf einem Knoten lokal erstellte Tokens und Credentials müssen interoperabel gemacht werden, sonst akzeptiert sie ein anderer Knoten nicht.
- Initiale Authentisierungen sind nicht über verschiedene Knoten hinweg möglich.
- Die Verfolgung von Aufrufen über verschiedene Knoten ist schwierig (Zusammenhang) und aufwändig (Logging).
- Die Infrastrukturkomponenten misstrauen sich eventuell und müssen sich gegenseitig authentisieren.

Die Problematik der Sicherheit in verteilten Systemen ist so wichtig und umfangreich, dass wir ihr weiter unten ein eigenes Kapitel gewidmet haben. Die Abb. 3.7 zeigt ein Beispiel eines Verbindungsgraphen für eine Web-Anwendung, der bereits in unterschiedliche Schutzzonen aufgeteilt ist. Zu klären wäre nun beispielsweise, ob die Knoten in der Production Zone sich gegenseitig vertrauen oder ob eine gegenseitige Authentisierung der Knoten und zusätzlich eine Absicherung der Kommunikation untereinander vonnöten ist. Weitere wichtige Fragen betreffen die Kommunikation des Reverse Proxy mit dem Portal Server und dem Access Manager. Auf die Aufgabe und Funktionsweise dieser Komponenten gehen wir später im Kapitel „Sicherheit der Infrastruktur“ ein.

Knoten und Verbindungen sind das zentrale Thema der Sicherheit im Enterprise-Umfeld. Die dabei auftretenden Probleme der Delegation von Aufrufen und Identitäten werden im Band „Sichere Systeme“ behandelt, zusammen mit den für nötigen Software-Frameworks und Architekturen.

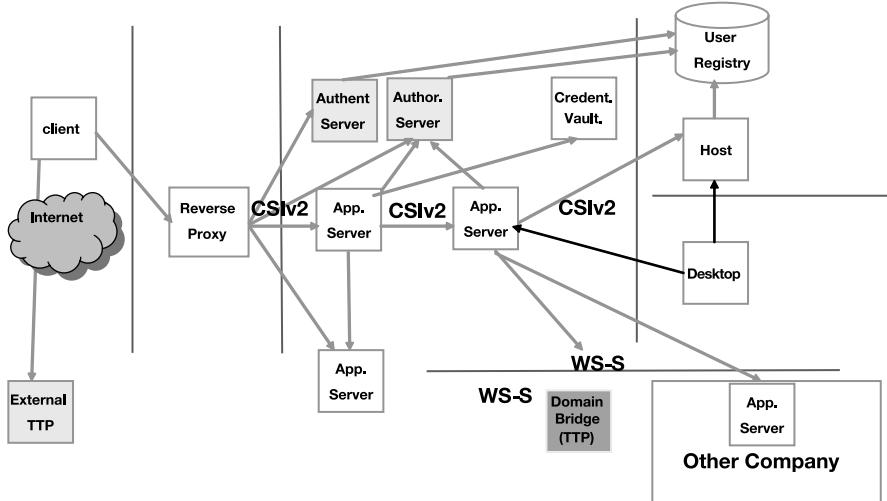


Abb. 3.7 Verbindungsgraph einer Beispiel-Webanwendung

3.2.3.2 Zonenkonzept

Ein Zonenkonzept gestattet es, Vertrauensbeziehungen zwischen unterschiedlichen Bereichen des Verbindungsgraphen klar zu definieren und die Bereiche gleichzeitig abzugrenzen. Somit werden mögliche Risiken besser einschätzbar. Wichtig beim Zonenmodell sind die zu verwendenden Protokolle zwischen den Komponenten. Dieses Modell wird letztlich ausgebaut durch immer spezifischere Informationen über Ports etc. bis hin zum so genannten *Deployment Model*. Das Beispiel eines Zonenkonzepts in Abb. 3.8 ist [IBM] entnommen.

Hier stellt sich die Frage, inwieweit das Knoten- und Zonenmodell auch die verschiedenen Möglichkeiten der Clientseite aufzeigen sollen. So kann der Kunde

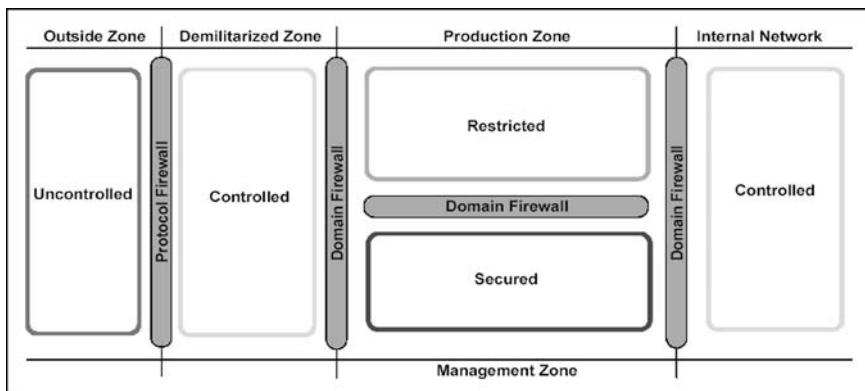


Abb. 3.8 Beispielhaftes Zonenkonzept

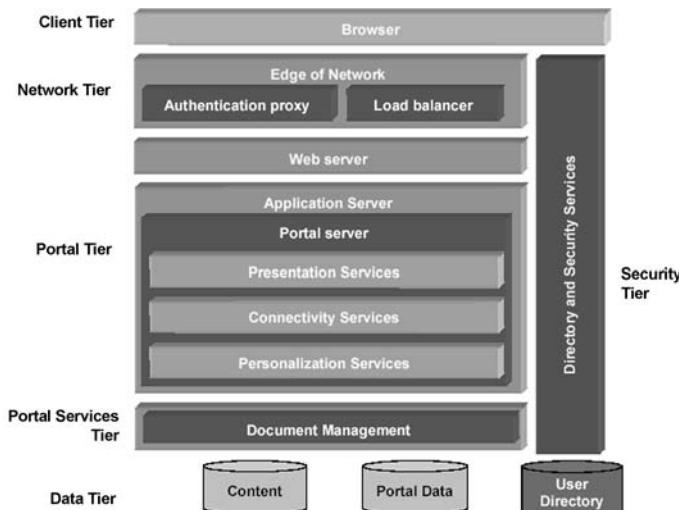


Abb. 3.9 Software-Komponenten einer Portal-Anwendung

ja durchaus Verbindungen gleichzeitig zu anderen Sites unterhalten und eventuell sogar ein föderatives Single-Sign-On-Modell, wie zum Beispiel Liberty Alliance, verwenden. Die Absicherung der Infrastruktur durch Zonen und die Konsequenzen für die Software-Architektur werden wir im Kapitel „Sicherheit der Infrastruktur“ behandeln.

3.2.3.3 Software Komponenten

Das Diagramm der Software-Komponenten zeigt die wesentliche im Portal eingesetzte Middleware sowie Applikationssoftware und Backend Services. Die Darstellung umfasst sowohl Services als auch Komponenten. Abbildung 3.9 zeigt ein Beispiel aus [Cred].

Die Absicherung der Software-Komponenten gegen Angriffe ist ein zentrales Thema der Plattform-Sicherheit und wird – zusammen mit der Sicherheit von Application Servern – im Band „Sichere Systeme“ behandelt.

3.2.4 Bedrohungsmodelle

Bedrohungsmodelle sind für die erstmalige Sicherheitsanalyse durch den Softwareentwickler von entscheidender Bedeutung, da sie den Aufmerksamkeitsbereich in Bezug auf die Wahrnehmung von Sicherheitsproblemen festlegen. Deshalb sollte jeder Entwickler als Teil der Projektdokumentation ganz klar feststellen, welche Bedrohungsmodelle der Implementation zu Grunde gelegt wurden. Mit anderen

Worten: Es sollte dokumentiert sein, gegen welche Gefahren etwas unternommen wurde und welche Bedrohungen bei der Implementation unberücksichtigt blieben.

An der zeitlichen Entwicklung der Bedrohungsmodelle lässt sich auch die Entwicklungsrichtungen der Attacken ablesen und damit indirekt auch die Entwicklungsrichtung von IT-Infrastrukturen und -Techniken, wie wir im Folgenden sehen werden. Bedrohungsmodelle stellen eine formale Technik dar, um die Bedrohungen gegen ein System zu erkennen und geeignete Gegenmaßnahmen zu ergreifen. Sie legen fest, welche Komponenten eines Systems als vertrauenswürdig, welche als potenziell gefährlich und welche als in jedem Fall gefährlich zu betrachten sind. Abb. 3.10 zeigt eine Übersicht der heute benutzten Bedrohungsmodelle.

In den Anfangszeiten der Computernutzung dominierte das *Host-Bedrohungsmodell*, speziell bezogen auf Multi-User Systeme wie Mainframes oder Unix-Maschinen, bei denen es darum ging, einzelne User und Prozesse voneinander zu trennen.

Mit Beginn der Vernetzung kam das *Netzwerk-Bedrohungsmodell* hinzu: Vernetzte Rechner bilden eine Gefahr füreinander und können für gegenseitige Angriffe genutzt werden. Spezielle Angriffe auf Basis von TCP/IP nutzen die fehlenden Authentisierungs- und Authorisierungsmechanismen dieser Protokolle aus. Für Netzwerk-Betriebssysteme spielt dieses Bedrohungsmodell nach wie vor eine große Rolle.

Das Netzwerk-Bedrohungsmodell mutierte schnell ins *Internet-Bedrohungsmodell*, bei dem es um die sichere Kommunikation von Applikationen über das Internet geht. Grundsätzlich liegt das Augenmerk hier auf der integren und vertraulichen Übertragung von Daten zwischen Beteiligten über unsichere und unzu-

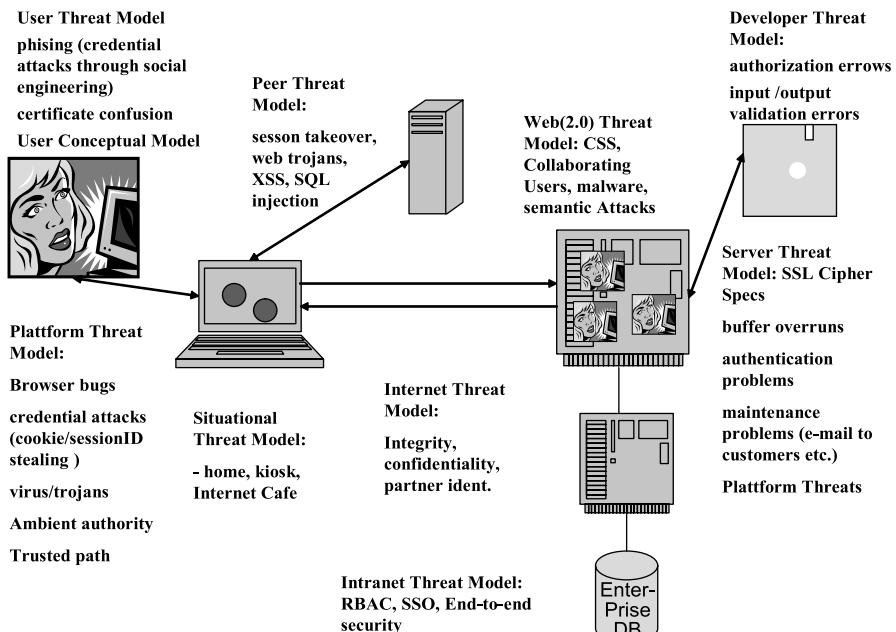


Abb. 3.10 Bedrohungsmodelle

verlässige Kanäle. Das Internet-BedrohungsmodeLL geht grundsätzlich von sich gegenseitig misstrauenden Teilnehmern aus, damit ist die Validierung von Input Pflicht in diesem Modell. In den letzten Jahren sind jedoch weitere BedrohungsmodeLLe hinzugekommen und ältere haben neue Aktualität erlangt.

Neu ist das *User-BedrohungsmodeLL* – gekennzeichnet durch eine große Zahl von Spoofing- und Phishing-Attacken. Dies sind semantische Attacken, die keine Protokollschwächen ausnutzen, sondern vielmehr einen Irrtum des Users herbeiführen bzw. ausnutzen wollen. Dieses BedrohungsmodeLL ist heute für alle serverbasierten Applikationen von großer Bedeutung und somit auch für ein Portal.

Auch speziell gefertigte Internetapplikationen werden mittlerweile als eigenes BedrohungsmodeLL erkannt: Sie enthalten mangels Standardisierung und Verbreitung meist eine große Anzahl von Sicherheitslücken, die häufig nur zufällig gefunden werden. Für die Abwehr der Bedrohungen aus diesem BedrohungsmodeLL wären automatische Tests der Sicherheit von Applikationen besonders wichtig.

Weitere BedrohungsmodeLLe betreffen Peer-to-Peer-Anwendungen: Statt auf zentrale Autoritäten setzen sie oft auf Reputationssysteme, die aber ihrerseits angegriffen werden können, wie das Beispiel ebay gezeigt hat.

Schließlich deuten neuerliche Skandale im Bereich Identitätsdiebstahl bei amerikanischen Firmen auf ein weiteres BedrohungsmodeLL hin: Das *Infrastruktur-BedrohungsmodeLL*, bei dem es um die Absicherung von Daten gegen diejenigen geht, die sie verarbeiten.

Durch den Anschluss einer großen Zahl von Windows-basierten PCs an das Internet sind die Host-BedrohungsmodeLLe wieder neu in den Mittelpunkt gerückt: Trojanische Pferde und Viren sind auf diesen Plattformen weit verbreitet und bedrohen dadurch auch die Serverapplikationen, zum Beispiel indem Trojanische Pferde Credentials der Kunden auslesen und an den Angreifer senden, der diese zur Impersonation der Kunden missbraucht. In diesem Zusammenhang stellt sich die Frage, inwieweit sich PCs auf Basis heutiger Technologie (das heißt Windows oder Linux) überhaupt absichern lassen. Microsoft scheint hier für private PCs das gleiche Modell wie für Firmengeräte vorzusehen: Zentrale Administration durch globale Daten-Zentren, die Bedrohungen registrieren und automatische Patches durchführen. Da es sich dabei jedoch immer nur um nachträgliche Korrekturen handeln kann, bleibt die Gefährdung der privaten Geräte durch so genannte „Zero-Day-Attacken“, also bis dato unbekannte Angriffe, zunächst bestehen. Aber wie soll dieses Modell in Zukunft funktionieren, wenn es sich nicht mehr um den heimischen PC handelt, sondern um Steuergeräte im Auto und Haus?

3.3 BedrohungsmodeLLe für bahn.de

Nachdem wir nun die Punkte, aus denen unser allgemeines Vorgehensmodell besteht, abgehandelt haben, zurück zu unserem Beispiel bahn.de: Wie kommt man zu BedrohungsmodeLLen für das bahn.de Portal und wie verwendet man sie anschließend?

Der einfachste Weg besteht darin, das Security-Context-Diagramm zu nehmen und darin die jeweiligen Bedrohungsmodelle einzuziehen. Für die Entwickler hat dieses Vorgehen einen entscheidenden Vorteil: Hinter den Modellen stehen bekannte Angriffsformen und deren Abwehrmöglichkeiten, das heißt, Entwickler können die jeweils möglichen Gefahren erkennen und gleichzeitig Techniken zu deren Beherrschung nachlesen. Damit lässt sich die mangelnde Wahrnehmung von Sicherheitsproblemen und die Unkenntnis bezüglich deren Lösung bekämpfen.

Letztlich stellen Bedrohungsmodelle analytische Patterns für Sicherheitsprobleme in komplexen Infrastrukturen unter menschlicher Beteiligung dar. Für bahn.de sind folgende Bedrohungsmodelle relevant:

- User-Bedrohungsmodell: Wie kommuniziert bahn.de mit Kunden? Eine Kommunikation per E-mail könnte zu Phishing-Attacken führen. Rechnet bahn.de deshalb mit gefälschten Klonen des Portals, die User Credentials auflesen und missbrauchen? Vertraut bahn.de auf die Sicherheit eines aufgebauten Kanals oder erwartet das Portal signierte Transaktionen von Seiten der Kunden? Soll eine Mehr-Faktor Authentisierung eingesetzt werden?
- Internet-Bedrohungsmodell: Wie werden die Kundendaten übertragen? Wie wird die Infrastruktur von außen gesichert? Wie identifizieren Kunden das Portal? Wie bleibt das Portal verfügbar? Wie wird die Sicherheit der Applikationslogik geprüft?
- Spezialapplikationen-Bedrohungsmodell: Wie wird die Sicherheit der Applikationslogik geprüft?
- Host-Bedrohungsmodell: Wie funktioniert die interne Trusted Computing Base? Welche Konsequenzen haben Buffer Overflows? Wie sehen die Mechanismen für Delegation und Impersonation aus?

Für die Softwareentwickler sind die Bedrohungsanalysen essentiell, um die möglichen Gefahren für das System rechtzeitig zu erkennen. Sie sollten deshalb auch möglichst früh daran beteiligt werden. Ein Problem bleibt jedoch: So wichtig eine gute Bedrohungsanalyse auch ist, sie stellt nur eine indirekte Hilfestellung im Entwicklungsprozess dar. Sie sagt den Softwareentwicklern nicht, wie auf einer speziellen Plattform entwickelt werden muss, um die gewünschte Sicherheit zu gewährleisten. Sie sagt auch nichts über die Verbindung zwischen Netzwerksicherheit und Softwaresicherheit aus. Dies ist ein Punkt, der gerade von Sicherheitsprofis (die häufig aus dem Netzwerkbereich kommen) gerne unterschätzt wird.

3.3.1 Sicherheit auf Clientseite

Für das sichere Design verteilter Applikationen ist es sehr wichtig, die speziellen Bedingungen beim Client zu kennen. Dazu gehören der Ort und die Situation, unter denen der Kunden Kontakt aufnimmt und Transaktionen durchführt, aber auch die Software, mit denen er dies tut. Im Falle Internet-basierter Dienste ist dies so gut wie immer ein Web-Browser. Typische räumliche Szenarien sind:

- Heimuser (alleine oder mit anderen geteilter PC),
- öffentlicher Kiosk,
- Internet Café oder fremder Rechner,
- Firmenrechner und
- Mobiler User via PDA oder Smart Phone.

Die drei erwähnten Aspekte der clientseitigen Security, Ort, Situation und Browsersoftware, gehören zum Host-BedrohungsmodeLL und zum User-BedrohungsmodeLL. Soll ein Portal wie bahn.de auf diese BedrohungsmodeLle Rücksicht nehmen und so weit wie möglich versuchen, die Sicherheit des Kunden zu gewährleisten, stehen die Entwickler jetzt vor dem Problem, die richtigen Maßnahmen auf Serverseite zu treffen. Wie nähert man sich so einem komplexen Sicherheitsproblem?

Eine in vielen Fällen recht brauchbare Heuristik besteht darin, sich zuerst auf die Daten zu konzentrieren und anschließend nach dem Status der Kommunikation und dem unterliegenden Vertrauensverhältnissen zu fragen. Zuletzt fragt man nach der Sicherheit der verwendeten Credentials (Geheimnisse zur Authentisierung wie Schlüssel, Passwörter etc.).

Konzentration auf die Daten bedeutet zu untersuchen, welche Daten eingegeben, übertragen und gespeichert werden. Bei der Übertragung der Daten darf man sich dabei nicht ausschließlich auf die kanalorientierten Aspekte Integrität und Vertraulichkeit konzentrieren, sondern in erster Linie ist zu prüfen, welche Qualität diese Daten besitzen und ob sie überhaupt übertragen werden sollten.

Anschließend werden die Daten nach einem einfachen Schema in „öffentliche“, „privat“ und „geheim“ klassifiziert. Schließlich spielt auch der Status der Kommunikation eine Rolle: Mit wem wird kommuniziert und besteht eine Verbindung zu dem Partner?

Welche Daten werden vom Kunden an das Portal übermittelt? Dazu können gehören:

- UserID,
- Passwort,
- Kreditkartennummer,
- Kurznummer,
- Adresse,
- Reisedaten,
- Bestelldaten und
- evtl. zusätzliche Authentisierungsdaten (PIN/TAN).

Hier die Daten vom Portal an den Kunden:

- elektronische Tickets,
- Reisedaten,
- Bestellformulare,
- Bestätigungen,
- evtl. Kontostände und Nummern und
- Session Data (SessionIDs, Cookies).

Man erkennt leicht, dass die meisten dieser Daten zumindest als „privat“ anzusehen sind, und daher von anderen Benutzern oder Angreifern nicht einzusehen sein sollten. Einige der Daten, wie zum Beispiel Passwörter oder Kreditkartennummern sind besonders sensitiv. Können sich Angreifer Zugang zu Ihnen beschaffen, ist der Erfolg des gesamten Portalprojekts gefährdet. Beginnen wir nun mit dem scheinbar einfachsten der räumlichen Szenarien, dem PC daheim.

3.3.1.1 PC zu Hause

Der häusliche PC ist zunächst einmal den vielfältigen Gefahren ausgesetzt, die der Betrieb am Internet mit sich bringt. Viren, Würmer und Trojanische Pferde könnten den PC unbemerkt infiziert haben und sorgen dafür, dass der heimische PC keineswegs als „Trusted Host“ angesehen werden kann, zumal bei einem Heim-PC nicht davon auszugehen ist, dass er durch eine Firewall oder einen Virenschanner geschützt ist. So könnten die Credentials eines Kunden durch ein Trojanisches Pferd mit der verborgenen Funktionalität eines Keyboard Loggers mitgeschnitten und an einen Angreifer übermittelt werden.

Sehr häufig wird der häusliche PC von der gesamten Familie oder auch von Gästen genutzt. Bei entsprechender Konfiguration des Browsers werden Anmeldeformulare für eine bestimmte Website automatisch und ohne Passworteingabe ausgefüllt. Es ist daher nicht auszuschließen, dass eine Bestellung eines registrierten Kunden in Wirklichkeit von einem Mitbenutzer des häuslichen PCs kommt.

Als weiteres Problem stellt sich die Frage, was eigentlich mit vertraulichen Anmeldedaten auf der Festplatte geschieht, wenn der Rechner einmal verschrottet wird. Auch hier ist ein gewisses Gefahrenpotenzial zu sehen.

3.3.1.2 Öffentlicher Kiosk/Internet-Café

Steht die Hardware an einem öffentlichen Ort wie einem Internet-Café, könnte es sogar sein, dass Malware vorsätzlich von Nutzern aufgespielt wird. Einem solchen Endpunkt kann also noch weniger Vertrauen entgegen gebracht werden wie einem Heim-PC.

Durch die Öffentlichkeit ergeben sich noch einige weitere Aspekte: So könnten die von einem Kunden über eine Web-Formular eingegeben Credentials für den Nachfolger an diesem PC weiterhin sichtbar sein. Die Kunden sollten in dieser Hinsicht durch entsprechende Aufklärungsarbeit sensibilisiert werden. Zudem besteht an öffentlichen Orten immer die Gefahr des Mitlesen persönlicher Daten durch andere Kunden des Cafés. Die vertrauliche Kommunikation mit dem Kunden per E-Mail ist an einem öffentlichen Ort erheblich erschwert.

Im Kontext unseres Beispiels bahn.de muss schließlich noch berücksichtigt werden, dass das auf den Zielrechner übermittelte Ticket in Form einer pdf-Datei wahrscheinlich auf dem Zielrechner verbleiben wird, wenn der Kunde den PC verlässt. Theoretisch wäre also ein Missbrauch des Tickets durch den nächsten

Kunden möglich. Die Bahn schützt sich dagegen, indem sie zur Validierung des Tickets im Zug noch eine zusätzliche Authentifikation des Kunden (zum Beispiel durch die BahnCard) verlangt.

3.3.1.3 Firmenrechner

Steht der Kundenrechner im Intranet einer Firma, ist es höchst wahrscheinlich, dass er sich hinter einer Firewall befindet. Wenngleich diese Tatsache das Risiko einer Infektion mit Malware mindert, könnte es doch sein, dass der Firewall die sichere Kommunikation mit dem Kunden via SSL unterbindet. Sofern Firmenkunden zum Geschäftsmodell des Portals gehören, sollten hier entsprechende Absprachen mit der IT-Sicherheitsabteilung der Firma getroffen werden. Zudem ist auch hier durch wechselnde Benutzer die Vertraulichkeit der Daten gefährdet.

3.3.1.4 Mobile Nutzer

Mobile Nutzer sind neben den Gefahren, die ihnen auch bei Nutzung des Heim-PC drohen, einigen zusätzlichen Gefahren ausgesetzt. Hier ist in erster Linie die bei einem mobilen Gerät deutlich erhöhte Gefahr des Verlusts oder Defekts zu nennen (siehe hierzu auch die interessante Diskussion in [Schn06]). Mobile Malware gewinnt immer mehr an Bedeutung, nachdem ihre Verbreitung einige Zeit durch die bei mobilen Geräten eingesetzten proprietären Betriebssysteme eingedämmt wurde. Dazu kommen weitere mobil-spezifische Sicherheitslücken wie etwa die, die sich aus fehlerhaften Implementationen von Bluetooth-Schnittstellen ergeben (siehe zum Beispiel [LL]).

3.3.1.5 Der Browser

Wenden wir uns nun der Schnittstelle zu, über die der Kunde mit dem Portal kommuniziert: Dem Browser. Wir haben bereits gesehen, dass die bloße Tatsache, dass ein Browser in einer öffentlichen Umgebung von mehreren Personen benutzt wird, für Sicherheitsprobleme sorgen kann. Diese betreffen nicht nur vom Browser zwischen gespeicherte Credentials, sondern auch Cookies, die zur Wiedererkennung bereits authentisierter Nutzer während einer Browsersession oder sogar darüber hinaus genutzt werden.

Aber auch in einer Single-User Umgebung können Credentials oder Cookies durch Malware gestohlen werden. Die Infektion mit der Malware bzw. deren Ausführung geschieht häufig durch den Browser selbst, indem Webseiten besucht werden, die mittels geeigneter präparierten Javascript-Codes erstellt wurden. Schließlich bleibt noch die Gefahr semantischer Attacken durch präparierte Webseiten, die in „Look&Feel“ das GUI legitimer Portale nachbilden.

Alle diese Bedrohungen betreffen sowohl den Internet-Explorer als auch Open-Source Software wie Mozilla oder Firefox. Es bleibt daher zu konstatieren, dass beim gegenwärtigen Stand der Technik der Kunde nur wenig wirkliche Kontrolle über die Sicherheit des Browsers besitzt, und zwar einerseits mangels Wissen, andererseits aber auch mangels Softwareunterstützung. Es stellt sich die Frage, ob Browser hier nicht deutlich mehr für die Sicherheit der Kunden tun könnten.

3.3.2 Die Verantwortung des Servers

Die Anforderungen bezüglich der Sicherheit auf Seite des Kunden legen dem Portal viel Verantwortung auf. Dies ist begründet in der engen Verzahnung von Browser und Website durch http bzw. html. So ist http bekanntlich ein verbindungsloses Protokoll, das aber durch serverseitige Maßnahmen „stateful“ gemacht wird, das heißt, es wird auf Applikationsebene eine permanente Verbindung zwischen Client und Server hergestellt, die den Client eindeutig identifiziert. Wie am Beispiel ebay gesehen, geschieht dies zumeist durch Cookies, also Token, die das Portal ausstellt und die bei jedem Request des Client wieder ans Portal zurückfließen, so dass das Portal den Kunden am Token wieder erkennen kann. Damit ist dieses Token ein zu schützendes Credential.

Auch html hat einige sehr kritische Eigenschaften, wie sich später an der Browseruntersuchung noch erweisen wird. Hier sei nur erwähnt, dass über Links der Browser zu Aktionen veranlasst werden kann, die ausschließlich vom Server kontrolliert werden und dass zum Beispiel das Form-Element einen versteckten Kanal nach außen darstellt, der ebenfalls vom Server kontrolliert wird.

Zur Serververantwortung in der direkten Verbindung zum Client gehören die folgenden Aspekte:

- Sicherung und Konsistenz der eigenen Zertifikate (www.bahn.de hatte zu Beginn Probleme mit selbst signierten Zertifikaten, abgelaufenen Zertifikaten oder ungültigen Zertifikaten von Fremdservices, vgl. auch den erwähnten „Domain Name Mismatch“ bei ebay).
- Sicherung der Vertraulichkeit und Integrität bei der Übertragung von Daten. Hier ist es die Aufgabe des Systemadministratoren auf der Serverseite die Cipherspecs für SSL (also die für die Absicherung der Kommunikation verwendeten Algorithmen) in ausreichender Qualität vorzuschreiben und gleichzeitig ein Downgrading auf Wunsch des Clients auszuschliessen – dadurch werden so genannte Downgrade-Attacken vermieden.
- Das Erraten eines Sessiontokens (Cookie) erlaubt einem beliebigen Angreifer, die existierende Session des Kunden zu übernehmen und Bestellungen etc. im Namen des Kunden auszulösen. Deshalb darf ein verwendetes Sessiontoken keinen erratbaren Wert haben. Die heute verwendeten Application Server erzeugen inzwischen sichere Tokens.

- Cookies können durch Malware auf der Seite des Clients gestohlen werden. Deshalb keine persistenten Cookies für das Speichern von SessionIDs verwenden (oder noch schlimmer: Passwortinformationen). Besser ist es nur temporäre Cookies zu verwenden. Die beste Lösung besteht darin, statt Cookies die SSL SessionID zu verwenden (siehe dazu die umfangreichen Tipps bei [Huse]).
- Das Portal hat keine Kontrolle über das Kundenverhalten, das heißt, der Kunde kann sich jederzeit vom Browser entfernen und eine etablierte Session (eingeloggt) zurücklassen. Damit kann eine andere Person im Namen des Kunden Geschäfte tätigen oder vertrauliche Daten einsehen. Dies ist ein besonderes Problem, wenn die Session auf einem Kiosk, zum Beispiel im Supermarkt stattfindet. Deshalb muss serverseitig ein besonders kurzer Timeout eingestellt werden. Nach einer kurzen Zeit der Inaktivität muss der Server die existierende Session terminieren und – sollte dennoch ein weiterer Request eintreffen – zu einer neuen Authentisierung auffordern. Die serverseitige Software muss daher unterschiedliche Timeouts pro Service bzw. Ort der Kommunikation unterstützen.
- Spoofing/Phishing: Als gefährliche Mechanismen des Browsers haben sich in der Vergangenheit immer wieder Frames, Pop-up Windows, Systemdialoge und vor allem Scripting erwiesen. Durch den Verzicht auf diese können Serverseitig einige Risiken vermieden werden.
- Scripting und Cross-Site-Scripting: Der Server hat eine besondere Verantwortung, wenn auf seinen Seiten Inhalte von anderen Usern eingestellt werden bzw. von anderen Servern geladen werden. Hierdurch wird eine evtl. bestehende Vertrauensbeziehung zwischen Kunde und Portal missbraucht, da sich der Kunde darauf verlässt, dass die Inhalte von dem ihm bekannten Portal stammen und seinem System nicht schaden werden.
- Privacy Erziehung und Tipps für die Kunden: Die meisten Portale haben mittlerweile einen gesonderten Bereich, in dem ihre Kunden über die Gefahren von Phishing-Mails und mögliche Kennzeichen dieser Mails aufgeklärt werden. Die Sicherheitshinweise könnten jedoch durchaus noch weitere Aspekte umfassen, wie das Beginnen einer neuen Browser-Session, wenn vertrauliche Daten übermittelt werden sollen, sowie die Schließung der Session und Löschung des Cache danach.

3.3.3 Kommunikation mit dem Kunden

Jede Firma, die Services über einen Webauftritt oder ein Portal anbietet, steht schnell vor der grundsätzlichen Frage, wie die Kommunikation mit Kunden (das heißt, von und zu den Kunden) gestaltet werden soll. Oft lautet die (vor)schnelle Antwort: per E-Mail. Stellen wir jedoch kurz einige Sicherheitsanforderungen auf, die aus Sicht des Business die Kommunikation mit den Kunden steuern sollen:

- Nachrichten von Kunden dürfen nicht verloren gehen.
- Nachrichten von Kunden müssen die richtigen Personen innerhalb der Firma erreichen.
- Nachrichten von Kunden müssen nachweislich von ihnen stammen (Authentisierung der Nachricht).
- Nachrichten von Kunden müssen dem korrekten firmeninternen Workflow folgen. So sollte beispielsweise bei Urlaub eines Sachbearbeiters die Nachricht an den definierten Stellvertreter gelangen.
- Nachrichten von Kunden müssen zentral pro Kunde verwaltet werden und global von Sachbearbeitern einsehbar sein, um die Aufsplittung der Kommunikation zu vermeiden.
- Nachrichten zum Kunden müssen nachweislich an den Kunden gelangen.
- Nachrichten von und zum Kunden müssen vertraulich und integritätsgeschützt übertragen werden.
- Nachrichten an den Kunden müssen für den Kunden verifizierbar von der Firma kommen.
- Nachrichten an den Kunden dürfen nicht gefälscht werden können.

Sieht man diese Anforderungen durch, so stellt sich schnell heraus, dass damit E-Mail als Lösung eigentlich ausscheidet – auch wenn von geschäftlicher Seite in Verkennung der technischen Probleme häufig in diesem Zusammenhang von „Mail“ gesprochen wird. Reguläre E-Mail ohne zusätzliche Sicherheitsfunktionalität deckt jedoch keine der obigen Anforderungen ab. Dies gilt selbstverständlich auch für den Fall, dass im E-Mail-Client „SSL“ als Verbindungsprotokoll angegeben wird. Zwar ist dies durchaus sinnvoll, um das eigene Passwort bei der Übertragung zum Mailserver zu sichern, jedoch bezieht sich diese sichere Übertragung lediglich auf diesen einen „Hop“ und keinesfalls auf eine Ende-zu-Ende Verbindung von Kunde zum Portal.

Auch die Installation eines PGP Plug-ins in Firefox oder Outlook zur sicheren Authentisierung des Kunden und zur Sicherung der Nachricht selbst (Verschlüsselung, Signatur) wird die Mehrzahl von Kunden überfordern (siehe hierzu auch [WT]) – vom zusätzlichen Aufwand auf Serverseite für die Verwaltung der öffentlichen Schlüssel aller Kunden ganz abgesehen. Das gleiche dürfte auch für den Einsatz von S/MIME² zur Signierung und/oder Verschlüsselung der E-Mails gelten.

Gerade die letzte unserer Anforderungen ist heute sehr kritisch geworden durch die hohe Zahl von Phishing-Attacken über E-Mail. Hierbei versendet ein Angreifer Mails mit gefälschter Absenderadresse, in die Links eingebettet sind, die nicht auf das Originalportal sondern auf eine Fälschung zeigen. Dort werden die Kunden dann in autoritativem Ton aufgefordert, ihre Credentials einzugeben („Loggen Sie sich bitte ein und kontrollieren Sie Ihre Daten“), die dann anschließend sofort missbraucht werden.

² S/MIME ist ein bislang im privaten Bereich eher wenig genutzter Standard, mit dem E-Mails verschlüsselt und digital signiert werden können, um dann als Anhang einer SMTP-Mail verschickt zu werden. Ursprünglich von der Firma RSA entwickelt, gibt es nun auch bei der IETF eine S/MIME Working Group (siehe <http://www.ietf.org/html.charters/smime-charter.html>).

Entscheidet sich eine Firma grundsätzlich gegen die Verwendung von E-Mail – sei es zur Kommunikation oder Werbezwecken – dann besteht auch weniger die Gefahr, dass Kunden eine Phishing-Mail akzeptieren. Leider gibt es aber sogar Banken, die nicht auf die Verwendung von E-Mail verzichten, obwohl gerade in diesem Bereich Phishing-Attacken besonders gefährlich sind.

Leider ist E-Mail häufig der billigste Weg – zum Beispiel bei einem vergessenen Passwort – dem Kunden eine kurze Bestätigung zu schicken. Auf diese Weise lässt sich der Kunde zumindest warnen, wenn ein Missbrauch seiner Credentials vorliegt. Das gleiche gilt natürlich für Bestätigungen bei Bestellungen, wie sie von den meisten Webshops versandt werden.

Im Portal stellt sich die Sicherheitsfrage in Bezug auf die Kommunikation mit dem Kunden noch einmal anders als bei einem Webshop. Ein Portal bezieht häufig Services von weiteren beteiligten Firmen (siehe unten: Föderatives Identitätsmanagement). Diese Firmen vertrauen häufig auf die initiale Authentisierung des Portals und übernehmen die Identität eines Portalkunden ganz einfach. Das bedeutet aber, dass der Portalbetreiber die Sicherheit der nachgeordneten Firmen und deren Anforderungen ebenfalls berücksichtigen muss. Welche Alternativen gibt es also zur Verwendung von E-Mail?

In vielen Fällen wird sich die Firma für ein datenbankgestütztes Nachrichtensystem (so genanntes *Database Backed Messaging System*) entscheiden. Hierbei muss sich der Kunde mit seinen Credentials wie gewohnt anmelden und kann dann Nachrichten innerhalb der Firma verschicken und für ihn bestimmte Nachrichten abrufen. Natürlich kann das auch anonym geschehen, etwa wenn der Kunde erst ein „Prospekt“ ist, das heißt, noch nicht ein wirklicher Kunde ist.

Datenbankgestützte Nachrichtensysteme haben den großen Vorteil, dass die Quelle der Nachricht durch die Authentisierung bekannt ist und der Übertragungsweg vom Portal zum Kunden durchgängig über SSL geschützt werden kann. Gleichzeitig können die Nachrichten über die Datenbank nahtlos in vorhandene Workflow-Systeme eingebunden werden. Allerdings muss sich der Kunde zunächst anmelden, um Nachrichten erhalten oder senden zu können. Muss ein Kunde relativ kurzfristig benachrichtigt werden, bieten sich auch nicht-elektronische Kanäle wie ein Brief an. Das ist zweifellos teurer als eine E-Mail, vermeidet aber das Problem der Phishing-Attacken. Nötige Änderungen von Stammdaten können auch beim nächsten Kunden-Login abgefragt werden.

3.4 Beispiel einer Sicherheitsanalyse im Embedded Control Bereich

Nicht jede Software-Lösung basiert auf dem Web. In diesem Abschnitt wollen wir die Welt der Internet-Portale kurz verlassen und eine kurze Sicherheitsanalyse einer Infrastruktur durchführen, wie sie vor allem im Bereich der embedded-control Applikationen häufig vorkommt. Interessant dabei ist vor allem auch, auf

welche Weise mögliche Sicherheitsprobleme entdeckt werden können. Dazu werden im Verlauf der Analyse einige Bestandteile des Szenarios abgeändert mit drastischen Folgen für die Sicherheit des Gesamtsystems. Auch hier kommen wieder Standard-Bedrohungsmodelelle zum Einsatz. Erklärt werden an diesem Beispiel:

- die Verwendung von sicherheitsbezogenen Metapattern;
- Transformationen als Heuristiken bei der Sicherheitsanalyse;
- die Auswirkung von mobilen Nutzern;
- die praktische Bedeutung von Nicht-Abstreitbarkeit;
- rechtliche Aspekte der Lösung;
- Key Management Probleme: HSM/Cards vs. Gespeicherte Schlüssel;
- Möglichkeiten, das Schadenspotential zu verringern durch den Einsatz von POLA (Principle of Least Authority) und
- Datensicherheit vs. Sicherheit audio-visueller Medien.

Die Verwendung von POLA Methodiken ist an dieser Stelle ein kleiner Vorgriff auf das Kapitel zur Plattform-Sicherheit im Band „Sichere Systeme“. Dort wird das Konzept im Detail dargestellt.

3.4.1 Ausgangsszenario

Das folgende Diagramm zeigt ein automatisches Erfassungssystem für Bilder bzw. Videos von Verbrechen. Denkbare Standorte wären der Eingangsbereich einer Bank oder eine als nächtlicher Drogenumschlagplatz bekannte Einkaufspassage. Unsere Aufgabe besteht in der Absicherung des gesamten Systems gegen Missbrauch bzw. der Bewertung der Systemarchitektur in Bezug auf ihre Sicherheit.

Das System besteht aus einer Kamera mit zusätzlichem Rechner, die an kritischen Stellen installiert wird. Zusätzlich gibt es einen Operator bei der Polizei, der über einen Rechner verfügt und die Kameradaten zu sich transportieren sowie Wartungsfunktionen durchführen kann.

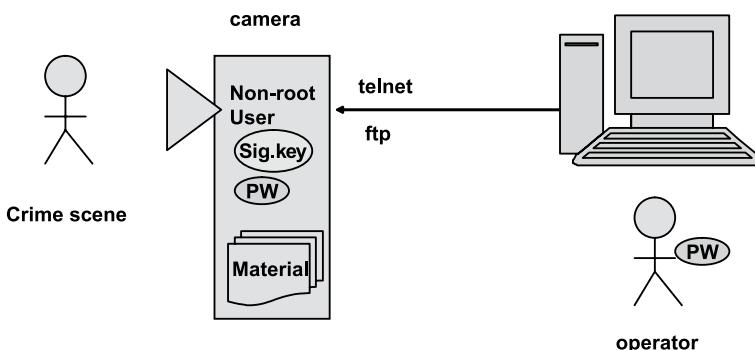


Abb. 3.11 Architektur einer automatischen Überwachungslösung

Der Hersteller des Systems schlägt Passwörter für die Authentisierung des Operatorzugriffs vor. Auf dem Kamerarechner führt der Login des Operators zu Sessions, die keinen Root-Login erlauben. Zur Datenübertragung wird klassisches ftp verwendet, zur Steuerung kommt telnet zum Einsatz. Entsprechende Server sind auf der Kameraseite installiert. Die aufgenommenen Bilddaten werden auf Kameraseite durch einen privaten Schlüssel signiert. Der Schlüssel befindet sich lokal auf der Festplatte und ist nur durch die Rolle Root lesbar. Die Aufgaben des Operators schließen den Transport der Daten sowie die Verwaltung auf dem Ziel-system bei der Polizei ein. Aus rechtlicher Sicht ergeben sich aus diesem Ausgangsszenario einige interessante Sicherheitsanforderungen:

- Es muss sichergestellt werden, dass die erfassten Daten nachweislich von einer dieser Kameras stammen.
- Es darf nicht möglich sein, selbst audiovisuelle Daten zu erzeugen und in das System einzuschleusen, ohne dass die Daten von der Kamera stammen. Falls nämlich eine auf Grund dieser Daten angeklagte Person vor Gericht nachweisen kann, dass sie selber Daten erstellen kann, die scheinbar von der Kamera stammen, dann wird die Anklage abgewiesen werden.
- Es muss ein Konzept geben, um zu verhindern, dass Daten der Kamera spurlos verschwinden. Gleichwohl müssen aber Daten durch autorisiertes Personal gelöscht werden können.

Die ersten beiden Punkte gehören zu dem Sicherheitsdienst „Nicht-Abstreitbarkeit“ (Non-Repudiation). Dieser Dienst sollte also von dem System bereitgestellt werden können, ansonsten sind die gelieferten Daten nicht vor Gericht verwertbar.

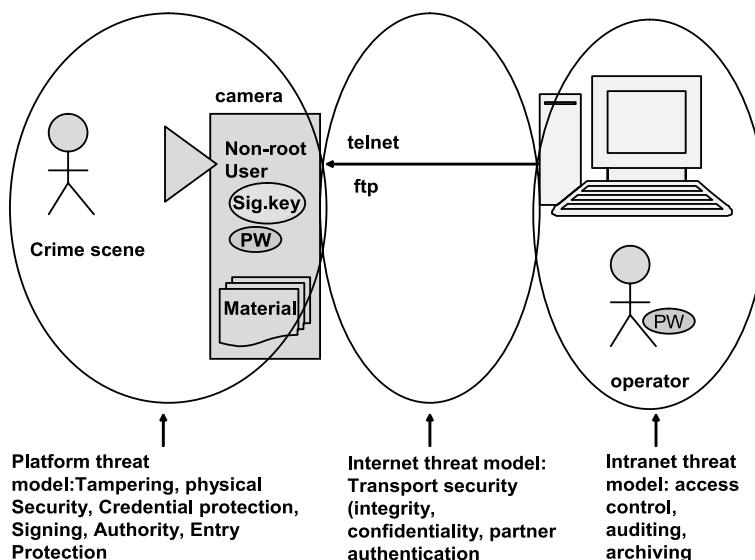


Abb. 3.12 Kritische Bereiche der Überwachungslösung

Aus der Architektur des Systems ergeben sich grob drei kritische Bereiche, die drei unterschiedlichen Bedrohungsmustern entsprechen:

Kamera und Umgebung sind durch das Plattform-Bedrohungsmodell bestimmt. Dieses Modell schließt die Umgebung der Plattform mit ein, enthält aber auch die Runtime-Umgebung selbst mit ihrer Software, den Daten, den Eingängen und nicht zuletzt den gespeicherten Geheimnissen.

Die Übertragung der Daten wird durch das Internet-Bedrohungsmodell definiert. Hier ist die Sicherung der Transportqualität (Integritätsschutz und Vertraulichkeit) sowie die Garantie, mit dem richtigen Kommunikationspartner (Authentizität) zu sprechen, von Bedeutung.

Für den dritten Bereich ist das Intranet-Bedrohungsmodell relevant: Wie wird die Sicherheit der Daten und ihrer Verarbeitung innerhalb der betreibenden Organisation garantiert? Hier sind die entscheidenden Kriterien die Authentisierung der Akteure zur Durchführung einer Zugriffskontrolle sowie der Nachweis von Aktionen über fälschungssicheres Auditing, so dass keine Daten verschwinden oder unbemerkt durch Mitarbeiter modifiziert werden können.

3.4.2 Analyse des Ausgangsszenarios

In einer ersten Phase der Analyse beginnt man praktischerweise mit einer Evaluierung des existierenden Systems, so wie es vom Hersteller beschrieben und geplant ist. Wir übernehmen dabei zunächst das Nutzermodell des Herstellers. Hier gilt es aufzupassen, dass man nicht selbst Opfer eines „Pawlowschen“ Sicherheitsreflexes wird: Manche Schwächen sind in der Fachpresse bereits so prominent diskutiert worden, dass sie einem sofort ins Auge springen. Dabei können wesentliche andere, nicht so offensichtliche Schwächen in der Wahrnehmung verdeckt werden. Die Verwendung von ftp bzw. telnet wird beispielsweise so einen Sicherheits-Reflex verursachen.

In einer zweiten Phase der Analyse werden wir gezielt von dem Nutzermodell des Herstellers abweichen – unter Einsatz von heuristischen Meta-Patterns – und dann die Reaktion des Systems erneut sicherheitstechnisch beurteilen.

3.4.2.1 Plattform-Sicherheit

Die Kameras operieren alleine im öffentlichen Raum und müssen daher gegen externe Einflüsse abgesichert werden. Der Hersteller lässt keinen remote-login mit Root-Berechtigung zu. Deshalb muss zur lokalen Administration mit Root-Rechten ein separates Terminal lokal angeschlossen werden. Dazu ist die Kamera zu öffnen. Das ist ein sehr aufwendiges Verfahren, das zudem durch die Verletzung von Schraubsicherungen entdeckt werden kann. Während es als unwahrscheinlich erachtet werden kann, dass ein Angreifer ein Terminal anschließen

kann, ist dies für die Administration selbst unumgänglich. Dieser Punkt wird bei der Betrachtung der Zugriffe auf die Kamera eine große Bedeutung erhalten.

Um die Anforderung der Nicht-Abstreitbarkeit erfüllen zu können, muss die Kamera ihre Bilddaten mit dem privaten Teil eines asymmetrischen Schlüsselpaares signieren (s. Kap. 5). Vor der eigentlichen Signatur werden die audiovisuellen Daten mit einer Hashfunktion gehasht. Natürlich sollten die dabei eingesetzten Hashfunktionen und Signaturalgorithmen sicher sein, d. h. je nach Anwendungsfall gewisse Anforderungen erfüllen, die wir später noch erläutern werden. Im Normalfall muss sich der Entwickler jedoch nicht um die Auswahl oder gar Implementierung geeigneter Algorithmen kümmern – die Schwachstellen einer komplexen Software-Lösung liegen meist an ganz anderen Stellen.

Der private Schlüssel selbst liegt auf der Festplatte der Kamera und ist nur für die Rolle Root lesbar. Da Remote-Logins keine Root-Rechte erhalten dürfen, muss die also die signierende Applikation der Kamera entweder selber als Root laufen oder (besser) ein setUid Programm zur Signierung verwenden. Das entspräche dem Prinzip der Least Authority (POLA) besser, als die gesamte, wesentlich komplexere Applikation mit vollen Root-Privilegien laufen zu lassen.

Auf die Frage nach der Speicherung des privaten Schlüssels gehen wir gleich noch näher ein. Es gibt jedoch vorher noch einen anderen Aspekt zu klären, der ebenfalls mit POLA in Zusammenhang steht: telnet und ftp zur Steuerung der Kamera bzw. zum Auslesen der Daten sind extrem generische Tools, mit denen weit mehr als nur die aus Applikationssicht nötigen Funktionen ausgeführt werden können. Im Sinne einer Einschränkung der Zugriffsmöglichkeiten auf die wirklich

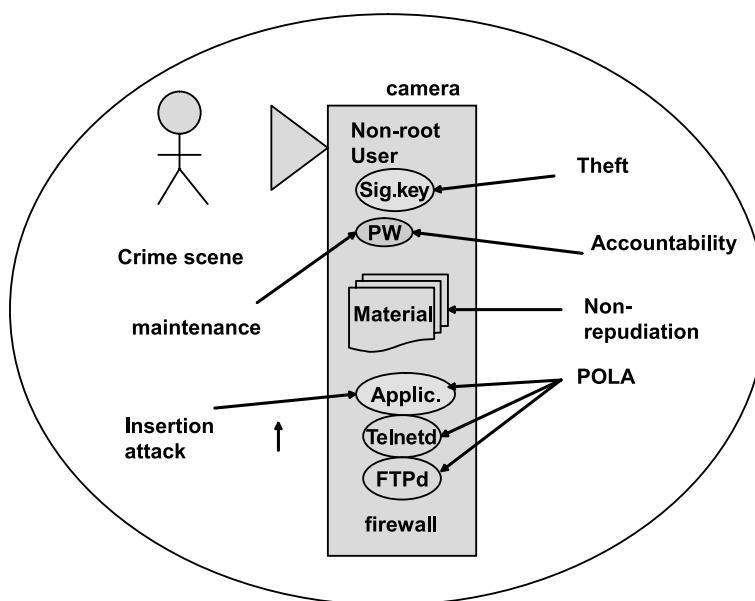


Abb. 3.13 Anforderungen und Bedrohungen aus Sicht der Plattform-Sicherheit

benötigten sind telnet und ftp also sehr ungünstige Lösungen. Für den Hersteller stellen sie zweifelsohne eine bequeme Lösung dar, jedoch verletzen diese generischen Tools POLA beträchtlich: Eine „kleine“ Applikation mit einem eigenen schmalen Interface wäre besser gewesen. Dabei hätte man dann auch eine Applikationseigene Nutzerverwaltung aufbauen können, die die Einrichtung von Betriebssystemusers für telnet und ftp unnötig machen würde. Wozu schließlich brauchen die Remote-Logins allgemeine Betriebssystemrechte (außer aus dem einen Grund, dass nur damit telnet und ftp funktionieren)?

Bleiben wir noch für einen Moment bei den Logins (remote bzw. lokal): Der Hersteller hat sich für eine klassische Lösung mit UserID und Passwort als Zugangscredentials entschieden. Dies wirft zunächst die bekannten Fragen nach der Qualität der Passwörter und ihrer sicheren Verwahrung auf. Aber dies ist nicht das entscheidende Problem: Wichtiger ist die Frage nach den Identitäten, die zum Login verwendet werden: Sind diese UserIDs tatsächliche Identitäten, die Mitarbeitern der Polizei persönlich zugeteilt sind? Kann man also von der für eine bestimmte Aktion verwendeten UserID auf eine persönlich verantwortliche Person zurück schliessen? Oder handelt es sich um eine so genannte „Functional UserID“, die in diesem Fall etwa „Operator“ heißen könnte?

Die Antwort hängt davon ab, ob es Sinn macht, dass die Kamera „weiß“, wer sie gerade bedient. Wir haben oben gesagt, dass sichergestellt sein muss, dass keine Daten verschwinden können. Im Sinne der Nachvollziehbarkeit von Datenlöschungen etc. wäre es also gut, wenn die tatsächliche Identität des Bearbeiters der Kamera bekannt wäre. Jetzt tritt allerdings ein Problem auf: Um wirkliche User per Passwort authentisieren zu können, müssen die User auf der Kamera eingerichtet werden – wir haben aber oben festgestellt, dass remote kein Root-Zugang gestattet sein soll. Zur Einrichtung eines neuen Users müsste die Kamera also aufgesucht, aufgeschraubt und per Extra-Terminal konfiguriert werden. Das ist bestimmt kein gangbarer Weg, so dass es aller Wahrscheinlichkeit auf die Einrichtung eines oder mehrerer Functional Users hinauslaufen wird. Deren Problematik wird weiter unten sehr klar, wenn wir kleine Abweichungen vom Standard-Szenario vornehmen.

Nun zurück zum Signaturschlüssel auf der Festplatte. Stellt dieses Konzept ein Problem dar? Sein Zweck ist klar: Datenmaterial kann durch die Kamera signiert werden – und zwar nur durch die Kamera, solange der Schlüssel nicht kompromittiert wird. Daten können daher einzelnen Kameras sicher zugeordnet werden und die Einschleusung von signiertem Datenmaterial durch die Operatoren oder Dritte Personen ist nicht möglich (vorausgesetzt, der Schlüssel ist nicht auf Seiten der Operatoren bekannt und die Applikation in der Kamera weist nicht ein grobes Sicherheitsproblem auf, dergestalt, dass sie beliebige Daten vom Netzwerk annimmt und signiert). Durch das Nicht-Root Konzept für telnet und ftp ist damit sichergestellt, dass Operatoren durch remote-logins keine Signaturen vornehmen können.

Es bleibt die Möglichkeit, den Signaturschlüssel mitsamt der Festplatte durch einen Einbruch zu stehlen. Ein solcher Einbruch könnte – je nach Kontrollintervallen der betreibenden Firma – relativ lange unentdeckt bleiben. Gelingt es dem

Angreifer, den Schlüssel auszulesen, könnte er selbst Daten signieren – wahrscheinlich das schlimmstmögliche Angriffsszenario in diesem System. In diesem Kontext ist es sinnvoll, den Schlüssel nicht auf der Festplatte, sondern in einem sog. *Hardware Security Module* (z. B. Smartcard Reader plus Signaturschlüssel auf einer Smartcard) zu halten und vor Benutzung durch eine PIN frei zu schalten? Natürlich müsste auch die PIN auf der Festplatte liegen (die Kamera muss ja automatisch, ohne manuelle Einwirkung eines Operators, signieren können). Um aber an den Signaturschlüssel zu kommen, müsste der Angreifer auch die Smart-Card stehlen, und deren Verlust könnte automatisch sofort festgestellt werden und einen Alarm auslösen.

Zweifellos stellt dies eine aufwändige Lösung für einen seltenen, aber schwerwiegenden Angriff dar. Es liegt jetzt an der Risikoabschätzung des Herstellers und des Betreibers, diese Gefahr nach dem oben beschriebenen Schema zu bewerten und gegen die Kosten der skizzierten Lösung abzuwägen.

Halten wir zusammenfassend fest, dass das Konzept des Anbieters auf Plattformseite einige Schwächen beinhaltet, und zwar in Bezug auf POLA, Schlüsselspeicherung und Nicht-Abstreitbarkeit. Wie schwerwiegend gerade der letzte Punkt ist, wird sich weiter unten in einem leicht geänderten Szenario zeigen.

3.4.2.2 Internet-Bedrohungsmode

Kommen wir nun zur Sicherheit des Übertragungsweges zwischen Kamera und Operator. Wie allseits bekannt sein dürfte, verwenden telnet und ftp zur Authentisierung das Username/Password – Schema, wobei beides im Klartext zwischen beiden Rechnern übertragen wird. Neben der Tatsache, dass das Passwort somit von einem Angreifer auf der Leitung abgelauscht werden könnte, wirkt noch wesentlich schwerer, dass keine Authentisierung der Endpunkte der Kommunikation stattfindet, d. h. dem Operator ist gar nicht klar, wem er eigentlich sein hochgeheimes Passwort anvertraut. Die offensichtliche Alternative zu telnet bzw. ftp ist das Protokoll SSH (s. [SSH]), das die Möglichkeit einer gegenseitigen Authentisierung der Endpunkte über Zertifikate sowie den Aufbau eines sicheren Kanals zwischen den Endpunkten bietet. Die Übertragung wäre damit sicher in Bezug auf die Kommunikationspartner und den Inhalt der Kommunikation. Der Eingang ins System ist jedoch nach wie vor viel zu breit, wie wir oben gesehen haben. Eine kleine (Web-)Applikation mit speziellem Interface, die über SSL angesprochen wird, wäre deshalb die bessere Lösung.

Ein wesentlicher Punkt beim Internet-Bedrohungsmode

ll ist die Art der Verbindung zwischen den Geräten: Handelt es sich um eine dedizierte Leitung, d. h. kann nur aus dem Büro der Operatoren auf die Kameras zugegriffen werden, oder sind die Kameras auch über öffentliche Leitungen (z. B. über Telefonleitungen und Modems) erreichbar? Dieser Punkt ist unbedingt zu klären und zwar auch für zukünftige Installationen, da es einen enormen Unterschied macht, wenn die Kameras öffentlich erreichbar sind. In Abhängigkeit des gewählten Anschlusses wird in diesem Fall auch ein Firewall auf Seiten der Kamera Pflicht.

3.4.2.3 Intranet-Bedrohungsmode

Laut Hersteller verwenden die Operatoren direkte Logins per ftp und telnet auf die Kameras. Nötig wären im Sinne der Nachvollziehbarkeit zumindest persönliche Logins. Aber es stellt sich eine viel grundsätzlichere Frage: Sind persönliche Logins auf der Kamera überhaupt die richtige Lösung? Will bzw. muss die Kamera tatsächlich eine unmittelbare Authentisierung von Nutzern vornehmen? Schließlich sind die Daten der Kamera doch nicht unterschiedlich je nach Operator. Die Kamera möchte lediglich sicherstellen, dass die Daten von dazu autorisierten Personen abgerufen und gelöscht werden. Diese Funktion könnte aber auch in eine Applikation auf Seiten der Operateure verlagert werden: Die Applikation und die Kamera authentisieren sich über Zertifikate. Der Schlüssel der Applikation ist ebenfalls in einer Smartcard untergebracht (ein Passwort-basiertes System auf Seiten der Applikation würde zu Sicherheitsproblemen führen, wenn das Applikationspasswort im Zuge von Wartungsarbeiten den Operateuren bekannt gemacht werden muss). Die Applikation meldet zusätzlich der Kamera, welcher Operator (persönlich authentisiert über die Authentisierungslösung des Intranets) das Kommando ausführen will. In der Folge kann diese Identität sogar Teil der Signaturdaten werden. Das Wartungsproblem der UserIDs und Passwörter ist verschwunden. Die Applikation signiert Verwaltungsakte der Operateure und speichert sie auf einem sicheren Logsystem zur späteren Prüfung. Ein positiver Nebeneffekt dieser Lösung besteht darin, dass sie aufgrund der zur Authentisierung der Applikation nötigen Smartcard auch nur vom Intranet aus ausgeführt werden kann, ein Punkt, dessen Bedeutung wir gleich sehen werden.

3.4.3 Modifiziertes Szenario

Wir verändern nun bewusst einzelne Teile des vorgegebenen Szenarios, und zwar basierend auf so genannten *Meta-Pattern* der Sicherheit. Diese Meta-Pattern sind quasi Archetypen von Ursachen, die Sicherheitsprobleme sichtbar werden lassen. Zwei dieser Archetypen lassen sich beschreiben als „Trennen“ bzw. „böse werden“. Wir nehmen dazu zwei einfache Transformationen auf der Intranet-Seite der Lösung vor, die einem Wechsel im ursprünglichen Nutzerkonzept des Herstellers entsprechen: Die erste Transformation ist eine räumliche: Wir entfernen den Operator aus dem Sicherheitskontext „Intranet“ zu sich nach Hause. Die zweite Transformation ist eine moralische – wir lassen ihn „böse“ werden.

Wie ändert sich die Sicherheit der Herstellerlösung durch die räumliche Transformation? Die Herstellerlösung verwendet Passwörter zum Remote Login via ftp und telnet. Der Operator kann nun, vorausgesetzt, die Kamera ist über eine öffentliche Leitung erreichbar, einen Functional User Account bei sich zuhause verwenden, um sich über via ftp und telnet auf einer Kamera einzuloggen. Die Bedeutung dessen wird klar, wenn wir den Operator nun zusätzlich „böse“ werden lassen: Nehmen wir beispielsweise an, ein Freund des Operators ist ein versierter Einbre-

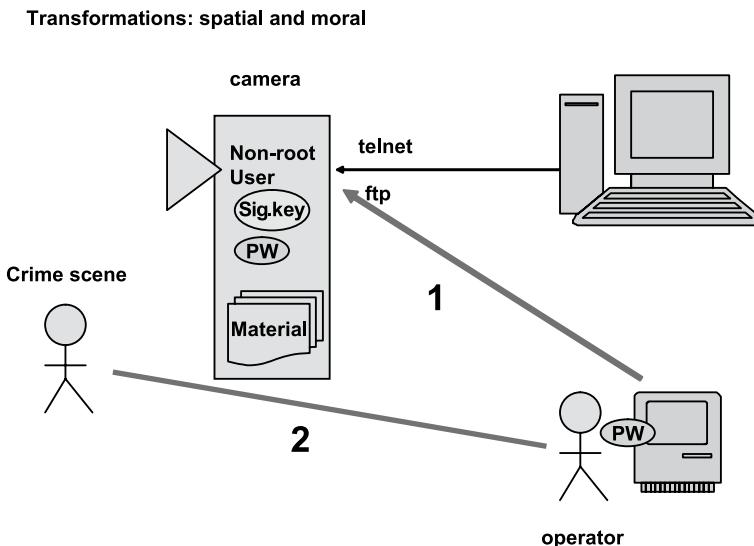


Abb. 3.14 Räumliche und „moralische“ Transformation des Ausgangsszenarios

cher. Zusammen mit dem Operator plant er einen Einbruch. Nach der Durchführung löscht der Operator die Daten der Kamera durch Zugriffe von außerhalb des Intranets. Alternativ kann der Operator auch die Logindaten an den Freund verkaufen.

Die vorgeschlagene Lösung des Herstellers auf Passwort-Basis übersteht also unsere Transformationen nicht, sondern erweist sich als unsicher. Wie aber würde sich die von uns vorgeschlagene Alternativ-Lösung mit Smartcard und Applikation im Intranet verhalten? Der Operator müsste die Software inklusive Smartcard stehlen, um sie daheim betreiben zu können. Das ist natürlich bereits deutlich riskanter als unter einem allgemeinen Account anonym zu arbeiten, zumal die durchgeführten Änderungen auf den Operator zurückgeführt werden könnten. Trotzdem – im Besitz der Smartcard und der dazu gehörigen PIN kann der Operator theoretisch die Kamera auch von zuhause aus bedienen. Eine weitere Transformation „Typwechsel“, die aus dem angenommenen PC im Intranet einen Laptop mit eingebauten Card Reader macht, zeigt, dass es gar nicht so unrealistisch ist, die Software samt Smartcard kurzzeitig zu entwenden.

Die Sicherheit kann bedeutend gesteigert werden durch eine letzte Transformation, diesmal wieder vom Typ „trennen“: Die Applikation im Intranet wird aufgeteilt in eine Multi-Tier Applikation. Die Operateure bedienen sie durch den Browser (Thin-Client), die Applikation selbst sitzt auf einem Server, der physisch abgesichert ist. Dort befindet sich auch die Smartcard mit dem Schlüssel (s. Abb. 3.15).

Diese Lösung ist bedeutend sicherer als die ursprüngliche, da das Stehlen von Software und SmartCard nun praktisch unmöglich geworden ist.

Die Nutzerkonzepte, also die Frage, wer bedient eigentlich die Software und welchen Hintergrund hat er, können also eine große Rolle bei der Sicherheitsanalyse

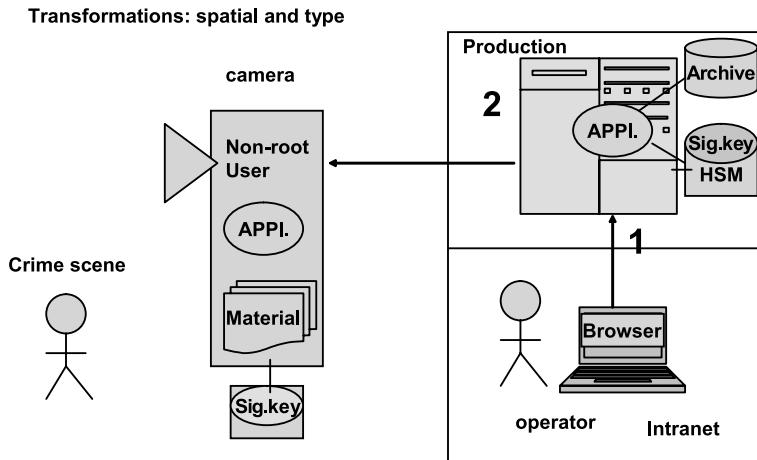


Abb. 3.15 Steuerungsapplikation mit mehreren Tiers und Thin Client

eines Systems spielen. Kann ein Nutzer, nur weil er bei der Polizei arbeitet, automatisch als „gutartig“ angesehen werden? Höchstwahrscheinlich lautet die Antwort „ja“. Aber wie groß ist das Restrisiko? Sobald man konkret daran geht, eine Abschätzung der identifizierten Risiken durchzuführen, stößt man auf zwei grundsätzliche Fragen: Wie zuverlässig sind eigentlich solche Einschätzungen? Und wie sicher ist die Einschätzung der Gegenmaßnahmen? Beide Fragen stehen direkt im Mittelpunkt unseres letzten Abschnitts in diesem Kapitel: „The People Problem“. Hier gehen wir auf die psychologischen Aspekte bei der Einschätzung von Sicherheitsrisiken ein, die vor allem beim Aufstellen der Bedrohungsmodelle eine wichtige Rolle spielen.

3.5 „The People Problem“

Peter Morville lässt in seinem kürzlich erschienenen Buch zu Suchmaschinen-technologie und Information Retrieval „Ambient Findability“ [Morv] ein Kapitel mit diesem Titel beginnen. Er stellt darin an zentraler Stelle die Abhängigkeit von Begriffen des Information Retrieval wie „Relevanz“ von ihren menschlichen Grundlagen dar. Damit steht er nicht allein: Die Ökonomen haben bereits vor zwei Jahrzehnten ihre rational fundierten Modelle des ökonomischen Handelns in Frage stellen müssen angesichts der Tatsache, dass die Menschen offensichtlich häufig ihre Entscheidungen nicht oder nicht ausschließlich rational begründen. In der IT-Security war lange Zeit allenfalls von der Psyche des Angreifers die Rede. In der letzten Zeit jedoch rückt generell der Umgang von Menschen mit Risiken in den Blickpunkt der Forschung. Das schließt die Entwickler von Software wie auch die Benutzer dieser Lösungen mit ein.

Bruce Schneier hat kürzlich einen Essay vorgelegt, der sich mit dem psychologischen Aspekt hinter der Risikoeinschätzung beschäftigt [Schn07]. Er verfolgt einen ähnlichen Ansatz wie Morville: Der Mensch ist auf Grund seiner biologischen Ausstattung heutzutage teilweise auf falsche Reaktionen festgelegt. Schneier belegt dies am Beispiel von Risikoeinschätzungen, die teilweise von den statistisch tatsächlich nachweisbaren Risiken drastisch abweichen. So werden besonders grausige Dinge als ein höheres bzw. häufigeres Risiko eingeschätzt als z. B. der alltägliche Tod im Straßenverkehr. Selbst die Wortwahl bzw. Anordnung bei der Beschreibung eines Risikos beeinflusst seine Abschätzung im empirischen Versuchen nachweislich drastisch. Durch diese Schwächen in der Risikoeinschätzung werden Menschen verführbar. Dies ist umso gefährlicher, als Sicherheit mittlerweile ein enormes Geschäft geworden ist, in dem Milliarden umgesetzt werden. So ist es Pflicht einer Sicherheitsanalyse, auch die Tauglichkeit von Abwehrmaßnahmen zu hinterfragen, genauso wie die erwähnten Risiken. Dies ist keineswegs so einfach: Viele e-Business Sites bieten beispielsweise zwei Möglichkeiten der Eingabe von vertraulicher Information (Passwörter etc.): Unverschlüsselt (Default) und verschlüsselt via SSL (s. Abb. 3.16).

Hinter dieser Trennung stehen natürlich Performance – sprich Hardwaregründe. Aber wie gefährlich ist es tatsächlich, wenn man seine persönlichen Daten ohne SSL überträgt? Wie wirkungsvoll ist ein automatischer Penetration-Test einer

The screenshot shows a web browser window with the URL <http://www.huk24.de/produkte/Mz/index.jsp?siteId=A1015&>. The page content is as follows:

- Left sidebar:** A vertical menu with items like "Wohnmobil", "Sonstige Fahrzeuge", "Rechtsschutz", "Amitshaftpflicht", "Privathaftpflicht", "Hausbesitzerhaftpflicht", "Tierhalterhaftpflicht", "Hausrat", "Unfall", "Wohngebäude", "Kranken", "Reisekranken", "Risikoleben", "Berufsunfähigkeit", "Bausparen", "Baufinanzierung", and "Angebotsspeicher".
- Main Content Area:**
 - A banner: "Leistungen zu günstigen Konditionen. Mit **persönlichem Ansprechpartner** im Schadensfall."
 - A callout box: "Ein Schaden pro Jahr frei! PKW-Versicherung" with text: "MIT dem neuen Rabattschutz in der PKW-Versicherung haben Sie einen Schaden pro Jahr frei. Trotz dieses Schadens fahren Sie in der Haftpflicht und Vollkasko weiterhin in der günstigen Schadenfreiheitsklasse." and a link "Berechnen Sie Ihren Beitrag jetzt!"
 - A section: "Top-Fragestellungen zur PKW-Versicherung" with links: "Wie können Sie Ihre bisherige KFZ-Versicherung kündigen und zur HUK24 wechseln?", "Wie erhalten Sie von der HUK24 eine Deckungskarte ("Doppelkarte")?", and "Wie ermitteln Sie die richtige Schadenfreiheitsklasse?"
 - A section: "Bitte halten Sie Folgendes bereit:" with links: "KFZ-Schein (neu: Zulassungsbescheinigung Teil I)", "Versicherungspolicie oder die letzte Beitragsrechnung", and "Kilometerstand Ihres PKW".
 - A section: "Direkt zur unverbindlichen Angebotsberechnung:" with two buttons: "verschlüsselter Verbindungsauflaufbau" and "unverschlüsselter Verbindungsauflaufbau", each with a "Zur Berechnung" link.
 - A note at the bottom: "Wir empfehlen die Wahl der verschlüsselten Verbindung, da nur hierdurch Ihre Daten wirkungsvoll vor unbefugter Kenntnisnahme und Missbrauch geschützt werden. Mehr Informationen..."
- Right sidebar:**
 - "Anreisezeit" with links: "Vertragsantrag", "Vertragsänderungen", "Bescheinigungen", "mehr Informationen".
 - "Ihre Daten werden verschlüsselt übertragen."
 - "Kunden-Login"
 - "Jetzt aktuell" with links: "Urlaubszeit", "So beantragen Sie eine Grüne Karte", "So ändern Sie Ihren Vertrag".
 - "Kunden werben Kunden" with text: "15 EUR für jede Empfehlung!"

Abb. 3.16 Screenshot der Webseite der HUK24-Versicherung mit verschlüsseltem und unverschlüsseltem Eingang (www.huk24.de)

Web-Applikation? Wie misstrauisch sollten Mitarbeiter (auch untereinander) sein? Macht es Sinn, Mitglieder eines Frequent-Flyer-Programms nach einer biometrischen Authentisierung mit Datenbankabgleich ohne weitere Kontrolle an Bord zu lassen? Nehmen Terroristen nicht an Frequent-Flyer-Programmen teil? Macht ein vollständiger „Background Check“ neuer Mitarbeiter sicherheitstechnisch Sinn oder ist er Teil einer Sicherheitshysterie?

Für alle vorgeschlagenen Sicherheitsmassnahmen sind nach Schneier die folgenden Fragen zu stellen:

- Erfüllt die Maßnahme wirklich ihren Zweck?
- Ist sie verhältnismäßig im Vergleich zum Risiko, das sie abwenden soll?
- Hat sie starke Neben- oder Seiteneffekte?
- Beruhigt sie nur die Nerven oder hilft sie tatsächlich?
- Sind diejenigen, die sie empfehlen auch diejenigen, die davon (z. B. finanziell) profitieren?
- Handelt es sich nur um so genannte CYA Maßnahmen (Cover Your Ass), also Maßnahmen, die lediglich die Verantwortung weiter schieben sollen, ohne wirklichen Nutzen?

Neben der Abschätzung der Sicherheitsmassnahmen und ihrer Wirksamkeit muss die Sicherheitsanalyse neben den psychologischen Eigenheiten der möglichen Angreifer auch die der normalen Nutzer sowie die Entwickler der Software einbeziehen, also ein so genanntes *User Conceptual Model* bilden. Im Rahmen einer Sicherheitsanalyse für eine Applikation sollten demnach auch für Nutzer und Entwickler deren mentale Modelle bezüglich der Applikation und ihres Umfeldes aufgestellt und bewertet werden. Diese sollten dann in die jeweiligen Bedrohungsmodelle einfließen. Leider ist das Menschenbild der Informatik diesen Anforderungen noch längst nicht gewachsen, wie Abb. 3.17 zeigt.

Zu welchen erstaunlichen Schlussfolgerungen man mit Hilfe dieser konzeptuellen Modelle kommen kann, sei abschließend anhand eines kleinen psychologischen Exkurses erklärt. Es geht dabei um die grundsätzliche Einschätzung von Risiken: Wie immer bei Risikoanalysen, die auf Basis von Wahrscheinlichkeit und Schadenshöhe erstellt werden, bietet der Fall „geringe Wahrscheinlichkeit, aber maximaler Schaden“ besondere Probleme bei der Abschätzung des Risikos (so ist etwa

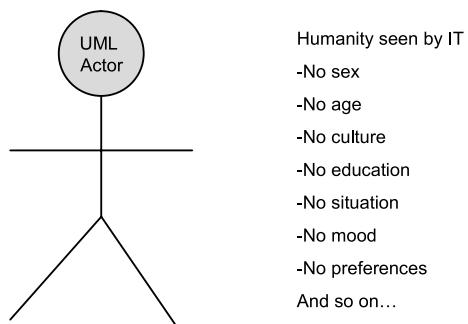


Abb. 3.17 Menschenbild der Informatik

bei Kernkraftwerken die Wahrscheinlichkeit eines GAUs sehr klein, der Schaden aber unermesslich – wie will man so etwas rational bewerten?) Darüber hinaus ist bekannt, dass Menschen seltene oder exotische Risiken über- und gewohnte Risiken unterbewerten. Auch ändert sich die Risikoeinschätzung mit dem Grad der Vertrautheit. Ein besonders gefährliches Verfahren, mit Risiken extremer Konsequenz umzugehen, ist das so genannte *Risikointegral*. Hier wird wie in der allgemeinen Risikoabschätzung auch ein Produkt aus Häufigkeit und Schaden gebildet, dabei allerdings übersehen, dass der Schaden unkalkulierbar groß ist. Ein klassisches Beispiel nennt der Verhaltensforscher Bernt Spiegel ([Spie]): Eine unübersichtliche Linkskurve auf einer selten befahrenen Landstrasse – soll überholt werden oder nicht? Dafür spricht die geringe Wahrscheinlichkeit, dass ein Fahrzeug entgegenkommt – dagegen, dass die Konsequenz ein wahrscheinlich tödlicher Unfall ist. Leider entscheiden sich auch in solchen Situationen immer wieder Menschen zur Integralbildung der beiden Faktoren – somit wird eben tatsächlich manchmal an so einer Stelle überholt. Bei der Erstellung von Sicherheitsanalysen für IT-Systeme sollte man sich dieses psychologischen Aspekts bewusst sein.

Kapitel 4

Sicherheitsdienste

Nachdem wir in den voran gegangenen Kapiteln die Vorgehensweise beim Erstellen einer Sicherheitsanalyse allgemein und an Beispielen betrachtet haben, wollen wir nun daran gehen, zu untersuchen, wie die sich aus der Analyse ergebenden Sicherheitsanforderungen implementiert werden können.

Wir beginnen damit, einige wichtige Eigenschaften eines IT-Systems, die den sicheren Betrieb des Systems ermöglichen und sicherstellen, begrifflich zu klären. Einige dieser Eigenschaften, wie etwa die sichere Authentifikation von Nutzern, sind in unseren Fallstudien bereits explizit oder implizit erwähnt worden. Diese Eigenschaften werden als *Sicherheitsdienste* (Security Services) bezeichnet (vgl. auch [Eck]). An dieser Stelle beschreiben wir zunächst grundsätzliche Eigenschaften und Einsatzmöglichkeiten von Sicherheitsdiensten, ohne dabei (zumindest im Moment) im Detail auf die technische Umsetzung einzugehen.

4.1 Authentifikation

In den Beispielen des vorigen Kapitels trat wiederholt die Situation auf, dass ein Kunde dem System eine Identität mitteilte und aufgrund dieser Identität den Zugang zu gewissen Diensten erwartete. Woher aber soll das System (hier der Webserver) wissen, ob sich hinter der angegebenen Identität tatsächlich der dazugehörige Kunde verbirgt? Der Sicherheitsdienst, der es einem System erlaubt, eine solche behauptete Identität zu verifizieren, heißt *Authentifikation*. Eine erfolgreiche Authentifikation bindet eine Identität an ein Subjekt, also eine Entität, die im System aktiv werden kann. Es ist klar, dass so gut wie jedes System, welches zum Beispiel eine Interaktion mit dem User bietet, einen solchen Sicherheitsdienst benötigt, um sicher gehen zu können, dass ein User, der bestimmte Dienste in Anspruch nehmen möchte, dazu auch berechtigt ist. In den meisten Fällen gilt dies auch umgekehrt, das heißt: Auch der Client muss verifizieren können, ob er tatsächlich mit dem gewünschten Server verbunden ist. Werden im Zuge einer Authentifikation sowohl die Identität des Client als auch die des Servers verifiziert, spricht man von *zweiseitiger Authentifikation*, im Gegensatz zu einer *ein-*

seitigen Authentifikation, falls sich nur einer der Kommunikationspartner authentifiziert.

Eine Authentifikation kann auf verschiedene Arten erfolgen: Durch Besitz (zum Beispiel eines Ausweises), durch Kenntnis eines Geheimnisses (zum Beispiel Kenntnis einer PIN oder eines Passworts) oder durch weitere, einer Person inhärente Eigenschaften wie Aussehen, Klang der Stimme, Fingerabdrücke, etc. Im Internet-Umfeld spielt natürlich die Authentifikation durch Wissen die bei Weitem größte Rolle. Im täglichen Leben begegnen uns jedoch die anderen Authentifikationsmöglichkeiten wesentlich häufiger. So authentifiziert man sich bei einem Telefongespräch gegenüber seinen Gesprächspartner über die Stimme. Auch Kombinationen, wie etwa die Authentifikation am Geldautomaten, die sowohl durch Besitz (der ec-Karte) als auch durch Wissen (PIN-Eingabe) erfolgt, kommen vor und werden auch im Internet-Umfeld bei erhöhten Sicherheitsanforderungen, wie etwa beim e-Banking, eingesetzt. Zunächst aber zu etwas einfacheren Authentifikations-schemata, die auf Passwörtern beruhen:

4.1.1 Authentifikation durch Passwörter

Ein Passwort besteht typischerweise aus 6 bis 10 Zeichen und wird als gemeinsames Geheimnis zwischen Client und Server zur Authentifikation durch Wissen genutzt. Das Passwort sollte zwischen Client und Server über einen vertrauenswürdigen, d. h. vor Abhören, unberechtigter Modifikation und Impersonifikation geschützten Kanal ausgetauscht werden. Ein solcher Kanal wird beispielsweise durch die Briefpost, im Internet zumeist durch das später behandelte Sicherheitsprotokoll SSL zur Verfügung gestellt.

Bei der eigentlichen Authentifizierung gibt der Client seine behauptete Identität, zum Beispiel eine UserID und das dazu gehörige Passwort zur Authentifizierung an. Der Server verifiziert anhand der bei ihm gespeicherten Daten, ob das angegebene Passwort zur UserID gehört. Sowohl bezüglich der Übertragung der Passwörter als auch bezüglich der Speicherung beim Server existieren unterschiedlich sichere Varianten. So können UserID und Passwort im Klartext (z. B. via http) oder verschlüsselt (via https, also http über SSL) übertragen werden. Fängt ein Angreifer die unverschlüsselt übertragene UserID und das Passwort (die bereits erwähnten *Credentials*) eines Users ab, so kann er gegenüber dem Server als dieser User auftreten, ihn also „impersonifizieren“. Auch der Server selbst könnte mit Hilfe der Credentials den User gegenüber anderen Systemen impersonifizieren, sofern dort die gleichen Credentials verwendet werden.

Die offensichtliche Methode der Speicherung von Passwörtern auf dem Server ist die, UserID und Passwörter im Klartext in einer Schreib/Lese-geschützten Datei abzulegen. Dies macht die Passwort-Datei natürlich zu einem überaus lohnenden Ziel für Angreifer von außen. Gravierender ist jedoch die Tatsache, dass Systemadministratoren oder andere Angestellte mit Superuser-Privilegien vollen Zugriff auf die Passwort-Datei haben. Zudem werden bei einem Backup die Pass-

wörter ebenfalls im Klartext gespeichert. Aus diesen Gründen werden im Normalfall die Passwörter bei der Ablage auf dem Server speziell geschützt. Dies geschieht in der Regel dadurch, dass das zu einer UserID gehörige Passwort nicht im Klartext, sondern in verschlüsselter oder „gehaschter“ Form, das heißt nach dem Anwenden einer Hash-Funktion (siehe hierzu das Kapitel „Kryptografische Algorithmen“) abgelegt werden. Wie Verschlüsselungsalgorithmen besitzen Hash-Funktionen unter anderem die wichtige Eigenschaft, dass aus dem Output, also dem Hashwert, nicht auf den Input, in diesem Fall also auf das Passwort im Klartext geschlossen werden kann. Bei Erhalt eines Paars (UserID, Passwort) bildet der Server zunächst den Hashwert des Passworts und sieht dann in seiner Passwort-Datei nach, ob der dort für die jeweilige UserID gespeicherte Hashwert mit dem gerade gebildeten Hashwert übereinstimmt. Auch diese Art des Schutzes ist jedoch nicht gegen Angriffe von Außen gefeit: Gelingt es einem Angreifer, sich in Besitz der Datei mit den gehaschten Passwörtern zu bringen, so kann er nach und nach eine große Menge von Passwörtern hashen und die Ergebnisse mit den Einträgen in der Passwort-Datei vergleichen. Da Passwörter häufig von den Usern selbst gewählt werden, besteht eine hohe Wahrscheinlichkeit, dass das Passwort aus einer wohlbekannten Menge von Wörtern stammt, dem so genannten Wörterbuch oder „Dictionary“. Man spricht deshalb auch von *Dictionary Attacks*. Selbst wenn ein Passwort nicht in einem Wörterbuch steht, so ist es aufgrund der begrenzten Anzahl von Zeichen eines Passworts durchaus denkbar, dass ein hinreichend motivierter Angreifer alle möglichen Passwörter durchprobiert, die eine gewisse Länge nicht überschreiten. In [MOV] wird die Zeit, um alle Passwörter durchzuprobieren, die aus höchstens sechs der 95 auf einer PC-Tastatur vorhandenen Zeichen bestehen, mit 4,7 Jahren auf einem „High-End PC“ angegeben. Diese Schätzung stammt allerdings von 1997 – mittlerweile dürften für dieselbe Aufgabe nur noch wenige Monate erforderlich sein – auf einem einzigen PC wohlgemerkt.

Als Gegenmaßnahme gegen Dictionary Attacks wird häufig das so genannte *Salting* eingesetzt. Dabei wird zusammen mit dem eigentlichen Passwort noch eine Zufallszahl, das „Salt“ gehasht. Das Salt wird zwar zusammen mit der UserID im Klartext abgelegt, verhindert aber dennoch, dass bei einer Dictionary Attack auf die Passworddatei bereits vorab berechnete Hashwerte eines Wörterbuchs verwendet werden können. Zudem müssen für jede UserID die Hashwerte wieder neu berechnet werden. Somit erhöht Salting zwar nicht die Sicherheit eines einzelnen Nutzerkontos, wohl aber die Sicherheit der gesamten Passworddatei.

Ist ein Passwort erst einmal einem Angreifer bekannt geworden, so kann er es so lange zur Impersonifikation des betroffenen Nutzers gebrauchen, bis das Passwort geändert wird. Je nach den Rechten des betroffenen Nutzers auf dem jeweiligen Zielsystem ergeben sich hieraus natürlich weitere Angriffsmöglichkeiten. Eine Möglichkeit zur weiteren Erhöhung der Sicherheit bei Passwort-basierten Authentifikationsschemata besteht daher darin, die Lebensdauer der Passwörter so weit wie möglich bis hin zu Einmal-Passwörtern, die nur für eine einzige Transaktion gültig sind, zu reduzieren. Zur Erzeugung von Einmal-Passwörtern existieren verschiedene Möglichkeiten. Diese basieren jedoch häufig auf der Annahme eines synchronisierten Zählers oder synchronisierter Uhren auf Client- und Serverseite,

was sich in der Praxis mitunter als schwierig zu realisieren herausstellt. Die einfachste und wohl auch am weitesten eingesetzte Methode stellen Listen von Einmal-Passwörtern dar, die auf Client- und Server-Seite vorliegen und sukzessive invalidiert werden. Diese Lösung wird zum Beispiel im e-Banking in der Form von TAN-Listen eingesetzt, die serverseitig generiert werden und dem User per Brief zugesandt werden.

4.1.2 Authentifikation durch Challenge-Response-Verfahren

Die im vorigen Abschnitt vorgestellten Authentifikationsverfahren leiden alle unter der Schwäche, dass das zur Authentifikation nötige Geheimnis vom Client tatsächlich preisgegeben und im Klartext zum Server übermittelt wird. Werden keine weitergehenden Maßnahmen zur Absicherung des Kommunikationskanals getroffen, besteht immer die Gefahr, dass ein Passwort durch einen Angreifer abgefangen und zur Impersonifikation genutzt werden kann. Dies wird durch *Challenge-Response-Verfahren* vermieden. Hierbei sendet der Server (in diesem Zusammenhang auch als *Verifier* bezeichnet) dem Client, der seine Identität nachweisen möchte (auch *Prover* genannt) eine Zufallszahl, die Challenge. Diese Zufallszahlen sollten nicht vorhersagbar sein (ansonsten könnte ein Angreifer versuchen, die dazu gehörigen Responses voraus zu berechnen) und sich nicht wiederholen, da ansonsten eine einmal abgehörte Response wieder eingespielt und zur Impersonifikation verwendet werden könnte („Replay Attack“). Aus der Challenge berechnet der Prover mit Hilfe eines Geheimnisses K und einer Hash- oder Verschlüsselungsfunktion eine Response, die an den Verifier übermittelt wird (s. Abb. 4.1). Natürlich darf es nicht möglich sein, aus der Challenge und der Response auf das verwendete Geheimnis zu schließen, was hohe Anforderungen an die zur Berechnung der Response verwendeten Algorithmen stellt. Das Geheimnis liegt zumeist auf beiden Seiten, also beim Prover und beim Verifier vor. In diesem Fall spricht man auch von einem *symmetrischen Challenge-Response-*

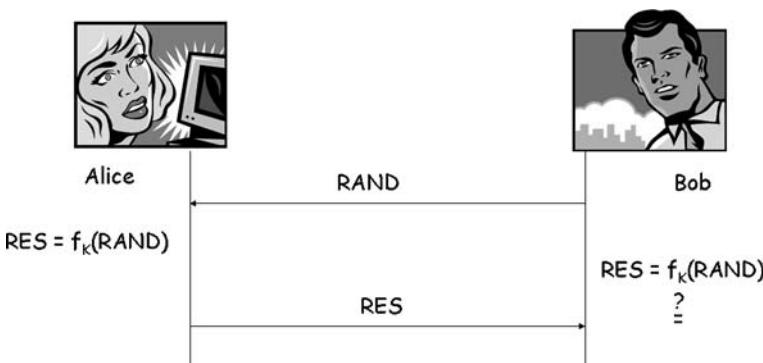


Abb. 4.1 Symmetrisches Challenge-Response-Verfahren

Verfahren. Der Verifier kann in diesen Fällen eine analoge Operation wie zuvor der Prover durchführen und somit die Richtigkeit der Response überprüfen.

Neben den symmetrischen Challenge-Response-Verfahren können auch asymmetrische Verfahren für die Authentifikation eingesetzt werden. Die Authentifikation beruht bei asymmetrischen Verfahren nicht auf einem gemeinsamen Geheimnis, das sowohl Client als auch Server bekannt ist. Vielmehr nutzt der Client zum Nachweis seiner Identität ein nur ihm bekanntes Geheimnis, den so genannten *Private Key*, indem er entweder eine verschlüsselte Challenge mittels seines Private Keys entschlüsselt und die Challenge dann unverschlüsselt zurück schickt oder aber eine ihm zugesandte Klartext-Challenge mit dem Private Key verschlüsselt (bzw. „signiert“ wird). Die Verifikation durch den Server geschieht über den zum Private Key gehörigen *Public Key*. Die Echtheit bzw. Authentizität der Public Keys muss durch die bereits angesprochenen Zertifikate verbürgt werden. An dieser Stelle gehen wir nicht weiter auf asymmetrische Authentifikationsprotokolle ein, sondern verweisen für eine vertiefte Diskussion auf das Kapitel „Kryptografische Algorithmen“, da uns erst dort die notwendigen kryptografischen Grundlagen zur Verfügung stehen.

4.1.3 Kontext-Weitergabe, Delegation und Impersonation

Nachdem wir in den voran gegangenen Abschnitten grundlegende Techniken zur Authentifikation betrachtet haben, betrachten wir nun auch kurz weiter gehende, mit der Authentifikation in verteilten Systemen zusammen hängende Aspekte, die die Behandlung bereits authentisierter User (so genannter *Principals*) durch eine komplexe, aus mehreren Subsystemen bestehende Hard- und Software-Architektur betreffen. Ein einfaches Beispiel wäre ein User, der sich bei einem Webserver per UserID und Passwort authentifiziert hat und nun eine Suche in einer Datenbank durchführen möchte. Der Webserver gibt also die Suche bei der Datenbank im Namen des Users in Auftrag, die ihrerseits prüft, ob sich der User hierfür erneut authentisieren muss und ob er zu der gewünschten Suche berechtigt ist. In dieser Situation sind verschiedene Szenarien denkbar:

Kontext-Weitergabe bezeichnet die Möglichkeit, die Identität eines authentifizierten Users an weitere Subsysteme weiter zu geben. Das Subsystem (die Datenbank) initiiert hierbei keine weitere Authentifikation, vielmehr vertraut es der übergebenden Entität (dem Webserver), eine korrekte und verifizierte Identität zu übertragen. Im Gegensatz dazu bezeichnet *Delegation* die Weitergabe einer Identität zusammen mit den dazu gehörigen Credentials. Somit hat auch das empfangende System (hier die Datenbank) die Möglichkeit zur erneuten Authentifikation. Bei der *Impersonation* schließlich nimmt ein Subsystem (hier der Webserver) mit Hilfe der Credentials die Identität des Principals an und handelt fortan in dessen Namen. Somit gehen alle Rechte des Principals unmittelbar auf das impersonifizierende Subsystem über, wodurch nicht immer klar ist, wer tatsächlich für einen bestimm-

ten Funktionsaufruf verantwortlich ist. Eine vertiefte Diskussion dieser Begriffe werden wir im Kapitel zu „Sicherheit in verteilten Systemen“ durchführen.

4.2 Vertraulichkeit

Vertraulichkeit ist der Sicherheitsdienst, der es ermöglicht, Informationen vor Unbefugten geheim zu halten und sie nur einem wohl definierten Kreis von autorisierten Personen zugänglich zu machen. Es kann sich hierbei um sehr kurze Stücke von Information, wie etwa Passwörter, handeln, oder auch um sehr lange Bitstrings, wie etwa Kinofilme oder Musikstücke in digitaler Form, die nur für einen berechtigten Personenkreis (d. h. denen, die dafür bezahlt haben) nutzbar sein sollen. Historisch betrachtet, ist die Vertraulichkeit der älteste Sicherheitsdienst: Schon bei den alten Griechen und Römern existierten algorithmische und auch mechanische Verfahren, um Informationen, die im Krieg durch Boten übermittelt wurden, vor dem jeweiligen Gegner geheim zu halten.

Neben kryptografischen, also Verfahren, welche die Klartext-Nachricht in einen für den Gegner unlesbaren Chiffretext verwandeln, sind in der Vergangenheit auch immer wieder steganografische, das heißt, das bloße Vorhandensein einer Nachricht verschleiernde Verfahren zum Einsatz gekommen. Zuletzt gelangten steganografische Algorithmen, die zum Beispiel eine Nachricht in den zur Darstellung der Bildinformation ungenutzten Bits eines digitalen Bildes verstecken, im Zuge der Diskussion um eine staatliche Regulierung und Kontrolle des Einsatzes kryptografischer Verfahren zur besseren Terrorismusbekämpfung in das Blickfeld der Öffentlichkeit. Im Rahmen dieses Buches werden wir uns jedoch auf kryptografische Verfahren zum Erreichen von Vertraulichkeit konzentrieren. Wie wir noch sehen werden, lassen sich mit Hilfe kryptografischer Verfahren durchaus auch noch andere Sicherheitsdienste realisieren.

4.3 Integritätsschutz

Daten, die in digitaler Form über ein offenes Netz verschickt werden, können von jedem, der Zugriff auf das Netz hat, verändert werden. Es ist prinzipiell unmöglich, solche Veränderungen zu verhindern, wenn man nicht das Netz als solches vor physikalischem Zugriff schützen will. Wie das Beispiel der elektronischen Bahnhafkarte gezeigt hat, sind aber auch Attacken durch den legitimen Empfänger einer Nachricht denkbar, wenn es ihm gelingt, eine bereits Nachricht in seinem Sinne zu verändern.

Ziel des Sicherheitsdienstes „Integritätsschutz“ kann es daher nicht sein, Veränderungen an sich zu verhindern, sondern vielmehr, diese für den Empfänger einer Nachricht erkennbar zu machen. In seinem Buch „Practical Cryptography“ [Schn04] vertritt Bruce Schneier die Ansicht, dass der Integritätsschutz sogar noch

wichtiger einzustufen als die Vertraulichkeit, da die Verwendung von verfälschten Informationen potenziell noch gravierendere Auswirkungen haben könne als die Weitergabe korrekter Informationen an Unbefugte.

In der Netzwerktechnik existieren diverse Arten von Prüfsummen, die eine zufällige Änderung eines Datenpaketes durch einen Übertragungsfehler erkennbar und sogar korrigierbar machen. Diese Prüfsummen sind jedoch nicht geeignet, wenn es darum geht, absichtliche Modifikationen durch einen Angreifer zu erkennen. Dieser könnte nämlich die Prüfsumme seiner Änderung anpassen. Deshalb sind in der IT-Security kryptografische Prüfsummen gefragt, die nicht ohne Kenntnis eines Geheimnisses gebildet und somit auch nicht gefälscht werden können.

4.4 Nicht-Abstreitbarkeit

Nehmen wir an, die Fahrplanauskunft auf bahn.de sei ein für die Kunden kostenpflichtiger Dienst. Nehmen wir weiterhin an, ein Kunde bestreitet, die in den Log-Dateien von bahn.de ausgewiesenen Anfragen getätigt zu haben und behauptet stattdessen, diese Dateien seien von der Deutschen Bahn AG manipuliert. Wie kann ein Anbieter seinen Kunden in einem solchen Streitfall nachweisen, einen kostenpflichtigen Dienst in Anspruch genommen zu haben? Ist eine solche Nachweisbarkeit gefordert, so muss die Anmeldung bei dem Dienst für den Kunden *nicht-abstreitbar* sein. Nicht-Abstreitbarkeit (oder auch *Verbindlichkeit*) bedeutet, dass ein Teilnehmer an einer Kommunikation dem anderen *beweisen* kann, dass dieser eine bestimmte Nachricht geschickt hat. Dies bedeutet eine wesentlich weit reichendere Anforderung als die bloße Nachrichtenauthentizität, bei der sich ein Teilnehmer zwar selbst davon überzeugen kann, dass eine Nachricht von einem bestimmten Absender stammt, diesen Nachweis aber unter Umständen nicht auch vor Dritten glaubhaft machen kann, da er den entsprechenden Nachweis auch selbst hätte erzeugen können.

4.5 Verfügbarkeit

Ein Dienst, mit dem ein Anbieter, wie etwa bahn.de mit seinem Portal, Geld verdienen will, oder für den Kunden bereits bezahlt haben, sollte ständig für berechtigte Parteien verfügbar sein. Diese Anforderung klingt trivial, zieht aber in der Praxis häufig weit reichende Konsequenzen in der Server-Architektur nach sich. Attacken auf die Verfügbarkeit bekannter Dienste werden unter dem Stichwort *Denial-of-Service-Angriffe* (DoS) zusammen gefasst. Heutzutage sind damit zu meist „Distributed“ DoS (DDos) Attacken gemeint, bei denen sehr viele (meist durch Malware kontaminierte) Clients in einer konzertierten Aktion zusammenwirken, um einen Ziel-Server zu überlasten.

Um die Antwortzeiten bei vielen gleichzeitig auftretenden Anfragen gering zu halten und eine Überlastung der Server zu vermeiden, werden häufig so genannte *Load-Balancing-Mechanismen* eingesetzt. Eine einfache Möglichkeit des Load-Balancing, das „Round-Robin-DNS“, besteht darin, mehrere IP-Adressen einer Server-Farm einem einzigen Domain Name zuzuordnen. Als Antwort auf eine entsprechende Anfrage durchläuft der DNS-Server alle dem Domain Name zugeordneten IP-Adressen, bevor er eine Adresse zum zweiten Mal ausgibt. Ein mögliches Problem bei dieser einfachen und kosteneffektiven Lösung liegt darin begründet, dass der DNS-Server keinerlei Rücksicht auf die tatsächliche Auslastung der einzelnen Server nimmt. Auch andere denkbare Load-Balancing-Mechanismen wie etwa das dynamische Entfernen überlasteter Server aus der DNS-Liste müssen sich mit der Problematik auseinandersetzen, die entsteht, wenn ein Client mit einem der Backend-Server – etwa im Zuge eines SSL-Handshakes – einen Schlüssel vereinbart hat und in der Folge über einen sicheren Kanal mit dem Server kommuniziert hat. Was soll geschehen, wenn bei einem weiteren Request demselben Client vom Load-Balancing-Schema ein anderer Server zugewiesen wird? Wir werden auch auf diese Problematik im Zusammenhang mit SSL weiter unten noch eingehen.

4.6 Authorisierung

Mit diesem Sicherheitsdienst verlassen wir das Gebiet der Network Security und betreten das Gebiet der Software Security. Die Dienste Authentifikation, Vertraulichkeit und Integritätsschutz können für einen sicheren Tunnel von einem System zum anderen sorgen, die Authorisierung klärt, was dann passiert: Nachdem sich eine Entität authentifiziert hat, das heißt ihre Identität verifiziert worden ist, ist vom System zu entscheiden, über welche Rechte diese Entität verfügt, das heißt insbesondere, ob sie das Recht hat, den angeforderten Service zu nutzen. Dieser Entscheidungsprozess heißt *Authorisierung* und wird normalerweise durch die Entität durchgeführt, welche den betreffenden Dienst anbietet. Bei der Rechtevergabe lassen sich vier grundsätzliche Modelle unterscheiden, die wir im Folgenden diskutieren. Allerdings kann auch der Fall eintreten, dass die betreffenden Zugriffsrechte von der Institution vergeben werden, zu der der Principal gehört, welcher den Dienst anfordert. In diesem Fall kann der Principal selbst durch ein passendes, von seiner Institution ausgestelltes Attributzertifikat (siehe auch hierzu das Kapitel „Kryptografische Algorithmen“) nachweisen, dass er die erforderlichen Rechte besitzt.

4.6.1 DAC – Discretionary Access Control

In diesem Modell hat jede Ressource (also zum Beispiel jede Datei) einen Inhaber, der entscheidet, welche Nutzer bzw. Nutzergruppen außer dem Inhaber auf die Ressource zugreifen darf, und welche Operationen im Einzelnen erlaubt sind.

	Alice	Bob	Users	Others
File A	read, write, execute, own	read	-	-
File B	read	read, write, execute, own	read	read

Abb. 4.2 Zwei einfache ACLs

Diese Informationen sind in einer *Access Control List* (ACL) für jede einzelne Ressource gespeichert, die ihrerseits aus Access Control Entries (ACEs) besteht. Erbittet ein authentisierter User Zugriff auf eine Ressource, wird anhand der ACL für diese Ressource entschieden, ob der Zugriff erlaubt ist oder nicht. Auch die klassische Zugriffskontrolle unter UNIX funktioniert auf diese Weise. Prozesse, die auf Ressourcen zugreifen wollen, werden dabei mit den Usern identifiziert, die sie initiiert haben, d. h. der Prozess besitzt die Rechte des Users, der ihn ausgelöst hat – eine der Hauptursachen für die Gefährlichkeit von Malware. Abbildung 4.2 zeigt ein kleines Beispiel für mögliche ACLs.

Alice und Bob können ihre Files gegenseitig lesen, Bob hat sein File zusätzlich noch für die Gruppe der User auf dem System und alle anderen zum Lesen frei gegeben. Bei komplexen Systemen ist es allerdings sinnvoll, nicht für jede einzelne Ressource eine ACL zu führen, sondern die Ressourcen in einem hierarchischen System (z. B. einer Ordnerstruktur in einem Filesystem) anzugeordnen und ACLs durch gewisse Vererbungsregeln von einem hohen Level auf die tieferen Level der Hierarchie zu übertragen. Dabei eventuell auftretende Konflikte zwischen den ACLs sind mit einer übergeordneten Security Policy aufzulösen.

Authorisierungen mit dem DAC Modell sind einfach zu verstehen und zu implementieren – sie besitzen aber auch gravierende Nachteile: Sie sind schwer zu administrieren und fehlerhafte, insbesondere solche ACLs, die nicht restriktiv genug sind, sind schwer zu aufzufinden: Ein User des Systems wird sich kaum beschweren, dass ihm zu viele Rechte eingeräumt werden (vgl. dazu auch [Pet], S. 31). Als generelle Regel bei der Aufstellung von ACLs sollte das „Principle of Least Privilege“ (POLA) gelten, das heißt, jedem Nutzer bzw. Prozess sollte nur das Minimum an Rechten eingeräumt werden, das er zum Ausführen einer Operation benötigt, für die er berechtigt ist. Zum Beispiel sollte der Kunde einer Bank nur das Recht besitzen, seine eigenen Kontodaten einzusehen, nicht jedoch die der anderen Kunden. Ein Kundenberater hingegen sollte das Recht haben, die Daten aller von ihm betreuten Kunden einzusehen, nicht jedoch auch die Daten von anderen Beratern betreuter Kunden.

4.6.2 MAC – Mandatory Access Control

Der MAC-Ansatz ist das allgemeinste der hier diskutierten Modelle zur Authorisierung. Der Zugriff durch einen Akteur (Subjekt) auf eine Ressource (Objekt) wird durch eine systemweite Security-Policy geregelt. Subjekte können sowohl

	File A	File B	Process 1	Process 2
Process 1	read, write, execute, own	read	read, write, execute, own	write
Process 2	append	read, own	read	read, write, execute, own

Abb. 4.3 Beispiel einer ACM nach [Bish]

Nutzer als auch Prozesse sein, so dass eine sehr feingranulare Rechtevergabe möglich wird. Die Rechte, die ein Subjekt s beim Zugriff auf ein Objekt o laut Policy besitzt, können durch einen Eintrag $a[s,o]$ in einer Matrix A abgebildet werden. Die so definierte Matrix wird auch als *Access Control Matrix* (ACM) bezeichnet (s. Abb. 4.3]).

Natürlich entsteht auch bei der Nutzung von ACLs unter dem DAC-Modell eine Matrix. Allerdings stehen dort die Ressourcen (also die Objekte) in den Zeilen der Matrix. Zudem ist die Menge der möglichen Subjekte im MAC-Modell viel größer als im DAC-Modell. Der wichtigste Unterschied besteht jedoch darin, dass die ACM (bzw. die zu Grunde liegende Policy) nicht durch einzelne Nutzer verändert werden kann, sondern nur durch speziell dazu berechtigte User. Um zu verhindern, dass solche Nutzer ihr Recht missbrauchen, um ihre eigenen Zugriffsrechte zu vergrößern und so „allmächtig“ zu werden, sollte bei der Änderung der ACM das Vier-Augen-Prinzip gelten.

4.6.3 RBAC – Role Based Access Control

Sowohl beim MAC- als auch beim DAC-Ansatz können die Nutzer eines Systems in Gruppen mit unterschiedlichen Rechten eingeteilt werden. Das macht es einfach, bestehende organisatorische oder regionale Strukturen im Software-System abzubilden. Was die Einteilung in Gruppen jedoch im Normalfall nicht leistet, ist eine Einteilung der User nach Kompetenzen bzw. Verantwortlichkeiten. Während eine regionale Zuordnung oder die Zugehörigkeit zu einer Organisationseinheit permanent besteht, können die Kompetenzen und Verantwortlichkeiten eines Principals, beispielsweise im Rahmen einer Stellvertreter-Regelung, durchaus öfters wechseln. In diesen Fällen kann eine rollenbasierte Authorisierung die beste Lösung darstellen: Eine Rolle spiegelt eine Kompetenz eines Nutzers, also etwas, was ein Nutzer tun darf. Im RBAC-Modell können die Principals zeitlich befristet unterschiedliche Rollen annehmen. Eine Security Policy legt fest, welche Rolle welche Rechte besitzt. In fortgeschrittenen RBAC-Lösungen werden Hierarchien von Rollen festgelegt, mit dem Vorteil, dass gewisse Rechte von untergeordneten Rollen übernommen werden können. So kann die übergeordnete Rolle „Teamleiter“ alle Rechte der Rolle „Teammitglied“ erhalten, ohne dass diese Rechte explizit vergeben werden müssen.

4.6.4 Multi-Level Security

In diesem Modell werden den Subjekten und Objekten unterschiedliche Security-Level zugeordnet, um ihre Sensitivität und Sicherheitsrelevanz zu beschreiben. Die Policy kann dann mit Hilfe der Level formuliert werden. Ein bekanntes Beispiel bildet das hauptsächlich an dem Erreichen von Vertraulichkeit ausgerichtete Bell-LaPadula Modell [BelLa]. Es besteht im Wesentlichen aus den folgenden beiden Regeln:

- Das Subjekt s besitzt das Leserecht bzgl. Objekt o , falls $\text{Level}(s) \geq \text{Level}(o)$ (No Read Up)
- Das Subjekt s besitzt das Schreibrecht bzgl. Objekt o , falls $\text{Level}(o) \geq \text{Level}(s)$ (No Write Down)

Abbildung 4.4 zeigt ein Beispiel für diese Policy mit drei Security Levels „Top Secret“, „Secret“ und „Confidential“.

Auf diese Weise wird verhindert, dass Informationen von einem höheren Sicherheitslevel zu einem niedrigeren Level fließen können, selbst wenn ein Subjekt eines höheren Levels, etwa durch einen Virus, kompromittiert sein sollte. Die beiden Grundregeln können durch eine DAC-Komponente ergänzt werden, etwa in der Form, dass Subjekt s das Objekt o lesen darf, falls $\text{Level}(s) \geq \text{Level}(o)$ und die ACL für o an der Stelle s das Recht „read“ enthält.

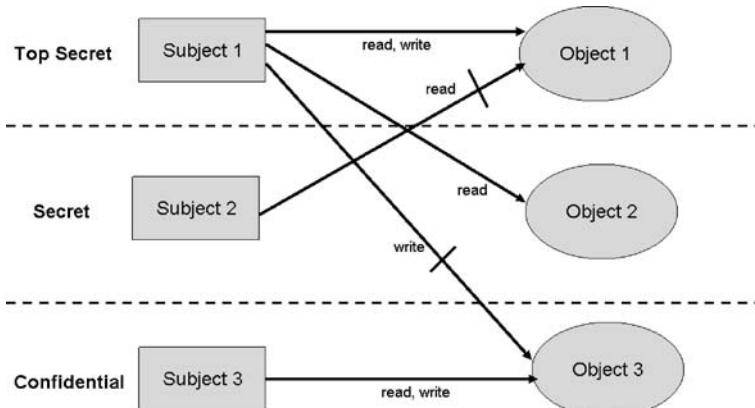


Abb. 4.4 Bell-LaPadula Modell mit Beispiel-Zugriffen für drei Security Level

Kapitel 5

Kryptografische Algorithmen

Dieses Buch soll keine Einführung in die Kryptografie bieten, auch nicht in die Kunst der effizienten Implementierung von Algorithmen. Im Rest des Buches werden wir Krypto-Algorithmen daher mehr oder weniger als "Black Boxes" mit einem wohl definierten Input und Output (oder auch als Module mit Wohldefinierten Schnittstellen, um in der Software-Sprache zu bleiben), mit denen sich komplexere Strukturen wie Protokolle aufbauen lassen und mit denen sich bestimmte Sicherheitsdienste wie etwa der Dienst „Vertraulichkeit“ realisieren lassen.

Der Grund dafür, warum wir keine ausführlichere Einführung in die Kryptografie mit aufgenommen haben, ist einfach: Es gibt bereits eine Fülle von guten und umfassenden Einführungen in das Thema. Zudem kann der Entwickler auf zahlreiche Kryptografie-Bibliotheken zugreifen, die ihm die gewünschten Algorithmen liefern, so etwa OpenSSL oder die im JDK 1.4 enthalten Krypto-Funktionen. Es gibt also im Allgemeinen keinen Grund, einen Algorithmus gemäß seiner Spezifikation selbst zu implementieren, geschweige denn, selbst einen zu entwickeln – vor letzterem sei sogar ausdrücklich gewarnt. Trotzdem wollen wir in diesem Kapitel den Versuch unternehmen, etwas genauer in die „Black Boxes“ hineinzuschauen und zu verstehen, was hinter den Kulissen vor sich geht. Eine gewisse grundlegende Kenntnis kryptografischer Algorithmen ist nämlich unabdingbar, um die Funktionsweise vieler Protokolle etwa zur Authentifikation zu verstehen. Zudem wollen wir versuchen, verständlich zu machen, warum Themen wie Schlüsselmanagement oder Public-Key-Infrastrukturen (PKI) so wichtig für heutige verteilte Anwendungen sind.

5.1 Allgemeines

Mit einem Krypto-Algorithmus allein lassen sich keine Sicherheitsziele erreichen. Zum Algorithmus, dessen Spezifikation nach dem Kerckhoffschen Prin-

zip³ immer öffentlich zugänglich sein sollte, muss ein Geheimnis, der *Schlüssel* hinzukommen. Algorithmus und Schlüssel zusammen werden auf die zu schützenden Daten angewandt und bewirken den angestrebten Sicherheitsdienst, beispielsweise Vertraulichkeit der Daten. Man könnte auch sagen, dass der Schlüssel den Algorithmus parametrisiert. Die folgende Formel drückt dies in prägnanter Form aus. Sie zeigt, wie Chiffretext C und Klartext M bei Verwendung des Schlüssels K und des Verschlüsselungsalgorithmus E voneinander abhängen:

$$C = E_K(M).$$

Die Begriffe „kryptografischer Algorithmus“ und „geheimer Schlüssel“ gehören also zusammen, sind aber dennoch deutlich voneinander zu unterscheiden.

Je nachdem, ob Sender und Empfänger einer Nachricht (in der Krypto-Literatur üblicherweise Alice und Bob genannt) zum Ver- und Entschlüsseln der Daten den gleichen Schlüssel oder unterschiedliche Schlüssel verwenden, spricht man von *symmetrischen* oder *asymmetrischen* Verfahren. Die bereits angesprochenen Passwörter bilden ein gutes Beispiel für einen symmetrischen Schlüssel, der beiden kommunizierenden Parteien bekannt sein muss. Bei symmetrischen Verfahren dient also der gleiche Schlüssel K auch zum Entschlüsseln, es gilt daher:

$$M = E_K^{-1}(C) = D_K(C).$$

Bei asymmetrischen Verfahren dient der Public Key PK zum Verschlüsseln, der Secret Key (oder auch Private Key) SK dient zum Entschlüsseln. Im asymmetrischen Fall gilt also:

$$C = E_{PK_{Bob}}(M), M = D_{SK_{Bob}}(C),$$

wobei Bob der Empfänger der verschlüsselten Nachricht sein soll.

Wir müssen uns in den meisten Fällen nicht entscheiden, ob wir einen symmetrischen oder einen asymmetrischen Algorithmus benutzen wollen, denn die beiden Klassen von Algorithmen ergänzen sich gegenseitig sehr gut: Während symmetrische Algorithmen gut geeignet sind, große Datenmengen schnell und effizient zu verschlüsseln, ist der Einsatz asymmetrischer Verfahren wesentlich aufwändiger. Sie lösen dafür jedoch ein Grundproblem, das mit dem Einsatz symmetrischen Verfahren zwangsläufig einhergeht, nämlich das des *Schlüsseltauschs*, das heißt: Woher bekommen die Kommunikationspartner ihr gemeinsames Geheimnis, und zwar auf eine sichere Art und Weise? Diese Frage ist gerade im Internet-Umfeld, wo wir es mit einer großen Anzahl potentieller Kommunikationspartner, die wir aber niemals persönlich treffen werden (und von denen wir zum Teil noch nicht einmal den Namen kennen), von nicht zu unterschätzender Bedeutung. Verfügt

³ Nach Auguste Kerckhoffs, der dieses Prinzip in seinem 1883 erschienenen Buch *La Cryptographie militaire* als Erster mit aller Deutlichkeit formulierte. Er hatte erkannt, dass sich ein militärisch eingesetzter Algorithmus, der auf dem Schlachtfeld in großer Zahl eingesetzt und durch ein mechanisches Gerät realisiert wird, grundsätzlich nicht geheim halten lässt.

man jedoch über ein asymmetrisches Verfahren, so ist dieses Problem – zumindest im Prinzip – gelöst: Alice veröffentlicht ihren öffentlichen Schlüssel, beispielsweise auf ihrer Homepage oder einem zentralen Webserver. Möchte nun Bob in der Folge mit Alice kommunizieren, muss er sich nur diesen öffentlichen Schlüssel besorgen, etwa indem er sich ihn von einem Server herunterlädt. In der Folge erzeugt Bob einen symmetrischen Schlüssel K (häufig als *Sitzungsschlüssel* bzw. *Session Key* bezeichnet), verschlüsselt ihn mit dem Public Key von Alice und schickt das Resultat an Alice. Somit verfügen beide über ein gemeinsames Geheimnis K und können in der Folge einen symmetrischen Algorithmus für ihre weitere Kommunikation verwenden (siehe auch Abb. 5.2).

5.2 Symmetrische Verschlüsselungsalgorithmen

Nehmen wir also an, die beiden Kommunikationspartner Alice und Bob verfügen über ein gemeinsames Geheimnis K . Nun können sie vertraulich kommunizieren, aber mit welchem (symmetrischen) Algorithmus? Zunächst ist eine grundsätzliche Wahl zu treffen: Sollen die Daten Bit für Bit verschlüsselt werden oder vor dem Verschlüsseln in Blöcken zusammengefasst werden? Mit anderen Worten: Soll eine Strom- oder eine Blockchiffre verwendet werden? Stromchiffren erzeugen aus dem geheimen Schlüssel K einen Strom von pseudozufälligen (das heißt von einem Algorithmus erzeugten, aber zufällig aussehenden) Bits, die dann zur Bitweisen Verschlüsselung des Klartextes verwendet werden. Typischerweise werden Stromchiffren in Umgebungen mit wenig zuverlässigen Transportkanälen wie zum Beispiel einer Funkübertragung eingesetzt, da sich Übertragungsfehler beim Chiffertext dann nur auf ein einziges Klartextbit auswirken. Ein typischer Vertreter dieser Klasse von Chiffren ist der im GSM-Mobilfunkstandard eingesetzte A5-Algorithmus mit seinen Varianten, aber auch der in den SSL Security Settings wählbare RC4-Algorithmus ist eine Stromchiffre. Üblicher ist im Internet-Umfeld jedoch der Einsatz einer Blockchiffre, die den Klartext blockweise, das bedeutet heutzutage in Blöcken mit einer Länge von 64 oder 128 Bit⁴ verschlüsselt.

5.2.1 DES und AES

Haben sich Alice und Bob für eine Blockchiffre entschieden, steht Ihnen eine große Auswahl von veröffentlichten und von der Fachwelt für gut befundenen Algorithmen zur Verfügung. Wir nennen hier nur die beiden bekanntesten von

⁴ Da der Klartext in den meisten Fällen keine „passende“ Bitlänge haben wird, muss er vor dem Verschlüsseln auf eine geeignete Weise aufgefüllt werden (sog. „Padding“). Hierfür existieren verschiedene Standardverfahren, über das sich Alice und Bob ebenfalls im Vorfeld einig sein müssen.

denen, die durch keine Patente geschützt sind, nämlich den schon etwas in die Jahre gekommene *Data Encryption Standard* (DES) sowie seinen modernen Nachfolger *Advanced Encryption Standard* (AES).

Zum DES ist zu sagen, dass er bis heute trotz seines hohen Alters von mittlerweile ca. 30 Jahren keine Attacken zulässt, die wesentlich weniger aufwändig sind als eine vollständige Schlüsselsuche (gäbe es eine solche Angriffsmöglichkeit, müsste man den Algorithmus als „gebrochen“ ansehen). Allerdings sieht sein ursprüngliches Design auch nur einen 56 Bit langen Schlüssel vor, d.h. es gibt nur 2^{56} mögliche Schlüssel. Ein solch kleiner Schlüsselraum kann mittlerweile mit spezieller Hardware oder mit parallel arbeitenden Workstations sehr schnell, d.h. in Tagen oder gar Stunden, durchsucht werden. Um diesem Problem abzuhelpfen, wird der DES heute nur in Form des TripleDES (oder auch 3DES) verwendet, bei dem der DES mit unterschiedlichen Schlüsseln dreimal hintereinander auf den Klartext angewandt wird. Somit erreicht man eine effektive Verdopplung der Schlüssellänge auf 112 Bit, was zumindest für die kommenden ca. fünf Jahre noch als sicher angesehen werden kann. Moderne Blockchiffren wie der AES setzen von Anfang auf eine Schlüssellänge von 128 Bit oder sogar mehr. Aber auch in Sachen Performance ist der AES dem DES überlegen, da er pro verschlüsseltes Klartextbyte viermal weniger Taktzyklen benötigt als der DES in seiner ursprünglichen Form; für den TripleDES kommen wir somit sogar auf einen Faktor Zwölf, um den der AES schneller ist. Aufgrund seines Alters stellt der DES aber so etwas wie einen kleinsten gemeinsamen Nenner für Verschlüsselungsalgorithmen dar. Alice und Bob sollten daher, vor allem wenn sie große Datenmengen zu verschlüsseln haben, nur dann den TripleDES benutzen, wenn sie sich nicht sicher sind, ob alle ihre Kommunikationspartner über eine AES-Implementation verfügen.

5.2.2 Betriebsmodi für Blockchiffren

Beim Einsatz von Blockchiffren stellt sich für Alice und Bob neben dem sicheren Schlüsseltransport ein weiteres Problem: Besteht ihre Nachricht M aus mehr als nur einem Block, gibt es verschiedene Möglichkeiten, mit den Klartextblöcken umzugehen. Diese Möglichkeiten werden als *Betriebsmodi* der Blockchiffre bezeichnet. Die Betriebsmodi können unabhängig von der tatsächlich eingesetzten Blockchiffre definiert werden. Der einfachste dieser Modi, der so genannte *Electronic Codebook Mode* (ECB-Mode) verschlüsselt einfach die Klartextblöcke nacheinander auf immer gleiche Weise, ohne dass es zu einer Interaktion der Blöcke kommt. Dies führt dazu, dass sich eventuell im Klartext befindende Muster aus gleichen Klartextblöcken auch in den Chiffreblöcken wieder finden. Der ECB-Mode sollte deshalb nicht eingesetzt werden, sobald die Länge der zu übermittelnden Nachricht über mehrere Blöcke hinausgeht. Beim Transport kürzerer Nachrichten hingegen spielt der Betriebsmodus keine Rolle und der ECB-Mode kann unbesorgt eingesetzt werden.

Beim vielleicht gebräuchlichsten Betriebsmodus hingegen, dem *Cipher Block Chaining* (CBC) Mode, wird die Musterbildung durch gleiche Klartextblöcke

dadurch verhindert, dass man vor dem Verschlüsseln des i -ten Klartextblocks B_i zunächst das Bitweise XOR mit dem vorhergehenden Chiffreblock C_{i-1} bildet. Im CBC Mode gilt also

$$C_i = E_K(C_{i-1} \oplus B_i), \quad 1 \leq i \leq k,$$

wenn die Nachricht M insgesamt aus k Klartextblöcken besteht. Der Operator \oplus bedeutet hierbei das bitweise XOR der Bits in C_{i-1} und B_i . Ein kleines Problem besteht allerdings noch, wenn CBC als Betriebsmodus gewählt wurde: Woher soll der „nullte“ Chiffreblock C_0 kommen? Auch dieser muss vor Beginn der eigentlichen Kommunikation zwischen Alice und Bob vereinbart werden. C_0 wird auch als *Initialisierungsvektor* (IV) bezeichnet. Er besitzt die gleiche Größe wie ein Klartextblock und kann durchaus im Klartext zwischen Alice und Bob ausgetauscht werden. Allerdings sollte der Algorithmus zur Erzeugung der IVs gewisse Anforderungen erfüllen, insbesondere sollte nicht für jede Nachricht der gleiche IV verwendet werden.

ECB und CBC sind nicht die einzigen Betriebsmodi für Blockchiffren. Andere Betriebsmodi ermöglichen es auch, die Blockchiffre wie eine Stromchiffre zur bitweisen Verschlüsselung zu betreiben. Zu einer tieferen Diskussion der Vor- und Nachteile der Betriebsmodi sowie zu den Anforderungen an den Initialisierungsvektor sei auf die einschlägige Literatur, zum Beispiel [MOV], verwiesen.

5.3 Hashfunktionen

In Kap. 4.1 haben wir gesehen, dass Alice und Bob, sofern sie über ein gemeinsames Geheimnis K verfügen, damit neben der Vertraulichkeit einen weiteren Sicherheitsdienst realisieren können, nämlich die Authentifikation. Authentifikation kann die Verifikation einer Identität oder auch Nachrichtenauthentifikation bedeuten, das heißt: Alice kann einer ihr gesendeten Nachricht M ansehen, ob sie von jemand gesendet wurde, der den geheimen Schlüssel K kennt. Auch wenn Alice und Bob ein Challenge-Response-Verfahren durchführen, um sich gegenseitig als Person zu authentifizieren, verifizieren sie letztlich, ob die Response auf ihre Challenge von einem Schlüsselhaber stammt. Um Nachrichtenauthentifikation zu erreichen, benötigen Alice und Bob aber neben dem geheimen Schlüssel K noch eine kryptografische Hashfunktion H . Definitionsgemäß ist eine *Hashfunktion* eine Funktion, die einen beliebig langen String auf einen String fester Länge abbildet.

5.3.1 Kollisionen

Lesern, die sich mit Datenbanken auskennen, wird der Begriff vielleicht bekannt vorkommen: Lange Datensätze werden „gehasht“ und gemäß ihres Hashwertes in

der Datenbank abgelegt – vergleichbar mit einer Bibliothek, die für jedes ihrer Bücher eine Karteikarte besitzt und diese gemäß der Anfangsbuchstaben der Autoren in einem Karteikasten ablegt. Der Anfangsbuchstabe ist also in diesem Beispiel der „Hashwert“. Gibt es mehrere Bücher mit dem gleichen Anfangsbuchstaben des Autors, haben also mehrere Bücher den gleichen Hashwert, spricht man von einer *Kollision*. Es gibt verschiedene Wege, mit solchen Kollisionen umzugehen und den richtigen Datensatz zu finden, dies soll uns hier aber nicht weiter beschäftigen. Man sollte sich aber klar machen, dass Kollisionen bei Hashfunktionen nichts Ungewöhnliches sind, ja sogar unausweichlich sind, weil eine Hashfunktion ja beliebig lange, und damit auch beliebig viele Strings auf eine begrenzte Zahl von Strings abbildet.

Wir haben oben von einer *kryptografischen* Hashfunktion gesprochen, die Alice und Bob für ihre Nachrichtenauthentifikation benötigen. Ein wesentlicher Unterschied zwischen einer kryptografischen und einer normalen Hashfunktion wie in obigem Beispiel skizziert besteht darin, dass es bei einer kryptografischen Hashfunktion „schwer“, das heißt so viel wie „praktisch unmöglich“ sein muss, Kollisionen zu einem gegebenen Datensatz mit einem gegebenen Hashwert zu finden. Diese Anforderung nennt man auch die *schwache Kollisionsresistenz* der Hashfunktion. Sie erklärt sich unter anderem aus einer Möglichkeit, mit Hilfe einer Hashfunktion H Nachrichtenauthentizität zu erreichen: Angenommen, Alice und Bob verfügen nicht nur über ein gemeinsames Geheimnis K , sondern sie haben sich auch auf einen Verschlüsselungsalgorithmus E und eine Hashfunktion H geeinigt. Sie können nun ihre Nachrichten M authentifizieren, indem sie an M einen so genannten Message Authentication Code (MAC) anhängen, der wie folgt gebildet wird:

$$\text{MAC}(M) = E_K(H(M)) .$$

Der MAC besteht also aus einem verschlüsselten Hashwert. Ändert sich auf dem Transport etwas an M , so sollte sich auch der Hashwert $H(M)$ ändern und somit auch der MAC. Der Empfänger der Nachricht kann den MAC überprüfen, indem er seinen eigenen Hashwert über die erhaltene Nachricht bildet und das Ergebnis mit dem Ergebnis der Entschlüsselung des MAC vergleicht. Nun wird deutlich, warum H schwach kollisionsresistent sein muss: Könnte die Angreiferin Eve bewusst eine Kollision \tilde{M} zu der abgehörten Nachricht M erzeugen (wir gehen in diesem Szenario davon aus, dass M im Klartext übermittelt wird, es Alice und Bob also nur auf die Nachrichtenauthentifikation ankommt), so könnte sie M durch \tilde{M} ersetzen, ohne den geheimen Schlüssel K zu kennen und ohne dass es der Empfänger der Nachricht bemerkt. Aus der schwachen Kollisionsresistenz einer Hashfunktion ergibt sich unmittelbar eine weitere Eigenschaft: Der Hashwert wird von jedem Bit des Input-Strings beeinflusst. Wäre dies nicht so, könnte ein Angreifer durch Ändern eines Bits, das nicht in die Bildung des Hashwertes eingeht, auf einfachste Weise Kollisionen zu einem gegebenen Hashwert erzeugen. Deshalb werden Hashfunktionen auch gern zur Bildung von Fingerprints, also kurzer Bitstrings, die repräsentativ für einen längeren Datensatz stehen, genutzt.

5.3.2 Schlüsselabhängige Hashfunktionen

Außer der Möglichkeit, den Hashwert zu verschlüsseln, gibt es einen weiteren Weg, einen MAC zu erzeugen, der sogar ohne einen Verschlüsselungsalgorithmus E auskommt, indem nämlich Alice und Bob ihren Klartext vom Schlüssel K abhängig machen, bevor sie ihn hashen. Der Standardweg, dies zu tun, heißt HMAC und ist auf folgende Weise definiert (s. [RFC2104]):

$$\text{HMAC}(M) = H(K \oplus a \parallel H(K \oplus b \parallel M)).$$

Hierbei sind a und b feste Strings, der Operator \oplus bedeutet wieder bitweise XOR und der Operator \parallel bedeutet *Konkatenation*, also ein einfaches Aneinanderhängen von Strings. Aus dieser Definition ergibt sich nun eine weitere Anforderung an eine kryptografische Hashfunktion H : Sie sollte die *One-Way-Eigenschaft* haben, was bedeutet, es sollte im obigen Sinne „schwer“ sein, zu einem gegebenen Hashwert $H(x)$ den Input x zu bestimmen. Hätte H nicht die One-Way-Eigenschaft, so könnte Angreiferin Eve aus dem HMAC den Input errechnen, in dem unter anderem der Bestandteil $K \oplus a$ vorkommt. Somit würde Eve auch über den geheimen Schlüssel verfügen und könnte damit Nachrichten fälschen.

Eine weitere Verschärfung der Forderung nach schwacher Kollisionsresistenz besteht in der Forderung nach *starker Kollisionsresistenz*: Während bei der schwachen Kollisionsresistenz ein Paar $(x, H(x))$ vorgegeben ist, zu dem der Angreifer eine Kollision zu finden hat, genügt es für eine Verletzung der starken Kollisionsresistenz, wenn es dem Angreifer gelingt, irgendwelche Kollisionen, also irgendwelche Stringpaare (x, y) mit $H(x)=H(y)$ zu konstruieren. Obwohl solche beliebigen Kollisionen nicht unmittelbar für Angriffe auf MACs verwendet werden können, so werden sie doch von vielen Experten als Vorstufe für ernstere Angriffe angesehen, die auch die schwache Kollisionsresistenz einer Hashfunktion gefährden.

Im Zusammenhang mit der starken Kollisionsresistenz, also der Widerstandsfähigkeit der Hashfunktion gegen das Finden beliebiger Kollisionen durch einen Angreifer, sollte auch das so genannte „Geburtstags-Paradoxon“ nicht unerwähnt bleiben. Dabei handelt es sich um die zunächst einmal überraschende Tatsache, dass nur ca. 21 Personen zusammenkommen müssen, damit mit einer mehr als 50%igen Wahrscheinlichkeit zwei von ihnen am gleichen Tag Geburtstag haben, da man aus 21 Personen immerhin 210 Paare bilden kann. Eine Verallgemeinerung dieser Überlegung für Hashwerte mit n Bit Länge (die insgesamt 2^n Hashwerte spielen jetzt die Rolle der Geburtstage) zeigt, dass man ungefähr $2^{n/2}$ Strings hashen muss, um – unabhängig von der Güte der Hashfunktion – mit mehr als 50-prozentiger Wahrscheinlichkeit eine Kollision zu erzeugen. Die Längen der Hashwerte professioneller Hashfunktionen tragen dieser Tatsache Rechnung: Sie variieren zwischen 128 und 160 Bit. Das bedeutet, ein Angreifer müsste 2^{64} bzw. 2^{80} Strings hashen (und die Hashwerte speichern), um eine mehr als 50%-Chance für eine zufällige Kollision durch das Geburtstagsparadoxon zu haben. Im Moment erscheint dies noch als ein zu großer Speicheraufwand, aber

eine zukünftig zu entwerfende Hashfunktion müsste sicher längere Hashwerte anbieten können.

Welche Hashfunktionen, die die oben skizzierten Anforderungen erfüllen, stehen nun für Alice und Bob zur Auswahl? Leider nicht besonders viele: MD5 und SHA-1 sind die gebräuchlichsten Kandidaten. Außerdem erscheint auch RIPE-MD160 als sichere Hashfunktion, wird aber im Moment von nicht allzu vielen kommerziellen Produkten unterstützt. Diese kleine Auswahl wird durch die Tatsache, dass mittlerweile Kollisionen bei MD5 konstruiert werden können (s. [Dobb]), nicht gerade vergrößert. Während solche Kollisionen zwar die Sicherheit von Produkten und Protokollen, in denen MD5 eingesetzt wird, im Moment nicht unmittelbar gefährden, so sollten sie doch als Warnsignal ernst genommen werden. Als einzige, mittelfristig sichere Wahl erscheint deshalb aus Kompatibilitätsgründen derzeit allein SHA-1⁵.

5.3.3 Password-Based Encryption (PBE)

Häufig werden Passwörter nicht zur Authentifikation eingesetzt, sondern auch als Schlüssel für symmetrische Verschlüsselungsalgorithmen. So dient unter PGP die „Passphrase“ als Schlüssel, um den auf der Festplatte abgelegten Private Key eines Nutzers zu verschlüsseln. In der „Reinform“ sind Passwörter jedoch nicht als Schlüssel geeignet, da sie zumeist nicht aus zufälligen Zeichenfolgen bestehen. Hier liegt ein weiteres Anwendungsfeld für kryptografische Hashfunktionen: Bei der *Password-Based Encryption* werden die Passwörter deshalb vor der Verwendung als Schlüssel gehasht, und zwar zusammen mit einem Zufallswert (Salt), der zusammen mit den verschlüsselten Daten im Klartext abgelegt wird. Wie wir bereits in unserer Diskussion von Passwort-basierten Authentifikationsprotokollen in Kap. 4.1.1 gesehen haben, kann das Salting zwar ein einzelnes Passwort nicht vor einer Wörterbuch-Attacke schützen, ein Wörterbuch-Angriff gegen eine ganze Menge von verschlüsselten Files, die mit Hilfe unterschiedlicher Passwörter verschlüsselt wurden, wird so aber erschwert. Außerdem ist es durch das Salting möglich, aus einem einzigen Passwort mehrere Schlüssel ableiten. So kann sich ein Nutzer beispielsweise bei mehreren Web-Services mit unterschiedlichen Schlüsseln anmelden, die alle aus einem einzigen Passwort erzeugt wurden.

Eine weitere Erschwernis von Wörterbuch-Angriffen wird bei der PBE dadurch erreicht, dass das Passwort bei der Erzeugung des Schlüssels nicht nur einmal, sondern mehrmals gehasht wird. Die notwendige Anzahl von Iterationen wird wie

⁵ Leider mehren sich auch für SHA-1 die Anzeichen, dass auch hier bald mit praktikablem Aufwand Kollisionen gefunden werden könnten (s. [WYY]). Als besonders sicherheitsbewusster Entwickler sollte man deshalb, so lange noch kein wirklich neuer Algorithmus verfügbar ist, auf den SHA-256 mit 256-Bit langem Hashwert zurückgreifen.

das Salt zusammen mit den verschlüsselten Daten abgelegt und kann, wenn sie hinreichend groß gewählt wurde, einen Wörterbuch-Angriff deutlich verlangsamen oder sogar unmöglich machen.

5.4 Asymmetrische Algorithmen

Wir haben bereits davon gesprochen, dass Alice und Bob die Möglichkeit haben, einen symmetrischen Schlüssel (das heißt, einen Schlüssel, der in einem symmetrischen Verfahren eingesetzt werden kann) auf sichere Weise zu vereinbaren, indem sie sich für den Schlüsselaustausch eines asymmetrischen Verfahrens bedienen. Darauf hinaus eignen sich asymmetrische Verfahren auch zur Authentifikation von Alice und Bob, wie in Kap. 4.1 bereits angesprochen.

Grundsätzlich lassen sich asymmetrische Verfahren auf zwei mathematischen Problemen aufbauen: Der Schwierigkeit, eine große Zahl n (das heißt, eine Zahl mit Bitlängen von mehr als 1000 Bit bzw. 300 Dezimalstellen) in ihre beiden Primfaktoren p und q zu zerlegen, also dem so genannten *Faktorisierungsproblem*, und der Schwierigkeit, so genannte *diskrete Logarithmen* zu berechnen. Während das erste Problem für jeden schnell verständlich ist, bedarf es einer kleinen mathematischen Vorrede, um erklären zu können, worum es beim Problem der diskreten Logarithmen überhaupt geht. Aber auch die tieferen Details des auf dem Faktorisierungsproblem beruhenden Verfahrens, wenngleich zunächst etwas leichter zugänglich, erfordern etwas Zeit und Mühe, um sie zu verstehen.

Wir begnügen uns an dieser Stelle mit einer extrem kurzen Einführung in die Mathematik, die hinter diesen beiden Verfahren steckt, und verweisen den interessierten Leser bei tiefer gehendem Interesse auf die Standardliteratur zu diesem Thema, zum Beispiel [Schn96], [MOV] oder auch Kap. 3 von [Bless].

5.4.1 RSA-Verfahren

Das auf dem Faktorisierungsproblem beruhende Verfahren heißt nach seinen drei Erfindern Ron Rivest, Adi Shamir und Len Adleman⁶ das *RSA-Verfahren*. Ohne auf die Details des Verfahrens eingehen zu wollen, die vor dem Software-Entwickler im Normalfall ohnehin durch Kapselung verborgen bleiben, halten wir fest, dass der Secret Key d beim RSA-Verfahren auf dem Wissen um die Primfaktoren p und q einer sehr großen Zahl n beruht⁷. Zusammen mit dem (im Gegensatz

⁶ Die drei Erfinder haben sich ihr Verfahren auch patentieren lassen, was die Verbreitung des RSA – Verfahrens zeitweise etwas behindert hat. Mittlerweile ist dieses Patent aber ausgelaufen.

⁷ Genauer gesagt, berechnet man bei gegebenen (e,n) den Secret Key d mit Hilfe der Gleichung $ed \bmod(p-1)(q-1) = 1$. Dies ist aber nur möglich, falls man die beiden Faktoren p und q kennt.

zu n meist recht kurzen) *Encryption Exponent* e bildet der *Public Module* n im RSA-Verfahren den Public Key eines Teilnehmers. Alle benötigten Rechenoperationen im RSA-Verfahren, also die Ver- und Entschlüsselung von Nachrichten sowie die Erzeugung und Verifikation digitaler Signaturen (siehe unten) spielen sich „*modulo n*“ ab, das heißt, wir rechnen mit sehr großen ganzen Zahlen, wobei aber n eine obere Schranke für alle vorkommenden Zahlen bildet. Genauer gesagt, bedeutet der Ausdruck " $a \text{ mod } n$ ": "Bilde den Rest bei ganzzahliger Division von a durch n ". Eine von Bob mit Alices Public Key (e,n) verschlüsselte Nachricht m nimmt im RSA-Verfahren die folgende Form an:

$$c = m^e \text{ mod } n.$$

Zum Entschlüsseln benötigt Alice ihren Secret Key d . Sie berechnet:

$$m = c^d \text{ mod } n = m^{ed} \text{ mod } n.$$

Der Grund dafür, warum die Potenz $m^{ed} \text{ mod } n$ wieder den Klartext m ergibt, liegt in der speziellen Art und Weise der Erzeugung des Secret Key d aus dem Public Key (e,n) . Details findet der interessierte Leser in der angegebenen Literatur.

5.4.2 Diffie-Hellman-Protokoll und diskrete Logarithmen

Das Diffie-Hellman-Protokoll, mit dem wir uns jetzt beschäftigen wollen, stellt aus historischer Sicht das erste in der frei zugänglichen Literatur veröffentlichte asymmetrische Kryptoeverfahren dar. Es beruht auf dem Problem des *diskreten Logarithmus*. Damit ist folgendes gemeint: Gilt für gewisse ganze Zahlen A , g , x , n die Beziehung

$$A = g^x \text{ mod } n,$$

so heißt x der *diskrete Logarithmus* von A zur Basis g (modulo n). Während die Berechnung der Potenzen $g^x \text{ mod } n$ für verschiedene Werte von x auch bei sehr großen n sehr schnell und effizient vonstatten geht, ist die Umkehrung, also die Berechnung eines diskreten Logarithmus x aus der Kenntnis von A , g und n , ein ungleich schwereres Problem, das von seiner Komplexität her in die gleiche Kategorie wie das Faktorisierungsproblem gehört.

Mit Hilfe diskreter Logarithmen lassen sich zunächst einmal auf sehr direktem Weg Schlüssel austauschen. Das dazu gehörige Protokoll heißt das *Diffie-Hellman-Schlüsselaustauschprotokoll* und ist in Abb. 5.1 dargestellt. Bob schlägt Alice zunächst die Verfahrensparameter g und p vor. Dabei ist p eine große Primzahl und $g < p-1$ die Basis für unsere diskreten Logarithmen. Aus Sicherheitsgründen kommt allerdings nicht jede Zahl $< p-1$ als Basis für die diskreten Logarithmen in

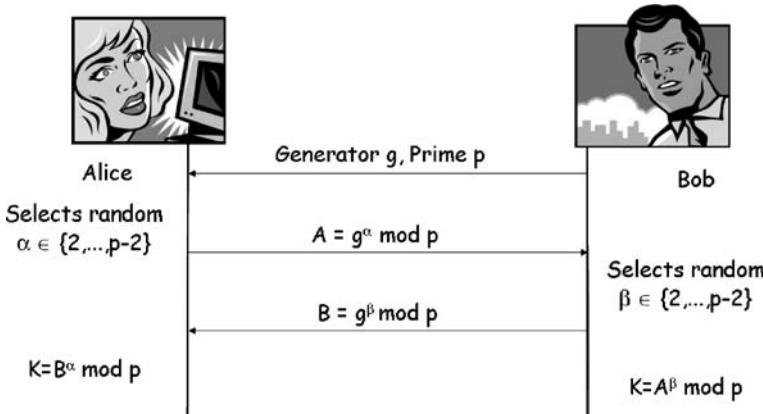


Abb. 5.1 Diffie-Hellman Schlüsseltausch-Protokoll

Frage, sondern nur diejenigen, die die Gruppe $\mathbb{Z}_p^* = \{1, 2, \dots, p-1\}$ erzeugen⁸. Diese Zahlen nennt man *Generatoren* modulo p . Ungefähr jede zweite Zahl $< p$ besitzt diese Eigenschaft. In Kapitel 4.6 von [MOV] findet man einen einfachen Algorithmus zum Finden eines Generators von \mathbb{Z}_p^* .

Bei diesem scheinbar so einfach aussehenden Protokoll sollte man sich kurz vor Augen halten, was hier eigentlich passiert: Alice und Bob unterhalten sich in aller Öffentlichkeit (das heißt, über einen ungeschützten Kanal) und doch haben sie am Ende der Unterhaltung ein gemeinsames Geheimnis $K = g^\beta \text{ mod } p$ vereinbart! Aufgrund der Schwierigkeit des diskreten Logarithmus-Problems ist ein Angreifer (bei hinreichend großem p , also bei einer Bitlänge von mehr als 1024 Bit) nicht in der Lage, aus den öffentlich ausgetauschten Daten auf die Secret Keys (von Alice) bzw. (von Bob) zu schließen. Genauso wenig kann er den Session Key $K = g^\beta \text{ mod } p$ errechnen⁹.

Eine leichte Modifikation dieses Protokolls, bei dem Bob zusammen mit g und p auch gleich den Wert $B = g^\beta \text{ mod } p$ als seinen Public Key veröffentlicht, ermöglicht Alice, sofort nach Herunterladen dieses Public Keys den Session Key K zu berech-

⁸ Das bedeutet: Die Zahlen $g^x \text{ mod } p$ durchlaufen die gesamte Gruppe \mathbb{Z}_p^* , wenn x die Menge \mathbb{Z}_p^* durchläuft. Vielleicht ist dies die richtige Stelle, um auch Kryptosysteme auf der Basis von elliptischen Kurven zumindest kurz zu erwähnen. Diese nutzen ebenfalls diskrete Logarithmen, allerdings werden diese nicht in der vertrauten Gruppe \mathbb{Z}_p^* gebildet, sondern in einer etwas exotischeren Struktur, den *elliptischen Kurven* über endlichen Körpern. Diese haben den Vorteil, dass das diskrete Logarithmus-Problem in ihnen so schwer zu lösen ist, dass man hier mit deutlich kürzeren Schlüsseln (p sollte ungefähr 160 Bit lang sein) auskommt. Die zum Ver- und Entschlüsseln anfallenden Berechnungen sind dadurch weniger aufwändig und machen Kryptosysteme auf der Basis elliptischer Kurven für mobile Geräte und Chipkarten besonders geeignet.

⁹ Allerdings kann der Angreifer versuchen, sich gegenüber Bob als Alice auszugeben und gegenüber Alice als Bob. Gelingt ihm dies, tauschen Alice und Bob nichts ahnend den Schlüssel nicht miteinander, sondern mit dem Angreifer aus (so genannter *Man-in-the-Middle* Angriff). Diesen Angriff können Alice und Bob vermeiden, indem sie ihre Public Keys A, g und p bzw. B, g und p von einer vertrauenswürdigen Instanz zertifizieren lassen (siehe unten).

nen. Somit kann sie eine Nachricht m für Bob mit diesem K verschlüsseln und sie zusammen mit $A = g \bmod p$ an Bob schicken. Diese Modifikation des Diffie-Hellman Protokolls zum Zweck der Verschlüsselung ist als das *ElGamal-Verfahren* bekannt.

5.4.3 Digitale Signaturen

Aufgrund der Tatsache, dass ein Secret Key nur einer einzigen Person bekannt ist, ergibt sich beim Einsatz von asymmetrischen Verfahren eine ganz neue Perspektive: Durch das Verschlüsseln eines Dokuments M mit ihrem Secret Key erzeugt Alice einen Bitstring, den nur sie allein erzeugen kann und der in untrennbarer Weise von dem signierten Dokument abhängig ist. Der mit einem symmetrischen Verfahren gebildete MAC über ein Dokument besitzt zwar vergleichbare Eigenschaften, hat aber den Nachteil, dass er immer von mindestens zwei Personen erzeugt werden kann, nämlich all denen, die den symmetrischen Schlüssel K kennen. Eine solche mit einem MAC gebildete digitale Unterschrift wäre also von Alice abstießbar, da sie auch von Bob hätte erzeugt werden können. Bei Erhalt eines von Alice digital signierten Dokuments M kann Bob nun verifizieren, ob die Signatur tatsächlich von Alice stammt und ob M auf dem Weg zu ihm verändert worden ist, indem er auf die Signatur Alices Public Key anwendet und die Verschlüsselung somit wieder rückgängig macht. Digitale Signaturen eignen sich somit hervorragend zur Authentifikation (die sogar nichtabstießbar ist) und zum Integritätsschutz von Nachrichten. Einen Pferdefuß hat das ganze allerdings: Wie wir noch sehen werden, ist das Handtieren mit Secret Keys sehr aufwändig. Außerdem wäre die mit dem Secret Key verschlüsselte Nachricht ebenso lang wie die Nachricht selbst – beides kann bei sehr langen Nachrichten zum Problem werden. Die Lösung für beide Probleme haben wir bereits kennen gelernt: Wir erstellen vor der eigentlichen Signatur einen repräsentativen Fingerabdruck des zu signierenden Dokuments, das heißt: Alice hasht ihr Dokument, bevor sie es mit ihrem Secret Key verschlüsselt. Zusammengefasst nimmt die Signatur von Alice unter dem Dokument M also folgende Form an¹⁰:

$$\text{sig}_{\text{Alice}}(M) = E_{\text{SK}_{\text{Alice}}}(H(M))$$

wobei H eine der bereits besprochenen Hashfunktionen darstellt. Der weit verbreitete Einsatz von Hashfunktionen im Rahmen digitaler Signaturen zeigt übrigens die Wichtigkeit der Eigenschaft der Kollisionsresistenz bei Hashfunktionen auf: Könnte ein Angreifer zu einem Dokument M ein weiteres Dokument mit dem gleichen Hashwert $H(M)$ konstruieren, so hätten beide die gleiche Signatur und könnten gegeneinander ausgetauscht werden, mit womöglich fatalen Folgen für Alice.

¹⁰ Diese Form gilt nur für das RSA-Verfahren. Bei Nutzung diskreter Logarithmen ist die Signaturbildung komplizierter.

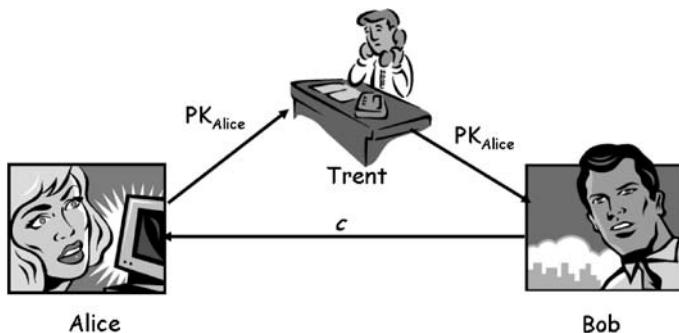
5.4.4 Performance-Fragen

Die Verwendung asymmetrischer Verfahren erfordert wesentlich aufwändigere Rechenoperationen als symmetrische Verfahren, da eine geeignete Langzahlarithmetik zur Durchführung der Rechenoperationen modulo n bzw. modulo p zur Verfügung gestellt werden muss. Dies gilt vor allem, wenn die betreffende Operation mit dem Secret Key arbeiten muss (also eine Entschlüsselung oder die Bildung einer digitalen Signatur beinhaltet), da die Public Keys in gewissen Grenzen frei und somit für die Verarbeitung im Rechner günstig gewählt werden können. Der Secret Key hingegen wird zum Beispiel beim RSA-Verfahren aus dem Public Key mit Hilfe der Faktorisierung des Public Module n errechnet und kann deshalb nicht „günstig“ gewählt werden. Der gemessen an symmetrischen Verschlüsselungsverfahren deutlich höhere Rechenaufwand ist auch durch die Länge der Schlüssel bedingt: Für beide asymmetrischen Systeme, RSA bzw. Diffie-Hellman/ElGamal beträgt die heute zeitgemäße Länge eines Public Key 1024 Bit. Im Vergleich dazu beträgt der typische Wert für symmetrische Schlüssellängen 128 Bit. In RSA entsprechen diese 1024 Bit der Länge des „Public Module“ n , bei Diffie-Hellman der Länge der verwendeten Primzahl p . Ein Angreifer, der beim RSA-Verfahren den Modul n faktorisieren kann, könnte auch den Secret Key d errechnen – der aktuelle Rekord (Stand Mai 2005) für die Faktorisierung großer Zahlen steht bei 200 Dezimalstellen, also etwa 660 Bit. 1024 Bit sollten also noch für die kommenden Jahre eine ausreichende Sicherheitsmarge bieten. Im Ergebnis dauert eine Verschlüsselung mit einem 1024 Bit RSA-Schlüssel ungefähr tausendmal so lange wie eine Verschlüsselung mit dem DES-Algorithmus (mit einem 64-Bit Schlüssel). Vergleichen wir eine RSA-Entschlüsselung mit einer DES-Entschlüsselung, erhalten wir sogar einen Faktor von nahezu zweitausend. Zum Verschlüsseln großer Datenmengen sollten deshalb asymmetrische Verfahren auf keinen Fall eingesetzt werden, sondern ausschließlich zur Schlüsselvereinbarung und Bildung digitaler Signaturen.

5.5 Zertifikate

Wir haben bereits die Möglichkeit angesprochen, mit Hilfe veröffentlichter Public Keys Schlüssel für symmetrische Verfahren auf sichere Art und Weise auszutauschen. Daneben werden auch zur Verifikation digitaler Signaturen Public Keys benötigt. Abbildung 5.2 zeigt das grundsätzliche Szenario, in dem sich Alice und Bob bewegen, um vertraulich zu kommunizieren.

Neben der – bei Einsatz von hinreichend langen Schlüsseln nur theoretischen – Möglichkeit, asymmetrische Verfahren auf mathematische Weise anzugreifen, stellt sich hier aber ein wesentlich ernsteres Sicherheitsproblem: Woher weiß Bob, ob der Public Key, den er sich soeben von Trents Webseite herunter geladen hat, wirklich zu Alice gehört und nicht von einem Angreifer dort platziert worden ist? Gelingt einem Angreifer ein solcher Austausch, so könnte er jede Nachricht, die ursprüng-



- Bildet Schlüsselpaar (PK_{Alice}, SK_{Alice})
- Veröffentlicht PK_{Alice} bei Trent
- Entschlüsselt: $m = D_{SK_{Alice}}(c)$
- Besorgt sich PK_{Alice} bei Trent
- Bildet $c = E_{PK_{Alice}}(m)$
- Schickt c an Alice

Abb. 5.2 Schlüsselvereinbarung bei Einsatz asymmetrischer Verfahren

lich für Alice bestimmt war, entschlüsseln. Dieser Angriff wäre vergleichbar mit dem unbemerkten Austausch eines Namensschildes an einem Briefkasten. Um dies zu verhindern, benötigt Bob eine Form von Garantie, dass ein bestimmter Public Key tatsächlich zu Alice gehört. Eine solche Garantie ist als *Zertifikat* bekannt. Ein Zertifikat ist im Wesentlichen eine von einer vertrauenswürdigen Instanz (einer so genannten *Trusted Third Party*, abgekürzt TTP) digital signierte Bestätigung, dass eine bestimmte Identität und ein bestimmter Public Key zusammen gehören. Diese Bestätigung kann jedoch – je nach Einsatzfeld des Zertifikats neben dem Zusammenhang einer Identität mit einem Public Key auch Attribute (Eigentumsverhältnisse, Augenfarbe etc.) oder Rechte (hat freien Eintritt zum Konzert) bescheinigen.

5.5.1 Zertifikate nach X.509

Die Zertifikate ausstellenden Trusted Third Parties werden auch als CA (Certification Authority) bezeichnet. Je nach Einsatzfeld des ausgestellten Zertifikats kann die CA eine Behörde, eine Firma, deren Geschäftsmodell darin besteht, Zertifikate zu erstellen oder auch eine innerbetriebliche Instanz sein, sofern die in den Zertifikaten enthaltenen Public Keys nur innerhalb einer bestimmten Firma verwendet werden. Um eine möglichst weitreichende Kompatibilität auch über System- und Firmengrenzen hinweg zu gewährleisten, existiert für das Format von Zertifikaten ein Standard, nämlich das X.509 Format der ITU (International Telecommunications Union). Dieser Standard (die aktuelle Version ist V3) gibt unter anderem einen minimalen Satz an Informationen vor, den ein X.509v3 Zertifikat enthalten muss. Dazu gehört die Identität des Antragstellers, die der zertifizierenden Stelle, Gültigkeitsperiode des Zertifikats und natürlich der Public Key des Antragsstellers nebst Informationen über die eingesetzten Algorithmen. Abbildung 5.3 zeigt das

Zertifikat, das den Public Key der Firma TC TrustCenter GmbH, einem bekannten deutschen Anbieter von Zertifikatsdiensten, enthält.

Entscheidend für die Qualität eines Zertifikats ist neben den eingesetzten Algorithmen und der Länge der Schlüssel die Qualität der Identitätsprüfung durch die CA. Damit ist der Aufwand gemeint, den eine CA betreibt, um die Identität eines Antragstellers zu überprüfen. Im einfachsten Fall kann eine solche Identitätsprüfung darin bestehen, zu testen, ob eine vom Antragssteller angegebene E-Mail Adresse tatsächlich existiert und durch den Antragsteller zugreifbar ist. Am anderen Ende des Spektrums steht das persönliche Erscheinen des Antragsstellers bei der CA und das Präsentieren eines Personalausweises. Diese Prozedur bedeutet natürlich für die ausstellende CA einen erheblichen Aufwand an Personal und Infrastruktur und macht solche High-End-Zertifikate entsprechend teuer.

Die Randbedingungen, unter denen das Zertifikat ausgestellt wurde, können mit Hilfe des Zertifikats selbst validiert werden. Wie Abb. 5.3 zeigt, enthält das Datenfeld „Extensions“ einen Unterpunkt „Certificate Authority Policy URL“, wo die URL der allgemeinen Geschäftsbedingungen von TC TrustCenter GmbH zu finden ist (in diesem Fall www.trustcenter.de/guidelines). Über das Datenfeld „Certificate Key Usage“ im Rahmen der Extensions kann auch das Einsatzfeld des zertifizierten Public Keys durch die CA eingeschränkt werden. In Frage kommen

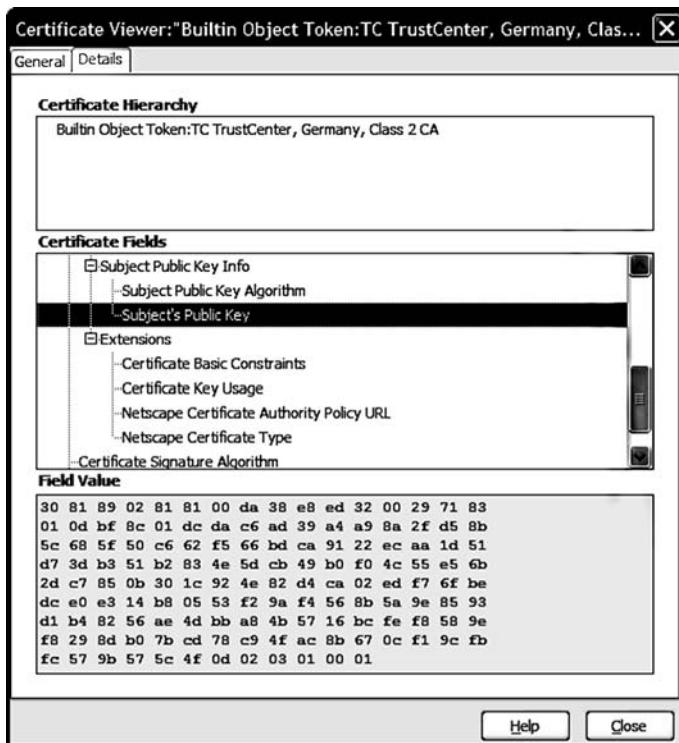


Abb. 5.3 Zertifikat der TC TrustCenter GmbH

etwa die Beschränkung des Schlüssels auf einen reinen Signaturschlüssel oder auf reine E-Mail Verschlüsselung.

Im besten Fall erhält der Antragssteller ein Zertifikat, mit dessen Hilfe er unter gewissen Bedingungen sogar rechtsgültig signieren kann. Die genauen Randbedingungen hierfür sind im deutschen Signaturgesetz und der dazu gehörigen Signaturverordnung¹¹ festgeschrieben und schließen neben sehr strikten Vorgaben an die ausstellende CA auch die Erfordernis an den Zertifikatsinhaber ein, seinen Secret Key „unauslesbar“ auf einer Chipkarte zu speichern. Im übrigen gilt für Zertifikate jeglicher Qualitätsstufe, dass bei der Antragsstellung der Secret Key nicht angegeben werden muss: Ein Antrag auf Zertifikaterteilung in digitaler Form ist nichts anderes als ein vom Antragssteller digital signiertes Dokument mit der vom Antragssteller gewünschten Identität, wie sie im Zertifikat erscheinen soll, und dem zu zertifizierenden Public Key. Das Schlüsselpaar existiert also schon vor der Antragsstellung, wobei der Secret Key nur in Form der Signatur unter dem Antrag in Erscheinung tritt. Jede Art von Speicherung des Secret Key durch die CA würde die Idee der Digitalen Signatur ad absurdum führen.

Natürlich ist es nicht immer notwendig, rechtsgültig zu signieren. Auch der Aufwand, etwa die Mitarbeiter einer Firma alle mit X.509-Zertifikaten auszustatten, ist nicht immer gerechtfertigt. In der Praxis kommen auch häufig einfachere, proprietäre¹² Formate zum Einsatz, sofern die eine Gültigkeit der Zertifikate über die Firmengrenzen hinaus nicht notwendig ist.

Egal, welches Format das Zertifikat besitzt: In der Praxis steht Bob, der eine Signatur von Alice verifizieren möchte, vor der Entscheidung, ob er der ausstellenden TTP und ihren Prozeduren bei der Ausstellung des Zertifikats vertraut. Darüber hinaus muss er entscheiden, ob ihm die im Zertifikat gemachten Angaben zur Identität von Alice genügen – schließlich steht es Alice frei, bei der Zertifikatsantragung ein Pseudonym anzugeben. Es ist somit ein Irrtum zu glauben, durch die Verwendung von Zertifikaten erübrigt sich der Aufbau jeglicher Vertrauensbeziehung zwischen den Kommunikationspartnern im Internet, oder man wüsste immer genau, mit wem man redet. Das Vertrauensproblem hat sich lediglich auf die Etablierung von Vertrauen zwischen den Empfängern von Zertifikaten und den ausstellenden Entitäten verlagert. Entscheidend für die Etablierung von Vertrauen sind der „Ruf“ der ausstellenden Instanz sowie die Prozeduren, die von ihr zur Verifikation der Identität des Zertifikatsinhabers durchgeführt werden.

5.5.2 Attributzertifikate

Die bislang betrachteten Zertifikate sind so genannte *ID-Zertifikate*: Sie bescheinigen die Zugehörigkeit einer Identität zu einem Public Key. Zusammen mit einem

¹¹ Parallel existiert mittlerweile auch eine EU-Rahmenrichtlinie zur rechtsgültigen Verwendung von digitalen Signaturen in der EU.

¹² PGP-Zertifikate stellen ein Beispiel für ein solches proprietäres Format dar.

geeigneten Authentifikationsprotokoll können sie deshalb zur Authentifikation und zur Vereinbarung eines Sitzungsschlüssels zwischen den beteiligten Parteien eingesetzt werden.

Zur Realisierung der Dienste „Authorisierung“ und „Delegation“ kann ein anderer Typ Zertifikat eingesetzt werden, die so genannten *Attributzertifikate* oder *Privilege Attribute Certificates* (PAC). Diese sind von einer *Source of Authority* (SOA) digital signierte Bestätigungen, dass der Inhaber (*Holder*) des Zertifikats gewisse Rechte (*Attributes*) besitzt. Für die Attribute selbst und deren Strukturierung existiert ebenfalls ein Standard: SAML (Security Assertion Markup Language), denn natürlich hängt die Interoperabilität solcher Rechtezusicherungen davon ab, dass alle beteiligten Systeme das gleiche unter den bescheinigten Attributen verstehen und diese auch automatisch verarbeiten können.

Theoretisch können die Rechte eines Zertifikatsinhabers auch im Rahmen seines ID-Zertifikats nach X.509v3 bescheinigt werden, und zwar über eine spezielle Extension mit Namen *subjectDirectoryAttributes*. Dieser Ansatz bringt jedoch eine mangelnde Flexibilität mit sich: Die ausstellende CA muss gleichzeitig auch SOA sein; die Gültigkeit des Zertifikats bezieht sich auf alle bescheinigten Attribute gleichzeitig, das heißt eine Vergabe individueller Gültigkeitsperioden für unterschiedliche Rechte ist nicht möglich. Zudem ist auch eine individuelle Weitergabe der Rechte nicht möglich, sondern es können lediglich alle (oder keine) Rechte an andere Nutzer weiter gegeben werden. Aus diesen Gründen werden meistens dedizierte Attributzertifikate vergeben, die eine höhere Flexibilität besitzen. Auch für Attributzertifikate existiert ein Standard nach X.509.

Durch Attributzertifikate lassen sich nun die bereits beschriebenen Authorisierungsmodelle leicht abbilden, auch das Modell des Role-Based-Access Control (RBAC): Hierbei wird in der speziellen Extension *roleSpecCertIdentifier* auf ein spezielles Rollen-Attributzertifikat verwiesen, in dem die Rechte der jeweiligen Rolle aufgelistet sind.

Möchte ein User seine Rechte an eine bestimmte Instanz delegieren, so kann er selbst zur SOA werden und der Zielinstantz ein Attributzertifikat mit den gewünschten Rechten und der gewünschten Gültigkeitsdauer ausstellen. Somit sind im Prinzip die möglichen Probleme, die bei Delegation von Rechten auftreten können, gelöst: Eine Instanz, der über ein Attributzertifikat gewisse Rechte übertragen werden, erhält diese Rechte nur für einen wohl definierten Zeitraum und handelt nach wie vor unter dem eigenen Namen. Da die Instanz die Credentials des delegierenden Principals nicht kennt, hat sie auch keine Chance, diese zu impersonifizieren.

5.5.3 Zurückziehen von Zertifikaten

Neben dem turnusgemäßen Ablauf eines Zertifikats nach einer gewissen Zeitspanne können Zertifikate auch auf andere Weise ungültig werden. Die Palette der Gründe für ein vorzeitiges Zurückziehen eines Zertifikats ist vielschichtig und

reicht vom Bekannt werden eines Private Key (*Key Compromise*) über das Ausscheiden eines Mitarbeiters aus seiner Firma bis hin zu der Möglichkeit, dass die ausstellende CA ihren Betrieb eingestellt hat. Bei der Nutzung von Attributzertifikaten kann auch ein bloßer Abteilungswechsel eines Mitarbeiters dafür sorgen, dass gewisse Rechte nicht mehr vorhanden sind und das dazu gehörige Zertifikat zurückgezogen werden muss.

Somit muss zum Beispiel bei der Gültigkeitsprüfung einer Signatur neben der rein mathematischen Prüfung und der Prüfung, ob das dazu gehörige Zertifikat für den Verifizierer akzeptabel und noch gültig (also noch nicht abgelaufen) ist, im Prinzip auch geprüft werden, ob das Zertifikat nicht aus anderen Gründen zurückgezogen worden ist. Die Notwendigkeit dieser Prüfung trägt in nicht unerheblichem Maß zum hohen Verwaltungsaufwand bei, den das Management öffentlicher Schlüssel mit X.509-Zertifikaten mit sich bringt. Bislang ist noch keine wirklich befriedigende Lösung für dieses Problem gefunden worden. Wir gehen hier kurz auf die beiden prinzipiellen Lösungsansätze mit ihren Vor- und Nachteilen ein.

5.5.4 Certificate Revocation List (CRL)

Eine CRL ist eine von der ausstellenden CA digital unterschriebene Liste mit zurück gezogenen Zertifikaten, die in regelmäßigen Zeitabständen aktualisiert wird. Ihre Struktur ist im X.509 Standard spezifiziert. Man unterscheidet zwischen *End-entity Public-Key Certificate Revocation Lists* (EPRL), die die Nummern zurück gezogenen Zertifikate von Endbenutzern enthalten, und *Certification Authority Revocation Lists* (CARL), die sich nur auf zurück gezogene CA-Zertifikate beziehen. Der Nachteil einer CRL besteht darin, dass ein verifizierender Client in regelmäßigen Abständen große Datenmengen vom Server der ausstellenden CA laden und die dazu gehörigen Signaturen verifizieren muss. Zudem kann eine CRL nie ganz aktuell sein. Zwischen zwei CRL-Updates besteht also immer eine gewisse Unsicherheit über den aktuellen Status eines Zertifikats. Während das letztere Problem bei der Nutzung von CRLs unausweichlich erscheint, wird das Problem der großen Datenmengen durch die Nutzung der oben genannten, inhaltlich unterschiedlichen CRLs bereits etwas abgemildert. Des Weiteren kommen in der Praxis zur Reduzierung des Suchaufwands auch partitionierte CRLs, die bereits nach gewissen Kriterien aufgeteilt sind, sowie so genannte *Delta-CRLs*, die nur die Einträge enthalten, die sich von einer Aktualisierung zur nächsten geändert haben, zum Einsatz.

5.5.5 Online Certificate Status Protocol (OCSP)

Das Problem der mangelnden Aktualität einer CRL lässt sich lösen, indem ein Protokoll zur Online-Überprüfung des Status eines Zertifikats eingesetzt wird.

Dieses OCSP genannte Protokoll ist in [RFC 2560] spezifiziert. Dabei schickt der OCSP-Client eine Menge von Zertifikat-IDs an den OCSP-Server und erhält als Antwort die Information, ob das Zertifikat widerrufen wurde oder nicht. Der hierfür zu verwendende Server kann in dem Erweiterungsfeld *AuthorityInfoAccess* zusammen mit der Zugriffsmethode (LDAP, http, etc.) angegeben werden. OCSP besitzt als Vorteile die Aktualität der Information und die Vermeidung großer Datenmengen, die zum Client transferiert werden müssen, was vor allem bei der Verwendung mobiler Clients von Bedeutung ist. Allerdings entsteht zusätzlicher Aufwand durch die Notwendigkeit, die Antworten des OCSP-Servers authentisieren zu müssen, das heißt, der Server muss jede seiner Antworten signieren.

5.6 Authentifikationsprotokolle nach X.509

Wir haben bereits in Kap. 4.1 über die verschiedenen Möglichkeiten der Authentifizierung durch symmetrische und asymmetrische Challenge-Response-Protokolle gesprochen. An dieser Stelle wollen wir insbesondere die Möglichkeiten der Authentifizierung durch asymmetrische Protokolle ansprechen.

Wie wir im voran gegangenen Kapitel gesehen haben, sind zur Realisierung von asymmetrischen Challenge-Response-Verfahren zum einen aufwändigere Algorithmen als bei symmetrischen Verfahren erforderlich, was die Rechenlast vor allem auf Seiten des Servers in die Höhe treibt, zum anderen benötigt man eine gewisse Infrastruktur, die *Public Key Infrastructure* (PKI), um mit Hilfe von Zertifikaten die Authentizität der Public Keys zu gewährleisten. Trotz dieses höheren Aufwandes bieten asymmetrische Verfahren zur Authentifikation entscheidende Vorteile gegenüber symmetrischen Verfahren: So entfällt durch die Nutzung asymmetrischer Verfahren die Notwendigkeit für den Server, vorab mit jedem seiner Clients auf einem sicheren Kanal ein gemeinsames Geheimnis zu vereinbaren und diese dann gesichert abzulegen. Zudem besitzen die Varianten, bei denen gewisse Zufallsdaten vom Client signiert werden, den Vorzug, dass eine damit durchgeführte Authentifikation nicht-abstreitbar wird, das heißt, die Beteiligung an der Authentifikation kann im Nachhinein vom Client nicht mehr verneint werden. Die im Folgenden vorgestellten Authentifikationsprotokolle sind im X.509-Standard der ITU spezifiziert [X.509]. Wie wir später sehen werden, greifen diverse Authentication Frameworks auf diese Protokolle als Mechanismus zur Authentifikation zurück.

5.6.1 One-Pass Authentication

Die One-Pass Authentication Protokolle nach X.509 sind vielleicht die interessantesten der hier vorgestellten Authentifikationsprotokolle, da hier die Authentifikation des Client ohne eine initiale Challenge des Servers erzielt werden kann. Somit

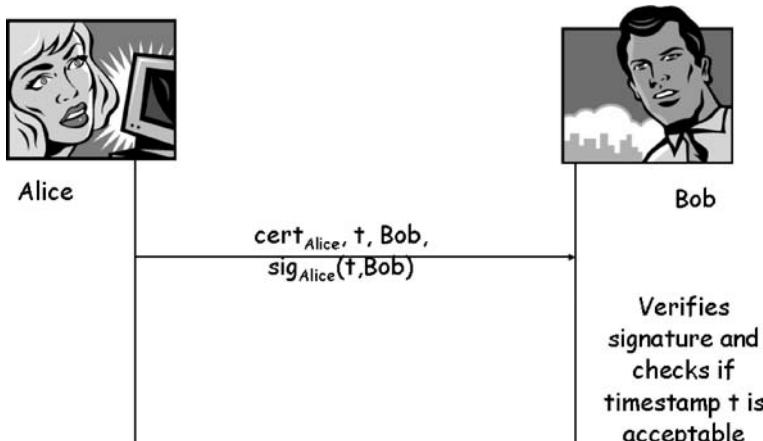


Abb. 5.4 Einseitige Authentifikation durch Alice mittels einer Nachricht

beruht die Authentifikation auf einer einzigen Nachricht vom Client zum Server (One-Pass). Voraussetzung hierfür ist allerdings, dass Client und Server beide über eine synchronisierte Uhr verfügen, die von beiden Seiten akzeptierte *Timestamps* (Zeitstempel) erstellen kann.

Abbildung 5.4 zeigt den vereinfachten Ablauf des Protokolls: Alice erzeugt einen Zeitstempel t und signiert ihn mit ihrem Private Key, zusammen mit einer Identität des Servers, bei dem sie sich authentisieren will. Da niemand außer Alice, der Besitzerin des Private Key einen solchen Datensatz erstellen kann, dient er zusammen mit dem Zertifikat von Alice als hinreichendes Authentisierungstoken für den Server, sofern der Zeitstempel t und das Zertifikat von Alice für ihn akzeptabel sind. Dadurch, dass Alice den Zeitstempel signiert und somit ein Token generiert, das Bob nicht hätte erzeugen können, wird die Authentifikation sogar nicht-abstreitbar für Alice.

Fängt allerdings ein Angreifer das Authentisierungstoken ab, so kann er Alice so lange impersonifizieren, wie der Zeitstempel gültig ist (*Replay Attack*). In der Praxis sollten deshalb nur Zeitstempel mit sehr kurzen Aktualisierungsintervallen verwendet werden. Wir werden dem Problem der Replay Attacks in Verbindung mit der Gültigkeit von Zeitstempeln später bei der Diskussion von Kerberos, einem symmetrischen Authentifikationsprotokoll auf Basis einer Trusted Third Party und von Zeitstempeln, wieder begegnen.

5.6.2 Three-Pass Authentication

Haben Alice und Bob keinen vertrauenswürdigen Zeitstempeldienst zur Verfügung, können sie auf ein klassisches Challenge-Response-Verfahren auf der Basis asymmetrischer Verfahren zurückgreifen. Zu diesem Zweck schickt Alice zu-

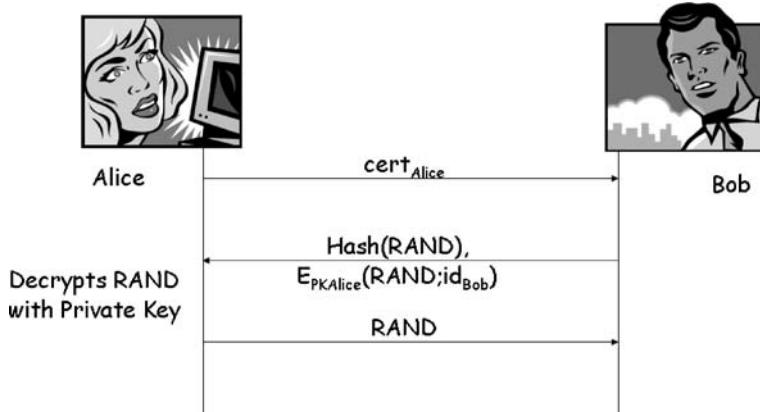


Abb. 5.5 Einseitige Authentifikation von Alice mittels drei Nachrichten

nächst an Bob ihr Zertifikat. Um zu verifizieren, dass er tatsächlich mit Alice, also der Inhaberin des im Zertifikat angegebenen Public Keys, verbunden ist, verschlüsselt Bob eine Zufallszahl RAND mit dem Public Key in Alices Zertifikat. Kann Alice diese entschlüsseln, so verfügt sie tatsächlich über den Private Key und hat sich somit authentifiziert.

Was aber, wenn es Bob gar nicht darauf ankommt, Alice zu authentifizieren, sondern sie in Wirklichkeit dazu verleiten will, für sie bestimmte verschlüsselte Daten, die Bob abgefangen hat, zu entschlüsseln und an Bob zurück zu schicken? Angesichts des häufigen Einsatzes von hybriden Verschlüsselungssystemen, bei denen ja kurze symmetrische Session Keys mit Public Keys verschlüsselt und versendet werden, ist dies kein unwahrscheinliches Szenario. Da Session Keys im Allgemeinen wie zufällige Daten aussehen, ist nicht zu erwarten, dass Alice einen solchen Angriffsversuch durch Bob erkennen würde. Aus diesem Grund muss Bob seiner Challenge den Hashwert der erwarteten Response beifügen. Somit weist er nach, dass er die richtige Antwort tatsächlich kennt und Alice nicht nur zum Entschlüsseln missbrauchen will. Dieser Nachweis wird auch als *Plaintext Awareness* bezeichnet.

5.6.3 Three Pass Mutual Authentication

Möchten sich Alice und Bob gegenseitig authentifizieren (*mutual authentication*), so können sie dies tun, indem sie sich gegenseitig Challenges schicken und diese signieren. Wie Abb. 5.6 zeigt, reichen dafür drei Nachrichten aus. Da hier wieder digitale Signaturen im Spiel sind, besitzt auch dieses Protokoll die Eigenschaft der Nicht-Abstreitbarkeit, diesmal für Alice und Bob. In der Praxis wird dieses Protokoll selten eingesetzt, da beide Kommunikationspartner dafür Zertifikate benötigen.

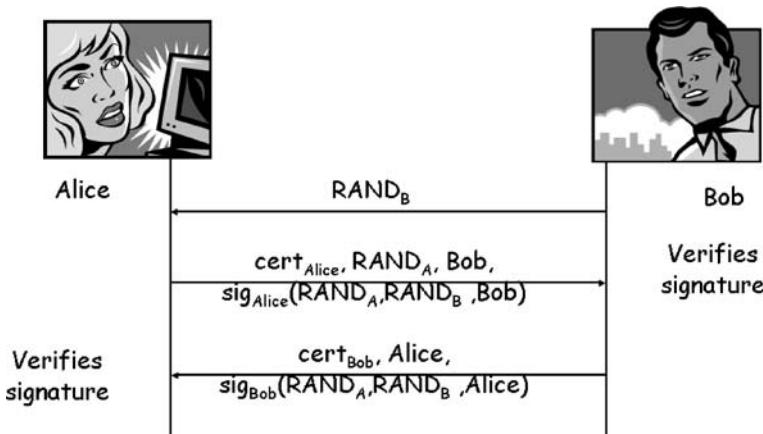


Abb. 5.6 Gegenseitige Authentifikation von Alice und Bob mit drei Nachrichten

gen. Zudem tritt auf beiden Seiten der erhöhte Rechenaufwand für die Erzeugung der digitalen Signatur auf.

5.7 Zufallswerte

Die Rolle von Zufallswerten beim Design sicherer IT-Systeme wird häufig unterschätzt. Dabei kommen sowohl echte Zufallswerte als auch pseudozufällige Werte mit unterschiedlichen Eigenschaften an vielen, häufig entscheidenden Stellen in der IT-Security zum Einsatz, auf die wir jetzt etwas genauer eingehen.

5.7.1 Schlüsselerzeugung

Sowohl symmetrische als auch asymmetrische Schlüssel müssen letztlich durch einen zufälligen, nicht vorhersagbaren Prozess erzeugt werden. Beim RSA-Verfahren startet die Schlüsselerzeugung mit der Auswahl zweier großer zufälliger Primzahlen mit der gewünschten Bitlänge, bei symmetrischen Verfahren benötigt man in den meisten Fällen einen zufälligen 128-Bit-String. In anderen Fällen nutzt man auch pseudozufällige, also durch einen algorithmischen Prozess (einen so genannten *Pseudozufallsgenerator*, abgekürzt PRNG) erzeugte Werte, aber auch diese Pseudozufalls-Algorithmen benötigen einen komplett zufälligen Anfangswert (*Seed*) zum Start. So wird im Zuge des SSL-Handshakes zunächst ein (echt) zufälliges „PreMasterSecret“ erzeugt und zwischen Client und Server ausgetauscht. Von diesem PreMasterSecret werden dann in einem pseudozufälligen Prozess durch sukzessives Hashen weitere Sitzungsschlüssel abgeleitet. Damit ist

klar, dass an die Nicht-Vorhersagbarkeit des PreMasterSecrets höchste Anforderungen gestellt werden müssen, da auf ihm letztlich die gesamte Sicherheit von SSL beruht. In der Vergangenheit sind diese hohen Ansprüche nicht von jeder SSL-Implementation erfüllt worden. Die Tageszeit, die ProzessID und die ID des Elternprozesses bilden beispielsweise keine gute Datengrundlage für die zufällige Erzeugung von Schlüsseln, da diese Daten in gewissen Grenzen vorhersagbar sind und auch gewissen Mustern gehorchen.

Bei der Erzeugung von Schlüsseln ist deshalb sowohl auf eine hohe Qualität der Seed als zufälligem Input als auch auf die Qualität der Pseudozufalls-Algorithmen zu achten, um die Sicherheit eines Systems nicht gleich „am Start“ zu gefährden (siehe zu dieser Thematik zum Beispiel auch den [RFC 1750]).

5.7.2 Challenges

Auch die Challenges, also die „Fragen“, die im Zuge eines Challenge-Response-Protokolls mit dem Ziel der Authentifikation gestellt werden, sollten aus zufälligen Werten bestehen. Allerdings sind die Anforderungen hier nicht ganz so streng wie im Fall der kryptografischen Schlüssel. Es reicht im Wesentlichen, sicherzustellen, dass sich die Challenges nicht wiederholen. Trotzdem sollte auch auf eine Nicht-Vorhersagbarkeit im weitesten Sinne geachtet werden, da ansonsten ein Angreifer versuchen könnte, im Rahmen einer Brute-Force-Attacke sämtliche möglichen Responses auf eine vorausberechnete Challenge zu bestimmen.

5.7.3 SessionIDs

Wie wir in der Diskussion von HTTP als verbindungslosem Protokoll gesehen haben, benötigt man Zufallszahlen als Identifikatoren, um eine bestehende Session wieder aufzunehmen. Insbesondere kann sich ein bereits authentifizierter Client durch das Vorweisen der richtigen SessionID ausweisen. Daraus ergibt sich, dass SessionIDs, auch wenn sie über einen sicheren Kanal wie SSL übermittelt worden sind, sich auf keinen Fall wiederholen oder für einen Angreifer vorhersagbar sein dürfen.

5.8 Kryptografie mit Java

Aufgrund seiner weitgehenden Plattformunabhängigkeit ist Java besonders für verteilte Anwendungen, insbesondere auch Internet-Anwendungen geeignet. Aus diesem Grund war Sicherheit seit der Einführung von Java ein wichtiges Thema für in Java geschriebene Anwendungen. Wir werden später noch die Sicherheits-

aspekte von Java auf Sprachebene genauer beleuchten – hier soll zunächst kurz beschrieben werden, wie sich die besprochenen Krypto-Algorithmen in Java realisieren lassen, wie also der Entwickler Kryptografie in Java-Anwendungen einbauen kann, ohne die besprochenen Algorithmen selbst implementieren zu müssen. Die Darstellung hier soll nur einen kleinen Einblick in die Möglichkeiten vermitteln, in Java Kryptografie zu realisieren. Eine weitaus ausführlichere Darstellung dieser Thematik finden interessierte Entwickler in [Hook] oder [Knud].

5.8.1 Java Cryptography Architecture (JCA)

Die JCA stellt eine Abstraktionsschicht zur Verfügung, über die der Entwickler auf die Implementierung der gewünschten Algorithmen zugreifen kann. Diese Implementierungen werden über ein Service Provider Interface (SPI) von unterschiedlichen Service Providern bereitgestellt (s. Abb. 5.7). Lädt man sich das JDK (1.4 oder später) von SUNs Webserver herunter, ist hierin bereits ein solcher Provider enthalten. Andere Provider, die in Form eines signierten JAR-Files vorliegen müssen, können durch entsprechende Modifikation des `java.security` Files installiert werden. Hier wird auch die Reihenfolge festgelegt, in der auf die Provider zugegriffen wird, wenn eine Instanz eines bestimmten Algorithmus aufgerufen wird. Es ist jedoch auch möglich, neben dem Namen eines Algorithmus beim Aufruf aus einer Applikation heraus auch den gewünschten Provider zu nennen, so dass die ursprüngliche Präferenzordnung nicht berücksichtigt wird. Sinn dieser Architektur ist es zum Einen, die Details der Implementierung vor dem Entwickler zu verbergen, zum anderen wird auf diese Weise dem Entwickler die Möglichkeit gegeben, einfach zwischen den unterschiedlichen Implementierungen der verschiedenen Provider hin und her zu wechseln.

Aus Gründen, die der US-amerikanischen Exportpolitik in Bezug auf Krypto-Algorithmen geschuldet sind, ist der JCA die Java Cryptography Extension (JCE) an die Seite gestellt. Die JCA beinhaltet alle die Algorithmen, mit denen sich der Dienst „Authentifikation“ (und nur dieser) realisieren lässt, wie zum Beispiel Hashfunktionen oder digitale Signaturen auf Basis diskreter Logarithmen. Auf der anderen Seite enthält die JCE innerhalb der Klasse `Cipher` Implementationen der wichtigsten symmetrischen und asymmetrischen Krypto-Algorithmen zur Realisierung von Vertraulichkeit. Dadurch konnten in der Vergangenheit die beiden Dienste voneinander getrennt werden und die entsprechenden Algorithmen getrennt voneinander exportiert werden, je nachdem in welchem Land sich der Kunde befand. Mittlerweile haben sich die US-Exportrestriktionen jedoch gelockert, und einem Export von JCA gemeinsam mit der JCE im Rahmen des JDK 1.4 in EU-Staaten steht nichts im Wege. Je nachdem, ob der gewünschte Algorithmus in der JCA oder in der JCE enthalten ist, müssen vom Entwickler zu Beginn die Pakete `java.security` bzw. `javax.crypto` importiert werden.

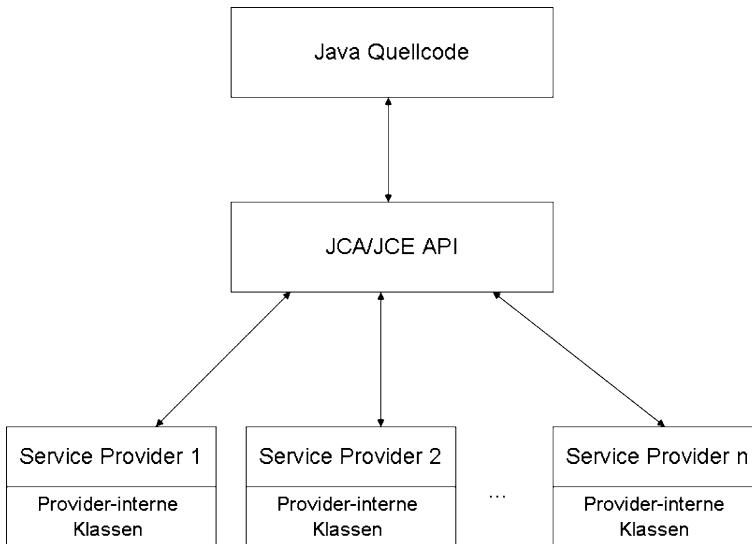


Abb. 5.7 Grundaufbau der Java Cryptographic Architecture

5.8.2 Symmetrische Verschlüsselung

Eine Instanz der gewünschten Verschlüsselungsalgorithmen erhält man über die Methode `getInstance()` der Klasse `javax.crypto.Cipher`. Dieser Methode können neben dem Namen des Algorithmus weitere Argumente wie der gewünschte Provider, Betriebsmodus, Padding-Schema usw. übergeben werden. Nach diesem Schema läuft auch die Erzeugung von Instanzen der anderen Typen von Krypto-Algorithmen ab. Das nötige Schlüsselmaterial wird entweder zufällig erzeugt (mit Hilfe der Klasse `javax.crypto.KeyGenerator`, hierbei ist darauf zu achten, dass man den `KeyGenerator` mit der richtigen Schlüssellänge initialisiert), aus in Form eines Byte-Array vorliegendem Schlüsselmaterial erzeugt (mit Hilfe der Klasse `javax.crypto.spec.SecretKeySpec`). Auch die Erzeugung eines Schlüssels aus einem Passwort ist möglich (Password-Based Encryption, die hierfür zuständige Klasse heißt `PBEKeySpec`).

Hat man eine Instanz `cipher` der gewünschten Chiffre und den dazu gehörigen Schlüssel, kann der Algorithmus über die `cipher.init()`-Methode mit dem Schlüssel initialisiert werden. Außerdem wird die Information, ob ent- oder verschlüsselt werden soll, an die `init()`-Methode übergeben. Der letzte Schritt besteht nun in der schrittweisen Verarbeitung des Klartexts mit Hilfe der Instanzmethoden `update()` bzw. `doFinal()`, die den Klartext in Form eines Byte-Array erwarten und den Chiffertext auch in dieser Form ausgeben. Die `doFinal()` Methode dient dabei dazu, ein evtl. nach dem Aufruf von

`update()` verbliebenes letztes Stück Klartext zu verschlüsseln und das Ergebnis an das Ende des Output-Array zu schreiben.

5.8.3 Hashfunktionen und MACs

Implementationen der gebräuchlichsten Hashfunktionen SHA-1 (und seiner Varianten SHA-256 und SHA-512) finden sich in der Klasse `java.security.MessageDigest` (man beachte, dass diese Algorithmen ausschließlich für Integritätsschutz und Nachrichtenauthentifikation genutzt werden können und deshalb nicht in die JCE ausgelagert sind). Die Nutzung erfolgt analog zu dem im letzten Abschnitt beschriebenen Verfahren: Zunächst erzeugt man sich eine Instanz des Algorithmus über die `getInstance()` Methode. Die Verarbeitung der Klartextdaten (die als Byte-Array vorliegen sollten) erfolgt mit den Instanzmethoden `update()` und `digest()`.

Da die zur MAC-Bildung verwendeten Algorithmen teilweise – zumindest theoretisch – auch zur Verschlüsselung eingesetzt werden können, ist die entsprechende Klasse `javax.crypto.Mac` in der JCE zu finden. Neben dem HMAC stehen in dieser Klasse auch MACs zur Verfügung, die mit Hilfe der symmetrischen Chiffren DES und AES arbeiten, das heißt, der MAC besteht in diesem Fall aus einem verschlüsselten Hashwert. Natürlich müssen für MACs auch wieder eigene Schlüssel generiert werden; dies geschieht mit den im letzten Abschnitt erläuterten Verfahren.

5.8.4 Asymmetrische Kryptografie

Was die reine Verschlüsselung betrifft, unterscheidet sich asymmetrische Kryptografie in Java nicht wesentlich von der symmetrischen: Auch hier müssen zunächst die gewünschten Instanzen der Klasse `javax.crypto.Cipher` erzeugt werden. Als Algorithmen stehen RSA und ElGamal zur Auswahl. (Für den Schlüsselaustausch über das Diffie-Hellman Protokoll steht eine eigene Klasse `KeyAgreement` zur Verfügung.) Wesentliche Unterschiede bestehen natürlich bei der Erzeugung der Schlüssel. Schlüsselpaare mit der gewünschten Länge können sowohl auf Basis zufällig gewählter Primzahlen mit Hilfe der Klasse `java.security.KeyPairGenerator` als auch mit vom Entwickler vorgegebenen Parametern erzeugt werden. Für die auf diskreten Logarithmen beruhenden Algorithmen Diffie-Hellman und ElGamal unterstützt die JCA auch Logarithmen, die über elliptischen Kurven gebildet werden.

Für digitale Signaturen ist die Klasse `java.security.Signature` zuständig. Auch hier müssen nach dem bekannten Muster zunächst Instanzen der gewünschten Algorithmen erzeugt werden. Danach wird die Instanz über die Methode `initSign()` initialisiert, das heißt, ein privater Signaturschlüssel wird an

sie übergeben. Die eigentliche Signaturerzeugung geschieht analog zu dem Verfahren bei symmetrischen Chiffren über eine `update()`-Methode, welche die zu signierenden Daten in das Signature-Objekt hineinzieht, und eine `sign()`-Methode, die den Prozess abschließt. Output ist auch bei Signaturen wieder ein Byte-Array. Sollen keine Signaturen erzeugt, sondern verifiziert werden, muss das Signature-Objekt im Verification Mode über `initVerify()` (mit dem dazu gehörigen Public Key als Argument) initialisiert werden. Danach kann die Signatur in Form eines Byte-Array an die `verify()`-Methode übergeben werden.

5.8.5 Zertifikate

Die Basisklasse zum Umgang mit Zertifikaten in Java lautet `java.security.cert.Certificate`. Sie stellt die wichtigsten Methoden für die Arbeit mit Zertifikaten bereit, so etwa `getPublicKey()` zum Auslesen des Public Key aus dem Zertifikat oder `verify()` zur Verifikation der Signatur unter dem Zertifikat. Die JCA (bzw. SUNs Krypto-Provider) unterstützt die Erzeugung eigener selbst-signierter Zertifikate nicht direkt, es gibt jedoch andere Provider, die dies tun (vgl. [Hook], Kap. 6).

Für den wichtigen Spezialfall der X.509-Zertifikate steht mit `java.security.cert.X509Certificate` eine eigene Klasse zur Verfügung. Diese unterstützt auch die gebräuchlichsten Zertifikatserweiterungen (*Extensions*), die ab der Version 3 Teil des X.509-Standards sind. Für die Prüfung, ob ein X.509-Zertifikat zurückgezogen ist, steht in Form von `java.security.cert.X509CRL` ebenfalls eine eigene Klasse zur Verfügung. OCSP hingegen wird von SUN nicht unterstützt, auch hier gibt es aber andere Provider, die diese Lücke füllen.

5.8.6 Erzeugung von Zufallszahlen

Das Java Development Kit stellt im Rahmen der Klassen `javax.crypto.KeyGenerator` (für symmetrische Schlüssel), `java.security.KeyPairGenerator` (für asymmetrische Schlüsselpaare) und `java.security.SecureRandom` Methoden für die sichere Erzeugung von Schlüsselmaterial und Zufallszahlen zur Verfügung¹³. Bei der Schlüsselerzeugung können sowohl der Algorithmus, für den der Schlüssel benötigt wird, als auch seine Länge und eine Quelle für die Seed spezifiziert werden. Wird keine solche Quelle

¹³ In der Klasse `java.util.Random` finden sich ebenfalls Methoden zur Erzeugung gleich- oder normalverteilter Zufallszahlen; diese arbeiten jedoch mit einem so genannten linearen Kongruenz-Generator (siehe zum Beispiel [Knut], Kap. 3.2.1) und sind für kryptografische Zwecke nicht geeignet.

angegeben, liefert der mit der SHA-1 Hashfunktion arbeitende Konstruktor `SecureRandom()` eine hinreichend sichere Seed für die Schlüsselerzeugung.

5.9 Übersicht

Zum Abschluss dieses Kapitels noch eine Übersicht in Baumform, die den besprochenen Kryptomodulen die Sicherheitsdienste, die sie realisieren, zuordnet (vgl. auch Fig. 1.1 in [MOV]). Im Folgenden werden wir auf die inneren Details der Module nicht mehr weiter eingehen und sie je nach benötigtem Sicherheitsdienst verwenden.

Man erkennt unter Anderem die große Bedeutung der digitalen Signatur bei der Umsetzung verschiedenster Sicherheitsziele. Die folgenden Kapitel werden zeigen, wie wichtig es ist, Nachrichten oder allgemeiner Objekte unmittelbar mit einer digitalen Signatur zu schützen (objektbasierte Sicherheit), anstatt sich auf einen sicheren Transportmechanismus zu verlassen (kanalbasierte Sicherheit).

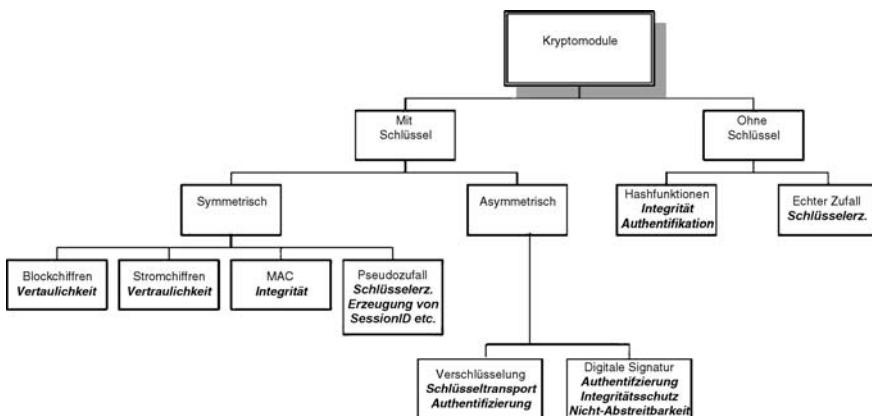


Abb. 5.8 Taxonomie der Kryptomodule

Kapitel 6

Sicherheit in Verteilten Systemen

bahn.de oder ebay.com sind ohne Frage hochgradig verteilte Systeme, wenn sie sich ihren Kunden im Internet auch als einziges „monolithisches“ System präsentieren. Verfolgt man beispielsweise einige Requests an bahn.de, sieht man, dass eine Reihe externer Systeme zum Tracking von Benutzern angesprochen werden. Auch ein Blick auf das Online-Ticket verriet ursprünglich das Vorhandensein verschiedenster Backend-Systeme.

Noch ausgeprägter ist die Verteilung auf ebay.de. Dort kommen verschiedenste Kunden in Kontakt, Inhalte und teilweise auch Code werden zwischen unterschiedlichen Maschinen ausgetauscht – wenn auch vermittelt durch die Zentrale des ebay-Portals.

Bei diesen verteilten Systemen stellt sich die Frage, wie die Subsysteme untereinander abgesichert sind. Wie werden beispielsweise Anfragen von Kunden von System zu System weitergeleitet, ohne dass sie – zum Beispiel von Innentätern – missbraucht werden können? Wie kann überhaupt so etwas wie eine „Distributed Trusted Computing Base“ realisiert werden?

Hinter jeder Kommunikation in verteilten Systemen steht das Problem des Vertrauens (Trust): Wer traut wem aus welchen Gründen? Vertrauen kann auf Autorität beruhen – aber welche Autoritäten gibt es in verteilten Systemen und wie funktionieren sie?

Dieses Kapitel hat als Aufgabe, einige zentrale Eigenschaften verteilter Systeme im Zusammenhang mit der eingesetzten Sicherheitstechnologie zu beleuchten. Damit wollen wir die Basis für das Verständnis komplexer Infrastrukturen in den folgenden Kapiteln legen. Speziell suchen wir Antworten zu den folgenden Problemstellungen:

- Welche Authentisierungsweisen eignen sich besonders für verteilte Systeme?
- Welche sicherheitsrelevanten Informationen (Identität, Authentizität) werden zwischen Subsystemen ausgetauscht?
- Wie fließen sicherheitsrelevante Informationen zwischen Subsystemen?
- Wie lässt sich ein Aufruf über verschiedene Systeme hinweg verfolgen?
- Wie kann eine einmal erfolgte Authentisierung weitergegeben werden? Was bedeutet das für die Vertrauensbeziehungen zwischen Systemen?

- Was bedeuten die Begriffe Delegation und Impersonation in verteilten Systemen?
- Wie hängen Identitäten und Rechte zusammen? Welche Identitäten sind an einem verteilten Request beteiligt?
- Was konstituiert Domänen und wie arbeiten sie zusammen?
- Wie funktioniert Föderation von Systemen?
- Wie werden die Identitäten und Credentials verschiedener Systeme aufeinander abgebildet?
- Wie werden Credentials in verteilten Systemen gesichert?

Noch stellen die meisten Firmenportale relative Inseln dar – ausgedrückt durch eine eigene Autorität zur Identifikation von Nutzern. Noch sind die Infrastrukturen der Firmen weit davon entfernt, eine interne Ende-zu-Ende Sicherheit von den Eingangsrechnern über mittlere Systeme mit Geschäftslogik bis hin zu den Backendsystemen auf Mainframes zu bieten. Zu unterschiedlich sind die vorhandenen Datenstrukturen und Schnittstellen im Bereich Sicherheit.

Dennoch zeichnet sich bereits ein neuer Trend ab: Webservices und Grid-Architekturen ermöglichen die Nutzung firmenübergreifender Dienste. Das Internet-Business nutzt Services der verschiedensten Partner. Und nicht zuletzt erscheinen im Peer-To-Peer Bereich total verteilte Applikationen, die sich nicht nur auf effizientes Filesharing beschränken.

Eine zentrale Autorität fehlt allen diesen Architekturen, und so ist das Problem der effizienten Föderation von Nutzern und Services ein sehr bedeutsames für die Zukunft.

6.1 Lokale versus verteilte Sicherheit

Was unterscheidet überhaupt die Sicherheitsanforderungen in einem verteilten System von denen in einem lokalen System, bei dem alle beteiligten Anwendungen sich auf demselben Rechner befinden? Zunächst einmal fällt im verteilten System die Notwendigkeit einer Kommunikation zwischen den Instanzen ins Auge. Dies impliziert als Sicherheitsanforderung die Absicherung der zur Kommunikation benutzten Kanäle (*kanalbasierte* Sicherheit) bzw. der ausgetauschten Nachrichten (*nachrichtenbasierte* Sicherheit). Hinzu kommt die Notwendigkeit der Authentifizierung der miteinander kommunizierenden Instanzen.

In beiden Fällen müssen sich Nutzer des Systems authentifizieren, jedoch gibt es in einem lokalen System logischerweise nur eine einzige zentrale Instanz, die authentisiert und auch autorisiert. Insbesondere besteht im lokalen Szenario keinerlei Notwendigkeit, das Ergebnis der Authentifikation und Authorisierung an andere Instanzen weiterzugeben. In verteilten Systemen stellt gerade diese Notwendigkeit eines der zentralen Sicherheitsproblem dar: Wie können Authentisierungen und damit verbundene Rechte auf sichere Weise von einer Instanz an die nächste weiter gegeben werden? Oder soll sich ein Nutzer für jede Instanz, auf die

er zugreifen möchte, separat authentifizieren? Eng damit verbunden ist die Frage, welche Instanzen in einem verteilten System überhaupt vertrauenswürdig sind: Wie kann Vertrauen in einem verteilten System aufgebaut werden? Und wie soll eine empfangende Instanz mit dem Ergebnis einer Authentifizierung umgehen? Führt sie daraufhin ihre eigene Authorisierung durch oder vertraut sie den Rechten, die ihr durch die sendende Instanz mitgeteilt werden? Schließlich stellt sich noch die Frage, welche Identität bei einer eigenen Authorisierung zu Grunde gelegt werden soll: Die des Nutzers, der die ursprüngliche Anfrage gestellt hat, oder die der weitergebenden Instanz?

Diese kurze Diskussion zeigt, dass die Sicherheit in einem verteilten System sehr viel mehr Aspekte beinhaltet als die in einem lokalen System. Als erstes werden wir die Frage der Authentisierung in einem verteilten System genauer untersuchen.

6.2 Authentisierung in verteilten Systemen

6.2.1 Authentisierung versus Identifizierung

Ganz selbstverständlich taucht die Authentisierung eines Clients häufig als erstes Softwareproblem bei der Entwicklung einer verteilten Applikation auf. Meist wird sie als unabdingbare Vorstufe zur Authorisierung gesehen, das heißt, dem Zugehen von Rechten an authentisierte Nutzer.

Bevor wir uns dem „wie, wann und wo“ dieses Problems zuwenden, möchten wir diesen Automatismus kurz hinterfragen. Dazu ist es nötig zu untersuchen, was bei der Authentisierung eines Requests eigentlich passiert. Betrachten wir zunächst den klassischen Fall, wie er auch im Portal von bahn.de auftritt:

- Die Firma klassifiziert ihre Ressourcen (Informationen, Applikationen) nach Sicherheitskriterien (öffentlicher Zugriff, Zugriff für authentisierte Kunden, Zugriff nur für Mitarbeiter etc.). Im Anschluss daran wird die Infrastruktur so konfiguriert, dass die geschützten Ressourcen nur über eine durchgeführte Authentisierung erreichbar sind. Softwaretechnisch taucht schnell das Problem auf, dass ein Kontrollfluss von öffentlichen Funktionen in geschützte verhindert werden muss.
- Die Identität, die einen Request an eine Firmeninfrastruktur stellt, wird geprüft. Diese Identität kann eine „echte“ sein, das heißt, eine reale Person (Mensch, Firma) repräsentieren. Die Identität kann aber auch eine Maschine darstellen. Diese Form der Authentisierung macht Sinn, wenn eine in der Realität gültige Identität für die Durchführung des Requests nötig ist, sei es zur Abrechnung (Bahnticket) oder auch zur Bestrafung bei Missbrauch.
- Schnell stellt man fest, dass hinter der Frage der Identität des Initiators des Requests eine Reihe großer Fragezeichen stecken: Erhalten beispielsweise registrierte Clients eine neue Identität, sobald sie Kunden werden? Welche Iden-

tät nutzen Mitarbeiter, die gleichzeitig Kunden sind? Akzeptiert die Firma Identitäten, die von anderen Institutionen vergeben wurden? In der Praxis wird das Problem der Identität häufig einfach durch Anlegen eines Verzeichnisses gelöst. Geschieht das für jede Applikation gesondert, so entsteht in kürzester Zeit ein völliges Chaos in Bezug auf Identitäten und Authentisierungen. Heute gibt es Infrastrukturen, die unter dem Schlagwort *Identity Management* dieses Chaos eindämmen wollen. Speziell Portale benötigen solche Infrastrukturen, die wir weiter unten vorstellen werden.

- Im Zuge der Prüfung einer vorgegebenen Identität legt die Infrastruktur der Firma ein softwaretechnisches Stellvertreterobjekt für den Client an. Dieser wird genannt und stellt einen authentisierten Client dar. Je nach Güte der Infrastruktur gilt dieser Principal nur auf der Maschine, die die Prüfung durchgeführt hat, oder kann im Verlaufe der Bearbeitung an andere Maschinen und Applikationen weitergegeben werden. Das Stellvertreterobjekt dient der Festlegung, mit welchen Rechten ein Request ausgeführt werden soll, und außerdem der Aufzeichnung aller Aktionen, die für diesen Request durchgeführt wurden (Auditing). Gleichzeitig wird meist eine so genannte *Session* erstellt, damit der Authentisierungsprozeß nicht beim nächsten Request des Clients sofort wieder durchgeführt werden muss. Das Ergebnis der Authentisierung muss also in irgendeiner Form festgehalten werden. (Je nach Industriestandard oder Framework wird statt Identität (Name) und Principal (erfolgreich authentisierte Identität) auch das Paar Principal (Name) und Subject (erfolgreich authentisierter Principal) verwendet. Wichtig ist nur die Unterscheidung zwischen Namen und erfolgreicher Authentisierung und im Anschluss daran die Frage in welcher Form die erfolgreiche Authentisierung festgehalten wird bzw. wie lange).

Etwas anders gelagert ist der Fall, wenn die Feststellung einer Identität zur Bearbeitung eines Requests streng genommen gar nicht nötig ist: Stellen wir uns die Prüfung eines elektronischen Tickets im Zug vor. Hier geht es lediglich darum festzustellen, ob das Ticket gültig ist. Derzeit stellt die Schaffnerin jedoch auch die Identität des Fahrgasts anhand seiner Bahncard als Teil der Ticketprüfung fest. Dies wäre im Prinzip jedoch erst nötig, wenn sich das Ticket als ungültig herausstellen würde, da dann aus Gründen der Bestrafung eine Authentisierung durchgeführt werden müsste. Allerdings bindet das Geschäftsmodell der Bahn an dieser Stelle das Ticket an den Besitzer der Bahncard um zu verhindern dass günstigere Tickets für Nicht-Bahncard Besitzer ausgestellt werden können. Natürlich bevorzugen Firmen grundsätzlich die Feststellung der Identität zur Authentisierung von Kunden, da sich über das Konzept der Identität hervorragend Daten über das Konsumverhalten der Kunden sammeln lassen und so Profile der Kunden entstehen. Es sei jedoch angemerkt, dass das Interesse der Firmen nicht immer unbedingt im Einklang mit dem Interesse der Kunden steht und aus technischer Sicht nicht immer eine Authentisierung nötig ist.

Bei näherem Hinsehen stellt sich schnell heraus, dass auch aus wirtschaftlichen Gründen die Authentisierung von Berechtigungen statt von Personen ein sehr attraktives Modell sein kann. Man denke beispielsweise an die Zulieferfirmen für

große Konzerne, die bei identitätsbasierter Authentifikation gezwungen sind, die Mitarbeiter der Konzerne in ihren Datenbanken zu führen und jede Änderung im Personalbestand nachzuvollziehen. In beiden Fällen jedoch, der Authentisierung zur Feststellung einer Identität oder der Authentisierung von Berechtigung durch eine Gültigkeitsprüfung, muss auf Empfängerseite ein Ergebnis dieser Prüfung festgehalten werden. Die Art und Weise, wie dies geschieht ist durch den jeweiligen Authentisierungsmechanismus festgelegt und wird weiter unten sowie im Kapitel zu föderativer Sicherheit diskutiert.

6.2.2 Authentisierung mit Passwörtern

Nichts scheint einfacher, als Passwörter zum Zwecke der Authentisierung von Sendern einer Nachricht einzusetzen. Kunden, aber auch die Rechner einer Webapplikation sowie „künstliche“ User (so genannte *Functional User*) besitzen Passwörter. Mitarbeiter melden sich mit Passwörtern am firmeninternen Intranet an. Häufig verschaffen sich sogar Administratoren von Webapplikationen per Passwort Zugang zu ihrer Applikation, um diese aus dem Internet heraus zu bearbeiten.

Aus sicherheitstechnischer Sicht und von ihrem Verwaltungsaufwand her müssten Passwörter jedoch längst der Vergangenheit angehören, wie wir noch sehen werden. Entwickler dürften eigentlich gar nicht mehr an ihre Verwendung denken. Die Realität sieht jedoch anders aus: Bei Entwicklern sind Passwörter geschätzt, weil sie das am leichtesten verständliche Mittel der Authentisierung sind. Schnell ist eine Passwortabfrage implementiert. Notfalls lässt sich auch ein eigenes Verzeichnis (eine so genannte *Registry*) erstellen, um einen kleinen Stamm von Benutzern zu verwalten.

Damit entsteht aber ungewollt schnell die Situation, dass Administrationsrechte von Fremden auf einfachste Weise erlangt werden können, beispielsweise aufgrund der Tatsache, dass nicht mehr existierende oder nicht mehr benötigte User in diversen Applikationen und deren Registries eingetragen bleiben.

Es lohnt sich also, die Gründe für die Probleme mit Passwörtern in verteilten Systemen einmal systematisch anzugehen.

6.2.2.1 Generelle Eigenschaften von Passwörtern

Wir haben die Authentifikation mittels Passwörter bereits im Kapitel „Sicherheitsdienste“ kurz angesprochen. Hier noch einmal zusammengefasst die wichtigsten Eigenschaften von Passwörtern:

- Passwörter sind Geheimnisse zwischen zwei Parteien, zwischen denen sie auf einem sicheren Weg etabliert werden müssen.
- Passwörter müssen durch Menschen merkbar sein.

- Passwörter müssen auf sicherem Weg übertragen werden
- Passwörter dürfen nicht durch Dritte erratbar sein.
- Werden Passwörter gespeichert, muss dies auf sichere Weise geschehen.

Der Punkt, dass Passwörter auf sicherem Weg zwischen zwei Partner etabliert werden müssen, spielt eine wesentliche Rolle im e-Business-Bereich, wie schon am Beispiel ebay diskutiert: Eine Firma bietet Services am Internet an. Ein Kunde möchte diese nutzen und registriert sich zu diesem Zweck über ein Web-basiertes Formular. Die Firma muss den Kunden jetzt über einen externen Mechanismus, zum Beispiel den Postweg authentisieren. Das bedeutet einen erheblichen Mehraufwand für die Firma und außerdem einen gewissen Zeitraum, in dem die Identität des Kunden und des Senders nicht erklärt ist.

Dieses Problem versuchen Webservices zu lösen, indem sie ein Sicherheitsmodell verwenden, das auf vertrauenswürdigen Dritten (*Trusted Third Parties* oder auch *Security Token Issuer*) aufbaut. Diese fungieren als zentrale Authentisierungsautorität für mehrere Services zugleich. Somit wird vermieden, dass sich ein Kunde selbst beim gewünschten Zielservice authentisieren muss und der Aufwand für den Serviceanbieter wird geringer (siehe Kapitel Web Services Security).

Das Kriterium der Merkbarkeit von Passwörtern stellt uns ebenfalls vor große Probleme: Generell lässt sich sagen, dass ein merkbares Passwort entweder relativ kurz ist oder aber eine Bedeutung trägt und somit mit hoher Wahrscheinlichkeit in einem Wörterbuch zu finden ist. In beiden Fällen kann das Passwort durch einen Angreifer erraten werden – im ersten Fall durch eine Brute-Force-Attacke, die alle möglichen Passwörter unterhalb einer bestimmten Länge durchprobiert, im zweiten Fall durch eine Dictionary-Attacke, bei der die in einem Wörterbuch enthaltenen Wörter als Passwörter durchprobiert werden.

Beide Angriffe funktionieren dann am besten, wenn der Angreifer das gesuchte Passwort in einer gehaschten oder auch leicht verschlüsselten Form kennt und dadurch die Vergleiche mit den geratenen Passwörtern offline in großer Geschwindigkeit durchführen kann.

Eine Weiterentwicklung von Passwörtern ist die so genannte Passphrase, wie sie zum Beispiel im populären Verschlüsselungsprogramm PGP zum Einsatz kommt: Ein längerer Ausdruck, der aus Wörtern und Zahlen gebildet wird, der zwar nicht mehr so leicht angreifbar ist, aber auf jeden Fall auch merkbar sein muss. Dies bedeutet im Normalfall einen Ausdruck, der entweder semantisch sinnvoll ist oder aber aufgeschrieben werden muss.

Darüber hinaus neigen wir dazu, Passwörter wieder zu verwenden, obwohl ja zur Definition eines Passwertes gehört, dass es ein Geheimnis zwischen genau zwei Parteien darstellen sollte. Sobald wir ein Passwort mehrfach verwenden, gefährden wir nicht nur unsere eigene Sicherheit, sondern auch die unserer Partner: Ein weiterer Service, bei dem ich mich mit dem gleichen Passwort anmelden, könnte sich beim ersten Service unter meinem Namen (und mit meinen Rechten) einloggen.

6.2.2.2 Sicherheitstechnische Bewertung

Wenn Sie einen Internet-basierten Dienst anbieten und dabei die Authentisierung via Passwort erlauben wollen, müssen Sie folgende Punkte klären:

- Sollen UserID und Passwort vom Service vorgegeben werden?
- Soll ein Pseudonym des Users nach außen hin erscheinen?
- Soll das Passwort gewechselt werden können?
- Wie verhalten Sie sich im Falle eines vergessenen Passworts?
- Auf welche Services erlauben Sie den Zugriff über Passwörter?
- Wie werden die Passwörter sicher übertragen?
- Wie werden die Passwörter sicher gespeichert, und zwar auf Server- und Clientseite?
- Wer erhält Zugriff auf die Passwörter?
- Wie sichern Sie sich vor mehrfach verwendeten Passwörtern?
- Wie sichern Sie sich gegen Brute-Force und Dictionary Attacken?
- Geben sie Usern Tipps zum Umgang mit Passwörtern?
- Sind Passwörter der einzige Zugang zu Ihren Services?

Das sind eine Menge Fragen für einen scheinbar so einfachen Mechanismus. Schon die erste Frage zeigt einen grundsätzlichen Gegensatz zwischen Benutzbarkeit (Usability) und Sicherheit auf: Kunden hassen es, wenn man ihnen eine neue Identität sowie ein halbwegs sicheres (das heißt schlecht zu merkendes) Passwort aufzwingt. Der Kunde ist gezwungen, beides aufzuschreiben bzw. auf seinem Desktop zu speichern.

Was sind die Konsequenzen? Für seltene Anmeldungen, die von immer dem gleichen Platz aus durchgeführt werden, mag das Verfahren tragbar sein. Sobald sich der Kunde jedoch häufig anmelden muss bzw. dies von wechselnden Orten aus tun muss, besteht die Gefahr, dass die aufgeschriebenen UserID/Passwort-Kombination mitgeführt wird, mit der entsprechenden Gefahr bei Verlust.

Überlässt man hingegen dem Kunden die Wahl von UserID und Passwort, besteht natürlich die Gefahr, dass einfachste Passwörter gewählt werden, die von Angreifern leicht erraten werden können. Zur Verdeutlichung der Problematik noch ein kleines Gedankenexperiment:

Angenommen, Sie bieten einen Service auf dem Web an, wo Sie gegen eine kostenlose Registrierung irgendeine Leistung versprechen. Mit den UserIDs und Passwörtern, die Ihre User bei Ihnen verwenden, versuchen Sie, Zugang zu bekannten Webdiensten zu bekommen. Das Ergebnis wird vermutlich erschreckend sein. Im Endergebnis bedeutet dies, dass der Versuch der Vermeidung von Brute-Force und Dictionary Attacken bei Passwörtern genau entgegengesetzt zur Benutzerfreundlichkeit verläuft.

Die Frage nach der Sichtbarkeit des Pseudonyms ist ebenfalls nicht einfach zu beantworten. Nehmen Sie einen Service wie ebay, bei dem die Nutzer untereinander mit der UserID bekannt sind. Diese UserID ist zwar ein Pseudonym, das frei gewählt werden kann, gleichzeitig stellt sie aber auch die LoginID dar, mit der die Benutzer ihren ebay-Account verwalten. Manche Systeme trennen diese beiden

Identitäten, um zu verhindern, dass durch Erraten von Passwörtern ein Angreifer die Identität eines legitimen Nutzers übernehmen kann (und zum Beispiel falsche Angebote in dessen Namen abgibt). Zum Schutz gegen das Erraten von Passwörtern ist ein automatischer Abschaltmechanismus denkbar, der zum Beispiel nach drei fehlgeschlagenen Login-Versuchen zu einer Sperrung des Accounts führt. Werden jedoch LoginID und sichtbares Pseudonym nicht voneinander getrennt, so stellt ein solcher Mechanismus gleichzeitig auch eine Einladung zu Denial-of-Service-Angriffen dar.

Wenn Ihr Service an öffentlichen Plätzen, zum Beispiel einem Internet-Café oder auch im Intranet genutzt werden kann, besteht die Gefahr, dass Passwörter abgeschaut werden. Sie können versuchen, dem Missbrauch durch einen regelmäßigen erzwungenen Wechsel der Passwörter zu begegnen. Sehr häufig stellt jedoch der erzwungene Wechsel des Passworts ein großes Problem sowohl für die User wie auch die Infrastruktur dar (Was passiert in einer komplexen Application Server Infrastruktur wenn während eines Passwort-Wechsels ein Absturz von Server, Verzeichnisdienst oder Datenbanken geschieht?). Oft erfolgt nach einem Zwangswechsel des Passworts nach kurzer Zeit der Fall des vergessenen Passworts, denn meist sind die letzten vom User verwendeten Passwörter gesperrt. Das bedeutet, dass die Nutzer sich – während sie eine Aufgabe in der Firma erledigen sollen – parallel ein komplett neues Passwort von möglichst hoher Qualität ausdenken und merken müssen, mit der Konsequenz, dass es schnell vergessen wird.

In diesem Fall müssen Sie sich für einen zweiten Mechanismus (Fallback-Mechanismus) entscheiden, der benutzt wird, wenn das eigentliche Passwort nicht mehr verfügbar ist. Soll der Mechanismus auf vorher abgefragten „Geheimnissen“ des Users basieren wie zum Beispiel der berühmt-berüchtigte „Mädchenname der Mutter“? Wie sicher kann ein solches Verfahren überhaupt sein? Oder soll per Post ein neues Geheimnis zugestellt werden, das dann vom User sofort durch ein eigenes neues Passwort ersetzt werden muss? Schließlich muss auch die Frage geklärt werden, wie der Ersatzmechanismus in Kraft treten soll: Reicht dafür ein einfacher Anruf beim Helpdesk aus oder müssen weiter gehende Bedingungen durch den User erfüllt werden?

Die Frage, wie Passwörter sicher übertragen werden können, scheint zu Zeiten von SSL/TLS eindeutig beantwortet zu sein. Aber erinnern wir uns daran, dass SSL nur einen sicheren Tunnel zwischen zwei Beteiligten zur Verfügung stellt. Es bleiben die Grundprobleme einer jeden kanalbasierten Verbindung:

- Zwischen wem besteht die Verbindung?
- Wie abhörsicher/vertraulich ist die Verbindung?
- Was passiert mit den Daten am Ende der Verbindung?

Serverseitig lässt sich das Problem, wer der Endpunkt des Tunnels ist, zwar durch ein Zertifikat lösen, jedoch muss die Vorstellung des Kunden vom Servicenamen nicht mit dem übereinstimmen, was im Server-Zertifikat steht. Auf Client-Seite ist völlig offen, wer Endpunkt des Kanals ist, da Ihre Kunden im Normalfall nicht über ein Zertifikat verfügen werden. Auch die Abhörsicherheit der Verbindung lässt sich zwar durch geeignete Wahl der Ciphersuites sicher stellen, dies

geschieht aber nicht automatisch, sondern muss durch eine geeignete Server-Konfiguration erzwungen werden. Zudem haben Sie keinerlei Kontrolle darüber, wie ihr Kunde mit dem Passwort verfährt, wenn es erst einmal auf seinem Rechner angekommen ist. Aber auch der Kunde muss sich darauf verlassen, dass der Server sorgfältig mit seinen Credentials umgeht – eine Gewähr hierfür gibt es nicht.

Dies bringt uns zur noch schwierigeren Frage nach der Speicherung von Passwörtern: Wo und wie sollen sie gespeichert werden und wo am besten gar nicht? Nicht gespeichert werden sollten sie beispielsweise beim Benutzer, da der heimische PC ein sehr verwundbares Speichermedium darstellt.

Leider bietet eine ganze Reihe von Werkzeugen wie Browser eine Speicherfunktion von Credentials, bei der diese wiederum mit einem einzigen Passwort geschützt sind. Dies ist zwar einerseits erfreulich, da es dem User erlaubt, eine ganze Reihe von Passwörtern zu verwalten (also eine Mehrfachnutzung zu vermeiden). Auch können dadurch wesentlich schwieriger merkbare Passwörter verwendet werden (keine Simplifikation). Der Nachteil liegt jedoch in der Qualität der Speicherung. Zur Speicherung durch den Browser wird nämlich ein so genanntes SSM – ein *Software Security Module* – verwendet. Es liegt ganz in der Hand der Softwareentwickler wie sicher ein solches SSM ist – da das SSM mit keinem externen System interagieren muss, braucht es auch keinen Standards zu genügen.

Aber auch in der eigenen Serviceinfrastruktur sollten keine Passwörter beispielsweise in den Logfiles diverser Server auftauchen, wo sie bereit liegen zum Mitlesen durch die Mitarbeiter in Betrieb oder Entwicklung. Damit nicht genug: Werden Passwörter über verschiedene SSL/TLS Verbindungen weitergereicht, so entstehen an jedem Tunnelendpunkt ungeschützte Momente, wo die Passwörter im Klartext vorliegen. Hier stellt sich die Frage, ob Passwörter im RAM von Servern längere Zeit verbleiben oder gleich nach der Verifizierung gelöscht werden.

Letztlich spielt auch der Ort, an dem serverseitig die Passwörter gespeichert werden, eine wichtige Rolle. Wie werden die Passwörter abgelegt? Wer erhält Zugriff darauf? Wie werden unberechtigte Zugriffe von Seiten von Mitarbeitern verhindert? Im engen Zusammenhang mit der Frage der Speicherung steht die Frage, wie und wo ein eingegebenes Passwort gegen ein gespeichertes verifiziert wird. Geschieht die Verifikation beim Application Server oder innerhalb eines Repositories? Werden hierbei Authentisierungsmethoden verwendet, die die Kenntnis des unverschlüsselten Passwortes auf einem vorgezogenen Server voraussetzen? Dies ist zum Beispiel bei Digest Verfahren nach Art der http Digest Authentication der Fall. Während diese Verfahren auf dem Transportweg vom Client zum Server Vorteile bieten, bedeutet die Notwendigkeit, dass vorgezogene Server die Passwörter im Klartext kennen müssen, gegenüber der Variante, bei der das gespeicherte Passwort die Registry nicht verlässt (und dort auch nur in verschlüsselter Form gehalten wird) auf Serverseite einen deutlichen Sicherheitsnachteil.

Schließlich müssen Sie die Frage klären, welche Services Sie über Passwörter absichern wollen. Dahinter steckt letztlich das Problem der Bewertung der Qualität einer Authentisierung durch Passwörter.

Nach dieser Diskussion, die mehr Fragen aufgeworfen als Antworten gegeben hat, lässt sich zusammenfassend das Folgende über Passwörter sagen:

- Passwörter sind eine „schwache“ Authentisierungsmethode.
- Sie sind nicht geeignet für jede Art von Administrationsservices.
- Sie sind nicht geeignet für Services, bei denen größere Werte im Spiel sind.
- Sie sind nicht geeignet für Services, auf denen der Dienstleister eine Form von Nicht-Abstreitbarkeit benötigt, wie es der Fall ist, wenn zum Beispiel teure Spezialanfertigungen für Kunden erstellt werden.

6.2.2.3 Verwaltung von Passwörtern

In diesem Abschnitt wollen wir etwas genauer auf die Schwierigkeiten eingehen, die vergessene Passwörter und ein automatischer Wechsel von Passwörtern (so genanntes *Password Aging*) für die Server-Infrastruktur mit sich bringen können.

Vergessene Passwörter stellen gerade bei qualitativ höheren Passwörtern ein großes Problem dar. Die meisten Firmen sind gezwungen, teure Helpdesks einzurichten, bei denen ihre Nutzer ein neues Passwort beantragen können. Für Firmen, die ausschließlich im Internet tätig sind, bleibt wie oben angesprochen oft nur ein automatischer Mechanismus, basierend auf vorher mit dem User abgesprochenen Daten. Als letzte Option kann der User eine neue Identität beantragen. In beiden Fällen handelt es sich um sehr teure Organisationsformen, deren Notwendigkeit noch größer wird, wenn die Security Policy der Firma einen regelmäßigen Wechsel der Passwörter mit Verbot der Wiederbenutzung alter Passwörter vorschreibt.

Aber nicht nur für die User bedeutet der erzwungene Passwortwechsel eine unangenehme Situation. Wesentlich weiter reichende Auswirkungen besitzt der Wechsel des Passworts für hochkomplexe Infrastrukturen wie zum Beispiel verteilte Webapplikationen mit ihren Datenbanken, Registries, Application Servern, Web Servern, Proxies, Caches etc.

In diesem Umfeld werden sehr viele verschiedene Identitäten benutzt, darunter auch so genannte *Functional User*, also Identitäten, unter denen eine Applikation eine andere aufruft. Die dazu gehörigen UserID und Passwörter finden sich häufig in den Konfigurationsfiles von Web Applikationen, Servern etc. Laufen auch diese Passwörter ab – zum Beispiel weil sie in einer Registry mit automatischem Password Aging gespeichert sind – dann kann eine Reihe von Problemen auftreten. So kann eine falsche Reihenfolge bei der Passwortänderung zum Stillstand des ganzen Systems führen. Manche Komponenten bemerken eine Änderung der Credentials „hinter ihrem Rücken“ mit der Folge, dass sie nicht mehr booten. Mit einem „Warnsystem“, das den Wechsel bzw. Ablauf des Passworts ankündigt, lassen sich einige dieser Probleme lösen, allerdings können die Probleme auch noch komplexer werden, wenn mehrere Registries hintereinander geschaltet sind, beispielsweise zuerst ein LDAP (Lightweight Directory Access Protocol) und anschließend ein RACF (Resource Access Control Facility) auf dem Mainframe.

Kann nun das User Interface zum LDAP auf eventuelle Hinweise zum Password Aging reagieren, die von RACF kommen?

Eine passwortbasierte Authentisierung ist also teurer und aufwändiger als man denkt – jedenfalls sobald man versucht, gewisse Sicherheitsregeln zu beachten.

6.2.3 Software-Architektur der Authentisierung mit Passwörtern

Zwei Fragen sind für das Verständnis der SW-Architektur der Authentisierung in verteilten Systemen wesentlich:

- Wer nimmt die Authentisierungsdaten (also die Credentials) entgegen?
- Wo werden die Credentials gehalten – in einer lokalen Registry oder einer zentralen, gemeinsam genutzten?

Als Antwort auf die erste Frage lassen sich Betriebssysteme und Applikationen unterscheiden. Betriebssysteme besitzen häufig bereits fortgeschrittene Verfahren der Authentisierung, während sich Applikationen oft noch wie Inseln in der Infrastruktur verhalten, so dass wir häufig auf das Szenario von Abb. 6.1 treffen.

Hier führt jede Applikation ihre Authentisierung selbst durch und speichert die Credentials darüber hinaus in einer eigenen Registry ab.

Aus Sicht der Softwareentwickler ist dies ein harmloses Szenario, da es für eine maximale Entkopplung der Entwicklungsteams in den Unternehmen sorgt. Jede Gruppe kann ihre eigene Userverwaltung aufbauen. Selbstverständlich finden auch die Zugriffskontrolle (Access Control) sowie die Verwaltung der Rechte (Authorisierung) der User innerhalb der jeweiligen Applikation statt. Noch immer findet man auch Conventional-off-the-shelf-(COTS-)Applikationen, die diese Architektur vorweisen.

Kein SSO	Verschiedene Repositories, Verschiedene Passwörter, Viele Prompts,
-----------------	---

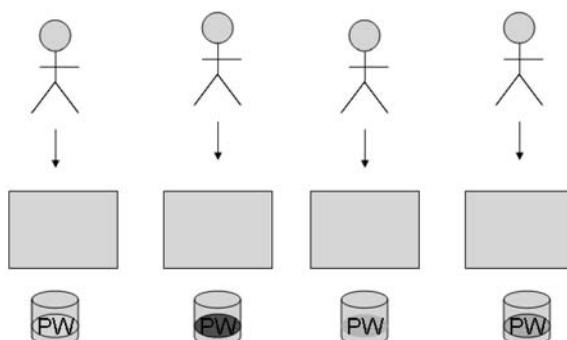


Abb. 6.1 Separate Authentisierung durch jede Einzelanwendung

Erst in der Integration in der Firma kommen die Probleme zu Tage: Jetzt müssen die speziellen Registries abgeglichen werden, sei es durch automatische Datenbankreplikationen oder durch selbst geschriebene Extract- und Importprogramme. Im Falle des Ein- oder Austritts von Mitarbeitern müssen alle Registries auf den neuesten Stand gebracht werden. Dies ist kaum einmal in der Realität der Fall, so dass oft noch nach Jahren ausgeschiedene User bei bestimmten Applikationen registriert sind.

Die User leiden in diesem Szenario an einer Vielzahl von Identitäten und Passwörtern, die sich kaum vereinheitlichen lassen – können doch zum Beispiel die Applikationen verschiedene eingebaute Regeln zum Passwort Aging halten. Die folgende Topologie erlaubt deshalb die Vereinfachung der Passwortbehandlung durch den User, indem die verschiedenen Beziehungen zwischen User und Applikation hinter einem Frontend versteckt werden (Abb. 6.2).

Diese Architektur wird als ein Beispiel für die Realisierung des so genannten *Single-Sign-On* (SSO) praktiziert und wird uns später bei der Behandlung von Single-Sign-On für Portale wieder begegnen. Beim Single-Sign-On hat der Benutzer nur noch ein Passwort für verschiedene Applikationen zu verwalten (und wird auch nur einmal danach gefragt). Die Bequemlichkeit für den User wird jedoch mit einem permanenten Aufwand für das Abgleichen der Registries erkauft. Allerdings gibt es für diese Aufgabe verschiedene Werkzeuge, die entweder durch Replikation der Datenbank oder durch regelbasierten Update die Daten abgleichen.

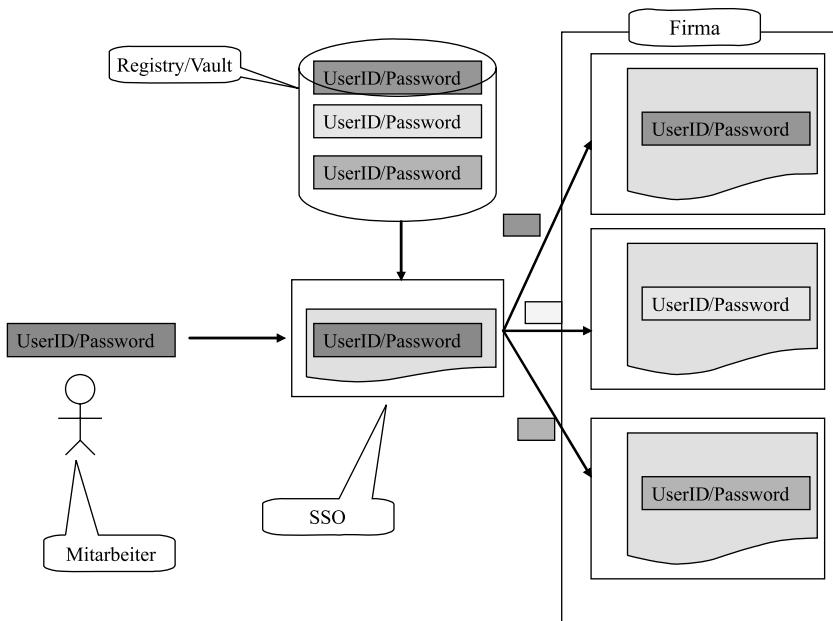


Abb. 6.2 SSO Frontend mit Legacy-Anschluss

Hinzu kommt noch eine andere Überlegung: Wir haben oben die passwortbasierte Authentisierung als „schwache Authentisierung“ bezeichnet. Mit dieser Architektur öffnen wir nun eine ganze Reihe von Applikationen durch eine einmalige, schwache Authentisierung.

Um diese Probleme zu umgehen, kann eine dezentrale Alternative eingesetzt werden: Eine Smartcard als „Digital Wallet“ zur Aufbewahrung aller nötigen Passwörter. Man spricht auch von einem *Hardware Security Module* (HSM), das zur Speicherung von Schlüsseln, aber auch zur Generierung von Signaturen und anderen Krypto-Operationen verwendet werden kann. In unserem Fall wird es lediglich zur Speicherung der Passwörter eingesetzt (Abb. 6.3).

Hier authentisiert sich der User mit einer PIN gegenüber der Smartcard. Je nach Applikation wird dann die korrekte UserID/Passwort-Kombination ausgelesen und zur Anmeldung verwendet. Somit sind in diesem Fall keine Änderungen an den Applikationen oder deren Registries nötig. Nun zur zweiten Frage: Wird von den Applikationen jeweils ein lokales oder eine gemeinsame Registry benutzt? Bei der Nutzung einer gemeinsamen Registry ist jetzt darauf zu achten, dass sie in der Lage sind, die Authentisierung eines Users gegenüber einem konfigurierbaren LDAP vorzunehmen (Abb. 6.4).

In einem zweiten Schritt können dann die UserID und Passwort Kombinationen vereinheitlicht werden (Abb. 6.5).

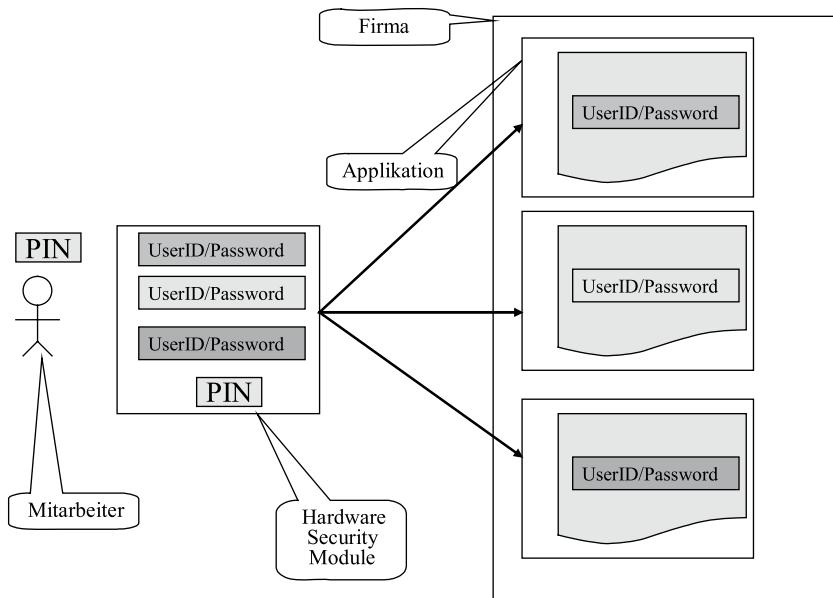


Abb. 6.3 Smartcard-basierter SSO mit Legacy-Anschluss

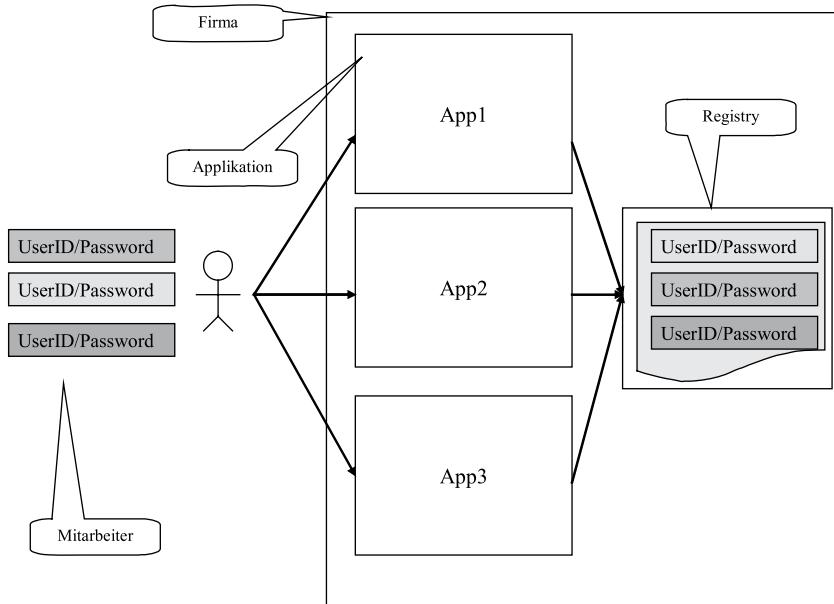


Abb. 6.4 Anpassung der Applikationen auf eine gemeinsame Registry

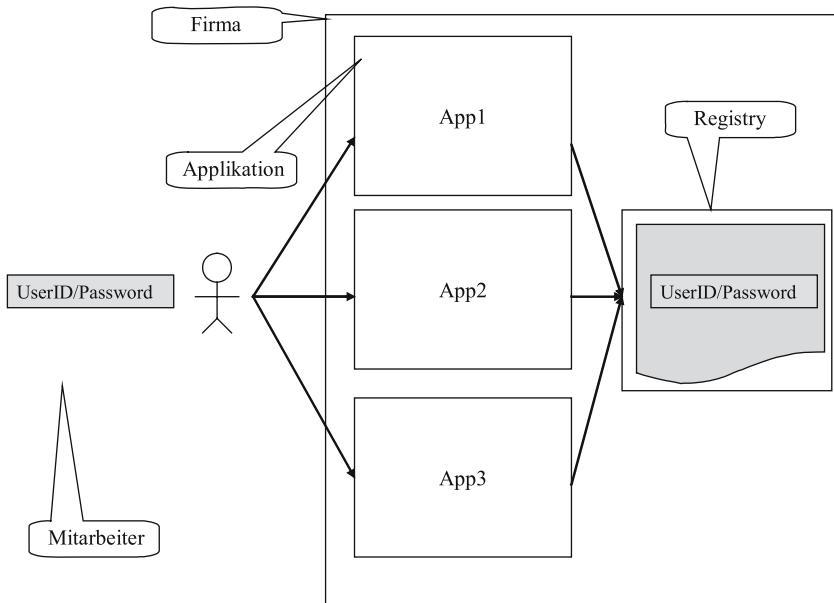


Abb. 6.5 Synchronisierung der Passwörter

Auch hier stellt sich das gleiche Problem wie oben: Jetzt schützt nur noch eine schwache Authentisierung eine ganze Reihe von Applikationen. Das hat ganz praktische Konsequenzen: Konnte vorher ein User aufstehen und in die Mittagspause gehen, ohne sich bei einer Applikation abzumelden, so bestand Gefahr nur für diese spezielle Applikation bzw. deren Daten. Verlässt der User jetzt bei Nutzung einer gemeinsamen Registry seine Arbeitsstation im eingeloggten Zustand, so stehen potentiell alle Applikationen, auf denen er Rechte besitzt, für Fremde offen.

Des Weiteren gelangen die User Credentials weiterhin an jede beteiligte Applikation. Nehmen wir nun an, eine Applikation besitzt einen Fehler oder wurde manipuliert, so kann jetzt diese Applikation andere Applikationen im Namen des Users aufrufen. Sie kann ihn auf beliebigen Applikationen impersonifizieren, ohne dass der User dies will – im schlimmsten Fall sogar, ohne dass überhaupt ein aktueller User-Request vorlag, indem die manipulierte Applikation die Credentials vom letzten Request speichert und wieder verwendet (Abb. 6.6).

Schauen wir uns obiges Szenario etwas genauer an, so stellen wir fest, dass es keine Möglichkeit gibt, zu prüfen, ob die Weiterleitung des Requests von App2 an App3 im Sinne des Mitarbeiters erfolgt (also legal ist) oder ob App2 manipuliert wurde. Der Mechanismus, mit dem App3 den Sender authentisiert, ist in beiden Fällen der gleiche, nämlich mittels UserID und Passwort. Das bedeutet, dass App3 auch gar keine Vorstellung von anderen Formen der Authentisierung besitzt, beispielsweise einer prüfbaren Delegation von Aufträgen an Dritte.

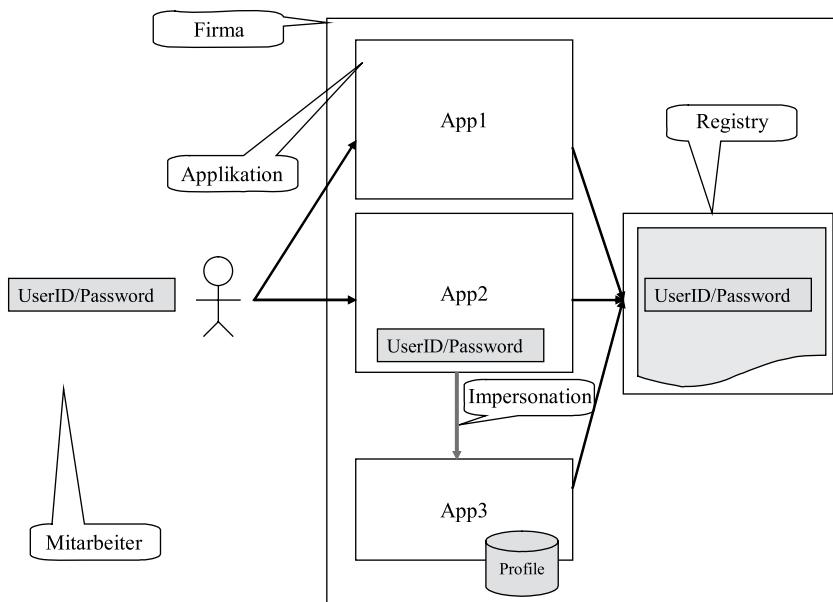


Abb. 6.6 Impersonation bei Nutzung gemeinsamer Passwörter

Letztlich bedeutet dies, dass nicht nur die Speicherung von Credentials in jeder Applikation ein Problem darstellt, sondern dass die Authentisierung durch jede Applikation selbst das zentrale Problem ist.

Solange wir also im Rahmen der Authentisierung mittels UserID und Passwort zulassen, dass alle Applikationen die Credentials erhalten, können wir nicht von einem sicheren Single-Sign-On sprechen und verfügen auch nicht über einen sicheren Delegationsmechanismus. In den folgenden Kapiteln werden wir Authentisierungsmethoden kennen lernen, die auf symmetrischen wie auch asymmetrischen Verfahren beruhen, und die sowohl sicheres Single-Sign-On wie auch eine sichere Delegation erlauben. Zuvor fassen wir noch einmal die Ergebnisse unserer Diskussion der passwort-basierten Authentifikation zusammen:

- Mit Passwörtern allein gibt es keine sichere Delegation.
- Passwörter skalieren nicht mit der Anzahl der Systeme und Applikationen, wenn nicht eine gefährliche Synchronisation betrieben wird.
- Auf jeden Fall sollte die Speicherung der Passwörter nicht in der Applikation selber stattfinden, sondern ausgelagert werden an eine spezielle gemeinsame Registry.
- Ein Single-Sign-On (SSO) ist zwar im Prinzip realisierbar, hat jedoch bedeutende Nachteile und verringert die Sicherheit des Gesamtsystems beträchtlich.
- Hardware Security Module können die User von der Verwaltung vieler Passwörtern entlasten, bieten jedoch ebenfalls kein wirkliches SSO. Dies wird für den User sichtbar, wenn er etwa eine Aufforderung einer einzelnen Applikation oder eines einzelnen Services bekommt, ein bestimmtes Passwort sei zu erneuern.

6.3 Delegation und Impersonation

Die Akteure (das heißt Personen oder Maschinen) in einem verteilten System interagieren miteinander, direkt oder über dritte und weitere Beteiligte. Um diese Interaktionen abilden zu können, sind die Entwickler von Applikationen mit Begriffen wie „Delegation“, „Impersonation“ oder „Propagation“ konfrontiert, deren Bedeutung je nach Kontext und Autor zu wechseln scheint, und die zudem untereinander Abhängigkeiten besitzen. Die Übersetzung der englischen Begriffe ins deutsche sorgt zudem für Unklarheiten. In der obigen Diskussion der Authentisierung in verteilten Systemen haben wir manche dieser Begriffe bereits angerissen. Im Folgenden wollen wir diese Begriffe genauer klären und versuchen, exakte Definitionen zu geben.

6.3.1 Begriffsklärung

Betrachten wir zur Einstimmung und Verdeutlichung der Begriffe zunächst folgende Szenarien aus dem täglichen Leben:

1. Jemand kommt in eine Polizeikontrolle und muss den Personalausweis vorzeigen. Die Polizisten vergleichen Gesicht mit dem Bild im Ausweis und können die Person so *identifizieren*.

2. Eine gehbehinderte ältere Frau bittet eine Nachbarin, für sie einen bestimmten Geldbetrag vom Automaten zu holen und gibt ihr dazu ihre EC-Karte und ihre PIN. Damit setzt sie Vertrauen (*Trust*) in die Nachbarin. Sie ist zwar Initiator (*Originator*) der Abhebung, *delegiert* sie aber an die Nachbarin.

Die Nachbarin geht mit der Karte und PIN zum Automaten und hebt Geld ab. Sie *impersonifiziert* die ältere Frau, denn der Geldautomat hat keine Möglichkeit, zu entscheiden, wer die Abhebung tatsächlich vorgenommen hat. Deshalb kann die Nachbarin auch das in sie gesetzte Vertrauen missbrauchen und zum Beispiel einen größeren Betrag als gewünscht abheben oder den Kontostand der älteren Frau einsehen. Falls sie selbst verhindert sein sollte, hat sie die Möglichkeit, die EC-Karte mit PIN an ihre Tochter weiterzugeben, also die Abhebung weiter zu delegieren.

3. Alice bittet Bob, ein Auto auf ihren Namen zuzulassen. Dazu stellt sie Bob (dem *Stellvertreter* oder *Proxy*) eine Vollmacht aus. Die Aufgabe, das Fahrzeug zuzulassen, wird also delegiert. Die Zulassungsstelle überprüft nun

- die Identität des Stellvertreters Bob
- die Gültigkeit der Vollmacht
- die Tatsache, dass sich die Vollmacht auf Bob bezieht.

Wenn alle drei Prüfungen erfolgreich sind, so behandelt die Zulassungsstelle Bob wie den Fahrzeughalter Alice – allerdings ohne dabei zu vergessen, dass es sich nur um einen Stellvertreter handelt, der „im Auftrag agiert“. Versucht Bob nun mit der gleichen Vollmacht, Geld von Alices Konto bei der Bank nebenan abzuheben (also Alice zu impersonifizieren bzw. sich als Alice zu *maskieren*), so wird er scheitern, da er sich nicht als Alice authentifizieren kann und sich seine Vollmacht nur auf die Zulassung eines Autos beschränkt.

4. Einem Agenten A wird von seiner Dienststelle ein Passwort zugesandt. Damit soll er am Ziel mit einem anderen, ihm persönlich nicht bekannten Agenten B Kontakt aufnehmen und sich durch das Passwort ausweisen. Das Passwort wird jedoch vom Geheimdienst abgefangen und dazu genutzt, anstelle von A einen eigenen Agenten X zu B zu schicken. Mit Hilfe des Passworts kann sich X gegenüber B als A ausgeben (so genanntes *masquerading*).

5. Die Kfz-Zulassungsstelle benötigt zur Durchführung einer Zulassung gewisse vertrauliche Fahrzeugdaten vom TÜV. Es gibt nun mehrere Möglichkeiten an diese Daten zu kommen:

- Die Zulassungsstelle schickt die Antragstellerin Carol direkt zum TÜV. Dort wird ihre Identität anhand des Ausweises geprüft.
- Ein Mitarbeiter der Zulassungsstelle ruft beim TÜV an, identifiziert sich, da er schon häufiger angerufen hat, über seine Stimme und fordert die Daten an.
- Die Zulassungsstelle ruft beim TÜV an und identifiziert sich wie unter b). Dann erklärt sie, dass sie die Daten im Auftrag von Carol benötigt.

- d) Die Zulassungsstelle handelt wie unter d), fäxt jedoch gleichzeitig eine von Carol ausgestellte Vollmacht zur Herausgabe der Daten an den TÜV.
- e) Gleicher Ablauf wie unter d), jedoch ruft der TÜV zusätzlich bei Carol an und prüft dadurch die Richtigkeit der Delegation über die Zulassungsstelle.

In diesen Szenarien aus dem täglichen Leben unterscheiden wir sehr genau zwischen einem Auftraggeber (*Originator*) und einem Mittelsmann oder Stellvertreter (*Proxy*). Hingegen reden wir im Umfeld verteilter Systeme oft wesentlich ungenauer nur vom „Requestor“, „Caller“, „Sender“ oder „Client“, ohne dabei anzudeuten, ob es sich wirklich um den Originator oder eine dazwischen geschaltete Instanz handelt, die im Auftrag agiert.

Im Folgenden wird häufig der Begriff *Originator* verwendet, und zwar im Sinne eines Akteurs, der sowohl die Inhalte einer Kommunikation (Nachrichten) erzeugt als auch der Absender einer Nachricht ist. Splitten sich diese beiden Eigenschaften auf, so unterscheiden wir zwischen *Sender* (Absender einer Nachricht) und *Autor* (Erzeuger einer Nachricht). Der Sender einer Nachricht kann dies im Auftrag des Autors tun oder nicht. Im ersten Fall spricht man auch von einem Stellvertreter, im zweiten Fall handelt es sich um einen Angreifer. Es muss also geprüft werden, ob der Sender tatsächlich im Auftrag des Autors handelt. Dabei ist wichtig, wo diese Prüfung stattfindet, welche Eigenschaften sie besitzt und auf welche Weise das Ergebnis der Prüfung weiter gegeben wird.

Auch beim Auftrag (*Request*) selbst unterscheiden wir in unserem sozialen Umfeld genau zwischen dem, was ein Originator ursprünglich gewollt hat und dem, was im Rahmen der Weitergabe des Auftrags von anderen getan wurde. Unsere Beispiele zeigen, wie wichtig es ist, dem Stellvertreter keine globalen Vollmachten zu erteilen, sondern diese eng auf den delegierten Auftrag zu begrenzen.

Aus Business-Sicht steht hinter der Frage nach dem Originator einer Nachricht natürlich oft schlicht die Frage, wer etwas bestellt, also letztlich wer etwas zu verantworten hat. Technisch ausgedrückt ist dies die Frage nach der Nicht-Abstreitbarkeit (Non-Repudiation) von Aufträgen oder Handlungen.

Damit kommt der ersten Prüfung der Identität des Originators im verteilten System eine besondere Bedeutung zu. Die Stelle, die eine Authentisierung durchführt, kann einen unmittelbaren Beweis der Identität eines Originators erhalten, indem der Originator nachweist, dass er einen behaupteten Namen besitzt. Dies kann nur derjenige sein, der tatsächlich den Request stellt, wie unser Beispiel 3 zeigt: Nur der persönlich anwesende Stellvertreter Bob kann unmittelbar bei der Zulassungsstelle authentisiert werden, und zwar durch eine Nachprüfung seiner Behauptung („Mein Name ist Bob“) durch einen Proof-of-Possession (Gesicht und Ausweis). Der Personalausweis garantiert dabei den Zusammenhang von Name und Gesicht – ausgestellt von einer allgemein akzeptierten Autorität (Trusted Third Party – in diesem Fall die Bundesdruckerei).

Schwieriger wird die Prüfung der Gültigkeit der Vollmacht. Natürlich muss darin stehen, dass Bob berechtigt ist, das Auto in Alices Namen zuzulassen. Um aber sicher zu stellen, dass Bob sich die Vollmacht nicht selbst ausgestellt hat, muss sie eine Unterschrift des Auftraggebers Alice beinhalten. Um diese Unter-

schrift beurteilen zu können, benötigt die Vollmacht noch eine Kopie des Personalausweises des Auftraggebers, Damit kann der Zusammenhang der Identität „Alice“ mit ihrer Unterschrift geprüft werden und anschließend die Übereinstimmung der Unterschrift im Ausweis mit der Unterschrift unter der Vollmacht. Wiederum ist an dieser Stelle Vertrauen nötig in den Herausgeber des Ausweises.

6.3.2 Delegation von Aufträgen

Wir sprechen von *Delegation*, wenn ein Akteur (so genannter *Intermediate*), der nicht der Originator ist, einen Auftrag (Request) zur weiteren Bearbeitung an einen anderen Akteur schickt. Dieser Akteur kann Endpunkt (*Target*) des Requests oder ein weiterer Intermediate sein. In den Beispielen oben ist die hilfsbereite Nachbarin ein Intermediate der gehbehinderten älteren Dame, Bob ein Intermediate von Alice und die Zulassungsstelle ein Intermediate der Autobesitzerin Carol. Das Beispiel der älteren Dame, die ihre geheime Authentisierungsinformation (in diesem Fall die PIN) weitergibt, zeigt, wie schädlich diese Form der Delegation ist, da sie dem Intermediate eine vollständige Impersonation des Originators erlaubt. Eine Einschränkung der Vollmachten ist nicht möglich, sowein wie die Möglichkeit, auf Seiten der Bank nachzuvollziehen, wer genau die Transaktionen durchgeführt hat. Schließlich hat die ältere Dame die Möglichkeit, die mit Hilfe ihrer ec-Karte und PIN durchgeföhrten Aktionen abzustreiten – ob sie damit vor Gericht Erfolg hätte, ist eine andere Frage. Wenden wir uns nun dem fünften Beispiel aus dem letzten Abschnitt zu und schauen uns an, wie das Target TÜV mit den an es gerichteten Anfragen umgeht. Dazu untersuchen wir die Optionen des TÜV zur Authentisierung von Anfragen:

- a) Persönliches Erscheinen des Antragstellers mit Personalausweis
- b) Anfragen von Mitarbeitern der Polizei – oder von der Polizei selbst
- c) Anfragen von Mitarbeitern der Zulassungsstelle – oder der Zulassungsstelle selbst, sofern sie im Zusammenhang mit einem konkreten Auftrag zur Zulassung stehen
- d) Anfragen von autorisierten TÜV Mitarbeitern
- e) Telefonische Anfragen von Privatleuten mit Namensnennung

Im Fall a) haben wir das schon bekannte Authentisierungsverfahren, mit dem auch Bob als Stellvertreter von Alice bei der Zulassungsstelle authentisiert wurde. Es benützt ein von einer Trusted-Third-Party ausgestelltes Zertifikat bei gleichzeitiger Prüfung gegen ein Biometrisches Merkmal (Gesicht).

Im Fall b) erhält eine ganze Institution Zugriff auf die Daten und zwar ohne Einschränkung. Das bedeutet, es muss entweder eine eigene Identität für die Institution „Polizei“ geschaffen werden und alle Mitarbeiter der Polizei können diese Identität beweisbar annehmen (die Institution impersonifizieren bzw. sich als diese maskieren) oder alle Mitarbeiter der Polizei erhalten persönliche Identitäten im System des TÜV. Eine dritte Variante besteht darin, dass Mitarbeiter sich unter

ihrer eigenen, von der Polizei vergebenen Identität anmelden. Dies setzt voraus, dass eine gemeinsame Domain (oder auch Realm) für die Rechner der beiden Ämter eingerichtet wurde. Der Rechner des TÜV vertraut dann dem Polizeirechner und übernimmt die dort ermittelte Identität des Mitarbeiters. Dasselbe gilt im Fall c), jedoch mit der zusätzlichen Einschränkung, dass die Anfrage nur dann erfüllt wird, wenn sie im Zusammenhang mit einem existierenden Auftrag steht. Das bedeutet, der TÜV möchte wissen, für wen der Auftrag auszuführen ist. Die Zulassungsstelle kann dazu entweder nur den Namen des Auftraggebers oder aber dessen Vollmacht plus Ausweiskopie an den TÜV weitergeben. In der ersten Variante muss das Target viel Vertrauen in den Intermediate setzen. Werden weitere Daten wie eine Vollmacht zur Auftragsvergabe weitergereicht, kann der TÜV die Gültigkeit in ähnlicher Weise prüfen, wie es die Zulassungsstelle selbst mit der von Alice für Bob ausgefüllten Vollmacht getan hat.

Die Variante d) bietet einen indirekten Zugang für externe Stellen zu den Daten: Die autorisierten Mitarbeiter könnten theoretisch ohne jeden Auftrag beliebige Daten ermitteln und weitergeben.

Variante e) ermöglicht letztlich jede Form von Impersonation oder Maskierung, da die im Telefongespräch behauptete Identität lediglich übernommen, jedoch nicht geprüft wird. Dies stellt natürlich einen besonders gefährlichen Umgang mit Authentisierungsdaten in verteilten Systemen dar.

Zusammenfassend halten wir folgendes fest:

- Die unmittelbare Authentisierung eines Originators ist oft nur an einem Punkt (der Stelle die den unmittelbaren Kontakt mit dem Originator besitzt) eines verteilten Systems möglich. Nachgeordnete Akteure (Intermediates) besitzen die Möglichkeit der Authentisierung mangels Authentisierungsdaten meist nicht.
- Die Weitergabe von geheimen Daten zur Authentisierung (zum Beispiel eines Passworts) ist eine Form von schädlicher Impersonation.
- Im Falle der Verwendung von PKI basierter Authentisierung auf Seiten des Originators ist eine Weitergabe der Authentisierungscredentials (des privaten Schlüssels) ohnehin nicht zulässig und die einfache Impersonation des Originators durch Intermediates entfällt.

Daraus ergeben sich als sicherheitstechnische Anforderungen an den Umgang mit delegierten Requests in verteilten Systemen:

- Der Originator authentisiert sich an der Eintrittsstelle in eine Infrastruktur gegenüber einer Stelle. Intermediates, die im Verlauf eines Requests aufgerufen werden, erhalten keine Credentials des Originators, die zu einer unmittelbaren Authentisierung verwendet werden könnten (z. B. Geheimnisse). Die Intermediates vertrauen auf die Authentisierung der Eintrittsstelle, erhalten aber Kenntnis der Identität des Originators und können in seinem Auftrag handeln bzw. die dabei verwendeten Rechte auf diejenigen des Originators einschränken.
- Die unmittelbare und gegenseitige Authentisierung von Intermediates untereinander oder gegen Targets ist unbedingt nötig, garantiert dem Target aber nicht, dass der Intermediate wirklich im Auftrag des Originators handelt.

- Es sollten nur solche Aktionen ausgeführt werden, die mit dem Willen, aber auch den Rechten des Originators in Einklang stehen. Das bedeutet, die beteiligten Systeme sollten sich im Sinne der Business-Logik so verhalten, als wäre der Originator persönlich auf jedem System angemeldet und hätte dort eine initiale Authentisierung vollzogen.
- Idealerweise sollte eine Kette von Daten existieren, die vom Originator über die beteiligten Intermediates bis hin zum Target genau belegt, wer was mit welcher Identität und mit welcher Berechtigung getan hat.

Wir werden in den folgenden Kapiteln technische Möglichkeiten kennen lernen, diese Anforderungen zu erfüllen. Anzumerken wäre noch, dass der Originator bei seiner Anmeldung an der Eintrittsstelle in einigen Standards auch als „Holder of Key“ bezeichnet wird, da er ja die unmittelbaren Authentisierungs-Credentials besitzt. Idealerweise benötigt der „Holder of Key“ seine Schlüssel nur gegenüber einer Stelle in einem verteilten System, die sich dadurch authentisiert. Dadurch wird eine Verbreitung der Schlüssel verhindert, dennoch müssen nachgelagerte Beteiligte ein hohes Maß an Vertrauen in vorgelagerte Intermediates aufbringen. Denn aus der bloßen Tatsache, dass sich ein Originator erfolgreich authentisiert hat, können nachgelagerte Targets nicht auf den Willen des Originators schließen. Hier zeigt sich die Grenze kanalbasierter Sicherheit. Bessere Möglichkeiten ergeben sich dann durch objektbasierte Sicherheit wie im Falle der Vollmacht in unserem Beispiel. Mit Vollmachten – also digital signierten Willenserklärungen – lässt sich auch für nachgeordnete Targets der Willen des Originators nachprüfen. Dadurch verringert sich auch die Bedeutung der Infrastruktur für die Sicherheit in einem verteilten System. Vollmachten erlauben darüber hinaus auch eine wesentlich bessere Umsetzung des POLA-Prinzips, da sie die Autorität der Beteiligten auf den jeweils vorliegenden Request des Originators begrenzen können.

Kapitel 7

Basisprotokolle

Im voran gegangenen Kapitel haben wir gesehen, dass die Verteiltheit eines Systems eine sichere Kommunikation der Subsysteme untereinander erfordert. Dies schließt neben der gegenseitigen Authentifikation der Endpunkte und einer verschlüsselten Kommunikation auch die Themen „Single-Sign-On“ und „Delegation“ ein. Das Kapitel „Kryptografische Algorithmen“ hat uns erste Anhaltspunkte gegeben, wie sich diese Dienste aus technischer Sicht realisieren lassen. Nun wollen wir einen Schritt über die Algorithmen hinaus gehen und zeigen, wie diese in weit verbreiteten Sicherheitsprotokollen eingesetzt werden, um die angesprochenen Sicherheitsziele für verteilte Systeme zu erreichen. Häufig enthalten diese Protokolle auch die Möglichkeit, einen symmetrischen Schlüssel zwischen zwei Kommunikationspartnern zu vereinbaren. Wie wir sehen werden, ist ein solches „Key Agreement“ zwischen zwei einander unbekannten Partnern grundsätzlich nur möglich, wenn beide einer „dritten Partei“, also einer *Trusted Third Party* (TTP) vertrauen.

7.1 http Authentication

Das Hypertext Transfer Protocol (http) spezifiziert, auf welche Weise Webseiten von einem Web-Server zum Browser des Client übermittelt werden. Häufig ist man in der Situation, dass gewisse Webseiten nur einem bestimmten Personenkreis zugänglich sein sollen. Die für http eingesetzten Authentifikationsmechanismen zur Kontrolle des Zugriffs auf geschützte Webseiten sind in [RFC 2617] spezifiziert. Beide Mechanismen basieren auf einem symmetrischen Schlüssel, beide bieten eine einseitige Authentifikation des Clients, jedoch ist das Sicherheitsniveau sehr unterschiedlich, wie wir gleich sehen werden. Moderne Frameworks für die Entwicklung von Web-Applikationen wie zum Beispiel ASP.NET unterstützen beide Arten der Authentifikation. Dabei wird, unabhängig von der tatsächlich eingesetzten Authentifizierungsmethode, ein nicht authentifizierter User zunächst auf eine individuell zu gestaltende LogIn-Seite geleitet, wo er seine Credentials eingeben kann.

7.1.1 Basic Authentication

Bei der Basic Authentication sendet der Client seine erste Anfrage ohne jegliche Credentials. Erfordert die angefragte Seite eine vorherige Authentifikation, antwortet der Server mit einer „Unauthenticated“-Fehlermeldung und der Aufforderung an den Client, seine Credentials, also UserID und Passwort, zu übermitteln. Diese werden vom Client im Rahmen des http-Headers im Klartext an den Server geschickt und dort verifiziert. Somit können die Credentials ohne Probleme von einem Angreifer abgefangen und zur Impersonation wieder verwendet werden. Diese Authentifikationsmethode sollte deshalb nicht ohne zusätzliche Sicherheitsmechanismen wie zum Beispiel einen SSL-Tunnel verwendet werden, zumal mit der Digest Authentication ohnehin eine sicherere Alternative zur Verfügung steht.

7.1.2 Digest Authentication

Die http Digest Authentication basiert auf einem einfachen Challenge-Response-Schema. Die Unauthenticated-Meldung des Servers enthält in diesem Fall eine so genannte *Nonce* (Abkürzung für „Number Used Once“), die als Challenge an den Client dient. Der Client berechnet daraufhin eine gültige Response, indem er die Nonce zusammen mit seiner UserID, seinem Passwort, der angeforderten URI und der gewünschten http-Methode hasht. Als Hashfunktion wird per Default MD5 verwendet. Somit wird die größte Schwäche des Basic Authentication Schemas umgangen, da das Passwort niemals im Klartext versendet wird. Allerdings muss der Server bei dieser Methode die Passwörter in einer Weise speichern, die es ihm möglich macht, die Response auch zu verifizieren. Eine Speicherung in gehaschter oder verschlüsselter Form wie etwa beim UNIX-Filesystem kommt deshalb nicht in Frage, was die Passworddatei beim Server zu einem sehr lohnenden Ziel für Angreifer macht.

Auch bei der Digest Authentication bleiben noch einige Fragen offen: Was passiert, wenn der Nutzer die Seite verlässt und später wieder auf sie zugreifen möchte? Im schlimmsten Fall muss er seine Credentials jedes Mal wieder neu eingeben. Üblicherweise wird dieses Problem über einen Cookie-Mechanismus, ähnlich dem bei ebay verwendeten, gelöst. Aber auch hier bleibt die Frage, wie der Cookie und die angeforderte Seite selbst sicher zu übertragen ist. Das Digest Authentication Protocol bietet hier keine Hilfestellung, da keine Session Keys oder Verschlüsselungsalgorithmen vereinbart werden. Außerdem bezieht sich die Digest Authentication nur auf jeweils genau eine URI. Für eine andere URI in derselben Domäne muss im Prinzip wieder eine neue Response berechnet werden, ganz zu schweigen davon, wenn eine Seite in einer anderen Domain angefordert wird.

7.2 Kerberos

Kerberos ([RFC 1510], [RFC 4120]) ist der Name des dreiköpfigen Hundes, der in der griechischen Mythologie den Zugang zum Hades bewacht. Heutzutage bewacht Kerberos den Zugang zu einem verteilten System, das dem Nutzer verschiedene Dienste (Datenbankzugriff, Druckservice, etc.) anbietet. Wie der gerade beschriebene Authentifikationsmechanismus für http stellt Kerberos ein auf symmetrischen Verfahren basierendes Authentifikationsprotokoll dar, allerdings erweitert um einige wesentliche Funktionalitäten: Kerberos unterstützt Single-Sign-On, also die einmalige Anmeldung eines Nutzers für alle Dienste, die unter der Kontrolle eines zentralen Servers stehen, und darüber hinaus auch Authorisierung und Delegation. Aufgrund dieser Features, die gerade in verteilten Systemen von großer Wichtigkeit sind, hat Kerberos in jüngster Zeit an Bedeutung gewonnen, nachdem es lange Zeit nur sehr selten eingesetzt wurde¹⁴.

7.2.1 Funktionsweise

Angenommen, Sie treffen auf einer Party eine Ihnen unbekannte Person B. Wenn Sie wissen möchten, wer B ist, haben Sie dazu drei Möglichkeiten (die Regeln der Höflichkeit einmal außer Acht gelassen):

- Sie authentisieren B selbst, indem Sie B nach dem Personalausweis fragen.
- Sie fragen die Person B nach ihrem Namen und vertrauen B, dass der Name stimmt.
- Sie bitten den Gastgeber A, Sie einander vorzustellen.

Welche Rolle spielt der Begriff „Vertrauen“ in diesen drei Fällen? Im ersten Fall setzen Sie Ihr Vertrauen in die Stelle, die den Ausweis ausgestellt hat. Die Authentisierung per Ausweis ist sehr sicher, aber auch sehr aufwändig: Der Ausweis muss erstellt werden, B muss ihn bei sich tragen und auf Verlangen vorzeigen. Schließlich müssen Sie noch verifizieren, dass der Ausweis tatsächlich der vor Ihnen stehenden Person gehört. Im zweiten Fall vertrauen Sie schlicht der Person B, ohne weitere Informationen zu haben. Dies ist die einfachste, aber auch unsicherste Variante der Authentisierung. Auf einer Party mag das zum Zwecke unverbindlichen Smalltalks in Ordnung sein, aber würden Sie B nach einer solchen Authentisierung auch Ihr Auto leihen? Die dritte Möglichkeit stellt einen Kompromiss dar: Sie vertrauen hier dem Ihnen bereits bekannten Gastgeber A. Vielleicht hat diese Person einmal den Personalausweis von B gesehen und mit dem Gesicht verglichen. Vielleicht wurde die Authentisierung von B aber auch sehr schlampig durchgeführt. Ihre Sicherheit in Bezug auf die Authentisierung

¹⁴ Ein Vorläufer von Kerberos, das Needham–Schroeder-Protokoll, wurde bereits 1978 veröffentlicht. Die Entwicklung der aktuellen Kerberos-Version V5 begann 1989.

von B ist also von der Sorgfalt und Vertrauenswürdigkeit von Person A abhängig. Bei einer gegenseitigen Vorstellung ist dies für die Person B ebenfalls der Fall. Sie setzen jetzt beide Vertrauen in A. Ihr Gastgeber ist in diesem Zusammenhang eine *Trusted Third Party* (TTP): Eine neutrale Instanz, der von anderen Vertrauen entgegen gebracht wird.

Das Konzept der TTP ist aus mehreren Gründen in verteilten Systemen sehr wichtig:

- Die Struktur der Vertrauensverhältnisse ändert sich. Es müssen nicht mehr alle Beteiligten einander vertrauen, sondern jeder Beteiligte vertraut nur noch der TTP. Das bedeutet, dass im Falle der Kompromittierung einzelner der Schaden für alle geringer ist.
- Wie wir gesehen haben, ist bei der Weitergabe von Aufträgen die ursprünglich durchgeführte Authentisierung nicht wiederholbar – zum Beispiel, weil der private Key des Originators natürlich nicht weitergegeben wird. Das bedeutet, dass die Weitergabe von Aufträgen auf Vertrauensverhältnissen beruhen muss.
- Mit Hilfe von TTPs können bisher Unbekannte sich gegenseitig authentisieren, obwohl sie vorher keine Geheimnisse ausgetauscht haben. Das war eines der Kernproblem der passwortbasierten Authentisierung: Sie ist nicht zur ersten (initialen) Authentisierung geeignet, da zuerst durch ein anderes Verfahren Sicherheit in Bezug auf die Identität hergestellt werden muss.

Die Rolle der TTP spielt im Kerberos-System der Kerberos-Server (manchmal auch Authentication-Server genannt). Er kennt alle Teilnehmer des Systems, das bedeutet, er hat mit ihnen eine initiale Authentifikation durchgeführt und einen geheimen, langlebigen Schlüssel vereinbart. Möchte nun die Teilnehmerin Alice sich bei Server Bob anmelden, so teilt sie im so genannten *Authentication Request* ihre Identität und ihren Verbindungswunsch dem Kerberos-Server mit. Daraufhin erzeugt der Kerberos-Server einen Sitzungsschlüssel K_{AB} für die Kommunikation von Alice und Bob und teilt ihn Alice mit, und zwar verschlüsselt mit dem Geheimnis K_A , welches sich Alice und der Kerberos-Server teilen. Auch Bob muss den Sitzungsschlüssel K_{AB} kennen. Deshalb verschlüsselt der Kerberos-Server K_{AB} zusammen mit der Identität von Alice auch mit Bobs Geheimnis K_B und übergibt das Ergebnis an Alice zur Weiterleitung an Bob. Dieser Bitstring heißt *Ticket Granting Ticket* (TGT). Möchte sich Alice nun bei Bob anmelden, schickt sie ihm das TGT. Zum Nachweis, dass sie wirklich Alice ist und nicht etwa ein Angreifer, der das TGT belauscht hat, schickt sie zusätzlich einen so genannten Authenticator an Bob. Dieser ist im Wesentlichen ein mit dem Sitzungsschlüssel K_{AB} verschlüsselter Zeitstempel. Damit beweist Alice, dass sie tatsächlich K_{AB} kennt, also das Geheimnis K_A kennen muss, also diejenige sein muss, die der Kerberos-Server als Alice authentisiert hat. Bei seiner Entscheidung, Alice aufgrund des Authenticators zu Zugang zu gewähren, verlässt sich Bob also darauf, dass die initiale Authentifizierung durch den Kerberos-Server hinreichend sicher war.

Das TGT besitzt nur eine gewisse begrenzte Lebensdauer L , die ebenfalls im TGT mit verschlüsselt wird. Ist L abgelaufen, muss sich Alice ein neues TGT besorgen. Bis dahin kann es Alice aber zu wiederholten Authentifizierungen bei

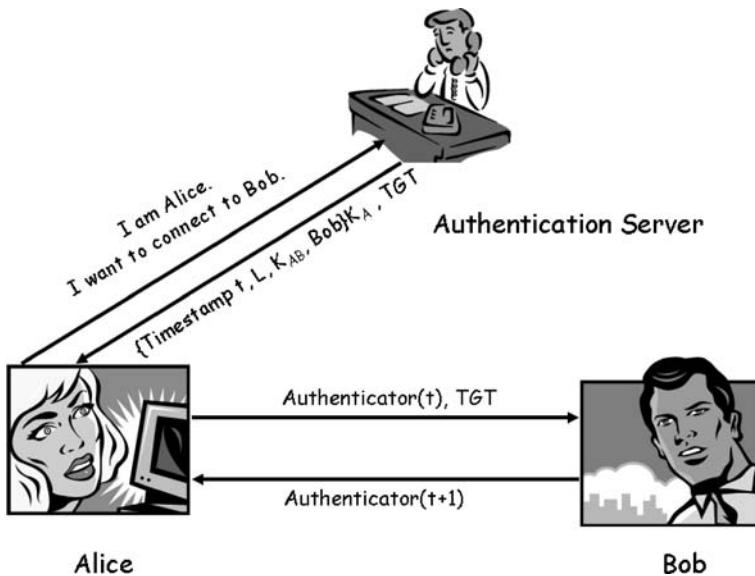


Abb. 7.1 Kerberos-Basisprotokoll mit beidseitiger Authentifikation von Alice und Bob. Die Bezeichnung $\{ X \}_{KA}$ bedeutet: „X mit dem Geheimnis von A verschlüsselt.“

Bob benutzen, sofern sie jeweils neue gültige Authenticators erzeugt. Die Abb. 7.1 verdeutlicht noch einmal den bisherigen Ablauf des Protokolls.

Wir haben gesehen, dass im Kerberos-System eine zentrale TTP, nämlich der Kerberos-Server, für einen Schlüsselaustausch zwischen den Teilnehmern und implizit auch für eine Authentifikation der Teilnehmer untereinander sorgt. Wie steht es aber mit dem Single-Sign-On und der Möglichkeit, die Authorisierung von Alice für die Nutzung einzelner Dienste zu prüfen? Dafür müssen wir das obige Bild etwas erweitern: Stellen wir uns vor, der Server Bob sei seinerseits für eine gewisse Anzahl von Services zuständig (dadurch wird Bob zum *Ticket Granting Server TGS*¹⁵, siehe auch Abb. 7.2).

Bei ihrer Anmeldung beim TGS gibt Alice an, welchen der Services sie gerne nutzen möchte. Aufgrund des TGT und des Authenticators kann der TGS Alices Identität verifizieren und deshalb auch entscheiden, ob Alice zur Nutzung des gewünschten Dienstes autorisiert ist. Ist dies der Fall, bekommt Alice von Bob ein *Ticket* für den gewünschten Dienst. Ganz analog zum TGT besteht das Ticket aus einem Session Key K_{AS} für Alice und den gewünschten Dienst S , der Identität von Alice und einer Lebensdauer. Alice bekommt den Session Key von Bob verschlüsselt mit K_{AB} mitgeteilt. Zur Nutzung des gewünschten Dienstes schickt Alice das Ticket und einen Authenticator (diesmal mit K_{AS} gebildet) an den Server S . Dieser kann damit das Ticket validieren und Alice die Nutzung gestatten. Für die Nutzung verschiedener Dienste stellt Alice mehrfache Anfragen an den TGS,

¹⁵ Häufig werden Kerberos-Server und Ticket Granting Server auch in einem so genannten *Key Distribution Center* zusammengefasst.

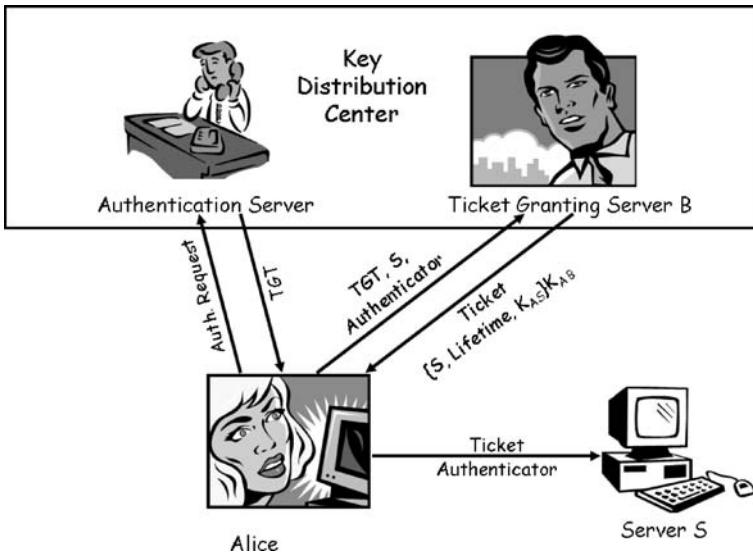


Abb. 7.2 Erweitertes Kerberos

jedes Mal unter Nutzung desselben TGT, aber immer wieder mit neuen Authenticators. Da hierfür Alices ursprüngliches Geheimnis K_A nicht benötigt wird, sondern nur der Sitzungsschlüssel K_{AB} , der von Alices Software zwischengespeichert wurde, kann man mit Recht von Single-Sign-On sprechen.

Durch die Vermittlung des Authentication bzw. Ticket Granting Servers wird also zwischen Alice und dem von ihr gewünschten Server S ein Geheimnis K_{AS} etabliert, welches beide Seiten (auch der Server kann an Alice auf deren Wunsch einen Authenticator schicken) zur gegenseitigen Authentifizierung nutzen. In der Folge kann dieses Geheimnis auch genutzt werden, um einen sicheren Kanal mit Vertraulichkeits- und Integritätsdiensten zwischen Alice und dem Server S aufzubauen.

7.2.2 Principals und Domänen

Ein Single-Sign-On-Konzept benötigt immer auch ein Konzept einer *Domäne* (auch Realm genannt). Damit ist zum einen der Gültigkeitsbereich der Authentifizierung gemeint, also die Frage: Auf welchen Maschinen bzw. bei welchen Applikationen gilt Alices initiale Anmeldung beim Kerberos-Server?

Da aber eine Anmeldung letztlich immer auf kryptografischen Schlüsseln beruht, ist mit Domäne gleichzeitig auch die Menge der Maschinen gemeint, die die Autorität besitzen, über die Gültigkeit der zum Nachweis der Identität vorgebrachten Credentials zu entscheiden. Somit könnte man sagen, dass die Tatsache, ob eine bestimmte Maschine die Anmeldung bei einem Kerberos-Server, der zu einer

bestimmten Firma X gehört, akzeptiert oder nicht, den Verlauf der „Grenze“ der Firma X bestimmt.

Neben dem Begriff der Domäne bringt Kerberos einige weitere zentrale Prinzipien mit, so zum Beispiel die Namensgebung und den Begriff des *Principals*: Damit innerhalb einer Domäne authentisiert werden kann, müssen die Namen der Teilnehmer innerhalb der Domäne einheitlich und bekannt sein. Kerberos regelt die Frage nach der Identität der Teilnehmer durch ein Name@domain-Konzept. Dieses Konzept gilt für alle aktiven Teilnehmer eines Systems, das heißt also auch für nichtmenschliche Teilnehmer wie Server, Services und Applikationen. Jeder dieser Teilnehmer wird *Principal* genannt und besitzt ein gemeinsames Geheimnis mit dem Kerberos-Server.

7.2.3 Attacken auf Kerberos

Hier wollen wir nicht auf reine Implementationsfehler, wie sie in fast allen Protokollen auftreten, eingehen, sondern fundamentale Designprobleme bei Kerberos behandeln, die sich nicht durch eine bloße Korrektur der Software beheben lassen.

An erster Stelle steht hier natürlich das Problem, dass das von den Nutzern mit dem Kerberos Server geteilte, langlebige Geheimnis von diesem zur Erstellung von Session Keys verwendet wird. Der Algorithmus, nach dem dies geschieht, ist nach gutem kryptografischen Brauch öffentlich. Dadurch hängt aber die Sicherheit jedes Sitzungsschlüssels in Kerberos von der Qualität des langlebigen Geheimnisses jedes Nutzers abhängt. Wird also – wie meistens – ein Passwort als Geheimnis verwendet, hängt die Sicherheit von Kerberos von der Qualität des Passworts ab.

Da dieses Geheimnis vom Benutzer am Anfang einer Anmeldung einmal eingegeben werden muss, um ein TGT und einen Sitzungsschlüssel zu erhalten, muss es merkbar sein, was seine Qualität, wie bereits oben besprochen, merklich verringert, und Wörterbuch-Attacken ermöglicht.

Wie sieht so ein Angriff konkret aus? Ein Angreifer besorgt sich unter Alices Namen ein TGT und einem mit Alices Schlüssel K_A verschlüsselten Sitzungsschlüssel. Er wendet nun den öffentlichen Algorithmus zur Sitzungsschlüsselerzeugung auf ein Wörterbuch an und erhält so eine Menge möglicher Sitzungsschlüssel (diese Menge kann schon im Voraus berechnet werden). Gleichzeitig wendet der Angreifer die geratenen Passwörter aus dem Wörterbuch auf den verschlüsselten Sitzungsschlüssel an. Stimmt das Ergebnis mit einem der Schlüssel aus der vorab bestimmten Menge überein, ist sowohl der Sitzungsschlüssel und – schlimmer noch – Alices Passwort K_A bekannt. Im Folgenden kann der Angreifer Alice gegenüber dem Kerberos-Server und damit gegenüber jeder anderen Maschine im System impersonifizieren.

Aber wie kommt der Angreifer an ein TGT? Hier besitzt Kerberos ein Designproblem: Im Original-Kerberos kann jeder ein TGT für einen bestimmten Nutzer verlangen. Man ging davon aus, dass ja nur der wirkliche Benutzer den verschlüsselten Sitzungsschlüssel entschlüsseln und verwenden kann. Ein Angreifer kann

sich also beliebige TGTs ausstellen lassen. Deshalb verlangen neuere Kerberos-Implementationen vom Benutzer so genannte *Preattentification Data* (das sind im Wesentlichen einen mit K_A verschlüsselten Zeitstempel), bevor vom Kerberos-Server ein TGT für diesen Nutzer ausgestellt wird. Natürlich können auch die Preattentification Data bei bekanntem Zeitstempel einer Wörterbuch Attacke unterzogen werden. Dazu ist jedoch zunächst einmal das Belauschen von Preattentification Data für einen ganz bestimmten Nutzer notwendig. Eine Vor- ausberechnung möglicher Preattentification Data im großen Stil ist nicht möglich, da sich diese mit dem Zeitstempel immer wieder ändern.

Ein weiteres Sicherheitsproblem von Kerberos besteht in der zentralen Bedeutung des Kerberos-Server. Er stellt einen „Single Point of Failure“ dar, denn dort sind die Geheimnisse aller Principals gespeichert und eine erfolgreiche Attacke darauf hätte die Kompromittierung des gesamten Systems zur Folge.

Schließlich dienen in Kerberos die Authenticators zur Authentifizierung und Validierung von Tickets. Diese bestehen aber im Wesentlichen aus verschlüsselten Zeitstempeln, das heißt, so lange ein Zeitstempel sich nicht ändert, können belauschte Authenticators von einem Angreifer zur Authentifizierung eigener Requests eingesetzt werden. Deshalb sollten die Zeitstempel in Kerberos sich möglichst schnell (der Default der Kerberos-Implementation von Windows 2000 liegt bei fünf Minuten) ändern. Dies setzt natürlich die Existenz synchronisierter und vertrauenswürdiger Uhren in allen Maschinen der Domäne voraus. Gelingt es einem Angreifer, diese Uhren zu manipulieren und die Zeit gleichsam zurück zu stellen, kann er ebenfalls vorab belauschte Authenticators wieder einspielen.

7.2.4 Delegation mit Kerberos

Angenommen, Nutzerin Alice hat sich via Kerberos bei einem Server S angemeldet, das heißt, sie besitzt ein TGT sowie ein Service Ticket für Server S . Außerdem teilen sich Alice und der Server einen Sitzungsschlüssel K_{AS} , der in dem Service Ticket enthalten ist. Nehmen wir weiter an, beim Erfüllen der Aufgaben für Alice ergibt sich für S die Notwendigkeit, auf ein Dokument zuzugreifen, dass auf einem weiteren Server T zu finden ist. Anders gesagt: Der Server S soll *im Auftrag* von Alice auf das entfernte Dokument zugreifen. Natürlich könnte S ein eigenes Service Ticket für T beim Ticket Granting Server B beantragen. Um aber Server T zu ermöglichen, den Zugriff auf das gewünschte Dokument anhand der tatsächlichen Identität des Urhebers der Anfrage zu erlauben oder zu verweigern, muss es eine Möglichkeit geben, die Identität von Alice an T weiter zu geben. Dazu transferiert Alice ihr TGT (das ihren Namen enthält) und den darin enthaltenen Sitzungsschlüssel K_{AB} an Server S , beides verschlüsselt mit dem Sitzungsschlüssel K_{AS} . Damit kann S ein Service Ticket für T beim TGS beantragen, und zwar im Namen von Alice, da das TGT ja von Alice stammt. Man beachte, dass der TGS glaubt, mit Alice zu sprechen und das Service Ticket unter dieser Prämisse ausstellt. S kann nun den vom TGS erzeugten Sitzungsschlüssel K_{AT} entschlüsseln und mit

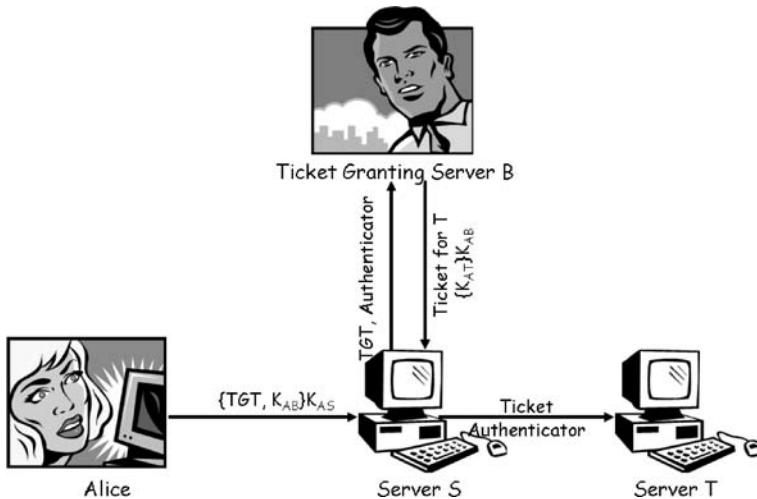


Abb. 7.3 Delegation mit Kerberos

seiner Hilfe einen gültigen Authenticator für die Anfrage bei T erzeugen. Somit ist das ursprüngliche Ziel erreicht und S kann tatsächlich eine Anfrage an T im Namen von Alice richten (siehe auch Abb. 7.3).

Was aber bedeutet dieses Vorgehen aus sicherheitstechnischer Sicht? Die Weitergabe des TGT von Alice und des dazu gehörigen Session Keys bedeutet für den Server S , dass er Alice impersonifizieren kann, so lange das TGT gültig ist: Weder der TGS noch der Zielserver T können wissen, dass sie in Wirklichkeit gar nicht mit Alice sprechen, sondern mit S . Die Vergabe von Zugriffsrechten erfolgt unter der Annahme, Alice stelle den Request. Im Normalfall macht es für den Zielserver T keinen Unterschied, über welche Maschinen und Standorte (evtl. im Ausland) sich ein Nutzer letztlich einloggt. Dies kann aber in hochsensiblen Anwendungsbereichen wie im Finanzwesen oder bei militärischer Kommunikation durchaus eine Rolle spielen. Ein böswilliger Server S könnte Alices TGT dazu missbrauchen, weitere Dienste im Namen von Alice in Anspruch zu nehmen, über die hinweg, für die er von Alice beauftragt wurde. Außerdem könnte S Alices TGT und Sitzungsschlüssel an weitere Server weitergeben, ohne dass Alice dies kontrollieren kann. Es liegt deshalb im Interesse aller Nutzer (und letztlich der Sicherheit des gesamten Systems), die Lebensdauer von TGTs möglichst kurz zu halten. Zudem enthalten TGTs nach KerberosV5 ein „Forwardable“-Flag, das vom Kerberos-Server nur dann gesetzt wird, wenn die beantragende Instanz keine besonders sensible Rolle wie etwa „Administrator“ innehat. Ist das Flag nicht gesetzt, wenn S im Namen von Alice Tickets beim TGS beantragt, kann der TGS aufgrund der Diskrepanz von Alices IP-Adresse und der derjenigen von S das Ausstellen eines Service Tickets verweigern (für einen böswilligen Server S wäre es aber natürlich kein Problem, seine IP-Adresse zu fälschen und seinen Request unter der IP-Adresse von Alice zu stellen). Außerdem ist die Weitergabe von TGTs nur an

solche Server möglich, die in der betreffenden Domäne als „Trusted“ geführt werden.

Eine andere Möglichkeit der Delegation besteht darin, dass Alice selbst ein Service Ticket für den Zielserver T beantragt und dieses dann zusammen mit dem Sitzungsschlüssel K_{AT} an Server S weitergibt. Diese Variante ist etwas sicherer, da S nun nicht mehr Alices TGT missbrauchen kann, um weitere Dienste in Anspruch zu nehmen, aber natürlich auch deutlich aufwändiger und unkomfortabler für Alice, da diese schon im Voraus wissen muss, welche Dienste auf welchen Servern sie in Anspruch nehmen möchte. Das Grundproblem bleibt aber bestehen: Der Mechanismus zur Delegation in Kerberos besteht im Wesentlichen darin, dass Alice ein Geheimnis (nämlich K_{AB} bzw. K_{AT}) an eine andere Instanz weitergibt. Trotzdem ist dieser Mechanismus deutlich besser als eine einfache Passwort-Weitergabe:

- Das TGT hat eine begrenzte Lebensdauer (und die Lebensdauer ist nur auf Wunsch des Benutzers verlängerbar).
- Ein Passwort erlaubt den Empfängern, beliebig oft eine initiale Authentisierung des Besitzers vorzunehmen. Das TGT und der dazu gehörige Sitzungsschlüssel K_{AB} stellen aber keine initiale Authentisierung dar, denn sie sind an einen Ticket Granting Server B gebunden.

Schaut man sich die anderen obigen Einwände gegen die TGT Weitergabe an, so stellt man fest, dass sich die Kritik hauptsächlich gegen das Fehlen weiterer Metadaten zur Authentisierung richtet. Zum Beispiel wäre ein Mechanismus wünschenswert, bei dem ein Nutzer für ein TGT angeben kann, dass es nur für bestimmte Zwecke verwendet werden darf.

Dem Wunsch nach mehr Metadaten kommt der neue Standard *Security Assertion Markup Language* (SAML) entgegen, der interoperable, standardisierte Aussagen über Rechte ermöglicht und im Bereich der Webservices und Grids Verbreitung findet (siehe Kap. 12).

7.2.5 Cross-Domain-Authentisierung

Heutzutage stellen Firmen immer weniger geschlossene Inseln dar. Durch Extraneets, VPNs etc. werden Verbindungen mit verschiedenen Partnerfirmen hergestellt. Die Firmen sind häufig in Töchter und Zweigstellen unterteilt, die möglicherweise organisatorische Besonderheiten haben, in unterschiedlichen juristischen Gebieten angesiedelt sind und deshalb ebenfalls eigene Domänen bilden. Unterstützen alle diese Domänen Kerberos, können sie hierarchisch oder horizontal angeordnet werden, und eine Domänen-übergreifende Authentisierung via Kerberos wird möglich: Dazu muss das Geheimnis des Ticket Granting Servers TGS_A von Domäne A im Authentication Server von Domäne B registriert werden und umgekehrt. Möchte nun ein Nutzer in Domäne B sich in Domäne A authentisieren, besorgt er

sich bei seinem Authentisierungsserver ein TGT für TGS_A und kann dann ganz einfach bei TGS_A Tickets für die gewünschten Services in Domäne A beantragen.

Wirklich ganz einfach? Im Prinzip ja, dennoch bedeutet die Unterstützung von Cross-Domain-Authentisierung mit Kerberos einen erheblichen administrativen Aufwand, da alle Identitäten in der einen Domäne auf Identitäten in der anderen Domäne abgebildet werden müssen. Ticket Granting Server der einen Domäne müssen mit Authentication Servern der anderen Domäne auf sichere Weise Geheimnisse austauschen und in regelmäßigen Abständen erneuern. Zudem steckt in dem beschriebenen Mechanismus natürlich die implizite Annahme, dass TGS_A dem Authentication Server der anderen Domäne bei seiner initialen Authentisierung der Nutzer vertrauen muss – beides Aspekte, die es fraglich erscheinen lassen, ob ein solches Verfahren wirklich gut geeignet ist für die flexible und spontane Kommunikation in virtuellen Organisationen oder im E-Business am Internet. Das Thema der Cross-Domain-Authentisierung wird im Kapitel „Föderative Sicherheit“ weiter vertieft.

7.2.6 Kerberos Erweiterungen

Einige der oben angesprochenen Kritikpunkte werden mittlerweile durch Erweiterungen zum ursprünglichen Kerberos-Protokoll adressiert. So haben wir oben die Problematik diskutiert, dass die initiale Authentifikation eines Nutzers beim Authentication Server über ein Passwort erfolgt und somit für eine Wörterbuch-Attacke anfällig ist. Auf Microsoft Windows 2000 basierende verteilte Systeme umgehen dieses Problem, da hier die initiale Anmeldung über ein Public-Key-Verfahren erfolgen kann. Dabei schickt der Nutzer in seinem Authentication Request spezielle Preauthentication Data an den Authentication Server, bestehend aus seinem X.509 Zertifikat, dem gewünschten Algorithmus zum Schlüsselaustausch, einem Zeitstempel und einer digitalen Signatur über alle diese Daten. Der Authentication Server verifiziert die Gültigkeit der Preauthentication Data und antwortet wie üblich mit einem TGT und einem verschlüsselten Sitzungsschlüssel für den Client. Diese Antwort wird allerdings vom Authentication Server zusätzlich verschlüsselt, entweder mit einem symmetrischen Schlüssel, der über den Public Key des Client verschlüsselt verschickt wird, oder aber mit einem Schlüssel, der sich aus einem Diffie-Hellman Schlüsseltausch Protokoll ergibt. Welches der Verfahren zum Einsatz kommt, ergibt sich aus den Preauthentication Data des Nutzers. Mittlerweile hat Microsoft auch einen Internet-Draft zu dieser Erweiterung veröffentlicht ([Zhu]).

Ein weiterer Hauptkritikpunkt liegt in der nicht granularisierbaren Delegation in Kerberos. Ein delegiertes TGT ermöglicht dem Empfänger den Zugang zu allen Diensten, zu deren Nutzung der Inhaber des TGT berechtigt ist. Die einzige Möglichkeit der Einschränkung besteht in der begrenzten Lebensdauer des TGT. Die im Rahmen eines europäischen Forschungsprojekts entstandene Technologie SESAME (Secure European System for Applications in a Multi-vendor Environ-

ment, [SESAME]) setzt auf der Kerberos-Architektur mit einem Authentication Server und einem Ticket Granting Server (in SESAME Privilege Attribute Server genannt) auf, ersetzt die Service Tickets aber durch Privilege Attribute Certificates (PACs), in denen die Zugriffsrechte des Nutzers genau spezifiziert sind. Die PACs sind delegierbar, wobei der Nutzer genau steuern kann, welche der darin genannten Rechte er an welche Ressource delegieren möchte. Allerdings wird SESAME im Moment nicht besonders häufig eingesetzt, was wahrscheinlich an der aufwändigen Infrastruktur liegt, die zur Ausgabe, Verwaltung und Verifikation der PACs benötigt wird.

7.2.7 Microsoft Passport

Passport wurde lange Zeit von Microsoft als Web-basierter Anmelde- und Bezahl-dienst mit einer zentralen Autorität für das Internet vertrieben. Es basiert auf den oben beschriebenen Kerberos-Tickets, die in Form von Cookies vom Passport-Server (der sowohl Kerberos Server als auch Ticket Granting Server ist) an die Clients übertragen werden. Das Kerberos-Protokoll ist dabei so modifiziert worden, dass auf Client-Seite ein gewöhnlicher Browser als Software ausreicht.

Bei einem ersten Anmeldeversuch bei einem Service A während einer Browser-Session wird der Kunde von A per http Redirect an den Passport-Server umgeleitet. Dem Passport-Server ist somit der gewünschte Service bekannt. Der Passport-Server präsentiert dem Kunden nun eine SignIn-Form, wo der Kunde seine UserID und Passwort einträgt. Die SignIn-Form wird per SSL (siehe unten) geschützt an den Passport-Server zurück übermittelt. Bei erfolgreicher Authentifikation setzt der Passport-Server beim Kunden einen Ticket Granting Cookie und einen Ticket Cookie für den gewünschten Service A. Danach wird der Kunde per http ReDirect an seinen Zielservice zurück verwiesen. Der Kundenbrowser kann nun sein Ticket Cookie an den Zielserver übermitteln. Dieser validiert das Ticket und gewährt dann dem Kunden Zugang. Bei weiteren Anmeldungen wird wiederum das Ticket Cookie verwendet, so lange, bis es abläuft. Dazu ist eine weitere geschützte Verbindung (aber keine erneute Authentifikation mit UserID und Passwort) zum Passport-Server nötig. Diesmal präsentiert der Kundenbrowser sein Ticket Granting Cookie und erhält, falls es noch gültig ist, ein neues Ticket Cookie für den gewünschten Service.

Derselbe Mechanismus wie beim gerade beschriebenen „Ticket Refreshing“ wird auch verwendet, falls sich der Kunde nun bei einem anderen Service B anmelden will.

Im Unterschied zu Kerberos werden in Passport keine Authenticators verwendet, da man nicht davon ausgehen kann, dass im System aus Client, beteiligten Internet-Diensten und Passport-Server eine systemweite, vertrauenswürdige Zeit existiert. Zudem würde die Notwendigkeit, Authenticators berechnen zu müssen, eine Modifikation der Browser-Software auf Kundenseite erfordern.

Wie man sieht, hat der Passport-Server eine zentrale Rolle im System inne: Er kennt alle Kunden, ihre Passwörter und ihre Beziehungen zu den Zielservices. Als Betreiber eines Service gibt man somit praktisch seine gesamten Kundendaten an den Passport Server weiter und muss darauf vertrauen, dass mit den Kundendaten verantwortungsvoll umgegangen wird. Auch aus Kundensicht stellen sich natürlich erhebliche Bedenken in Hinsicht auf den Datenschutz. Wir werden weiter unten in Kapitel 12 andere Single-Sign-On-Systeme kennen lernen, die weniger auf eine zentrale Instanz im System setzen und diese Schwächen nicht besitzen. Aus diesen Gründen (und nach einer schwerwiegenden Sicherheitspanne im Datenbanksystem des Passport-Servers¹⁶) verabschiedeten sich immer mehr große Dienstbetreiber von Passport als Authentifizierungssystem, als letzter ebay im Jahr 2005. Allerdings plant Microsoft offenbar, Passport unter anderem Namen (Windows Live ID) zukünftig für seine geplanten Online-Dienste Windows Live und Office Live einzusetzen.

7.3 SSL/TLS

Das *Secure Sockets Layer (SSL)*-Protokoll ist das im Internet bei weitem am häufigsten eingesetzte Sicherheitsprotokoll. Heutzutage kann es sich kein zu kommerziellen Zwecken betriebener Web Server mehr leisten, SSL nicht zu unterstützen, und auf Client-Seite sieht es ganz ähnlich aus: Praktisch jeder heutzutage erhältliche Webbrower unterstützt SSL bzw. das von der Internet Engineering Task Force (IETF) verabschiedete Standardprotokoll *Transport Layer Security (TLS)*. Im Gegensatz zu TLS wurde SSL nicht von einer Standardisierungsorganisation entwickelt, sondern von der Firma Netscape, die es seit März 1995 als SSLv2¹⁷ in ihrem Webbrower Netscape Navigator einsetzte – eine weise Entscheidung, die sowohl zur Popularität des Navigators als auch zu der von SSL beitrug. Nachdem sich SSLv3 als Quasi-Standard für Web-Security etabliert hatte, nahm sich die IETF 1996 der Sache an, um SSLv3 mit einigen konkurrierenden, aber ähnlichen Ansätzen zu vereinen und verabschiedete nach einigen Streitigkeiten 1999 den auf SSLv3 basierenden [RFC 2246], in dem TLS spezifiziert wird. Obwohl es gewisse Unterschiede zwischen der aktuellen SSL-Version 3.0 und TLS gibt (vgl. hierzu [Schw], Kap. 4.5), sind diese Unterschiede gerade aus Sicht des Nutzers bzw. Programmierers doch so gering, dass es gerechtfertigt erscheint, im Folgenden zwischen SSL und TLS nicht weiter zu unterscheiden.

SSL ist ein Protokoll, das einen gesicherten Kanal zwischen zwei Maschinen (genauer: zwischen zwei Prozessen) herstellt. Besteht ein solcher gesicherter Kanal zwischen Client und Server, wird dies im Browser durch ein geschlossenes Vorhängeschloss angezeigt – auch dies eine Erfindung von Netscape, die sich mittlerweile allgemein durchgesetzt hat und die eines der seltenen Beispiele für

¹⁶ siehe <http://www.heise.de/newsticker/meldung/36693>.

¹⁷ SSLv1 besaß gravierende Sicherheitsmängel und wurde niemals kommerziell eingesetzt.



Abb. 7.4 Mit SSL gesicherte Verbindung. Man beachte das Schloss in der rechten unteren Ecke des Browser und die mit https beginnende URL

ein gelungenes User Interface eines Sicherheitsprodukts darstellt. Gesichert bedeutet in diesem Zusammenhang, dass die transferierten Daten verschlüsselt und mit einem Integritätsschutz versehen sind. Darüber hinaus ist es bei einem gesicherten Kanal wichtig zu wissen, wer eigentlich die Endpunkte des Kanals sind, mit anderen Worten: Client und Server sollten sich authentifizieren, bevor sensible Daten durch den gesicherten Kanal geschickt werden. Auch hierfür gibt es in SSL geeignete Mechanismen. Im Folgenden werden wir uns einen groben Überblick darüber verschaffen, wie dies geschieht und auf weiterführende Fragen wie Sicherheit und Effizienz von SSL eingehen.

7.3.1 Aufbau von SSL

Der von SSL zur Verfügung gestellte, gesicherte Kanal ist transparent, das heißt, im Prinzip können beliebige Anwendungsdaten durch ihn geschickt werden. SSL benötigt aber ein Transportprotokoll, das Pakete zuverlässig und mit einer Kontrolle der Reihenfolge transportieren kann. Im Internet-Umfeld erledigt diese Aufgaben das Transmission Control Protocol TCP. Unterhalb von SSL im Protokollstapel liegt also TCP, darüber kann praktisch jedes Anwendungsprotokoll liegen, also z. B. http, ftp oder SMTP. Ebenso kann es zum Absichern von Remote Object Access via RMI, CORBA oder IIOP eingesetzt werden (siehe Kapitel „Middleware Security“). Aufgrund dieser Flexibilität wird SSL auch zunehmend außerhalb des BusinessToCustomer-Umfelds eingesetzt, etwa um in einem Intranet gesichert Daten von einer Datenbank zu einem Application Server übertragen zu können. Der Hauptanwendungsfall ist und bleibt jedoch die Absicherung von http.

Üblicherweise beginnt ein Uniform Resource Locator (URL) mit http, das heißt, es wird das Protokoll, das zum Holen der gewünschten Ressource benötigt

wird, angegeben. Ist die Ressource jedoch mit SSL geschützt, wird also http über SSL übertragen, so beginnt die URL mit https. Der Client kann also bereits an der URL erkennen, ob der Datentransfer zum und vom Server gesichert sein wird. Unklar hingegen bleibt zunächst, welche Art von Sicherheit eingesetzt werden soll: Welche Algorithmen mit welchen Schlüsselgrößen werden zum Einsatz kommen? Wie sollen sich Client und Server authentifizieren, und wie sollen sie sich auf geeignete Schlüssel einigen? Die Klärung dieser Fragen übernimmt das SSL Handshake Protocol, ein Teilprotokoll von SSL. Nach Abschluss dieses Protokolls können Client und Server gesichert miteinander kommunizieren. Die dazu nötigen Arbeiten, das heißt das Fragmentieren der Anwendungsdaten, das Bilden eines MAC zur Integritätssicherung und schließlich die eigentliche Verschlüsselung, erledigt ein weiteres SSL Teilprotokoll, das SSL Record Protocol, das unmittelbar auf TCP aufsetzt. Treten Fehler auf, etwa wenn ein Server ein abgelaufenes Zertifikat präsentiert (`certificate_expired`) oder das Zertifikat von einer nicht als vertrauenswürdig eingestuften bzw. unbekannten CA stammt (`no_certificate` bzw. `unknown_ca` in TLS), so werden diese vom Alert Protocol signalisiert, wie der unten stehende Screenshot zeigt.

Die einzige Ausnahme unter den Alerts bildet der `close_notify` Alert, der keine Fehlermeldung darstellt, sondern mit dem ein Sender signalisiert, dass er nun alle Daten gesendet hat und die Verbindung zu schließen gedenkt. Wie das Handshake Protocol wird auch das Alert Protocol über das Record Protocol versendet; ist der Handshake noch nicht abgeschlossen, haben sich also Client und



Abb. 7.5 Fehlermeldung des Mozilla-Browsers, ausgelöst durch den TLS Alert `unknown_ca`

Anwendungs-protokoll	SSL Handshake	SSL ChangeCipherSpec	SSL Alert Protocol
SSL Record Layer			
TCP			
IP			

Abb. 7.6 Lage von SSL im Internet-Protokollstapel

Server noch nicht auf gemeinsame Geheimnisse geeinigt, werden die Daten im Klartext durch das Record Protocol verschickt.

Als letztes Teilprotokoll besteht das ChangeCipherSpec Protokoll aus einer einzigen festen Nachricht, die als Trigger dient, von Erhalt der Nachricht an die vorher ausgehandelten Krypto-Parameter zu nutzen. Da die ChangeCipherSpec Nachricht im Rahmen des Handshakes „eingeschoben“ wird, kann man sich fragen, warum diese Nachricht als eigenes Protokoll firmiert. Der Grund hierfür besteht darin, dass mehrere Nachrichten des Handshake vom Record Protocol zu einem einzigen Paket zusammengefasst werden können. Wäre ChangeCipherSpec eine solche Einzelnachricht im Rahmen des Handshake, so könnten weitere Nachrichten vor und nach ChangeCipherSpec Teil eines Record-Pakets sein, obwohl der Trigger ChangeCipherSpec ja gerade bewirken soll, dass im Record Protocol auf einen neuen Security Kontext umgeschaltet werden soll.

Bevor wir auf den Handshake genauer eingehen, verdeutlicht Abb. 7.6 noch einmal die Lage der SSL-Protokollfamilie im (Internet-)Protokollstapel zwischen Anwendungsprotokoll und TCP.

Wie man sieht, kann der Record Layer vier verschiedene Protokolltypen transportieren. Der Typ des transportierten Protokolls wird in dem Datenfeld Content Type im Rahmen des Headers eines SSL Record Pakets angezeigt, wobei zwischen den verschiedenen Anwendungsprotokollen nicht mehr weiter unterschieden wird.

7.3.2 Handshake

Bevor irgendwelche Anwendungsdaten über SSL versandt werden können, muss das Handshake Protokoll durchgeführt werden. Es ist der komplexeste Teil von SSL. Dies ist nicht weiter verwunderlich, hat es doch anspruchsvolle Aufgaben zu erfüllen: Es muss zwischen zwei Rechnern, die womöglich noch niemals miteinander Kontakt hatten, einen Satz von zu verwendenden Algorithmen vermitteln, es müssen Schlüssel vereinbart werden und schließlich müssen bzw. können sich Client und Server gegenseitig authentifizieren.

In einem ersten Schritt sendet der Client dem Server die von ihm unterstützte Version von SSL bzw. TLS und eine Liste der von ihm unterstützten Algorith-

men. Die Algorithmen umfassen Verschlüsselungs-, Hash-, Schlüsselaustausch- und Signaturalgorithmen. Ein solcher Satz von Algorithmen wird als Ciphersuite bezeichnet und kann im Mozilla Browser vom User spezifiziert werden, wie bereits im Kapitel über Browsereinstellungen beim Kunden erwähnt. Zusätzlich zu dieser Liste von Ciphersuites übermittelt der Client auch eine Zufallszahl, die später für die Erzeugung von Schlüsselmaterial genutzt wird. Diese Nachricht wird auch als `ClientHello` bezeichnet. Der Server antwortet darauf mit einem `ServerHello`: Sofern der Server die vom Client angegebene SSL Version unterstützt, wählt er darin aus der vom Client vorgeschlagenen Liste eine Ciphersuite aus. Auch der Server gibt eine Zufallszahl vor, so dass beide Kommunikationsparteien zur Schlüsselerzeugung Input beitragen. Als letzte, wichtige Information findet sich im `ServerHello` eine SessionID, die dazu dient, im weiteren Verlauf der Verbindung eine Session wieder aufnehmen zu können, ohne dazu den aufwändigen Handshake noch einmal durchführen zu müssen (Session Resumption).

Damit sind die wesentlichen Randbedingungen für den weiteren Ablauf des Handshakes geklärt, obwohl man sich vor Augen halten sollte, dass die bisherige Aushandelphase über einen ungesicherten Kanal abgelaufen ist. Der weitere Ablauf hängt nun wesentlich davon ab, welches Schlüsseltauschverfahren von den beiden Parteien ausgewählt worden ist. Am gebräuchlichsten ist die Methode, bei der der Server dem Client seinen Public Key in einem X.509-Zertifikat mitteilt. Dies geschieht in der `Certificate` Nachricht, gefolgt von der `ServerHelloDone` Nachricht. Um dem Client die Möglichkeit zu geben, die Signatur unter dem Server-Zertifikat unmittelbar verifizieren zu können, kann die `Certificate` Nachricht in SSLv3 bzw. TLS noch weitere Zertifikate der ausstellenden CAs enthalten. Hat sich der Client von der Authentizität des Public Key des Servers überzeugt, generiert er eine 48 Byte lange Zufallszahl: Das `PreMasterSecret`. Das `PreMasterSecret` wird vom Client mit dem Public Key des Servers verschlüsselt und schickt das Ergebnis an den Server. Das verschlüsselte `PreMasterSecret` bildet die Nachricht `ClientKeyExchange`. Kann der Server das `PreMasterSecret` entschlüsseln, bedeutet dies zunächst, dass Client und Server über ein gemeinsames Geheimnis verfügen, das sie im Folgenden zum Ableiten weiteren Schlüsselmaterials benutzen können. Zusätzlich aber beweist der Server durch die gelungene Entschlüsselung, dass er den Private Key zu dem im Zertifikat angegebenen Public Key kennt, und somit authentifiziert sich der Server gegenüber dem Client.

Nachdem Client und Server aus dem `PreMasterSecret` mit Hilfe einer aus den Hashfunktionen MD5 und SHA-1 zusammengesetzten Funktion (diese ist für SSLv3 und TLS unterschiedlich) einen weiteren Schlüssel, das `MasterSecret` abgeleitet haben, kann der Kanal mit dessen Hilfe abgesichert werden. Das bereits erwähnte `ChangeCipherSpec` Protokoll gibt das Signal hierfür. Danach ist es Zeit, die über ungesicherte Kanäle ausgetauschten `ClientHello` und `ServerHello`-Nachrichten nachträglich durch einen MAC abzusichern. Dieser MAC bildet die beiden `Finished`-Nachrichten vom Client zu Server und umgekehrt am Ende des Handshakes. Würden sie fehlen (und sie fehlen in SSLv2

tatsächlich), könnte ein Angreifer die am Anfang des Handshake ausgetauschten Listen von Ciphersuites verändern und die unterstützten Algorithmen mutwillig abschwächen (Ciphersuite Rollback Attack). Durch die MAC-Bildung im Rahmen der Finished-Nachrichten wird ein solcher Angriff verhindert. Nach den beiden Finished-Nachrichten ist der Handshake (in seiner einfachsten Form) abgeschlossen. Abbildung 7.7 zeigt noch einmal den beschriebenen zeitlichen Ablauf.

Oben haben wir die gebräuchlichste Version des SSL-Handshakes behandelt. Aufmerksamen Lesern wird sicher nicht entgangen sein, dass noch etwas Wesentliches fehlt, nämlich die Authentifizierung des Client. Sie fehlt mit gutem Grund, da der Client, um sich ebenfalls zu authentifizieren, ebenfalls ein X.509-Zertifikat benötigen würde. Zumindest im E-Commerce Umfeld möchte man aber den Client nicht damit belästigen, ein Zertifikat zu beschaffen und managen zu müssen, zumal dies auch nicht immer kostenfrei ist. Zudem entfallen damit für den Client die aufwändigen Operationen mit dem eigenen Secret Key, was den Ablauf des Handshake beschleunigt. Wird SSL hingegen innerhalb eines Intranet eingesetzt, kann eine gegenseitige Authentifikation von Client und Server durchaus sinnvoll sein. Gehen wir weiterhin davon aus, dass beide Partner über Zertifikate verfügen, so erweitert sich der oben beschriebene Handshake um eine CertificateRequest-Nachricht, die der Server nach dem Senden des eigenen Zertifikats an den Client schickt. Die clientseitige Authentifikation wird also immer durch den Server angestoßen – der Client hat keine Möglichkeit, dies von sich aus anzubieten. Nach dem ServerHelloDone schickt der Client sein eigenes Zertifikat; an der ClientKeyExchange-Nachricht und damit an der Erzeugung des gemeinsamen Geheimnisses ändert sich nichts. Die Authenti-

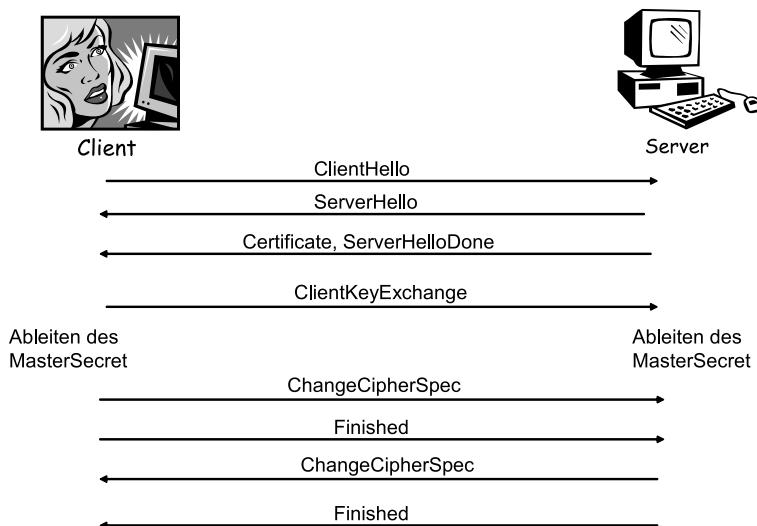


Abb. 7.7 Kurzer SSL Handshake

fikation des Client erfolgt über die darauf folgende CertificateVerify Nachricht, die eine digitale Signatur über die bisherigen Handshake-Messages darstellt.

7.3.3 Ciphersuites

SSL bietet dem Client und dem Server die Möglichkeit, aus unterschiedlichen Krypto-Algorithmen auszuwählen. Eine vollständige Konfiguration wird als Ciphersuite bezeichnet und beinhaltet Algorithmen zum Schlüsseltausch, zur Digitalen Signatur, zum Verschlüsseln durch den SSL Record Layer sowie zur MAC-Bildung. Diese Optionen unterscheiden sich sowohl hinsichtlich ihrer Sicherheit als auch hinsichtlich ihrer Performance und sollen hier kurz besprochen werden.

Im folgenden Screenshot finden wir eine Auswahl von Ciphersuites, wie sie vom Mozilla-Browser angeboten wird. Allen dort angezeigten Ciphersuites ist gemeinsam, dass sie auf RSA als Schlüsselaustauschmechanismus und zur digitalen Signatur setzen. Bei den angebotenen Verschlüsselungsalgorithmen kann sich der User zwischen RC4 und DES mit unterschiedlichen Schlüssellängen entschei-

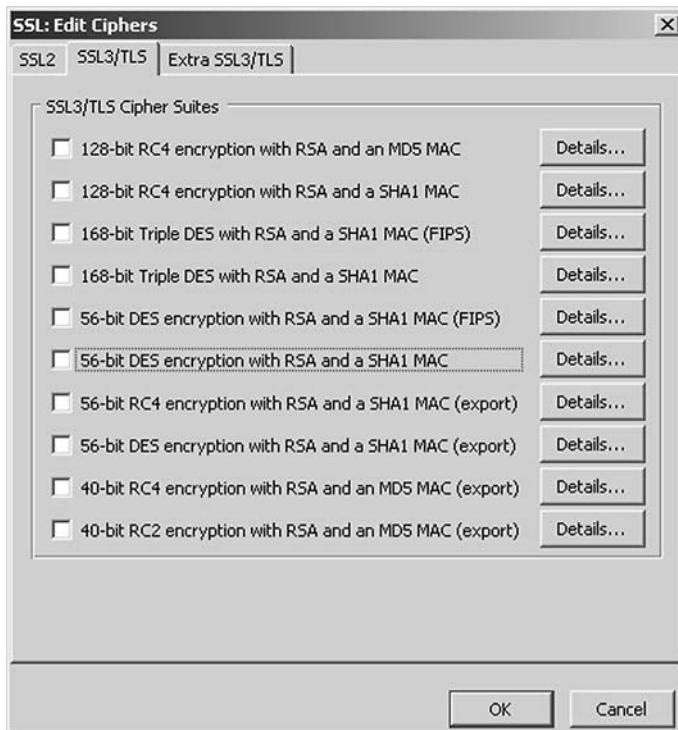


Abb. 7.8 SSLv3 CipherSuites

den. Die Optionen mit 40 bzw. 56 Bit Schlüssellänge sind mittlerweile definitiv als unsicher aufgrund der zu geringen Schlüssellänge anzusehen und sollten nicht ausgewählt werden (und SSL-Server sollten so konfiguriert sein, dass entsprechende Ciphersuites nicht akzeptiert werden). Weniger bekannt ist die Tatsache, dass bei den mit „Export“ gekennzeichneten Ciphersuites auch die verwendeten RSA-Keys mit 512 Bit deutlich zu kurz sind. Die Optionen 128-Bit RC4 und 168-Bit Triple DES weisen in etwa das gleiche Sicherheitsniveau auf, zumal der Triple DES nur eine effektive Schlüssellänge von zweimal $56 = 112$ Bit aufweist (ein spezieller Angriff auf mehrfach hintereinander ausgeführte Chiffren, der so genannte *Meet-in-the-Middle*-Angriff, sorgt für diese Reduktion der effektiven Schlüssellänge). Die weitere Unterscheidung zwischen MD5 und SHA-1 als Hashfunktion ist trotz der mittlerweile aufgekommenen Sicherheitsbedenken bei MD5 relativ unerheblich, da MD5 in diesem Zusammenhang nur zur MAC-Bildung eingesetzt wird, bei dem ein (für den Angreifer unbekanntes) Geheimnis mitgehasht wird. Allerdings wird MD5 wegen der erwähnten Probleme in neueren Anwendungen nach und nach abgelöst, so dass allein aus Kompatibilitätsgründen SHA-1 als bessere Wahl erscheint.

Ein Klick auf den Reiter „Extra SSL3/TLS“ fördert weitere interessante Optionen zutage: Nun taucht auch DHE als Option für den Schlüsselaustausch auf. DHE steht für „Ephemeral Diffie-Hellman“ (s. Abb. 7.9). Hierbei wird der Schlüssel über das auf dem diskreten Logarithmus-Problem beruhende Diffie-Hellman Schlüsselaustausch-Protokoll vereinbart. Bei diesem Protokoll tragen beide Kommunikationspartner, also Client und Server, gleichberechtigt mit ihren KeyExchange-Nachrichten zur Bildung des PreMasterSecret bei. Der kurze SSL-Handshake aus Abb. 7.7 erweitert sich deshalb um eine ServerKeyExchange-Nachricht vor dem ServerHelloDone. *Ephemeral* bedeutet hierbei, dass für jeden SSL-Handshake von Client und Server ein neuer Diffie-Hellman Public Key erzeugt wird, was eine interessante Eigenschaft des Protokolls zur Folge hat, auf die wir gleich noch eingehen werden. Zunächst aber zurück zu den angezeigten Optionen: Wenn der Schlüssel über DHE ausgetauscht wird, was haben dann noch RSA bzw. DSA in den entsprechenden Ciphersuites zu suchen? Die Public Keys beim DHE-Verfahren müssen wie beim Schlüsselaustausch via RSA authentifiziert sein, andernfalls besteht die Gefahr eines Man-in-the-Middle Angriffs, bei dem ein Angreifer die Public Keys der Kommunikationspartner durch seinen eigenen ersetzt. Die Authentifizierung der kurzlebigen „ephemeral“ Diffie-Hellman Schlüssel besteht darin, dass sie vom Sender mit einem langlebigen Signaturschlüssel signiert werden, und RSA bzw. DSA bezeichnet den dazu verwendeten Algorithmus. Nun können wir auch verstehen, worin die interessante Eigenschaft des ephemeral Diffie-Hellman Protokolls besteht: Geht nämlich besagter langlebiger Signaturschlüssel verloren, so kann der Angreifer, der diesen Schlüssel erbeutet hat, sich ab diesem Zeitpunkt als Inhaber dieses Schlüssels ausgeben und hat somit als Man-in-the-Middle die gesamte Kommunikation unter seiner Kontrolle. Dies gilt aber nur *ab* dem Zeitpunkt des Schlüsselkompromisses, die Kommunikation *vor* diesem Zeitpunkt bleibt für den Angreifer unlesbar, da er die Private Keys zu den verwendeten kurzlebigen Public Keys nicht kennt. Somit ist nur die Kommunikation zwischen

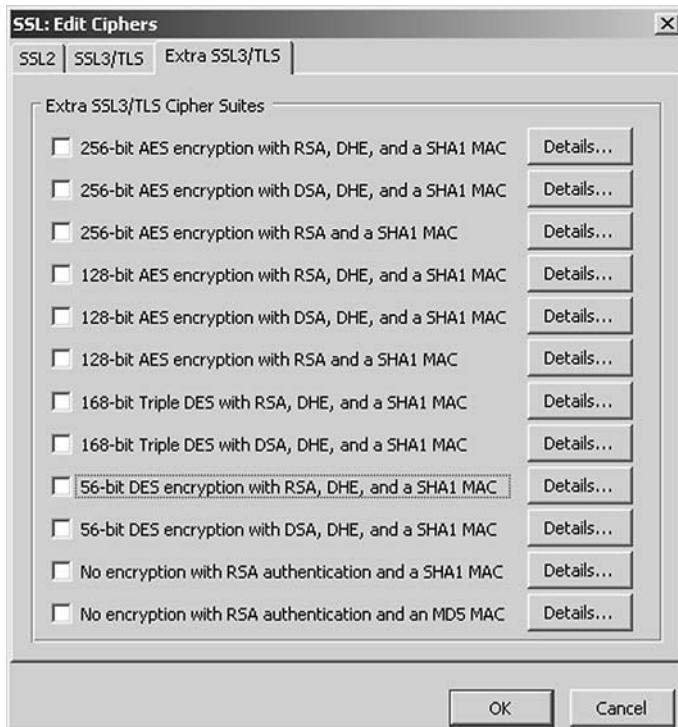


Abb. 7.9 Zusätzliche SSLv3 Ciphersuites

dem Verlust des Signaturschlüssels und dem Zeitpunkt, an dem der Verlust bemerkt wird, gefährdet – ein (hoffentlich) überschaubarer Zeitraum. Diese Eigenschaft des DHE-Protokolls wird auch als *Perfect Forward Secrecy*¹⁸ bezeichnet.

Als zusätzliche Verschlüsselungsoption wird in den zusätzlichen Ciphersuites 128-Bit bzw. 256-Bit AES Verschlüsselung angeboten. Diese Optionen sollen unbedingt aktiviert werden, da der moderne AES-Algorithmus bei einem vergleichbaren Sicherheitsniveau bei weitem schneller als der Triple-DES ist.

Im Umfeld verteilter Systeme gewinnen weitere Ciphersuites, die in den Screenshots nicht auftauchen, stark an Bedeutung: An erster Stelle ist der in [RFC 2712] spezifizierte Schlüsseltausch via Kerberos zu nennen, der mit symmetrischer Kryptografie auskommt, aber natürlich, wie oben beschrieben, einen zentralen Kerberos Server und einen Ticket Granting Server benötigt. Bevor der Client den SSL Handshake mit dem gewünschten Server anstoßen kann, besorgt er sich ein passendes Ticket vom Ticket Granting Server. Der Handshake läuft dann im

¹⁸ Natürlich ließe sich diese Eigenschaft auch mit temporären RSA-Keys erreichen. Allerdings sind nur die in den mit „Export“ gekennzeichneten Ciphersuites verwendeten kurzen RSA-Keys temporär, alle anderen sind fest, werden also immer wieder zum Schlüsseltausch verwendet. Bis auf den Verlust der Perfect Forward Secrecy tut dies der Sicherheit der Schlüsselvereinbarung jedoch keinen Abbruch.

Prinzip so wie in Abb. 7.4 ab, jedoch nimmt die ClientKeyExchange-Nachricht eine andere Form an: Sie enthält neben dem Ticket für den Server das mit dem im Ticket enthaltenen Session Key verschlüsselte PreMasterSecret. Kann der Server das Ticket entschlüsseln, kennt er den Session Key und kann damit auch das PreMasterSecret entschlüsseln. Darauf hinaus hat die TLS Arbeitsgruppe bei der IETF einen Internet Draft zur Authentifikation über SSL mit so genannten PreSharedKeys veröffentlicht, der sowohl eine gegenseitige Authentifikation ohne Zertifikate als auch eine clientseitige Authentifikation über ein Passwort nach einem normalen „kurzen“ Handshake mit serverseitiger Authentifikation über ein Zertifikat behandelt.

7.3.4 Performanzfragen

Möglicherweise sind Sie nach Lektüre des letzten Abschnitts etwas verwirrt ob der vielen Verschlüsselungsoptionen. Für welche soll man sich nun entscheiden, gerade wenn man einen SSL-Server aufsetzen will? Natürlich sollte bei der Auswahl zunächst einmal der Sicherheitsaspekt im Vordergrund stehen. Die zur Verfügung stehenden Optionen bei der Schlüsselvereinbarung und digitalen Signatur sind bei gleicher Schlüssellänge (RSA und Diffie-Hellman mindestens 1024 Bit) als gleich sicher anzusehen. Ephemeral Diffie-Hellman (DHE) bietet zusätzlich den Vorteil der Perfect Forward Secrecy. Die drei Verschlüsselungsalgorithmen RC4, Triple DES und AES bieten bei gleicher Schlüssellänge ebenfalls ein vergleichbares Sicherheitsniveau.

Serverseitig spielt jedoch auch der Performanzaspekt eine große Rolle. [Rescorla] widmet diesem Thema ein ganzes Kapitel, und wir verweisen den Leser für eine tiefergehende Diskussion auf ihn. Grundsätzlich können mit SSL gesicherte Verbindungen um Faktor 2 bis 100 langsamer sein als ungesicherte TCP-Verbindungen. Bei der Analyse dieser Performanzverluste müssen wir zwischen der Handshake-Phase und der eigentlichen Verschlüsselung der Anwendungsdaten durch den SSL Record Layer unterscheiden. Im Laufe des Handshake müssen durch den Server in den meisten Fällen aufwändige asymmetrische Krypto-Operationen mit dem private Key durchgeführt werden. Dabei erweist sich RSA insgesamt als deutlich schneller als Diffie-Hellman. Dazu kommt im Fall von DHE die Zeit, die benötigt wird, um die temporären DH Public Keys zu erzeugen. Da das Patent auf den RSA Algorithmus im September 2000 ausgelaufen ist, gibt es auch von rechtlicher Seite kein Argument gegen den Einsatz von RSA. Deshalb sollte sich, wer auf beste Performance Wert legt (und für den Perfect Forward Secrecy nicht wichtig ist) sich für RSA als Schlüsseltausch- und Signaturalgorithmus entscheiden. Noch schneller verläuft natürlich ein Handshake über Kerberos, da man hier ganz ohne asymmetrische Kryptographie auskommt – Kerberos ist aber, wie bereits erwähnt, nur innerhalb von verteilten Systemen ein Thema.

Hat man große Datenmengen zu transportieren, gewinnt die Schnelligkeit der Verschlüsselung im Record Layer gegenüber dem Aufwand für den Handshake an

Gewicht. Es stehen drei Verschlüsselungsalgorithmen zur Auswahl: RC4, Triple DES, und AES. RC4 und AES sind ungefähr gleich schnell, Triple DES um einen Faktor 10 langsamer als die beiden Alternativen. AES ist der modernste der drei Algorithmen und dürfte auch der sicherste von ihnen sein (wir sind hier etwas vorsichtig, gerade weil AES aufgrund seines geringen Alters noch nicht besonders lange analysiert werden konnte), somit erscheint AES als erste Wahl unter den Verschlüsselungsalgorithmen.

Da die Public-Key-Operationen während des Handshake der zeitaufwändigste Teil von SSL sind, liegt es nahe, sich diese bei einem erneuten Handshake möglichst häufig zu sparen, indem ein einmal ausgehandelter Schlüssel bei einer eventuellen Wiederkehr des Client zum Server wieder verwendet wird. Der Mechanismus hierzu heißt *Session Resumption*. Dabei schickt der Client in seiner ClientHello-Nachricht an den Server die SessionID mit, die der Client bei der erstmaligen Verbindungsaufnahme mit dem Server erhalten hatte. Die SessionID dient nun als Identifikator für das bereits ausgehandelte MasterSecret. Durch den Austausch der ChangeCipherSpec-Nachricht wird in der Folge auf die Nutzung dieses MasterSecret umgeschaltet (s. Abb. 7.10).

Wie nicht anders zu erwarten, führt der Session-Resumption-Mechanismus zu einer deutlichen Verbesserung der Performance. Allerdings ist er serverseitig auch mit einigen Nachteilen verbunden, da die SessionIDs und dazu gehörigen MasterSecrets von eventuell sehr vielen Clients im Cache gehalten werden müssen. Wird eine Lastverteilung auf mehrere SSL-Server eingesetzt, hat man als zusätzliches Problem, dass ein Client ein MasterSecret mit Server A vereinbaren kann, um dann bei einer Wiederkehr auf die SSL-geschützte Seite eventuell auf einen anderen Server geleitet zu werden. Somit ist ein Mechanismus zum sicheren Transport von MasterSecrets von Server zu Server erforderlich. Außerdem muss die SessionID zusammen mit der Identität des vergebenden Servers gespeichert werden. Möchte ein SSL Server aus diesen Gründen auf den Session

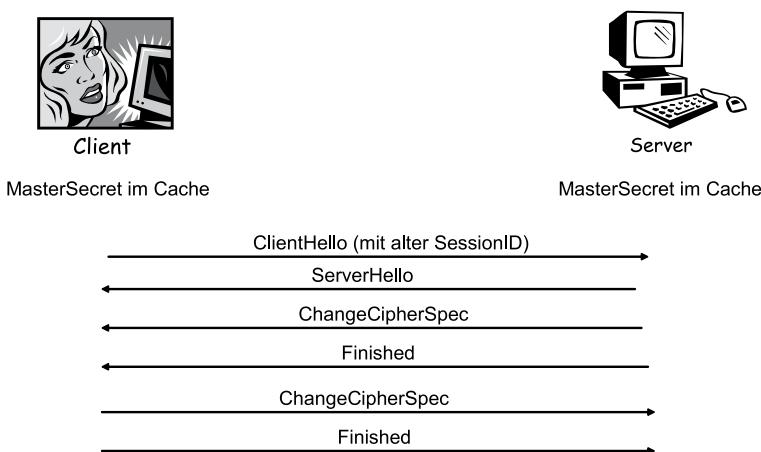


Abb. 7.10 SSL Session Resumption

Resumption Mechanismus verzichten, so vergibt er beim initialen Handshake keine SessionID¹⁹.

7.3.5 SSL Security

Die im SSL Record Layer zur Verschlüsselung und zum Integritätsschutz eingesetzten Algorithmen sind bei richtig gewählter Schlüssellänge (d. h. mindestens 112 Bit effektiv) durchweg als sicher zu bezeichnen. Somit bleiben für einen Angreifer zwei grundsätzliche Möglichkeiten: Ein Angriff auf den Handshake oder der Versuch, die vereinbarten Schlüssel an den Endpunkten der Kommunikation, also SSL Client oder Server, anzugreifen. Betrachten wir zunächst den Handshake. Der heikelste Moment ist sicher der Austausch der initialen Hello-Nachrichten, die anfangs über einen ungesicherten Kanal transportiert werden. In SSLv2 geschieht dieser Austausch, bei dem sich Client und Server auf die verwendete Ciphersuite einigen, komplett ungesichert. Ein aktiver Angreifer könnte hier eingreifen und das Angebot an Ciphersuites, das der Client dem Server bietet, willkürlich abschwächen: Eine so genannte *Ciphersuite Rollback Attack*. Konkret heißt das, dass der Angreifer dafür sorgt, dass im ClientHello nur noch die schwachen „Export“-Ciphersuites enthalten sind. Diese sind sowohl durch kurze (40 Bit) Schlüssellängen für den RC4 Algorithmus als auch durch kurze (512 Bit) temporäre RSA Schlüssel gekennzeichnet und liegen somit durchaus in der Reichweite eines hinreichend motivierten und gut ausgestatteten Angreifers. SSLv3 verhindert den Ciphersuite Rollback dadurch, dass nach Vereinbarung des MasterSecret in der Finished-Nachricht alle vorhergehenden Nachrichten inklusive Hello-Nachrichten mit einem MAC versehen werden. Eventuelle Manipulationen an den Nachrichten auf dem Transportweg werden so sichtbar und der Handshake wird gegebenenfalls abgebrochen.

Ein kleines Schlupfloch bleibt dem Angreifer jedoch noch: In seiner Hello-Nachricht gibt der Client auch die von ihm unterstützte SSL-Version an. Ein Angreifer hätte also immer noch die Möglichkeit, die Versionsinformation von v3 auf v2 abzuändern, was zur Folge hätte, dass die Finished-Nachricht im Handshake entfällt (so genannter *Version Rollback*). Zum Schutz gegen diese Attacke hat der Server zum einen die Möglichkeit, Verbindungswünsche über SSLv2 grundsätzlich abzulehnen. Ist dieses nicht gewünscht, so kann der Server einen manipulierten Verbindungsauflauf über SSLv2 daran erkennen, dass die SSL-Clients in v2 und v3 unterschiedliche Padding-Schemata bei der Verschlüsselung mit RSA verwenden. So lange aber ein Angreifer nicht auch den verwendeten RSA-Schlüssel des Servers faktorisieren kann, hat er keine Chance, auch das Padding-Schema des Clients zu manipulieren.

¹⁹ Die OpenSSL-Bibliothek enthält eine Funktionalität, mit der Anwendungen das Load Balancing über die Vergabe von SessionIDs steuern können.

Kann sich ein Angreifer mit Administratorrechten Zugriff auf Client oder Server verschaffen, kennt er auch das `MasterSecret` der laufenden SSL-Verbindungen. Da die Schlüssel sowohl für Vertraulichkeit als auch für den Integritätsschutz der Verbindung vom `MasterSecret` abgeleitet werden, liegen diese Verbindungen für den Angreifer komplett offen. Neben dem Mitlesen der Daten ist es für den Angreifer somit auch möglich, eigene SSL-Records zu fabrizieren und in die Verbindung einzuschleusen. Gelingt es dem Angreifer nicht, Administratorrechte zu erlangen, hängt es von seinen Userrechten ab, inwieweit er Zugriff auf die `MasterSecrets` laufender oder bereits abgeschlossener Verbindungen erhält. Aus diesem Grund sollten `MasterSecrets` grundsätzlich nicht permanent auf der Festplatte gespeichert werden. Anders verhält es sich natürlich mit dem Private Key des Servers, der in irgendeiner Form permanent gespeichert sein muss. Der übliche Fall ist der eines statischen RSA-Keys, das heißt, zum Schlüsselaustausch und zur Authentifikation des Servers dient dieselbe Private Key. Geht er verloren, kann der Angreifer nicht nur alle früheren und zukünftig über diesen Schlüssel durchgeföhrten SSL-Sessions mitlesen, sondern er kann auch den Server selbst impersonifizieren und sich gegenüber den Clients als legitimer Server auftreten. Bei nicht-statischem (ephemeralen) Schlüsselaustausch sind die Folgen eines Schlüsselkompromisses nicht ganz so ernst: Wie bereits erwähnt, sorgt die Eigenschaft der Perfect Forward Secrecy für den Fall, dass ein privater Signaturschlüssel verloren geht, dafür, dass nur zukünftige Sessions bis zum Austausch des Signaturschlüssels, nicht aber bereits abgeschlossene Sessions vom Angreifer gelesen werden können. Die Gefahr der Impersonifikation des Servers besteht aber nach wie vor.

Deshalb müssen an die Speicherung eines Private Key besonders hohe Sicherheitsanforderungen gestellt werden. Geschieht die Speicherung über die Browser-Software (mit dem Software Security Device, s. Abb. 7.11), wird der Schlüssel mit einem symmetrischen Verfahren verschlüsselt auf der Festplatte abgelegt. Der dazu gehörige symmetrische Schlüssel wird aus einem Passwort (oder besser einer Passphrase) abgeleitet. Besser als die Speicherung in Software ist es aber allemal, den Schlüssel auf einer PCMCIA-Karte oder einer Smart Card (so genanntes *Hardware Security Module*, HSM) zu speichern, die physikalisch vom Server selbst getrennt werden können. Diese Karten sind in der Lage, selbst die nötigen Krypto-Operationen mit dem Private Key auszuführen, so dass der Schlüssel die Karte niemals verlassen muss. Bevor die Karte ihren Dienst aufnimmt, muss sich der User, der mit ihrer Hilfe signieren oder entschlüsseln möchte, mit einer Personal Identification Number (PIN) authentifizieren. Ein Angreifer benötigt deshalb sowohl physikalischen Zugriff auf die Karte als auch die PIN, um mit dem Private Key arbeiten zu können.



Abb. 7.11 Passworteingabe beim Software Security Device

Der Schutz des Private Key durch Verschlüsselung oder externe Speicherung auf einer Smart Card bringt jedoch den Nachteil mit sich, dass beim Neustart eines Server-Systems eine Passphrase oder eine PIN durch einen menschlichen Administrator eingegeben werden muss. Die Alternative wäre ein hohes Risiko in Form des Speicherns der Passphrase/der PIN im Startup-Script des Servers, wo sie wiederum dem Zugriff eines Angreifers mit Administratorrechten ausgesetzt wäre. Für dieses Dilemma gibt es keine wirklich zufrieden stellende Lösung.

7.3.6 Zusammenfassung

SSL stellt ein gutes Beispiel für ein weithin eingesetztes und flexibles Sicherheitsprotokoll dar, das im Laufe der Zeit in einem evolutionären Prozess immer sicherer geworden ist. Darüber hinaus basiert es durchweg auf bewährten und lange veröffentlichten Kryptomodulen. Zudem ist SSL frei verfügbar: Das OpenSSL Project bietet unter openssl.org eine hochqualitative Implementation in C an, die nicht nur ein SSL-Toolkit, sondern eine ganze Krypto-Bibliothek bereitstellt (vgl. auch [Viega]). Es ist verlockend, SSL als Antwort auf die verschiedensten Sicherheitsproblematiken einzusetzen (und viele tun dies auch). Trotzdem sollte man sich ganz klar machen, was SSL bietet und was es nicht bietet: SSL bietet eine kanalbasierte Sicherheit an, als Gegenmaßnahme zu Bedrohungen, wie sie sich aus dem Internet-BedrohungsmodeLL ergeben: Zwei vertrauenswürdige Endpunkte mit einem unsicheren Kanal dazwischen. Durch den Einsatz von SSL können sich die Endpunkte authentisieren und der Kanal zwischen ihnen gesichert werden. SSL hat jedoch keine Antwort auf die Fragen, die sich ergeben, wenn wir dieses BedrohungsmodeLL verlassen: Zum Beispiel wie vertrauenswürdig die Daten sind, die von einem Client zur Verfügung gestellt werden (insbesondere, wenn auf Client-Authentifikation verzichtet wird), oder in welche Hände geheime Passwörter oder Kreditkartennummern von Kunden gelangen, wenn sie einmal den sicheren Kanal zwischen Client und Server verlassen, oder wie gut diese Daten auf dem Server geschützt sind. SSL bietet keinerlei Unterstützung für die im E-Commerce durchaus wünschenswerte Nicht-Abstrebbarkeit von Nachrichten auf der Anwendungsebene, da diese nur mit einem symmetrischen MAC geschützt werden. Wer diesen Dienst benötigt, etwa zum Auditing von Zugriffen auf Serverobjekte via CORBA, muss die Daten zunächst auf Anwendungsebene signieren und kann sie erst dann in den sicheren SSL-Tunnel schicken. Schließlich müssen auch weitergehende Dienste wie Delegation und Authorisierung auf höheren Schichten, unabhängig von SSL, implementiert werden. Das folgende Kapitel zeigt Möglichkeiten dazu auf.

Die Grenzen kanalbasierter Absicherung durch SSL werden auch im Umfeld der Enterprise- und Web-Services Security deutlich: In Enterprise Infrastrukturen gibt es aus Sicherheitsgründen keinen durchgehenden sicheren Kanal zwischen Originator und den Backend-Systemen. Stattdessen wird ein Request über eine Vielzahl beteiligter Server geleitet, die jeweils neue sichere Verbindungen zu

ihren unmittelbaren Kommunikationspartnern aufbauen. Die übertragenen Daten sind jedoch dazwischen ungeschützt und die empfangenden Server können sie bzw. den Originator meist nicht verifizieren. Im Web-Services Umfeld existieren häufig Business Prozesse mit mehreren unabhängigen Beteiligten, denen ein Originator (Client) u. U. gern unterschiedliche Teile einer Nachricht zur Verfügung stellen würde, d. h. es wäre wünschenswert, dass beteiligte Server unterschiedlich verschlüsselte Teile der Nachricht bearbeiten könnten.

In beiden Fällen – Enterprise-Infrastruktur wie flexible Multi-Party Business Prozesse – ist eine andere Form von Sicherheit nötig, die sog. *objektbasierte* Sicherheit, bei der die Nachricht selbst durch kryptografische Maßnahmen abgesichert wird. Im Alltagsleben gibt es dafür viele Beispiele, wie Verträge, Tickets oder signierte Vollmachten. In der IT-Sicherheit hingegen wird objektbasierte Sicherheit bislang nur relativ selten eingesetzt.

Kapitel 8

Authentication Frameworks

Wir klettern nun die Abstraktionsleiter eine Stufe höher: Nicht mehr konkrete Authentifikationsprotokolle, sondern abstrakte Frameworks, die das Aushandeln eines spezifischen Authentifikationsprotokolls ermöglichen, dabei die konkrete Implementation der Authentifikation aber vor dem aufrufenden Protokoll verborgen, stehen jetzt im Mittelpunkt unseres Interesses. Häufig können durch diese Frameworks nicht nur Authentifikationsdienste, sondern ganz allgemein Kryptomodule angesprochen werden. Durch die Einführung einer weiteren Abstraktionsschicht wird es auf einfache Weise möglich, zwischen unterschiedlichen Authentifikationsmechanismen zu wechseln, ohne diese selbst implementieren zu müssen. Nach der Authentifizierung und Etablierung gemeinsamer Geheimnisse können dann auf höheren Protokollschichten weitere Dienste wie Nicht-Abstreitbarkeit oder Delegation realisiert werden.

8.1 GSS-API

Der offensichtliche Weg, einen Sicherheitsmechanismus in eine Anwendung zu integrieren, besteht darin, diesen selbst als Teil der Anwendung zu implementieren. Dieser Ansatz ist jedoch zum einen fehleranfällig, zum anderen müssen bei einem eventuellen Austausch der verwendeten Algorithmen und Protokolle weite Teile des Quellcodes geändert werden. Besser ist es in jedem Fall, die Sicherheitsfunktionalitäten aus dem eigentlichen Quellcode auszulagern und nur über eine wohldefinierte Schnittstelle auf sie zuzugreifen. Für die Bereitstellung der Algorithmen selbst kann man sich spezieller Provider bedienen, der Austausch der Algorithmen geschieht einfach über das Austauschen des Providers bzw. durch Ändern der entsprechenden Argumente in den aufrufenden Funktionen.

Das GSS-API (Generic Security Service Application Programming Interface [RFC 2743]) stellt eine weithin akzeptierte und universell einsetzbare Schnittstelle für das Anfordern von Sicherheitsdiensten bereit, und zwar unabhängig von der verwendeten Programmiersprache oder dem anfordernden Protokoll. Die aufrufende Instanz (der so genannte *Caller*) weiß dabei nichts über den verwendeten

Mechanismus, das heißt die konkrete Art und Weise, wie der angeforderte Dienst realisiert wird. Die verfügbaren Dienste sind Authentifikation von Client und Server (je nach Mechanismus einseitige oder beidseitige Authentifikation), Delegation der Authentifikation sowie Integritätsschutz und Vertraulichkeit. Die letzten beiden Sicherheitsdienste werden jeweils auf einzelne Protokollnachrichten angewandt, wir haben es hier also mit nachrichtenbasierter Security zu tun (im Gegensatz zu kanalbasierter Security wie zum Beispiel SSL). Wird ein auf asymmetrischer Kryptografie basierender Mechanismus verwendet, ist zusätzlich auch nachrichtenbasierte Nicht-Abstreitbarkeit verfügbar, mit anderen Worten: Die versandten Nachrichten können auch einzeln digital signiert werden.

Ein typischer GSS-API Caller besteht in einem Kommunikationsprotokoll, welches seine Kommunikation durch Authentifikation der Kommunikationspartner, Integritätschutz, Vertraulichkeit und eventuell weitere Sicherheitsdienste schützen möchte. Dazu ruft es über das GSS-API die Funktionalität des darunter liegenden Providers auf und erhält ein entsprechendes Token als Antwort. So liefert zum Beispiel der Call `GSS_GetMIC()` einen Message Authentication Code (in diesem Kontext *Message Integrity Code* genannt) für eine zu versende Nachricht. Über den Call `GSS_VerifyMIC()` kann der Empfänger die Gültigkeit des Tokens und somit die Integrität der Nachricht verifizieren.

8.1.1 Überblick

Bei der Nutzung des GSS-API wird zunächst ein so genannter Security Context zwischen den kommunizierenden Peers etabliert. Dazu dienen die Calls `GSS_Init_sec_context()` und `GSS_Accept_sec_context()`. Im Laufe der Etablierung des Security Context einigen sich Client und Server zunächst auf die verwendeten Verfahren zur Authentifikation und zum Schutz der ausgetauschten Nachrichten, danach authentifizieren sie sich und vereinbaren einen Sitzungsschlüssel zum Schutz der nachfolgenden Kommunikation. Der grundlegende Ablauf sieht somit ganz ähnlich wie zum Beispiel beim SSL-Handshake aus, jedoch ist das GSS-API aufgrund der Unabhängigkeit von den verwendeten Protokollen und Mechanismen und der Orientierung an Nachrichten wesentlich flexibler einsetzbar. Die Kommunikation zwischen Client und Server läuft in dieser Phase über so genannte context-level tokens ab, die die sendende Seite als Ergebnis eines Calls vom GSS-API erhält. Die empfangende Seite übergibt das Token seinerseits als Argument einem Call an das GSS-API und lässt es vom darunter liegenden Mechanismus verarbeiten.

Nach der Etablierung des Security Contexts treten Client und Server in die Kommunikationsphase ein. Der Sender einer Nachricht übergibt diese als Argument an den Call `GSS_Wrap()` und erhält die gemäß dem vereinbarten Security Context geschützte, das heißt integritätsgeschützte und eventuell auch verschlüsselte Nachricht in Form eines message token zurück. Zur Prüfung des Integri-

tätsschutzes und zur eventuellen Entschlüsselung übergibt der Empfänger das message token als Argument an `GSS_Unwrap()`.

Nach Beendigung der Kommunikation können beide Seiten durch einen Aufruf von `GSS_Delete_sec_context()` das vereinbarte Schlüsselmaterial zerstören.

8.1.2 Ein Beispiel

Im folgenden, beispielhaften GSS-API Message Flow gehen wir von einer einseitigen Authentifikation des Client beim Server aus. Der hierfür eingesetzte Mechanismus soll so beschaffen sein, dass eine einzige Nachricht vom Client zum Server ausreicht (One-Pass Authentication). Beispiele für solche Authentifikationsmechanismen sind Kerberos oder das weiter unten erläuterte, auf dem RSA-Verfahren basierende SPKM-Verfahren.

Als weitere Voraussetzung nehmen wir an, dass der Client Zugriff auf seine Credentials (Private Key oder Ticket Granting Ticket im Falle von Kerberos) besitzt. Die Credentials werden durch den Call `GSS_Acquire_cred()` im Vorfeld bereitgestellt. Eine initiale Authentifikation wie die initiale Passwort-Eingabe des Nutzers in Kerberos, um einer bestimmten Instanz Zugriff auf die

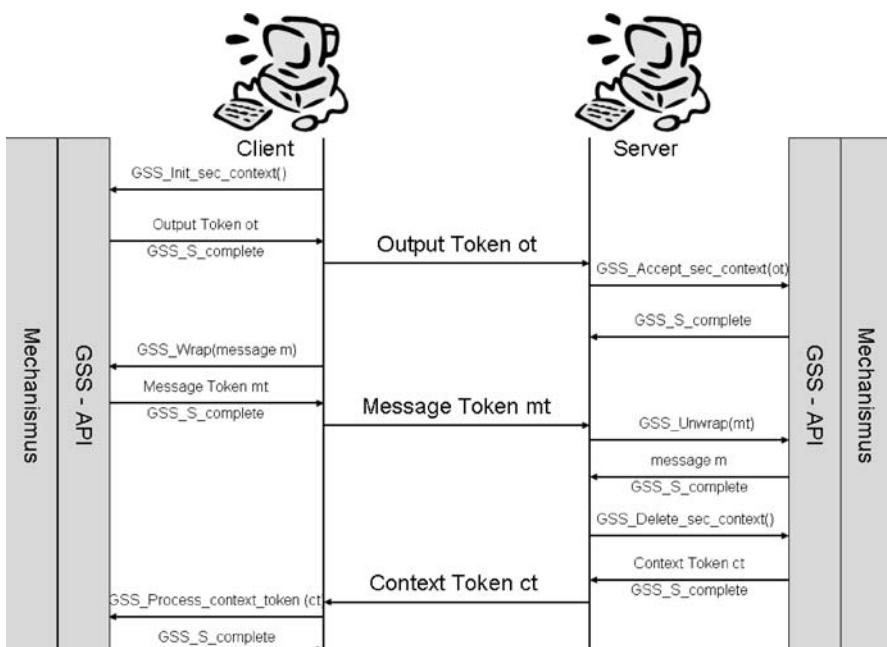


Abb. 8.1 Beispielhafter GSS-API Message Flow

Credentials zu ermöglichen, ist nicht Teil des GSS-API und sollte auf einer unteren, System-spezifischen Schicht realisiert werden.

Der Aufruf von `GSS_Init_sec_context()` liefert ein Output Token zurück, das der Client an den Server transferiert. Dieser übergibt es zur Verifikation mittels `GSS_Accept_sec_context()` an das GSS-API, welches bei erfolgreicher Authentifikation des Clients `GSS_S_complete` zurück liefert. Daraufhin nutzen beide Seiten den über das Output Token vereinbarten Sitzungsschlüssel zum Schutz der weiteren auszutauschenden Nachrichten via `GSS_Wrap()` bzw. `GSS_Unwrap()`. Nach Austausch aller Nachrichten (im Beispiel nur eine einzige) entfernt der Server den etablierten Security Context über `GSS_Delete_sec_context()`. Das resultierende Context Token übergibt der Client zur Interpretation an `GSS_Process_context_token()`, was dazu führt, dass auch auf Client-Seite der Security Context entfernt wird.

8.1.3 GSS-API Mechanismen

Wir diskutieren an dieser Stelle die gebräuchlichsten symmetrischen bzw. asymmetrischen Authentifikationsmechanismen KerberosV5 und SPKM für das GSS-API. Weitere Mechanismen sind selbstverständlich aufgrund der flexiblen Struktur des GSS-API ebenfalls denkbar, darunter insbesondere ein einfaches Username/Passwort-Schema wie GSSUP, das wir später im Kapitel zu CORBA noch kurz ansprechen werden.

Die genauen Spezifikationen für diese Mechanismen finden sich in [RFC 1964] (KerberosV5) und [RFC 2025] (SPKM-1 und SPKM-2).

8.1.3.1 KerberosV5

Im Kapitel über Basisprotokolle haben wir ja bereits den allgemeinen Mechanismus, der Kerberos zu Grunde liegt, diskutiert. An dieser Stelle werden wir sehen, wie sich Kerberos in den vom GSS-API vorgegebenen Rahmen einfügt.

Üblicherweise wird beim Login eine Verbindung zum lokalen Kerberos-Server hergestellt, der ein Ticket Granting Ticket und einen Session Key für die Kommunikation mit dem Ticket Granting Server erzeugt. Über den Call `GSS_Acquire_cred()` erhält der Client Zugriff auf das TGT und den dazu gehörigen Session Key. Ein weiterer Call von `GSS_Init_sec_context()` liefert ein Service Ticket für den gewünschten Service und legt es zusammen mit dem dazu gehörigen Session Key in Verbindung mit dem Kontext beim Client ab. Danach erzeugt `GSS_Init_sec_context()` ein Output Token, welches das Service Ticket und einen Authenticator enthält. Dieses Output Token schickt der Client an den Zielserver, der es seinerseits an `GSS_Accept_sec_context()` übergibt, der den Authenticator mit Hilfe des im Service Ticket enthaltenen Session Key verifizieren kann. Darüber hinaus kennt der Server nun den Namen des Client, der ja im

Authenticator in verschlüsselter Form enthalten ist. Beide Parteien verfügen nun über einen Session Key, den sie zum weiteren Schutz ihrer Kommunikation mit Hilfe von `GSS_Wrap()` etc. nutzen können.

8.1.3.2 Simple Public Key Mechanism (SPKM)

Die *Simple Public Key Mechanismen* (SPKM-1 und SPKM-2) dienen im GSS-API als asymmetrische Alternative zu dem symmetrischen Authentifikationsprotokoll Kerberos. Mit ihnen lassen sich einseitige Authentifikation sowie Nicht-Abstreitbarkeit auf Nachrichtenbasis erzielen – ein Dienst, der ja bei Kerberos als symmetrisch basiertem Mechanismus nicht zu realisieren ist. Als weitere Sicherheitsdienste stehen Vereinbarung eines Sitzungsschlüssels und Integritätschutz (basierend auf der Hashfunktion MD5) sowie Vertraulichkeit (basierend auf dem DES im CBC-Mode) unter Nutzung des vereinbarten Sitzungsschlüssel (der so genannte *Context Key*) zur Verfügung.

Die in SPKM verwendeten Mechanismen zur Authentifikation entsprechen den in Kap. 5.6 vorgestellten Protokollen nach X.509. Dabei nutzt SPKM-1 bei einseitiger Authentifikation des Client das Protokoll „Three-Pass-Authentication“ und bei beidseitiger Authentifikation das Protokoll „Three-Pass-Mutual-Authentication“. Beide Protokolle beruhen auf geeigneten Zufallszahlen als Random Challenges.

Stehen auf beiden Seiten synchronisierte Zeitstempel zur Verfügung, kommt die Variante SPKM-2 zum Einsatz. Hierbei wird zur einseitigen Authentifikation das Protokoll „One-Pass-Authentication“ genutzt, bzw. zur beidseitigen Authentifikation mit Zeitstempeln die offensichtliche Erweiterung der One-Pass-Authentication zur Two-Pass-Authentication, bei der der Server seinerseits den Zeitstempel signiert und an den Client schickt.

Durch den Call `GSS_Acquire_cred()` erhält der Client Zugriff auf seinen Private Key und kann dann mittels `GSS_Init_sec_context()` die nötigen Signaturen erzeugen.

Die Vereinbarung des Context Keys geschieht entweder dadurch, dass der „Initiator“ des Protokolls einen Context Key erzeugt und diesen mit dem RSA-Public Key des „Target“ verschlüsselt oder aber durch eine Variante des Diffie-Hellman-Schlüsselvereinbarungs-Protokolls, bei der Client und Server ihre Schlüsseltausch-Nachrichten signieren. Da dieses Protokoll aber zwei Nachrichten vom Initiator zum Target und wieder zurück erfordert, wird es nicht verwendet, falls im Rahmen von SPKM-2 nur eine einseitige Authentifikation gewünscht wird.

8.1.4 Simple and Protected Negotiation Mechanism (SPNEGO)

Die GSS-API-Spezifikation sagt nichts darüber aus, wie sich Client und Server auf den zu verwendenden Authentifikationsmechanismen einigen können. Dies ist

insbesondere dann von Bedeutung, wenn beide Seiten mehrere Mechanismen unterstützen. SPNEGO [RFC 2478] bietet eine einfache (und integritätsgeschützte, falls dies von beiden Seiten unterstützt wird) Möglichkeit, den Mechanismus auszuhandeln. Der Aushandlungsprozess fällt in die Phase der Etablierung eines Security Contexts. Die nötigen Daten werden im Rahmen von context-level tokens transportiert. Formal gesprochen stellt SPNEGO also einen zusätzlichen Mechanismus wie KerberosV5 oder SPKM dar. Die Nutzer des GSS-API müssen lediglich bei der Etablierung eines Security Contexts SPNEGO als Mechanismus wählen.

Der Initiator des Protokolls schickt dem Empfänger einen vorgeschlagenen Mechanismus oder eine geordnete Liste von Mechanismen. Optional kann der Initiator gleich ein zu seinem bevorzugten Authentifikationsmechanismus gehörendes Output Token mitschicken, so dass Nachrichten eingespart werden. Der Empfänger akzeptiert den vorgeschlagenen Mechanismus bzw. wählt einen der Mechanismen aus der Liste aus, oder er lehnt die vorgeschlagenen Mechanismen ab. Im Falle der Ablehnung liefert die GSS-API Implementation des Empfängers `GSS_S_BAD_MECH` und als context-level Token für den Initiator `reject`. Wird vom Zielserver ein bestimmter Mechanismus gewählt, liefert der `GSS_Accept_Sec_context()` Call entweder `GSS_S_complete` (falls genügend Informationen zur Authentifikation des Client vorliegen) oder aber `GSS_S_Continue_needed` zurück. Das context-level Token für den Initiator enthält den Object Identifier (OID) des ausgewählten Mechanismus sowie, falls erforderlich, je nach gewähltem Mechanismus Authentifizierungsdaten des Servers oder aber eine Challenge für den Initiator.

Auf diese Weise können beide Seiten einen Security Context etablieren. Der angesprochene Schutz des Aushandlungsprozesses lässt sich dann realisieren, wenn der vom Zielserver ausgewählte Mechanismus ein `MIC_token` (Message Integrity Code Token), also einen Message Authentication Code für die ausgetauschten Nachrichten, unterstützt. In diesem Fall wird vom Zielserver über die Liste der vom Initiator vorgeschlagenen Mechanismen ein MAC gebildet und in Form eines `MIC_token` an den Initiator zurück gesandt. Ein Fehlen dieses Tokens oder ein inkorrekt er MAC führt beim Aufruf von `GSS_Init_sec_context()` auf Seiten des Initiators zum Status `GSS_S_BAD_MIC`.

8.1.5 Delegation in GSS-API

Möchte der Initiator beim Etablieren eines Security Context seine Rechte an den Zielservice delegieren, so kann er dies beim Aufruf von `GSS_Init_sec_context()` durch das Setzen des Flags `deleg_req_flag` deutlich machen. Der Empfänger des Security Context bekommt über den Parameter `delegated_cred_handle` des Calls `GSS_Accept_Sec_Context()` die Identität mitgeteilt, in deren Namen der Zielservice zukünftig weitere Security Contexts etablieren kann. Je nach verwendetem Mechanismus kann dies die Identität des Initiators

sein oder die eigene, die im Auftrag des Initiators handelt. Eine feingranularere Delegation, bei der die transferierten Rechte auf eine bestimmte Zeitdauer oder ein bestimmtes Ziel eingeschränkt werden können, wird vom GSS-API nicht unterstützt.

8.2 SASL

SASL (spezifiziert in [RFC2222bis-15]) steht für *Simple Authentication and Security Layer*. Das Protokoll verdient seinen Namen tatsächlich: Es beschreibt eine einfache Methode, um Authentifikation und weitere Sicherheitsdienste oberhalb eines verbindungsorientierten Protokolls wie TCP zu implementieren, wobei von den konkreten Authentifizierungsmechanismen weitgehend abstrahiert wird. Auch SASL ist somit als Framework anzusehen, welches eine Schnittstelle zwischen verbindungsorientierten Protokollen und (austauschbaren) Authentifizierungsmechanismen bereitstellt.

Neben der Authentifikation von Client und Server besteht für den Client die Möglichkeit, neben der authentifizierten Identität eine weitere „Authorization Identity“ zu übermitteln, woraufhin der Server überprüft, ob der authentifizierte Client tatsächlich das Recht besitzt, sich unter der Authorization Identity anzumelden. Ein typisches Anwendungsbeispiel für SASL bildet die Situation, in der ein Proxy Server sich an einen Application Server wendet mit der Bitte, im Auftrag eines Users eine Aktion durchzuführen. Zunächst authentifiziert sich der Proxy Server mit der Identität, unter der er im Netzwerk bekannt ist und für die er Credentials in Form eines Passworts oder eines Zertifikats besitzt. Diese Identität wird im SASL-Kontext als *Authentication Identity* bezeichnet. Nach der Authentifikation übermittelt der Client die ID des Users, in dessen Auftrag er handelt. Diese UserID stellt die so genannte *Authorization Identity* dar. Ist der Proxy Server berechtigt, diesen User zu vertreten, kann der Application Server die Rechte des Users überprüfen und eventuell die angeforderte Aktion durchführen. Somit bietet SASL, anders als etwa SSL, eine einfache Möglichkeit, die zusätzlichen Dienste Delegation und Authorisation zu realisieren. Insbesondere können die vom Server durchgeführten Aktionen nun den einzelnen Usern zugeordnet werden und müssen nicht mehr global dem Proxy Server zugeschrieben werden. Je nach

SMTP	LDAP	andere		
SASL				
EXTERNAL	GSS-API	PLAIN	ANONYMOUS	DIGEST

Abb. 8.2 SASL als zusätzliche Abstraktionsschicht zwischen Anwendungen und Authentifikationsmechanismen

Ausgestaltung der „Authorization Identity“ kann der empfangende Server sogar deren Gültigkeit prüfen (z. B. anhand der Signatur eines vorgelagerten Identitätsdienstes. Er kann also prüfen ob sich der behauptete Client tatsächlich beim Aussteller der Authorization Identity authentisiert hatte. Selbst damit gelingt jedoch noch nicht der sichere Nachweis für den Server dass der Proxy Server tatsächlich im Auftrag des Clients arbeitet.

8.2.1 Funktionsweise

Im Normalfall wird im Rahmen eines Protokolls, das SASL zur Authentifizierung nutzen möchte, ein Server die von ihm unterstützten Authentifizierungsmechanismen bekannt geben. Der Client übermittelt dem Server daraufhin den vom Client gewünschten Mechanismus. Der Mechanismus muss als SASL-Mechanismus vorab bei der IANA (Internet Assigned Numbers Authority) registriert sein²⁰ und kann sehr einfach sein, wie etwa der unten erläuterte PLAIN SASL Mechanismus.

Erfordert der gewünschte Mechanismus einen initialen Input des Clients, so wird auch dieser in der ersten Nachricht vom Client an den Server übermittelt. Abhängig von der Funktionsweise des gewählten Mechanismus, schließt sich nun eine Abfolge von Challenges und Responses an, bei der der Server sich von der Identität des Clients überzeugt, wie sie vom SASL nutzenden Anwendungsprotokoll vorgegeben ist. Je nach Mechanismus kann auch zusätzlich der Client den Server authentifizieren. Schließlich übermittelt der Client an den Server die Authorization Identity, und der Server entscheidet, ob der Client berechtigt ist, im Auftrag der Authorization Identity zu handeln. Nur wenn beide Prüfungen, die Verifikation der Identität des Client und die Authorisierungsprüfung, erfolgreich waren, gilt der SASL-Austausch als erfolgreich durchgeführt. Die weitere Kommunikation wird nun mit dem im Rahmen des gewählten Authentifizierungsmechanismus vereinbarten Sicherheitsdiensten abgesichert.

8.2.2 SASL Mechanismen

Die SASL Working Group bei der IETF hat bislang zirka 20 SASL-Mechanismen in Form von Internet-Drafts veröffentlicht, von denen wir die wichtigsten hier kurz vorstellen:

²⁰ Eine aktuelle Liste der registrierten SASL-Mechanismen erhält man unter <http://www.iana.org/assignments/sasl-mechanisms>

- Der ANONYMOUS Mechanismus bietet dem Client die Möglichkeit eines anonymen Login ohne jegliche Authentifikation. Dies Option sollte per default vom anbietenden Server abgeschaltet sein und nur in wenigen Ausnahmefällen aktiviert sein.
- Der PLAIN Mechanismus besteht aus einer einzigen Nachricht vom Client zum Server, bestehend aus der Authorization Identity, der Authentication Identity (also der Identität, die tatsächlich vom Server verifiziert wird) und dem zur Authentication Identity gehörendem Passwort im Klartext. Da dieser Mechanismus keine weiteren Sicherheitsdienste zur Verfügung stellt, versteht es sich von selbst, dass er nur über einen sicheren Tunnel durchgeführt werden sollte, wie er beispielsweise von SSL/TLS zur Verfügung gestellt wird.
- Die Mechanismen CRAM-MD5 und DIGEST AUTHENTICATION sind beides auf der Hashfunktion MD5 und einem Passwort basierende Challenge-Response-Protokolle, sehr ähnlich der oben beschriebenen http-Digest-Authentification, wobei im Rahmen von CRAM-MD5 keine weiteren Sicherheitsdienste ausgehandelt werden können. Auch die Übermittlung einer Authorization Identity ist nicht vorgesehen. Im DIGEST AUTHENTICATION Mechanismus hingegen übermittelt der Client in seiner Response sowohl Authentication Identity als auch Authorization Identity. Darüber hinaus können aus dem Passwort und vom Client bzw. Server gebildeten Zufallszahlen mit Hilfe von MD5 weitere Schlüssel zum Integritätsschutz und Vertraulichkeit gebildet werden; dieser Mechanismus unterstützt also auch weitere Sicherheitsdienste neben der Authentifikation.
- Der GSS-API-Mechanismus greift über das oben beschriebene GSSAPI-Interface auf Kerberos oder andere Authentifikationsprotokolle zu. Außerdem kann über das gleiche Interface auch auf das SPNEGO Protokoll (siehe oben) zugegriffen werden, das zur gesicherten Vereinbarung eines Authentifikationsmechanismus dient.
- Schließlich bietet SASL noch den EXTERNAL Mechanismus an, bei dem zur Verifikation der Authentication Identity auf einen beliebigen, von SASL unabhängigen Mechanismus zurück gegriffen wird, wie etwa eine zertifikatsbasierte Authentifizierung via TLS oder auch IPSec. Nach erfolgter Authentifikation durch den externen Mechanismus wird wie bei den anderen Mechanismen die Authorization Identity übermittelt und die Berechtigung, im Namen dieser Identität handeln zu dürfen, durch den Server überprüft.

8.3 Zusammenfassung und Bewertung

Mit dem GSS-API und SASL stehen dem Entwickler von Anwendungen bzw. Protokollen zwei Frameworks zur Verfügung, die es ihm ermöglichen, in ein Protokoll oder eine Anwendung generische Authentifikationsdienste zu integrieren, ohne die konkreten Mechanismen selbst implementieren zu müssen. Bei Wechsel des Authentifikationsmechanismus, etwa von Kerberos auf einen zertifikatsbasierten

ten Mechanismus, muss somit der Quellcode so gut wie nicht modifiziert werden. Beide Frameworks unterstützen auch eine einfache Form der Delegation der Authentifikation, in beiden Fällen ist aber eine feingranulare Einschränkung der Delegation auf bestimmte Zugriffsrechte oder eine zeitliche Einschränkung der Delegation nicht möglich.

Mit JAAS existiert im Rahmen der Java Plattform Sicherheit ein weiteres Framework zur Authentisierung, dass es Java Applikationen erlaubt, die Authentisierung und Authorisierung der Laufzeitumgebung eines Application Servers zu bestimmen, ohne dass der zugehörige Code wirklich Teil der Applikation wird und damit bei organisatorischen Änderungen auch Änderungen im Code der Applikation erzwingt.

Kapitel 9

Middleware Security

In einem verteilten System sind die Software-Komponenten auf verschiedene Netzknoten verteilt. Middleware dient dazu, die Kommunikation dieser Software-Komponenten untereinander und ihre gegenseitige Nutzung zu ermöglichen. Die Tatsache, dass die Komponenten physikalisch auf mehrere Knoten verteilt sind, soll dabei für die nutzenden Komponenten möglichst transparent bleiben. Dazu nutzt die Middleware zum Verbindungsaufbau und zur Verbindungskontrolle die Dienste von Protokollen auf der Transportschicht, stellt also selbst eine Technologie auf der Anwendungsschicht dar. Für die Middleware Security stellen sich deshalb grundsätzlich die folgenden Fragen:

- Soll die Anwendung ihre eigene Security mitbringen oder soll sie Sicherheitsdienste auf tieferen Schichten des OSI-Modells nutzen, wie sie von SSL/TLS auf der Transportschicht oder auch von IPSec auf der Internet-Schicht angeboten werden?
- Wie lassen sich Delegation und Authorisierung bei der Nutzung von Middleware realisieren?

Im Folgenden werden wir die wichtigsten Middleware-Technologien und ihre Funktionsweise kurz vorstellen und ihre Security mit Blick auf obige Fragestellung diskutieren.

9.1 CORBA

CORBA (Common Object Request Broker Architecture) stellt eine Technologie dar, die es dem Entwickler erlaubt, verteilte Anwendungen in einer standardisierten Art und Weise zu implementieren, die Portabilität und Interoperabilität zwischen verschiedenen Plattformen und Programmiersprachen garantiert. Der *Object Request Broker* (ORB) stellt die zentrale Komponente der CORBA-Architektur dar (siehe Abb. 9.1).

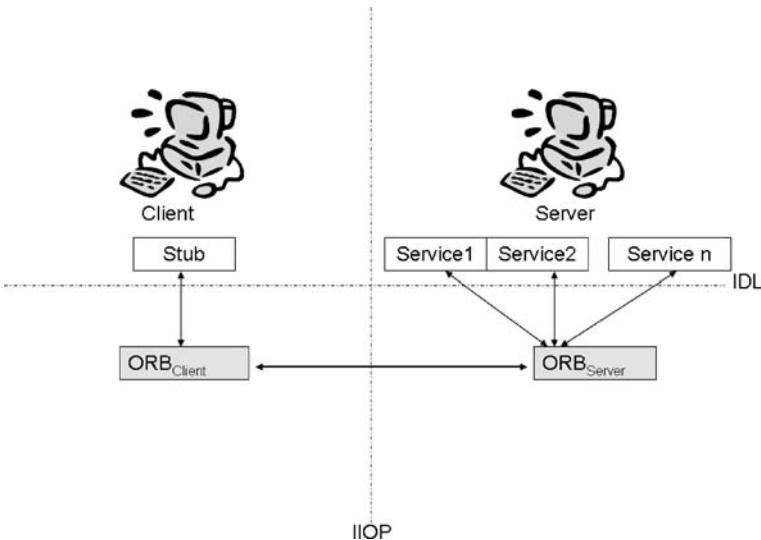


Abb. 9.1 Schnittstellen der CORBA-Architektur (nach [ALP])

Eine verteilte Anwendung interagiert mit der darunter liegenden CORBA-Infrastruktur über eine horizontale Schnittstelle, die CORBA-spezifische, von der benutzten Programmiersprache unabhängige Interface Definition Language (IDL). Serverseitig werden den Clients Dienste angeboten, die via IDL spezifiziert sind. Die Applikation auf Client-Seite kann die angebotenen Dienste nutzen, indem sie auf einen geeigneten Stub (Stummel) zugreift, der auf Clientseite als Stellvertreter-Objekt dient. Für die konkrete Implementierung kann nun im Prinzip jede beliebige Programmiersprache genutzt werden, sofern eine geeignete Umsetzung von IDL in diese Sprache existiert.

9.1.1 SECIOP und SSLIOP

Die Interaktion zwischen verschiedenen Object Request Brokern (ORBs) auf Client- und Serverseite geschieht bei TCP/IP basierten Netzen über ein zweites (vertikales) Interface mit dem Namen IIOP (Internet Inter-ORB Protocol). IIOP bildet das CORBA GIOP (Generic Inter-ORB Protocol) auf TCP ab. Diese Kommunikation auf der Transportschicht kann nun via SSL abgesichert werden (so genanntes SSLIOP). Alternativ kann die Kommunikation auf der Transportschicht auch über das SECIOP-Protokoll, das IIOP auf die Dienste des GSS-API zugreifen lässt und somit über die Sicherheitsdienste von SSL hinausgehende Sicherheitsdienste wie etwa eine Delegation bietet, gesichert werden. Von SECIOP unterstützte Authentifikationsprotokolle sind Kerberos (GSS-Kerberos oder CSI-

ECMA²¹⁾ oder die beiden RSA-basierten Protokolle SPKM (einer der im vorigen Kapitel erwähnten GSS-API Mechanismen) bzw. die asymmetrische Variante von CSI-ECMA (siehe auch [CORBASec]).

9.1.2 CSIV2 und SAS

Aufgrund der hohen Komplexität von SECIOP werden im Moment nur relativ wenige Implementationen angeboten. Deshalb wird zur Absicherung von IIOP häufig SSLIOP eingesetzt, mit den bereits diskutierten Defiziten von SSL wie die kanalbasierte Sicherheit und fehlende Sicherheitsdienste wie Nicht-Abstreichbarkeit und Delegation. Um diese trotzdem in CORBA integrieren zu können, wurde in der Common Secure Interoperability Specification, Version 2 [CSIV2] mit dem *Security Attribute Service* (SAS) ein Sicherheitsprotokoll auf der Anwendungsschicht spezifiziert, mit dem Ziel, genau die in SSLIOP fehlenden Dienste anzubieten (siehe Abb. 9.2).

SAS nutzt GIOP Requests und Replies, um einen so genannten *Security Context* zwischen Client und Server zu etablieren. Dieser kann neben einer Authentifikationsmethode auch Privilegien des Clients und eine Identität, in deren Name der Client handelt, beinhalten. Dazu wird der bereits aus dem GSS-API bekannte und bewährte Token-basierte Mechanismus übernommen, wobei die Token hier mit Hilfe der GIOP-Nachrichten übertragen werden. Für den Transport der Tokens geht SAS davon aus, dass auf der darunter liegenden Transportschicht Sicherheitsdienste Server Authentifikation, Vertraulichkeit und Integritätsschutz vorhanden sind, also genau die Dienste, die normalerweise von SSL bereit gestellt werden. Das darauf aufsetzende SAS-Protokoll ist in zwei Schichten (Layers) aufgeteilt, wie Abb. 9.2 zeigt.

Der Authentication Layer führt eine Authentifikation des Client durch, sofern diese nicht bereits auf der Transportschicht stattgefunden hat.

Der Attribute Layer kann vom Client genutzt werden, um weitere so genannte Security Attributes an den Server zu übertragen. Zu den Security Attributes gehört ein `authorization_token`, mit dem Informationen über die Rechte des Client

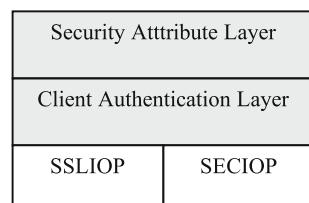


Abb. 9.2 SAS Service Context Protocol im Überblick

²¹ CSI-ECMA bietet drei Alternativen zur Schlüsselvereinbarung: Eine asymmetrische Variante via RSA, eine symmetrische via Kerberos und eine hybride Variante, bei der innerhalb einer Domäne Kerberos und Domänen-übergreifend RSA eingesetzt wird.

an den Server übertragen werden. Ebenso ist denkbar, dass der Client die Rechte, die im `authorization_token` (beispielsweise in Form eines Attributzertifikats oder Privilege Attribute Certificate, PAC) enthalten sind, an den Server überträgt (Delegation). Schließlich ist auch, ähnlich wie in SASL, die Angabe einer weiteren Identität möglich, in deren Auftrag der Client handelt. Diese Identität wird im `identity_token` übertragen. Über den Attribute Layer werden also in CSIV2 die Dienste Authorisierung, Delegation und Impersonation realisiert.

Die `authorization_token` haben den Datentyp `Authorization ElementType`. Dieser Typ kann von den implementierenden Organisationen selbst definiert und bei der OMG registriert werden. Ein spezieller Datentyp `X509AttributeCertChain`, der einer Kette von Attributzertifikaten nach X.509 entspricht, ist in [CSIV2] bereits vorgegeben.

SAS arbeitet mit dem gleichen Token-basierten Schema zur Realisierung von Sicherheitsdiensten wie das GSS-API. Zur Client-Authentifikation können allerdings nur solche Mechanismen eingesetzt werden, die mit einer einzigen Nachricht von Client zu Server auskommen. Als minimal zu unterstützenden Authentifikationsmechanismus für den Client definiert CSIV2 den GSSUP-Mechanismus, der im einfachen Senden von Username und Passwort besteht (man beachte, dass von einem unterhalb von SAS liegenden geschützten Kanal zwischen Client und Server ausgegangen wird).

Darüber hinaus muss eine CSIV2-konforme Implementation auf dem niedrigsten Conformance Level (Level 0) die Absicherung der Transportschicht mit SSL/TLS unterstützen und außerdem Impersonation mit Hilfe des bereits angesprochenen `identity_tokens` ermöglichen. Eine Level 1-konforme Implementation muss zusätzlich die Authorisierung mit Hilfe eines `authorization token` unterstützen, das heißt, insbesondere muss ein Zielsystem ein Attributzertifikat validieren und die darin enthaltenen Rechte interpretieren können. Der höchste Conformance Level 2 ermöglicht schließlich die Delegation von Rechten mit Hilfe eines delegierbaren `authorization_token` in Form eines Attributzertifikats, das von dem Rechteinhaber für den Client ausgestellt wurde. Da dieses Token nur eine begrenzte Gültigkeit besitzt, lässt sich so auch die gewünschte Flexibilität erzielen, die im Delegationsmechanismus des GSS-API fehlt.

Die Abb. 9.3 zeigt zunächst eine SAS-Kommunikation zwischen einem Client P1, der einen anderen Client P2 autorisiert, für ihn zu agieren. Daraufhin präsentiert P2 beim Zielserver das von P1 erhaltene `authorization token`, in welchem P1 seinem Proxy P2 das Recht zuspricht, für ihn zu handeln (s. Abb. 9.4). Die hier eingesetzten Implementationen müssen also den Conformance Level 2 unterstützen.

In Abb. 9.3 authentifiziert sich nach dem SSL-Handshake P1 gegenüber P2 durch Username und Passwort. Danach autorisiert P1 den Proxy P1, in seinem Namen zu handeln. Das FALSE-Flag zeigt an, dass der hergestellte Security Context verbindungslos ist, also auf einer per-message-Basis gültig ist. In Abb. 9.4 ist die Situation eine andere: Hier ist zu beachten, dass sich P2 bereits im Laufe des SSL-Handshake authentifiziert, das heißt, P2 beweist, dass er im Besitz des Private Key ist, der zum Public Key in seinem Zertifikat gehört. Somit hat der Server

Abb. 9.3 Übergabe eines authorization_token von P1 an P2

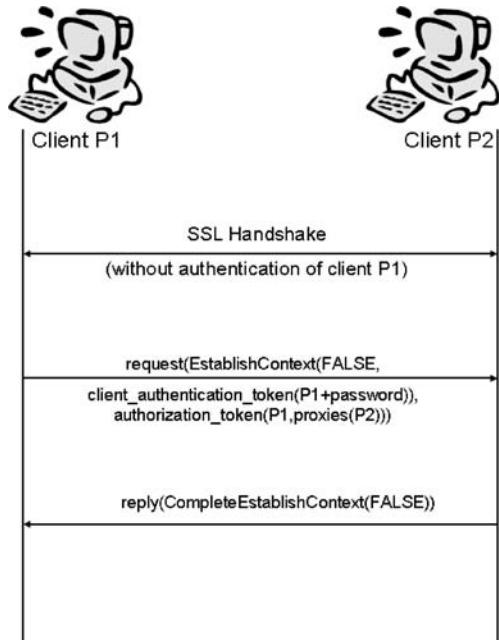
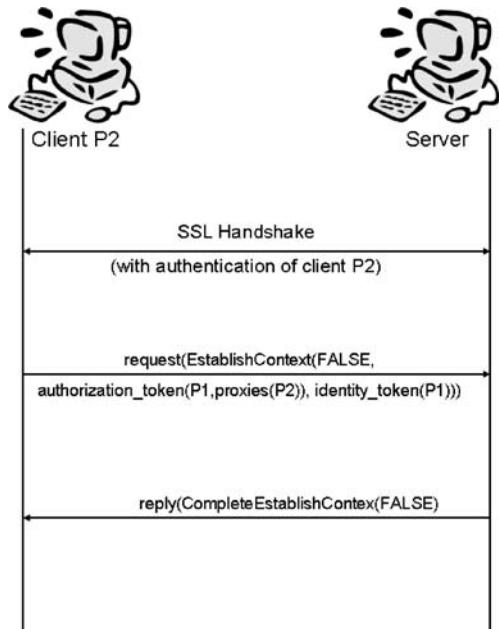


Abb. 9.4 Nutzung des authorization_token durch P2



die Möglichkeit, auch das präsentierte `authorization_token` zu validieren, welches ja der Identität P2 gewisse Rechte bescheinigt. Voraussetzung hierfür ist natürlich, dass der Server den Public Key von P1 kennt, also ein Zertifikat von P1 besitzt.

Ganz simpel ausgedrückt erfordert der CSIV2 Mechanismus zunächst die gegenseitige Authentisierung unmittelbarer Partner. Dies wird üblicherweise auf Basis von SSL durchgeführt und kann beim Aufrufer auf Basis von Zertifikaten oder Passwörtern stattfinden. Anschliessend kann auf Basis der Authentisierung eine Vertrauensbeziehung beider Rechner so genutzt werden, dass der Server Informationen über den Client bzw. sogar den Originator (der eigentliche Aufrufer) akzeptiert, z. B. dass der Originator XYZ heisst und authentisiert wurde.

Alle Informationen, die über die blosse gegenseitige Authentisierung der unmittelbaren Partner hinausgehen, können kryptographisch abgesichert werden (Token) und erlauben dann eine Verifizierung auf Seiten des Servers. Sicherheitstechnisch gesehen wird das Protokoll umso mächtiger, je mehr signierte Token eingesetzt werden können. So können z. B. die Rechte des Originators bei seiner Authentisierung bereits einmal ermittelt worden sein und anschliessend allen Requests beigefügt werden. Dies spart dem Server einen Netzwerk-Request, um der übergebenen Identität die entsprechenden Rechte z. B. aus einem LDAP Server zuordnen zu können.

Ohne Token hingegen hängt die Sicherheit der Vertrauensbeziehungen sehr stark an der jeweiligen Infrastruktur, da der Server zwar einen vorgelagerten Partner identifizieren kann, aber ansonsten darauf angewiesen ist, dass dieser nicht kompromittiert wurde (und jetzt z. B. falsche Originator-Identitäten behauptet). Die entscheidende Frage für nachgelagerte Server ist ja immer, ob der empfangene Request wirklich mit einem Request eines berechtigten Originators zusammenhängt oder durch eine Schwachstelle in der Infrastruktur künstlich erstellt wurde.

Die Sicherheitsarchitektur in Web-Services aber auch die Technik der sog. Proxy-Zertifikate im Grid-Computing Bereich versuchen, durch objektbasierte Sicherheit mit Hilfe von signierten Nachrichten die Abhängigkeit von der Infrastruktur weiter zu verringern. Die Route eines Requests durch die jeweilige Infrastruktur wird dabei weniger wichtig, da der Server im Extremfall Antworten an den Originator mit Hilfe dessen öffentlichen Schlüssels sichern könnte. Nicht verschwiegen werden darf an dieser Stelle allerdings, dass der Einsatz objektbasierter Sicherheit mit PKI innerhalb von Infrastrukturen zu Performance Problemen führen kann, so dass häufig dennoch mit Vertrauenszonen gearbeitet wird.

9.2 Remote Method Invocation (RMI)

Mit RMI ist im Java Development Kit (JDK) seit der Version 1.1 ein Mechanismus enthalten, der es erlaubt, Java-basierte Objekte im Netz zu verteilen und über das Netzwerk auf sie zuzugreifen. Das Grundprinzip ist ganz ähnlich dem in CORBA verwendeten: Die entfernten Objekte werden in einer RMI-Registry registriert, wo sie von anfragenden Clients aufgefunden und angefordert werden

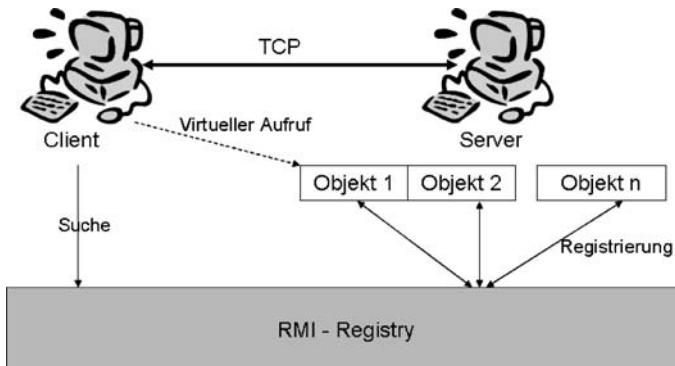


Abb. 9.5 Funktionsweise von RMI (nach [Kru], S. 1092)

können (vgl. Abb. 9.5). Die eigentliche Kommunikation zwischen Client geschieht über TCP und darauf aufsetzend über JRMP (s. unten) oder IIOP. Deshalb muss auf dem Zielrechner ein `RMISecurityManager` installiert sein, und die dazu gehörige Policy-Datei muss eine TCP-Kommunikation mit dem Quellrechner und das Herstellen einer Verbindung über den entsprechenden Port erlauben. Der eigentliche Aufruf der entfernten Objekte erfolgt dann für den Zielrechner weitgehend transparent.

RMI selbst bietet im Moment kaum Sicherheitsfunktionalitäten an. Der oben erwähnte `RMISecurityManager` ermöglicht lediglich das dynamische Laden von Bytecode über das Netzwerk, hat aber keinen Einfluss auf die Absicherung des eigentlichen Datenverkehrs. Aufsetzend auf TCP/IP, kann RMI entweder über JRMP (Java Remote Method Protocol) oder das oben bereits erwähnte IIOP transportiert werden.

JRMP ist ein proprietäres, von Sun entwickeltes Protokoll, das lediglich die Kommunikation von Java-basierten Objekten untereinander ermöglicht. Gerade bei größeren Projekten oder solchen, bei denen bereits bestehende Objekte in anderen Sprachen integriert werden müssen, stellt das von Sun und IBM gemeinsam entwickelte RMI über IIOP (auch RMI-IIOP genannt, verfügbar seit JDK v1.3) die bessere Wahl dar, zumal dann auch die Sicherheitsdienste von SSLIOP bzw. SECIOP für den Transport der Objekte genutzt werden können.

9.3 .NET

9.3.1 Allgemeines

Das .NET Framework kann mit einiger Berechtigung als Microsofts Antwort auf SUNs Java betrachtet werden. Die Entwicklung startete Mitte 2000, die aktuelle Version 2.0 wurde Ende 2005 veröffentlicht. Ziel ist wie bei Java, Plattformunab-

hängigkeit für den in einer Hochsprache vorliegenden Sourcecode zu erreichen. Während dies bei Java allerdings nur für eine einzige Hochsprache gilt, zielt das .NET Framework auf Plattformunabhängigkeit für Code in (nahezu beliebigen) Hochsprachen. Dieses Ziel lässt sich ohne eine virtuelle Maschine, die für alle Plattformen eine einheitliche Schnittstelle bietet, nicht realisieren. In .NET heißt diese virtuelle Maschine *Common Language Runtime* (CLR). Mittlerweile existieren für die meisten Hochsprachen .NET Compiler, zum Teil müssen diese Sprachen aber für den Einsatz in .NET leicht modifiziert werden (aus C++ wird C#, aus Java wird J#).

Der .NET Compiler übersetzt den Sourcecode in eine gemeinsame Sprache, die *Common Intermediate Language* (CIL). Diese ist in etwa vergleichbar mit dem Java-Bytecode. Der übersetzte CIL-Code eines Pakets bekommt die Endung .exe und wird unter .NET als *Assembly* bezeichnet.

Bei der eigentlichen Programmausführung übersetzt die CLR mit Hilfe eines Just-in-Time-Compilers (JIT) den CIL-Code in Maschinencode für den jeweiligen Prozessor und führt diesen aus. Dabei prüft die CLR insbesondere, ob der Code die nötigen Zugriffsrechte (so genannte *Permissions*) auf die angeforderten Ressourcen besitzt (*Code Access Security*) und auch, ob der Nutzer der Anwendung die für die Ausführung der Anwendung erforderlichen Rechte besitzt (*Role Based Security*). Von diesen beiden Rechtemengen wird immer die Schnittmenge ausgewählt, das heißt, ein Nutzer kann seinen Rechtevektor nicht dadurch vergrößern, dass er besonders vertrauenswürdigen Code ausführt. Umgekehrt vergrößert sich aber auch nicht die Menge der Permissions einer .NET-Anwendung, wenn sie von einem besonders mächtigen Nutzer ausgeführt wird. Die .NET Security ruht also auf zwei Säulen, der Code Access Security und der Role Based Security. Diesen beiden Säulen wollen wir uns nun im Detail zuwenden.

9.3.2 *Code Access Security (CAS)*

Die Rechte, die eine Assembly zugewiesen bekommt, hängen von der so genannten *Code Group* ab, zu der die Assembly gehört. Über die Zugehörigkeit zu einer Code Group bestimmen zum einen die Herkunft des Code (lokal, aus dem Intranet, aus dem Internet, von einer bestimmten Webseite kommend, usw.) sowie ein eindeutiger Identifikator für die Assembly, der so genannte *Strong Name*. Dabei handelt es sich um ein ganz neues Konzept, das die vorher Microsoft verwendeten Global Unique Identifiers (GUID) ablöst. Ein Strong Name besteht aus dem vom Entwickler vergebenen Namen, einer Versionsnummer, der Culture der Assembly sowie – und das ist entscheidend – einem Public Key, der vom Entwickler zum Signieren der Assembly und ihrer Metadaten verwendet wird. Dieser Public Key (oder das so genannte *Public Key Token*, das sind die letzten acht Bytes des SHA-1 Hashwerts des Public Keys) ist der eigentliche Name der Assembly. Ein Angreifer könnte zwar eine eigene Assembly mit einem fremden Public Key versehen, von dem er weiß, dass dieser für die CLR vertrauenswürdig ist, würde

jedoch daran scheitern, seine eigene Assembly mit diesem Key digital zu signieren. Die CLR prüft also zur Verifikation eines Strong Name immer auch die Signatur über die Assembly und ihre Metadaten – mit einer Ausnahme: Kommt die Assembly aus einem besonders vertrauenswürdigen lokalen Verzeichnis, dem *Global Assembly Cache* (GAC), wird auf die Signaturverifikation aus Performanzgründen verzichtet. Allerdings wird in diesem Fall die Signatur einmalig in dem Moment geprüft, in dem die Assembly in den GAC installiert wird. Welche Assemblies mit welchen Strong Names dort installiert werden dürfen, legt der Administrator fest, der auch als Einziger Schreibrechte auf die GAC besitzt. Ein Angreifer mit Administratorrechten auf einer Machine könnte also eigene Assemblies in den GAC einbringen und die Code Access Security weitgehend aushebeln.

Zurück zu den Code Groups: Sie sind in voneinander unabhängigen Baumstrukturen angeordnet, den so genannten *Policy Levels*. Es gibt vier Policy Levels, und zwar (in absteigender Reihenfolge der Priorität): Enterprise Level, Machine Level, User Level, Application Domain Level. Zuerst wird also der Enterprise Level-Baum durchlaufen und die mit diesem Baum verbundenen Rechte bestimmt, danach die anderen drei Bäume. Der letzte dieser Level dient dazu, einem einzelnen Nutzer mehrere Policies zuordnen zu können, je nach Anwendung, die dieser ausführt. Am Ende werden die Rechte aus den vier Baumdurchläufen miteinander kombiniert, im einfachsten Fall durch einfache Durchschnittsbildung der Rechtemengen²². Wie aber werden innerhalb eines Policy-Level-Baums die Rechte berechnet, die sich aus dem Durchlaufen des Baums ergeben? Im Allgemeinen wird eine Assembly immer Mitglied mehrerer Code Groups sein, die für die Herkunft des Codes, ihren Publisher und ihren Namen stehen. Im einfachsten Fall werden die Rechte aus diesen Code Groups miteinander vereinigt. Durch unterschiedliche *Code Group Types* ist es aber auch hier wieder möglich, abweichende Berechnungsmuster festzulegen: So wird bei einer `FirstMatchCodeGroup` das Durchlaufen des Teilbaums, dessen Wurzel die aktuelle Code Group ist, beim ersten Kindknoten abgebrochen, zu dem die Assembly ebenfalls gehört. Andere Code Group Types betreffen das Recht einer Assembly, Verbindung zum Quellsystem aufzunehmen (`NetCodeGroup`) oder das Recht, auf das Ursprungsverzeichnis zuzugreifen (`FileCodeGroup`).

Wir haben nun geklärt, wie die CLR den Ursprung einer Assembly auf sichere Weise bestimmen und die Rechtemenge berechnet, die der Assembly gewährt werden kann. Ein Problem bleibt jedoch: Wie kann verhindert werden, dass eine Assembly, der nur wenig Vertrauen entgegen gebracht wird, eine andere Assembly mit höherem Vertrauen und entsprechend mehr Rechten aufruft und sich so auf einem Umweg Rechte „erschleicht“, die ihr gemäß Policy nicht zustehen?

²² Durch die einfache Durchschnittsbildung erhalten alle Policy Levels das gleiche Gewicht. Ist dies nicht erwünscht, können die Code Groups in einem Baum mit dem `LevelFinal`-Attribut (das heißt, tiefere Level werden nicht mehr ausgewertet) oder dem `Exclusive`-Attribut (zugewiesene Rechtemenge ist genau gleich der Rechtemenge, die sich aus diesem Level ergibt) ausgestattet werden.

Hierzu führt die CLR einen so genannten *Stack Walk* durch: Ein Stack Walk durchläuft von unten nach oben den „Stapel“ der Assemblies, der zu der aktuellen Anforderung einer Permission P geführt hat. Nur wenn alle Assemblies des Stapels diese Permission auch besitzen, wird der Zugriff gewährt. Würde man den Stack Walk konsequent bei jeder Assembly anwenden, würde die Verwendung von .NET Assemblies durch „natiiven“, nicht der CLR-Kontrolle unterworfenen Code (so genannter *unmanaged code*) unmöglich werden, da der Stack Walk unweigerlich beim Erreichen des unmanaged code zum Halten käme. Deshalb besteht die Möglichkeit für eine .NET Assembly, beim Anfordern einer bestimmten Ressource, den Stack Walk so zu modifizieren, dass er genau bei der aktuellen Ressource zum Halten wird. Dies geschieht mit Hilfe der `assert()`-Methode. Mit dem Verwenden dieser Methode gibt die Assembly der CLR zu verstehen, dass sie für eventuelle Aufrüfer bürgt, auch wenn diese nicht das erforderliche Recht besitzen sollten.

Neben der CLR auf der Zielmaschine hat auch der Entwickler selbst die Möglichkeit, seinen .NET-Code sicherer und zuverlässiger zu machen, indem er selbst Mengen von Rechteanforderungen in der Assembly definiert. Diese umfassen die minimale Rechtemenge MinP , die zum Ausführen unbedingt benötigt wird, eine optionale Rechtemenge OptP aus Rechten, die nicht unbedingt benötigt werden, und eine Menge RefP aus Rechten, die auf keinen Fall benötigt werden und die deshalb auf keinen Fall von der CLR gewährt werden sollen, auch wenn die Policy dies zuließe.

Mit der Code Access Security unter .NET steht also eine sehr leistungsfähige Sicherheitsarchitektur zur Verfügung, die es ermöglicht, sehr feingranular Rechte an Anwendungen zu vergeben und die Berechtigungen auf sichere Weise nachzuprüfen.

9.3.3 Role Based Security (RAS)

Mit der rollenbasierten Sicherheit wird eine Brücke geschlagen von der rein codebasierten Sicherheit, die wir im letzten Abschnitt betrachtet haben, hin zu dem aus der Betriebssystem-Sicherheit kommenden Konzept der Nutzer, die in unterschiedlichen Rollen unterschiedliche Rechte besitzen. An erster Stelle einer rollenbasierten Sicherheit muss natürlich die sichere Etablierung der Nutzeridentität stehen. Dazu dienen in .NET die so genannten Authentication Provider, über die der Entwickler auf diverse Authentifikationsprotokolle zugreifen kann.

Dazu gehören die bereits angesprochene http-Authentifikation, bei der nach gelungener initialer Authentifikation per UserID und Passwort ein Cookie gesetzt wird, mit dem darauf folgende Authentifikationen durchgeführt werden können, weiterhin die von Microsoft entwickelte Passport-Authentifikation und die unter Microsoft Windows verfügbaren Authentifikationsmechanismen.

Nachdem die Identität eines Nutzers etabliert ist, kann der Entwickler die dazu gehörige Authorisierung innerhalb des Code implementieren, indem entweder die angeforderte Ressource auf die dazu gehörige physikalische Datei abgebildet wird

und somit die Zugriffsmechanismen des Betriebssystems greifen, oder aber durch eine analoge Operation auf URL-Ebene, bei der der Entwickler die Zugriffsrechte selbst spezifiziert.

Neben der Authentifikation und Authorisierung lassen sich auch zwei einfache Formen der Impersonation innerhalb von .NET-Anwendungen realisieren. Neben der üblichen Form der Impersonation, bei der eine Anwendung jeweils mit der Identität und allen Rechten der aufrufenden Entität läuft, ist auch eine Impersonation auf Anwendungsebene möglich. Hierbei nimmt der Prozess, der eine Anwendung aufruft, eine innerhalb der Anwendung spezifizierte Identität an. Wenn diese Identität entsprechend geringe Rechte besitzt, kann dadurch kann das Risiko durch eine kompromittierte Anwendung minimiert werden.

9.4 Simple Object Access Protocol (SOAP)

9.4.1 Allgemeines

Die Knoten in einem verteilten System können über XML-basierte Dokumente Nachrichten oder auch Funktionsaufrufe (so genannte *Remote Procedure Calls*) austauschen. Das Protokoll, das die entsprechenden Dokumente spezifiziert, heißt SOAP (Simple Object Access Protocol, [SOAP]). Häufig werden beispielsweise Web-Services mit Hilfe von SOAP implementiert. Hier zunächst etwas SOAP Terminologie:

Die Teilnehmer des Systems werden auch als SOAP *Nodes* bezeichnet. Eine SOAP-Nachricht wird vom SOAP *Sender* zum SOAP *Receiver* gesendet. Dies kann auch über mehrere Zwischenstationen, die so genannten SOAP *Intermediaries* geschehen. Eine SOAP-Nachricht wird auch als *Envelope* bezeichnet und stellt für sich genommen ein XML-Dokument dar (s. [XML]). Der Envelope besteht wiederum aus einem (optionalen) *Header*-Element und einem (mandatoryschen) *Body*-Element. Im Body-Element wird die eigentliche, vom SOAP Sender für den SOAP Receiver bestimmte Nachricht transportiert. Der Header transportiert so genannte SOAP *Extensions* (siehe auch Abb. 9.6).

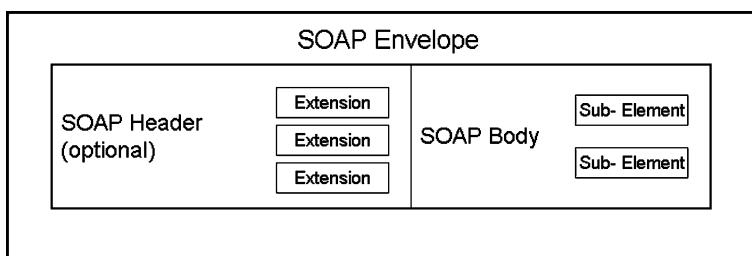


Abb. 9.6 Aufbau einer SOAP-Nachricht

Die Extensions bieten die Möglichkeit, gewisse, zum Body gehörige Metadaten zu transportieren. Diese können auch für Intermediaries bestimmt sein und von diesen verarbeitet werden. Insbesondere ist es auch möglich, dass Intermediaries Extensions löschen und eigene hinzufügen.

9.4.2 *SOAP Security*

Da SOAP-Nachrichten meist über http (also über TCP Port 80) transportiert werden, können diese Nachrichten ohne Schwierigkeiten Firewalls durchtunneln, da diese häufig so konfiguriert sind, dass sie Nachrichten über den „ungefährlichen“ Port 80 durchlassen. Gerade wenn ein Remote Procedure Call via SOAP transportiert wird, ist es deshalb von besonderer Wichtigkeit, im Rahmen der SOAP-Nachricht weitere Metadaten zu transportieren, die es dem Empfänger ermöglichen, zu entscheiden, ob der Sender zu dem Call überhaupt autorisiert ist.

Mit SSL steht zwar ein Sicherheitsprotokoll für http zur Verfügung, aufgrund unserer obigen Diskussion von SSL sollte aber klar sein, dass das bloße Transportieren einer SOAP-Nachricht via http und SSL keine wirkliche Lösung für dieses Problem darstellt. Zwar ist die SOAP-Nachricht durch den SSL-Tunnel auf dem Transportweg vertraulich und integritätsgeschützt, aber der Empfänger hat nach dem Verlassen des Tunnels eine „nackte“ SOAP-Nachricht ohne jegliche Metadaten vor sich. Insbesondere lässt sich durch SSL keine Nicht-Abstreitbarkeit auf Nachrichtenebene realisieren. Zudem bietet SSL keine „Ende-zu-Ende-Sicherheit“, sondern nur „Punkt-zu-Punkt-Sicherheit“, das heißt, die Tunnels werden jeweils von einer Zwischenstation zur nächsten aufgebaut. Auf den Zwischenstationen liegen die Nachrichten ungeschützt vor. Es gibt keine Garantie, dass diese Zwischenstationen die Nachricht nicht modifizieren oder gar im Klartext zur nächsten Station weitersenden. Die gleichen Argumente sprechen auch gegen eine Absicherung auf noch tieferen Schichten, etwa durch IPSec.

Trotz dieser Problematik enthält die aktuelle Version V1.2 der SOAP-Spezifikation keinerlei Sicherheitsfunktionalitäten. Zwar wird die Sensitivität von SOAP-Nachrichten durchaus erkannt, die nötigen Sicherheitsdienste sollen aber in Form so genannter SOAP Security-Extensions realisiert werden, die im Moment nicht Teil des Standards sind. Es existiert jedoch ein Vorschlag für eine Security-Extension, die es ermöglicht, SOAP Nachrichten digital zu signieren [SOAP-Sig]. Dazu wird eine Extension <SOAP-Sec:Signature> definiert. Diese Extension enthält die zur Validierung der Signatur nötigen Informationen wie die zur Erstellung der Signatur benutzten Algorithmen, der dazu gehörige Public Key, usw. Da es sich bei einer SOAP-Nachricht letztlich um ein XML Dokument handelt, werden diese Daten in einem zur XML-Signatur-Spezifikation [XML-Sig] kompatiblen Format dargestellt. Mit Hilfe dieser Extension kann der Body oder ein beliebiger anderer Teil des SOAP-Envelope digital signiert werden und die Signatur über den Header transportiert werden. Damit sind einige, jedoch noch längst nicht alle Probleme der SOAP-Security gelöst.

Das Nachfolgedokument [WS-SEC] geht deshalb einen Schritt weiter als [SOAP-Sig] und beschreibt eine neue, allgemeinere Security-Extension, die auch Vertraulichkeit nach [XML-Enc], den Transport symmetrischer Schlüssel und die Weitergabe so genannter Security Tokens (also zum Beispiel Kerberos-Tickets) einschließt. Damit wird eine echte Ende-zu-Ende Sicherheit für SOAP-Nachrichten realisiert, in Verbindung mit der Möglichkeit, verschiedene Token zur Authentifikation und Authorisierung im Rahmen des SOAP-Headers zu transportieren.

Kapitel 10

Content-Level Security

Der Bereich der Content-Level Security unterscheidet sich von der allgemeinen Sicherheitsarchitektur einer Firma, da er weniger die Autorisierung von Operationen oder Methoden im Blickpunkt hat, sondern die sicherheitsrelevanten Eigenschaften von Dokumenten – das heißt, öffentlichen wie vertraulichen. Selbstverständlich unterliegen auch öffentliche Dokumente einer Sicherheitspolicy: Zumindest ihre Integrität sollte gesichert werden, damit kein Imageschaden entsteht.

Die meisten Firmen besitzen neben Applikationen zur Verwaltung operationaler Daten (Kunden, Bestellungen etc.) große Mengen unstrukturierter oder semi-strukturierter Daten, im Allgemeinen als *Dokumente* bezeichnet. Diese Daten sind oft sowohl vertraulich als auch öffentlich – allerdings zu unterschiedlichen Zeitpunkten. Sie sind hochgradig mobil, d. h. sie werden unter bestimmten Umständen und zu bestimmten Zeitpunkten anderen Personen zugänglich gemacht. Ihre Korrektheit hat rechtliche und finanzielle Gründe.

Content-Level Security meint einmal die korrekte Erzeugung und Verwaltung von Dokumenten inklusive des Publizierens. Die Sicherheitsfragen, die hier auftauchen, beziehen sich auf die korrekte Authentisierung von Publishern, die Verwaltung von Sites und generell auf die mit der Erzeugung von Inhalten befassten Rollen. Zugriffskontrolle findet also hauptsächlich über Rollen und zusätzlich über Rechte an einzelnen Instanzen von Dokumenten oder Sites statt.

Auch im eingangs geschilderten Portal der Bahn AG war eine Anforderung, dass die Portalinhalte sicher verwaltet werden können. Dies bedeutet ganz konkret, dass nur autorisierte Personen Inhalte erzeugen und publizieren können und dass der Verlauf der Bearbeitung eines Dokumentes nachvollzogen werden kann. Dies bedingt im Allgemeinen den Einsatz eines Content-Management-Systems (CMS). In bestimmten Fällen ist auch die Anzeigezeit eines Dokumentes auf einem öffentlich zugänglichen Kanal recht wichtig (wie lange wurde Angebot X beworben?). Im ersten Teil dieses Kapitels wollen wir wichtige Trends im Bereich semi-strukturierter Daten und ihren Einfluss auf die Sicherheit der Systeme aufzeigen. Danach werden wir die dahinter stehenden Konzepte der Zugriffskontrolle darstellen, soweit sie für die Software bzw. Systemarchitektur wichtig sind. Dies betrifft insbesondere auch den Unterschied zwischen Objektbasierter und Prozessbasierter Sicherheit.

10.1 Aktuelle Trends

Der Bereich der unstrukturierten Daten in einer Firma umfasst weit mehr als nur die als „geheim“ eingestuften Dokumente, die nur einem bestimmten Personenkreis zugänglich sein dürfen. Mit Systemen zur Verwaltung hochsicherer Dokumente können die meisten Firmen auch nicht aufwarten.

Mit den elektronischen Kanälen Internet und Intranet haben sich sehr viele Dokumente verlagert in elektronische Medien, die vielen Leuten einen sehr schnellen Zugriff erlauben. Sobald etwas einmal auf einer Homepage publiziert ist, wird es auch gelesen und Leser orientieren sich in ihrem finanziellen Verhalten daran.

Gleichzeitig – und das ist wahrscheinlich der bedeutsamste Trend – ist die rechtliche Relevanz von Firmendokumenten (seien es Homepage Inhalt, Intranet oder Mail-Verzeichnisse der Mitarbeiter) enorm gestiegen: Kaum ein Gerichtsverfahren, bei dem nicht Inhalte beschlagnahmt werden oder sich der Mailverkehr zwischen Mitarbeitern als wichtiges Argument für die Gegenseite herausstellt.

Zusätzlich haben Gesetzgeber die Verpflichtung der Firmen und ihrer Manager zum sauberen und nachvollziehbaren Umgang mit Dokumenten erheblich verschärft – als Folge der Bilanzskandale in den USA und Italien. So droht zum Beispiel der Sarbanes-Oxley Act [SOX] Managern sogar Gefängnis an im Falle einer Verletzung der Regelungen, und Grosskonzerne investieren zweistellige Millionenbeträge um die so genannte „Compliance“ zu erreichen.

Aber Relevanz und Compliance sind noch längst nicht alles. Viele Firmen kaufen multi-mediale Inhalte wie Bilder, Audio und Video ein und verwenden sie auf verschiedenen Kanälen. Die jeweiligen Inhaber der Rechte möchten natürlich ihre Tantiemen erhalten, die oft von der Art der Verwendung und ihrer Häufigkeit abhängen. Dafür ist ein System nötig, das die Rechte für den Zugriff und die Verwendung digitaler Inhalte regelt und verfolgt, nämlich ein so genanntes Digital Rights Management (DRM) System.

Gleichzeitig muss das System aber auch sicherstellen können, dass nicht unnötig Tantiemen gezahlt werden und dass bezahlter Inhalt möglichst auch mehrfach verwendet werden kann. Medien-Datenbanken und Medien-Logistik-Systeme sind auf die Wiederverwendung und korrekte Abrechnung medialer Inhalte spezialisiert (vgl. [Kretz]).

Neben Fragen der Compliance waren es jedoch auch veränderte Business Modelle, die für grossen Druck auf die Verwaltung von Dokumenten gesorgt haben: Collaboration und Service Orientierung. Service Orientierung werden wir im Bereich Application Server und Web Services genauer behandeln, hier wollen wir uns stattdessen eher auf die Zusammenarbeit zwischen Personen unterschiedlicher Firmen an Dokumenten konzentrieren. Dazu ist nötig, dass externe wie interne Personen schnellen Zugriff auf bestimmte Dokumente erhalten können. Anderseits muss sichergestellt sein, dass der Zugriff autorisiert, zielgerichtet und vor allem auch terminiert ist.

Das bedeutet, dass der Zugriff auf Dokumente nicht mehr unbedingt durch feste Rollen geregelt wird, sondern durch spontane Zuteilung von Personen auf

Themen. Dem Thema werden dann Regeln für den Zugriff zugeordnet. Dadurch entsteht ein nicht-hierarchisches Element, das zu Problemen in der Rechteverwaltung führen kann, falls diese bisher rein hierarchisch ausgelegt war.

Weitere Probleme in kollaborativen Umgebungen sind „work-in-progress“ Dokumente, die dem Willen des momentanen Autoren nach anderen Autoren – selbst in der gleichen Gruppe – noch nicht zugänglich sein sollen. Schließlich benötigen Content Management Systeme heutzutage die Fähigkeit, nicht nur ganze Dokumente sondern auch Fragmente von solchen verwalten zu können – am Besten durch Verlinkung und nicht durch Kopieren. Aber dadurch entstehen diffizile Probleme der Rechtevergabe und Verwaltung: Ein Dokument D nutzt ein Fragment eines anderen zum Zeitpunkt A, da Publisher X das Recht darauf hatte. Was passiert später bei einem Update dieses Fragments? Muss Dokument D ebenfalls angepasst werden? Darf es überhaupt automatisch angepasst werden? Müssen die Rechte der beteiligten Publisher neu geprüft werden?

Flexibilität, Geschwindigkeit und Sicherheit sind keine natürlichen Partner, wie man hier schnell erkennen kann. Solche sich widersprechende Forderungen bereiten den meisten Systemen zur Verwaltung von Dokumenten enorme Probleme.

Auch die Systeme selbst stellen ein weiteres Problem dar: Es sind sehr viele und die dafür benötigte Infrastruktur aus Maschinen und Wartung frisst immer grösere Teile des IT-Budgets einer Firma. Deshalb geht ein weiterer Trend zu einer Vereinfachung und Konsolidierung von Content Management Infrastrukturen, das bedeutet eine Reduktion auf wenige Systeme, die dann aber gleichzeitig international arbeiten können (Zweigstellen und Partnerfirmen).

Eine Reihe sehr interessanter Entwicklungen im Content Bereich kann hier leider nur kurz erwähnt werden. Darunter fällt die ad-hoc Bildung von Organisationen angesichts von Naturkatastrophen, die die Sicherheit von Dokumenten in Bezug auf Verfügbarkeit, Routing und Änderungen vor große Probleme stellen. Aber auch die Zusammenarbeit vieler Content-Ersteller im Rahmen von Web2.0 Lösungen (Wikis etc.) wirft neue Fragen der Sicherheit auf, wie auch das Publizieren von Dokumenten in Peer-to-Peer Netzen. Hier bestehen teilweise widersprüchliche Anforderungen bzgl. des Schutzes vor Zensur einerseits und der Möglichkeit des Rückrufs von Dokumenten bzw. von nachträglichen Änderungen durch den Autor andererseits.

10.2 Architektur und Infrastruktur

Verwaltet werden Dokumente und multimediale Inhalte überwiegend in so genannten *Content Management Systemen* (CMS), die teilweise auch bereits das Digitale Asset Management von Bildern, Filmen etc. einschließen, dieses aber häufig auch an spezialisierte DAMS (Digital Asset Management Systeme) delegieren. Viele der CMS Produkte sind in den letzten Jahren zu Enterprise CMS Produkten geworden, die von der Unterstützung für Marketingaktionen über

Records-Mangement bis hin zum automatischen Update verteilter Sites in globalen Firmen alles beinhalten.

Der Umgang mit Dokumenten in einer Firma lässt sich momentan grob in die Bereiche Office Applikationen, Content Management und Auslieferung über verschiedene Kanäle aufteilen. Die Funktionen eines Enterprise CMS umfassen die folgenden Punkte:

- Capture (authoring, integration),
- Deliver (syndication, transformation),
- Manage (workflow, records, web content, collaboration),
- Store (versioning, retrieval) und
- Preserve (archive, media).

Die großen ECMS Systeme sind durch Zusammenkaufen der verschiedenen Funktionalitäten entstanden, deren Integration große Schwierigkeiten bereitet. Trends wie die Betonung des Records-Managements aus rechtlichen Gründen haben weiteren Druck auf die Sicherheit dieser Systeme ausgeübt. Die meisten der Systeme sind übrigens J2EE basierte Architekturen.

Trotz der Entwicklung von ECMS verfügen größere Firmen heutzutage häufig über eine ganze Reihe verschiedener CMS Systeme, die gleichzeitig im Einsatz sind und unterschiedliche Organisationsteile, Inhalte oder Zielgruppen bedienen.

10.2.1 Infrastruktur

Auffällig an Content-Management Lösungen innerhalb von Firmen ist oft die Vielzahl der beteiligten Systeme – einer der Gründe für die oben erwähnten Kostenprobleme. Was sind die Gründe für diese Aufteilung? Neben historischen Gründen spielt häufig der Sicherheitsgedanke dabei eine Rolle. Betrachtet man zum Beispiel den Internet-Auftritt einer Firma, so stellt man fest, dass die dort publizierten Inhalte häufig eine Kopie von Teilen des Intranets der Firma darstellen, die lediglich durch ein unterschiedliches System ausgeliefert werden. Erstaunlicherweise werden die Dokumente jedoch oft tatsächlich in zwei verschiedenen Content Management Systemen gehalten und von einem ins andere kopiert.

Hier wird ein Sicherheitsprinzip deutlich, nämlich das der getrennten Sicherheitszonen oder *Perimeter-Security*. Das bedeutet: Daten und Funktionen, die nicht vermischt werden dürfen, werden in getrennten Zonen (Systemen) gehalten. Die Eingänge zu diesen Zonen werden überwacht. Das Prinzip ist durchaus erfolgreich: Daten, die nicht auf einem System sind, können auch nicht durch falsche Zugriffsrechte exportiert werden. Berechtigungen für die Systeme können durch unterschiedliche Rollen sicher verwaltet werden.

Wie erfolgreich dieses Prinzip der Sicherheitsbereiche funktioniert, lässt sich am Beispiel der Search-Funktionalität einer Firmen-Website zeigen: Wenn Intranet- und Internet-Dokumente auf getrennten Systemen lagern, dann lässt sich ein Search Mechanismus sehr einfach durch einen so genannten *Spider Mechanismus*

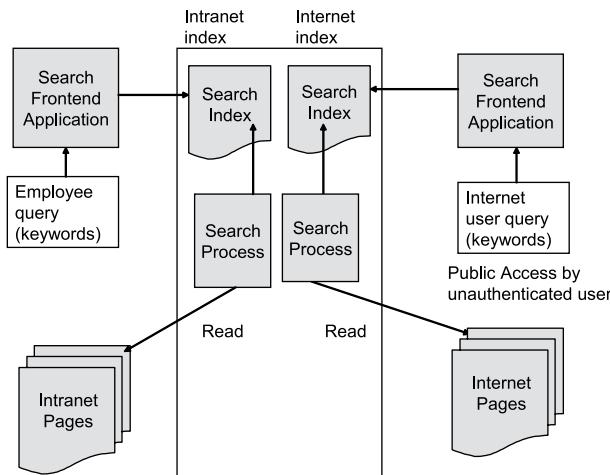


Abb. 10.1 Getrenntes Durchsuchen von Intranet- und Internet-Dokumenten

implementieren. Beide Sites, Intranet und Internet, werden getrennt „gespidert“, d. h. jede Seite wird geladen, verarbeitet und auf weitere Links untersucht (s. Abb. 10.1). Die Ergebnisse bilden zwei getrennte Indizes, die nun auf die jeweiligen Sites geladen werden und dem Search-Mechanismus zur Verfügung stehen.

Da die Internet-Inhalte getrennt indiziert wurden, besteht keine Gefahr, dass Inhalte des firmeninternen Intranets aus Versehen ebenfalls durch die Search-Engine veröffentlicht werden. Dass dies eine reale Gefahr ist, hat nicht zuletzt erst kürzlich Cisco bewiesen. Dort wurden Passwörter durch die Search-Engine des Webportals veröffentlicht. Hier besteht diese Gefahr zumindest im Search Bereich nicht, da der Spider nur auf getrennten Inhalten operieren kann.

Getrennte Sicherheitsbereiche sind einfach in der Installation wie auch in der Verwendung durch die Benutzer, da sie klar umrissene Bereiche bilden. Und sie sind auch heute noch essentiell z. B. in der Absicherung von Netzwerken. Leider haben sie jedoch auch einige massive Nachteile, nämlich:

- Hohe Infrastrukturstarkosten,
- Hohe Wartungskosten,
- Mangelnde Flexibilität und
- Gefahr bei bereichsübergreifenden Aktionen.

Die Infrastrukturstarkosten sind durch die doppelten Systeme unmittelbar einleuchtend. Weniger klar sind die hohen Wartungskosten und die mangelnde Flexibilität. Die Wartungskosten entstehen unter anderem durch die Notwendigkeit der doppelten Datenhaltung: Ein Dokument, das sowohl für das Intranet als auch das Internet gültig ist, muss doppelt gehalten werden. Im Falle von Änderungen besteht also immer die Gefahr, dass nur eine Version angepasst wird.

Die mangelnde Flexibilität wird sichtbar, wenn wir folgendes Szenario annehmen: Eine Mitarbeiterin der Firma möchte von ausserhalb die Search-Funktionali-

tät der Firmen-Website nutzen. Da sie Mitarbeiterin ist, möchte sie jedoch die Intranet-Version der Site nutzen und selbstverständlich auch die mächtigere Search-Funktionalität. In der obigen Architektur ist das nicht einfach möglich, da diese Daten überhaupt nicht auf der Internet-Site vorhanden sind. Jetzt stellen wir fest, dass wir mit dem Zonenkonzept zwei sehr unterschiedliche Dinge verknüpft haben: Datensicherheit und Auslieferungskanäle. Stillschweigend sind wir davon ausgegangen, dass die Internet-Site öffentliche Daten beinhaltet, während die Intranet-Site vertrauliche Dokumente der Firma enthält. Das Zonenkonzept scheint nicht in der Lage zu sein, die Daten mit der richtigen Granularität abzusichern. Dan Geer hat dieses Phänomen in [Geer] als „The Shrinking Perimeter“ bezeichnet. Um die Daten entsprechend ihrer Mobilität und Sicherheitsstufe richtig behandeln zu können, müssten die Schutzzonen immer kleiner werden, bis sie nur noch die jeweiligen Daten kapseln. Wahrscheinlich wäre es sinnvoller gewesen, eine richtige Datenklassifizierung (Tagging) durchzuführen, z. B. in die Label

- öffentlich,
- firmenintern,
- vertraulich und
- geheim.

und diese Klassifizierung bei den Daten zu halten, anstatt einfach Daten mit spezifischen Auslieferungskanälen gleichzusetzen und sie praktisch als Instanzen auf Kanäle aufzuteilen.

Der Zugriff von Mitarbeitern von ausserhalb ist ein schönes Beispiel dafür, wie ein einfaches, verständliches und deshalb meist auch sicheres Zonenkonzept mit der organisatorischen Flexibilität heutiger Arbeitsprozesse in Konflikt gerät. Denn zusätzlich ist das Konzept gleichzeitig auch sehr grob. Es gehört damit zu einer ganzen Klasse von Isolationsmechanismen, die zur Zugriffskontrolle verwendet werden. Ein weiteres Beispiel bilden die unterschiedlichen Modi in Betriebssystemen (Privileged/Unprivileged), deren Vor- und Nachteile im Kapitel zu Plattform-Sicherheit diskutiert werden. Letztlich gliedert der Mechanismus die Zugriffsrechte nach groben räumlichen, modalen oder temporalen Kriterien.

Selbst wenn wir jetzt schnell einen Authentisierungsmechanismus im Portal einführen – damit sich die Mitarbeiter anmelden können – nutzt das nichts, da die gewünschten Daten auf dem System selbst nicht verfügbar sind. Das Gleiche gilt natürlich auch für Mitarbeiter von anderen Firmen, die an gemeinsamen Projekten teilnehmen. Damit erfüllt unsere Publishing und Content Management Architektur die obigen Grundforderungen von oben, wie etwa die Unterstützung von Kollaboration, nicht.

Das Grundproblem scheint die mangelnde Granularität unseres Sicherheitssystems zu sein. Die Zonen bieten zwar Sicherheit für die Dokumente, aber sie tun das zu grob und unflexibel. Anders ausgedrückt: Sicherheit und Daten sind zu weit getrennt voneinander.

Bisher haben wir im Rahmen der Content-Level-Security drei Mechanismen der Zugriffskontrolle erwähnt: Rollen und Instanzen, Daten-Tagging und jetzt auch

Isolation durch modale Konzepte. Alle drei Mechanismen können durchaus gleichzeitig und nebeneinander in CMS Software und Infrastruktur verwendet werden.

10.2.2 CMS Sicherheitsarchitektur

Das Content Management kennt zwei grundsätzliche Arten von Benutzern: Solche, die Content produzieren und solche, die ihn konsumieren. Entsprechend teilt sich auch die Sicherheit dieser Systeme in diese beiden Bereiche auf (die etwas dickere Linie in Abb. 10.2). Das Diagramm zeigt grob ein Operational Model eines Internet Publishing Systems, basierend auf einem CMS. Die Netznoten sind Bestandteile der Firmeninfrastruktur und dienen hauptsächlich der Authentisierung und Autorisierung von Editoren wie Besuchern. Die waagrechten Striche zeigen unterschiedliche Sicherheitszonen an, die durch den Einsatz von Firewalls realisiert werden können. Im Zentrum jedoch stehen die Ressourcen – hier die Dokumente und die Teile, die das CMS verwaltet.

Welche Sicherheitsbedürfnisse haben nun die Dokumente? Das hängt natürlich ganz von ihrem Inhalt ab. Häufig treffen Firmen jedoch eine Entscheidung, die sich am Zonenkonzept orientiert: Dann wird entschieden, dass das CMS X nur öffentliche oder firmeninterne Daten halten darf. Die Publisher sind dann angewiesen, dies

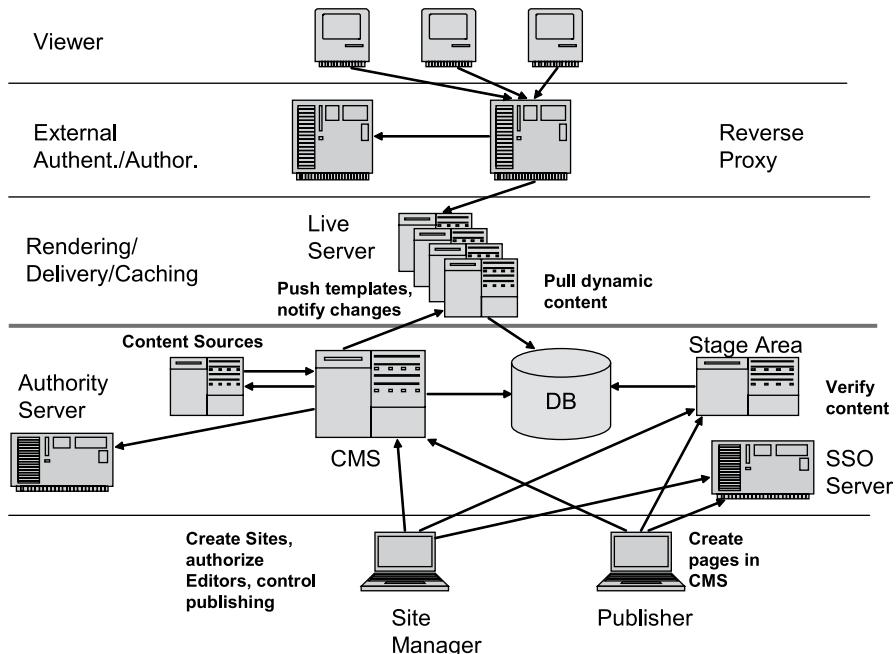


Abb. 10.2 Operational Model eines Internet Publishing Systems

zu respektieren. Wichtig daran ist, dass die Klassifikation der Dokumente damit implizit erfolgt und die Einhaltung der Sicherheit an die beteiligten Personen delegiert wird. Das System selbst kennt das Konzept „öffentlich“ oder „intern“ nicht. Am Beginn eines CMS steht daher die Definition und Klassifikation der Ressourcen, die es zu verwalten gibt. Bei einem CMS sind häufige Ressourcen:

- Dokumente und Fragmente,
- Container-Ressourcen wie „Site“,
- Navigation,
- externe Dokumente,
- Publishing Plattform und
- Systemressourcen

Von Seiten des Software-Designs ist hier anzumerken, dass es sehr hilfreich ist, wenn diese Ressourcen über ein einheitliches Sicherheitsmodell unter Verwendung von Vererbung und Polymorphie verfügen. Dies erleichtert es den Entwicklern später ungemein, die dahinter steckenden Regeln auch in der Implementation korrekt umzusetzen.

Den Typen von Ressourcen werden nun meist spezielle Verwaltungsrollen zugeordnet, z. B.

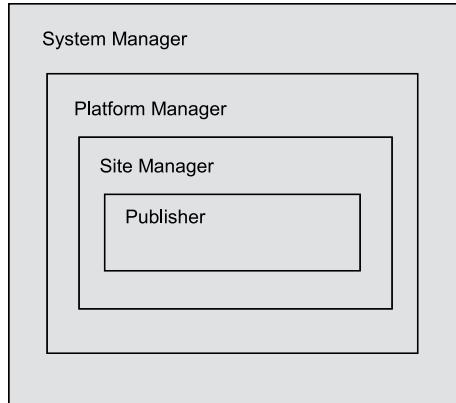
- Publisher (dürfen Dokumente erstellen und verändern);
- Sitemanager (dürfen Dokumente erstellen, verändern sowie zur allgemeinen Ansicht freigeben);
- Plattform-Manager, die eine ganze Reihe sog. Sites erzeugen, verwalten sowie Dokumente lesen und verändern können und
- System-Manager, die vollen Zugriff auf Dokumente und Systemkonfigurationen (auch die Security) besitzen und Rollen an bestimmte User vergeben können. Diese Verwaltungsrolle kann explizit modelliert oder implizit über Betriebssystem-Rechte durchgeführt werden (z. B. durch direkten Zugriff auf die Datenbank).

Schnell sieht man, dass dahinter ein hierarchisches Konzept von Berechtigungen durch Rollen steckt, etwa in dieser Form (s. Abb. 10.3):

Jede übergeordnete Rolle besitzt alle Rechte der untergeordneten Rollen. In dieser Form ist die Verwaltung der Berechtigungen extrem einfach, sowohl im Verständnis der Benutzer als auch für die Entwickler bei der Implementation. Sie bringt aber sicherheitstechnisch wie auch für den Betrieb des Systems etliche Probleme mit sich:

- Business und Systemadministration teilen sich Zugriffe auf Ressourcen.
- Die Businessrollen haben keine Erlaubnis zur Selbstadministration ihrer Bereiche in Bezug auf die Vergabe von Rechten (Rollen).
- Dokumente verschiedener Vertraulichkeitsstufe sind schwierig zu handhaben, da übergeordnete Rollen alles lesen und ändern können.
- Bereits die „niedrigste“ Rolle hat das Recht, beliebige Dokumente zu erstellen oder andere abzuändern.

Abb. 10.3 Hierarchisches Rollenkonzept in einem CMS



- Die Vergabe der Rechte hängt an einem allmächtigen Systemverwalter, der wiederum die Situation im Business nicht kennt.
- Ein Benutzer könnte nicht mehrere Rollen erhalten, da streng genommen nur die „höchste“ Rolle für einen Benutzer von Bedeutung wäre. Eine Aufteilung der Berechtigungen pro Ressource ist daher nicht möglich.

Das bedeutet, das einfache Modell reflektiert weder Business- noch Sicherheitsbedürfnisse. In einem ersten Schritt muss deshalb ein Konzept von „Ressource-Instanz“ eingeführt werden: Ein Publisher darf demnach nur die Dokumente bearbeiten, die er selber angelegt hat bzw. auf denen er berechtigt wurde. Dazu muss ein Publisher als möglicher Publisher innerhalb einer Site-Instanz eingetragen werden. Diese Eintragung wiederum kann ein Sitemanager vornehmen.

Jetzt entsteht ein Rollenkonzept, das nicht mehr strikt hierarchisch ist, sondern eines, in dem alle Ressource-Instanzen eine zusätzliche Access Control List erhalten haben. In dieser ACL stehen dann für jedes Dokument die zulässigen Publisher. Was passiert dabei mit den Rollen? Das CMS benötigt ja weiterhin den Unterschied zwischen der Verwaltung einer Site oder eines Dokuments, also müssen die Rollen selbst erhalten bleiben. Ändern muss sich jedoch die Art und Weise der Zuordnung zu Usern, und hier sind verschiedene Szenarien möglich (Eine sehr gute Übersicht über Rollenmodelle und deren Zusammenhang mit Berechtigungen auf Instanzebene findet sich bei [Eck], Seite 114 ff.).

Es kann durchaus mehr als eine Rolle pro User geben, jeweils in Verbindung mit bestimmten Ressourcen. Zum Beispiel kann ein User in Bezug auf eine Ressource mehrere Rollen einnehmen, aber dies nur nacheinander (z. B. kann ein User ein Dokument erstellen in der Rolle *Publisher* und danach das gleiche Dokument in der Rolle *Sitemanager* dann veröffentlichen, s. Abb. 10.4). Dazu muss sich der User jedoch neu am System (in seiner neuen Rolle) anmelden, oder das System erlaubt die spontane Kombination der aus den Rollen resultierenden Rechte.

Entscheidend ist hier, wie „nacheinander“ sicherheitstechnisch interpretiert wird. Wenn damit erreicht werden soll, dass die Ausübung der zweiten Rolle bewusst wahrgenommen wird, dann muss die Benutzerführung dies sicherstellen –

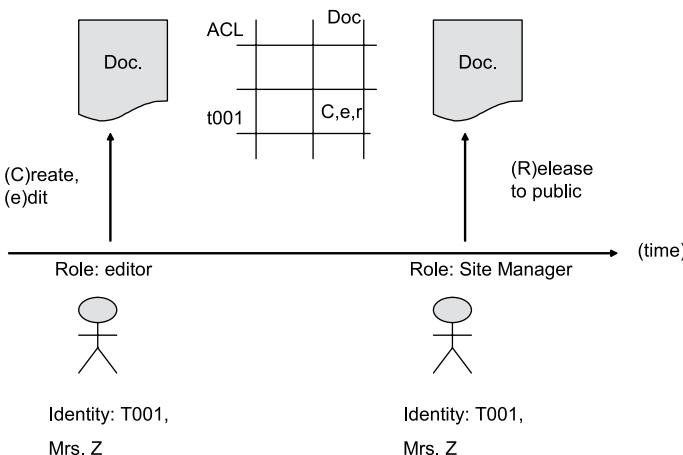


Abb. 10.4 Rollenwechsel eines Users für unterschiedliche Aktivitäten

entweder durch erneuten Login mit neuer Rollenwahl, mit GUI Änderungen (Farbwahl) oder durch andere Techniken. Die blosse Kombination der Rechte als Summe der Rollen in eine ACL repräsentiert den Sicherheitsunterschied nicht.

Als Konsequenz muss nun allerdings die Verbindung eines Benutzers und einer Instanz gespeichert werden, damit sie später für die Entscheidung bei der Access Control verwendet werden kann. Immer noch ungelöst sind aber die folgenden Probleme:

- Wer vergibt die Rechte bzw. Rollen und die zulässigen Instanzen?
- Wie werden die Rechte vergeben? Einzeln in den ACLs oder automatisch per Rolle?
- Wie kann man vertrauliche Instanzen innerhalb einer Site erzeugen, die nicht von übergeordneten Rollen gelesen werden können? Wie kann also ein Durchbrechen der hierarchischen Rechtevergabe realisiert werden?
- Wie können im Rahmen von Notfällen Ausnahmeregelungen eingeführt werden, ohne dass die Sicherheitskonzepte völlig zusammenbrechen?

10.2.3 Abbildung der Berechtigungen auf das Sicherheitsmodell der Firma

Im letzten Abschnitt wurde eine Access Control List pro Dokument oder Site Ressource eingeführt. In dieser ACL können die Identitäten von Usern gehalten werden, einschliesslich der jeweiligen Operationen, die diesen Identitäten erlaubt sind. Jetzt stellt sich die Frage: Brauchen wir dann überhaupt noch Rollen?

Rollen sind – wie weiter oben definiert – Sets von Rechten, die dann wiederum Benutzern oder Gruppen von Benutzern zugeordnet werden können. Dies ist eine

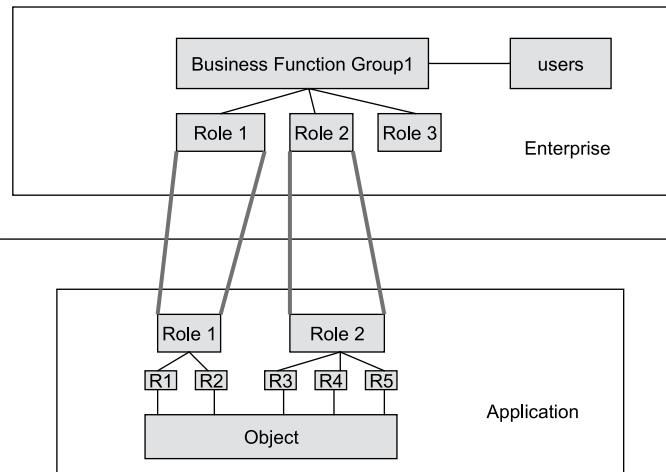


Abb. 10.5 Top-Down versus Bottom-Up Definition von Rollen

Bottom-up Sicht auf Rollen, bei der von der Ressource ausgehend sinnvolle Sets von Rechten definiert werden, die dann Arbeitseinheiten bilden. Hier sind es die Entwickler, die diese Art von Rollen definieren. Es ist jedoch auch möglich, die Rollen „Top-Down“ zu definieren, das heißt, eine Firma identifiziert für ihre Mitarbeiter typische Aufgaben und definiert damit firmenspezifische Rollen. Hier ist es die Organisation, die diese Rollen definiert, unabhängig von konkreten Applikationen (s. Abb. 10.5).

Wir werden im Folgeband im Zusammenhang mit Enterprise-Security-Fragen auf diese beiden Definitionsmöglichkeiten zurückkommen und untersuchen, wie sie in J2EE/EJB Architekturen aneinander angepasst werden können. Dabei werden wir auch der Frage nachgehen, wie unabhängig beide Definitionsformen tatsächlich voneinander sind, bzw. wie die Interfaces gestaltet sein sollten, damit flexible Zuordnungen zwischen Rechten und Rollen möglich sind. Die Gefahr dabei ist, dass auf der untersten Ebene in der Software bereits Konzepte von Rollen als „Tasks“ so eingeführt werden, dass sie sich nur schwer an das Rollenmodell des Unternehmens anpassen lassen. Mit einem schlichten Namens-Mapping ist es hier nicht getan.

Zurück zur Ausgangsfrage: Brauchen wir noch Rollen, wenn wir ACLs haben? Dazu müssen wir uns ansehen, welche Funktion nun eine Rollendefinition besitzt. Der Besitz einer Rolle ist eine Abkürzung für ein Set von Rechten und damit eine Vereinfachung der Organisation durch Abstraktion. Statt mühsam einzelne Rechte pro Applikation zu vergeben, können einer firmenspezifischen Rolle mehrere technische Rollen zugeordnet werden. Der Besitzer der firmenspezifischen Rolle erhält dann automatisch alle Rechte der technischen Rollen. Für die Organisation erleichtert das nicht nur die Rechteverwaltung, es erlaubt den Verantwortlichen im Business auch auf einer für sie semantisch verständlichen Ebene (den firmenspezifischen Rollen) Rechte zu vergeben, ohne sich auf einer tieferen Ebene mit technischen Rechten auseinandersetzen zu müssen. Wichtig ist es hier zu erkennen, dass

die Rollendefinition ähnlich wie im Softwaredesign aus verschiedenen Stufen bestehen kann mit jeweils ganz unterschiedlichen Sprachen und Usern (Business vs. Technik).

Es ist also schlicht eine Frage der Verwaltbarkeit von Rechten, wenn nicht User einzeln in ACLs eingetragen werden, sondern nur die benötigten Rollen. Aber Vorsicht: Die ACL selbst kann durchaus die einzelnen berechtigten User beinhalten, unter einer Bedingung: Der Grund der Einfügung eines Users direkt in eine ACL im Business-Modell bzw. im Sicherheitsmodell der Firma muss festgehalten werden, d. h. der Zusammenhang zwischen Sicherheitsmodell und dem ACL Eintrag ist vorhanden. Unter diesen Umständen bleibt die Möglichkeit, eine Rolle und/oder die einzelne Berechtigung wieder zu entziehen, auf der Business Ebene bestehen. Ohne ein Modell der Berechtigungen führt die einzelne Eintragung in eine ACL sehr schnell dazu, dass nach einiger Zeit niemand mehr nachvollziehen kann, wieso gewisse Rechte vorhanden sind bzw. wie lange sie eigentlich bestehen bleiben sollten.

Wir halten also fest, dass es zwischen Rollen und einzelnen Usern in ACLs einen großen Unterschied bezüglich der Granularität der Rechtedefinition gibt. Im Einzelfall kann die Möglichkeit, einen einzelnen User direkt bezüglich einer Ressource zu autorisieren, indem er ohne Berücksichtigung einer firmen- oder applikationspezifischen Rolle direkt in die ACL eingetragen wird, von Vorteil sein. Als Dauerzustand wird diese Möglichkeit jedoch zum Alptraum: Denken Sie an die Probleme der Rücknahme von Rechten bei der Reorganisation von Verantwortlichkeiten oder Arbeitsgebieten von Seiten der Firma.

Kommen wir zur nächsten Frage: Wer stellt die Verknüpfung zwischen Usern und ihren Rollen bzw. zwischen Usern und ihren Ressource-Instanzen her? Innerhalb einer Firma gibt es verschiedene Rechtsbereiche mit unterschiedlichem Gültigkeitsbereich. Zuerst kommen die firmenglobalen Regelungen, dann die Regelungen innerhalb verschiedener Geschäftsbereiche und der Technik und zuletzt Regelungen innerhalb von Applikationen. Über das Ausmass des jeweiligen Rechtsbereichs entscheidet die Sicherheitspolicy der jeweiligen Firma, zum Beispiel welche globalen Rollen es gibt, und wer diese Rollen zuweisen und verwalten darf. Zentralistische Ansätze erlauben nur die zentrale Vergabe von Rollen und Rechten. Dies erlaubt es den Firmen, organisatorische Änderungen leichter durchzuführen. Andererseits besteht nicht zuletzt im Bereich Content Management ein starker Trend in Richtung größerer Selbständigkeit der Geschäftsbereiche in Bezug auf Verwaltung ihrer Dokumente, das heisst die Geschäftsbereiche möchten selber festlegen, wer welche Rolle besitzen kann und auf welchen Instanzen Berechtigungen erhält. Die Rechteverwaltung erfolgt dabei meist in Anlehnung an die innere Organisation des Geschäftsbereichs. Das Stichwort hier ist *Service Management Delegation* und bedeutet die Delegation der Rechteverwaltung an die einzelnen Einheiten.

Service Management Delegation bedeutet letztlich nichts anderes, als dass höhere Verwaltungsrollen im Content Management einzelnen Usern selbstständig bestimmte Rollen oder Recht einräumen können. Business-Rollen erhalten die Rechte zur Rechtevergabe an weitere User innerhalb ihres Zuständigkeitsberei-

ches, der durch die ACLs der Ressourceninstanzen definiert ist. So kann ein Site-Manager von sich aus einem User X das Recht als Publisher auf der Dokumenteninstanz Y einräumen. Plattform-Manager hingegen können weitere Site-Manager auf ihren Sites definieren.

Softwaretechnische Voraussetzung für Service Management Delegation ist die komplette Modellierung der Administrationstätigkeiten und der entsprechend notwendigen Rechte innerhalb der Applikation. Es müssen Rollen geschaffen werden, die ihrerseits das Recht haben, Benutzern gewissen Rollen sowie gewisse Instanzen zuzuordnen. Dies bedeutet jedoch nicht, dass kein externes Berechtigungssystem verwendet werden könnte. Die Rollenzuordnung auf Firmenebene wird dadurch weiter konkretisiert – kann allerdings auch auf den Kopf gestellt werden, indem Benutzer mit geringen Firmenrollen Berechtigungen für sensible Dokumente eingeräumt werden. Daher ist die Vergabe solcher Rollenzuordnungen selbst eine sehr sensitive Operation und wird deshalb häufig durch zentrale Systeme kontrolliert und protokolliert.

Interessant ist an dieser Stelle auch, welche Rollen der IT-Abteilung in diesem Rollenkonzept zugeschlagen sind. Typischerweise benötigt ein größeres Content Management System einige Service Techniker, die das System technisch betreuen können. Welche Rollen sollen diese Personen erhalten? Bisher haben wir uns auf Geschäftsräume konzentriert mit der Aufgabe der Verwaltung von Sites und Dokumenten. Jetzt geht es um die technische Betreuung: Ist sie unabhängig von Geschäftsbereichen oder orthogonal dazu? Mit anderen Worten, kann der Service Techniker alle Sites bearbeiten oder ist er ebenfalls Geschäftsbereichen zugeordnet und damit auf bestimmte Instanzen von Sites und Dokumente beschränkt? Gelten für ihn die gleichen Regeln in Bezug auf den Zugriff auf Dokumente wie für die Sachbearbeiter? Diese Problematik, die letztlich auf das Konzept der „Stellvertretung“ hinausläuft, wollen wir im Abschnitt zu „Geschäftsprozessen“ weiter unten diskutieren.

Ein Beispiel für eine mögliche Aufteilung der Rollen und Rechte im Bereich Content Management: Die Firma vergibt globale Rollen in Abhängigkeit von der geschäftlichen Funktion der Benutzer. Diese Rollen werden dann auf die technischen Rollen der Content Management Applikation abgebildet, und zwar in erster Linie die wichtigen Verwaltungsrollen zur Verwaltung größerer Container wie Site oder Plattform. Auf Firmenlevel werden dann diese Rollen an bestimmte User vergeben, die dadurch Verwaltungsfunktionen im Content Management einnehmen können.

Problematisch ist hier nur die Behandlung der Ressource-Instanzen: Welche Rolle darf welche Site bearbeiten? Dazu gibt es grundsätzlich zwei Ansätze: Entweder wird die Verbindung Rolle-User-Instanz auf globaler Ebene festgehalten in einem sog. Benutzerberechtigungssystem (BBS), oder die Verbindung wird innerhalb einer Applikation festgehalten.

Jedes Verfahren hat spezifische Vor- und Nachteile: Die zentrale Verwaltung der Instanzbindungen erlaubt eine zentrale Kontrolle der Rechte, wichtig zum Beispiel bei organisatorischen Änderungen der Benutzer und der nötigen Revokation von Rechten. Allerdings muss das zentrale System nun feingranulare Rechte-Instanzen, also Beziehungen pro Benutzer, speichern und verwalten können.

Bei der dezentralen Verwaltung haben wir natürlich die gegenläufigen Effekte, also hohe Flexibilität der Verwaltung und schnelle Reaktionen durch Selbstverwaltung. Das Prinzip der Selbstverantwortung der Geschäftseinheit bewirkt nicht zuletzt effektive und schnelle Speicherung der Bindungsdaten. In Verbindung damit erwächst aber das Problem der Rechteverwaltung bei globalen Änderungen sowie die Möglichkeit, beliebige Bindungen zu erstellen, auch wenn sie nicht den Firmenrichtlinien entsprechen. In den nächsten Abschnitten wollen wir die beiden Lösungsansätze genauer untersuchen.

10.2.4 Rollenmodellierung am Beispiel Portal Access Control

Zugriffsrechte im Portalbereich sind häufig ein komplexes Gebilde aus Rollenmodellen und Instanzberechtigungen auf einzelne Ressourcen. Die Berechtigung, eine Instanz zu verwenden (als Editor oder Manager), wird dabei nicht im Sourcecode der Applikation verankert (siehe Beispiel Kontozugriff), sondern soll extern verwaltet werden. Ein reines hierarchisches Rollenmodell wie oben beschrieben genügt nicht, da es durch seine hierarchische Natur Zugriffe auf viel zu viele Instanzen erlauben würde. Deshalb haben wir im obigen Beispiel auch die ACL bei einer Ressource eingeführt und daneben davon gesprochen, dass wir Managementrollen auf höheren Instanzebenen Containern zuordnen, die dann für alles in den Containern enthaltene ebenfalls gelten.

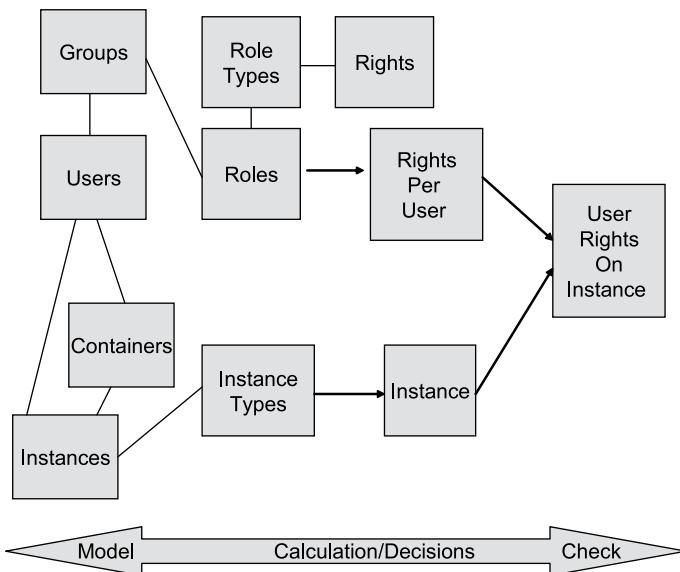


Abb. 10.6 Berechnung der Nutzerrechte in einem komplexen Rechtemodell

Ein schönes Beispiel für instanzbasierte Zugriffskontrolle auf Portalressourcen findet sich im IBM Portal Server Bereich (s. [Bueh]). Wie gewohnt unterscheiden wir dabei zwischen dem Bereich der Konfiguration und dem Bereich der tatsächlichen Zugriffskontrolle, bei der das nötige Recht abgeprüft wird. Während im ersten Fall die Klarheit der Modellbildung, die Flexibilität der Zugriffssteuerung aus Businesssicht und die Wartbarkeit der Rechtevergabe entscheidend sind, geht es im zweiten Fall um die nackte Performance bei der Durchsetzung der Entscheidung. Dazwischen steckt die eigentliche Entscheidungsfindung, die entweder dynamisch das Konfigurationsmodell auswertet oder statisch vorausberechnet wurde. Wir werden sehen, dass ein gutes Zugriffsmanagement alle drei Bereiche in einem Kompromiss verbinden muss.

Besonders schwierig wird die Berechnung des Zugriffs, wenn zusätzlich zu den hierarchischen Rollendefinitionen Rechte auf Instanzebenen definiert werden und diese sich ebenfalls über verschiedene Hierarchieebenen auswirken können, wie das Beispiel in Abb. 10.6 zeigt.

10.2.4.1 Konfigurationsebene:

Die Portal Server Lösung kennt auf Aggregatebene die folgenden Konstrukte:

- *Virtual Ressource*: Das sind Einstiegspunkte ins Portal wie etwa Portlets. Sie entsprechen im obigen Beispiel den „Sites“ in einem CMS.
- *Protected Ressource Hierarchy*: Die Weitergabe von Rollenzuordnungen der Ressourcen über Zugehörigkeit zu einer Container-Hierarchie (vgl. Abb. 10.7a und b). Es wird hierbei der Begriff „Inheritance“ verwendet, obwohl es sich eher um das Pattern „Environmental Acquisition“ handelt, denn die abhängigen Ressourcen sind meist nicht vom Typ der übergeordneten Ressource.
- *Role Types*: Sie bestimmen innerhalb des Portals eine Job Funktion (User, Administrator, Editor etc.). Sie entsprechen dem klassischen RBAC Konzept, jedoch mit einem wichtigen Unterschied: Sie allein gestatten keinen Zugriff auf Ressourcen, d. h. sie sind nicht direkt den Usern und Ressourcen zugeordnet.
- *Ownership Group*: Eine Menge von Usern, die gemeinsam eine Ressource besitzt und dadurch besondere Rechte auf der Ressource hat.

Auf Instanzebene sind die folgenden Begriffe definiert:

- *Roles*: Sie schaffen die Verbindung zwischen Usern/Gruppen und konkreten Instanzen, allerdings in Abhängigkeit von den obigen Role Types. Das bedeutet, ein User X, der in einer Rollenbeziehung vom Typ „Editor“ zum Dokument Y steht, kann dadurch indirekt auch in der gleichen Rollenbeziehung zu anderen Dokumenten stehen, die in der Protected Ressource Hierarchy unter dem Dokument Y stehen (s. Abb. 10.7a).
- *Ownership*: Mit der Rolle Ownership sind spezielle Rechte auf der Instanz verbunden, die dem Owner gehört.

- **Role Block:** Ein „Stopper“, der die Weitergabe von Rechten innerhalb einer Ressource Hierarchy stoppt, z.B. um ein bestimmtes Dokument speziell zu schützen. Das ist zum Beispiel praktisch für die Implementation eines Schutzes für Dokumente, die gerade in Bearbeitung sind.

Wichtig ist dabei, dass wir mit einer solchen Konfiguration die Rechte eines bestimmten Users in Bezug auf konkrete Ressourcen in einer möglichst normalisierten Form beschreiben. Dies vermeidet unnötige Einträge auf Instanzebene, die später zu Wartungsproblemen führen können, falls User Rollenänderungen erfahren oder die Besitzverhältnisse sich ändern. Der Kontrollmechanismus führt daher Optimierungen durch, wie in Abb. 10.7b gezeigt wird:

Die konkreten Zugriffsrechte werden daher erst berechnet, wenn eine Prüfung tatsächlich durchgeführt werden muss. Von welchen Dingen diese Berechnung der effektiven Rechte weiterhin abhängt, und welche Performanceprobleme dabei auftreten können, klärt der nächste Abschnitt.

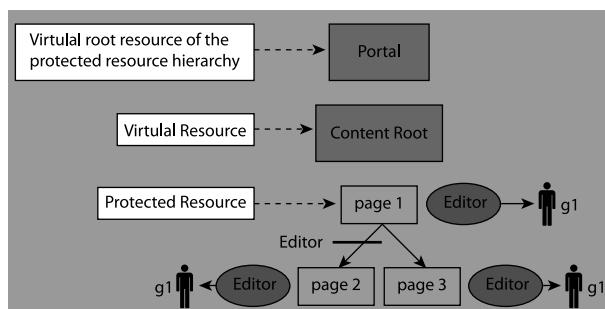


Abb. 10.7a Protected Resource Hierarchy (aus [Bueh]): Die Rolle „Editor“ überträgt sich auf untergeordnete Dokumente

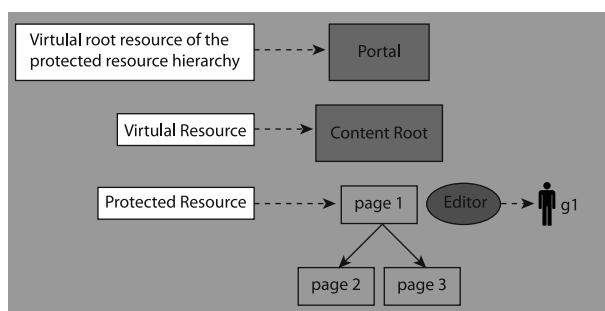


Abb. 10.7b Protected Resource Hierarchy (aus [Bueh]): Optimierung durch automatische Rechteübertragung auf untergeordnete Dokumente

10.2.4.2 Prüfungsebene

Auf Ebene der tatsächlichen Zugriffsprüfung müssen verschiedenste Daten korreliert werden. Für den jeweiligen User muss geprüft werden, welche Rollen definiert sind, auf Gruppen- wie auf Individuenebene. Daraus ergibt sich ein Set von Rechten. Pro Ressource muss nun geprüft werden, welche Rollen definiert sind, und zwar in der Hierarchie aufsteigend, da die einzelne Ressource möglicherweise gar keine Rechteinträge enthält, sondern sie vom Kontext (Position in der Hierarchie) bekommt. Dabei ist zu prüfen, ob irgendwelche Role Blocks die Propagation der Rechte verhindern.

Es dürfte klar sein, dass diese Berechnungen innerhalb eines Requests zu großen Performanceproblemen führen würden. Deshalb wird die Berechnung der effektiven Rechte in Teilbereiche zerlegt, z. B. werden zunächst die Rollen bestimmt, die einer Gruppe zugeordnet sind, und die Ergebnisse anschliessend in einem Cache gehalten. Das gleiche gilt natürlich für die Rollen einer bestimmten Ressource. Es findet also ein Pre-Computing der effektiven Rechte statt. Das ist nicht nur zur Beschleunigung eines beliebigen individuellen Zugriffs nötig. Man denke nur an die Startphase eines großen Portals oder CMS, wenn die Caches für Content oder Rechte ursprünglich leer sind. Diese Situation würde zu großen Verzögerungen im Zugriff führen, da z. B. der Content erst aus Templates erzeugt (gerendert) werden muss. Deshalb gestatten diese Systeme die Vorausberechnung sowohl von dynamischen Inhalten als auch der Zugriffsrechte, zum Beispiel pro Gruppe. Zum Aufwand bei der Bestimmung der effektiven Rechte tragen also die folgenden Faktoren bei:

- Die Komplexität der Rollenzuweisungen: Nach Möglichkeit sollten Rollen nur an Gruppen vergeben werden. Insgesamt sollte es nur wenige Gruppen geben. Diese Faktoren beeinflussen den Aufwand bei der Berechnung der zwischengespeicherten Werte.
- Die Häufigkeit von Änderungen (Rollenzuweisungen, Role Blocks einfügen oder entfernen, Ausloggen des Users), da diese Änderungen ein Flushen der Caches oder Teilbereichen davon bedingen. (Das ist nicht unähnlich dem Caching von Dokumentfragmenten in Content Management Systemen oder Portalen. Auch dort müssen ganze Dokumente im Cache gelöscht werden, wenn sich Teile ändern.)
- Der Frequenz, mit der Basisdaten wie User, Rollen etc., sofern sie aus einem externen Repository kommen, aus dem Cache gelöscht und neu gelesen werden. Dies dient der Konsistenz zwischen Portal und externem Repository.
- Der Menge der Authorisierungsdaten, die in einem externen Repository gehalten werden, da dort der Zugriff deutlich langsamer ist als in einem lokalen Verzeichnis. LDAP Server etc. müssen hier hochperformant sein.

Das Design des Caches für die Autorisierung von Requests muss also über mehrere Stufen hinweg Rechte aus der Konfiguration sowie den Instanzdefinitionen in eine de-normalisierte Sicht pro User überführen, auf die im Moment der Prüfung sehr effizient zugegriffen werden kann. Gleichzeitig muss der Cache

Änderungen an Konfiguration oder Instanzrechten sofort nach oben propagieren und die von den Änderungen abhängigen Rechteinformationen invalidieren.

Eine Alternative dazu könnte die Verwendung der lokalen Datenbank darstellen. Views oder Stored Procedures könnten hier die Vorberechnung der Zugriffsrechte übernehmen, und der Prüfungscode könnte eine sehr einfache Tabelle aus User und Zugriffsrecht pro Ressource erhalten. Über Trigger können Invalidierungen durch Stored Procedures durchgeführt werden.

Ist der Autorisierungsalgorithmus bereits auf einem lokalen System eine teure Angelegenheit, so gilt dies erst recht in einem verteilten System. Hier ist es für die Performance kritisch, wie oft die effektiven Zugriffsrechte berechnet werden müssen. Gelingt es, sie in einem global zugänglichen Cache zu halten, müssen sie nicht auf jeder Maschine neu berechnet werden, wenn ein Request an weitere Rechner weitergegeben wird. Alternativ könnten die Rechte auch in Form eines Token weitergegeben werden.

10.2.5 Benutzer-Berechtigungs-Systeme (BBS)

Das Hauptaugenmerk bei vielen Sicherheitsanalysen und -designs liegt oft auf der Authentisierung von Benutzern. Leicht vergisst man dabei, dass die Authentisierung nur ein Mittel zum Zweck ist – nämlich eine feingranulare Zuordnung von Benutzern, Rechten und Ressourcen durchzuführen. Häufig findet dieser Prozess innerhalb von Applikationen statt: Die meisten Applikationen können Benutzer nicht nur authentisieren, sondern gestatten auch eine Einordnung in Gruppen oder Rollen, denen dann anschließend bestimmte Rechte (Aktionen) auf Ressourcen der Applikation zugeordnet werden können.

Genau dieses Abwälzen der Beziehung Benutzer-Ressource auf die Applikation ist aber problematisch, wenn es auf der Ebene von Enterprises geschieht. Nur wenige Firmen entscheiden sich jedoch für die konsequente Einführung eines Benutzer-Berechtigungssystems, bei dem diese Zuordnung aus den Applikationen in ein unabhängiges Subsystem verlagert wird. Zu hoch erscheinen die Aufwände für Modellierung und Implementierung eines solchen Systems. Im Zuge der gestiegenen Anforderungen an IT-Infrastrukturen durch rechtliche Vorgaben könnte diese Zurückhaltung jedoch schnell aufgegeben werden. Die folgenden Abschnitte stellen die Grundkonstrukte eines Berechtigungssystems dar²³.

10.2.5.1 Was macht ein BBS und wieso?

Aus den Sicherheitsprinzipien des „Need-to-know“ und „Need-to-do“ folgt, dass niemand innerhalb einer Firma „allmächtig“ sein sollte. Jeder Benutzer sollte nur über die für seine Arbeit nötigen Rechte auf Ressourcen verfügen. Ein BBS sorgt

²³ Die Autoren danken Hendrik Fuchs für die Unterstützung bei diesem Kapitel.

für eine einheitliche Verwaltung dieser Rechte. Dabei spielen fachliche Rollen, Organisationseinheiten, Daten, Services (SOA) eine wichtige Rolle. Typisch ist dabei die Trennung in *Funktion* (Rolle) und *Daten* (Ressource). Die Rollen eines Benutzers werden bestimmt durch die fachliche Funktion innerhalb der Firma. Die Rolle eines Benutzers genügt jedoch meist noch nicht für die Berechtigung auf eine Ressource oder Aktion. In einem nächsten Schritt können der konkreten Rolle noch Rechte auf Applikationen zugewiesen werden, und in einem weiteren Schritt können die speziellen, von der Applikation verwalteten Ressourcen wie z. B. die Sites eines Publishing-Systems ebenfalls an bestimmte Benutzer und deren Rollen gebunden werden. Oft verbleibt dieser letzte Schritt jedoch innerhalb der Applikation. Je mehr „Wissen“ über Rechte an Ressourcen aber innerhalb von Applikationen verbleiben und nicht in das BBS exportiert werden, desto größer sind die Anpassungsprobleme bei organisatorischen Änderungen von Benutzern (Wechsel) oder Strukturen (z. B. Reorganisationen). Die Frage, mit welcher Granularität Rollen und Rechte modelliert werden, ist deshalb von großer Bedeutung sowohl für den Aufwand bei der Einführung eines BBS als auch für die längerfristige Betreibbarkeit. Häufig wird bei der Modellierung mit Containern (Gruppen) oder Vererbung von Rollen gearbeitet, was eine nachträgliche Bearbeitung („flattening“) voraussetzt.

Damit sind die Funktionen eine BBS jedoch keineswegs erschöpft. Die Vergabe von Rechten und Rollen ist ebenfalls ein extrem wichtiger Vorgang innerhalb von Enterprises und so müssen auch dafür Rollen und Ressourcen definiert werden: Wer also kontrolliert und verwaltet das BBS? Das Gebot der Nachvollziehbarkeit verlangt, dass ein BBS als autarkes, streng kontrolliertes Subsystem implementiert wird, das über sichere Verwaltungsfunktionen für die Anforderung bzw. Erteilung von Rechten und der Protokollierung dieses Vorgangs verfügt. Diese Kontrollarchitektur durch Genehmigungen von Rechten durch deren Eigentümer setzt natürlich eine korrekte Ressourcenklassifizierung voraus. Letztlich stellt ein BBS also eine Autorisierungsdefinition dar, bei der Funktionen (z. B. Leserechte) mit den zugehörigen Daten gekoppelt werden: Welche Ressourcen dürfen von welchen Benutzern mit welchen Rollen bearbeitet werden?

10.2.5.2 Auswirkungen auf die Applikationsarchitektur

Die Einführung eines BBS hat großen Einfluss auf die Architektur der Applikationen. Damit die zentrale Verwaltung von Rechten und Ressourcen gelingt, muss eine Applikation die Autorisierungsentscheidungen externalisieren, d. h. sie darf eine Zugriffsentscheidung nicht ohne Konsultation des BBS treffen. Jede innerhalb der Applikation selbstständig getroffene Autorisierungsentscheidung bereitet im Falle von organisatorischen Änderungen innerhalb der Firma große Probleme und erfordert nachfolgende Änderungen im Code der Applikation. Je nach Granularität der Zugriffskontrolle (nur Rolle, Rolle und Ressource etc.) und Anforderungen an die Aktualität der Entscheidungsgrundlagen sind dazu unterschiedliche Softwarearchitekturen nötig. Die meisten Applikationen, vor allem eingekaufte

COTS Applikationen, besitzen außerdem bereits eine eigene Verwaltung von Rechten und Ressourcen sowie ein dazugehöriges Sicherheitsmodell, das mit dem vorhandenen Enterprise-BBS mehr oder weniger gut verträglich ist.

In einem ersten Schritt muss das Sicherheitsmodell mit den zugehörigen Rollen und Ressourcen in das BBS abgebildet und dort auf die entsprechenden funktionalen Rollen aufgeteilt werden. Damit entsteht im BBS Wissen darüber, wer welche Applikationen mit welchen Rollen verwenden darf. Wird nur auf der Granularität von Rollen gearbeitet – also ohne eine genauere Bindung an spezielle Ressourcen innerhalb der Applikation – dann muss der Zusammenhang Benutzer-Rolle vom BBS in die Applikation transportiert werden (mehr dazu gleich). Meist geschieht dies dadurch, dass eine Applikationsdatenbank mit den entsprechenden Rollen- und Benutzerdaten gefüllt wird. Alternativ können immer mehr Applikationen auch ein LDAP Repository mit externen Definitionen verwenden. Dieses muss dann entsprechend mit BBS Daten gefüllt werden.

In dem Fall, dass nur mit Rollenzuordnungen im BBS gearbeitet wird, ist das GUI für die Rechteverwaltung der Applikation weiterhin nötig. Mit Hilfe des GUIs werden dann den jeweiligen Benutzern – basierend auf den Rollendefinitionen des BBS – spezielle Ressourcen zugeteilt. Diese Beziehung ist dann jedoch nicht im BBS abgebildet. Klassische Beispiele sind hier die Zuordnung von Managern oder Publishern zu konkreten Sites.

In dem Fall, dass auch die konkrete Ressourcenzuordnung zentral verwaltet werden soll, verlagert sich das ganze Wissen über Benutzer und Ressourcen in das BBS und die der Applikation eigene Verwaltung wird damit praktisch sinnlos. Die Applikation darf dann lediglich die Benutzer, Rollen und Ressourcenzuordnungen aus dem BBS übernehmen, aber keine eigenen erstellen (s. Abb. 10.8).

Dies wirft eine Reihe von Problemen auf: Für die Verwaltung all dieser Rechte ist jetzt praktisch die Verwaltung des BBS selbst zuständig. Diese ist jedoch in den meisten Fällen eher generisch ausgelegt und nicht dafür gedacht, z.B. die

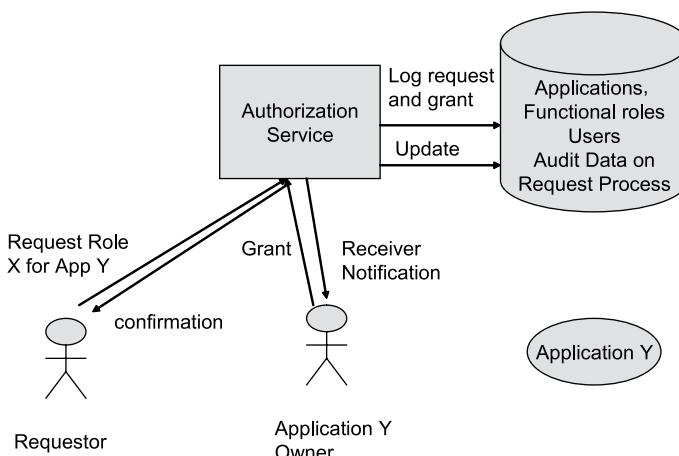


Abb. 10.8 System zur zentralen Authorisierung bzgl. Applikationen

Rollen und Ressourcen eines Content Management Systems zu verwalten. Andererseits operiert die eigene Verwaltung der Applikation außerhalb des zentralen BBS und damit unkontrolliert. Gleichzeitig bestünde natürlich auf Seiten des Business der Wunsch nach Service-Management-Delegation: Manager einer Ressource sollten in der Lage sein, daraus abgeleitete Rechte an weitere Mitarbeit zu vergeben. Dies erlauben auch mittlerweile viele Applikationen – allerdings eben außerhalb der Kontrolle durch ein BBS. Daraus ergeben sich operationale Probleme: Wenn ein Benutzer durch einen Manager ein Recht auf eine vom Manager verwaltete Ressource erhalten soll, so ist dafür einmal ein Antrag an das BBS zu stellen und je nach Granularität muss dazu auch noch in der Verwaltung der Applikation selbst die Zuordnung eingetragen werden. Idealerweise würde die Applikation auch diese Verwaltungsfunktionen über ein Interface externalisieren, so dass mit den Verwaltungsfunktionen der Applikation zwar direkt gearbeitet wird, dahinter jedoch transparent der Berechtigungsprozess des BBS eingehalten wird. Dazu gehört z. B. die Protokollierung der Rechteerteilung. Leider sind aber momentan die wenigsten Applikationen in der Lage, ein solches Interface anzubieten.

10.2.5.3 Schnittstellen

Im BBS gespeichert sind die Beziehungen zwischen Benutzern, Rollen, Applikationen und Funktionen sowie gegebenenfalls Ressourcen bis hinunter zu Instanzdaten. Diese Berechtigungsinformationen müssen einerseits verwaltet werden, andererseits dienen sie den Applikationen als Entscheidungsgrundlage im Rahmen der Access Control.

Die Schnittstelle für die Verwaltung wurde im letzten Abschnitt bereits diskutiert. Ideal wäre eine interaktive Verbindung zwischen den Verwaltungsfunktionen der Applikationen und dem Berechtigungssystem, das die korrekte Protokollierung und Rechteerteilung garantiert. Die Last dieser interaktiven Schnittstellen ist überschaubar, da sie für die Eintragung neuer Rechte bzw. den Update bestehender verwendet werden, nicht jedoch für die Access Control selbst.

Neben diesen interaktiven Schnittstellen der Applikationen mit dem BBS ist weiterhin für die Verwaltung von Massenmigrationen eine eigene Verwaltung des BBS nötig, mit der größere Änderungen der Firmenorganisation durchgeführt werden können. Über diese Schnittstelle kann das BBS auch abgefragt werden, z. B. zu Prüfungszwecken.

Bei der Verwendung der BBS Daten zur Access Control hingegen muss von ganz anderen Datenmengen ausgegangen werden. Manche Applikationen haben zehntausende von Berechtigten und entsprechend viele Kontrolloperationen müssen innerhalb der Applikationen durchgeführt werden. Ein „pull“ Interface, bei dem die Applikation bei jeder Zugriffsentscheidung im BBS nachfragt, klingt angesichts der sehr hohen Zahl von Abfragen eher problematisch. Der Vorteil einer solchen Real-Time Abfrage des BBS liegt einzig darin, dass damit die jeweils aktuellen Zugriffskontrolldaten bei der Entscheidung herangezogen werden. Der Preis ist eine gute Netzwerkinfrastruktur sowie ein redundant ausgelegtes,

extrem performantes BBS, das in der Lage ist, die Entscheidungsdaten schnell zu produzieren. Ein langsames oder schlimmer noch nicht-verfügbares BBS bedeutet in dieser Architektur, dass sehr viele Applikationen ebenfalls nicht verfügbar sind.

Vor diesem Hintergrund ist verständlich, dass in manchen Firmen ein „push“ Modell als Interface vorgezogen wird: Nächtliche Updates gelangen z. B. per FTP auf die Applikationsmaschinen, werden dort von einem daemon Prozess ausgelesen und in die jeweilige Applikationsdatenbank oder das lokale LDAP übernommen (s. Abb. 10.9).

Im Falle großer Datenmengen sind diese Filetransfers jedoch ebenfalls eine große Belastung für die Netzwerke und auch der Import kann relativ lange dauern. Optimiert werden kann diese Lösung durch das Anbieten von so genannten „Deltas“ zum Vortage, bei denen nur die Änderungen gegenüber dem Vortag übertragen werden und die die Datenmengen sehr verringern. Applikationen können dann entweder die vollständigen Daten oder nur ein Delta anfordern. Somit wird aus dem „push“ ein batchgetriebener „pull“.

Der große Nachteil des Batchbetriebs ist jedoch, dass Änderungen der Berechtigungsdaten sich erst einen Tag später auswirken. Dies kann je nach Anwendung u. U. nicht mehr akzeptabel sein. Entweder müssen Benutzer schneller berechtigt werden oder der Entzug von Rechten muss schneller wirksam werden. Hier hilft nur ein Real-Time Interface, möglichst durch Events des BBS an die jeweiligen Applikationen, um Rechteentzug schneller zu propagieren. Hier stellt sich die Frage, was passieren soll wenn eine Applikation momentan nicht erreichbar ist? Wie kann verhindert werden, dass der Event verloren geht?

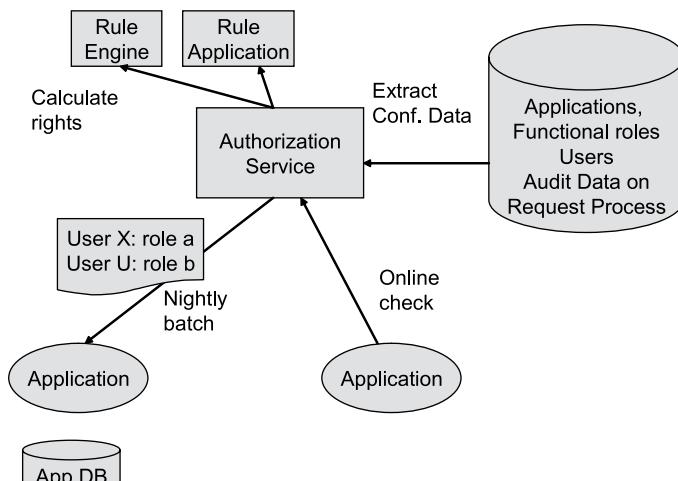


Abb. 10.9 Optimierung der zentralen Authorisierungslösung durch nächtlichen Transfer der Rechte zu den Applikationen

Die Frage, ob sich der Rechteentzug wirklich in Real-Time auswirken kann, hängt noch von weiteren Faktoren ab: Ist der betroffene Benutzer z. B. gerade in einer Session bei der Applikation aktiv, dann kann die Applikation aus Performancegründen die Berechtigungsdaten in einem Cache gespeichert haben. Dieser müsste dann im Falle eines Events von Seiten des BBS gelöscht werden. Dies ist unabhängig davon nötig, ob die Berechtigungsdaten per Batch oder interaktiv vom BBS abgeholt wurden. Bei einer geplanten Access Control in Real-Time sind in jedem Fall genaue Berechnungen der Netzwerk- und Serverlast nötig.

10.2.5.4 Probleme beim Betrieb eines BBS

Beim Betrieb eines BBS sind eine ganze Reihe von Problemen zu lösen, die häufig aus der Menge der berücksichtigten Daten sowie deren Anpassungen auf Grund organisatorischer Veränderungen resultieren. Einige davon sollen im Folgenden kurz angesprochen werden.

Ist ein BBS eine Applikation oder eine Datenbank? Hinter dieser Frage versteckt sich letztlich die Frage der Modellierung von Berechtigungen. Zugriffskontrolle benötigt eine Schnittstelle, die eine einfache Abfrage von Rechten eines Principals erlaubt. Modelliert wird die Berechtigungsinformation jedoch häufig durch Vererbungsprinzipien und Containerstrukturen: Rollen, Gruppen, Organisationseinheiten etc. dienen letztlich nur der einfacheren Verwaltung von Rechten pro Benutzer. Aus der Gruppenzugehörigkeit können sich Rollen ergeben, die Rechte auf Applikationen verschaffen. Aus der Organisationseinheit wiederum können Rechte auf konkrete Ressourcen entstehen. Die Frage ist nun, wo die „Berechnung“ der endgültigen Rechte eines Benutzers durchgeführt wird: Innerhalb des BBS oder erst innerhalb der Applikationen? Im Allgemeinen wird man versuchen, dieses Wissen aus den Applikationen herauszuhalten – wiederum, um sie vor Änderungen der Organisation zu schützen. Es gibt jedoch Spezialfälle wie z. B. Intranet-Suchmaschinen, bei denen aus Performancegründen diese Berechnungen innerhalb der Suchmaschine stattfinden (FAST, Autonomy).

Wenn das BBS die endgültigen Rechtevektoren berechnet, dann geschieht dies in Form eines Pre-Processing, bei der Containerstrukturen und Vererbungsprinzipien aufgelöst werden. Die Rechte Daten werden somit de-normalisiert. Dies bläht die Berechtigungsinformation auf, erlaubt aber die einfache Auswertung auf Seiten der Applikationen und verhindert Performanceengpässe bei der Berechnung. Kann in diesem Fall noch vom BBS als Datenbank mit Batch-Verarbeitung gesprochen werden, so wird das schwieriger sobald weitere Faktoren die Rechte eines Benutzers bestimmen: Bei Kunden als Nutzern können z. B. Marketinginformationen andere Rechte erzeugen oder die Kundensegmentierung wird grundätzlich angepasst. Jetzt entsteht der entgültige Rechtevektor möglicherweise als Resultat von Business-Rule-Engines, die dynamisch weitere Daten in die Entscheidungen einfließen lassen. Dabei können die Benutzerrechte sehr schnell angepasst werden. Der Nachteil besteht darin, dass die Korrektheit der jeweiligen Rechtevergabe immer komplizierter zu prüfen ist.

Sollen Rechte „altern“, d. h. automatisch ablaufen bzw. ungültig werden können? Sicherheitsgründe sprechen natürlich gegen Rechte, die für unbegrenzte Zeit vergeben werden. Ein automatisches Altern von Rechten führt wenigstens nach einiger Zeit dazu, dass Benutzer, die nicht mehr existieren, ihre Rechte verlieren. Allerdings ist das automatische Altern auch nicht unproblematisch und kann zu Problemen bei der Verfügbarkeit von Applikationen führen. Aus dem Bereich von Server-Zertifikaten kennen wir das Problem eines nicht funktionierenden Dienstes, das sich im Nachhinein als abgelaufenes Zertifikat entpuppt. Überhaupt, je komplexer die Rechtevergabe modelliert ist und je mehr dynamische Berechnungen dazu kommen, umso schwieriger wird es die Korrektheit der Rechtevergabe ange- sichts von Änderungen im BBS zu sichern. Dies gilt leider in zweifacher Hinsicht, denn es muss sichergestellt sein, dass

- kein Benutzer mehr als die ihm auf Grund seines Organisationbereichs, Funktion, etc. zustehenden Rechte besitzt und dass
- jeder Benutzer die Rechte, die er auf Grund seiner Tätigkeit in der Firma benötigt, besitzt.

Die IT-Security beschäftigt sich traditionsgemäß eher mit der Verhinderung ungerechtfertigter Zugriffe als mit der Garantie gerechtfertigter Zugriffe. In der heutigen Praxis ist die Frage der Verfügbarkeit essentieller Business-Services jedoch lebenswichtig für Firmen. Eine Änderung der BBS-Daten, die am nächsten Morgen tausende von Mitarbeitern oder Kunden von wichtigen Diensten ausschließt, ist eine Katastrophe. Leider existieren in diesem Bereich der Modellprüfung wenige Werkzeuge und Methoden, und dies führt in der Praxis dazu, dass im Zweifel lieber Rechte hinzugefügt als weggenommen werden. Besonders brisant ist die Frage der Korrektheit der BBS Daten bei sog. Massenmigrationen, d. h. wenn ganze Organisationseinheiten innerhalb der Firma verschoben werden und tausende Benutzer andere Aufgaben und damit Rechtevektoren erhalten müssen. Auch dies geschieht aus Performancegründen häufig über ein Pre-Processing der Rechtevektoren.

Ein System, das Berechtigungen verwaltet, muss seinerseits ebenfalls verwaltet werden. Es müssen also Rollen und Rechte für die Definition und Modifikation von Daten und Funktionen des Berechtigungssystems bestimmt werden. Diese Rechte stellen quasi Meta-Rechte gegenüber den gewöhnlichen Rechten dar. Sie sind extrem sicherheitskritisch, denn sie erlauben die Rechtevergabe für Applikationen und Daten und sollten daher an einen sicheren Bewirtschaftungsprozess gebunden werden. Eine allmächtige Administratorenrolle, die definieren und genehmigen kann, ist hier nicht die richtige Lösung. In diesem Zusammenhang müssen deshalb weitere Fragen beantwortet werden, z. B.:

- Wer hat welche Rollen und Rechte definiert?
- Wer hat die Definitionen genehmigt?
- Wer bestimmt den Genehmigungsprozess?
- Was bedeutet eine Rolle konkret?
- Wieviele Rollen sind nötig?
- Macht eine konkrete Rolle noch einen fachlichen Sinn?

- Wie oft können/sollen Änderungen vorgenommen werden?
- Wie werden Entscheidungen herbeigeführt und protokolliert?
- Wird bei der Rechtevergabe ein Vier-Augen Prinzip verwendet?
- Werden Stellvertreter-Rollen modelliert?

Die Fragen führen tief ins Sicherheitsframework einer Firma. So lässt sich die Frage nach der Genehmigung von Rechten nur dann beantworten, wenn klar ist, wer für welche Daten zuständig ist, und wie die jeweiligen Daten klassifiziert wurden. Zudem gibt es bezüglich der Bewilligung von Rechten jeweils verschiedene Sichten: Vorgesetzsicht, Sicht der Fachabteilung, etc. Was soll passieren, wenn eine Rolle 30 Systeme betrifft, die zusammen mehrere Abteilungen, Bereiche oder Sub-Organisationen umfassen? Wer ist dann zuständig für Bewilligungen von Rechten? Wie sehen die Stellvertreter-Regelungen aus?

Der Zwang zur Nachvollziehbarkeit von Entscheidungen trifft natürlich auch die Berechtigungsdaten. Dies macht mehrere Zeitachsen nötig (so genannte *bitemporale* Datenhaltung). So muss nachträglich jederzeit festgestellt werden können, wann welche Rechte mit welchen Benutzern wie verbunden waren. Jeder Zustand der Vergangenheit braucht gültige Entitäten mit dazu gehörigen „von-bis“ Daten. Es müssen Abfragen von der Art „Wie war ein bestimmter Zustand zu einem bestimmten Zeitpunkt?“ möglich sein. In diesem Zusammenhang stellen bereits die Stammdaten der Benutzer ein großes Problem dar: Durch Namenswechsel, Organisationswechsel, Ausscheiden etc. können sie einer hohen Fluktuation unterliegen. Zusätzlich gibt es künstliche Benutzer-Identitäten (Functional User), die z. B. für Serverapplikationen benötigt werden, und die ebenfalls Rechte benötigen. Je strenger die Zugriffe innerhalb der Applikationen dann kontrolliert werden, desto mehr werden auch Testaccounts (Test-User) für die Entwicklung und den Test der Applikationen gebraucht, da die Applikationen sonst in ihrer endgültigen Form und Umgebung nicht testbar wären.

Die Kommunikation mit den Benutzern ist ein weiteres wichtiges Element eines Benutzerberechtigungssystems. So muss ein Benutzer über Rechteänderungen, die sich z. B. aufgrund von organisatorischen Wechseln ergeben, informiert werden. Bei manchen geplanten Änderungen kann es sinnvoll sein, die Benutzer vorher um Zustimmung zu bitten. Und nicht zuletzt sollten Entwickler über einfache Schnittstellen verfügen, um im Laufe der Entwicklung von Applikationen neue Rechte definieren (aber nicht genehmigen) zu können. Dies ist umso wichtiger, je granularer diese Rechte Funktionen oder Daten abbilden. Der Prozess der Rechtebewirtschaftung und -vergabe kann außerdem im Top-Management auf Probleme stoßen, bis hin zur völligen Nicht-Akzeptanz.

10.2.6 Geschäftsprozesse: Workflow und Realität

Ein wichtiges Element im Publishing-Workflow stellt die Regelung der Freigabe von Dokumenten dar. Hier wird meist das so genannte „Vier-Augen-Prinzip“

verwendet: Ein normaler Publisher darf nicht von sich aus die Freigabe eines Dokuments veranlassen. Er kann lediglich in der Statusverwaltung des Dokuments erklären, dass es „fertig editiert“ ist. Ein definierter Workflow innerhalb des CMS führt dann automatisch dazu, dass dieses Dokument einem Sitemanager zur endgültigen Freigabe vorgelegt wird. Dieser trägt dann auch die Business-Verantwortung als Herausgeber. Sobald das geschehen ist, sorgt das CMS dafür, dass das Dokument über die konfigurierten Kanäle (z. B. das Internet) zur Verfügung steht.

Dies ist eine standardisierte Vorgehensweise in den meisten Publishing-Prozessen und ähnelt anderen Geschäftsprozessen, bei denen beispielsweise zwei Schlüssel für einen Tresor nötig sind oder zwei Unterschriften unter einer Bestellung. Die Idee dahinter besteht darin, eine doppelte Kontrolle durchzuführen bzw. zwei Personen statt einer in eine rechtliche Verantwortung zu nehmen und so die Sicherheit des Prozesses zu erhöhen. Bei der Implementierung derartiger Workflows gibt es jedoch einige Besonderheiten zu beachten:

- Die Implementation muss Ausnahmen zulassen können.
- Jegliche Ausnahme muss in Bezug auf Identität und Durchführung genau protokolliert werden.

Welche Ausnahmen sind hier gemeint? Jeder Geschäftsprozess ist eine idealtypische Festlegung eines Ablaufs, an dem Personen und Dinge beteiligt sind. Personen werden jedoch manchmal krank oder gehen in Urlaub, Dinge gehen kaputt und müssen repariert werden. Es muss also möglich sein, in bestimmten Situationen von den vorhandenen Regeln abzuweichen. Das Abweichen kann auf verschiedene Weisen erfolgen:

- Die Sicherheitsregeln werden komplett aufgehoben.
- Eine Kollegin erhält die Login Credentials der erkrankten Person und führt die Aktionen durch.
- Eine „allmächtige“ Rolle erlaubt es, die nötigen Aktionen durchzuführen
- Es wird ein Stellvertreter mit genau definierten Rechten eingeführt, der an Stelle der ursprünglich vorgesehenen Person die Aktionen durchführen kann. Der Stellvertreter handelt dabei unter seiner eigenen Identität.
- Die Eigentümerin der Ressourcen stellt granulare Schlüssel (so genannte *Capabilities*) aus, deren Besitz zum Zugriff auf die Ressource berechtigen. Hier spielt die Identität des Trägers des Schlüssels keine Rolle.

Natürlich machen sicherheitstechnisch nur die letzten beiden Punkte – eine Stellvertreter-Regelung bzw. ein schlüsselbasierter Zugriff – Sinn. In der Praxis sieht man allerdings oft, dass zu den ersten drei Mitteln gegriffen wird: Manche Software kennt eben nur die zwei Modi „sicher“ oder „offen“. Das Fehlen von Ausnahmeregelungen zwingt dann zum kompletten Abschalten der Sicherheit. Hier liegt eindeutig ein Softwareproblem vor: Die Software ist nicht in der Lage, sich der jeweiligen Firmensituation anzupassen.

Gerade im Urlaubsfall kann man auch in vielen Firmen beobachten, dass Credentials (also UserID/Passwort) getauscht werden. Dies hat den negativen Effekt, dass die Nachvollziehbarkeit von Aktionen leidet, da unter einer falschen Identität

gearbeitet wird. Oft erfolgt nach der Rückkehr aus dem Urlaub auch kein Wechsel des Passwortes und der Empfänger (es ist ja kein „Stellvertreter“, denn ein Stellvertreter würde unter der eigenen Identität arbeiten) erhält dadurch ein Geheimnis, mit denen er die ursprüngliche Person jederzeit impersonifizieren kann.

Viele Systeme kennen eine „allmächtige“ Instanz eines Users. Meist handelt es sich dabei um einen so genannten *Functional User* – eine bloße Rolle, die mit keiner bestimmten Identität einer realen Person verbunden ist und daher auch nur mit Schwierigkeiten einer realen Person zugeordnet werden kann. Beispiele hierfür sind die UserID „root“ unter Unix oder „Administrator“ unter Windows. Allerdings gibt es diese funktionalen User nicht nur auf Systemebene, sondern auch innerhalb vieler Applikationen. Das Problem liegt hier in der Zuordnung dieser potenziell sehr gefährlichen Rolle zu realen Personen und der Einschränkung ihrer Rechte. Kombiniert mit einem hierarchischen Rollenmodell, bei dem eine höher gestellte Rolle alle Rechte der unteren Rollen besitzt, erlaubt der Besitz der „allmächtigen“ Rolle, dass an den üblichen Einschränkungen vorbei beliebige Aktionen durchgeführt werden können, ohne dass eine klare Verantwortlichkeit gegeben ist. Ein solches Sicherheitsdesign der Software macht es sehr schwer, die Vertraulichkeit von Dokumenten und die Einhaltung von Business-Regeln wie etwa das Vier-Augen-Prinzip zu garantieren. Allerdings erlaubt eine solche Instanz die schnelle Behandlung von Notfällen.

Wie sollten also die Rechte dieser „allmächtigen“ Rolle definiert werden? Nach Möglichkeit sollten ihre Aktionen sehr eingegrenzt sein, zum Beispiel auf die Zuordnung von Rechten zu konkreten Usern beschränkt sein. Das bedeutet, der Administrator der Applikation ist nur in der Lage, Usern Rechte zu geben, besitzt jedoch selbst keine Rechte an Dokumenten und kann daher auch vertrauliche Dokumente nicht unter der funktionalen Identität lesen.

Gehen wir in einem kurzen Exkurs auf den „Systemadministrator“ als Betriebssystemrolle ein: Besonders problematisch aus Sicht der Security einer Applikation ist es, wenn zur Verwaltung der Applikation auf Administratoren des Betriebssystems zugegriffen werden muss. Da diese Rollen nur auf Betriebssystemressourcen zugreifen können und von der Business-Logik der Applikation nichts verstehen, besteht die Gefahr, dass zum Beispiel durch Änderungen von Files oder Zugriffsrechten auf Betriebssystemressourcen unbeabsichtigte Änderungen in der Sicherheit der Applikation entstehen können. Für das Design sicherer Applikationen bleibt festzustellen, dass für die Verwaltung der Applikation eine interne, eingeschränkte Administrationsrolle nötig ist.

Kommen wir nun zu den beiden aus Security-Sicht bevorzugten Lösungen für das Problem einer Abweichung vom vorgegebenen Geschäftsprozess: Dies waren entweder eine Stellvertreterregelung oder ein schlüsselbasiertes, granulares Zugriffskonzept. Stellvertreterregelungen gehören zu einer Klasse von Softwareproblemen, die seit je her gefürchtet sind. Zu dieser Klasse gehören auch die Mandantenfähigkeit von Software oder die Frage, wie man Authentisierungsmethoden verschiedener Qualität beim Zugriff auf Backend-Services behandeln soll (das sog. Step-Up Authentication Problem). Wir werden im Band „Sichere Systeme“ geeignete Software-Architekturen zur Lösung dieser Probleme behandeln, an

dieser Stelle wollen wir nur das Grundproblem hinter diesen schwierigen Sicherheitsproblemen offen legen. In unzähligen Spezifikationen von Applikationen tauchen diese Anforderungen genau einmal auf, und zwar unter der Rubrik „und außerdem brauchen wir...“, ohne dass sich dann später weitere Hinweise darauf, geschweige denn eine Implementierung entdecken lassen.

Allen diesen Problemen ist gemeinsam, dass sie sich über den gesamten Code einer Applikation hinweg auswirken können, ohne dass dies dem Entwickler in irgendeiner Form angezeigt werden muss. Ein Beispiel zum Stellvertreterproblem wird das verdeutlichen:

Nehmen wir an, eine Applikation gestattet es, durch einen nachvollziehbaren und abgesicherten Prozess einen Stellvertreter für einen bestimmten User zu definieren. Dieser Stellvertreter arbeitet nun auf den Ressourcen des Users, aber unter seiner eigenen Identität. Dabei tritt schon das erste Problem auf: Soll die Applikation bei der Überprüfung von Zugriffsrechten die gesamten Rechte des Stellvertreters berücksichtigen oder nur das Subset, das den Rechten des zu vertretenden Users entspricht? Es besteht schließlich die Gefahr, dass der Stellvertreter mehr Rechte besitzt als der zu vertretende User und deshalb Aktionen auslösen kann, die der normale User nicht auslösen könnte. Das wäre ein Beispiel für eine so genannte *Privilege Elevation* – also die fälschliche Erhöhung von Privilegien während der Ausführung einer Aktion. Dieses Phänomen wird als das *Confused-Deputy-Problem* bezeichnet.

Aber die Probleme der Stellvertretung sind noch viel subtiler: Nehmen wir an, dass der Stellvertreter während seiner Tätigkeit als Stellvertreter einen Fehler entdeckt und diesen über ein applikationsspezifisches Nachrichtensystem an jemanden meldet. Unter welcher Identität soll der Fehler gemeldet werden? Wird der Fehler unter der Identität des Vertretenen gemeldet und das Fehlersystem schickt eine Bestätigung, dann erhält sie der abwesende User, der den Fehler aber gar nicht gemeldet hat. Der Stellvertreter hingegen erhält keine Bestätigung, es sei denn, er kann auch die Nachrichten für den vertretenen User lesen. Dann stellt sich wiederum die Frage, ob dieses System die geschickte Bestätigung als gelesen anzeigen soll, wenn der Originaluser wieder da ist?

Schnell merkt man, dass sich das Problem der Stellvertretung durch die gesamte Software-Architektur zieht. Kaum eine Aktion bleibt davon unberührt, und als erfahrener Software-Entwickler sieht man schon Unmengen von "if (Stellvertreter) else"- Abfragen vor dem geistigen Auge. Gleichzeitig kann man davon ausgehen, dass es viele Stellen im Code geben wird, wo die Entwickler die Berücksichtigung der Stellvertretung vergessen haben. Das ist auch ganz natürlich: Nichts in einem Stück Code weist Entwickler darauf hin, dass sie mit zwei verschiedenen Identitäten rechnen müssen. Stellvertretung bedeutet eben nicht „Impersonation“, bei der eine Identität komplett und ohne Ausnahme von einer anderen Entität übernommen wird.

Welche Möglichkeiten im Softwaredesign gäbe es überhaupt, das Prinzip Stellvertretung zu implementieren?

- Verdoppelung aller Rechte durch Hinzufügen einer Methode `doAs StellvertreterActionXYZ` für jede Aktion. Gleichzeitig Verdoppelung aller Aktionen durch die jeweilige Stellvertreteraktion mit anderem Namen. Dadurch könnte ein Stellvertreter die originale Aktion gar nicht ausführen. Der Preis dafür besteht in einer wesentlichen Vermehrung von Methoden und Rechten. Zumindest die Vermehrung von Methoden könnte über geschickte Delegation und den Einsatz von Inheritance aufgefangen werden.
- Filtern aller Aktionen an zentralen Stellen und Prüfung auf Stellvertretung. Dies ist ein recht grobes Vorgehen und bedarf eines guten Interface Designs, damit Stellvertretung überhaupt möglich ist. Das Verfahren ist aber performant und gut geeignet, bestimmte Aktionen von der Stellvertretung auszuschliessen.
- Einsatz von „Inversion of Control“-Techniken und Closures.

Es bleibt als letzte Möglichkeit, den granularen Zugriff auf Ressourcen zu diskutieren: Das bedeutet schlüsselbasierte Zugriffe *ohne* Berücksichtigung der Identität des Aufrufers. Betrachten wir dazu das in den Anforderungen an ein CMS erwähnte Prinzip der Kollaboration mit externen Usern. In vielen Fällen bedeutet dies, dass diese User in das eigene System der Authentisierung aufgenommen werden müssen, mitsamt ihrer Credentials. Ein typisches Beispiel dafür sind Supply-Chain-Management Systeme, bei denen zum Beispiel Mitarbeiter einer Automobilfirma Teile für die Produktion elektronisch bei Zulieferern bestellen. Diese Mitarbeiter müssen bei den Zulieferern in deren User-Repositories eingetragen sein, damit sie sich authentisieren können. Anschließend erhalten sie die entsprechende Rollen und Rechte zugewiesen. Aber streng genommen sind die Zulieferer gar nicht daran interessiert, diese fremden Mitarbeiter in das eigene Repository aufzunehmen und zu verwalten. Wird zum Beispiel beim Automobilhersteller der Mitarbeiter Meier durch Frau Müller abgelöst, so tritt beim Zulieferer Wartungsaufwand in Form einer nötigen Anpassung der Einträge in der User Registry auf.

Im Falle der gemeinsamen Bearbeitung von Dokumenten kann die Situation ähnlich sein: So haben Businessmitarbeiter oft den Wunsch, externen Personen Zugriff auf gewisse Dokumente zu geben, ohne dass diese externen User Mitglieder der Firmen-Registry werden sollen – vielleicht weil es sich nicht um Businesspartner, sondern um Journalisten handelt. In diesem Fall wäre eine Zugriffskontrolle, die auf Schlüsseln basiert, sehr attraktiv. Der Automobilhersteller erhält lediglich einen Schlüssel zur Bestellung und kann diesen Schlüssel weitergeben an seine Mitarbeiter. Das gleiche Prinzip könnte auch bei der Bearbeitung von Dokumenten verwendet werden.

10.2.7 Zugriffskontrolle beim Endnutzer

Neben der Erzeugung des Contents beschäftigt sich ein Content Management System mit der Auslieferung des Inhalts an die Endbenutzer. Handelt es sich dabei nur um öffentlichen Inhalt, der noch dazu auf einem getrennten System verwaltet

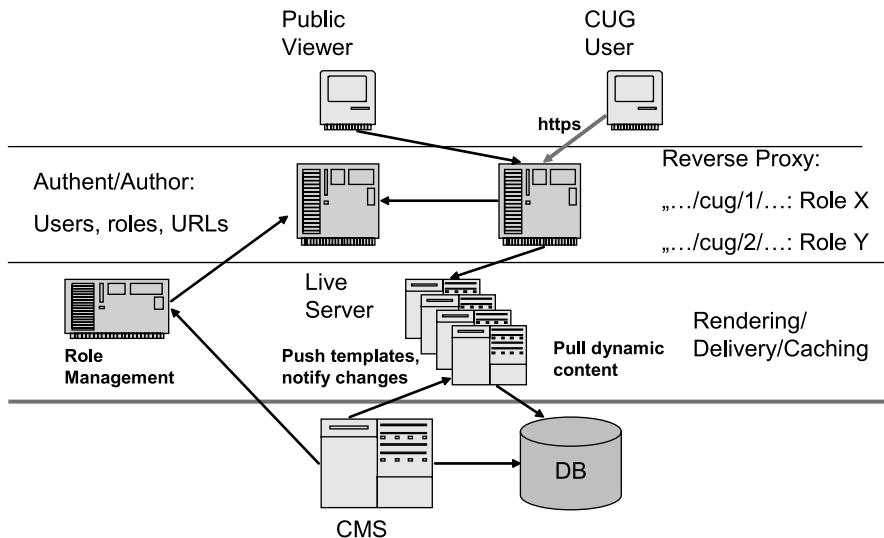


Abb. 10.10 Abgesichertes CMS mit unterschiedlich berechtigten Nutzergruppen

wird, sind die Sicherheitsrisiken für die Daten gering. Die größte Gefahr besteht darin, dass veränderter oder inhaltlich falscher Content angezeigt wird, oder dass durch Fehler in der Input-Validierung Cross-Site Scripting Attacken mit dem Ziel von Phishing ermöglicht werden. Bei öffentlichen Inhalten ist keine Authentisierung der Endbenutzer nötig.

Anders sieht die Sache jedoch aus, wenn auf einem System Content verschiedenster Klassifizierung gehalten wird und auch an Endbenutzer mit unterschiedlichen Berechtigungen geliefert werden soll. In Abb. 10.10 dazu eine Darstellung eines CMS mit unterschiedlichen Rechtebereichen, die zu so genannten Closed-User-Groups (CUG) korrespondieren.

Die Einführung geschlossener Usergruppen in eine abgesicherte Infrastruktur erweist sich als erstaunlich aufwendig. Es beginnt mit Fragen der generellen Datenklassifizierung: Welche Sorte Daten dürfen über an die geschlossene Nutzergruppe ausgeliefert werden? Manchmal ist der Anlass für den Wunsch nach geschlossenen Gruppen die Notwendigkeit, Journalisten mit Vorab-Informationen zu versorgen. In diesem Fall handelt es sich bei den Informationen um keine kundenbezogenen Daten, die sehr hohe Vertraulichkeit genießen. Aber ist in allen Fällen den Publishern auf dem System klar, dass keine vertraulichen Informationen in dem für die CUG zugänglichen Bereich abgelegt werden dürfen?

Das Dilemma wird schnell deutlich, wenn man die typischen externen User einer Firma betrachtet. Diese sind meist entweder unbekannt – d. h. nicht authentisierte Besucher der Homepage – oder es sind registrierte Kunden, die mit User-ID und Passwort im Kundenverzeichnis eingetragen sind. Journalisten sind aber häufig weder das eine noch das andere. Soll man nun diese Gruppe als „Dumm“-Kunden eintragen, nur damit sie sich authentisieren bzw. eine Rolle erhalten

können? Noch schlimmer ist das Problem, wenn die Firma nur zwei Mechanismen der Authentisierung kennt: Keine (d. h. öffentlicher Zugang) oder starke Authentisierung, beispielsweise via SSL mit PIN/TAN. Soll für die halb-öffentliche Gruppe der Journalisten auch so eine Authentisierung eingerichtet werden? Ist es zumutbar, dass die Journalisten sich per PIN/TAN authentisieren müssen, nur um bestimmte Artikel vorab lesen zu können? Wie aufwendig wäre es, für die Journalisten einen Zugang als „Dummy“-Kunden einzurichten, und wie gefährlich wäre es? Beispielsweise könnten dadurch Seiteneffekte entstehen (im harmlosen Fall erhalten die Journalisten plötzlich unsinnige Benachrichtigungen über Sonderangebote, im weniger harmlosen Fall brechen aber plötzlich Geschäftsprozesse zusammen).

Schnell kommt in dieser Situation der Vorschlag auf, doch einfach für diese Gruppe UserIDs und Passwörter als Authentisierungsmechanismus neu einzuführen. Abgesehen davon, dass dies einen großen Aufwand in Bezug auf die Organisation bedeutet (Passwörter müssen erstellt, zugeschickt und wieder gesperrt werden), wird plötzlich eine neue Qualität an Authentisierung eingeführt, und zwar in eine Firma, die bisher nur „keine“ bzw. „starke“ Authentisierung gekannt hat, und deren Software, Entwickler und Business User sich auf diese zwei Stufen eingestellt hatten. Jetzt entsteht die Gefahr, dass Services, die bisher eine starke Authentisierung voraussetzen konnten, plötzlich mit schwach authentisierten Requests aufgerufen werden.

Im Bereich der öffentlich zugänglichen Pages stellt sich die Frage, wie mit Links auf spezielle, eingeschränkte Informationen verfahren werden soll. Sollen die User merken, dass es spezielle, gesperrte Bereiche gibt? Wie sollen die Publisher mit solchen Links umgehen? Nicht zuletzt darf die Search-Engine die Inhalte der CUG-Seiten nicht einfach indizieren – aber heißt das auch, dass CUG-User für diesen speziellen Content die Search-Engine der Site nicht benutzen können?

Die Authentisierung wird wahrscheinlich in einem vorgelagerten Reverse-Proxy Server stattfinden. Dies hat weitere Konsequenzen für die Sicherheit: Die SSL Session sollte dort enden, um Filtering zu erlauben. Es muss einen Authentisierungs- und Autorisierungsservice geben, der die gesicherten URLs sowie die dafür nötigen Rollen der User verwalten kann. Dies wirft sofort das Problem auf, wie die nötigen Autorisierungsdaten dorthin gelangen und wie der Service in das firmenweite Berechtigungskonzept eingebunden werden kann.

Das kleine Beispiel zeigt, dass sich bei der Einführung geschlossener Nutzergruppen Sicherheitsfragen von der Ebene der Geschäftslogik bis hinunter zur Performance von Proxy Servern auftun. Besonders kritisch sind dabei alle Fragen, die mit den Vorstellungen der Programmierer und Benutzer des Systems zu tun haben, und die sich auf die Datenklassifikation sowie die Delegation von Services richten. Mit der Absicherung der Gesamtlösung inklusiver aller Server und Datenbanken beschäftigt sich das Kapitel zur Infrastruktur-Sicherheit.

10.3 Spezielle Probleme von Content Security

Zum Abschluss unserer Diskussion der Security auf Dokument-Ebene wollen wir einige spezielle Aspekte erwähnen, die wir an dieser Stelle zwar nicht erschöpfend behandeln können, aber die wir zumindest anschneiden wollen, da sie in letzter Zeit an Bedeutung gewonnen haben. So hat sich das so genannte Records Management in den letzten Jahren als die „Cash-Cow“ der IT-Abteilungen in großen Firmen herausgestellt. Unter dem rechtlichen und finanziellen Druck von Gesetzen und Verträgen wie Basel II und Sarbanes-Oxley [SOX] sahen sich die Firmen gezwungen, die Speicherung ihrer Dokumente auf eine rechtsfähige Basis zu stellen. Der zweite Aspekt – die fein granulare Sicherheitsklassifikationen von Daten und Werkzeugen sowie Subjekten – ist in seiner Bedeutung für die Allgemeinheit bislang eher umstritten. Das dahinter stehende allgemeine Prinzip des Data-Taggings mit Security Labeln jedoch könnte sich in Software-Architekturen von CMS bis hin zu Browsern als brauchbar herausstellen. Schließlich wollen wir unter dem Titel „Mobile Dokumente“ noch kurz die Sicherheitsprobleme ansprechen, die entstehen, wenn Firmenmitarbeiter sensible Dokumente auf mobilen Rechnern außerhalb der Firmengrenzen bringen.

10.3.1 Records Management

Records Management – also die nachprüfbare, rechtlich verbindliche Verwaltung wichtiger, firmeninterner Dokumente – ist ein großes Thema der Content Level Security geworden, nicht zuletzt durch die Nachwirkungen milliardenschwerer Skandale um Top-Manager und deren kreativer Buchführung.

Ganz ähnlich wie bei einer professionellen Buchführungsapplikation müssen beim Records Management alle Änderungen eines Dokuments protokolliert und das Dokument selbst gegen Verlust oder Verschwinden geschützt werden. Als Dokument gilt in diesem Zusammenhang natürlich auch E-Mail. Die geforderte Funktionalität geht damit über ein Versionskontrollsystem hinaus, ähnelt diesem jedoch stark. Sicherheitstechnisch interessant sind dabei vor allem vier Aspekte:

- Definition eines „Records“
- Absicherung eines Records gegen Änderungen
- Verwaltung der Zugriffsrechte auf Records
- Langzeit-Speicherung der Records

Das Diagramm in Abb. 10.11 ist einer Studie zum Records Management der IBM entnommen. Es zeigt die wesentlichen Typen der „Records“ und die zum Bearbeiten verwendeten Applikationen.

Für Software-Entwickler ist bei der Umsetzung der Anforderungen an das Records Management entscheidend, ob eine rein prozess-orientierte Sicherung der Daten ausreicht – das bedeutet, sobald die Art und Weise der Archivierung als

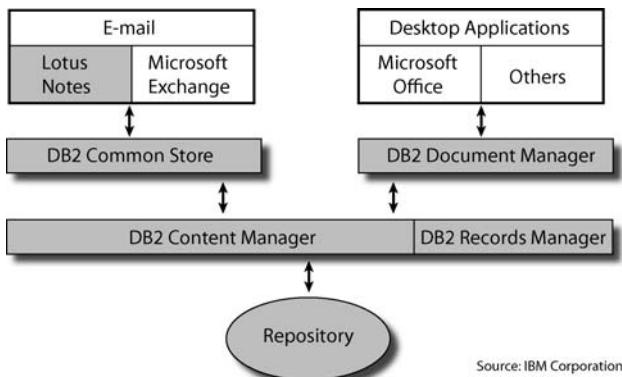


Abb. 10.11 Basisarchitektur des Records Management der IBM

korrekt angesehen wird, dann liegt ein gültiges Record Management vor, oder ob zusätzlich durch kryptografische Verfahren wie digitale Signaturen die Gültigkeit der Records abgesichert werden muss. Ein Anwendungsfall für letzteres ist sicher die elektronische Verwaltung von Grundbucheinträgen. Dort kommen meist „qualifizierte“ elektronische Signaturen (das bedeutet, die dazu gehörigen Zertifikate und deren Aussteller erfüllen spezielle Randbedingungen, die im deutschen Signaturgesetz [SigG] festgelegt sind), deren private Schlüssel sich auf Smartcards befinden, zum Einsatz. Sowohl die bearbeitenden Systeme als auch die Bearbeiter signieren dann die Daten.

Alle diese Verfahren beruhen auf Zertifikaten, welche die Authentizität des Public Key bescheinigen. Das Ablauen der Gültigkeitsdauer dieser Zertifikate kann deshalb ein großes Problem werden, wenn die betreffenden Dokumente nicht rechtzeitig mit gültigen Zertifikaten nachsigniert werden.

10.3.2 Mobile Dokumente

Content-Level Security meint in der letzten Zeit zunehmend auch Konzepte mobiler Sicherheit. Darunter fallen die folgenden Ziele aus Sicht der Security Policy einer Firma:

- Die Bearbeitung von Dokumenten durch nicht zulässige Werkzeuge sollte verhindert werden.
- Das bewusste Entfernen von Dokumenten über Geräteschnittstellen aus der Firma durch Berechtigte darf nicht möglich sein.
- Durch den Verlust von mobilen Geräten dürfen keine Dokumente verloren gehen.
- Die Benutzung von Dokumenten außerhalb der Firma ist ebenfalls einer Policy unterworfen (aus einem Sicherheitsbedürfnis heraus oder aus finanziellen Interessen an Lizenzien).

Eine umfangreiche Palette von Maßnahmen steht für diese Anforderungen zur Auswahl. Am Beginn steht die Sicherung der Verarbeitung innerhalb eines Rechners, indem die Werkzeuge, mit denen Daten behandelt werden, grundsätzlich eingeschränkt werden. Dieses Vorgehen ist typisch für die Multi-Level-Security, also das Zuweisen unterschiedlicher Sicherheitslevel an die Dokumente. Damit kann auch ein Teil der Mobilitätsproblematik gemildert werden, indem z. B. die Geräteschnittstellen zu USB-Laufwerken kontrolliert und damit das Kopieren von Dokumenten auf USB-Sticks verhindert werden kann.

Alternativ zu diesen Massnahmen auf Schnittstellenebene steht die Absicherung auf Inhaltsebene durch Verschlüsselung und Formen des Digital Rights Managements (DRM). Diese Verfahren können zum Einen die Nutzung der Dokumente beschneiden, zum Anderen auch mit Hilfe so genannter *Digitaler Wasserzeichen* den Inhaber einer unrechtmäßig weiter gegebenen Kopie kenntlich machen. Eine Übersicht der hierbei verwendeten Verfahren findet sich im Kapitel „Mediensicherheit“ in [Schm]. Zur Einbindung von DRM-Maßnahmen in ein firmenweites Sicherheitskonzept siehe [Butz].

Ein anderes wichtiges Sicherheitselement in diesem Zusammenhang ist die Klassifikation der Daten (*Labelling*) zusammen mit der Anforderung, dass die Klassifikation unmittelbar bei den Daten verbleibt. Wir werden im folgenden Abschnitt auf die Konzepte des Labellings eingehen, da es sich dabei um einen der zentralen Mechanismen der Isolation bzw. der Zugriffskontrolle handelt. Sehr schön lässt sich an diesem Beispiel auch der Unterschied zwischen Sicherheit, die im Prozess liegt und Sicherheit, die an den Daten (Objekten) selbst ansetzt, demonstrieren.

Abbildung 10.12 zeigt den Ansatz der aus dem Labelling resultierenden so genannten *Multi-Level Security* (auch Label-based Security genannt), die, ausgehend

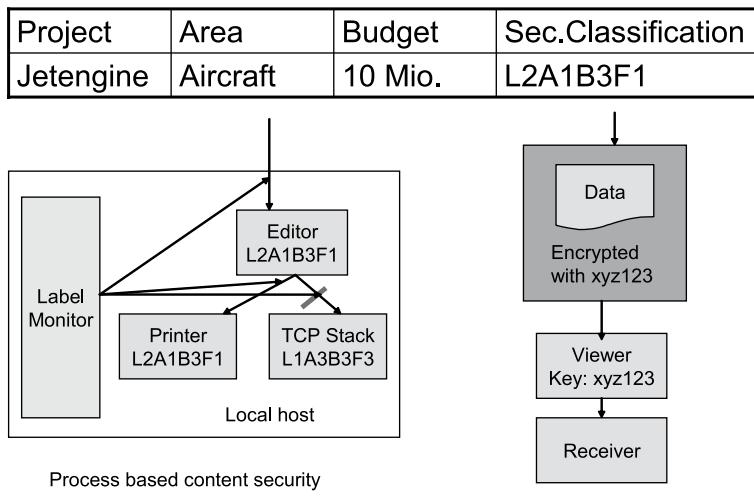


Abb. 10.12 Multi-Level-basierte Content Security vs. DRM-basierte Content Security

von einer Klassifikation von Daten, die Verarbeitung und den Transport der Daten an diese Klassifizierung bindet. Der Unterschied zwischen MLS und DRM-Ansätzen liegt im Gültigkeitsbereich der Absicherung: Im MLS System muss der Referenz-Monitor die Einhaltung der durch Labels definierten Sicherheitsklassen garantieren. Verlassen die Daten den Kontrollbereich des Monitors, sind sie ungeschützt. Bei DRM Systemen können die Daten selbst mobil sein und den Hersteller verlassen. Das Vertrauensverhältnis wird zwischen Hersteller und Player individuell aufgebaut auf Basis von Schlüsseln des Players und optional auch des Empfängers der Daten.

Das Thema „mobile Sicherheit“ würde allerdings allein ein separates Buch rechtfertigen. Für eine ausführliche Behandlung mobiler Sicherheitskonzepte verweisen wir an dieser Stelle auf die Arbeit von J. Butz ([Butz]), in der die technischen Probleme, Angriffsvektoren und Sicherheitsmaßnahmen mobiler Sicherheit in Firmen umfassend behandelt werden.

10.3.3 Multi-Level Security (MLS)

Die Meinungen zur bereits erwähnten MLS gehen weit auseinander. Einige Sicherheitsexperten halten sie für einen der großen Fehlentwicklungen der letzten dreissig Jahre im Sicherheitsbereich und werfen ihr sogar vor, dass für diese Technik ungerechtfertigt Forschungsmittel von anderen Themen abgezogen worden seien. MLS-basierte Lösungen seien bislang an ihrer Komplexität und Starrheit zu Grunde gegangen.

MLS wird allgemein unter die „Mandatory Access Control“ Verfahren gerechnet, im Gegensatz zur „Discretionary Access Control“ (vgl. Kap. 4). Wir werden hier mit „Mandatory“ einfach alle Verfahren bezeichnen, die die Zugriffskontrolle nicht ausschließlich durch die Rechte des ausführenden Principals bestimmen. Darunter fallen Sandboxes, Capabilities oder eben auch Verfahren wie MLS, die sich einer Label-Technik bedienen, um spezielle Constraints an Daten zu binden.

Abbildung 10.13 zeigt die beiden wesentlichen Methoden der Klassifizierung von Dokumenten in einem MLS System: Hierarchisch über Level und Nicht-hierarchisch über Kategorien. Beides zusammen ergibt ein so genanntes *Security Label*. Die Regeln für den Zugriff sind sehr einfach und orientieren sich an dem im Kap. 4 angesprochenen Bell-LaPadula Modell: Träger unterer Level dürfen auf höhere Level schreiben, aber nicht Dokumente eines höheren Levels lesen (kein „Read-Down“). Träger höherer Levels dürfen von niedrigeren Levels lesen, jedoch nicht auf sie schreiben (kein „Write-Down“).

Die Klassifizierung der Daten ist prinzipiell unabhängig vom Mechanismus ihrer Überwachung. Eine Möglichkeit, die Regeln zu überwachen, stellt ein Referenzmonitor dar, der verhindert, dass im obigen Sinne „verbogene“ Zugriffe passieren.

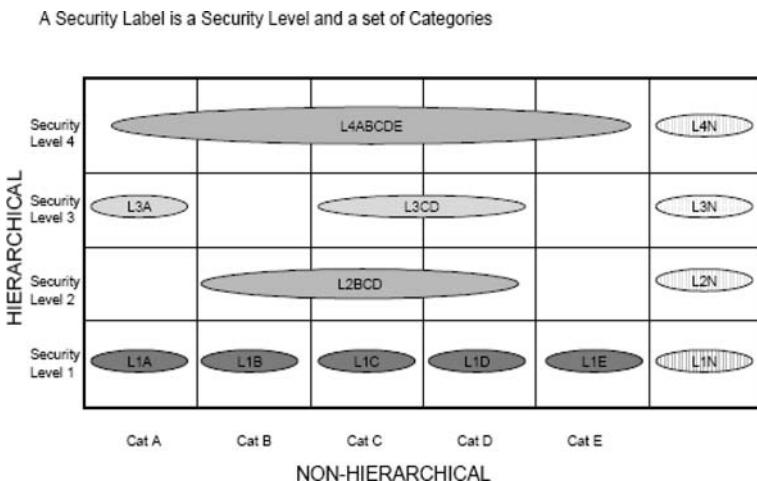


Abb. 10.13 Security Labels (aus [Rayns])

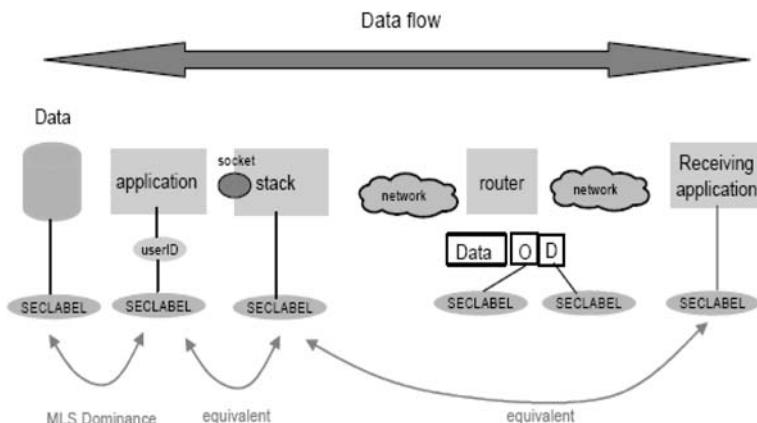


Abb. 10.14 Datentransfer über ein Netzwerk im MLS-Modell (aus [Rayns]): Alle beteiligten Komponenten müssen die Security Labels respektieren

Die Netzwerk-Schnittstelle ist von besonderer Bedeutung für ein MLS System: Sie macht die Daten potentiell mobil und entlässt sie aus der Kontrolle des lokalen Referenz-Monitors. Zwar kann während des Exports geprüft werden, ob die empfangende Stelle die richtigen Labels besitzt (die sie zum Empfang ermächtigen), jedoch kann der Sender nach dem Export die weitere Einhaltung nicht mehr erzwingen.

DRM-Systeme könnten in Abb. 10.14 unten das Vertrauensverhältnis zu der Routing Komponente unnötig machen (durch Verschlüsselung für den Empfänger). Spätestens auf Seiten des Empfängers sind jedoch auch DRM Systeme darauf angewiesen, dass die dortigen Werkzeuge die Labels respektieren. Vor allem

im mobilen Bereich gibt es deshalb Anstrengungen, den Softwarestatus der Geräte durch Schlüssel und Hardware (TNC) zu überwachen bzw. zu garantieren.

Letztlich geht es beim Labelling darum, unabhängig von Nutzer- oder Code-Rechten Sicherheitskriterien in Bezug auf die *Daten* auszudrücken und diese – egal durch welche Technologie – an die Daten zu binden, und zwar in dem Sinn, dass deren Verarbeitung nicht ohne Rücksicht auf diese Klassifizierungen durchgeführt werden kann.

Damit wird deutlich, dass die Granularität der Klassifizierung in Bezug auf die Daten sehr klein sein kann, bis hin zu beispielsweise einzelnen Zeilen einer Datenbank. Das ist eine Granularität, die rollenbasierte Systeme und selbst Sandboxes normalerweise nicht erreichen. Dort wird meist Rechte auf eine ganze Tabelle oder ein ganzes File vergeben. Also könnte man vielleicht sagen, dass Labelling-Verfahren zur objektbasierten Sicherheit zu zählen sind? In Bezug auf die Granularität der Sicherheit ist dies sicher richtig. Allerdings bezeichnet man heute mit „objektbasierter“ bzw. „messagebasierter“ Sicherheit meist die Sicherung eines Datenobjektes durch kryptografische Verfahren, während MLS die korrekte Zuordnung von Werkzeugen und Daten z. B. meist über einen Referenz-Monitor sicherstellt. Wir haben aber eingangs schon gezeigt dass man im Prinzip ein MLS-System auch mit Hilfe von DRM-Verfahren realisieren kann.

Ist nun der schlechte Ruf von Labelling Systemen gerechtfertigt? Bevor wir auf ein praktisches Beispiel eingehen, sei erwähnt, dass es auch in Programmiersprachen, z. B. in Perl, das Prinzip von Labelling in Form von „tainted variables“ gibt: Variablen, die z. B. noch ungeprüften Input aus unsicheren Quellen beinhaltet, können als „tainted“ ausgezeichnet werden. In der Folge werden sie von regulären Funktionen nicht mehr akzeptiert. Labelling kann also auch ähnlich einer Erweiterung des Typsystems realisiert werden. Durch die Definition von Sicherheitsmerkmalen als eigene Typen kann deren Verarbeitung in Software sehr genau kontrolliert werden. Auch das Security Konzept des Netscape Browsers war ursprünglich einmal auf der Basis von Labelling geplant. Der Ansatz wurde dann jedoch durch traditionelle Sandbox- und Permission-Techniken aus der Sprache Java ersetzt. Dennoch gibt es eine hohe Zahl von Sicherheitsproblemen im Netscape-Browser und dessen Nachfolgern, die anscheinend durch die Ausführung von JavaScript in einem anderen Kontext als dem Ursprünglichen verursacht werden – ein klassischer Fall für den Einsatz von Labelling Systemen also. Man sollte die durch Labelling erreichbare Granularität von Sicherheit und ihre Persistenz gegenüber Kontextwechseln nicht vorschnell abtun und sich als Entwickler einfach merken, dass Labelling auf verschiedene Weisen implementiert werden kann:

- als Prozess (in Form eines Referenz Monitors)
- als DRM-System
- im Typsystem einer Sprache

Dass diese Überlegungen keinesfalls nur bloße Theorie sind, zeigte kürzlich ein peinlicher Zwischenfall bei der hessischen Polizei. Umfangreiches, sehr sensitive Datenmaterial aus Befragungen von Verdächtigen wurde statt auf dem Intranet auf

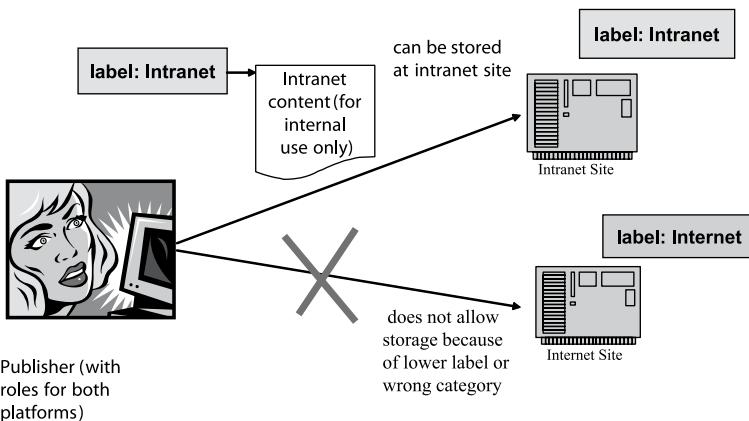


Abb. 10.15 Die „No-Write-Down“ Regel verhindert in einem Label-basierten Publishing-System eine schwerwiegende Sicherheitspanne.

der öffentlichen Homepage der Polizei im Internet publiziert. Die Untersuchungen zur Ursache des Fehlers ergaben, dass es sich schlicht um einen Bedienungsfehler eines Mitarbeiters gehandelt hatte. Dieser hatte Berechtigungen sowohl für den Zugriff auf die Daten als auch für das Publizieren im Intranet und im Internet (s. [heise83767]). Mit einem Label-basierten System wäre der Fehler vermeidbar gewesen (s. Abb. 10.15]) – in einem Zugriffskontrollsystem auf Basis von Discretionary Access Control, d. h. nur basierend auf den Rechten einer Person, gibt es hingegen keine Möglichkeit, einen Zugriff aus Gründen des Codes (falsches Programm) oder der Daten (nicht passende Sicherheitsstufe) zu verhindern.

10.3.4 Enterprise Search Engine Security

Sehr viele Projekte im Bereich von Enterprise Content Management und Publishing beinhalten auch den Aufbau einer Suchfunktion, meist auf Basis einer kommerziellen Search Engine von FAST Search, Autonomy oder der Open Source Lösung Lucene. Im Umfang dieser Lösungen, ihrer Einsatzzwecke und Gebiete vollzieht sich derzeit ein dramatischer Wandel: Immer mehr Datenquellen werden von den Search Engines ausgewertet. Dies schließt mittlerweile operationale Datenbanken ebenso ein wie die traditionellen semi- oder unstrukturierten Quellen wie Webpages und textuelle Dokumente. Sogar Multimedia-Dokumente werden mehr und mehr Teil von Suchergebnissen.

Dieser Wandel (der übrigens auf Seiten der klassischen Analysewerkzeuge wie Data Warehouses oder Statistischen Werkzeugen wie SAS, SPSS etc. ebenfalls nachvollzogen wird) bringt sehr interessante Konsequenzen für die Sicherheitsanforderungen von Suchmaschinen: Die Benutzer wollen ihre Suche über interessante Daten durchführen – und die sind natürlich meist an Zugriffskontrollen gebunden.

Die Dokumente und Daten selbst werden immer intelligenter durch statistische und linguistische Verfahren aufbereitet und der Trend geht eindeutig dahin, dass Benutzer nicht mehr nur ein Dokument als Antwort auf eine Suchanfrage erhalten, sondern eine eventuell aus mehreren Quellen und Dokumenten synthetisierte Antwort. Das Verhalten der Benutzer selbst, ihre Search-Historie, ihre Gruppenzugehörigkeit etc. wird dabei ebenfalls von der Suchmaschine zur Optimierung der Ergebnisse verwendet, was wiederum enorme Problem im Bereich des Schutzes der Privatheit aufwirft.

In diesem Abschnitt wollen wir kurz ein Licht auf zentrale Probleme für die Sicherheit von Unternehmen und Personen im Bereich von Enterprise Search Engines untersuchen. Ein guter Einstieg in die Problematik findet sich bei [Vivi].

10.3.4.1 Architektur einer Enterprise Search Lösung

Oft ergeben sich bereits aus der Architektur einer kommerziellen Lösung wichtige Ansatzpunkte für eine Analyse ihrer Sicherheitsproblematik. Die folgende Abb. 10.16 stellt grob schematisch die Architektur einer Enterprise Search Engine dar. Wichtig dabei sind die jeweiligen Datenflüsse, Kontrollflüsse und eventuelle Bedrohungsmodelle, die sich daraus ableiten lassen.

Man kann deutlich erkennen, dass der Fluss von Daten bei einer solchen Lösung klar von den Backend-Datenquellen hin zu den Benutzern geht. Moderne Search Engines halten im Index der Platform übrigens große Teile der indizierten Dokumente, um bei Treffern auch die Umgebung der Treffer den Benutzern anzeigen zu können. Die Folge ist, dass der Index praktisch den Inhalt der Datenquellen dupliziert.

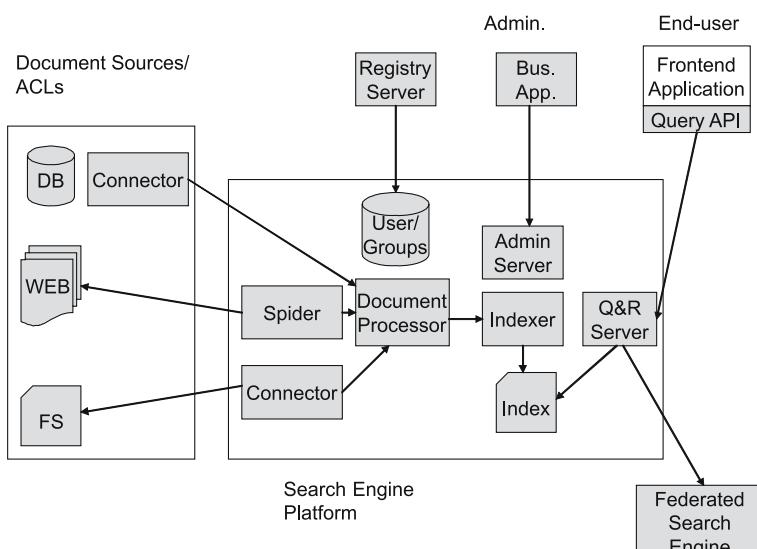


Abb. 10.16 Enterprise Search Engine

Die Daten der Backend-Systeme gelangen über sogenannte *Spider* oder *Connectors* in den Document Processor wo sie aufbereitet werden (z. B. durch Zugriffe auf Dictionaries angereichert, mit zusätzlichen Metadaten ausgezeichnet etc.). Das entscheidende Sicherheitsproblem entsteht in dem Moment, wo die Daten der Backend-Systeme nicht mehr öffentlich, sprich frei zugänglich sind, sondern einer Zugriffskontrolle unterliegen.

Search Engines im Intranet lösen das Problem meist so, dass sie zusätzlich zu den Daten auch die jeweiligen Access Control Lists auslesen und in den Index mit aufnehmen. Bei einer Query eines Benutzers – der nun natürlich authentisiert sein muss – werden dessen Zugriffsrechte mit denjenigen der Ressource abgeglichen. Nur wenn die Rechte deckungsgleich sind, wird der Zugriff erlaubt.

Ein interessanter Fall sind sog. „Federated Searches“, bei denen die Anfrage eines Benutzers an andere Search Engines weitergegeben wird. Dies scheint sicherheitstechnisch ohnehin eine sehr elegante Lösung zu sein: Statt die Daten den Backends zu entreißen, wird die Anfrage an deren eigene Suchfunktion weitergegeben und nur die Resultate werden zurückgeliefert. Dadurch bleiben die Backend-Systeme in der Lage, eine eigene Rechteprüfung anhand der Identität des Benutzers, der die Anfrage gestellt hat, durchzuführen. Diese Identität muss natürlich (z.B. in Form eines signierten Tokens) von der Search Engine und die föderierten Systeme weitergeleitet werden.

Leider ist die Qualität von föderierten Search Engines bei weitem nicht so gut wie die einer großen zentralen Engine. Der Grund dafür liegt darin, dass die jeweiligen Engines sehr verschiedene Kriterien für die Relevanz von Treffern verwenden und einige Berechnungsformeln für die Relevanz eines Treffers die Häufigkeit von Termen innerhalb einer Kollektion von Dokumenten verwenden [Cher]. Diese Häufigkeit ist natürlich in jeder Datenquelle verschieden und entsprechend verschieden fällt die Berechnung der Relevanz aus. Anders gesagt: Damit die hundert „relevantesten“ Ergebnisse aus mehreren Search Engines zusammengefasst werden könnten, müssten sich diese Engines vorher auf gemeinsame Kriterien einigen und ihre Meta-Daten bezüglich ihrer Kollektionen austauschen.

Auf Grund dieser Problematik wird föderatives Suchen nur dann eine Rolle spielen, wenn ein Backend absolut nicht erlaubt, dass seine Daten von einer Search Engine extrahiert werden.

Manche Search Engines bieten noch eine weitere Möglichkeit, die Access Control der Backend-Systeme dennoch weiter nutzen zu können: Sie extrahieren zwar alle Dokumente im Index, jedoch fragen sie bei jedem Treffer die entsprechende Datenquelle danach, ob der anfragende User auch die nötigen Rechte besitzt. Dieses Verfahren kann jedoch lediglich verhindern, dass sich die Rechte des Users in der Zeit zwischen dem Indizieren des Dokumentes und seiner ACL geändert hätten, ohne dass die Search Engine dies bemerken würde. Das Hauptargument gegen dieses Verfahren ist dann auch die dramatische Verschlechterung der Performance, da jede Query zu einer hohen Anzahl von Rechte-Anfragen an die Backend-Systeme führt. Besser sind Möglichkeiten, per Push-Nachricht die Search-Engine über Änderungen der Rechte an Dokumenten zu informieren.

10.3.4.2 Überlegungen zur Sicherheit

Das Kernproblem der Sicherheit wurde bereits angesprochen: Die Search Engine „entreißt“ den Datenquellen ihre Daten und verwaltet sie selbst. Mechanismen der Zugriffskontrolle innerhalb der Datenquellen sind damit hinfällig – ein Faktum, das bei IT-Sicherheitsexperten ziemliche Unruhe aufkommen lässt.

Die Konsequenzen für die Sicherheit der Search Plattform sind daher enorm: Die Enterprise Search Plattform erbt alle Sicherheitsklassifikationen ihrer jeweiligen Datenquellen. Anders gesagt: Verwaltet die Search Engine Daten mit einer gewissen Sicherheitsklassifikation in den Backend-Systemen, dann gelten die Regeln für den Umgang mit den Daten auch für die Search Engine: Datenklassifikation ist sicherheitstechnisch transitiv!

Das Security Context Diagramm unten zeigt die Search Plattform in ihrem Systemkontext, annotiert mit sicherheitsspezifischen Anforderungen. Es ist leicht zu sehen, dass die Verbindungen zwischen Systemen und der Search Plattform einen sicheren Transport garantieren müssen – z. B. auf der Basis von SSL. Systeme, aus denen Input kommt (Backends), bzw. die Output aufnehmen (Front-Ends), müssen sich gegenüber der Plattform authentisieren und umgekehrt. Dies kann auf Basis von Zertifikaten geschehen und wird im folgenden Kapitel beschrieben. Die Search Plattform hingegen muss sowohl Quell- als auch Zielsysteme zusätzlich autorisieren und darf sich nicht nur auf eine gelungene Authentisierung verlassen: Die Tatsache, dass ein Konnektor oder ein Backendsystem ein gültiges Zertifikat vorlegt und damit durch die Search Plattform authentisiert werden kann, reicht keineswegs aus, um auch automatisch einen Zugriff durch das System zu erlauben. Auch diese Problematik wird gleich noch im Detail besprochen.

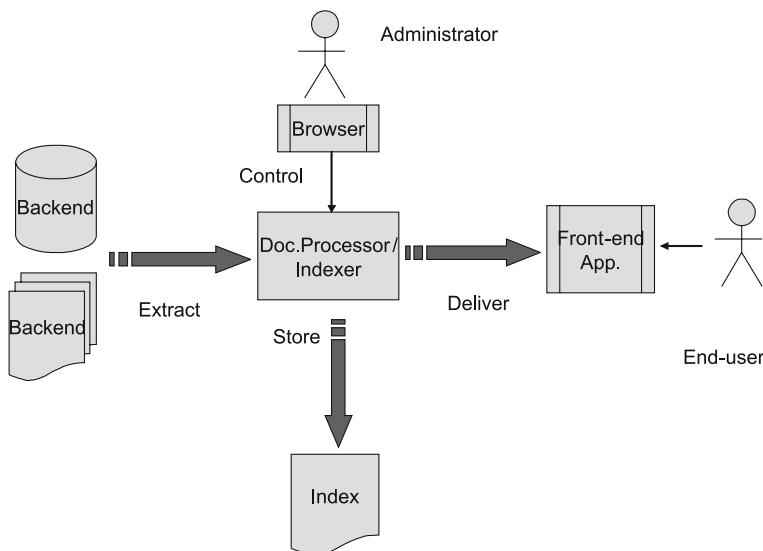


Abb. 10.17 Extraktion von Daten aus dem Backend

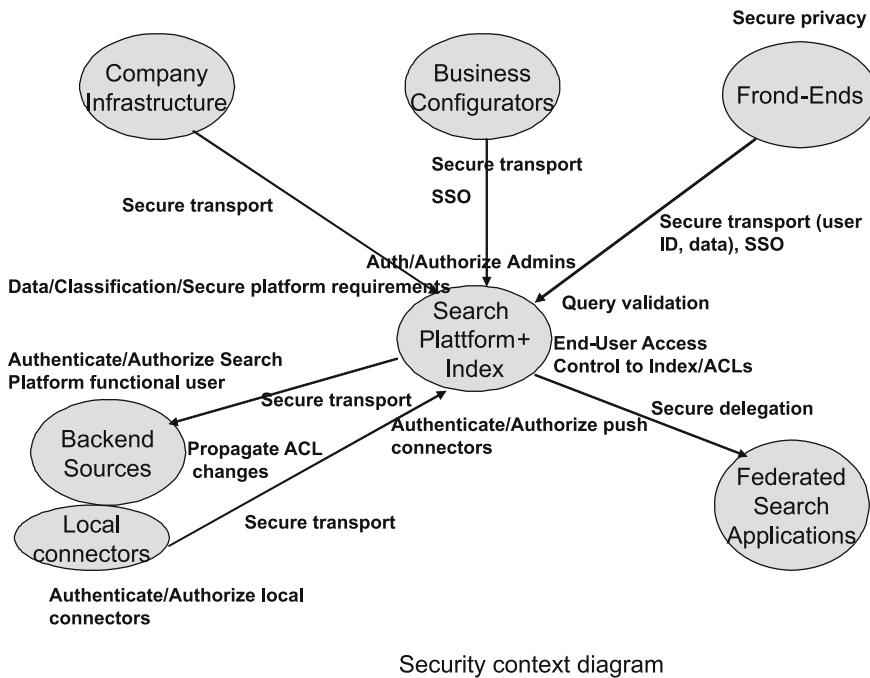


Abb. 10.18 Security Context Diagramm für die Search Engine

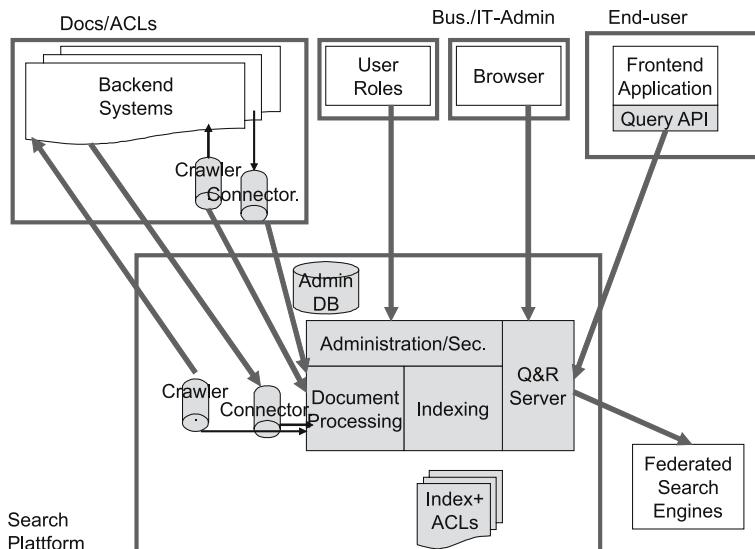


Abb. 10.19 Bedrohungsmodelle für die Search Engine

Ganz grob lässt sich die Sicherheitsproblematik in weitere Teile aufteilen: Die Extraktion der Daten und ihr Transport, die Verarbeitung der Daten innerhalb der Plattform inklusive Zugriffskontrolle und die Authentisierung der User und ihre Queries über die Front-end Applikationen. Damit haben wir die klassischen Bedrohungsmodelle „Transport“, „Plattform“, „Zugriffslogik“, „Intranet-Backends“ und „User“ (s. Abb. 10.19).

Beginnen wir unsere Diskussion mit dem Bereich der Transport-Bedrohungen. Hierunter fällt die integre und vertrauliche Übertragung von Daten wie auch die korrekte Adressierung des Ziels. In unserem Fall würde man verlangen, dass sich die beteiligten Systeme gegenseitig authentisieren und durch eine zusätzliche Access Control verhindern, dass unbefugte Systeme Daten z. B. in das Document Processing einspeisen könnten. Die Übertragung würde dann über einen sicheren Kanal (SSL) erfolgen. Im Falle der Verwendung anderer Search Engines muss über das Mittel der sicheren Delegation die Identität des Anfragenden auch an diese geleitet werden, damit sie ihre Zugriffskontrollen durchführen können.

Wesentlich kritischer sind die Überlegungen zur Backend-Security: Wie sollen Daten extrahiert werden? Wenn das Backend System über eine Zugriffskontrolle verfügt, dann muss dem Spider bzw. dem Connector der Search-Engine eine Identität zugeordnet werden, die über genügend Rechte verfügt, um sämtliche Daten extrahieren zu können. Diese (funktionale) Identität sollte unbedingt über ein BBS verwaltet werden.

Das Diagramm in Abb. 10.20 zeigt die beiden grundsätzlichen Methoden der Extraktion: Entweder durch eine Software, die im Kontext der lokalen Search Plattform läuft oder durch eine Software, die als Teil des Backends läuft und die Daten an die Search Plattform schickt. Beide Lösungen bieten unterschiedliche Angriffsmöglichkeiten.

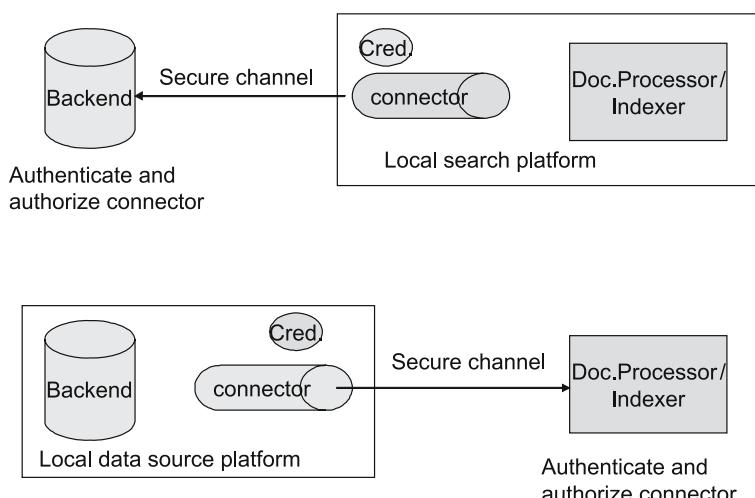


Abb. 10.20 Möglichkeiten der Datenextraktion aus dem Backend

Wieso ist die Identität der Zugriffssoftware bzw. deren Rechte von solcher Bedeutung? Natürlich gibt es auch die Möglichkeit, die Zugriffe auf Backend-Systeme innerhalb der Search-Plattform zu konfigurieren. Natürlich wird man dort versuchen, keine Systeme anzusprechen, deren Daten nicht über Search verfügbar sein sollen. Aber dies ist nicht gleichbedeutend mit echter Security: Eine falsche Konfiguration der Search Engine führt schnell dazu, dass die falschen Backend-Systeme ausgewertet werden.

Es gibt noch eine subtile Gefahr bei der Datenextraktion: Der Einsatz von Spidern, also Programmen, die den Links innerhalb von Dokumenten immer weiter nachgehen und weitere Dokumente einbeziehen, stellt in dem Moment eine Gefahr dar, in dem Mitarbeiter klassifizierte Daten als Kopien auf öffentlichen Datenquellen herumliegen lassen. Diese können gefunden werden und – da nicht durch Access Control geschützt – versehentlich über die Search Engine zugänglich werden.

Da die Identität der Connectors und Spider an bestimmte Credentials gebunden ist, müssen diese sicher innerhalb der Search Engine verwaltet werden. Dies führt uns über zum Problem der Plattform-Sicherheit. Da wie bereits besprochen die Daten der Backend-Systeme im Index der Search Engine gehalten werden, stellt die Search Plattform selbst ein beträchtliches Sicherheitsrisiko mit entsprechenden Sicherheitsanforderungen dar (s. Abb. 10.21).

Nicht nur müssen die Anfragen der Benutzer korrekt mit den gespeicherten ACLs der Dokumente verglichen werden, sondern Zugriffe über andere Wege (Datensystem, Remote Access, Administration etc.) müssen verhindert oder kontrolliert und notiert werden. Das bedeutet, dass die Search Plattform auf einer Trusted Computing Plattform mit streng kontrollierten Zugriffen laufen muss. Nur dadurch kann

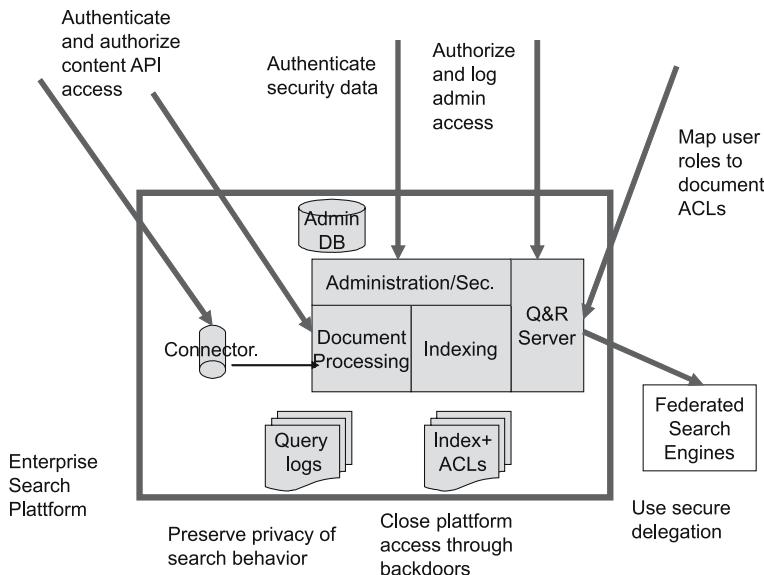


Abb. 10.21 Sicherheitsanforderungen für die Search Engine

die Plattform dann auch die Anforderungen an die Privatheit der Benutzer garantieren: Suchanfragen geben eine sehr deutliche Auskunft über die Interessen und Arbeitsgebiete von Personen. Die Benutzer haben deshalb ein Recht darauf, dass ihre Aktionen nur im Rahmen des rechtlich Zulässigen verwendet werden.

Einige Hersteller gehen daher so weit, für ihre Plattform eine Einbettung in eine Firewall-Zone zu verlangen, so dass alle Zugriffe von Systemen, Konnektoren oder Benutzern genau gefiltert werden können.

Die sichere Speicherung des Indexes selbst ist ebenfalls Teil der Plattform-Security. Dazu gehört auch die Verfügbarkeit der Search-Engine zu sichern. Dies ist angesichts Terabyte großer Indizes nicht ganz einfach. So sollte eine Cluster-Lösung Pflicht sein. Den Index löschen und anschließend einen neuen aufbauen stellt hingegen keine sehr praktikable Lösung dar, wenn das Aufbauen eines neuen Index mehrere Tage benötigt.

Selbst Authentisieren sollte die Search Plattform die Benutzer hingegen nicht. Stattdessen sollten bereits authentisierte User anhand von SSO Token identifiziert werden. Die Search-Plattform braucht darüber hinaus Zugriff auf ein Repository mit Usern und deren Rollen, um eine Suchanfrage korrekt mit den gespeicherten ACLs der Dokumente vergleichen zu können.

Die Services der Plattform sollten unter einem Functional User mit geringen Rechten laufen. Der User sollte ebenfalls in einem Berechtigungssystem verwaltet werden.

Welche Probleme ergeben sich durch die Benutzer? Da sehr viele Enterprise Search Plattformen im Intranet von Firmen betrieben werden, spielen DOS Angriffe hier eine geringere Rolle. Da die meisten Plattformen ihre Indizes als normale Files und nicht über Datenbanken verwalten, stellen SQL Injection Angriffe ebenfalls keine Gefahr dar. Interessanter ist die Frage, ob Benutzer durch geschickte Queries an Informationen gelangen könnten, für die sie streng genommen keine Berechtigung besitzen. Aus dem Bereich der Datenbanken sind derartige Techniken bekannt. Für Search-Engines stellt dies nur dann eine Gefahr dar, wenn sich die Betreiber dazu entschieden haben, Treffer immer anzuzeigen, auch wenn ein Benutzer auf das dahinterliegende Dokument keine Leserechte besitzt. Dann besteht die Möglichkeit, durch geschickte Queries Stück für Stück das Originaldokument quasi zu rekonstruieren. Dieses Vorgehen ist im Detail bei [KleinA] beschrieben.

Das Problem der Privatheit im Zusammenhang mit den Nutzern der Search Engine wurde bereits erwähnt. Vorsicht sollte man walten lassen bei kollaborativen Features wie dem Anzeigen der „10 populärsten Suchanfragen“. Manipulationen, wie sie sonst in öffentlichen Suchmaschinen wie Google unter dem Namen „Search Engine Optimization“ bekannt geworden sind, brauchen für Intranets hingegen nicht befürchtet werden.

10.3.4.3 Test und Deployment

Search Engines stellen ebenso wie Dokument-Management-Systeme ein Problem für die kontrollierte Ausbreitung von Software innerhalb einer Firma dar: Sie müs-

sen häufig angepasst werden und die Benutzer erwarten sehr kurze Reaktionszeiten für ihre Änderungswünsche. Dies widerspricht den Forderungen der IT-Security nach genau festgelegten Testzyklen mit entsprechend hohem Zeitaufwand.

Wie lassen sich dennoch im Bereich Search gewisse Tests bezüglich Funktion und Sicherheit durchführen? Die Feststellung der Relevanz von Suchergebnissen ist die wohl schwierigste Fragestellung im Bereich Search überhaupt. Hier bleibt wohl zunächst nicht viel anderes übrig, als bestimmte Anfrage/Antwort Kombinationen automatisch regelmäßig durchzuführen und die Ergebnisse auszuwerten. Die überaus große Zahl von Einstellungsmöglichkeiten der Search-Engines macht es sehr schwer, im Einzelfall nachzuweisen, wieso etwas gefunden bzw. nicht gefunden wurde.

Wie lässt sich testen, ob eine Search Engine ungewollte Inhalte liefert? Hier hilft nur die regelmäßige Suche nach kritischen Termen wie speziellen Tags („confidential“, „secret“ etc.), um falsch behandelte Dokumente zu finden.

Natürlich hängt das Risiko einer Search Engine zu einem guten Teil von der Kritikalität der verarbeiteten Daten ab. Die Benutzer werden jedoch zunehmend fordern, dass auch ihre Mail, Datenbanken etc. indiziert werden, um einen einfachen und integrierten Zugriff auf ihre Informationen zu erhalten. Als Notlösung für besonders kritische Daten bleibt dann noch die Verwendung von Föderation: Dabei werden die Anfragen zusammen mit der Identität des Benutzers an das jeweilige Backend weitergeleitet.

Kapitel 11

Sicherheit der Infrastruktur

Die Absicherung der eigenen Infrastruktur ist eng an die folgenden Fragen gekoppelt: Wo sind die Grenzen des Abzusichernden, und wo sind die Grenzen der Firma oder der Heiminstallation? Früher waren diese Fragen nach den Grenzen des Sicherheitsbereiches einfach entschieden durch eine Mauer oder einen Zaun. Heute sind diese Fragen viel schwieriger zu beantworten: Wo hört das Internet auf und wo fängt das eigene Netzwerk an? Viele Firmen sind schließlich weltweit verteilt und ihr Intranet erstreckt sich über alle Kontinente.

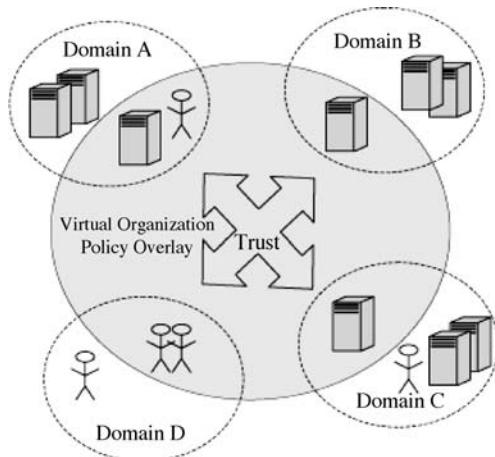
Entsprechend verliert ein Sicherheitskonzept, das auf räumlicher Abgrenzung beruht, langsam an Bedeutung. Es ist zu starr in Zeiten der mobilen Arbeit, weil es den wechselnden Standorten der Rechner (Laptops, PDAs) nicht Rechnung tragen kann. Diese überwinden leicht die Firmengrenzen. Das räumliche Abgrenzungs-konzept – auch *Perimeter Security* genannt – wird ersetzt durch Zonen verschiedener Sicherheit wie sie z. B. typisch sind für Flughäfen. Solche Zonenkonzepte finden sich jetzt auch in den Firewalls und DMZs der größeren Firmen.

Aber das Kernkonzept des sicheren Raumes bleibt der Schlüssel. Es ist zwar nicht mehr der physische Schlüssel, der wirkliche Schlosser sperrt, sondern dieser Schlüssel wird ersetzt durch den kryptografischen Schlüssel, der eigene Räume mit Zutrittsrechten schaffen kann. So kann die Frage nach den Grenzen einer Firma mit Dan Geer folgendermaßen beantwortet werden: Die Grenzen einer Organisation werden bestimmt durch die Macht über Schlüssel und ihre Akzeptanz. Überall, wo eine Organisation ihre Schlüssel durchsetzen kann ist sie „daheim“.

Ganz extrem wird dieses Konzept sichtbar in der sog. Virtuellen Organisation (VO) (s. Abb. 11.1) – einem rechtlichen, wirtschaftlichen oder sonstigen Überbau über bestehende Infrastrukturen, die sich zu einem Rechnerverbund zusammengeschlossen haben. Die VO besitzt selbst keinerlei physisches Eigentum an den Rechnern und deren Verwaltung. Ihr Eigentum schützt sie ausschließlich über ihre kryptografischen Schlüssel.

In diesem Kapitel wollen wir zunächst am Beispiel von Demilitarisierten Zonen zeigen, wie Infrastrukturen im Allgemeinen abgesichert werden können. Später gehen wir auch auf Spezialfälle wie die Absicherung von Servern auf einer Maschine ein.

Abb. 11.1 Virtuelle Organisation



11.1 Absicherung der Infrastruktur durch Firewalls und DMZs

In diesem Abschnitt wollen wir nicht etwa die Grundlagen der Netzwerksicherheit und des Filterns von Requests anhand von Regelsets erarbeiten. Dazu gibt es ausreichend Literatur, die sich vor allem an die Spezialisten für Netzwerksicherheit richtet, z. B. [ZCC] oder [BSI_FW]. Kennzeichnend für die Filter-Maßnahmen ist ihre relative Unabhängigkeit von der speziellen Softwarearchitektur einer Applikation. So werden zum Beispiel Filter auf Maschinen konfiguriert, die selbst keine Teile der Applikation beherbergen. „Relative Unabhängigkeit“ bedeutet, dass die Applikationsentwickler den für die Infrastruktur zuständigen Leuten bestimmte Informationen über die Applikation (z. B. verwendete Protokolle, Ports, Request/Response Struktur etc.) mitteilen müssen, dass die Implementation der Sicherheitsmaßnahmen jedoch im allgemeinen keine Änderungen der Applikation bzw. ihrer Installation benötigen. Dies ist jedoch nur für einen Teil der Sicherheitsmaßnahmen wahr. Ein anderer Teil ist eng mit der Applikationsarchitektur verknüpft. Fehler in der Architektur (oder mangelnde Rücksichtnahme auf die Regeln innerhalb der Sicherheitszonen einer Firewall) können dazu führen, dass eine Applikation aus technischen oder Sicherheitsgründen keine Erlaubnis zum Einsatz erhält.

Genau diese Form von Security – an der Nahtstelle zwischen Netzwerk- und Software-Sicherheit – wollen wir hier besprechen, um speziell Softwareentwicklern eine bessere Planung ihrer Architekturen in Bezug auf Installation und Ausführung innerhalb von Sicherheitszonen zu ermöglichen. Einige dieser Sicherheitsmaßnahmen wirken sich tatsächlich aus bis in die Kernarchitektur einer Web-Applikation.

Zunächst wollen wir anhand einer einfachen Firewall-Architektur die wichtigsten Grundregeln für die Absicherung der Infrastruktur und ihren sicherheitstechnischen Hintergrund besprechen.

11.1.1 Firewall-Architekturen

Firewalls (eigentlich *Firewall-Architekturen*) sind heute überwiegend zentral aufgebaute Wächter an der Grenze von Intranet und Internet. Sie folgen dem Design Pattern des Interceptors und blockieren bzw. filtern Netzwerk-Verkehr durch das Abarbeiten von Filterregeln, die sich auf die eintreffenden Pakete bzw. Requests beziehen. Durch ihre zentrale Rolle als gewollter „Flaschenhals“ in der physischen Infrastruktur sind ihre Konfigurationen extrem komplex und Performanceprobleme unvermeidlich. Alternative Architekturen, die vermehrt auf so genanntes Host-based-Filtering setzen (analog zu den bekannten Personal Firewalls auf Heim-PCs, jedoch auf der Basis sicherer Betriebssysteme) oder sogar Network-Edge Firewalls, die in die Netzwerkkarten der Rechner eingebaut sind, haben sich bisher kaum durchsetzen können. Ihr Potential zur Vereinfachung der Administration und Konfiguration ist aber sehr hoch, da jeder Host-based Firewall nur ein sehr kleines Regelset beachten muss – eben das für einen einzigen Host. Auch Intrusion-Detection-Systeme (IDS) können ähnliches leisten und haben in der Vergangenheit bereits bewiesen, dass sie Angriffe schneller erkennen können als z. B. ein Hersteller von Anti-Virus Software neue Signaturfiles erstellen und versenden könnte. Host-basierte Firewalls und IDS können z. B. an den Protokollen der netzinternen Kommunikation einen Angriff erkennen, ohne dass sie hierfür die Signatur der Payload (also den Virus bzw. Wurm) kennen müssen.

Der zentrale Netzwerk-Firewall ist traditionellerweise als Sandwich aufgebaut und wird auch als *Demilitarized Zone* (DMZ) bezeichnet. Die Sandwich-Architektur erklärt sich aus der Tatsache, dass ein einziger Firewall vor dem Webserver nicht ausreichend ist zum Schutz der hinter dem Webserver auf den Application Servern gelagerten, hoch sensiven Daten: Der Webserver muss für Anfragen aus dem Internet erreichbar sein und ist somit einem erhöhten Risiko durch bösartig geformte Requests ausgesetzt. Es besteht deshalb immer die Gefahr, dass der Webserver trotz des vorgelagerten Firewall einem Angriff zum Opfer fällt. Um das dahinter liegende Intranet vor Angriffen durch einen kompromittierten Webserver zu schützen, ist ein zweiter Firewall erforderlich, der die Requests des Webservers filtert.

Die einfache DMZ (s. Abb. 11.2) bildet drei Sicherheitszonen aus: Ein äußeres und ein inneres Subnetz sowie das Intranet, geschützt durch Paketfilter an den Grenzen und in der Mitte verbunden durch einen Bastion Host (oder Application Gateway), der intelligente Proxy-Software verwendet, um die beiden Netze zu verbinden. Fehlt ein solcher Proxy für eine Applikation, dann ist sie nicht von außen erreichbar, da das IP-Forwarding auf dem Netzwerkevel des Bastion Hosts nicht erlaubt ist. Die beiden Netzwerkkarten im Bastion Host (deshalb der Name *dual homed*) sind somit nicht direkt verbunden und die Daten müssen über den Umweg der Proxyssoftware von der äußeren in die innere Zone gebracht werden. Somit sind sie der Inspektion zugänglich und es wird dadurch ein starker Schutz der inneren Zone erreicht. Verbunden mit den Zonen sind gewisse Sicherheitskonzepte. Sie bestimmen,

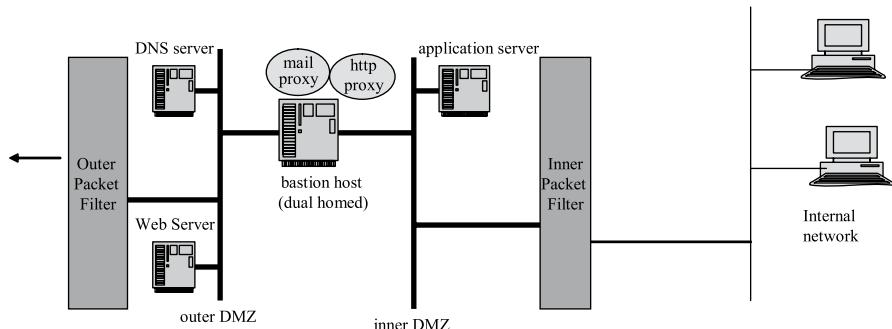


Abb. 11.2 Einfache DMZ

- welche Art von Applikation in welcher Zone laufen darf;
- welche Protokolle in welcher Zone zulässig sind;
- ab wann Requests authentisiert sein müssen und
- wer wie auf diese Zonen zugreifen darf, z. B. für Software-Wartung.

Da die äußere Zone durch das Fehlen eines Bastion Hosts zweifellos weniger geschützt ist als die innere Zone, dürfen bei vielen Firmen dort nur Applikationen laufen, die z. B. keine direkte Verbindung zum Intranet der Firma haben. Oft ist das ein Web-Server mit statischen Seiten, die über einen Replikationsmechanismus auf den Server kopiert werden und in der Folge völlig ohne Verbindung nach hinten auskommen. Auch sog. Victim-Hosts – auf ihnen laufen im Allgemeinen als gefährdet oder gefährlich angesehene Applikationen – werden dort aufgestellt, in der Erwartung, dass sie zwar eventuell von einem Angreifer übernommen werden, aber andererseits keinen direkten Zugriff auf die inneren Zonen oder das Intranet ermöglichen.

Ein weiteres Beispiel für in der äußeren Zone untergebrachte Applikationen sind solche, die nicht authentisierte Requests von außen zulassen, d.h. Applikationen, die keine Authentisierung benötigen. Die öffentlichen Seiten eines Firmenauftritts gehören dazu, wobei die Tatsache, dass diese Seiten öffentlich sind, nicht bedeutet, dass sie nicht besonders geschützt werden müssten – die vielen Fälle von so genannten *Homepage-Defacing*, bei denen die Webpräsenz einer bekannten Firma von Angreifern durch eigene Inhalte ersetzt wurde, zeigen, dass diese Regel in der Vergangenheit nicht immer beachtet worden ist.

In Abb. 11.3 ein Modell einer DMZ mit verschiedenen Applikationen und Sicherheitszonen. Die DMZ ist um eine weitere Zone erweitert worden, um Angriffe aus dem Intranet besser begegnen zu können. So erfolgt die Wartung von Applikationsservern nur über Software, die sich in dieser Zone befindet und nicht direkt aus dem Intranet.

Die Grenzen einer zentralen Firewall-Architektur zeigen sich, falls die am Internet angebotenen Dienste erfolgreich sind: Je mehr Applikationen und Applikationsserver in der DMZ untergebracht werden (mit jeweils verschiedenen Protokollen zum Intranet), umso komplexer wird die Konfiguration der beteiligten

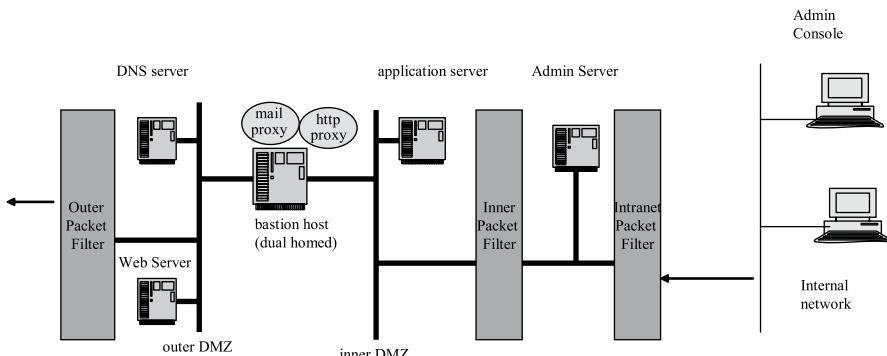


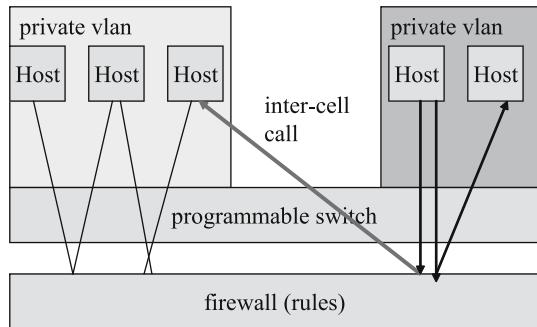
Abb. 11.3 Erweiterte DMZ

Firewalls. Der Trend zu intelligenten User-level Proxy-Programmen zur Filterung statt simpler Kernel-level Paketfilterung führt darüber hinaus schnell zu Performanceproblemen.

Komplizierte Protokolle erschweren die Filterung zusätzlich. Dabei handelt sich beispielsweise um Multimedia-Protokolle wie etwa für Videokonferenzen oder SIP (Session Initiation Protocol), bei denen mit mehreren, teils dynamisch zugeordneten Ports auf unterschiedlichen Protokollen gearbeitet wird. Da die Regeln der meisten Firewalls nicht dynamisch zur Laufzeit geändert werden können (bzw. werden sollen), ist das Filtern solcher Applikationen problematisch. Entweder werden zu viele Löcher geöffnet oder die Applikationdaten fließen nicht korrekt. Erste Ansätze zu einer dynamischen Steuerung von Firewalls und Filterung gibt es mit FCP (Firewall Control Protocol, siehe hierzu <http://www.ietf.org/fcp/>), das im Umfeld von SIP entstanden ist. Weitere, für die Filterung ungünstige Protokolleigenschaften bestehen darin, dass Kommandos von beiden Seiten eintreffen (bei Client/Server Protokollen) oder in verschlüsselten Protokollnachrichten, die nicht am Firewall terminieren und somit auch nicht entschlüsselt werden sollen – sie können somit auch nicht gefiltert werden.

Für die Softwareentwicklung ist es unabdingbar, sich mit den Regeln der jeweiligen Sicherheitsinfrastruktur intensiv auseinanderzusetzen und problematische Designs und Protokolle von Beginn an zu vermeiden. Ein besonderes Augenmerk ist dabei auf die Zulassung von Protokollen für Zonen zu richten. Beispielsweise kann es Vorschriften geben, dass beim Übergang von einer Zone in eine andere das Protokoll gewechselt werden muss (zum Beispiel von http nach SQL). Schreibt das Softwaredesign überdies Verschlüsselung vor, zum Beispiel mit SSL, ist die Infrastruktur (also der Bastion Host etc.) über diese zusätzliche Last zu informieren. Ebenso schreiben viele Firmen genau vor, welche Werkzeuge für File-Transfer innerhalb bzw. von und zur DMZ verwendet werden dürfen. Zentrales Merkmal von solchen Werkzeugen ist gar nicht so sehr die Sicherung der Daten durch Verschlüsselung während des Transportes, sondern vielmehr die Autorisierung und Protokollierung des Transfers. Mehr dazu und zum Sessiondesign weiter unten.

Abb. 11.4 VLAN-basierter Firewall



Neuere zentrale Firewall-Architekturen versuchen dem Problem des gewollten „Flaschenhalses“ durch VLAN-Technologie zu begegnen (s. Abb. 11.4).

Damit können mehr Zonen mit verschiedener Konfiguration geschaffen werden, die dann granulare Sicherheitskonzepte repräsentieren können. Die Anzahl der physischen Firewalls lässt sich zudem reduzieren, indem alle Kommunikation über ein und denselben Firewall gezwungen wird. Meist wird in solchen Umgebungen auch eine sog. Administrationszone definiert. Die Wartung von Applikationen innerhalb dieser Firewall-Umgebung darf ausschließlich vom Intranet aus und nur über diese Zone durchgeführt werden. Die Konfiguration wird dadurch nicht wirklich einfacher, jedoch auf weniger Maschinen reduziert.

Abschließend zu dieser kurzen Einführung in Firewall-Architekturen noch zwei Fakten, die viele Softwareentwickler sicher verblüffen werden:

- Der Zugriff vom Intranet auf die DMZ ist potentiell genauso kritisch zu sehen wie der vom Internet aus. Eine gute DMZ muss deshalb auch gegen falschen Input und (böswillige oder fehlerhafte) Umkonfiguration vom Intranet aus geschützt sein. Die Grenzen von Paketfiltern werden schnell sichtbar, wenn man sich einmal vor Augen führt, auf Grund welcher Daten der Filter seine Entscheidung treffen muss. Dazu braucht man sich nur einmal die Datenstrukturen eines IP-, TCP- und UDP- Paketes vergegenwärtigen und die Felder markieren, die für Filtering interessant sind. Schnell zeigt sich, dass auf dieser Ebene nur relativ grobe Blockaden errichtet werden können, während das interne Applikationsprotokoll unbekannt bleibt. In Zukunft wird also vermehr Wert auf intelligente, Applikation-Level Proxy Software gelegt werden müssen, die auch den *Inhalt* der Applikationsdaten bei der Filterung berücksichtigen kann.

Hier ein Beispiel zu dieser Problematik von Tobias Klein ([Klein], s. auch Abb. 11.5): Ein Angreifer platziert mit Hilfe eines Buffer-Overflows einen kleinen Proxy-Server (so genannte „Zecke“) in einer laufenden Anwendung mit bestehender Verbindung. Daraufhin sendet der Angreifer innerhalb regulärer Nachrichten des Anwendungsprotokolls kleine Betriebssystem-Aufrufe an den Proxy, die dieser an das Betriebssystem weiterreicht und die Ergebnisse an den Angreifer zurück liefert. Somit ist keine neue Verbindung erforderlich. Selbst ein „stateful“ Firewall, der sowohl den Status der Verbindungen wie auch die

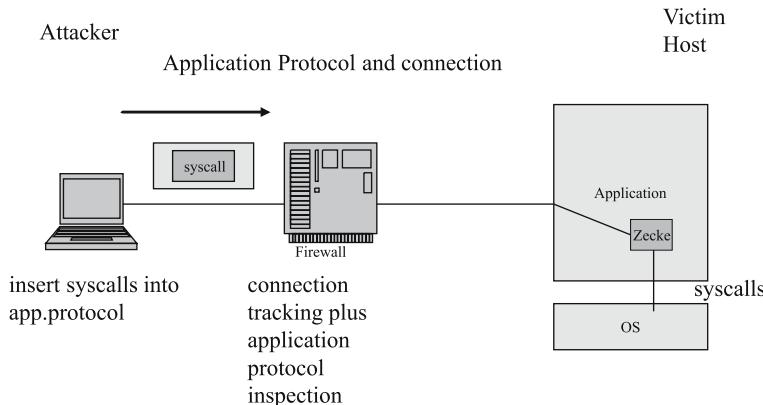


Abb. 11.5 Angriff, der einen Firewall auf Anwendungsebene überwindet

verwendeten Anwendungsprotokollen analysiert, wird nichts verdächtiges finden, da eine bestehende Verbindung als Vehikel benutzt wird und die Betriebssystem-Aufrufe innerhalb der Pakete des Anwendungsprotokolls platziert sind.

- Schutzversuche für Applikations- oder Administrationsfunktionen, die auf der Prüfung einer Absenderadresse beruhen, sind ziemlich wirkungslos, da das Fälschen des Absenders eines IP-Paketes lediglich das Einfügen einer anderen IP-Adresse in das Absenderfeld des IP-Paketes mit Hilfe einer geeigneten Software erfordert.

11.1.2 Reverse Proxy als Point-of-Contact

In den letzten Jahren hat sich der sog. *Reverse Proxy* (RP) als Kernstück einer jeden Firewall herausgestellt. Es handelt sich dabei um einen hochsicheren Rechner (dort enden die SSL-Sessions der Kunden, deshalb kennt er auch deren Credentials), der als erster Knoten der Infrastruktur Requests von außerhalb – sprich dem Internet – in Empfang nimmt. Er stellt somit den „Point of Contact“ für Kundenrequests dar.

Ein Reverse Proxy implementiert das Interceptor Design Pattern, indem er Requests abfängt und manipuliert, prüft, etc. Voraussetzung dafür ist, dass es keine Möglichkeit gibt, den Reverse Proxy zu umgehen. Als Reverse Proxy vertritt der Proxy die eigentlichen Application Server, die sich weiter hinten in der Firewall-Infrastruktur befinden.

Von außen gesehen, steht der Reverse Proxy tatsächlich für die eigentlichen Server, denn diese bleiben für den Kunden unsichtbar. Eine Konsequenz daraus ist, dass der Reverse Proxy die Domain der Firma vertritt und eine eventuelle SSL Verbindung zwischen Kunden und Firma genau beim Reverse Proxy endet. Dies ist eine wichtige Tatsache für die Softwareentwickler, da sie im Endeffekt bedeu-

tet dass es keine wirkliche Ende-zu-Ende Sicherheit zwischen einem Kundenbrowser und einem Applikationsserver der Firma gibt. Wir werden die Weitergabe des Kundenrequests und seine Absicherung innerhalb der Firewall-Infrastruktur weiter unten noch besprechen. Der Reverse Proxy hat heutzutage meist die folgenden Aufgaben:

- Abweisen nicht authentisierter Requests, die in das Intranet gerichtet sind;
- Feststellung der Identität und des Ortes von Requestern;
- Tokenbasierte Delegation von Identität;
- Session Handling;
- Regelung des Zugangs zum Intranet bzw. Internet bei Mitarbeitern und
- Logging und Filtering.

Lassen Sie uns diese Punkte nun im Einzelnen besprechen:

Kommt ein Request aus dem Internet beim Reverse Proxy an, so erfolgt die weitere Verarbeitung des Requests in Abhängigkeit von der Konfiguration des Proxies. Je nach Art des Requests (z. B. angeforderte URL), Protokoll, Ort etc. kann der Request unterschiedlich geroutet werden (beispielsweise auf einen Webserver, der statische HTML Seiten enthält und die ohne Authentisierung verfügbar sind).

Alternativ kann der Reverse Proxy prüfen, ob der Request zu einer bereits authentisierten Session gehört und ihn an einen Applikationsserver weiterleiten – falls der Request die entsprechenden Rechte besitzt. Der Proxy führt also eine Authentisierung sowie eine grobe Authorisierung durch – über eine Prüfung der URL. Wichtig ist dabei, dass der Reverse Proxy zur Authentisierung und Authorisierung weitere Subsysteme verwendet, die sich typischerweise weiter hinten in der Firewall-Infrastruktur befinden.

Die Art und Weise, wie ein Reverse Proxy eine durchgeführte Authentisierung an die nachgeordneten Applikation Server weitermeldet, ist je nach Infrastruktur und verwendeten Protokollen verschieden. Die Möglichkeiten reichen hier vom Einfügen eines Remote User Labels mit dem Namen des Users, der in den http-Header des Requests eingefügt wird, bis hin zur Erstellung eines signierten Tokens mit genauer Beschreibung wann, wo und wie dieser Request bzw. sein Urheber authentisiert wurde. Der Reverse Proxy verwaltet danach die Session mit dem Useragenten im Internet. Dabei kommen Cookies, URL-Rewriting oder auch die Auswertung der SSL-Session als Techniken in Betracht.

Hinter dem Punkt „Zugang zum Intranet bzw. Internet bei Mitarbeitern“ stecken mehrere wichtige Grundsatzfragen, die auch die Softwareentwicklung betreffen:

- Wird die Administration einer Webapplikation ausschließlich vom Intranet aus erlaubt, oder kann ein Administrator seine Aufgaben auch aus dem Internet erledigen? Viele Webapplikationen enthalten Interfaces für Wartung und Administration, die teil harmlos (Cache Control) aber auch hochgefährlich (Rechteverwaltung) sein können. Auch Middleware-Komponenten wie J2EE enthalten spezielle Services wie z. B. die Java Management Extension, die die generelle Administration der gesamten Plattform erlaubt. Diese Extension ist zwar durch ein Passwort schützbar, wird aber oft aus Unwissen mit dem Default-Passwort

am Internet angeboten. Wie also stellt man sicher, dass kritische Interfaces ausschließlich vom firmeninternen Intranet aufgerufen werden können?

- Welche Identität haben Mitarbeiter, wenn sie die eigenen Webapplikationen aus dem Internet heraus aufrufen?
- Können Web-User System-User werden (die sog. Impersonation, bei der z. B. aus einer Kundenidentität ein Domainadmin in Windows werden kann)?

Was kann innerhalb der Infrastruktur getan werden, um die gefährlichen Funktionen einer Webapplikation zusätzlich abzusichern? Sicherlich müssen diese zunächst einmal überhaupt erkannt werden. Schnell ist es passiert, dass kritische Testfunktionen aus Versehen in einer Applikation verbleiben oder dass die Gefährlichkeit einer Funktion nicht erkannt wird. Dagegen helfen lediglich Aufklärung der Entwickler und das gemeinsame Durchsprechen der Applikation durch Entwickler und Sicherheitsexperten. Selbst im Webauftritt von Grossbanken fanden sich lange Zeit öffentlich zugängliche Testservlets wie das bekannte „snoop“. Dieses Servlet gibt die Konfiguration der Servlet-Engine für Testzwecke an einen Browser zurück. Damit kann ein Angreifer wichtige Infrastrukturinformationen erhalten.

Sind die gefährlichen Funktionen erst einmal erkannt, ergeben sich mehrere Möglichkeiten, die alle mit dem Reverse Proxy-Ansatz zu tun haben. Eine populäre Möglichkeit besteht darin, beim Zugriff vom Intranet aus zu erzwingen, dass der Request durch einen speziellen Reverse Proxy geht. Dieser Reverse Proxy kann dann entweder bestimmte URLs erlauben, die beim öffentlich zugänglichen Proxy gesperrt sind, oder er ist genauso konfiguriert, wie der öffentlich zugängliche. In diesem Fall prüfen die Applikationsserver bzw. die Applikation, von welchem Proxy der Request kam und erlauben bzw. sperren den Request.

Je nach Lösung werden die kritischen Verwaltungsfunktionen entweder durch Konfiguration im Proxy oder per Software innerhalb der Applikation gesperrt. In ersten Fall bleibt die Applikation von Änderungen der Infrastruktur verschont, aber die Sicherheit ist absolut abhängig von der korrekten Konfiguration des Reverse Proxy. Im zweiten Fall muss die Applikation bei Änderungen der Infrastruktur geändert werden, behält sich aber die Prüfung der kritischen Requests selbst vor. Diesen Trade-off findet man sehr oft innerhalb der Software-Security: Je mehr Sicherheitsdetails nicht mehr innerhalb einer Applikation programmiert sind, desto weniger ist die Applikation anfällig für Änderungen der Infrastruktur. Gleichzeitig kann sich aber die Applikation nicht mehr gegen Fehler in der Infrastruktur schützen.

Noch kritischer wird die Sache, wenn wir die Administration auch aus dem Internet heraus erlauben wollen. Soll es uns in diesem Fall nicht so wie im in der Einführung erwähnten OBSOC-Beispiel ergehen (dort gab es keinen Unterschied in Bezug auf die Qualität der Authentisierung zwischen normalen Kunden und Administratoren, mit dem Erfolg, dass viele Administratoren kein oder nur ein Standard-Passwort verwendeten), dann müssen wir sehr genaue Regeln für diese Zugriffe aufstellen. Dazu gehören:

- Für Administrationszugriffe ist eine starke Authentisierung via PIN/TAN oder sogar Smartcard nötig.
- Eine Impersonation eines Systemadministrators darf nicht möglich sein (d.h. die Administrationstätigkeit darf zwar innerhalb der Applikation stattfinden, aber es kann sich dabei nicht um eine Systemadministration handeln. Damit soll verhindert werden, dass aus dem Urheber eines Internet-Request ein Domain Administrator werden kann).

Grundsätzlich ist von so einem Vorgehen dennoch abzuraten. Wenn extreme Sicherheit nötig ist, dann bleibt noch die Möglichkeit eine spezielle Zone innerhalb des Firewall-Komplexes zu bauen – eine so genannte *Administrationszone* – und durch Zugriffe über diese Zone die Administration von Applikationen zu erlauben. Dies ist vor allem ein Schutz vor Manipulationen aus dem eigenen Intranet.

Nun zum Punkt Logging und Filtering: Da jeder Request grundsätzlich durch den Reverse Proxy geroutet wird, ist dieser natürlich auch der ideale Ort um die Requests zu loggen bzw. auf gefährliche Inhalte hin zu prüfen. Gerade die Input-Validierung von Web-Requests ist besonders wichtig, da ansonsten eine Vielzahl von Attacken droht (z. B. SQL Injection, Cross-Site Scripting etc. Wir behandeln diese Angriffe im Detail im Band „Sichere Systeme“). Andererseits muss, wie bereits erwähnt, der Reverse Proxy ein hochsicheres System darstellen und darf deshalb keinen leichten Zugriff auf Ressourcen auf dieser Maschine bieten. Es ist also keine leichte Aufgabe, die Logging- und Filtering-Funktionalität mit den Anforderungen der Wartung in Einklang zu bringen.

11.1.3 Beispiel Nevis Web

Reverse Proxy-Installationen innerhalb einer DMZ können eine beliebige Komplexität annehmen. In einem späteren Kapitel besprechen wir eine einfache Absicherung eines Lizenzservers durch einen Apache/PHP-basierten Reverse Proxy. Hier wollen wir eine umfangreiche und kommerzielle Lösung vorstellen. Es handelt sich dabei um die Nevis Web Architektur der Firma AdNovum (s. [AdNo]). In Abb. 11.6 zunächst eine High-Level-Sicht auf die Gesamtarchitektur.

Der Reverse Proxy (hier nevisProxy genannt) ist also, wie gerade besprochen, innerhalb einer DMZ aufgestellt. Für eine genauere Analyse benötigen wir eine detaillierte Ansicht der Architektur (Abb. 11.7).

Das Diagramm zeigt die wichtigsten Komponenten der DMZ. Clientseitig muss nicht nur mit den typischen „thin clients“ (also Webbrowsersn) gerechnet werden, sondern auch mit Legacy-Applikationen, die über TCP/IP und SSL bzw. mit CORBA kommunizieren oder J2EE-Clients, die sicheres IIOP (IIOPS) verwenden. Diese Heterogenität macht die Aufgabe für den Reverse Proxy relativ schwierig, da über alle diese Protokolle hinweg eine gemeinsame und sichere Authentisierung realisiert werden muss. Hier nun die Aufgaben des Reverse Proxies in dieser Architektur im Einzelnen:

Abb. 11.6 Übersicht der Nevis Web Architektur

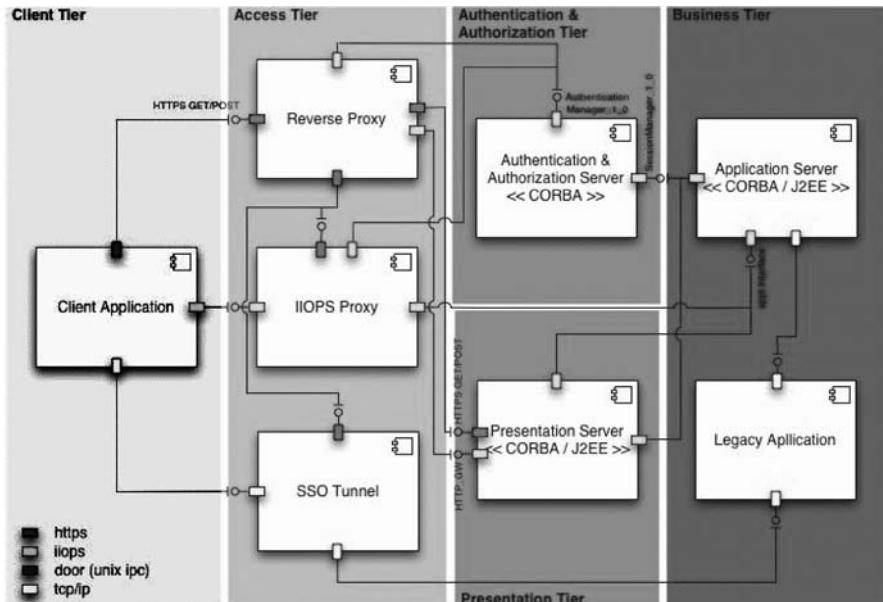
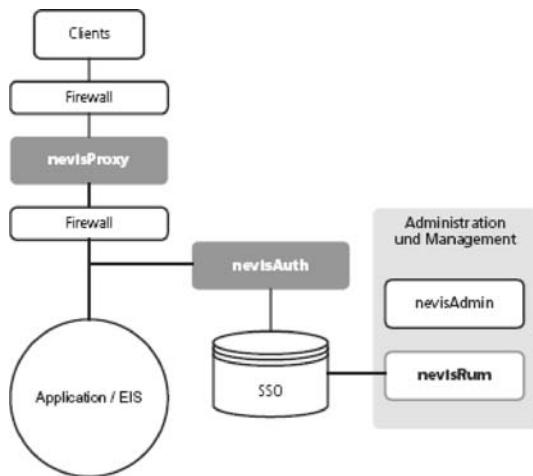


Abb. 11.7 Architekturdiagramm und Aufteilung in Schichten

- Identifikation des Clients: Der Proxy erzwingt die Authentisierung von Requests durch den Authentisierungsservice und ordnet nachfolgende Requests einer Session des Clients zu. Zur Identifikation der Client Session können Cookies oder die SSL Session ID verwendet werden.
- Ressourcenschutz: Je nach Konfiguration dürfen nur authentisierte und autorisierte Requests auf bestimmte Ressourcen zugreifen. Dieser Schutz ist ausbau-

bar zum Content-based Routing und zur vorgezogenen Input-Validierung für Applikationen.

- Step-up/Step-down initialization: Manche Firmen benötigen Authentisierungen mit unterschiedlicher Qualität in Abhängigkeit von den einem Request zugeordneten Ressourcen. Der Reverse Proxy sichert die richtige Quality of Service (QoS), bevor der Resourcenzugriff erlaubt wird und erzwingt notfalls eine weitere Authentisierung.
- Single Sign On (SSO): Der Reverse Proxy verwaltet signierte Token des Authentisierungsservers und leitet sie an weiter hinten gelegene Application Server weiter. Diese entnehmen die Identität des Clients dem Token, statt eine eigene Authentisierung vorzunehmen.
- Reverse Proxying: Weiterleitung von HTTP(S) Requests an Web Server. Die Header-informationen, Protokolle und Inhalte können manipuliert werden.
- Cookie Manager: Unterstützung für die Manipulation von Cookies der Application Server. Assoziation der Cookies mit SSL-SessionIDs, falls clientseitig keine Cookies erlaubt sind.
- Failsafeness: Failover Unterstützung im Backend
- Multi-Instance Capability: Zur Erleichterung von Entwicklung und Test können mehrere Instanzen eines Reverse Proxies auf einem Server laufen.
- Context Propagation: Um Ende-zu-Ende Sicherheit und sichere Delegation realisieren zu können, wird die Propagation von Informationen, die die Authentisierung oder Authorisierung betreffen, unterstützt. Die Integrität der Propagation muss verifizierbar sein.

Neben dem Reverse Proxy stellt der Authentisierungs- und Authorisierungsserver eine zentrale Komponente dar. Er wird vom Reverse Proxy konsultiert, sobald ein Request eintrifft. Seine wichtigste Aufgabe ist das Management von Sessions. Da sich die Firmen grundsätzlich in der Art und Weise, in der sie Authentisierungen durchführen, stark unterscheiden, muss der Authentisierungs- und Authorisierungsservice verschiedenste Authentisierungsmethoden verstehen und eine Vielzahl verschiedener Datenbanken und Backends nutzen können. Die Erweiterbarkeit des Dienstes ist eine Voraussetzung für die Integration in Firmenumfelder.

An dieser Stelle ist auch daran zu denken, dass auch weiter hinten liegende Applicationsserver den Service ansprechen werden, z. B. zur genaueren Authorisierung eines Clients, und der Dienst deshalb auch die aktuellen oder zukünftigen Java Standards zur Authorisierung anbieten muss. Die enge Verzahnung von Reverse Proxy, Authentisierungsservice und nachgelagerten Applikationen zeigt die Abb. 11.8.

Eine Präsentationskomponente erlaubt es, den Client mittels einer auf einem Web-Formular basierten Authentisierung ohne Mithilfe der Applikation zu identifizieren – dies vermeidet unauthentisierte Requests in inneren Zonen der Firewall.

Interessanterweise ist der sichere Reverse Proxy als Servlet implementiert, allerdings in C++ und auf Basis eines C++ Web Containers, der die Version 2.3 des Servlet APIs implementiert. Damit wird versucht, in großen Umgebungen den Anforderungen an die Performance zu genügen und dennoch einem Standard bezüglich Erweiterungen durch den Kunden und Konfiguration zu folgen.

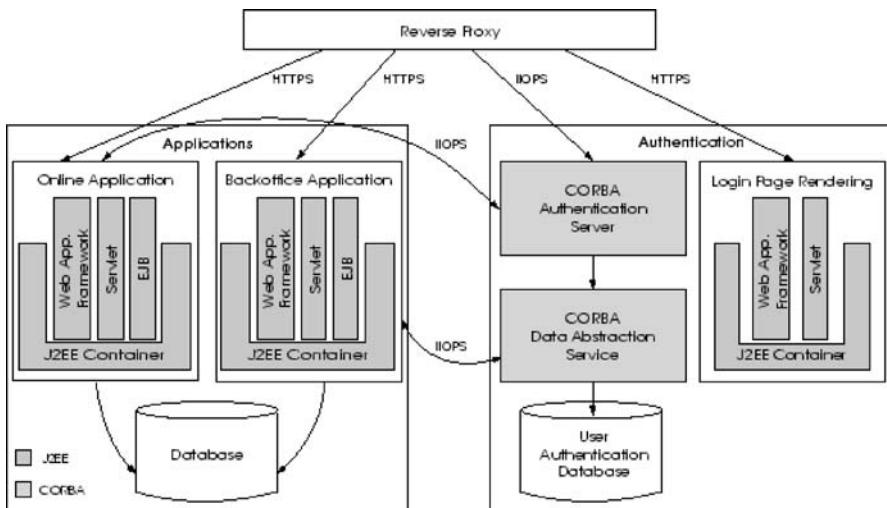


Abb. 11.8 Reverse Proxy und die von ihm verwendeten Kommunikationsprotokolle mit dem Backend

Der Reverse Proxy hat in dieser Architektur u. A. die Aufgabe, signierte Tokens vom Authentication Server an die Application Server weiterzureichen. Dies geschieht in Form von Werten, die in den http-Header eingefügt werden. Von entscheidender Bedeutung in einer solchen Infrastruktur sind somit die für den Inhalt und Absicherung der Token verwendeten Standards. Hier bietet der CORBA CSIV2 Standard mit der PAC-Struktur (*Privilege Attribute Certificate*, vgl. hierzu das Kapitel „Middleware Security“) als Authorisierungstoken eine gute Basis. Grundsätzlich bietet dieses Token die Möglichkeit der Propagation von sicherheitsrelevanter Information an nachgeordnete Instanzen. Das hat verschiedene Vorteile, wie etwa die Vermeidung von zusätzlichen Requests z. B. für Authorisierung im Application Server – die Rollen des Clients könnten im Token vorhanden sein. Das Problem liegt hier darin, dass die Nutzer dieser Informationen deren Syntax und Semantik genau kennen müssen. Der Ort des Zugangs, die Qualität der Authentisierung sowie Informationen über die Abteilung des Clients, falls es sich um einen Mitarbeiter handelt, sind Beispiele für nützliche Daten, die im weiteren Verlauf des Requests ausgenutzt werden können – oder besser gesagt müssen, wenn z. B. verschiedene Authentisierungsqualitäten angeboten werden. Dann muss verhindert werden, dass im weiteren Verlauf eines Requests eine falsche Zuordnung zu einem Service erfolgt, der zu seiner Nutzung eine bessere Authentisierung voraussetzt. Diese Problematik werden wir im Kapitel zu Ende zu Ende Sicherheit nochmals aufgreifen.

Je nach Aufbau der Netzwerk-Infrastruktur müssen die Transportkanäle, über die die Requests und Tokens transportiert werden, zusätzlich abgesichert werden. Wir werden diese Problematik im folgenden Abschnitt behandeln. Außerdem müssen die ausgestellten Token von den Empfängern validiert werden können.

Dazu können symmetrische Schlüssel verwendet werden. Allerdings haben diese Schlüssel den Nachteil, dass sie auf allen Application Servern gleichermaßen vorhanden sein müssen. Zudem erlauben sie nicht nur die *Prüfung* eines Tokens sondern auch die (evtl. unberechtigte) *Erstellung* eines Tokens. Besser, jedoch weniger performant, sind hier eindeutig asymmetrische Schlüssel, mit denen die Token digital signiert werden können.

11.1.4 Gegenseitige Authentisierung von Komponenten und Knoten

Die Protokolleigenschaften verteilter Systeme können entweder sehr niedrig innerhalb der Architektur angesiedelt werden, d.h. im Betriebssystem oder in der Middleware, oder aber die Applikationen können diese selber implementieren. Die Erfahrung hat gezeigt, dass das Implementieren von Funktionen auf einer sehr niedrigen Ebene diese Funktionen zwar allen Applikationen zur Verfügung stellt, allerdings auch alle Applikationen gleichermaßen mit ihnen belastet.

Das gleiche Prinzip scheint auch in Bezug auf Sicherheitsfunktionen zu gelten: Im Betriebssystem oder auf Netzwerkebene untergebracht sind sie allgemein verfügbar, aber von der Applikation weit entfernt. Sie lassen somit Raum für Angriffe in den oberen Schichten. Direkt innerhalb von Applikationen implementiert sind sie zwar teuer zu entwickeln, aber bieten Ende-zu-Ende-Sicherheit. Momentan geht der Trend ganz stark in Richtung der Extraktion von Security-Funktionen aus den Applikationen (z.B. in Application Server). Dies bringt teilweise ein Problem mangelnder Flexibilität mit sich – und gefährdet die Applikationen, wenn beispielsweise Fehler bei der Konfiguration der Middleware gemacht werden.

Typische Webapplikationen sind heute heterogene, mehrschichtige Applikationen, die aus mehreren, miteinander kommunizierenden Rechnern (Nodes) bestehen. Die Kommunikation wird dann deutlich sicherer, wenn sich die einzelnen Nodes über sichere Transportkanäle austauschen. Dies ist nötig, da sehr häufig noch mit kanalbasierter Authentisierung gearbeitet wird, d.h. am Beginn einer Kommunikation wird eine Authentisierung des Clients vorgenommen und eine Session aufgebaut. Solange die Session existiert, werden Requests des Clients akzeptiert (natürlich nach Prüfung der Zugriffsrechte). Wenn die zur Authentisierung nötigen Credentials unverschlüsselt transportiert werden, dann besteht die Gefahr einer Attacke, z.B. dass sich ein anderer Rechner in die Kommunikation einschleicht. Nehmen wir als Beispiel die Kommunikation zwischen Reverse Proxy, Authentisierungsserver und Applikationsserver. Zwischen diesen Rechnern werden Authentisierungstokens ausgetauscht. Wird so ein Token abgefangen, so besteht grundsätzlich die Möglichkeit, dass ein Angreifer dieses Token innerhalb eines eigenen Requests an den Application Server verwendet und den Request dadurch als scheinbar authentisiert darstellen kann.

Die Sicherheit der Infrastruktur kann dadurch gesteigert werden, dass eine gegenseitige Authentisierung vorgenommen wird, d. h. der Application Server erwartet nur Requests vom Reverse Proxy (und umgekehrt). Da ein einfaches Patchen eines IP Paketes genügt, um einen Absender zu fälschen, bedeutet gegenseitige *starke* Authentisierung in jedem Fall den Einsatz von symmetrischer oder asymmetrischer Keys.

Neben Verbindungen über IPSec bietet sich hierfür vor allem eine gegenseitige Authentisierung via SSL an. Während diese Absicherung vom Konzept her sehr einfach ist, ergeben sich dennoch in vielen Fällen praktische Schwierigkeiten, die immer wieder den Einsatz von gegenseitiger Authentisierung durch SSL verhindern. Zunächst jedoch erst einmal zum Konzept.

Zuerst muss die Frage geklärt werden ob sich der Rechner nach außen auf dem Internet authentisieren soll, d. h. sollen Useragenten im Internet den Rechner authentisieren können oder soll die Authentisierung lediglich im eigenen Netzwerk stattfinden. Im ersten Fall brauchen wir ein von einer öffentlichen CA erstelltes Zertifikat mit dem DNS Namen des Rechners. Nur wenn das Root-Zertifikat der CA bereits auf dem Useragenten installiert ist, kann das Zertifikat unseres Rechners vom Useragenten problemlos verifiziert werden. Andernfalls müsste der User explizit erklären, dass er unserem Zertifikat (signiert von einer unbekannten CA oder von uns selbst) dennoch vertraut, oder aber sogar ein von uns selbst hergestelltes Root-Zertifikat bei sich installieren.

Im Fall der Infrastruktur gegen wir jedoch davon aus, dass nur eigene Knoten sich gegenseitig authentisieren wollen. In diesem Fall genügen selbst hergestellte Schlüssel bzw. Zertifikate, wie wir im Folgenden zeigen wollen. Typische Produkte wie der IBM WebSphere Application Server verwenden so genannte *Key Files*, um die zu verwendenden Schlüssel (sowohl öffentliche als auch private Schlüssel) zu managen (s. auch [Botz06]). Ein Key File besteht aus zwei Teilen, dem *Trust Store* und dem *Key Store* (s. Abb. 11.9). Der Trust Store enthält die Zertifikate der signierenden Stellen (d. h. CAs), denen der Server vertraut. Der Keystore hingegen enthält die eigenen Zertifikate und auch die dazu gehörigen privaten Schlüssel – der Zugang zum Key Store muss deshalb besonders geschützt sein.

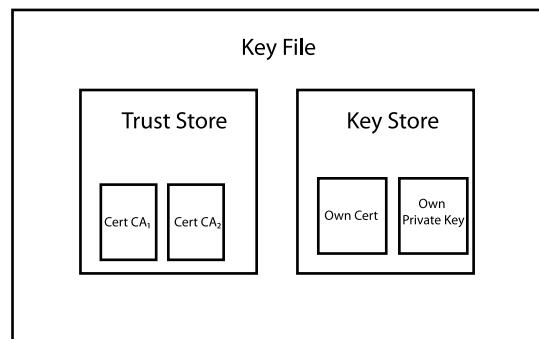


Abb. 11.9 Aufbau eines Key Files

Beim Aufbau einer geschützten Verbindung via SSL prüft der Server, ob das präsentierte Client-Zertifikat von einer vertrauenswürdigen Stelle stammt, die im Trust Store vertreten ist. Ist dies der Fall, und verläuft der SSL-Handshake auch ansonsten problemlos, hat sich der Client authentifiziert als derjenige, dessen Identität im Zertifikat genannt wird. Eine darauf folgende Authorisierung, basierend auf dieser Identität, liegt aber außerhalb des SSL-Protokollrahmens und muss von einer anderen Instanz durchgeführt werden. Leider besitzen nicht alle Systeme diese Fähigkeit, was bedeutet, dass allen Clients mit einem Zertifikat von einer CA aus dem Trust Store das gleiche volle Vertrauen entgegen gebracht wird. In diesen Fällen sollte die Menge der Endpunkte, die überhaupt einen SSL-Handshake mit dem Server durchführen und damit eine Verbindung zum Server aufbauen können, stark beschränkt werden. Dies gelingt, indem nur sehr wenige selbst-signierte Zertifikate überhaupt in den Trust Store aufgenommen werden. Dadurch können nur die Inhaber dieser selbst-signierten Zertifikate überhaupt den SSL-Handshake mit dem Server beenden.

Entsprechendes gilt, wenn eine gegenseitige Authentifikation zweier Knoten via SSL gewünscht ist und niemand sonst eine Verbindung zu den Knoten aufbauen können soll: Beide Partner erzeugen dann für sich ein selbst-signiertes Zertifikat, welches (als einziges) in den Trust Store des jeweils anderen importiert wird (s. Abb. 11.10).

In der Firmenpraxis entzündet sich dennoch oft ein Streit an der Frage, ob eine gegenseitige Absicherung nötig ist. Während sie technisch kein Problem darstellt und auch von der Bedienung her einfach ist, stellt sie dennoch manche Firmen vor administrative Probleme. Es stellt sich nämlich die Frage, wer für die Konfiguration der SSL Verbindung einschließlich der Generierung der Zertifikate zuständig sein soll: Die Applikationsentwickler, die Leute von der Netzwerksicherheit oder das Bedienpersonal?

Sinnvollerweise sollte aber die Konfiguration nur vom Personal der Infrastruktur vorgenommen werden. Der Grund liegt unter anderem darin, dass Zertifikate nach einiger Zeit ungültig werden. Dies äußert sich typischerweise so, dass eines Tages ein wichtiger Service nicht mehr funktioniert und nach stundenlangem Suchen endlich jemand entdeckt, dass ein wichtiges Zertifikat abgelaufen ist und

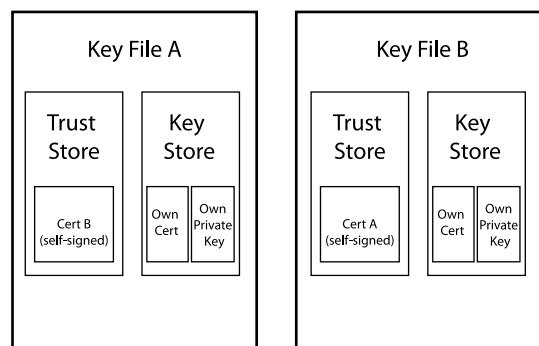


Abb. 11.10 Aufbau der Key Files zweier Knoten bei gegenseitiger Authentifikation via SSL

ein Knoten deshalb Requests ablehnt. Die notwendige Verwaltung aller ausgestellten Zertifikate lässt sich am besten zentral regeln. Zudem sind Kenntnisse über die Generierung, Zertifikation und die Konfiguration der SSL Verbindung meist nicht sehr verbreitet bei Applikationsentwicklern.

11.1.5 Applikationsdesign für zentrale Firewall-Umgebungen

Die folgenden Punkte sind für das Applikationsdesign in zentralen Firewall-Umgebungen neben der vorgezogenen frühen Authentisierung (die wir im nächsten Kapitel in Bezug auf ihre softwaretechnischen Auswirkungen noch genauer untersuchen) von großer Bedeutung:

- korrekte Angaben nötiger offener Ports und Protokolle;
- die kritische Sicht auf Verbindungen vom Intranet in die DMZ und
- Functional User in der DMZ

Beginnen wir die Diskussion dieser Punkte mit den ersten beiden, da sie meist in einem engen Zusammenhang stehen. Wir haben bereits gesagt, dass eine ernsthaft geführte DMZ Regeln für die verwendeten Ports und Protokolle definiert. Die Applikation muss ihrerseits genau ihre Anforderungen definieren in Bezug auf zulässige, aber notwendige Ports und Protokolle, die sie verwendet. Diese Informationen werden später Input für die Konfiguration der Firewall-Software bzw. der Router und Paketfilter der DMZ.

Häufig neigen Entwickler zur Ansicht, dass sich im Internet die Bösen und im Intranet die Guten befinden. Es dauert längere Zeit, bis verstanden wird, dass der Aufbau von Verbindungen aus dem Intranet ins Internet die gleiche Gefahr birgt, wie der umgekehrte Weg. Wie sonst werden wichtige Firmendaten etc. nach draussen geschafft? Wie sonst können Trojaner, Bots und Viren ihre Kontakte nach aussen knüpfen? Das bedeutet, dass jede Verbindung aus dem Intranet in die Firewall-Zone genauso genehmigt werden muss wie umgekehrt eine Verbindung ins Intranet.

Oft wird verlangt, dass zum Zwecke der Wartung oder Administration eine Verbindung über spezielle Rechner innerhalb der Firewall aufgebaut wird. Manchmal muss zu diesem Zweck spezielle Applikationssoftware dort installiert werden. Ein permanenter wunder Punkt zwischen Applikationsentwicklern und Infrastrukturverantwortlichen ist nun der File Transfer in die Firewall. Was von Applikationsentwicklern oft nicht gesehen wird, ist dass solche Filetransfers, wenn sie mit Standardwerkzeugen (z. B. ftp) durchgeführt werden:

- einen User Account innerhalb der DMZ verlangen, und zwar auf hochsicheren Maschinen. Dieser User Account ist noch dazu für interaktiven Zugriff ausgelegt und damit besonders gefährlich. Dieser User Account ist oft noch ein sog. Functional User Account. Diesen Typ von Account haben wir bereits in Kap. 10 kennen gelernt: Er ist keiner Person, sondern einer Aufgabe zugewiesen und lässt

keine Rückschlüsse auf eine wirkliche Person zu. „Root“ oder „Administrator“ sind Beispiele für solche Accounts. Neben dem fehlenden Rückschluss auf Personen (und dadurch fehlende Verantwortung) zeichnen sie sich oft auch durch viel zu viele Rechte aus, die zur Erfüllung ihrer Aufgaben nicht benötigt werden.

- die gängigen Übertragungstools keinerlei Nachvollziehbarkeit aufweisen, d. h. im Nachhinein kann niemand nachweisen, was genau hochgeladen wurde.

Diese Einwände gegen Filetransfers in die DMZ sind eigentlich recht gut nachvollziehbar. Wieso erscheinen sie Applikationsentwicklern dennoch oft als Schikane?

Wie so oft in der IT liegt im Kern eines Problems zwischen Benutzern und Verwaltern ein Usability-Problem: Eine sichere Lösung für das Filetransfer Problem muss auch für Entwickler einfach zu bestellen, bedienen und warten sein. Andernfalls wird nach Workarounds und Umgehungen gesucht. Es nutzt wenig, ein hochsicheres Werkzeug mit schlechter Bedienbarkeit anzubieten, bei dem schon die Beantragung einem Spiessrutenlauf gleichkommt. Hier zeigt sich sogar schon in der Infrastruktur von Firewalls die enge Verzahnung von Usability und Security, auf die wir weiter unten noch speziell eingehen werden.

Eine weitere Forderung aus der Infrastruktur an die Applikationsentwicklung ist oft ein Wechsel des Kommunikationsprotokolls zwischen verschiedenen Zonen. Beispielsweise landet ein http-Request über den Reverse Proxy beim Application Server in der inneren Zone der Firewall. Die Applikation kommuniziert dann mit einem Datenbankserver innerhalb des Intranets mittels einer sicheren SQL Verbindung.

Ein sehr kritischer Punkt ist die Notwendigkeit von User Accounts für die Applikation auf hochsicheren Rechnern der Firewall. Im schlimmsten Fall ist auch noch ein interaktiver Zugang nötig (etwa für SSH, Telnet). Dabei ist weniger das Protokoll ein Problem, sondern die Tatsache, dass Änderungen durch das Personal damit schlecht nachvollziehbar sind und dass eine Attacke auf so ein System im Erfolgsfall eine interaktive Shell erlangen kann. Deshalb sollte das Applikationsdesign möglichst ohne funktionale Systemuser auskommen. Die Frage unter welchen Accounts die Application Server selbst laufen sollten, wie Datenbanken angesprochen werden etc., werden wir im Kapitel zur Application Server Security behandeln.

11.1.6 Sessionkonzept

Die Entwicklung eines Sessionkonzepts, das gleichzeitig performant, mit der Infrastruktur verträglich (Security und Loadbalancing) ist, und außerdem den gewünschten Quality of Service bringt, stellt für Applikationsentwickler generell eine schwierige Aufgabe dar. Besonders schwierig wird es, wenn die Techniken, die für das Sessionmanagement nötig sind, auch noch sicherheitskritisch untersucht werden müssen (sind Cookies erlaubt? Welche Konsequenzen hat URL Rewriting? Wie kann ich eine SSL SessionID nutzen?) Fragen nach sinnvollen

Session Timeouts an die Applikationsentwickler vervollständigen die Problematik. Wann benutzt eine Applikation eigentlich Sessions? Rein technisch gesehen immer dann, wenn sich eine Applikation Daten für einen Requestor merkt. Das Speichern kann dabei sowohl temporär im RAM als auch persistent in einer Datenbank stattfinden. Wichtig dabei ist, dass der Requestor bei jedem Request etwas mit-schickt, das es der empfangenden Applikation erlaubt, den momentanen Request mit den früher gespeicherten Daten zu verbinden. Dieses „etwas“ kann dabei sein:

- eine TCP Sequenznummer, die mit jedem Request um eins erhöht wird;
- ein beliebiges Datum, das mit jedem Request mitgeschickt wird (Cookie oder spezielle URL);
- eine SSL SessionID und
- eine UserID/Passwort Kombination

Muss bei einer Session die Identität eines Requestors dem Empfänger bekannt sein? Keineswegs, wie das Beispiel der TCP/IP-Sessions zeigt. Häufig findet man allerdings ein doppeltes Session-Konstrukt: Zunächst identifiziert sich ein Requestor mit Hilfe seiner Credentials bei einem Server (einmalig), dann wird eine Verbindung aufgebaut, die auf niedrigerem Level eine Verbindung der Nachrichten innerhalb einer Session zulässt. In diesem Sinne kann man die Anmeldung eines Kunden bei einem Service Provider als High-Level-Session bezeichnen, die besonders langlebig ist. Der Provider erstellt dafür die so genannten Stammdaten über den Kunden. Schickt der Kunde dann tatsächlich einen Request, so muss er sich authentifizieren und es wird auf Basis von TCP/IP bzw. Cookies eine temporäre Verbindung erstellt.

Für die Empfängerapplikation ist dabei häufig entscheidend, wo die Daten, die im Rahmen einer Session angelegt sind, gespeichert werden. Dieser Session-State ist in der Architektur verteilter Systeme oft ein Problem. Es geht dabei schlicht um die Frage, wie lange die Applikation den State des Kunden halten muss. Das Grundproblem wird wieder am Besten bei der Registrierung eines Kunden sichtbar: Nehmen wir an, die Registrierung für einen E-Mail-Account ist kostenlos und erfordert keine starke Authentisierung. Dann sieht sich ein Provider schnell mit einer Vielzahl von vergebenen UserIDs konfrontiert, die anscheinend auch nach Monaten nicht benutzt werden. Kann der Provider diese Accounts löschen? Ab wann?

Auch auf niedrigeren Protokoll-Ebenen kann man das gleiche Problem finden: So sind die berüchtigten TCP-SYN-Attacken nichts anderes als nicht zu Ende geführte Verbindungsaufbauten: Bei jedem Request zum Verbindungsaufbau merkt sich der Empfänger etwas über den Sender und erwartet dann weitere Angaben vom Sender. Nur kommen im Fall der Attacke diese Angaben nicht. Statt dessen wird ein weiterer Verbindungsaufbau-Request gesendet, den sich der Server wieder merkt. Am Ende hat der Server seine Speicherkapazität erschöpft und kann keinerlei Requests mehr annehmen, auch keine von legalen Kunden mehr.

Dieses Beispiel zeigt, dass das Halten von States für nicht-authentisierte Clients eine grosse Gefahr birgt. Eine Applikation muss so gebaut sein, dass im Falle fehlender Authentisierung am Besten kein State angelegt wird. Problematisch ist

dies gerade im Falle eines Logins, der sich über mehrere Request/Response Paare hinweg erstreckt. Hier ist es am Besten, wenn der Login aus der Applikation ganz herausgenommen wird. Hierfür wird für jeden Login zunächst eine anonyme Session eröffnet, die dann nach erfolgreicher Anmeldung durch eine authentisierte Session ersetzt wird (z. B. durch ein neues Cookie).

Damit haben wir den ersten Zusammenhang zwischen dem Halten eines State und Sicherheit einer Session geklärt. Als nächstes muss eine Applikation definieren, mit welchem Mechanismus der State gehalten werden soll, und zwar sowohl, was die Zuordnung eines Requests zum State auf der Serverseite betrifft, als auch den im Server gehaltenen State selbst. Der Zuordnungsmechanismus ist in den meisten Fällen Cookie-basiert: In einem Cookie auf Seiten des Kunden wird ein Datum gehalten, das dem Server die Zuordnung des Clients erlaubt.

Die Software-Industrie ist mit diesen Cookies durch eine ganze Reihe fataler Fehler gegangen, vom Abspeichern von UserID/Passwort Kombinationen bis hin zu SessionIDs, die jedoch am Anfang meist so schlecht erzeugt waren, dass man sie voraussagen konnte. Kann jemand aber eine SessionID voraussagen oder erraten, dann kann er natürlich ohne Probleme in eine bestehende Session einsteigen bzw. unter einer falschen Identität eine Session aufbauen. Anzumerken ist noch, dass selbst Hersteller von Betriebssystemen und Entwickler von Applikationsservern zu Beginn häufig so schlechte Generatoren von Pseudozufallszahlen (so genannte Pseudo Random Number Generators, PRNGs) verwendet haben, dass falsche Requests in bestehende TCP-Sessions eingeschleust werden konnten (die Nicht-Erratbarkeit der so genannten Sequence Numbers ist ein wichtiges Sicherheitselement in TCP, da die Sequence Number eines Antwort-Pakets gleich der Nummer des beantworteten Pakets + 1 sein muss). Auch das Auslesen von privaten Mailboxen etc. war kein Problem dank erratbarer Cookies. Heute ist die sichere Erzeugung von SessionIDs, z. B. mit Hilfe kryptografischer Hashfunktionen, kein Problem mehr. Dafür zeigt sich eine andere Problematik: Wie sicher ist ein solches Cookie auf Seiten des Users aufgehoben? Trojanische Pferde können Angreifern das Auslesen dieser Cookies ermöglichen. Aber auch Cross-Site Scripting Attacken gefährden die SessionIDs in den Cookies. Aus diesem Grund sollte eine Applikation heute vermeiden, jegliche Art von Session-Information in permanenten Cookies unterzubringen. Ganz besonders sensible Applikationen sollten außerdem die temporären Cookies mit einer kurzen Lebenszeit versehen. Aber reicht das aus, wenn z. B. der User in einem Internet Cafe sitzt?

Am wirksamsten ist deshalb die Methode, ganz auf eine Speicherung von Informationen auf Seiten des Kunden zu verzichten und stattdessen die Applikationssession an die ID einer aufgebauten SSL-Session zu binden, wie oben im Nevis-Web Beispiel gesehen. Dazu muss der Reverse Proxy in der Lage sein, Session Cookies des Applikationsservers abzufangen und in einer Tabelle mit SSL-SessionIDs zu korrelieren. Aber auch der State, den die Applikation intern für einen Client hält, hat einen Bezug zur Sicherheitsarchitektur der Infrastruktur. So hängt die Sicherheit einer Applikation immer auch mit ihrer Verfügbarkeit und Performance zusammen. Beide Anforderungen – hohe Verfügbarkeit und Perfor-

mance – führen heute oft zu einem Applikationsdesign, das die Applikation in mehrere Instanzen aufteilt, die auf verschiedenen Rechnern laufen.

Im einfacheren Fall – der Sicherung der Performance – wird der mit einem Client verbundene State meist im RAM der Applikation gehalten. Bei einem Absturz muss der Client eine neue Session mit einem anderen Rechner eröffnen. In den meisten Fällen bedeutet das eine neue Anmeldung. Die Applikation verwendet in diesem Fall so genannte *Sticky Sessions*, d.h. sie erwartet, dass alle Requests, die zu einer Session gehören, auf derselben Instanz der Applikation landen (weil eben dort im RAM der Session State untergebracht ist).

Die Verteilung der Requests auf verschiedene Maschinen ist Sache eines vorgeschalteten Loadbalancers, der natürlich die jeweilige Zugehörigkeit eines Requests zu einer existierenden Session kennen muss. Dies gilt insbesondere, wenn ein Client im Rahmen einer SSL-Session mit einem bestimmten Server eine SessionID erhalten hat und die Session mittels des Session-Resumption Mechanismus von SSL später erneut wieder aufnehmen möchte (vgl. unsere Diskussion von SSL in Kapitel „Basisprotokolle“). Die Applikationsentwickler müssen dies klar als Anforderung an die Infrastruktur stellen. Requests für Services, die keinen State unterhalten, können natürlich auf alle verfügbaren Maschinen verteilt werden.

Wesentlich kritischer ist die Frage nach dem Ort, wo der State gehalten wird – nämlich in dem Moment, wo wir die Verfügbarkeit der gesamten Applikation sichern müssen und zwar so, dass bestehende Client-Sessions nicht unterbrochen werden, wenn ein Application Server ausfällt. Das bedeutet letztlich nichts anderes, als dass der Application State auf mehreren Servern repliziert werden muss. Alternativ könnte der State in einer von allen Maschinen erreichbaren Datenbank vorgehalten werden. Letzteres ist jedoch nicht unbedingt performant und schränkt die Grösse des gehaltenen States doch sehr ein (ca. 4–5 KB pro Session). Sehr elegant ist es dagegen, den State über das Netzwerk auf weitere Maschinen zu verteilen oder einen so genannten *Distributed Cache* zu verwenden. Auch hier muss der Loadbalancer jedoch die Maschinenpaare eines Clusters kennen, die einen gemeinsamen State halten.

Einen Sonderfall eines State müssen wir noch ansprechen, nämlich den der verifizierten Identität des Clients als State. Wir werden dies im Zusammenhang mit tokenbasierter Identität und Delegation bei Applikationsservern noch genauer behandeln, deshalb sei hier nur kurz die grundlegende Problematik angesprochen: Egal, ob ein Application Server die Identität eines Clients mittels eines Tokens (von einem Authentisierungsserver und Reverse Proxy) erhält, oder ob er mittels JAAS eine eigene Authentisierung vornimmt: Das Ergebnis ist ein verifizierter Client mit einem Profil, das seine Rechte – meist in Form von Rollen – enthält (so genannter *Autorisierungsvektor*). Ein Teil des Profils kann direkt dem Token entnommen sein, ein anderer Teil stammt meist aus einem LDAP Lookup. Das Ergebnis dieser Prüfung ist ein authentisierter Client, der in einer Datenstruktur im Application Server festgehalten wird. Daraufhin erzeugt der Application Server ein Cookie und speichert es beim Client (bzw. beim Reverse Proxy, wenn keine Client-Cookies erlaubt sind). Kommt nun ein weiterer Request des Clients, dann erkennt der Application Server am Cookie, um welche verifizierte Identität es sich

handelt und kann zur Verarbeitung des Requests das korrekte Userprofil heraussuchen. Wenn wir diesen Fall nun auf eine Clusterumgebung ausweiten, dann kann es sein, dass im Falle eines Absturzes eines Servers ein anderer Server den nächsten Request erhält – und damit auch das Cookie. Dieser Server kann jetzt zwar am Cookie erkennen, dass ein verifizierter User vorgelegen hat (er kann auch das Cookie prüfen, was aber gleiche symmetrische Schlüssel oder die Verwendung digitaler Signaturen bei den Application Servern voraussetzt – dies erzeugt neuen Overhead und kann zu weiteren Sicherheitsproblemen führen). Dem Cookie kann er die Identität des Users entnehmen (für eine nicht-Clusterumgebung hätte eine Zufallszahl im Cookie genügt, damit der Application Server das Profil zuordnen kann) und mit Hilfe dieser Identität versuchen, wieder Profildaten zu erhalten. Je nach Clusterkonfiguration können diese Profildaten (im einfachsten Fall) in serialisierter Form in einem gemeinsamen Cache vorliegen, oder aber sie müssen neu erzeugt werden, indem mit Hilfe der bekannten Identität des Clients ein neuer LDAP-Lookup durchgeführt wird, um die Autorisierungsdaten zu erhalten.

Das größte Problem stellt sich jedoch in dem Moment, wo Teile des Userprofiles aus dem Token des Authentisierungsservers entnommen wurden. Dieses Token ist nicht verfügbar auf dem Ersatzserver. Entweder muss dieser jetzt den Request zurückweisen und den Kunden zu einer neuen Authentisierung zwingen, oder der Server wendet sich mit einem direkten Request an den inzwischen hoffentlich wieder lauffähigen Originalserver mit der Bitte um einen serialisierte Form des originalen Userprofiles. Je nach Konfiguration muss ein solcher Austausch hochkritischer Userdaten auch durch Signaturen oder Schlüssel der beteiligten Server abgesichert werden.

Ein weiteres Problem im Zusammenhang mit Sessions, das die Applikationsentwicklung betrifft, ist die Frage des Session-Timeouts. In der Fallstudie zum Bahnportal haben wir verschiedene Szenarien für den Ort des Clients durchgesprochen: Zuhause, im Internet Cafe, am Kiosk etc. In jedem Fall wird der Ort des Kunden einen Einfluss auf die maximale Sessiondauer ohne Interaktion haben. Natürlich kommt es zusätzlich stark darauf an, wie kritisch die Applikation selbst ist. Am Internet geht man häufig davon aus, dass 30 Minuten eine für beide Seiten erträgliche maximale Sessiondauer darstellt. E-banking Applikationen haben allerdings häufig wesentlich kürzere Session Timeouts. Auf der anderen Seite halten manche Verwaltungsapplikation eine Session über mehrere Stunden.

Für die Applikationsentwicklung ist hier von Bedeutung, dass die Laufzeitumgebung (Web Container) unterschiedliche Session-Timeouts verwalten können muss. Das Gleiche muss für vorgelagerte Reverse Proxies gelten, wenn sie eigene Sessions zum Client unterhalten. Im Single-Sign-On-Umfeld müssen zudem die Timeouts der einzelnen Applikationen abgestimmt werden. Hier ist günstig, wenn ein globales Session Management existiert, das über Events den Applikationen mitteilen kann, wenn eine Session vom Kunden beendet wurde oder dies per Timeout geschehen ist.

Schließlich wollen wir noch auf eine Besonderheit gekoppelter Sessions hinweisen: Wenn die Timeouts von Reverse Proxy und Applikation nicht überein-

stimmen, dann können seltsame Effekte entstehen. Selbst wenn die Timeouts synchronisiert sind, kann folgender Fall eintreten:

- Ein Kunde meldet sich bei einer E-Business Applikation an.
- Der Reverse Proxy prüft die Credentials und erzeugt eine authentisierte SSL-Session mit dem User-Agenten des Kunden und leitet den Request an den Application Server weiter.
- Der Applikation Server erzeugt nun seinerseits eine Session mit dem dazugehörigen Cookie, das für die Identität des Kunden steht.
- Nun vergeht eine Stunde ohne eine Aktion des Kunden. Beide Session-Timeouts sind damit abgelaufen. Jetzt klickt der Kunde auf einen Logout-Link.
- Der Reverse Proxy unterbricht den Request, stellt fest, dass die Session abgelaufen ist und schickt eine Anmeldeform an den User-Agenten: „Sie müssen sich neu anmelden, um sich abmelden zu können.“

Ein solches Szenario läuft zwar aus technischer Sicht völlig korrekt ab, entspricht aber kaum den Erwartungen eines Users. Ähnliche Fälle werden wir weiter unten bei der Diskussion föderativen Identitätsmanagements kennen lernen, bei dem ebenfalls unterschiedliche Sessions gekoppelt werden.

Fassen wir nun die für die Applikationsentwicklung wichtigen Fragen bezüglich des Sessionmanagements nochmals kurz zusammen:

- Ist der Mechanismus zur Sessionverwaltung (SessionIDs, SSL-Sessions etc.) sicher und erprobt?
- Wird der State innerhalb der Applikation gehalten? Wenn ja: In authentisierter Form oder nicht?
- Erwartet die Applikation, dass „Sticky Sessions“ möglich sind, d. h. dass alle Requests eines Clients am gleichen Server ankommen?
- Ist der Mechanismus des Applikationsservers für „Sticky Sessions“ kompatibel mit der Loadbalancing-Infrastruktur?
- Erwartet die Applikation einen Session Fail-over auf weitere Maschinen im Cluster? Sind diese Maschinen fix und definiert?
- Ist der Loadbalancing-Mechanismus in der Lage, die Maschinenpaare, die zusammen die States verwalten, innerhalb eines Clusters zu erkennen?
- Kennt die Applikation die Größe ihres Sessionstates pro Kunde und wird die Größe bezüglich der Performanceauswirkungen überwacht?
- Ist der maximal mögliche Session-Timeout einer Applikation aus Business- und IT-Security Sicht abgestimmt und erlauben die verwendeten Techniken zur Absicherung der Transportkanäle diese Länge?
- Muss die Applikation auf das Ende einer Session reagieren können und welche Interfaces stehen für die Benachrichtigung zur Verfügung?

Die Frage einer externen, vorgelagerten Authentisierung wird im nächsten Abschnitt behandelt. Sie hat ebenfalls großen Einfluss auf die Architektur der Applikation.

11.2 Verkleinerung der Angriffsfläche

In diesem Abschnitt wollen wir zwei allgemeine Sicherheitsprinzipien bzw. „Security Patterns“ im Einsatz zeigen, nämlich:

- Die Verkleinerung der Oberfläche der Systeme, die für einen Angriff zu Verfügung steht (Attack Surface) durch geeignetes Filtern.
- Das Prinzip „Defense in Depth“, das automatisch zu einem Zonenkonzept führt und zu geplanten Protokollbrüchen.

Das geht von Paketfiltern, die Systeme und Ports (also Protokolle) filtern, über das Zwischenschalten von Protokollumsetzern und intelligenten Proxies zur Netzwerk-trennung bis hin zur Authentisierung, die dann die Oberfläche in der Software selbst (zum Beispiel die angezeigten Seiten oder Services eines Web Servers) verkleinert.

11.2.1 Maßnahmen

Das folgende Diagramm zeigt die wichtigsten Maßnahmen, um die Angriffsüberfläche einer DMZ zu verkleinern und gleichzeitig „Defense in Depth“ zu erreichen. Von links nach rechts nähert man sich dabei immer weiter dem Internet, d. h. die erste Maßnahme greift direkt auf den Hosts in der DMZ, während die letzte Maßnahme, der zweite Paketfilter (L) tatsächlich im Internet sicht- und angreifbar ist.

„Defense in Depth“ erreichen wir durch eine Aufspaltung der Infrastruktur in fixe Zonen mit überwachten Übergängen. Aber auch der Zwang zum Wechsel des Protokolls erschwert die Attacken, wenn ein Protokoll kompromittiert wurde. Hier eine kurze Beschreibung der eingesetzten Maßnahmen:

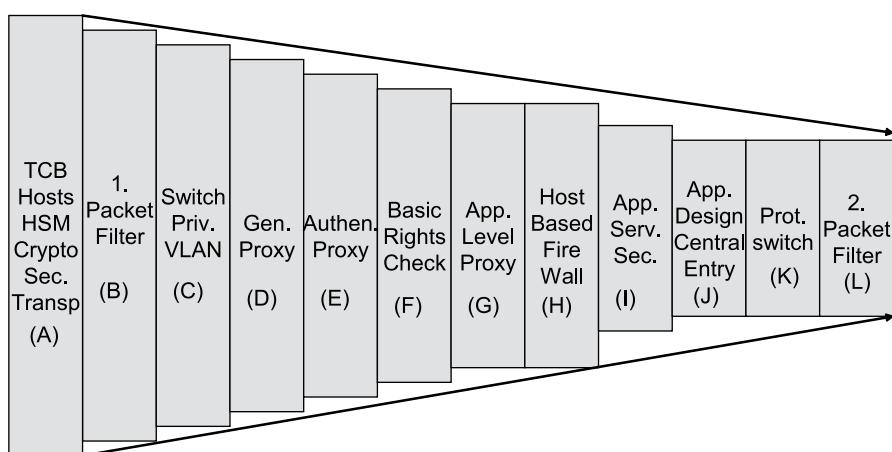


Abb. 11.11 Verkleinerung der Angriffsüberfläche in der DMZ

Kat.	Mechanismus	Wirkung
A	Trusted Computing Base, Hardware Security Modules. Secure Transport	Hochsichere Rechner mit speziell konfigurierter Software und Administration verhindern gefährliche interactive Logins und Angriffe über das Netzwerk auf Software. Credentials wie Passwörter, Schlüssel etc. werden in speziellen Hardware Modulen pro Rechner gehalten. Alle Verbindungen zwischen Maschinen sind über SSL abgesichert und können somit nicht mitgelesen oder modifiziert werden.
B	Packet Filer	An den Rändern der Zonen sorgen spezielle Paketfilter für die Filterung von Requests aus dem Netzwerk. Sie sperren Ports und Protokolle.
C	Switches, Private VLAN	Konfigurierbare Switches erlauben es, die Verbindungen der Rechner untereinander zu kontrollieren und getrennte, private, virtuelle Sub-Netze zu erzeugen. Dadurch lassen sich bequem Sicherheitszonen erzeugen und jeder Zugriff kann über einen Firewall/Filter gezwungen werden. Es lassen sich Zonen unterschiedlicher Sicherheit verwalten.
D	Generic Proxy	Ein generischer Proxy trennt bereits die verschiedenen Netzwerke voneinander und kann Redirektionen vornehmen. Dadurch lässt sich der Zugriff auf bestimmte Ports von Applikationen beschränken, ohne dass der Proxy die Applikationen oder deren Protokolle wirklich kennen muss.
E	Authenticating Proxy	Ein authentisierender Proxy erlaubt nur authentisierte Requests an weitere Maschinen. Öffentliche Zugriffe scheitern an ihm. Damit verringert er den potentiell möglichen Zugriff auf Backend-Maschinen stark. Außerdem erlaubt er auch die Kontrolle ausgehender Requests wie im Fall von http-Proxies.
F	Basic Rights Check	Die Prüfung, ob eine Identität grundsätzlich das Recht zu einer Aktion besitzt, z. B. den Aufruf einer URL eines Web Servers. Vorgelagert verhindert diese Prüfung, dass illegale Requests überhaupt an die Applikation gelangen. Zentrale Rechteverwaltung steigert die Sicherheit zusätzlich, indem sie eine konsistente Regelung pro Firma erzeugt.
G	Application Level Proxy	Er kennt das Protokoll der Applikation und kann selbst in Datensegmenten der Aufrufe und Antworten Prüfungen und Änderungen vornehmen. SSL Verbindungen von Seiten des Kunden enden bei ihm. Deshalb hat er Zugriff auf die Kommunikationsdaten und beliebige Kontrollmaßnahmen sind möglich. Beispiel dafür ist die Prüfung von Schlüsseln wie im Fall von Mod-Security für Web-Service Applikationen.
H	Host-based Firewall, Intrusion Detection	Lokale Paketfilter auf einem Host können einfach konfiguriert werden (wenige Regeln, da nur der Verkehr einer Maschine geregelt werden muss) und sind sehr schnell (kein Netzwerk-Flaschenhals wie beim zentralen Firewall). Intrusion Detection Systeme wie Tripwire erlauben die Überwachung der Maschinenkonfiguration und entdecken verdächtige Aktionen. Sie sichern die TCB ab.

Kat.	Mechanismus	Wirkung
I	Application Server Security	Dies reicht von der Verwaltung von Schlüsseln für Single-Sign-On bis zur Minimierung von Rechten für Datenbankzugriffe. Das Härt(en) der Application Server wird in Teil 2 eingehend behandelt. Generell geschieht die Absicherung von Servern über Rechte-Reduktion bei der Ausführung, und Abschottung durch Virtuelle Maschinen
J	Application Design	Zentrale, überwachte Eingänge in Applikationen, die durch Regelsets kontrolliert werden. Vermeidung von Alias-Eingängen, die Kontrollen umgehen können.
K	Protocol Switch	Wechsel von Kommunikationsprotokollen (z. B. Eingang: http, Ausgang: Net-SQL) machen es für Angreifer schwerer, durch Fehler eines Protokolls tief in die Infrastruktur zu gelangen. Ähnliches gilt für den Wechsel von Betriebssystemen entlang eines Request-Pfades vom Internet ins Intranet. Ein klassisches Beispiel für „Defense-in-Depth“.
L	Packet Filter	Auch Ausgänge und Übergänge von Zonen müssen durch Filter zusätzlich gesichert werden. Bei DMZ Architekturen ist dabei auch auf Angriff von „Innen“ auf die Administration von Applikationen oder Servern zu achten. Deshalb sollten direkte Zugriffe vom Intranet durch eine weitere Zone gezwungen werden wo sie über Proxy-Rechner endgültigen Zugang in die DMZ erhalten.

Ähnliche Maßnahmen lassen sich natürlich auch weiter hinten im Bereich des Intranets durchführen:

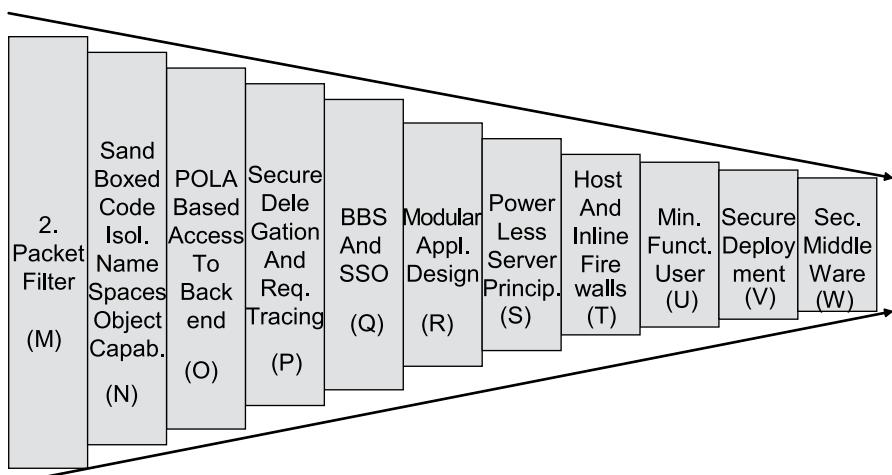


Abb. 11.12 Verkleinerung der Angriffsfläche im Intranet

Kat	Mechanismus	Wirkung
M	Packet Filter	Auch innerhalb des Intranets und beim Übergang von der DMZ sind Packet Filter zur Kontrolle von Adressen, Ports und Protokollen nötig.
N	Sandboxed Code, Isolated Namespaces, Object Capabilities	Maßnahmen zur Reduktion von Autorität innerhalb der Software. Die Rechte des Aufrufers werden gekürzt und eingeschränkt durch die Rechte der Software selbst. Isolierte Namespaces vermeiden globale Zugriffe auf Ressourcen. Sie bieten zugeschnittene Umgebungen für Applikationen. Object Capabilities sind granulare Zugriffsmöglichkeiten auf Ressourcen, die weitergegeben werden können. Zugriffe sind nur durch sie möglich.
O	POLA based backend access	Zugriffe auf Backend Funktionalität sollen durch möglichst geringe Rechte erfolgen. Backends sollen selbst Zugriffsrechte prüfen können. Ein breiter Zugriff auf Backends durch funktionale User ist zu vermeiden.
P	Secure Delegation and Tracing	Credentials, die zur Authentisierung verwendet werden, dürfen nicht weitergegeben werden (falsche Impersonation). Die Delegation von Requests muss für die Beteiligten sichtbar sein. Die Erlaubnis erfolgt über kryptografisch abgesicherte Token, die es den Empfängern erlauben, die Identität des Callers zu prüfen. Der genaue Pfad eines Requests über Intermediate Server ist für Empfänger-Server nachprüfbar.
Q	SSO and BBS	Single-Sign-On verhindert die Weitergabe von Authentisierungs-Credentials an viele Server. Ein zentrales Benutzerberechtigungssystem erlaubt die zentrale Kontrolle von User-Rechten und verhindert die lokale Verwaltung durch Applikationen, die bei Änderungen der Organisation fehlerhaft wird. Voraussetzung dafür ist die Modellierung Firmenspezifischer Rollen.
R	Modular Application Design	Multi-Tier Applikationsdesign verhindert den direkten Zugriff von Clients auf Backends und erlaubt das Filtern von Requests durch Business Logik auf den mittleren Schichten. Einfache Beispiele hierfür sind die Bereitstellung von festen Queries im Middle-Tier. Dies verhindert, dass Clients die Backend-Systeme gefährden können.
S	Powerless-Server Principle	Server, die mit den Rechten mächtiger Principals laufen, stellen eine große Gefahr dar. Hier ist die Impersonation ein wichtiges Instrument, diese Gefahr zu verringern, ebenso wie der Einsatz von Code-Access Control wie z. B. der Java 2 Security.
T	Host and Inline Firewalls	Auch innerhalb des Intranets lassen sich Rechner durch spezielle Host-based Firewalls zusätzlich absichern. Network-Edge Firewalls in Netzwerkkarten oder sogar transparente Firewalls ohne IP Adressen können effektive Barrieren darstellen.

Kat	Mechanismus	Wirkung
U	Minimal Functional User	Die Verwendung von Functional User Identitäten führt zum Problem der Nicht-Nachweisbarkeit von Aktionen. Ihre Verwendung ist daher zu minimieren.
V	Secure Deployment	Installation und Konfiguration sind abzusichern. Beispielsweise sollten Credentials zum Zugriff auf Services nicht bereits in der Entwicklung vergeben werden, sondern sollten im Rahmen des Deployments in einer sicheren Umgebung eingetragen werden. Die Code-Basis der installierten Applikationen, Middleware und Betriebssysteme ist strikt zu überwachen.
W	Middleware Security	Eine durchgängige Security-Schicht in Form von Middleware, die alle Requests durch Kontrollen zwingt, verhindert dass z. B. unterschiedliche Authentifizierungsniveaus (oder sogar unauthentizierte Requests) von Applikationen einfach akzeptiert werden. Gleichzeitig vermeidet so eine Schicht, dass die Applikationen die Sicherheitsdefinitionen selbst beachten müssen. Nutzer-Informationen begleiten über diese Schichten die Requests und erlauben deren Prüfung und Autorisierung.

Neben diesen Maßnahmen gibt es natürlich auch Entscheidungen, die die Angriffsfläche vergrößern statt verkleinern (s. Abb. 11.13). Klassische Beispiele dafür sind z. B. unauthentizierte Eingänge neben authentisierten Eingängen in Applikationen. Schnell führt hier ein Programmierfehler zur Weiterleitung unauthentizierter Requests auf Funktionen, die eigentlich eine strenge Authentifizierung voraussetzen würden. Die Weitergabe von Login-Credentials an beliebige Server, ein unter Root-Rechten laufende Server, weit reichende Datenbank-Rechte für Server sowie die Durchführung von wichtigen Sicherheitsmaßnahmen

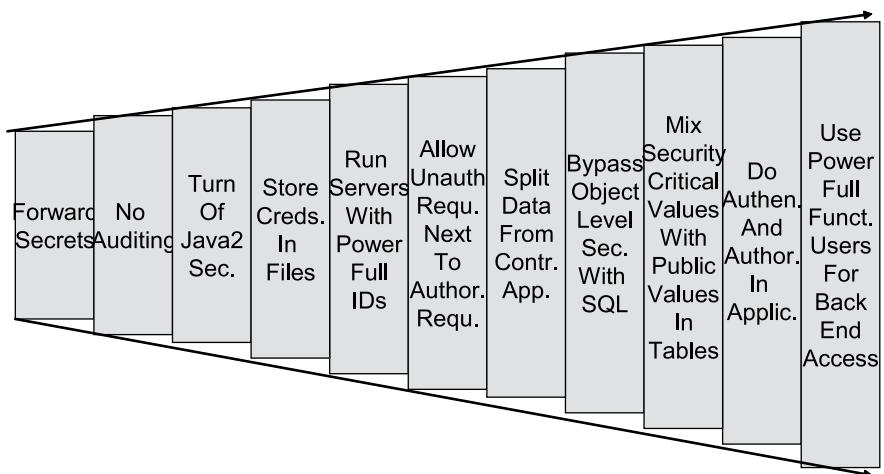


Abb. 11.13 Vergrößerung der Angriffsfläche im Intranet durch Designfehler

wie Authentisierung und Autorisierung in den Applikationen gehören ebenfalls dazu.

11.2.2 Ein Beispiel aus der Praxis

Eine Firma vertreibt Software für den Embedded Control Bereich und möchte sich gegen Raubkopierer durch Einsatz eines Lizenzierungsservers schützen. Kunden, die einen Vertrag abgeschlossen haben, erhalten Zugangsdaten (UserID, Passwort) und können dann selbstständig Lizenzen für ihre Maschinen erstellen. Diese Lizenzen sind an die MAC-Adresse der Maschinen gebunden.

Die Softwarefirma plant den Dienst als Self-Service in Form einer Webapplikation anzubieten, d. h. die Kunden holen und verwalten die Lizenzen selbst. Es ist die erste Applikation dieser Firma, die direkt mit dem Internet verbunden sein soll. Als Fremdsoftware wird ein Lizenzserver-Paket auf Basis des Tomcat Servlet Engine eingesetzt, das für den eigenen Bedarf konfiguriert werden kann (siehe Abb. 11.14).

Neben der optischen Integration mit einer existierenden Homepage (von dort sollen nur die Rahmen mit dem firmenspezifischen „Look & Feel“ übernommen werden) stellt sich jetzt die Frage, wo der Lizenzserver aus netzwerktechnischer Sicht aufgestellt werden soll. Die Firma besitzt bereits eine rudimentäre DMZ, in der sich jedoch lediglich nach außen gerichtete Proxy Server für Mail, News etc. befinden.

Entscheidend ist letztlich die Position des Application Servers – hier Tomcat – und seine Verbindung zu anderen Systemen. Der Firma bieten sich zwei Alternativen für die Aufstellung, die jeweils unterschiedliche Angriffsflächen besitzen.

- a) Aufstellung des Lizenzservers im Intranet
- b) Aufstellung des Lizenzservers in der DMZ

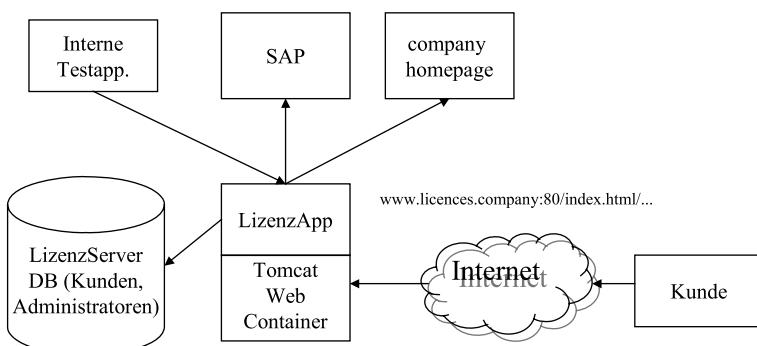


Abb. 11.14 Web-Infrastruktur einer Beispiel-Softwarefirma



The programmer conceptual model of her internet application is drastically different from the system/network security point of view.

Abb. 11.15 Aufstellung des Application Server im Intranet

Softwaretechnisch benötigt die Alternative a) lediglich eine einzige Verbindung vom Internet zu dem auf Port 80 laufenden Lizenzserver (siehe Abb. 11.15). Für Softwareentwickler entsteht daraus leicht die Vorstellung eines engen Filters, der die Kundenrequests auf die Homepage des Lizenzservers leitet. Es sieht so aus, als wäre nur ein kleiner Teil des Services überhaupt am Internet sichtbar. Das ist leider eine Illusion aus folgenden Gründen:

- Der Server `licenseserver.company.com` ist am Internet sichtbar. Gleichzeitig steht er in Verbindung mit anderen Rechnern im Intranet. Eine blinde Denial-Of-Service Attacke durch einige Rechner am Internet – also das Bombardieren dieses Rechners mit irgendwelchen Paketen zur Lasterzeugung – kann bereits die Verfügbarkeit nicht nur dieses Rechners, sondern des gesamten Intranets in Frage stellen.
- Natürlich antwortet der Lizenzserver nur auf Requests an Port 80. Das heißt aber nicht, dass andere Ports auf dieser Maschine nicht angreifbar wären. Je nach Konfiguration können weitere Services (auch unbewusst) von außen verfügbar sein. Fehler jeglicher Art (Remote Procedure Call subsystem, External Data Representation Algorithmen etc.) im Betriebssystem der Maschine können für Attacken ausgenutzt werden. Die Sicherheit hängt somit vom momentanen Patchlevel der Software ab und ist damit automatisch immer hinter aktuellen Bedrohungen im Nachteil.
- Der auf der Maschine laufende Application Server ist ebenfalls von außen angreifbar. Es können weitere Applikationen unbemerkt laufen (die vielleicht für Debug-Zwecke einst installiert und dann vergessen wurden), und Lücken im Applikation Server selbst erlauben ebenfalls die Übernahme der Maschine. Nicht zuletzt verstehen jedoch viele Softwareentwickler das WWW Konzept bezüglich des Zugangs zu Homepages nicht. Natürlich besitzt jede Site eine spezielle Seite für den Einstieg, in diesem Fall vielleicht `www.licenseserver.company.com:80/index.html`. Allerdings zwingt das niemanden, der aus dem Internet auf den Server zugreift, sich daran zu halten. In Wirklichkeit ist der gesamte Namensraum hinter dem „/“ für Zugriffe aus dem Internet verfügbar. Diese Tatsache wird auch *Deep Linking* genannt und ist oft – gerade bei statischen Seiten – ein erwünschtes Feature. Bei Applikationen allerdings möchte man verhindern, dass jemand auf diese Weise an Seiten gelangt, die nur durch

vorherige Authentisierung zugänglich sein sollten. Das bedeutet letztlich, dass die Applikation bei jedem Zugriff prüfen muss, ob bereits eine Authentisierung vorliegt. Wird das vergessen, hat die Applikation eine Sicherheitslücke. Das hört sich reichlich selbstverständlich an, jedoch haben Erfahrungen in Großprojekten gezeigt, dass solche Mechanismen, wenn sie programmatisch implementiert sind, das heißt, wenn die Programmiererin die Methode zur Authentisierungsprüfung explizit aufrufen muss, beim Einfügen von neuen Seiten oft vergessen werden – so geschehen beim in der Einleitung erwähnten OBSOC Vorfall. Im Ergebnis vertrauen wir bei der Alternative a) also darauf, dass Programmierer Prüfungen nie vergessen und dass die Konfiguration der Infrastruktur fehlerlos ist.

Für die Alternative a) spricht die enge Verbindung des Lizenzservers zu wichtigen internen Systemen wie SAP. Häufig benutzen auch interne Testapplikationen Techniken, die nicht webbasiert sind und eignen sich daher ebenfalls nur schlecht für den Betrieb in einer DMZ.

Eher gegen die Positionierung des Servers im eigenen Intranet spricht die direkte Verbindung aus dem Internet zu diesem Server (vgl. Abb. 11.16).

Wenden wir uns nun Alternative b), der Aufstellung des Application Servers in der DMZ zu (siehe Abb. 11.17). Für diese Alternative spricht, dass dadurch externe Zugriffe nicht sofort im mission-critical Bereich Intranet landen. Damit hat sich die Angriffsfläche zunächst deutlich verringert. Zwischen DMZ und Intranet wird zudem das ursprüngliche Requestprotokoll des Kunden (http) umgesetzt in andere Protokolle (z. B. Datenbank-Zugriffsprotokoll). Allerdings entstehen jetzt deutlich mehr Verbindungen in beiden Richtungen zwischen Intranet und DMZ, und die Absicherung dieser Verbindungen selbst ist ebenfalls ein Problem. Außerdem hat sich an der Angreifbarkeit des Lizenzservers selbst dadurch nichts geändert. Alles, was zur Angriffsfläche der Alternative a) gesagt

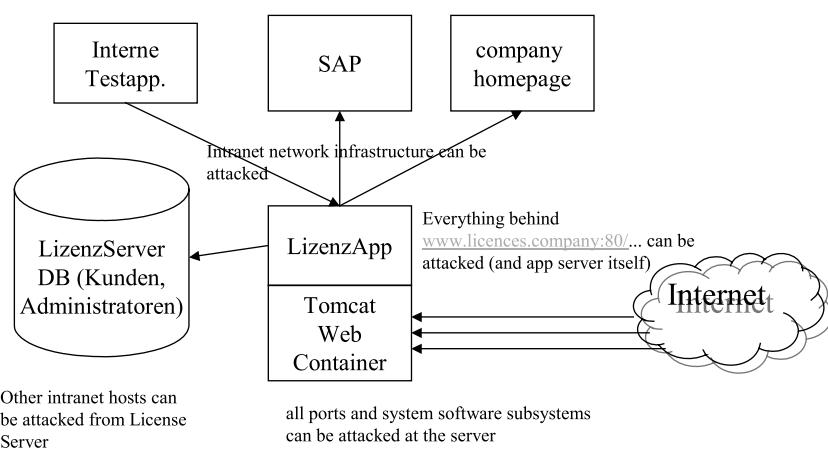


Abb. 11.16 Angriffsmöglichkeiten durch Positionierung des Application Server im Intranet

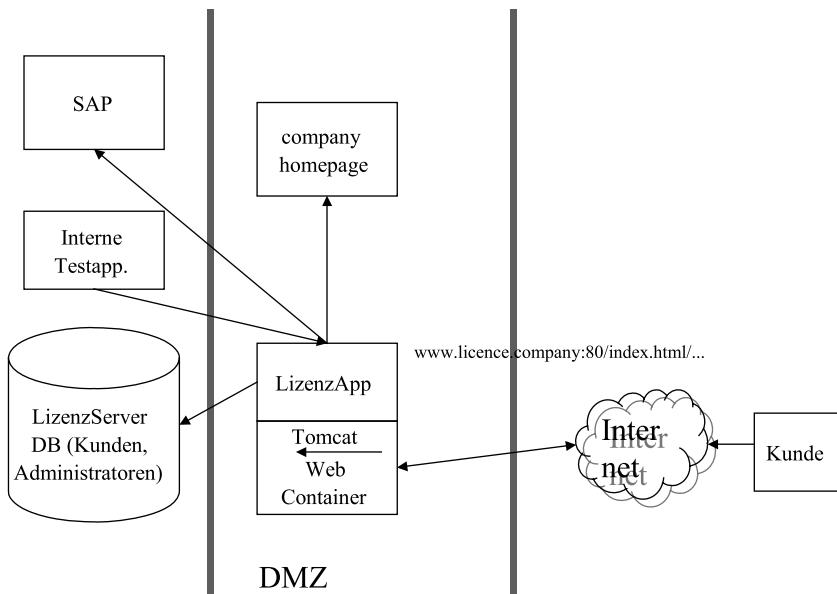


Abb. 11.17 Positionierung des Application Servers in der DMZ

wurde – mit Ausnahme der DOS Gefahr sowie der Gefahr für weitere Systeme – gilt hier ebenfalls.

Nehmen wir nun an, die Firma entscheidet sich dafür, den Lizenzserver im Internet zu positionieren, möchte aber trotzdem die oben genannten Gefahren begrenzen. Das waren, noch einmal kurz zusammengefasst:

- die Sichtbarkeit des Servers und seiner Subsysteme
- die Sichtbarkeit der gesamten Applikationen für unauthentisierte Zugriffe
- das Vertrauen auf die zuverlässig programmierte Authentisierung aller Pages
- Die Verwundbarkeit des internen Netzwerks

Dazu muss die im Konzept der Softwareentwickler ohnehin schon angenommene Verengung der Angriffsfläche implementiert werden. Im ersten Schritt werden vorgeschaltete Paketfilter konfiguriert, die alle Requests, die nicht für `www.licenseserver.company.com` auf Port 80 bestimmt sind, ausfiltern, protokollieren und verwerfen. Damit können unauthentisierte Zugriffe vom Internet nur noch auf den http-Port der Applikation gelangen.

Beim Versuch, die Lizenzapplikation weiter vor nicht authentisierten Zugriffen zu schützen, tritt jedoch ein konzeptionelles Problem auf: Die Applikation authentisiert selbst und muss daher zunächst einmal nicht authentisierte Zugriffe auf ihre Seiten zulassen, abfangen und dann zur Authentisierung zwingen. Das bedeutet aber beispielsweise, neben einer höheren Anfälligkeit für semantische Attacken (also Attacken, die gezielt nach Sicherheitslücken in der Applikation suchen) auch eine allgemeine Schwachstelle in Bezug auf Denial-of-Service Attacken, die da-

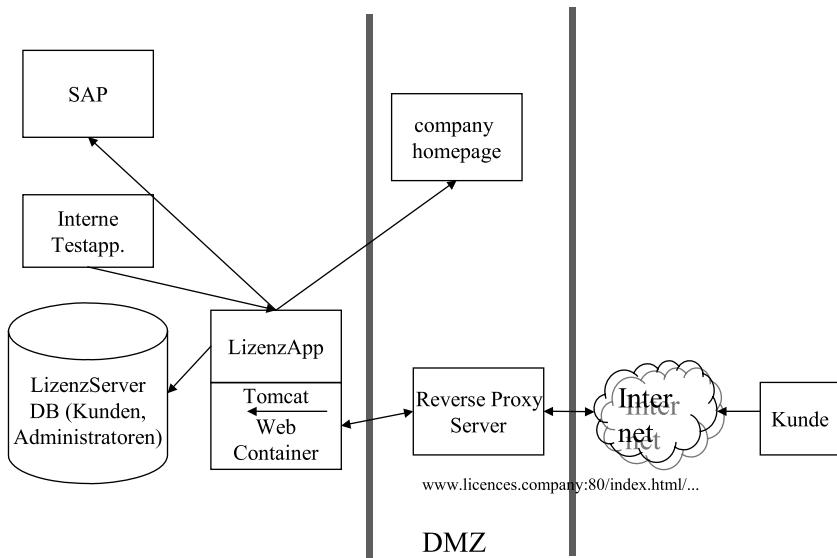


Abb. 11.18 Application Server im Intranet mit Reverse Proxy in der DMZ

durch bis in das Intranet vorstoßen können. Um dieses Problem zu lösen, wird im zweiten Schritt in der DMZ ein Reverse Proxy Server (RP) installiert. Dadurch entsteht die Netzstruktur in Abb. 11.18.

Der Reverse Proxy wird als hochsicheres System mit getrennten Netzwerkkarten (Bastion Host) ausgelegt und erzwingt damit eine physische Entkopplung der Netzwerke. Dies ist ein zusätzlicher Filter, der verhindert, dass schädliche Pakete durch automatisches IP-Forwarding ins interne Netzwerk gelangen können.

Wichtiger als dieser Entkopplungseffekt ist aber die Tatsache, dass der Reverse Proxy die IP Adresse des ursprünglichen Lizenzservers enthält und damit für die Kunden als solcher sichtbar wird. Selbst bei einer Schwäche in der Konfiguration der Paketfilter am Rande der DMZ können Internet Requests jetzt die Subsysteme des Lizenzservers nicht mehr direkt erreichen und attackieren.

Seine eigentliche Schutzwirkung erreicht der RP aber erst dann, wenn er nicht mehr nur als generischer Proxy arbeitet (das heißt Requests zwischen Kunde und Lizenzserver hin und herträgt), sondern wenn er das Authentisierungsprotokoll des Lizenzservers lernt und von da ab unauthentisierte Requests gar nicht mehr an den Lizenzserver weitergibt.

Jetzt stellt sich natürlich die Frage, wie sich ein Kunde denn überhaupt authentisieren kann, wenn jeder nicht-authentisierte Zugriff abgewiesen wird. Gehen wir momentan davon aus, dass der Lizenzserver eine so genannte Login-Page besitzt, auf die ein Kundenrequest verzweigt (page forward), wenn er noch nicht authentisiert ist. Diese Login Page fragt den Kunden nach UserID und Passwort und über gibt die Antwort dem Lizenzserver, der sie gegen die Datenbank prüft. Aber woran erkennt der Lizenzserver, dass noch keine Authentisierung vorliegt? Typischerwei-

se erzeugt die Applikation bei erfolgreicher Authentisierung ein Cookie (also ein Token, das auf Clientseite im Browser entweder temporär oder permanent gehalten wird). Dieses Token fließt danach mit jedem Request des Client zum Server, der daran die Authentizität des Clients erkennen kann (siehe auch Abb. 11.19). An dieser Stelle machen wir uns noch keine tiefer gehenden Gedanken zur Sicherheit dieses Verfahrens, aber es ist bereits jetzt klar, dass eine sichere Übertragung dieses Zugangstokens nötig ist.

Der Reverse Proxy kann sich jetzt das Wissen um das Verhalten der Applikation bei der Authentisierung zu Nutze machen und die Angriffsfläche der Applikation weiter verringern. Jeder Request vom Internet wird jetzt durch den RP auf die Existenz eines gültigen Cookies geprüft. Kommt ein Request ohne Cookie, so erkennt der RP, dass eine Authentisierung nötig ist. Der RP könnte jetzt den Request ganz gezielt nur an die login.html form der Applikation weiterreichen (bzw. einen redirect auf genau diese Page veranlassen). Das wäre bereits eine bedeutende Verengung der Angriffsfläche, da nicht authentisierte Requests nur noch genau bei dieser Page landen könnten und der RP außerdem eine Authentisierung erzwingen würde, selbst wenn die Applikation dies auf einer Page vergessen würde.

Alternativ könnte der RP jedoch selber eine Form zur Eingabe der UserID und des Passwortes aufbauen und die Werte vom Client abfragen. Damit würde verhindert, dass überhaupt noch nicht authentisierte Zugriffe direkt beim Lizenzserver landen. Es stellt sich jedoch die Frage, wie der RP in diesem Fall die Angaben des Clients prüfen würde. Auch hier ergeben sich wieder mehrere Möglichkeiten:

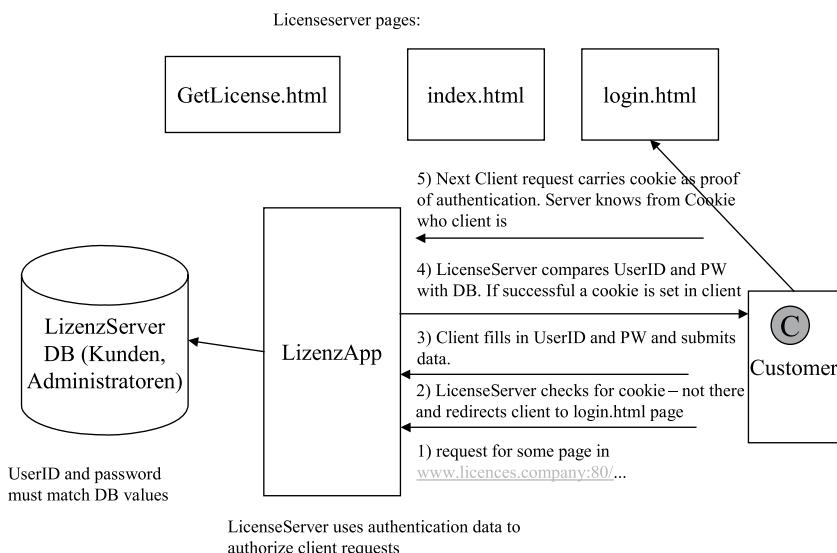


Abb. 11.19 Cookie-Basierter Authentifikationsmechanismus

- Der Reverse Proxy spricht direkt die Datenbank des Lizenzservers an. Diese Lösung ist elegant und schnell, aber leider bindet sich der RP damit an die Implementation der zugekauften Applikation und ihrer Datenstrukturen. Außerdem müsste der RP jetzt auch noch das Ergebnis der Prüfung – das Cookie – nachbauen können, und die Applikation würde plötzlich Cookies für authentisierte Sessions erhalten, die sie selber gar nicht angelegt hatte.
- Der Reverse Proxy führt selbst die Authentisierung bei der Applikation durch Aufruf der Formpage durch und gibt das dabei erstellte Cookie an den Client weiter.

In diesem Fall ist die zweite Lösung sicher die richtige. Allerdings zieht die Entscheidung, auf welche Weise die Authentisierung in der Software der Applikation zu vollziehen ist, unschöne Konsequenzen für die Infrastruktur nach sich, wie im Falle der Single-Sign-On Portallösung weiter unten noch viel deutlicher werden wird.

Wie könnte die Software für so einen RP aussehen? Für einen ersten Versuch genügt mit Sicherheit ein Web Container sowie ein Servlet, das die Client Requests prüft. Das neue ServletFilter API bietet sich dazu ebenfalls an.

Das Servlet kann die Gültigkeit der Cookies prüfen. Außerdem ist sichergestellt, dass die Authentisierungsseite nicht durch manipulierte Requests unterlaufen werden kann. Einige Fragen bleiben jedoch noch offen: Soll zum Beispiel ein unauthentisierter Clientrequest nach erfolgreicher Authentisierung (redirect auf login.html etc.) automatisch wieder aufgenommen werden? Falls ja, können Clients weiterhin Bookmarks tief in die Applikation setzen und müssen nicht neu dorthin navigieren. Alternativ könnte die Applikation durch eine Form des URL Rewriting sicherstellen, dass nur wirklich von der Applikation erzeugte Forms Pages durch den Client zurück geschickt werden können (etwa durch Einfügen von Zufallszahlen in die URL, die auf Seiten der Applikation zwischen gespeichert werden).

Auch die Frage der Gültigkeitsdauer der Cookies ist noch unklar. Hier ist vor allem zu bedenken, dass beispielsweise der Hibernation Mode moderner Laptops dazu führen kann, dass selbst temporäre Cookies Tage oder Wochen im Browser aktiv bleiben können.

Aber die zwei wichtigsten Fragen sind die nach der Absicherung der Verbindungen, etwa durch SSL, und wie gut die hier vorgeschlagene Lösung zukünftig erweiterbar sein würde, das heißt also die Frage nach ihrer Qualität in Bezug auf Wartung und Erweiterungen.

Zunächst zum Problem der Absicherung des Kommunikationspfades durch SSL. Wie bereits diskutiert, erlaubt SSL zwar auch die gegenseitige Authentisierung von Client und Server, aber in den wenigsten Fällen werden Clients über ein vom Service akzeptiertes Zertifikat verfügen. An dieser Stelle erwarten wir deshalb durch die Verwendung von SSL lediglich eine Authentisierung des Servers durch den Client sowie die Absicherung der Kommunikation in Bezug auf Vertraulichkeit und Integrität.

Aber unsere RP Lösung stellt uns jetzt vor ein neues Problem: SSL ist ein kanalbasiertes Protokoll. Das bedeutet: Die SSL-Verbindung des Clients zur Firma endet schon am Reverse Proxy und nicht erst am Lizenzserver. Natürlich könnte

auch direkt zwischen Lizenzserver und Client Browser eine SSL-Verbindung aufgebaut werden, allerdings wäre dann der Effekt des RP verloren: Der RP kann den Inhalt des SSL-Tunnels nicht auslesen und damit auch nicht filtern.

Die Kanalbasiertheit von SSL bedeutet auch, dass zwischen RP und Lizenzserver eine neue Verbindung aufgebaut werden muss. In diesem Fall wird man sich für eine gegenseitige Authentisierung entscheiden, ebenfalls durch SSL, jedoch unter Verwendung von Zertifikaten auf beiden Seiten. Je nach Architektur des Lizenzserver ist das lediglich eine Frage der Konfiguration oder aber eine massive Softwareanpassung. Wichtig ist hier, dass der Lizenzserver auch bei gegenseitiger Authentisierung die Identität des Clients nicht der SSL Session entnehmen darf – denn dieser Client ist ja immer der Reverse Proxy.

Für den Reverse Proxy ergibt sich durch diese doppelte Session jedoch eine interessante Möglichkeit, die Sicherheit der Applikation zu verbessern. Der originale Lizenzserver würde nach erfolgreicher Authentisierung ein Cookie für die Session beim Client setzen, das heißt ein Token im Browser des Clients platzieren. Jetzt landet dieses Token aber beim RP, der es nicht zum Client weiterreichen muss. Er kann statt dessen eine einfache Tabelle halten, in der er sich in einer Spalte das jeweilige Cookie und in der anderen Spalte der gleichen Reihe die dazugehörige SSL SessionID merkt. Auf diese Weise kann der RP beide Sessions aneinanderkoppeln, ohne dass das Cookie der Applikation auf den Browser gelangt, wo es angreifbar für Scripting Attacken und Browser Bugs ist.

Interessant im Zusammenhang mit gekoppelten Sessions ist die Frage nach dem Timeout der Sessions, da sich hier Überraschungen für die Benutzer ergeben können. Ist zum Beispiel der Timeout der SSL-Session abgelaufen und der Client klickt erst danach auf den „Logout“ Link der Applikation, so wird dieser Request vom RP abgefangen. Der RP stellt fest, dass die Session abgelaufen ist, und verlangt vom Client einen erneuten Login – nur um sich auszuloggen. Dieses Verhalten ist völlig normal, wenn auch für den Client ein wenig überraschend.

Zuletzt zur Frage der Wartbarkeit unserer Lösung. Das bedeutet, wir müssen klären, wie sich das Konzept verhält im Fall, dass

- Authentisierungsmethoden geändert werden;
- Kunden auch andere Applikationen benutzen können sollen (Single-Sign-On)
- die Kundenverwaltung zentralisiert werden soll um Karteileichen etc. zu entfernen;
- eine zentrale Rechteverwaltung aufgebaut werden soll und
- eine vorgezogene Authorisierung implementiert werden soll.

Schnell zeigt sich, dass unsere Lösung in allen diesen Fällen einen massiven Softwareaufwand nach sich ziehen wird. Das Grundübel ist schlicht darin zu sehen, dass die Applikation selber authentisiert, autorisiert und sogar ein eigenes Verzeichnis (user registry) besitzt – ein Fehler der Applikationsarchitektur und nicht nur ein Problem für die Sicherheit der Applikation.

Die Fragen nach dem Wie, Wo und Wann der Authentisierung einer Internet-Applikation lassen sich also für diese Beispiel-Anwendung so beantworten:

- Wann: So früh wie möglich, am Besten nicht erst innerhalb des Intranets;
- Wo: Nicht in der Applikation;
- Wie: Möglichst flexibel und mehrere Methoden unterstützend.

Dabei ist zu beachten, dass im Falle von applikationsübergreifender Authentisierung wie sie bei Single-Sign-On erforderlich ist (oder gar einer Domänen-übergreifenden Authentisierung) noch wesentlich mehr Regeln für die Authentisierung gelten. Beispielsweise muss geklärt sein, wie das Ergebnis der Authentisierung auf interoperable und sichere Weise an andere Systeme weitergereicht werden kann.

11.3 Single-Sign-On für Portale

Wir haben bereits in Kapitel zur Sicherheit in verteilten Systemen verschiedene Single-Sign-On Szenarien allgemein angesprochen. Nun wollen wir die Problematik speziell für Portale etwas genauer beleuchten. Grundsätzlich bietet ein Portal dem Kunden verschiedene Dienste an, die im Normalfall durch eigene Frontends bereitgestellt werden. Die Herausforderung besteht hier also in einer sicheren Weitergabe der Kundenidentität vom Portal an verschiedene Applikationsserver, die aber innerhalb der eigenen Domäne angesiedelt sind.

11.3.1 Vorstellung einer Portallösung mit Weitergabe der Identität

Selten haben Firmen die Chance, ihre Serviceangebote am Internet völlig neu zu entwickeln. Häufig haben unterschiedliche Abteilungen bereits vor der Portalentwicklung einzelne Dienste angeboten. So könnte im obigen Beispiel ein Service zur Bestellung von Hardware existieren. Schlimmstenfalls hat die Abteilung dahinter ihren Kunden eigene Identitäten gegeben, die in einer separaten Datenbank gehalten werden. Die Kunden der Firma müssen demnach verschiedene Identitäten und Credentials zur Authentisierung bei den verschiedenen Diensten unterhalten. Das „Look & Feel“ dieser Firmenapplikationen dürfte sich ebenfalls kräftig unterscheiden. Diese Situation ist natürlich für beide Seiten unbefriedigend (und potenziell unsicher).

Es liegt deshalb nahe, alle existierenden Service unter dem Dach eines Portals zu vereinen und bei dieser Gelegenheit eine Single-Sign-On Lösung zu schaffen, bei der die Kunden Zugang zu allen Diensten mit nur einer Identität und nur einem Passwort erhalten – natürlich im Rahmen der zugeteilten Rechte.

Die Grundstruktur einer solchen Lösung beruht auf dem „Trusted Third Party“ Design Pattern: Der Reverse Proxy des obigen Beispiels übernimmt jetzt die Rolle einer zentralen Autorität – dem Kernstück einer jeden Single-Sign-On Lösung – und bietet den angeschlossenen Diensten einen Authentication Service an

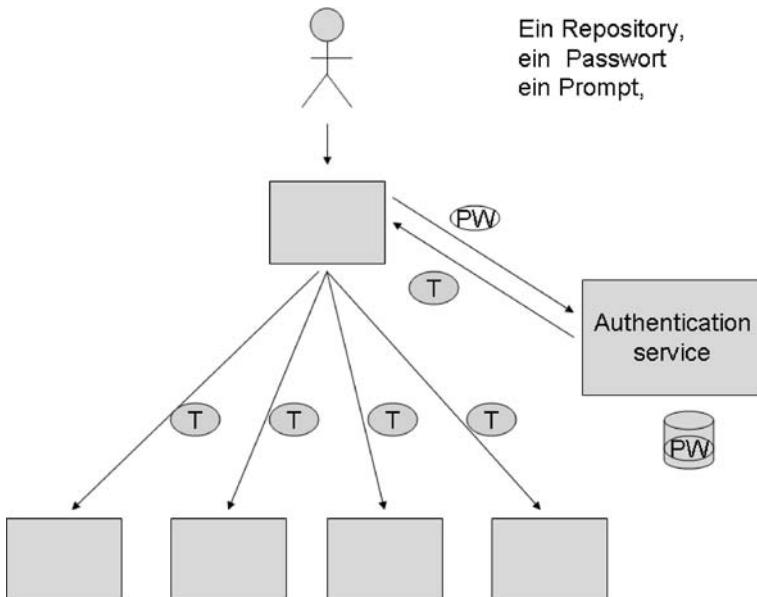


Abb. 11.20 Zentraler Authentication Service für ein Portal

(s. Abb. 11.20). Auf Seiten der Services fallen umfangreiche Änderungen an: In Zukunft führen sie keine eigene Authentisierung mehr durch, sondern stellen sich darauf ein, die Identität der Kunden mitgeteilt zu bekommen. Eventuell bieten sie jedoch auch noch zusätzlich Seiten an, die keine Authentisierung benötigen. In diesem Fall muss streng darauf geachtet werden, dass es zu keinen Verwechslungen kommt, das heißt, dass keine authentisierungspflichtigen Seiten von nicht authentisierten Kunden aufgerufen werden können.

Woher aber weiß der Reverse Proxy, ob eine Seite eines Dienstes authentisierungspflichtig ist? Diese Information enthält eine Datenbank, die jeweils ein Mapping zwischen Ressource (Seite) und Zugangsvoraussetzungen enthält. Also erweitern sich die Aufgaben des Reverse Proxy durch Access Control Methoden. Die dadurch entstehende Komplexität führt meist dazu, dass professionelle Lösungen für Identitätsmanagement und Zugriffskontrolle diese Aufgaben an getrennte Services delegieren:

Der Einsatz eines separaten Authentisierungsservers erlaubt es auch auf leichtere Weise, mehrere verschiedene Authentisierungsmethoden einzusetzen. Sicherheitstechnisch hat die Trennung von Authentisierungs- und Access Control Server zwei weitere Vorteile: Der Reverse Proxy als vorgelagerter Server ist besonders gefährdet. Deshalb sollte hier die Softwarekomplexität gering und überschaubar bleiben. Die Auslagerung der Access Control- und Authentisierungsdienste erlaubt außerdem, die Server dieser Dienste in besser geschützte Bereiche der DMZ oder sogar im Produktionsbereich des Intranets unterzubringen (Defense in Depth).

11.3.2 Aufgaben der Autoritäten

Lassen Sie uns kurz die wesentlichen Aufgaben der beiden zentralen Autoritäten in unserer Portallösung, dem Authentication Server und dem Access Control Server, festhalten. Dies sind:

- Identifizierung der Kunden durch verschiedene Verfahren;
- Mapping der Credentials (zum Beispiel UserID, Zertifikat) auf eine eindeutige interne Kennung;
- Festhalten der Metadaten des Requests: IP Adresse, Uhrzeit, Ort;
- Festhalten der Qualität der Authentisierung;
- Protokollieren des Eingang ins System und
- Ermitteln eines Rechte-Set für den authentisierten Kunden und dessen Weitergabe

Eine zentrale Autorität am Eingang zu einem System hat drei wichtige Vorteile: Zum einen isoliert sie die Software der weiter hinten angesiedelten Dienste vor Änderungen in den Methoden der Authentisierung (wenn beispielsweise ein neues biometrisches Verfahren zusätzlich angewendet werden soll). In diesem Fall müssten die Dienste alle ihre Eingangsrouterien der Authentisierung ändern. Authentisierungsframeworks für diesen Zweck wie PAM (Pluggable Authentication Modules) gibt es auf Unix Betriebssystemen schon lange und J2EE stellt mit JAAS (Java Authentication und Authorization Service) ähnliches zur Verfügung.

Der zweite Vorteil besteht darin, dass Geheimnisse der Kunden, die zur Authentisierung verwendet werden, nicht mehr allen Beteiligten einer Infrastruktur bekannt gemacht werden müssen.

Der dritte Vorteil wirkt sich vor allem in verteilten Systemen aus: Es besteht die Möglichkeit, Meta-Daten zum Request bereits ganz am Anfang durch einen einzigen Aufruf an Backend-Systeme zu erhalten und dann – zusammen mit dem Originalrequest – an Dienste weiter zu geben. Dies erspart den Diensten permanente Aufrufe an Backend-Systeme, um zum Beispiel bestimmte Rechteprüfungen durchzuführen. Dieses Verfahren nennt man *Propagation*. Jetzt wollen wir aber einige offene Fragen in Zusammenhang mit der obigen Lösung klären:

- Wie erfahren die Dienste die Identität des Kunden?
- Welche Informationen erhalten die Dienste?
- Auf was müssen die Dienste jetzt vertrauen und wie wird dieses Vertrauen abgesichert?
- Was geschieht auf der Seite der Dienste, nachdem sie die Identität erfahren haben?

11.3.2.1 Mitteilung der Identität und Security Context

In Webumgebungen bieten sich hierfür zwei Mechanismen an: Zum einen Eintragen als Name/Value Paare in den http-Header, der bei jedem Request dabei ist.

Zum anderen ist die Verwendung eines Cookie-Mechanismus möglich. Entscheidend ist dabei, dass der empfangende Service das jeweilige Format versteht und dass es keine Unklarheiten darüber gibt, ob ein Request bereits authentisiert wurde oder noch nicht.

11.3.2.2 Welche Informationen werden übertragen?

Diese Frage hängt natürlich von der Art des Single-Sign-On-Systems ab und kann von einer einfachen ID des betreffenden Kunden im System bis hin zu speziellen Rechten, Rollen oder der Art der Authentisierung reichen. Fast noch wichtiger ist aber hier die Frage, welche Information *nicht* mehr dabei sein muss, nämlich das vom Kunden verwendete Passwort! Die beteiligten Dienste erhalten so keine Möglichkeit mehr, einen Kunden durch Kenntnis seines Passworts einfach zu impersonifizieren. Grob lassen sich drei Typen von übertragener Information unterscheiden:

- Authentisierungsinformationen,
- Authorisierungsinformationen und
- Umgebungsinformationen.

Zusammen bilden sie den so genannten *Security Context*, in dem alle Informationen, die im Zusammenhang mit der Sicherheit eines Requests zur Verfügung stehen, zusammen gefasst sind. Dienste, die diesen Context empfangen, können bzw. müssen richtig darauf reagieren. Das Ziel ist hier, dass alle an einer Infrastruktur beteiligten Systeme diesen Context verstehen und weitergeben können, so dass ein Request während der Laufzeit durch das System immer anhand des Contextes sicherheitstechnisch bewertet werden kann. Eine Möglichkeit, wie ein Security Context etabliert werden kann, ist uns im Kapitel Middleware Security in Gestalt von CSIV2 bereits begegnet.

Hier zeichnet sich bereits ein größerer Fragenkomplex ab: Wer legt die Inhalte des Security Contexts fest? Wie kann er erweitert werden? Welche Einflüsse hat das auf die Interoperabilität des Contexts? Gilt der Context nur innerhalb der Infrastruktur einer Firma oder kann er auch global im Internet oder in virtuellen Organisationen wie z. B. Grid Computing genutzt werden? Wir werden diesen entscheidenden Punkt weiter unten beim Portalbeispiel und dann speziell im Kapitel zu föderativer Sicherheit weiter vertiefen. Dort werden wir mit Webservices und SAML Ansätze kennen lernen, die dieses Problem lösen können.

11.3.2.3 Absicherung des Security Context

Im einfachsten Fall besteht das Identitäts-Credential lediglich aus der internen UserID – eingebettet in den http-Header des Requests. Der empfangende Dienst muss in diesem Fall darauf vertrauen, dass der Request tatsächlich unverändert

vom Reverse Proxy kommt und dass eine ordnungsgemäße Authentisierung stattgefunden hat.

Eine erste Absicherung bestünde in der Verwendung eines sicheren Kanals zwischen Reverse Proxy und empfangendem Dienst, über den das Identitäts-Credential sicher transportiert werden kann. Ein sicherer Kanal lässt sich durch den Einsatz von SSL und einer zusätzlichen gegenseitigen Authentisierung leicht etablieren. Dies ändert aber nichts an der Tatsache, dass das Credential an sich keinen Beweis einer ordnungsgemäßen Authentisierung mit sich trägt.

Außerdem kann ein zusätzliches Passwort verwendet werden, welches zwischen Reverse Proxy und empfangendem Dienst vereinbart worden ist, und das über einen sicheren Kanal via http-Header transportiert wird. Der Empfänger kann das Passwort einen LDAP Aufruf prüfen (beispielsweise verwendet Tivoli Webseal diese Option). Dabei ist die UserID für den LDAP Aufruf auf Seiten des Dienstes fest konfiguriert.

Mehr Schutz als diese einfachen Lösungen bietet der Übergang zu einem *Identitätstoken* – einer kryptografisch abgesicherten Aussage über die Identität des Kunden. Dies kann durch die Verwendung eines symmetrischen Schlüssels geschehen, mit dem das Token über eine MAC-Bildung authentisiert oder verschlüsselt aber wird. Ausstellende Autorität und Empfänger teilen sich in diesem Fall einen Schlüssel. Dabei tritt der übliche negative Effekt symmetrischer Schlüssel in verteilten Systemen auf: Je mehr Teilnehmer, umso mehr Schlüssel müssen erzeugt und auf sichere Weise verteilt werden, wobei in diesem Fall der Schlüssel ausgerechnet für eine höchst sensitive Authentisierung verwendet wird. Noch mehr Sicherheit bringt deshalb ein durch einen asymmetrischen Schlüssel vom Authentication Server signiertes Token, das zudem noch über einen sicheren Kanal transportiert wird.

11.3.3 Heuristiken für Softwareentwickler

Die so genannte *Vulnerability Analysis* ist ein Kernthema von Sicherheits-Spezialisten. Entweder man sucht dabei formale Verfahren, um die Abwesenheit von Schwächen zu beweisen, oder es wird ein Konzept für gezielte Angriffe entwickelt, um Fehler zu finden. Die beweisenden Verfahren scheitern zumeist an der Komplexität der zu untersuchenden Systeme: Es gelingt uns selten unsere Systeme so von einander zu trennen, dass separate Sicherheitsabnahmen möglich sind – ein großes Problem der Webapplikationen, das die Kosten für die Sicherheit deutlich in die Höhe treibt. Die testenden Verfahren (Penetrationstests) auf der anderen Seite sind aufwändig und werden erst spät gegen Auslieferungszeit eingesetzt.

Gerade aber während der Planung von Softwaresystemen wäre es nötig, spontan Sicherheitsprobleme zu erkennen, um aus der Architektur resultierende Schwächen zu vermeiden. Aber welche Hilfestellungen kann man Entwicklern an die Hand geben, um Sicherheitsprobleme besser zu erkennen und in ihrer Gefahr bewerten zu können? In der Praxis haben sich dazu einige Heuristiken bewährt:

- Welche Informationen werden zwischen zwei Komponenten kommuniziert und wie ist ihre Bedeutung im Gesamtsystem?
- Welches Vertrauen setzt eine Komponente in eine andere?
- Welches Vertrauen wird in die Verbindung gesetzt?
- Was würde passieren wenn ein Stück Software durch bösartige Software ersetzt würde?
- Wenn Benutzer interaktiv beteiligt sind – wie wahrscheinlich ist ein Irrtum und was wären die Konsequenzen?
- Wie wahrscheinlich ist ein Irrtum auf Seiten der Entwickler? Können Rechte oder Authentisierungen verwechselt werden?
- Welche Stellen bieten sich für einen Angriff? Wie könnte der Angriff aussehen?

Schnell erkennt man, dass hier die bekannten Bedrohungsmodelle (Internet, Host, User etc.) genommen und mit systemspezifischen Eigenschaften (Bedeutung von Informationen z. B.) verknüpft werden.

11.3.4 Das Firmenportal

Die Einführung eines zentralen Identitäts- und Zugriffsmanagements mit zentraler Autorität allein sorgt noch nicht für ein gemeinsames Look & Feel aller Firmendienste. Erst deren Einbettung in ein Portalkonzept bringt einen einheitlichen Servicecharakter.

Dabei taucht jedoch ein Problem auf: Der Anschluss der beiden Dienste kann auf unterschiedliche Weise geschehen: Wenn die Dienste bereits vorbereitet sind auf eine Single-Sign-On Lösung, das heißt, nicht mehr unbedingt selbst authentisieren, dann kann das Portal die Identität des Kunden eventuell an die nachgeordneten Dienste weitergeben. Hier wollen wir kurz die Voraussetzungen dafür diskutieren. Im nächsten Abschnitt wird dann eine Lösung vorgestellt für den Fall, dass die Dienste noch nicht auf Single-Sign-On umgestellt sind und weiterhin eine eigene Authentisierung durchführen, und zwar gegen eigene, proprietäre Repositories mit speziellen Identitäten.

An dieser Stelle lohnt es sich, die beabsichtigte Semantik beim Ansprechen der Dienste durch das Portal nochmals zu untersuchen. Wir wollen, dass das Portal für einen Kunden die beiden Dienste benutzt, also im Auftrag des Kunden handeln soll. Diese Funktionalität nennt man, wie bereits besprochen, Delegation und sie erfordert natürlich eine Vollmacht des Kunden. Welche Credentials hat aber das Portal von der Single-Sign-On Autorität erhalten? Entweder einen String, der die Kundenidentität enthält, oder aber zwei Token mit dem gleichen Inhalt, die lediglich noch kryptografisch behandelt wurden.

Mit keinem dieser Credentials kann das Portal eine initiale Authentisierung gegenüber den Diensten durchführen. Dazu muss es eine eigene Authentisierung gegen die Dienste durchführen, am Besten eine gegenseitige. Dann allerdings kann es die über Single-Sign-On erhaltenen Credentials an die betreffenden Dien-

te weitergeben, also diese darüber informieren, dass das Portal im Auftrag dieses Kunden handelt. Dies zu akzeptieren liegt jedoch ausschließlich beim Dienst, da es eine totale Vertrauensbeziehung gegenüber allen vorhergehenden Intermediates voraussetzt (Backward Trust). Im Falle des simplen Strings als Identitäts-Credential kann der Dienst noch nicht einmal prüfen, ob überhaupt eine Authentisierung dieses Kunden stattgefunden hat. Bei dem authentisierten oder signierten Token hingegen ist dies möglich und deshalb werden diese Credentialformen auch als *forwardable* bezeichnet, das heißt, sie können so weitergegeben werden, dass sie ein Empfänger prüfen kann.

Das ist aber keineswegs eine Vollmacht (Delegation) für den Aufruf weiterer Dienste. Deren erste Stufe nennt man Impersonation: Hier übergibt der Caller dem Intermediate Credentials, die dafür geeignet sind, dass sich der Intermediate gegenüber dem Zieldienst als Caller ausgibt. Das wäre zum Beispiel dann der Fall, wenn der Intermediate UserID und Passwort oder einen Secret Key des Kunden erhält – was wir ja auf keinen Fall wollen. Nicht umsonst bezeichnet Impersonation in Wirklichkeit nichts anderes, als was unter dem Stichwort „Masquerading“ als eine Form von Attacke bekannt ist: Jemand gibt sich für jemanden anderen aus. Diese rudimentäre Form der Delegation bringt für Caller wie Dienstempfänger Gefahren mit sich.

Eine bessere Möglichkeit besteht in der Möglichkeit, eine echte Vollmacht für die Weiterleitung von Aufrufen auszustellen. Unter CORBA CSIV2 wurde dazu die Datenstruktur PAC (*Privilege Attribute Certificate*) definiert, die es erlaubt, eine solche Vollmacht auszusprechen. Ein PAC ist also ein Authorisierungstoken, kein Identitätstoken. Ein Protokoll zur Übertragung der Vollmacht ist durch CSIV2 SAS (Security Attribute Service, vgl. das Kapitel zu Middleware Security) gegeben.

Der empfangende Dienst muss nun durch gegenseitige Authentisierung prüfen, wer der Initiator (nicht der Originator) des Requests ist. Anschließend vergleicht er dessen Identität mit der Identität des in der Vollmacht genannten Bevollmächtigten (Intermediate). Bei Übereinstimmung kann er nun auch noch die Identität des Originators anhand der Signatur prüfen. Im Extremfall (so genannte *traced delegation*) kann der Empfänger die gesamte Kette der Delegation über alle Intermediates hinweg zurückverfolgen und die Entscheidung über die Akzeptanz des Requests darauf basieren.

Solange Dienste die beteiligten Strukturen und Protokolle, in diesem Fall die von CORBA CSIV2, verstehen, können also Requests von Kunden beliebig delegiert werden.

11.3.5 Integration vorhandener Legacy Systeme

Was passiert jedoch im – leider noch sehr häufigen – Fall, dass die Dienste noch auf einer eigenen Authentisierung des Kunden bestehen, die womöglich noch mit speziellen IDs durchgeführt wird? Hier bieten einige Infrastrukturen Single-Sign-On Lösungen an, die auf dem Prinzip der Impersonation beruhen: Der Intermedia-

te bekommt hierbei Credentials, die es erlauben, den Kunden gegenüber einem Dienst zu imitieren, und zwar so, dass der Dienst nicht erkennt, dass in Wirklichkeit ein Intermediate den Aufruf durchführt. Schauen wir uns diese Lösungen etwas genauer an:

11.3.5.1 Credential Vault

Ein Credential Vault ist ein Service, der zu einer bestimmten Identität Credentials speichert, beispielsweise UserID und Passwörter für verschiedene Dienste (s. Abb. 11.21). Dieser Tresor kann als Netzwerkdienst implementiert sein oder in Form einer Smartcard auf Seiten des Clients. Die meisten Web-Infrastrukturen bieten den Netzwerkdienst an.

In unserem Fall würde sich das Portal – nachdem es einen Request eines Kunden erhalten hat – an den Tresordienst wenden und für den jeweiligen Dienst, den es im Auftrag des Kunden verwenden soll, eine UserID und ein Passwort oder ein anderes Credential abholen, welches das Portal zu einer initialen Authentisierung befähigt.

Unter Verwendung dieser Credentials loggt sich das Portal bei den gewünschten Diensten ein und führt Requests durch. Die Dienste erkennen nicht, dass hier

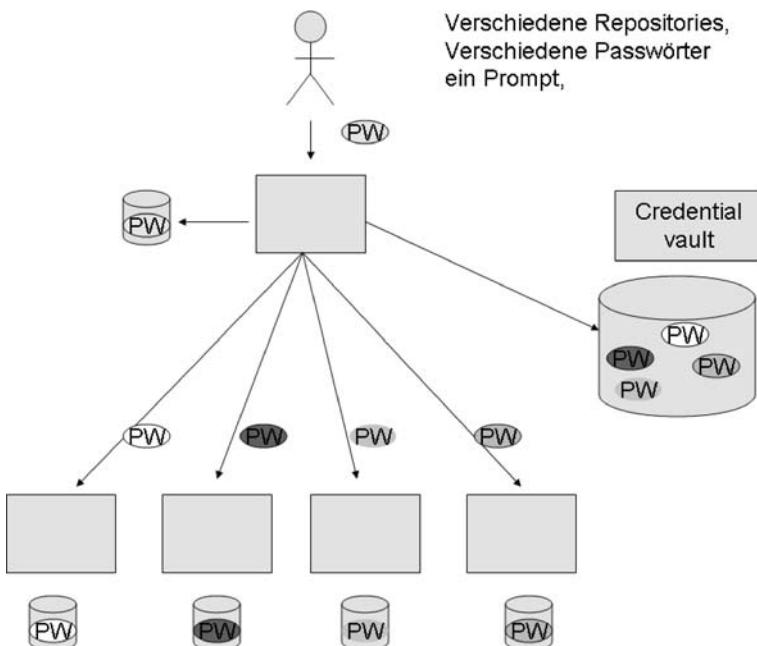


Abb. 11.21 Credential Vault

nur im Auftrag gehandelt wird, da die verwendeten Credentials ja zur initialen Authentisierung fähig sind.

Dieses Verfahren stellt ein klares Zugeständnis an die vorhandenen, zueinander inkompatiblen Dienste dar. Es verstößt gegen die grundlegende Eigenschaft von Passwörtern als Geheimnis zwischen zwei Parteien. Wie aber authentisiert sich das Portal gegenüber dem Tresor? Muss es wenigstens ein überprüfbares Identitätstoken des Kunden mitbringen, dessen Passwörter verwendet werden sollen, möglichst mit einer zeitlichen Terminierung? Oder loggt sich das Portal als ein Functional User beim Tresor ein und erhält Zugriff auf sämtliche Credentials?

11.3.5.2 Synchronisierte Registries

Eine weitere Möglichkeit der Integration von Legacy Systemen besteht darin, nur ein einziges Single-Sign-On Passwort für alle Dienste zu verwenden und dieses über alle vorhandenen Registries zu synchronisieren – also bei einer Änderung des Passworts die Registries aller beteiligten Dienste über einen Replikator sofort anzugeleichen. Dabei bleiben die separaten Registries bestehen, werden jedoch im Inhalt angeglichen (s. Abb. 11.22).

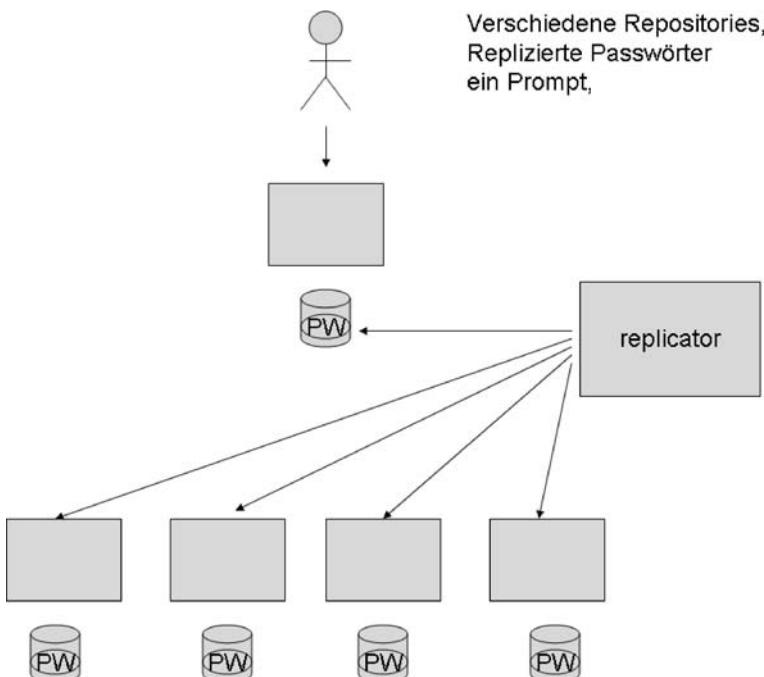


Abb. 11.22 Repliziertes Passwort für unterschiedliche Dienste

11.3.6 Ausbau des Security Contexts

Der Inhalt eines Single-Sign-On Tokens muss sich keineswegs auf Authentisierung oder Authorisierung der Delegation beschränken. Jegliche Information, die von der zentralen Single-Sign-On Autorität bestätigt werden kann, ist potentiell für spätere Entscheidungen nützlich. Dazu gehören beispielsweise Ortsinformationen (wo hat sich ein Kunde angemeldet?), Kundenrollen und Rechte, Zeitpunkt der Anmeldung, Gültigkeitsdauer usw.

Die frühe Ermittlung dieser Merkmale und die Tatsache, dass sie mit dem Request mitfließen, erlauben feingranulare Entscheidungen bei den Diensten ohne zusätzliche Anfragen. Probleme können auch hier wieder bei der Kompatibilität auftreten, da alle Beteiligten die Informationen auswerten müssen.

11.3.7 Step-Up Authentication

Die Step-Up Authentication stellt verschiedene Eingänge mit unterschiedlicher Authentisierungsqualität bereit, z. B. für Kunden, Mitarbeiter, Admins, etc. über Passwörter, PIN/TAN Verfahren, symmetrischem Challenge-Response bis hin zu PKI-Verfahren mit gegenseitiger Authentisierung. Dies kommt einem Interesse des Business entgegen, da häufig der Wunsch nach flexibler, unterschiedlicher Behandlung von bestimmten Kundenkreisen besteht.

Für die Applikationen sind verschieden gewichtete Zugänge jedoch sicherheits-technisch kritisch. Es muss sichergestellt werden, dass an allen Stellen, die den Request entgegen nehmen, die Unterschiede in der Qualität der Authentisierung beachtet werden.

11.3.8 Sicherheitsanmerkungen zu Single-Sign-On

Es steht ausser Frage, dass SSO einen positiven Beitrag zur Sicherheit von Infrastrukturen leisten kann, denn es verhindert die Notwendigkeit der Propagation von Authentisierungs-Credentials. Allerdings ergeben sich auch einige kritische Fragen. So sind Mechanismen, die zwar die verschiedenen Passwörter auf eines reduzieren, aber weiterhin eine unmittelbare Authentisierung bei allen Systemen erfordern, schlicht unsicher, da sie das einzige, wertvolle Geheimnis allen Systemen mitteilen.

Die gesteigerte Usability von SSO Lösungen wirft weitere Fragen auf: Was passiert, wenn eingeloggte Benutzer ihren PC verlassen? Kann ein schnell und automatisch aktivierter Bildschirmschoner dafür sorgen, dass die Applikationen nicht zweckentfremdet werden?

Und wie automatisch soll sich die Anmeldung an den Applikationen vollziehen? Soll der User überhaupt noch bemerken, dass eine Anmeldung durchgeführt

wird (z. B. indem er einen Mausklick durchführen muss?) Im Falle von Browserbasierten Attacken durch Script auf Intranet-Server ist der vollautomatische, transparente Login ein Sicherheitsproblem.

11.4 Mobile Infrastruktur

In den letzten Jahren haben sich aufgrund der Einführung immer leistungsfähigerer, kleiner mobiler Rechner die Firmen grundlegend mit der Tatsache auseinandersetzen müssen, dass es keine wirkliche Trennung von „Innen“ und „Außen“ mehr geben kann. Firewalls, durch die eigentlich jeglicher Datenverkehr von und zur Firma geschleust werden sollte, verlieren durch das Hereinbringen mobiler Geräte in die Firmeninfrastruktur viel von ihrer isolierenden Wirkung (s. Abb. 11.23, aus [Butz]). Die Abwehrmaßnahmen müssen dementsprechend ebenfalls am Ort des Einsatzes mobiler Geräte stattfinden und nicht mehr nur zentral in einer DMZ.

Die Vielfalt an Geräten und Angriffsvektoren im mobilen Bereich ist überwältigend und stellt Firmen vor allergrößte Probleme. Abwehrmaßnahmen sind nur beschränkt möglich und greifen teils sehr tief in die Usability der Geräte ein. Zur Verdeutlichung der Problematik ein Blick auf die Vielfalt der Geräte und Schnittstellen (Abb. 11.24).

Was also können Firmen tun, um ihre Infrastruktur zu schützen, wenn sie ihren Firmen die Nutzung mobiler Gerät nicht komplett untersagen wollen? Hier eine

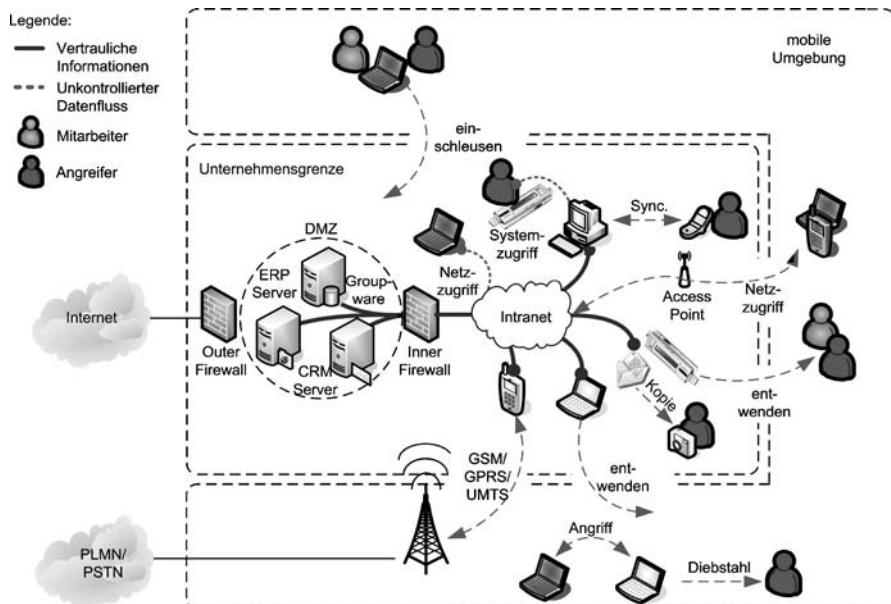


Abb. 11.23 Angriffsvektoren durch die Nutzung mobiler Geräte durch Firmenmitarbeiter

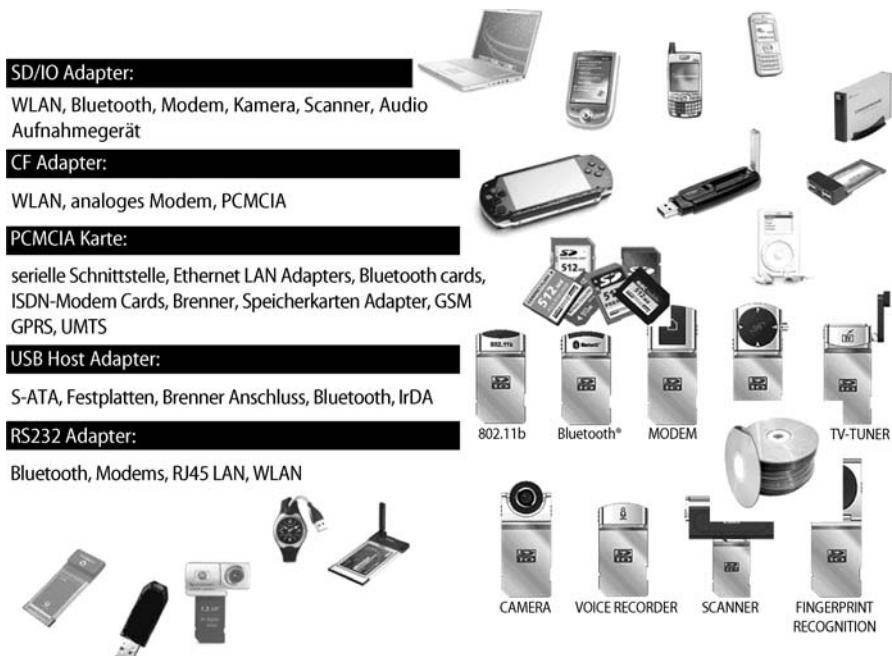


Abb. 11.24 Mobile Geräte und ihre Schnittstellen (aus [Butz])

kurze Auflistung möglicher Maßnahmen und der dahinter stehenden Technologien (für eine detailliertere Behandlung s. [Butz]):

- **Geräte-Authentisierung** durch Trusted Plattform Module-Maßnahmen. Das Gerät kann dadurch eindeutig identifiziert werden. Insbesondere kann auch sicher gestellt werden, dass keine fremde Hardware „eingeschleppt“ wird.
- **Überprüfung des Sicherheitsstatus** einer mobilen Plattform durch die Firma, z. B. Prüfung, ob sich das System genau im Auslieferungszustand befindet. Dies geschieht auf der Basis von Trusted Network Connections (TNC).
- **Authentisierung** durch Maßnahmen nach IEEE 802.1x (Port-Authentisierung). Die User-Authentisierung kann auch über ein VPN in das TNC-Framework eingebunden werden.
- **Rechtedurchsetzung** Analog zu den Konzepten von Microsoft im Unternehmensumfeld wird versucht, Firmenrechte auch auf den mobilen Geräten der Mitarbeiter durchzusetzen. Ein Beispiel hierfür wäre ein Verbot, unter lokalen Administrationsrechten zu arbeiten. Dies führt über zum nächsten Punkt:
- **Geräteschutz** Die Kontrolle der Benutzung mobiler Devices wie USB-Sticks, insbesondere:
 - Kontrolle der Daten (Protokollierung oder Verhinderung der Kopie) die auf Sticks gespeichert werden.
 - Kontrolle der Anschlussmöglichkeiten von Geräten an mobile Geräte.

- **Festlegung der erlaubten Kommunikationspartner.**
- **Verbindung zur Firma** Zum regelmäßigen Update mobiler Plattformen ist eine Verbindung zur Firma nötig. Dies geschieht am Besten über einen unabhängigen Public Carrier (z. B. über UMTS) statt durch das WLAN fremder Firmen, bei denen der Mitarbeiter evtl. als Consultant arbeitet.
- **Schutz der Daten bei Verlust des Geräts** und physikalischen Zugriff des Angreifers auf das Gerät durch Verschlüsselung des kompletten Contents, also z. B. Festplattenverschlüsselung (durch physikalischen Zugriff auf das System können viele andere Schutzmechanismen umgangen werden, zudem können durch Funktionalitäten wie den Hibernation-Modus beim Laptop verdeckte Kanäle auftreten). Zusätzlich:
 - **Einsatz von DRM** zum Schutz vor internen Attacken.
 - **Absicherung der Festplatten** durch Pre-Boot Mechanismen.
- **Ausbildung** Der Einsatz mobiler Geräte bedingt heute, dass die Benutzer Teil des Sicherheitskonzeptes werden müssen, da eine Automatisierung der besprochenen Konzepte noch nicht möglich ist. Aus diesem Grund müssen die Mitarbeiter geschult werden.

Von besonderer Bedeutung ist die doppelte Authentisierung von Gerät und User sowie die Kopplung der beiden auf Seiten der Firma. Durch die Verwendung von UMTS lässt sich auch eine zwangsweise Durchführung von Updates auf entfernten Geräten erzwingen. Wenn zusätzlich der Gerätestatus sicher überprüft werden kann, dann besteht die Möglichkeit, dass Firmen auch ihre mobilen Geräte einigermaßen beherrschen können.

Heute bieten viele mobile Geräte noch keinerlei Möglichkeiten, solche Maßnahmen durchzusetzen. Die Einführung von Trusted Computing Plattformen wird hier einiges ändern, da damit praktisch auch ein Host Intrusion Detection System auf den Geräten verfügbar ist. Da dieses IDS hardwaregebunden operiert, ist es gegen Angriffe besser geschützt als normale, Software-basierte IDS Systeme. Die mobilen Betriebssysteme werden sich zukünftig wohl in die Richtung entwickeln, dass sie die Nutzerrechte von der Verfügungsgewalt über das Gerät trennen und somit dem normalen Benutzer nicht mehr jede Aktion erlauben werden. Auf den mobilen Geräten bereits vorhandene Mechanismen wie die SIM-Karte und andere Smartcards werden auch für DRM-Maßnahmen verwendet werden können, wie wir sie bereits im letzten Kapitel diskutiert haben.

Dennoch: Die Identifizierung von Geräten und Personen, die sie betreiben bzw. besitzen, wird verbessert werden müssen. Selbst bekannte Zwei-Faktor Authentisierungs-Mechanismen wie Smartcard plus PIN verlieren bei mobilen Systemen schnell ihre Wirkung, etwa durch die Angewohnheit, die Smartcard immer im Laptop zu belassen.

Abschließend sei noch erwähnt, dass nicht nur die Gefahr, ein mobiles Gerät zu verlieren, sehr hoch ist, sondern auch die Bereitschaft, gefundene Geräte wie USB-Sticks oder CDs an die eigenen Geräte anzuschließen. Versuche an der HdM, bei denen CDs innerhalb der Hochschule, an S-Bahn Stationen etc „verlo-

ren“ wurden, haben dies eindeutig belegt. In Verbindung mit den heutigen unsicheren Betriebssystemen werden Features wie „Autorun“ – also das automatische Starten von Programmen auf mobilen Geräten – zur Falle. Natürlich sind solche Features per Default eingeschaltet, sie auszuschalten erweist sich jedoch als erstaunlich schwierig.

Kapitel 12

Föderative Sicherheit

Wir wollen in diesem Kapitel anhand von Beispielen die föderative Zusammenarbeit zwischen ansonsten selbständigen Rechtezonen (auch Domains oder *Realms* genannt) untersuchen, die zur Verfügung stehenden Techniken, sowie die Problematik der Interoperabilität zwischen den Domänen und deren Sicherheitstechniken. Konkret wollen wir webbasiertes SSO a la Liberty Alliance untersuchen sowie die Web Services-basierte Integration der Bestell- und Buchungsdienste im Backend.

Ein besonderes Augenmerk wollen wir auf die Vertrauensbeziehungen legen, die zwischen den Teilnehmern nötig sind und darauf, wie die verwendete Sicherheitstechnik diese beeinflusst. Außerdem stellt sich schnell die Frage nach dem Datenschutz angesichts der hohen Informationsdichte pro User, die sich durch die Verwendung des „Trusted Third Party“ Patterns ergeben. Außerdem ergeben sich durch die Föderation ganz spezifische Sicherheitsprobleme und Möglichkeiten für Attacken.

Besondere Bedeutung bei allen Cross-Domain Sicherheitsmechanismen und Geschäftsmodellen hat die Sprache, in der die Beziehungen ausgedrückt werden. Hier scheint sich die Secure Association Markup Language SAML von OASIS immer mehr durchzusetzen.

Abschliessen wollen wir das Kapitel der Föderativen Sicherheit mit einem Blick auf das Grid Computing und die besondere Form der *Virtual Organization* (VO). VOs bezeichnen ad-hoc gebildete Organisationen, die quasi auf Basis existierender Infrastrukturen und realer Organisationen eine eigenen Rechtebereich bilden, der auf die darunter liegenden echten Träger abgebildet wird. Die Probleme, die hierbei auftreten sind unserer Meinung nach stellvertretend für die zukünftigen Sicherheitsprobleme, die sich beispielsweise durch eine durchgängige Service-Orientierung als Business und IT-Modell ergeben.

Cross-Domain Security ist ebenso wie die Zusammenarbeit vieler Firmen und Organisationen zur Optimierung von Produktionsketten heute ein oft schmerzhafter Prozess – morgen jedoch Realität und ein ökonomisches Muss. Dabei spielt die Integration vorhandener Sicherheitsarchitekturen und die Entwicklung neuer, föderativer und integrierender Mechanismen eine entscheidende Rolle. Anhand von Grid Computing Techniken und Ideen wollen wir deshalb zeigen, wie Authentisierung und Authorisierung zwischen Partnern ohne eine zentrale Autorität funktionieren kann. Nicht zufällig ist diese Thematik eng verwandt mit den Problemen

der Peer-to-Peer Sicherheit, wie sie beispielsweise bei Peer-to-Peer-Dokumentensystemen auftreten.

12.1 Föderatives Identitätsmanagement

Föderative Strukturen entstehen meist aus technischen und/oder wirtschaftlichen Gründen. Der entscheidende technische Grund sind die Skalierungsprobleme verteilter Systeme. Direkte Kontakte und Vertrauensbeziehungen zwischen einer grossen Anzahl von Teilnehmern unterliegen dem Netzwerk-Effekt, der besagt, dass jeder weitere Teilnehmer einen Verwaltungsaufwand für *alle* Beteiligten bedeutet. Stellen Sie sich vor, Sie müssten im Falle eines Kaufvorganges bei jedem neuen Händler einen persönlichen Prozess zur Geldübergabe erstellen. (Netzwerk-Effekte sind natürlich keineswegs nur negativ: So steigert jedes weitere Telefon oder Faxgerät den Nutzen aller Besitzer, da sie dadurch mehr Teilnehmer erreichen können).

Eine technische Alternative zur totalen Verteilung aller Beteiligten mit hohem Aufwand für die Einführung eines neuen Teilnehmers besteht in einem zentralen Repository. Allerdings leiden sie nicht nur unter wirtschaftlichen und politischen Problemen (siehe unten), sondern skalieren ebenfalls schlecht bei sehr vielen Teilnehmern und Transaktionen. Weitere technische Mängel sind das Bestehen eines Single-Point-Of-Failure und Replikationsprobleme (s. Abb. 12.1).

Sieht man sich auf der anderen Seite die Topologien verteilter Systeme einmal genauer an, so stellt man schnell fest, dass sie häufig eine so genannte Scale-free

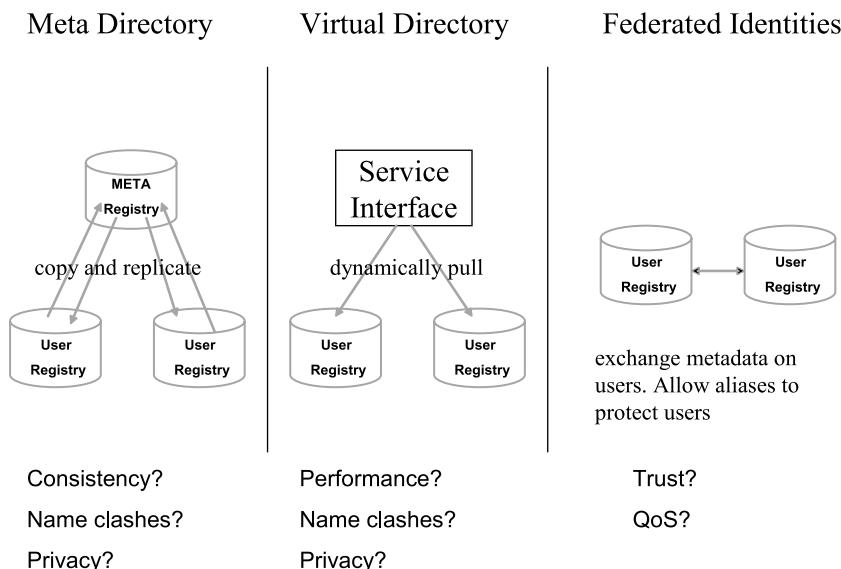


Abb. 12.1 Mögliche Alternativen beim Umgang mit mehreren User Registries

Architektur aufweisen, das heißt: Einer Vielzahl von kleinen Knoten stehen eine geringe Zahl mittlerer und eine noch geringere Zahl von Superkonten gegenüber. Diese Superknoten verbinden die Teilnehmer untereinander und sorgen für effiziente Kommunikationswege. Beispiele für solche Strukturen sind die großen Hub-Airports, die von den Airlines zur Erhöhung der Effizienz weiter ausgebaut werden (s. Abb. 12.2), oder das Internet. Die Effizienz des gesamten Netzwerkes wird durch die Existenz der Superknoten stark erhöht. Typisch für diese Superknoten ist, dass sie zwar selbständige Verwaltungszentren bilden, aber dennoch die Verbindung zu anderen Verwaltungseinheiten herstellen können. Diese Eigenschaft ist typisch für föderative Strukturen.

Aber nicht nur technische Gründe sprechen für föderative Strukturen: Der Traum von einem weltumspannenden Repository, das alle Menschen enthält, bringt nicht nur ein technisches Skalierungsproblem mit sich: Der Verwalter eines solchen Repositories hätte enorme politische Macht. Deshalb wird eine solche zentrale Struktur von der Mehrheit der Teilnehmer nicht geduldet werden. Klassisches Beispiel dafür ist die Ablehnung der von Microsoft vorgeschlagenen Passport/Hailstorm Initiative. Natürlich stellt dieser Vorschlag auch einen Albtraum in Bezug auf Datenschutz und Privatsphäre dar.

(Ein negativer Seiteneffekt eines globalen Repositories besteht auch in der Tatsache, dass die Mitgliedschaft dort kein spezielles Merkmal einer Person mehr ist. Kunde einer bestimmten Bank zu sein, kann hingegen durchaus für das Vertrauen von Dritten in eine Person von Bedeutung sein. X.509 Zertifikate haben dasselbe Problem: Sie bestätigen zwar den Zusammenhang zwischen einem Namen und einem Public Key, sagen jedoch über den Träger des Namens, insbesondere seine Vertrauenswürdigkeit, nichts aus. Entsprechend schwer tut sich ein Empfänger eines solchen Zertifikates mit der Validierung des Namens: Wer ist das überhaupt, der diesen Key besitzt?)

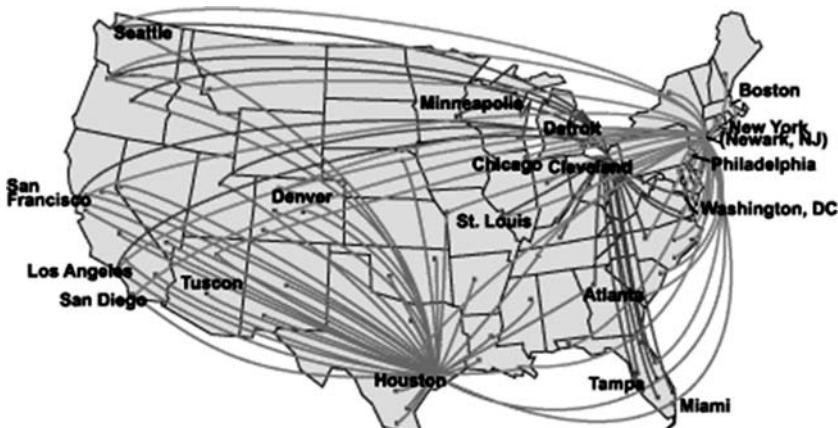


Abb. 12.2 Routendiagramm einer fiktiven Airline mit Super-Hubs in New York und Houston und einem mittleren Hub in Cleveland

Phillip Windley schildert in „Digital Identity“ ([Wind]) sehr schön die Entstehung der VisaCard als ein Banken – übergreifendes Zahlungsinstrument aus der ursprünglichen BankAmeriCard. Der Einsatzzweck einer Kreditkarte konnte nur dadurch wirklich erweitert werden, indem Besitzer der Karte bei Händlern damit bezahlen konnten, ohne dass diese Händler mit der die Karte ausstellenden Bank irgendeine Geschäftsbeziehung haben mussten. Die beteiligten Banken untereinander mussten hierzu föderativ zusammenarbeiten.

Die Kreditkarte erleichtert Kaufvorgänge enorm – sowohl für Kunden als auch Händler. Die beteiligten Händler bleiben dabei im Besitz ihrer Daten und Kundenbeziehungen.

In Bezug auf Identitätsmanagement ist es heute jedoch immer noch so, dass die Firmen und Organisationen im Wesentlichen ihr eigenes Identitätsmanagement betreiben. Damit sichern sie sich die Kontrolle über ihre Kundenbeziehungen und verhindern, dass Konkurrenten Daten über Kunden erhalten. Der Preis dafür wird von Geschäften wie den einzelnen Privatpersonen gezahlt: Der Einzelne muss sich eine Vielzahl von Passwörtern merken bzw. mehrere PIN/TAN Verfahren nutzen. Auf den Geschäften lasten die Kosten der Infrastruktur zur Verwaltung der Kunden sowie zur Erstellung von Identitäten und Prüfung der Authentizität.

Überträgt man die technischen und wirtschaftlichen Vorteile des Föderativen Managements von Zahlungen auf das Management von Identitäten, so kann man sich leicht eine Zukunft vorstellen, in der es – wie im Falle der Kreditkarte – mehrere Identitätsmanagementsysteme gibt, die in sich wiederum föderativ aufgebaut sind. Es dürfen jedoch nicht zu viele sein, anderenfalls nimmt die negative Wirkung des Netzwerkeffekts wieder zu. Technisches Kernstück von föderativen Lösungen ist letztlich immer das Abbilden (Mapping) der Kundeninformationen einer Domäne in die Sprache der anderen Domäne und die Reduktion der Anzahl der direkten Authentisierungen bis hin zum Single-Sign-On. Wirtschaftliches Kernstück ist die Sicherstellung der eigenen Kundenbeziehungen für die Unternehmen bei gleichzeitiger Reduktion der Authentisierungskosten. Darüber hinaus besteht die Chance, durch die Zusammenarbeit mit einem anderen Identity Provider mehr über einen neuen Kunden zu erfahren.

12.2 Föderatives Trust Management

Am Anfang jeder Transaktion steht das Erzeugen und Verwalten von Vertrauen zwischen Teilnehmern an Transaktionen. Dabei wird im föderativen Fall – ähnlich wie bei der Delegation – Vertrauen zunächst initial an einer Stelle erzeugt und dann über mehrere Zwischenstationen weitergegeben. Wie lässt sich dies technisch realisieren?

In Abb. 12.3 authentisiert sich ein Kunde gegenüber einem sog. *Identity Provider* (IP) und erhält dafür ein vom Identity Provider signiertes Token. Dieses Token präsentiert er einem Service Provider SP, der die Signatur des Tokens (das heißt seine Gültigkeit) prüft. Bei gültiger Signatur akzeptiert der Service Provider

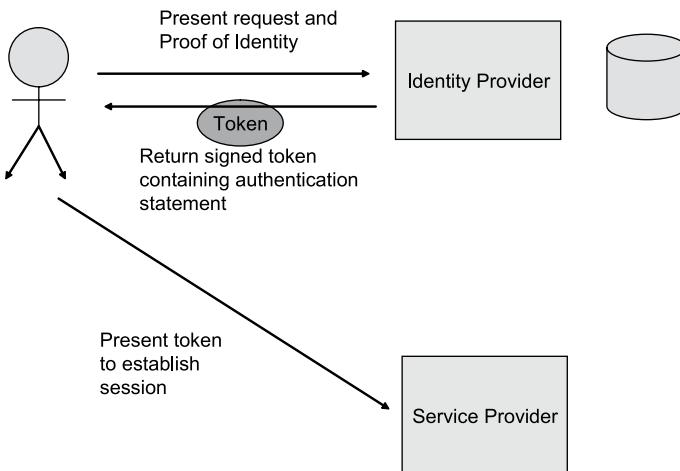


Abb. 12.3 Auslagerung der initialen Authentifikation an einen Identity Provider

die vorhergehende Authentisierung beim IP, ohne selber eine weitere Prüfung der Authentizität vorzunehmen, und gewährt dem Kunden den gewünschten Dienst. Er vertraut also dem IP in Bezug auf die Identität des Kunden. (Auf die Art und Weise, wie das Token zum SP transportiert wird, sowie die Angriffsmöglichkeiten, gehen wir weiter unten ein).

Bedeutet diese Abhängigkeit von einem Dritten nun weniger Sicherheit für den Service Provider? Der SP vertraut hinsichtlich der Authentisierung zweifellos einem Dritten. Ist der Prozess beim IP fehlerhaft, dann entsteht ein Schaden beim SP, der sich unter Umständen nur schwer rückgängig machen lässt.

Dennnoch gibt es Fälle, bei denen es sogar sicherer ist, einem Dritten zu vertrauen, als die Authentisierung selbst durchzuführen. Nehmen wir den Fall eines Mitarbeiters, der im Auftrag seiner Firma Produktionsteile bei einem Lieferanten bestellt. Typischerweise erstellt der Lieferant für diesen Mitarbeiter einen Eintrag in einem Verzeichnis (zum Beispiel LDAP) mit zugehörigem Passwort oder Zertifikat. Der Lieferant trägt die Kosten der Verwaltung, wenn der Mitarbeiter die Firma wechselt. Er trägt aber auch das Risiko, wenn der Mitarbeiter plötzlich anfängt, absichtlich falsche Bestellungen aufzugeben: Nehmen wir an, der Mitarbeiter ist im Streit von der Firma geschieden, sein Eintrag beim Lieferanten aber ist geblieben. Nach längerer Zeit fällt dem Mitarbeiter ein, er könnte seine alte Identität beim Lieferanten für Bestellungen nutzen. In vielen Fällen wird der Ex-Mitarbeiter damit Erfolg haben.

Betrachtet man den Vorgang aus dem Sichtwinkel von Vertrauensbeziehungen, so wird schnell klar, dass der Lieferant eigentlich an der Authentisierung des Mitarbeiters als Person überhaupt nicht interessiert ist, sondern nur an seiner Eigenschaft als zu Bestellungen berechtigter Mitarbeiter der Kundenfirma. Das Einrichten einer eigenen ID für den Mitarbeiter des Auftraggebers ist also eigentlich eine falsche Maßnahme.

Letztlich möchte der Lieferant, dass Aufträge authentisiert werden. Dies könnte durch eine Signatur von Transaktionen durch den privaten Schlüssel der Auftraggeberfirma geschehen. Wenn dies noch nicht möglich ist, zum Beispiel, weil die Software noch kanalbasiert arbeitet – das heisst, ein Login ist nötig zum Erzeugen einer authentisierten Session und alles, was in dieser Session passiert, ist vom Rechtevektor des eingeloggten Mitarbeiters abhängig – dann möchte der Lieferant zumindest eine Bestätigung durch die Auftraggeberfirma, dass der Mitarbeiter tatsächlich für die Firma arbeitet. Das könnte dadurch geschehen, dass die Auftraggeberfirma ein signiertes Token erstellt, indem sie bestätigt, dass die Identität des Mitarbeiters geprüft wurde. Die Authentisierung des Mitarbeiters gegenüber der eigenen Firma wird also umgewandelt in eine Bestätigung der Identität gegenüber dem Lieferanten (s. Abb. 12.4). Vorausgesetzt, die Auftraggeberfirma hat ihr Identitätsmanagement unter Kontrolle, kann ein ausscheidender Mitarbeiter somit sofort nach Ausscheiden keine Bestellungen beim Lieferanten durchführen, ohne dass dort ein zusätzlicher Verwaltungsaufwand entstanden wäre.

Der Lieferant kann dieses Token dann auf eine eigene definierte Identität („Besteller“) abbilden und loggt die im Token genannte Identität. Die Vertrauensbeziehungen sind jetzt so, wie sie sein sollen: Der Lieferant vertraut der Firma als Auftraggeber, nicht aber dem Mitarbeiter als Person.

Im oben genannten Beispiel ist es von Vorteil, dass der Mitarbeiter nach dem Ausscheiden aus der Firma keine Bestellungen mehr beim Lieferanten durchführen kann. Es gibt jedoch auch Fälle, in denen das Vertragsverhältnis zwischen Mitarbeiter und Drittfirmen auch nach Ausscheiden aus der Firma bestehen bleibt. Nehmen wir zum Beispiel an, dass der Mitarbeiter ein Frequent-Flyer Agreement

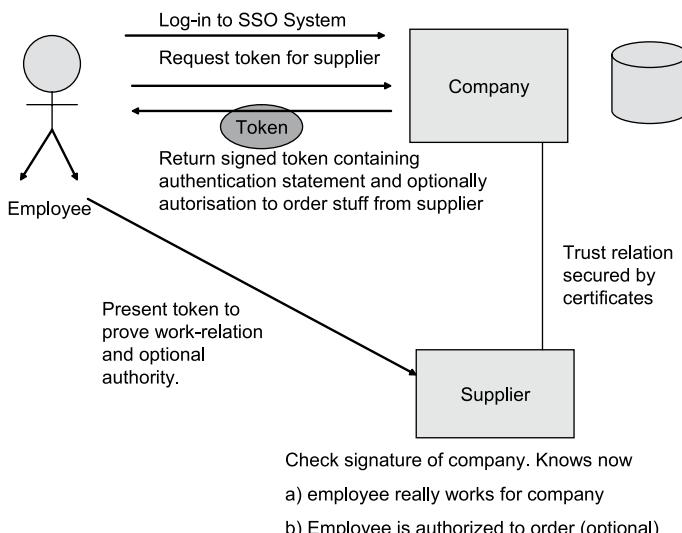


Abb. 12.4 Erweitertes SSO-Token bestätigt Identität und Berechtigung zum Bestellen eines Angestellten

mit einer bestimmten Fluggesellschaft hatte. Dieses Agreement bleibt auch nach dem Ausscheiden aus der Firma bestehen. In diesem Fall ist es nötig, dass die Drittirma (hier die Fluggesellschaft) dem Mitarbeiter eine eigene Identität erstellt, die unabhängig von der Firmenzugehörigkeit ist. Beim Eintritt in eine Firma kann der Mitarbeiter dann diese Identität mit der lokalen Identität innerhalb der Firma assoziieren. Wenn der Mitarbeiter bei seiner Firma angemeldet ist und versucht, einen Flug bei seiner Fluggesellschaft zu buchen, dann erhält die Fluggesellschaft ein signiertes Token der Firma mit einer unbekannten Identität (Alias). Die Portalsoftware der Fluggesellschaft erlaubt dann dem Mitarbeiter seine bei der Fluggesellschaft bestehende Identität mit dem Alias zu assoziieren. Dadurch erkennt die Fluggesellschaft dann in Zukunft, um wen es sich handelt, ohne dass sie beispielsweise den Loginnamen des Mitarbeiters in seiner Firma kennen muss. Das Mapping auf Seiten des Auftraggebers von der lokalen Identität des Mitarbeiters in eine bestätigte Identität für den Lieferanten muss durch Middleware automatisch erfolgen.

12.3 Föderative Sicherheit am Beispiel eines Portals

Im Folgenden wollen wir die Aufgaben sowie die physische Architektur föderativer Sicherheit an einem Beispielportal durchspielen.

Im Rahmen eines Portals oder einer Website, die Dienste anbietet wie etwa einen Webshop, gibt es drei föderative Aufgaben. Die erste Aufgabe, allgemein als Portal Single-Sign-On bezeichnet, haben wir gerade kennen gelernt. Sie besteht darin, dass alle Applikationen, die die Firma nach aussen anbietet, in einer Domäne vereint werden. Domäne meint hier, dass die Identitäten und Rechte der Benutzer verwaltet und an die beteiligten Rechner weitergegeben werden können, so dass keine erneuten Anmeldungen nötig sind. Bei den Identitäten handelt es sich ausschliesslich um die vom Portal selbst vergebenen/erstellten Identitäten, zum Beispiel, wenn sich ein neuer Kunde angemeldet hat. Wo findet sich in diesem Beispiel eine Föderation von Identitäten? Es handelt sich doch schliesslich nur um eine Organisation oder Firma? Nicht allen Firmen und Organisationen gelingt es, verschiedene Identitäten, die auf Grund historischer Entwicklungen entstanden sind, zu konsolidieren. Fusionen und Hinzukaufe führen oft dazu, dass innerhalb einer rechtlichen Organisation ein Kunde verschiedene Identitäten besitzt, die dann entweder in einem sehr aufwendigen Prozess konsolidiert (Phillip Windley beschreibt die dabei auftretenden organisatorischen und technischen Probleme am Beispiel des Staates Utah sehr eindrücklich – man denke nur an die Probleme einer globalen Namensgebung, Replikation und Synchronisation) oder völlig neu angelegt werden müssen.

Ein Application Server und Middleware sind deshalb oft in der Lage, verschiedene Identitäten einer Person innerhalb einer Firma an verschiedene Applikationen zu senden bzw. Mappings durchzuführen. Diese Mappings können dynamisch bei Bedarf oder im Batchbetrieb an Systeme verteilt werden. Es ist allerdings nicht

umstritten, dass innerhalb einer Organisation eine zentrale Verwaltung von Identitäten von Vorteil ist.

Die zwei anderen Aufgaben lassen sich am besten im Rahmen einiger typischer geschäftlicher Szenarien einführen. Im ersten der Szenarien beschliesst die Geschäftsleitung von bahn.de, in Zukunft Teil eines globalen Authentisierungsverbundes „IAuthenticated“ zu werden. Benutzer, die sich bei IAuthenticated anmelden, sollen auch von bahn.de ohne weitere Authentisierung erkannt werden. Die Technik von IAuthenticated beruht auf dem föderativen Konzept von Liberty Alliance (siehe unten) und erlaubt ein webbasiertes Single-Sign-On über verschiedene Domains hinweg.

Im zweiten Szenario entscheidet die Geschäftsleitung von bahn.de, dass das Portal den Kunden eine nahtlose Buchung internationaler Strecken und Verkehrsmittel ermöglichen soll. Konkret bedeutet dies, dass der Kunde über bahn.de fremde Verkehrsmittel gleich mitbuchen und bezahlen kann, ohne sich dafür bei anderen Anbietern erst wieder anmelden zu müssen. Die Abbuchungen erfolgen über das Buchungssystem von bahn.de.

In beiden Fällen ist das Portal Teil eines grösseren Verbundes und vertraut zu einem gewissen Grad Dritten. Im zweiten Fall gibt es zudem verschiedene Möglichkeiten der Implementierung, sei es direkt über peer-to-peer Verbindungen oder über die Teilnahme an einem internationalen Verbund von Verkehrsbetrieben „AllWays.com“.

In einem dritten Szenario entscheidet sich bahn.de, noch enger mit einer auf Reisen spezialisierten Bank zusammenzuarbeiten. Kunden sollen nach Anmeldung bei bahn.de bei dieser Bank ohne weitere Authentisierung Finanztransaktionen vornehmen können. An diesem Beispiel wollen wir das Problem von Session Timeout, Gültigkeit und Qualität von föderativer Authentisierung diskutieren.

12.3.1 Physische Architektur

Die Lösung für die physische Architektur von föderativem Identitätsmanagement besteht in erster Linie aus dem Einsatz von speziellen Reverse Proxies, die fremde Identitäten prüfen und gegebenenfalls auf eigene Identitäten abbilden können. Sie bilden den „Point of contact“ (siehe Abb. 12.5). In der gleichen physischen Zone (üblicherweise einer DMZ) befinden sich dann auch weitere Proxies, die ein Identitätstoken nach einer erfolgreichen Anmeldung erstellen können. Diese Proxies sind dann auch im Einsatz bei der anschließenden authentisierten Business-to-Business Kommunikation.

Ein weiterer Vorteil beim Einsatz von Reverse Proxies liegt darin, dass damit die eingesetzte Technologie ausserhalb der Kernbereiche einer Firma angesiedelt werden kann. Das bedeutet: Sollten sich die verwendeten Standards noch weiter entwickeln bzw. obsolet werden, so betreffen die Änderungen eng begrenzte Bereiche der Infrastruktur, aber nicht die gesamte End-to-End Security einer Firma bzw. deren Middleware. Aus diesem Grund ist auch das Aufkommen von so ge-

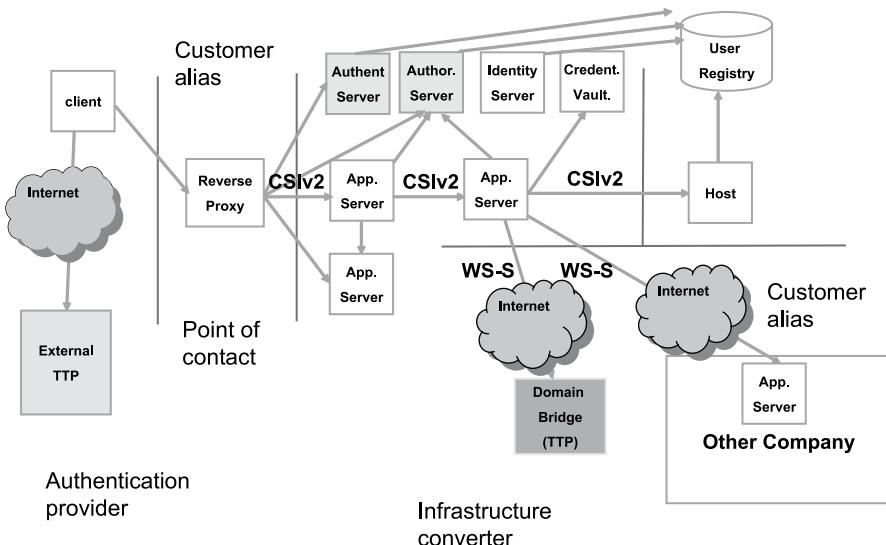


Abb. 12.5 Physische Architektur einer Lösung für föderatives Identitätsmanagement über Domänengrenzen hinweg

nannten XML oder Web Services Firewalls keine Überraschung. Diese verwenden zur Validierung externer Token oder zum Ausstellen von eigenen Token einen WS-Trust Service.

12.3.2 Einfacher föderativer Web-SSO zwischen Domänen

Wie funktioniert nun webbasiertes SSO über verschiedene Domänen hinweg? Lassen Sie uns zum besseren Verständnis diesmal mit einer eigenen, prototypischen Implementation eines webbasierten SSO Systems anfangen. Wir erinnern uns zunächst aus dem letzten Kapitel an die Tatsache, dass webbasiertes SSO innerhalb eines Portals darauf beruhte, dass alle teilnehmenden Server innerhalb der gleichen Domain angesiedelt waren und deshalb das SSO Token (das heißt im webbasierten Fall einen Cookie) empfangen konnten.

Des Weiteren greifen wir wieder auf das Security Pattern der Trusted Third Party (oder *Security Token Issuer*, STI) zurück, wie wir es auch im Portal-SSO Fall getan haben. Dort hat der Authentication Server die Rolle des STI gespielt, allerdings mit dem Unterschied, dass er Tokens nur für die eigene bahn.de Domäne ausgestellt hatte.

Als letzte Voraussetzung brauchen wir ein Verständnis des Redirect-Mechanismus von http. Über diesen Mechanismus kann eine Site einen Browser gezielt auf eine andere Site leiten und dabei noch Metainformationen an die neue Site übermitteln, unter anderem die eigene Absenderadresse (s. Abb. 12.6).

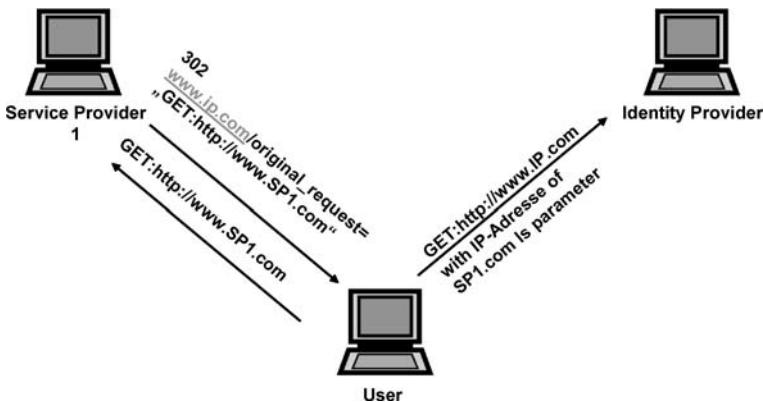


Abb. 12.6 Redirect eines nicht authentisierten Requests zum Identity Provider

Die Zielseite kann dann diese Informationen verwenden, um nach getaner Arbeit den ursprünglichen Request des Users wieder an die ursprünglich adressierte Site zurückzuverweisen. Im Fall des Identity Providers erzeugt dieser nach erfolgreicher Authentisierung durch den User eine Session zwischen User und Identity Provider sowie ein Token, das die Identität des Users sowie die Tatsache seiner Authentisierung gegenüber dem Identity Provider ausdrückt – am Besten natürlich kryptographisch gesichert durch eine Signatur (eine Verschlüsselung auf Basis eines symmetrischen Keys, der zwischen IP und SPs geteilt wird, würde bedeuten, dass ein böswilliger SP jederzeit Authentisierungstoken erstellen könnte).

Nun erzeugt der Identity Provider seinerseits einen Redirect Request, zurück an den Service Provider, von dem der erste Redirect kam. Zur Erinnerung: Der Originalrequest des Users an den SP1 war als Parameter beim ersten Redirect mitgegeben worden. Der IP nutzt nun diesen Wert zur Erzeugung der Zieladresse (zu-

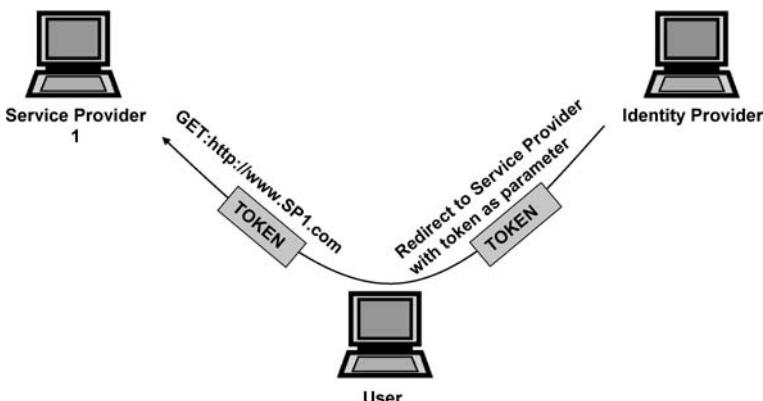


Abb. 12.7 Der ursprüngliche Request des (jetzt authentisierten) Users wird an den Service Provider zurück verwiesen.

rück an den SP1), fügt aber seinerseits als Parameter das Authentisierungstoken an (s. Abb. 12.7).

Ziel des webbasierten Cross-Domain SSO Verfahrens ist es, dass sich ein User nur einmal bei einem STI anmelden muss und dann von allen beim STI angegeschlossenen Webdiensten (z. B. Webshops) akzeptiert wird – ohne weitere Anmeldung. Beginnen wir mit einer Beschreibung der Implementation und untersuchen ihre Sicherheitsqualitäten und Problemzonen im Anschluss daran. Das Verfahren kann noch weiter ausgebaut werden zu einem Meta-Cross-Domain SSO: Dann werden sogar Identitäten von User und Service Provider bei verschiedenen Identity Providern in einer Business-Transaktion verknüpft.

12.3.2.1 Implementation

In unserem Szenario sind als Voraussetzungen gegeben:

- ein User, der den Browser zum Einkaufen verwenden will;
- ein Security Token Issuers, bei dem der User Mitglied ist und wo er Credentials zum Anmelden hinterlegt hat, also zum Beispiel UserID und Passwort;
- die beteiligten Services haben Verträge mit dem STI.

Dieses Szenario lässt sich mit ein paar Servlets auf einem Application Server sehr leicht simulieren und studieren. Der Source Code in Java und XML sowie begleitende Dokumentation und Sequenzdiagramme finden sich bei [ERS]. Nun der zeitliche Ablauf einer Authentifikation:

- Der User John Doe startet den Browser und tippt eine URL ein, die auf eine Bestellseite von www.shop1.com zeigt: www.shop1.com/orders.
- Shop1.com empfängt den Request mit Hilfe des Order Servlets und untersucht die URL auf weitere Parameter. Er stellt fest, dass keine vorhanden sind, insbesondere kein SSO Token des STI, das über die Identität des Users Aufschluss geben könnte.
- Shop1 erzeugt einen http-redirect Befehl, an dessen Ende er die URL der verlangten Site anfügt und als dessen Ziel er die Login Page von STI angibt: www.STI.com/login?source=www.shop1.com/orders.
- Der Browser empfängt den Redirect Request und richtet einen Get-Request mit der obigen URL an STI.com.
- Die STI.com login-Page erhält den Request und schickt – da der User noch nicht authentisiert ist – eine Form Page an den User zurück mit der Aufforderung sich durch Eingabe von UserID und Passwort oder mit anderen Mitteln zu authentisieren.
- Der User füllt die html-Form aus und schickt sie zurück an den STI. Dort werden UserID und Passwort extrahiert und gegen das STI Repository verglichen. Wenn sie korrekt sind, erzeugt der STI ein Token, in dem unter anderem steht: User = John Doe und verschlüsselt das Token mit seinem Schlüssel.

- Anschließend erzeugt der STI einen Redirect Request zurück zum ursprünglichen Ziel `www.shop1.com/orders` und fügt als Parameter `?token= "aölksjfajsfjdkjöfja"` – also das verschlüsselte Token – an. Der STI hatte sich die ursprüngliche Site entweder als verstecktes Feld innerhalb der Login Form oder über ein Session Cookie gemerkt.
- Der Browser führt den Redirect Request aus und `shop1.com` empfängt den ursprünglichen Request ein zweites Mal – diesmal allerdings mit einem angehängten SSO Token des STI.
- Diesem Token entnimmt `shop1` nun die Identität des Users und eventuell weitere Authorisierungsdaten. Alternativ führt `shop1.com` mit dem Token einen Lookup direkt beim STI durch und erhält dadurch mehr Daten über den User John Doe.
- `Shop1.com` setzt nun ein eigenes Session Cookie im Browser des Users, das ab diesem Moment jeden weiteren Request des Users an `shop1.com` begleitet und seine Identität ausweist.
- Nachdem User seine Einkäufe auf `shop1.com` erledigt hat, geht er mit dem Browser auf `shop2.com`. Dort spielt sich genau das Gleiche ab, wie zuerst bei `shop1.com`: Es ist kein Token vorhanden und es findet ein Redirect zurück auf den STI statt.
- Der STI kann jedoch – vielleicht anhand eines vorher gesetzten Cookies – die vorher authentisierte Session mit dem User erkennen und stellt sofort wieder ein Token aus – vielleicht das gleiche wie bei `shop1`. Ab da läuft alles wie für `shop1.com` beschrieben.

Wenn der User seinen Einkauf beendet hat, kann er auf den Seiten von `shop1.com` oder `shop2.com` zwischen einem lokalen oder globalen Logout wählen: Wählt er „lokal“, vernichtet der jeweilige Service sein Session Cookie. Wählt er „global“, erfolgt ein Redirect auf den STI. Dort wird die Login-Session beendet durch Vernichtung des STI Cookies, worauf ein Redirect erfolgt an jeden beteiligten Service (der STI kennt sie ja durch das Ausstellen der Token), wo wiederum die jeweiligen service-spezifischen Session Cookies vernichtet werden. Schließlich wird der Request wieder an den STI zurückgeleitet. Damit ist der User bei allen Services ausgeloggt.

12.3.2.2 Funktionale Erweiterungen und Alternativen

Im Folgenden werden wir kurz einige Erweiterungen des obigen Single-Sign-On Beispiels in Richtung Föderation diskutieren. Bei dieser Erweiterung sind die folgenden Aspekte zu beachten (detaillierte Beschreibungen der verschiedenen Formen von Föderation von Identität und ihre Bereitstellung für Services finden sich bei [Bueck]. Die folgenden Darstellungen orientieren sich an der dort verwendeten Terminologie):

- WAYF (Where are you from): Wie entdeckt der Service Provider den STI, der zuständig ist?
- Account Linking (SP und Kunde sind Mitglied bei mehreren IPs),

- Umfang und Inhalt des Tokens,
- Umgang mit anderen User Agents und Transportprotokollen, wie etwa Web Services.

Das obige Beispiel beschreibt zunächst nur eine einfache Form webbasierten Single-Sign-Ons (vergleichbar mit der Microsoft Passport eingesetzten Technik). Föderativ wird es in dem Moment, wo

- der Service Provider ein eigenes Identitätsmanagement betreibt und der Kunde ein beim SP registrierter Kunde ist;
- der Kunde mehrere IPs benutzt, bei denen er angemeldet ist;
- der SP vertragliche Beziehungen mit mehreren anderen SPs hat.

In diesen Fällen tauchen zwei der obigen Probleme auf: Account linking und WAYF (Where are you from?). Der Service Provider steht in diesen Fällen vor dem Problem, dass ein Kunde einen Service Request stellt, ohne dass dem SP bekannt ist, mit welchen IPs der Kunden eine Beziehung hat bzw. welchen IP er in diesem Moment benutzen möchte. Verschärft wird das Problem, wenn der SP seinerseits Verträge mit mehreren IPs hat. Außerdem könnte es sein, dass der SP noch ein eigenes Identitätsmanagement betreibt und der Kunde somit auch den „hauseigenen“ Benutzernamen verwenden könnte.

Föderatives Identitätsmanagement muss nun Funktionen zur Verfügung stellen, die es dem Service Provider erlauben, den richtigen IP zu identifizieren (also das WAYF Problem zu lösen). Außerdem muss es dem Kunden erlauben, seinen existierenden Account beim SP mit seinem Account beim Identity Provider zu verbinden (Account linking), ohne dass der Service Provider notwendigerweise den Accountnamen des Kunden beim IP erfährt.

Der Service Provider kann das WAYF Problem auf zwei Weisen lösen: Durch ein eigenes User Interface, in dem der Service Provider eine Liste von Identity Providern aufführt, mit denen er eine vertragliche Beziehung unterhält (sprich: deren Authentisierung er vertraut). Der Kunde kann dann durch Auswahl seines IPs den oben beschriebenen Redirect zum IP auslösen und letztendlich dem SP ein Token des IPs präsentieren. In diesem Token enthalten ist ein Alias für den Kunden (also nicht der echte Benutzername beim IP), der jedoch immer für diesen Kunden steht. Diesen Alias kann der SP nun mit dem Benutzernamen des Kunden in der Domäne des SP verknüpfen und speichern. Erzeugt der Kunde in Zukunft (nach Anmeldung beim ISP) einen Request an den SP, dann kann der SP den Alias sofort mit dem internen Usernamen verknüpfen und den Kunden damit als authentisiert anerkennen. In diesem Fall spricht man von einem „Pull“ der Authentisierungsinformation vom IP.

Die zweite Möglichkeit besteht darin, dass der Service Provider mit mehreren IPs eine vertragliche Bindung eingeht und diese IPs dem Kunden nach seiner Anmeldung eine Liste von personalisierten Links präsentieren. Diese Links enthalten bereits den Alias des Kunden im Token. Der IP führt hier also einen „push“ der Authentisierungsinformation zum SP durch (s. Abb. 12.8).

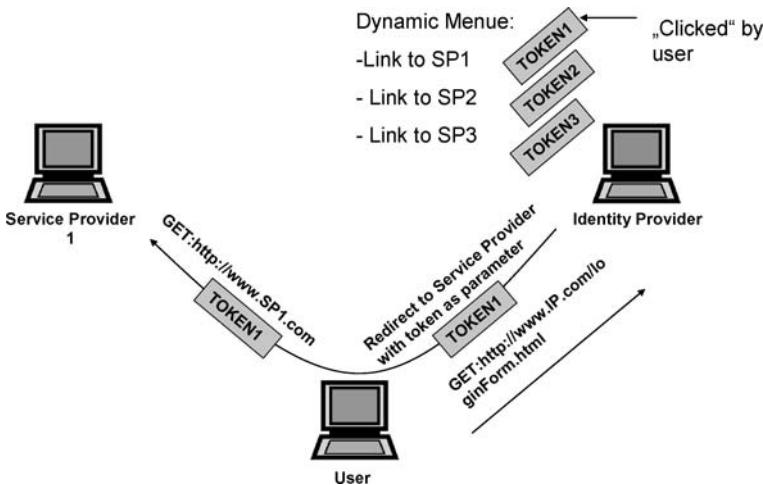


Abb. 12.8 Push der Authentisierungsinformationen zum gewählten Service Provider

Die Gestaltung des Tokens erzwingt unterschiedliche Kommunikationswege zwischen Service Providern und Identity Providern. Das Token kann lediglich aus einer Random Number bestehen, unter der der Service Provider dann weitere Informationen zum Kunden beim IP einholen kann. Alternativ kann das Token auch bereits die Informationen enthalten, die der IP zum jeweiligen Kunden anbieten kann – formuliert z. B. in Form von SAML Aussagen (Assertions). In diesem Fall wäre ein so genannter Back Channel zwischen SP und IP zum Einholen weiterer Informationen nicht nötig.

Für das Problem des Account Linking gibt es ebenfalls mehrere Szenarien. Hier gehen wir davon aus, dass der Kunde bereits einen eigenen Account beim SP besitzt und der SP dem Kunden die Möglichkeit eröffnen möchte, sich in Zukunft auch durch Anmeldung bei seinem IP einloggen zu können. Dazu muss sich der Kunde zunächst beim SP mit der SP-eigenen ID anmelden. Daraufhin bietet der SP dem Kunden eine Liste von möglichen IPs an (oder erfährt davon über andere Mechanismen wie zum Beispiel Cookies). Der Kunde wählt einen seiner IPs aus und es folgt der oben beschriebene Redirect Mechanismus, der zur Erstellung eines Tokens durch den IP führt. Der SP kann nun den im Token enthaltenen Alias mit dem eigenen Benutzernamen des Kunden verknüpfen und speichern. In Zukunft wird der SP Requests akzeptieren, die ein gültiges Token des IPs mitführen bzw. beim IP ein solches Token beantragen, falls ein Kunde einen Request stellt. Alternativ kann ein IP Identity Provisioning auch auf Verlangen über den Back Channel durchführen. Wenn der Kunde zustimmt, werden Accounts bei den SPs direkt im Batch Modus eingerichtet.

Entscheidend aber ist, dass der Username des Kunden beim IP nicht an die Service Provider gemeldet wird. Diese erfahren nur einen globalen Alias (global im Sinne des ausstellenden IPs) des Kunden und können eine eigene Identität dafür anlegen. Einen Alias dieser Identität können sie wiederum dem IP melden und dort mit der

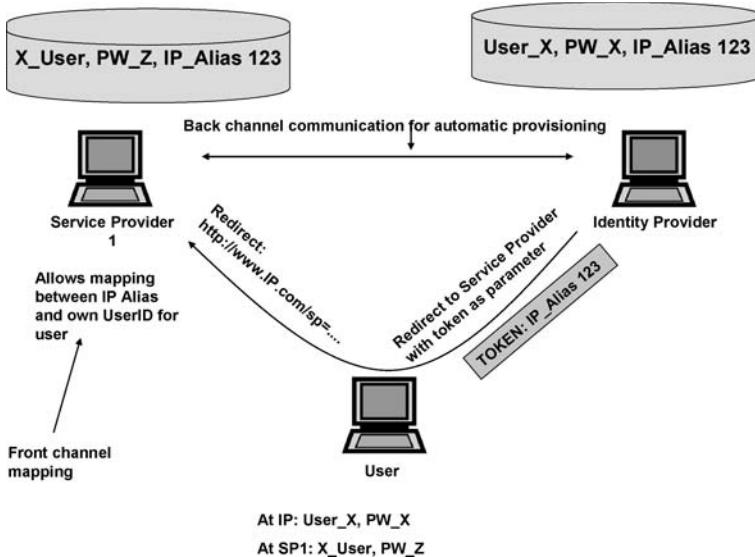


Abb. 12.9 Account Linking: Derselbe User ist unter zwei verschiedenen UserIDs und einem gemeinsamen Alias bei IP und SP bekannt

tatsächlichen UserID des Kunden verknüpfen lassen. Somit kommunizieren IP und SP jeweils nur Aliasnamen eines Kunden statt die echten UserIDs (s. Abb. 12.9).

Durch dieses Verfahren ist echte Föderation getrennter Domänen möglich, d.h. die einzelnen Identitäten des Kunden bei den Providern bleiben bestehen.

WAYF und Account Linking sind zentrale Bestandteile des Liberty Alliance Standards, der sich speziell mit dem Problem der Föderation von Identitätsdaten auseinandersetzt.

Kernpunkt des SAML Standards hingegen ist das Token oder besser: Der Inhalt der Aussagen des IPs über einen Kunden und dessen Anmeldung. Token bedeutet hier ganz streng eine signierte Aussage eines Identity Providers und hat nichts mit den Credentials des Kunden zu tun (also den Daten, die der Kunde verwendet hat, um sich gegenüber dem IP auszuweisen und damit erst ein Token zu erhalten.). Ziel des föderativen SSO ist ja gerade, dass der Kunde seine Credentials nur gegenüber einem einzigen Identity Provider angeben muss.

Aus technischer Sicht kann das Token, wie gerade gesehen, aus einer nicht vorhersagbaren Zahl bestehen, mit deren Hilfe der Service Provider dann über den Back Channel Daten zum Kunden vom IP abfragen kann. Alternativ kann das Token auch die komplette, signierte Kundeninformation direkt enthalten, zum Beispiel als signierte XML Datei. Dies ist abhängig von der Implementation und dem verwendeten Profil. So ist beispielsweise das Browser-GET Profil nicht geeignet zum Transfer großer Datenmengen über die URL.

Letztlich ist aber für den Austausch von Accountinformation oder Authentisierungsinformation die verwendete Sprache entscheidend. Der Austausch zwischen Sicherheitsdomänen kann nur dann funktionieren, wenn ein gemeinsames Ver-

ständnis über die Daten und deren Austauschformat existiert. Genau dieses Format wird in SAML definiert. SAML ist eine auf XML basierende Sprache zum Erstellen von sicherheitsrelevanten Aussagen.

Für den Fall, dass wir nicht auf http basierende Protokolle für die Erzeugung und den Transport von Tokens verwenden wollen (das heißt, keinen Browser nutzen wollen, sondern eine Verbindung von Applikation zu Applikation aufzubauen), stehen wir erneut vor einem Problem: Wir müssen ein Protokoll zum Beispiel zur Erzeugung eines Tokens definieren. Wenn wir Account Linking unterstützen wollen, müssen wir ein Protokoll definieren, welches das Linking von Informationen erlaubt. Programmatische Sicherheit war und ist das Kernthema der so genannten Web Services Security.

Der oben geschilderte SSO Vorgang schildert eine mögliche Weise, wie das Vertrauensverhältnis zwischen Client und Identity Provider auf einen Service Provider übertragen werden kann. Die Art der Abbildung eines Security-Protokolls auf konkrete Techniken wird gerne *Binding* oder auch *Profile* genannt. Diese Begriffe finden sich sehr häufig in den Spezifikationen von SAML, Liberty Alliance und Web-Services Security.

12.4 Standards für föderatives Identitätsmanagement

Mehrere verschiedene Standards konkurrieren bzw. verschmelzen hier momentan und ergeben leider ein recht unübersichtliches Bild. Wie ähnlich sich die Interaktionen zwischen Kunden, SPs und IPs bei den verschiedenen Standards allerdings sind, lässt sich sehr schön am bereits erwähnten Redbook der IBM zu Föderativem Identity Management [Bueck] erkennen. Dort sind Beispiele von Sequenzdiagrammen für die Durchführung von Transaktionen mittels SAML, Liberty Alliance und Web Services Federation aufgeführt.

Das größte Problem für Entwickler ist hier mit Sicherheit die Vielfalt neuer Terminologien und die daraus entstehenden Unsicherheiten. Wir werden hier deshalb nur ganz grob auf die Grundprinzipien hinter den Standards eingehen und für die Details auf die entsprechende Literatur verweisen.

Allen hier kurz vorgestellten Standards ist eines gemeinsam: Die Sicherheit des Protokolls wird nicht mehr ausschließlich durch ein kanalbasiertes Verfahren – im allgemeinen SSL – hergestellt. Föderatives Identitätsmanagement bezieht sich notwendigerweise immer auf Interaktionen zwischen mehreren Partnern und Intermediates. Deshalb reicht es nicht aus, eine einmal erfolgte Authentisierung über einen sicheren Kanal per SSL von Intermediate zu Intermediate zu transportieren (Kanalbasierte Security). Stattdessen müssen alle diese Standards ein Konzept einer sicheren Nachricht (Messagebasierte Security) verwenden. Dies wird üblicherweise durch digital signierte Nachrichten verwirklicht. Föderatives Identitätsmanagement verwendet diese Nachrichten (Tokens), um die Tatsache einer erfolgreichen Authentisierung gegenüber einem Identity Provider an Service Provider zu transportieren. Diese Service Provider erstellen daraufhin eine Session

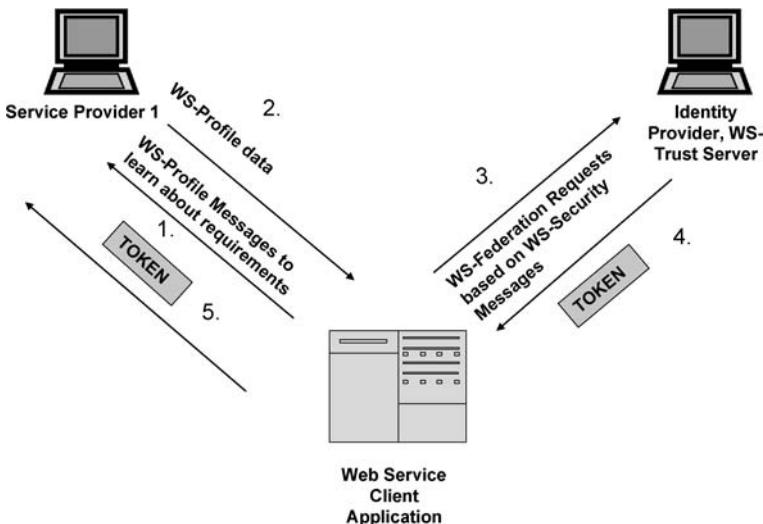


Abb. 12.10 Föderatives Identitätsmanagement mit Nutzung von Web Services

mit dem Kunden ohne weitere Authentisierung. Der Mechanismus dieser anschließenden Session ist nicht mehr im Scope der Standards.

Wichtig ist festzuhalten, dass durch die Einführung einer sicheren Nachricht grundsätzlich die Türe in Richtung sicherer Transaktionen aufgetan wird. Besonders deutlich wird das im Falle der Web Services Federation, wenn dort das „Active Profile“ (das heißtt, ein Web-Service fähiger User Agent) verwendet wird (s. Abb. 12.10).

12.4.1 SAML

Der SAML (Secure Markup Association Language) Standard der Oasis-Open Group [SAML] stellt eine Policy Expression Language dar und besteht aus zwei Teilen. SAML ist erstens eine XML basierte Sprache zur Erstellung sicherheitsrelevanter Aussagen (Assertions) und enthält damit Elemente wie User, Attribut, Gültigkeitsbedingung, Quality-of-Service etc. SAML definiert auch sog. Autoritäten die in der Lage sind Sicherheitsstatements auszugeben. Es gibt drei Hauptkategorien für Assertions:

- Authentication Assertion,
- Attribute Assertion und
- Authority Assertion.

Damit kann ein Identity Provider ausdrücken, wie und wann ein Kunde authentisiert wurde, welchen Namen der Kunde hat und ab wann welche Aussage gültig sein soll.

Der zweite Teil von SAML ist eine Abbildung der Aussagen auf ein konkrete Transportprotokolle und Szenarien (z. B. Browser-basiertes Protokoll), die sog. Bindings und Profile, mit denen Domain-übergreifend Sicherheitsaussagen transportiert werden können. Damit lässt sich dann u. A. ein Single-Sign-On realisieren. In diesen Bindings sind zum Beispiel Redirect-Mechanismen beschrieben analog zu dem oben geschilderten Beispiel. Die Profile hingegen beschreiben die Einbindung von SAML Tokens in andere Sicherheitsframeworks wie Liberty Alliance oder WS-Security. Die Abb. 12.11 zeigt beispielhaft eine SAML Assertion über die Authentisierung einer User-Identität durch einen Security Token Issuer.

Da über das http-GET Profil die Größe der Information in einem Token sehr begrenzt ist, bietet SAML die Möglichkeit eines reinen Artefakt-Tokens an: Dieses dient nur dazu, dass im Anschluss der Service Provider mit Hilfe des Artefakt Tokens die eigentlichen Aussagen vom Identity Provider über den Backchannel abrufen kann.

SAML 2.0 wurde gegenüber SAML 1.0 um Mechanismen zum Single-Log-Out etc. erweitert. Für Softwareentwickler besonders interessant sind die Anmerkungen zu Sicherheitsproblemen und Gefahren, die sich in den SAML Spezifikationen finden. Dort werden Angriffsmöglichkeiten auf Token bzw. Clients geschildert und Abwehrmaßnahmen diskutiert. Die sicherheitsrelevanten Teile finden sich vor allem in den Spezifikationen zu den Bindings. Es wird deutlich dass je nach Transport- und Kommunikationsprotokoll unterschiedliche Gefahren existieren.

Der auf die Sicherheitsaussagen bezogene Teil von SAML dürfte in Zukunft von zentraler Bedeutung für alle föderativen Ansätze sein. Ohne eine gemeinsame Sprache und Semantik zur Erstellung von Aussagen zu sicherheitsrelevanten Anforderungen oder Erwartungen ist keine sinnvolle B2B- oder C2B-Technologie

```

<saml:Assertion Version="2.0,, ID="_34234se72,, IssueInstant="2005-04-01T16:58:33.173Z">
  <saml:Issuer>http://authority.example.com/</saml:Issuer>
  <ds:Signature>...</ds:Signature> <!-- issuer signature -->
  <saml:Subject>
    <saml:NameID format="urn:oasis:names:tc:SAML:2,0:nameid-format:persistent">
      jygH5F90I
    </saml:NameID>
  </saml:Subject>
  <saml:AuthnStatement AuthnInstant="2005-04-01T16:57:30.000Z">
    <saml:AuthnContext>
      <saml:AuthnContextClassRef>
        urn:oasis:names:tc:SAML:2,0:ac:classes:PasswordProtectedTransport
      </saml:AuthnContextClassRef>
    </saml:AuthnContext>
  </saml:AuthnStatement>
</saml:Assertion>

```

Abb. 12.11 SAML Assertion (aus [Anton])

über mehrere Beteiligte hinweg denkbar. Es bleibt jedoch abzuwarten, wie weit sich SAML hier als Standard durchsetzen kann und wie offen für Interpretation sich die Definitionen erweisen.

12.4.2 Liberty Alliance

Ursprünglich als Konkurrenzprojekt zum zentralen Identitätsmanagement von Microsoft Passport gedacht, hat sich Sun's Liberty Alliance Projekt von Beginn an auf das Prinzip des föderativen Identitätsmanagements konzentriert. Man ging also von vornehmesten davon aus, dass es mehrere Identity Provider gibt, bzw. dass Firmen nicht gewillt sind, ihr eigenes Kundenmanagement komplett an Dritte auszulagern – eine Einschätzung, die sich als richtig erwiesen hat. Zentrale Bedeutung sollte dabei auch der Privatsphäre des Kunden zukommen, dem man nicht zumuten wollte, dass er alle seine privaten Informationen (Transaktionsdaten, Präferenzen) einem einzigen Identitätsprovider gegenüber offen legen sollte. Stattdessen sollte der Kunde in die Lage versetzt werden, nach Bedarf und ausschnittsweise Information gegenüber Service Providern offen zu legen bzw. wieder zu entziehen. Account Linking und De-linking standen daher im Mittelpunkt. Eng verbunden damit war natürlich das Problem, wie Accounts eines Kunden bei mehreren IPs und SPs so verbunden werden könnten, dass einerseits ein Single-Sign-On möglich wurde, andererseits die Beteiligten aber nicht dazu gezwungen wurden, eine einzige Identität zu verwenden, oder ihre Identitäten allen offen zu legen.

Das Resultat dieser Anforderung war die Einführung des Alias-Namens. Bei der Anmeldung bei einem Identity Provider erstellt der IP einen Alias-Namen für den Kunden, der Teil des Tokens bzw. der Assertion wird. Ein Service Provider, der diesen Namen erhält, kann ihn einerseits mit einem eigenen Kundennamen verlinken und andererseits mit Hilfe des Alias mit dem Identity Provider kommunizieren und so Informationen über den Kunden austauschen. Wichtig daran ist, dass der Service Provider den echten Benutzernamen, mit dem der Kunde beim Identity Provider angemeldet ist, nicht erfährt. Der Service Provider kann nun seinerseits einen Alias für den Kunden erstellen und diesen mit Identity Providern austauschen, so dass die Föderation von Identitätsinformation auch rückwärts vom SP zum IP funktionieren kann. Auch die Identity Provider untereinander kennen nur die Alias Namen der Kunden.

Identity Provider, die untereinander so zusammenarbeiten, bilden einen so genannten Circle of Trust, ohne dass die Zusammenarbeit einen gemeinsamen Usernamen (und damit einen globalen Namensraum mit eindeutigen Namen) erfordern würde, was sowohl technisch wie auch organisatorisch kaum machbar wäre. Dies haben letztlich die gescheiterten Versuch zum Aufbau globaler Namensräume (wie X.500 Directories) gezeigt. Die Liberty Alliance Spezifikationen haben einen engen Bezug zu SAML bekommen. Interessant sind daran ebenfalls die Anmerkungen zur Sicherheit in den Bindings. Auf Sicherheitsfragen in Bezug auf die webbasierte Föderation wird weiter unten noch eingegangen.

12.4.3 Web Services Federation

Dieser noch relativ neue Teil der Web Services Security Plattform baut auf den bekannten Web Services Security Spezifikationen wie WS-Security ([WS-SEC]) und WS-Trust auf. Web Services Federation ([WS-FED]) bietet zwei verschiedene Profile an: Aktiv und Passiv. Beim aktiven Profil wird von einem intelligenten, Web-services fähigen User Agent als Client ausgegangen, während das passive Profil mit einem typischen Browserszenario gleichzusetzen ist. In der Funktionalität ist die Spezifikation ähnlich denen von SAML und Liberty Alliance und ist in der Lage, SAML Token und Expressions zu verwenden.

Bei [Bueck] findet sich ein Beispiel für die Verwendung des aktiven Profiles. Ausgangspunkt ist ein Firmenportal mit dem Mitarbeiter verschiedene Produktivitätstools nutzen können (s. Abb. 12.12). Eine extra Anmeldung ist nicht nötig, da über SPNEGO die Systemanmeldung über den Browser zum Portal transportiert wird. Zum Buchen von Telefonkonferenzen verweist ein Link (FORMS Post) auf ein Portal eines Telekommunikationsanbieters. Alle Mitarbeiter dürfen dort Telefonkonferenzen buchen.

WS-Federation erlaubt nun – unter Verwendung einer SAML Assertion – die erfolgreiche Authentisierung innerhalb der Firma zum Provider fließen zu lassen. Dort werden alle Mitarbeiter zur Zugriffskontrolle auf einen künstlichen User (FirmaXY-Gast) gemappt. Zum Zweck des Auditing fliesst die E-Mail-Adresse des jeweiligen Users sowie der Name ebenfalls mit. Die Textdatei mit der dafür nötigen SAML Assertion findet sich online in [Bueck, S.212 ff]. Die Abbildung

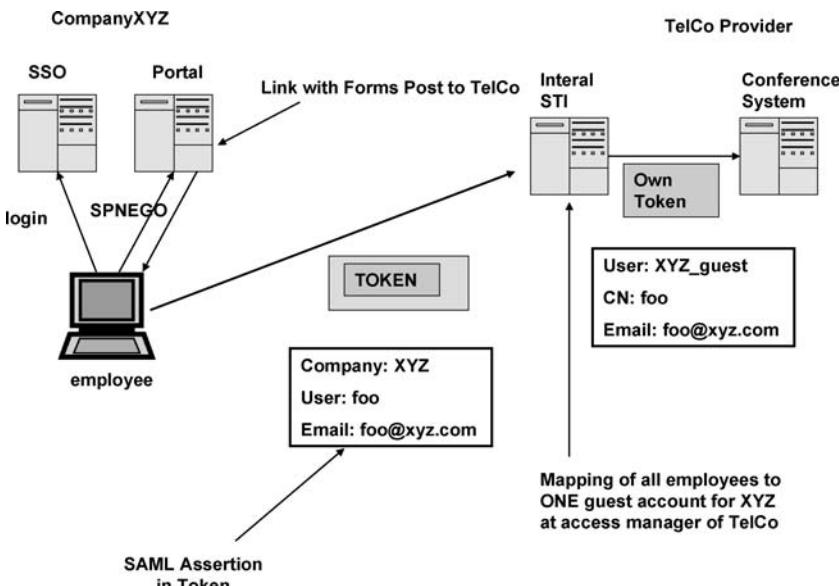


Abb. 12.12 Beispielszenario zur Verwendung des aktiven Profils der Web Services Federation

```

<saml:Assertion AssertionID=".... IssueInstant="2007-03-20T12:30:50Z"
Issuer="https://www.xyz.com/wsf" MajorVersion="1", MinorVersion="1", xmlns:ds="http://www.w3.org/2000/09/xmldsig#
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">

    <saml:Conditions NotBefore="2005-07-16T15:14:29Z", NotOnOrAfter="2005-07-16T15:34:29Z">
        <saml:AudienceRestrictionCondition>
            <saml:Audience>https://www.telco.com/wsf
            </saml:Audience>
        </saml:AudienceRestrictionCondition>
    </saml:Conditions>

    <saml:AuthenticationStatement AuthenticationInstant="2005-07-16T15:24:29Z"
AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
        <saml:Subject>
            <saml:NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
                emp1@xyz.com
            </saml:NameIdentifier>
        </saml:Subject>
    </saml:AuthenticationStatement>
    <saml:AttributeStatement>
        <saml:Subject>
            <saml:NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
                emp1@xyz.com
            </saml:NameIdentifier>
        </saml:Subject>
        <saml:Attribute AttributeName="cn", AttributeNamespace="http://www.xyz.com/cn">
            <saml:AttributeValue>Employee One </saml:AttributeValue>
        </saml:Attribute>
    </saml:AttributeStatement>
</saml:Assertion>

```

Abb. 12.13 Föderativer Teil der Assertion: Beschreibung der Rahmenbedingungen und des Subjekts

```

<ds:Signature Id="uuid203f1582-0105-efbb-6039-8ce3efd72411", xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#Assertion-uuid203f1557-0105-f23c-5b82-8ce3efd72411">
            <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                <c14n:InclusiveNamespaces PrefixList="saml ds", xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue> sWS4qUyQXSgMRHM62ADxLHGFD4=
            </ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue> signature over assertion (according to XMLDSig Spec.)...
    </ds:SignatureValue>
    <ds:KeyInfo>
        <ds:X509Data>
            <ds:X509Certificate> XYZ Corp. Certificate,.....
        </ds:X509Certificate>
        </ds:X509Data>
        </ds:KeyInfo>
    </ds:Signature>
</saml:Assertion>

```

Abb. 12.14 Nicht Föderativer Teil der Assertion: Sicherstellung der messagebasierten Security für die Assertion

gen 12.13 und 12.14 verwenden die originale Assertion in leicht abgewandelter und kommentierter Form.

Der erste Teil der Assertion enthält die inhaltlichen Aussagen über das authentisierte Subjekt und seine Eigenschaften. Außerdem werden die zentralen Eigenschaften (ID, Time) und Bedingungen (when, till) für die Gültigkeit der Assertion beschrieben. Attribute sind einfache Name/Value Paare, die einem Subjekt zugeordnet werden.

Im zweiten Teil der Assertion muss die Assertion selber gegen Fälschung gesichert werden. Dies geschieht durch eine digitale Signatur des Ausstellers in Form eines in XML-DSig definierten Signature-Elementes. Die Methoden der Erstellung der Signatur, ihr Hash-Wert sowie der Wert der Signatur selbst sind enthalten. Zum Schluss wird noch das Zertifikat mit dem öffentlichen Schlüssel des Signierers mitgeschickt damit der Empfänger – in diesem Fall die TelCo Firma – die Signatur überhaupt prüfen kann.

12.5 Sicherheitsanalyse

Föderative SSO Techniken unterscheiden sich unter Anderem durch folgende Kriterien:

- User Agent (Browser oder Fat client);
- Push oder Pull (Wird beim Request des Clients an den Service Provider bereits ein SSO Token mitgegeben?);
- Aktive oder passive IP Identifizierung im Pull Profile (Wie erkennt ein SP, welcher IP für einen Kunden zuständig ist? Durch ein passives Cookie oder eine dem Kunden präsentierte Auswahlliste?);
- Front Channel und Back Channel (Welche Informationen werden über den Client zwischen IP und SP kommuniziert? Welche Informationen tauschen SP und IP direkt aus?);
- Browser GET oder POST Profile (Bei GET wird nur ein kurzes Token übertragen vom IP zum SP, bei POST kann eine signierte Nachricht bereits Nutzdaten für den SP enthalten.);
- Message Protokoll (HTML Forms oder WS-Trust Requests und Responses);
- Message Syntax und Semantik (Wie werden die Informationen beschrieben? Mittels SAML oder Web Services Security Syntax?);
- Support für Global Sign-Off;
- Pre-Provisioning oder On-Demand Erzeugung von Userinformationen bei Service Providern.

Je nach Binding bzw. Profile entstehen dadurch unterschiedliche Sicherheitsrisiken zusätzlich zu den im Sicherheitsprotokoll bereits enthaltenen Risiken. Um diese zu untersuchen, ist es nützlich, uns noch einmal kurz vor Augen zu führen, was dabei tatsächlich passiert:

Wir haben Interaktionen zwischen mindestens drei Beteiligten, dem Kunden, dem Identity Provider und mindestens einem Service Provider. Es gibt also keine Ende-zu-Ende Sicherheit auf Basis eines einzigen sicheren Kanals (z.B. via SSL). Typischerweise existieren zwei verschiedene Sessions (Kunde–IP, Kunde–SP), wobei die Session Kunde–SP auf Grund einer signierten Aussage des Identity Providers aufgebaut wird. Diese signierte Aussage (Token) wird über den so genannten Front Channel über den Kunden und dessen User Agent zum Service Provider transportiert.

Der Service Provider prüft das Token und baut daraufhin eine unabhängige Session mit dem Kunden auf (beispielsweise über SSL). Auf Seiten des Service Providers werden dann alle Aktionen des Kunden gegen die Authorisierungsinformation beim Service Provider geprüft.

Für die Session des Kunden mit dem Service Provider ist also ausschließlich das Token verantwortlich. Der Kunde führt keine weitere Authentisierung gegenüber dem Service Provider durch (was ja auch das Konzept des SSO aufheben würde).

Sollte allerdings das Token des Kunden – zum Beispiel durch eine Attacke auf den User Agenten (Browser) des Kunden – an einen Angreifer gelangen, so könnte dieser ohne Mühe eine Session mit dem Service Provider aufbauen und Services im Namen des Kunden nutzen. Genauso kritisch ist natürlich der Transportweg des Tokens zum Kunden bzw. zum Service Provider. Beide Wege müssen durch SSL geschützt werden.

Sicherheitstechnisch bedeutet das, dass wir zwar einerseits ein kryptografisch abgesichertes Token haben, das vom Angreifer nicht verändert werden kann (aufgrund der Signatur des IP). Andererseits sind wir für die Kundensession mit dem Service Provider vollständig auf die Sicherheit der Implementation des User Agents (Gefährdung durch Plug-ins oder Aktive Componenten) wie auch auf die Sicherheit der Implementation des Service Providers (Gefährdung durch Cross Site Scripting) angewiesen. Der Service Provider wiederum hat keinerlei rechtliche Garantie im Sinne von einer Nicht-Abstreitbarkeit, das die Aufträge tatsächlich vom Kunden stammen. Die Aufträge (Transaktionen) sind ja nicht durch den Kunden signiert, es existiert lediglich eine authentisierte Session.

Damit zeigen sich auch die Grenzen des föderativen Identitätsmanagements: Letztlich führt es lediglich zum Aufbau einer sicheren, kanalbasierten Kommunikation. Die Sicherheit liegt in der Implementation, nicht in der Kryptografie, und bleibt hinter der Sicherheit, wie sie die Signatur von einzelnen Transaktionen bietet, weit zurück.

Damit kommen der Qualität und der Handhabung des Tokens eine große Rolle zu. Viel spricht in diesem Zusammenhang dafür, dass für jede Session des Kunden mit einem anderen Service Provider ein neues Token erstellt werden sollte (sonst könnte ein gestohlenes Token ja gegenüber mehreren Service Providern eingesetzt werden). Nach den schlechten Erfahrungen mit der Erzeugung sicherer SessionIDs (siehe die Untersuchung von David Endler [End]) muss auch sichergestellt werden, dass die generierten Token nicht einfach erratbar sind.

Einen klaren Vorteil hat föderatives Identitätsmanagement allerdings dann, wenn es sich herausstellt, dass die Dreierbeziehung Kunde, Identity Provider, Service Provider in Wirklichkeit eine Beziehung Mitarbeiter, Firma, Lieferant ist, das heißt in Wirklichkeit eine Zweierbeziehung Firma–Lieferant besteht. Hier ist es für den Service Provider (also den Lieferanten) von Vorteil, wenn die Firma die Identität des Mitarbeiters bestätigt, anstatt eine selbständige Authentisierung der Mitarbeiter durchzuführen, da das Vertrauensverhältnis in Wirklichkeit zwischen Firma und Lieferant besteht. Im Extremfall spielt die Identität des Mitarbeiters hier überhaupt keine Rolle, nämlich dann, wenn jede Bestellung selbst durch die Firma signiert wird. Dann wäre die Identität des Mitarbeiters lediglich noch für den Fall der missbräuchlichen Bestellung von Seiten des Mitarbeiters nötig.

Auch die Frage der Gültigkeitsdauer der Token ist recht interessant. Kritisch könnte man fragen, ob beispielsweise der Identity Provider eine Gültigkeitsdauer des Tokens angeben sollte oder nur das Erstellungsdatum des Token. Schließlich ist es ja eine Sache des Service Providers, wie lange er ein solches Token als gültig akzeptieren wird. Letztlich ist also die Frage, wie lange der Service Provider die auf Grund des Tokens erstellte Session als gültig erachten wird. Die einzelnen Services der Provider können dabei von sehr unterschiedlicher Dauer und Kritikabilität sein und die Bestimmung von Session Timeouts ist davon abhängig.

Die Funktionalität des Global Sign-Off wirft ebenfalls sicherheitsrelevante Fragen auf: Wer ist für die Durchführung eines Global Sign-Off verantwortlich? Geschieht der Sign-Off über den Front-Channel (per Redirection) oder über den Back Channel (durch direkte Interaktion IP-SP). Wie zuverlässig ist das und welche Garantien erhält der Client? Was bedeutet „Global Sign-off“ bei lange dauernden Transaktionen mit asynchronen Antworten?

Die Behandlung des „Where are you from“ Problems bei Liberty Alliance wirft ebenfalls einige kritische Fragen auf: Was kann ein Service Provider tun, wenn er einen Kundenrequest empfängt ohne gültiges Token eines Identity Providers? Im Allgemeinen wird der Service Provider nach Ermittlung des zuständigen IPs einen Redirect zum IP auslösen und auf die Antwort warten. Bei der Durchführung dieses Redirects übermittelt der Kunde im Zuge der Anmeldung seine Credentials an den Identity Provider. Dies ist für den Kunden ein sehr kritischer Moment, denn wenn er die Credentials an einen Angreifer gibt, kann dieser sich für beliebige Services des Kunden anmelden. Es ist also von entscheidender Bedeutung, dass der Kunde erkennen kann, bei wem er seine Credentials angibt. Gleichzeitig besteht natürlich ein Interesse auf Seiten des Service Providers, diesen globalen Anmeldevorgang so transparent und einfach wie möglich für den Kunden zu gestalten. Das bedeutet, die Anmeldung könnte in einem Frame oder IFrame des Service Provider Portals stattfinden. Es gibt sogar einen Vorschlag, dass ein Service Provider die Anmeldung eines Kunden beim Identity Provider im Auftrag durchführen könnte. In diesem Fall würde der Service Provider die Credentials des Kunden erfahren, was das ganze Prinzip des SSO auf den Kopf stellen würde, was die Vertraulichkeit der Credentials angeht.

Die Möglichkeit des Phishings der Credentials wäre dann gegeben, wenn auf Grund einer Cross-Site-Scripting Schwäche eines Service Providers der Kunde auf

eine Kopie der Anmeldungsseite eines offiziellen Identity Providers gelockt werden könnte. In diesem Licht betrachtet, scheint das PUSH Profil für die Übergabe von Authentisierungstokens sicherer zu sein.

Eine große Bedeutung besitzt natürlich die Qualität der Authentisierung, die beim Identity Provider durchgeführt wird. So ist das Beispiel der Weitergabe von Identität von einem Reiseportal an eine Bank kaum realistisch, da für das Buchen einer Reisen kaum eine starke Authentisierung (zum Beispiel mit PIN/TAN) verlangt werden kann, die Bank aber mit Sicherheit darauf bestehen wird.

Überhaupt scheint der Vorgang der Delegation über mehrere Service Provider hinweg bei gleichzeitiger Nutzung von verschiedenen Identitäten eines Kunden bei Firmen noch recht vage zu sein. Hier müssten Identity Mappings für mehrere Kundenidentitäten automatisch durchgeführt werden – etwas, das normalerweise von der Zustimmung des Kunden abhängt. Eine durch den Kunden zertifizierte Delegation mit Hilfe von Proxy – Zertifikaten könnte hier Abhilfe schaffen.

Kapitel 13

Schlussbetrachtungen

In diesem Band wird Sicherheit in erster Linie als Schutz vor Angriffen bzw. als Sicherung von Geschäftsmodellen interpretiert. Als wichtigste Sicherheitsdienste in diesem Zusammenhang haben wir Authentisierung und Autorisierung diskutiert mit den dazu gehörigen konkreten Mechanismen. Mit Hilfe dieser „klassischen“ Sicherheitstechniken lässt sich eine Infrastruktur für die Abwicklung von Geschäftsprozessen aufbauen, vor allem unter Verwendung von kanalbasierter Sicherheit: Die sichere Übertragung von Daten wird dabei durch SSL bzw. TLS garantiert. Rechner innerhalb einer Infrastruktur können sich gegenseitig mit Hilfe von SSL unter Verwendung von Zertifikaten authentisieren.

Aber die kanalbasierte Sicherheit hat auch Grenzen, die z. B. im Falle föderativer Geschäftsmodelle sichtbar werden: Organisationen und Firmen, die miteinander nur lose verbunden sind, müssen nun Geheimnisse miteinander austauschen und gegenseitiges Vertrauen aufbauen. Auch ohne den Austausch von Geheimnissen bereitet die kanalbasierte Sicherheit Probleme, z. B. bei der Delegation oder der Gewährleistung von Nicht-Abstreitbarkeit. Gerade die wichtige Funktionalität der Delegation hat sich als ein wirklich hartes Problem erwiesen: Wie kann aus der bloßen Tatsache einer erfolgten Authentisierung auf die Absichten der nun bekannten Person geschlossen werden? Welche Rechte genau sollen bei der Delegation weiter gegeben werden und für wie lange? Diese Fragen lassen sich auf sichere und fein-granulare Weise nur durch eine signierte Willenserklärung des Kunden klären, die in einer standardisierten Rechtebeschreibungssprache abgefasst ist.

Diese Überlegung hat uns zum Konzept der objektbasierten Sicherheit gebracht. Hier führt jeder Datensatz, jede Anfrage, eben jedes „Objekt“ seine eigene Absicherung durch kryptografische Mittel mit sich. Durch elektronische Signaturen können die Anforderungen des Integritätsschutzes und der Nicht-Abstreitbarkeit in natürlicher Weise erfüllt werden, selbst wenn die Objekte über unsichere Kanäle *und* unsichere Netzketten transportiert werden. Damit verlassen wir auch das herkömmliche Internet-Bedrohungsmoell und gehen über zu dem wesentlich umfassenderen (und für das heutige Internet relevanteren) Host-Bedrohungsmoell. Durch die Nutzung objektbasierter Sicherheit können Infrastrukturen vereinfacht und typische Attacken wie Phishing verhindert werden. Das

Plädoyer für die objektbasierte Sicherheit sollte eine der Kernaussagen dieses Bandes sein.

Wie geht es nun weiter? Im Nachfolgeband „Sichere Systeme“ werden wir das Spannungsfeld zwischen kanalbasierter Sicherheit und objektbasierter Sicherheit weiter untersuchen. Die Sicherheit von Web-Services, Grid-Computing, sichere Updates, etc. fallen unter diese Thematik.

In einem Punkt jedoch vollzieht der zweite Band einen wesentlichen Schwenk: Sicherheit wird dort nicht nur als Schutz vor Angriffen verstanden, sondern allgemeiner, nämlich als Methode zur Minimierung von Schaden. Dies erfordert die Beantwortung von sehr grundsätzlichen Fragen: Kann ein Modul X ein Modul Y beeinflussen? Welche Eigenschaften von Betriebssystemen und Sprachen erlauben sichere Aussagen zur kausalen Beeinflussung zwischen Softwareteilen? Wie lassen sich überhaupt Sicherheitsanalysen von Software durchführen, wenn grundsätzlich jedes Modul von jedem beeinflusst werden kann? Wie lässt sich andererseits eine Isolation von Komponenten erreichen?

Wir untersuchen dazu Kerntechniken der Isolation von Komponenten durch ACLs, Rollen, Capabilites etc. und betonen die Bedeutung der Software-Architektur für die Sicherheit von Systemen. Daneben werden Kenntnisse im Umgang mit populären Frameworks für sichere Software wie JAAS, Java2 Security, Spring, Grids, Webservices etc. vermittelt. Daneben spielt die Untersuchung von Attacken und ihren Ursachen/Voraussetzungen eine zentrale Rolle. Wir werden sehen, dass die Reduktion von Autorität auf das absolut Notwendigste (Principle of Least Authority – POLA) ein zentrales Element bei der Abwehr von Angriffen und der Eingrenzung möglicher Schäden darstellt. Dabei zeigt sich eine erstaunliche Verbindung: Autoritätsreduktion erlaubt auch grundsätzlich andere und bessere Formen von Usability: Statt unverständlichen Sicherheitswarnungen erhält der Nutzer klare Gesten zur Vermittlung von Autorität. Mit dieser vielleicht überraschenden, aber hoffnungsvollen Perspektive möchten wir diesen Band beenden.

Abkürzungsverzeichnis

ACL	Access Control List
ACM	Access Control Matrix
AES	Advanced Encryption Standard
API	Application Programming Interface
BBS	Benutzer-Berechtigungssystem
CA	Certificate Authority
CAS	Code Access Security
CBC	Cipher Block Chaining
CIL	Common Intermediate Language
CLR	Common Language Runtime
CMS	Content Management System
CORBA	Common Object Request Broker Architecture
CSI	Common Secure Interoperability
CUG	Closed User Group
DAC	Discretionary Access Control
DAMS	Digital Asset Management System
DES	Data Encryption Standard
DHE	Diffie-Hellman
DMZ	De-Militarized Zone
DNS	Domain Name Service
DoS	Denial of Service
DRM	Digital Rights Management
DSA	Digital Signature Algorithm
ECMS	Enterprise Content Management System
EJB	Enterprise Java Beans
FCP	Firewall Control Protocol
GAC	Global Assembly Cache
GIOP	Generic Inter-ORB Protocol
GSS	Generic Security Service
GUI	Graphical User Interface
HSM	Hardware Security Module
IDS	Intrusion Detection System

IOP	Inter-ORB Protocol
IIOP	Internet IOP
IP	1) Internet Protocol 2) Identity Provider
JAAS	Java Authentication and Authorization Service
LDAP	Lightweight Directory Access Protocol
MAC	1) Mandatory Access Control 2) Message Authentication Code
MLS	Multi-Level Security
OID	Object Identifier
ORB	Object Request Broker
PAC	Privilege Attribute Certificate
PAM	Pluggable Authentication Module
PIN	Personal Identification Number
PKI	Public Key Infrastructure
RACF	Resource Access Control Facility
RBAC	Role-Based Access Control
RMI	Remote Method Invocation
RP	Reverse Proxy
RSA	Rivest/Shamir/Aadleman
SAML	Security Assertion Markup Language
SAS	Security Attribute Service
SASL	Simple Authentication and Security Layer
SHA	Secure Hash Algorithm
SOAP	Simple Object Access Protocol
SPNEGO	Simple and Protected Negotiation Mechanism
SSL	Secure Sockets Layer
SSO	Single-Sign-On
STI	Security Token Issuer
TCP	Transmission Control Protocol
TGS	Ticket Granting Server
TGT	Ticket Granting Ticket
TLS	Transport Layer Security
TNC	Trusted Network Computing
TTP	Trusted Third Party
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VO	Virtual Organization
XML	Extended Markup Language

Literaturverzeichnis

- [AdNo] AdNovum Informatik AG, Informationsblatt Nevis Web Security,
http://www.adnovum.ch/pdf/info/AdNovum_NevisWeb_D.pdf
- [ALP] A. Alireza, U. Lang, M. Padelis, R. Schreiner, M. Schumacher: The Challenges of CORBA Security, in: M. Schumacher, R. Steinmetz (Hrsg.): Sicherheit in Netzen und Medienströmen, Springer 2000
- [Anton] E. Anton, Web Services in realen Business-Anwendungen – Sicherheit, Transaktionalität, Geschäftsprozess-Modellierung, Diplomarbeit HdM 2006,
<http://www.kriha.de/krihaorg/dload/uni/anton.pdf>
- [BelLa] D. Bell, L. LaPadula, Secure Computer Systems: Unified Exposition and Multics Interpretation, Technical Report MTR-2997 Rev. 1, MITRE Corporation 1975. Available online:
<http://csrc.nist.gov/publications/history/bell76.pdf>
- [Bish] M. Bishop, Introduction to Computer Security, Addison-Wesley 2005
- [Bless] R. Bless et al.: Sichere Netzwerkkommunikation: Grundlagen, Protokolle und Architekturen, Springer-Verlag, 2005
- [Botz06] K. Botzum, WebSphere Application Server V6 advanced security hardening – Part 1 and 2,
http://www-128.ibm.com/developerworks/websphere/techjournal/0512_botzum/0512_botzum1.html
- [BSI_FW] BSI Firewall Studie II,
<http://www.bsi.bund.de/literat/studien/firewall/fw01eng/fwstuden.htm>
- [Bueck] A. Buecker, W. Filip, H. Hinton, H. Hippensiel, M. Hollin, R. Neucom, S. Weeden, J. Westman, Federated Identity Management and Web Services Security, IBM Redbook 2005
- [Bueh] D. Buehler, Rollenbasierte Zugriffskontrolle in WebSphere Portal, in: P. Horster (Hrsg.): D-A-CH Security 2004, syssec-Verlag 2004
- [Butz] J. Butz, Sicherheitskonzepte für mobile Systeme in Firmen, Diplomarbeit HdM 2007,
<http://www.kriha.de/krihaorg/dload/butz.pdf>
- [Castro] N. Forsyth, E. Castro, A con as big as the Ritz (Teil 1 und 2).
www.guardian.co.uk/print/0,,329756625-103425,00.html
- [CCC] <http://www.ccc.de/t-hack/>
- [CG] L. F. Cranor, S. Garfinkel (Ed.): Security and Usability – Designing Secure Systems that People Can Use, O'Reilly, 2005
- [Cher] S. Chernov et al., Enabling Federated Search with Heterogeneous Search Engines,
<http://base.ub.uni-bielefeld.de/download/FedSearchReport.pdf>
- [CORBAsec] Object Management Group (OMG): Security Service Specification V1.8, March 2002, <http://www.omg.org/docs/formal/00-06-25.pdf>
- [Cred] R. Credle et al, IBM WebSphere Portal V4.1 Handbook Volume 1, IBM Redbook, January 2003
- [CSIV2] Object Management Group (OMG): Common Secure Interoperability V2 Specification,
<http://www.omg.org/docs/ptc/01-06-17.pdf>

- [Dobb] H. Dobbertin, The Status of MD5 After a Recent Attack, *CryptoBytes* Vol. 2, No. 2 (1996), also available at <ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf>
- [ebay] <http://pages.ebay.de/sicherheitsportal/mitteilungen.html>
- [Eck] C. Eckert, IT-Sicherheit, DeGruyter-Verlag 2006
- [End] D. Endler, SessionID Hacking, <http://www.idefense.com>
- [ERS] J. Eisenmann, A. Rauber, S. Simon, Single Sin On, Software-Projekt HdM 2003, http://www.kriha.de/krihaorg/dload/uni/eisenmann_rauber_simon.zip
- [Garman] J. Garman, Kerberos: The Definitive Guide, O'Reilly 2003
- [Gates] B. Gates: Security: Raising the Bar, Remarks at RSA Conference 2005, <http://www.microsoft.com/billgates/speeches/2005/02-15RSA05.asp>
- [Geer] D. E. Geer, The Shrinking Perimeter. Making the Case for Data-Level Risk Management, <http://www.verdasys.com>
- [heise50569] <http://www.heise.de/security/news/meldung/50569>
- [heise83744] <http://www.heise.de/newsticker/meldung/83744>
- [heise83767] Polizei stellt Vernehmungsprotokolle versehentlich ins Internet, www.heise.de/newsticker/meldung/83767
- [Hook] D. Hook: Beginning Cryptography with Java, Wiley 2005.
- [Husey] S. Huseby: Sicherheitsrisiko Web – Anwendung, dpunkt-Verlag 2004
- [Klein] T. Klein, Zecke, *Linux Magazin* Nr. 4 (2005), S. 6 ff.
- [KleinA] A. Klein, The Insecure Indexing Vulnerability, Attacks Against Local Search Engines, <http://www.webappsec.org/projects/articles/022805-clean.html>
- [Knecht] T. Knecht, Underground Economy, Stream des Vortrags vom Security Day 2007 an der HdM Stuttgart (über die Autoren erhältlich)
- [Knud] J. Knudsen: Java Cryptography, Wiley 1998
- [Koss] A. Kossel: ebay-Passwortklau, <http://www.heise.de/security/artikel/54271>
- [Kretz] O. Kretzschmar, Roland Dreyer, Medien-Datenbank- und Medien-Logistik-Systeme. Anforderungen und praktischer Einsatz, Oldenbourg-Verlag 2004
- [Krü] G. Krüger: Handbuch der Java-Programmierung, Addison-Wesley 2002
- [LL] A. Laurie, B. Laurie: Serious flaws in bluetooth security lead to disclosure of personal data, <http://www.thebunker.net/security/bluetooth.htm>
- [Mit] K. D. Mitnick, W. L. Simon: The Art of Deception, John Wiley 2002
- [Morv] P. Morville, Ambient Findability. What We Find Changes Who We Become, O'Reilly 2005
- [MOV] A. J. Menezes, P. van Oorschot, S. Vanstone: Handbook of Applied Cryptography, CRC Press 1997
- [Pet] R. Peteanu, Best Practices for Secure Development, <http://members.rogers.com/razvan.peteanu>
- [Rayns] C. Rayns (Hrsg.), Multilevel Security And DB2 Row-level Security Revealed, IBM Redbook 2005
- [Rescorla] E. Rescorla, SSL and TLS: Designing and Building Secure Systems, Addison-Wesley 2000
- [RFC 1510] J. Kohl, C. Neuman: The Kerberos Network Authentication Service (V5), <http://www.ietf.org/rfc/rfc1510.txt>
- [RFC 1750] D. Eastlake, S. Crocker, J. Schiller, Randomness Recommendations for Security, <http://www.ietf.org/rfc/rfc1750.txt>
- [RFC 1964] J. Linn, The Kerberos Version 5 GSS-API Mechanism, <http://www.ietf.org/rfc/rfc1964.txt>
- [RFC 2025] C. Adams, The Simple Public-Key GSS-API Mechanism (SPKM), <http://www.ietf.org/rfc/rfc2025.txt>
- [RFC 2104] H. Krawczyk, M. Bellare, R. Canetti, HMAC: Keyed-Hashing for Message Authentication, <http://www.ietf.org/rfc/rfc2104.txt>
- [RFC 2246] T. Dierks, C. Allen, The TLS Protocol, Version 1.0, <http://www.ietf.org/rfc/rfc2246.txt>
- [RFC 2478] E. Baize, D. Pinkas, The Simple and Protected GSS-API Negotiation Mechanism, <http://www.ietf.org/rfc/rfc2478.txt>

- [RFC 2560] M. Myers et al, X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP, <http://www.ietf.org/rfc/rfc2560.txt>
- [RFC 2617] J. Franks et al., HTTP Authentication: Basic and Digest Access Authentication, <http://www.ietf.org/rfc/rfc2617.txt>
- [RFC 2712] A. Medvinsky, M. Hur, Addition of Kerberos Cipher Suites to Transport Layer Security (TLS), <http://www.ietf.org/rfc/rfc2712.txt>
- [RFC 2743] J. Linn, Generic Security Service Application Program Interface Version 2, Update 1, <http://www.ietf.org/rfc/rfc2743.txt>
- [RFC 4120] C. Neuman et al., The Kerberos Network Authentication Service (V5), <http://www.ietf.org/rfc/rfc4120.txt>
- [RFC2222bis-15] A. Melnikov, K. Zeilenga, Simple Authentication and Security Layer (SASL), <http://www.ietf.org/internet-drafts/draft-ietf-sasl-rfc2222bis-15.txt>
- [SAML] Security Assertion Markup Language(SAML) 2.0: Technical Overview. 2006, <http://www.oasis-open.org/committees/download.php/20645/ssc-saml-tech-overview-2%200-draft-10.pdf>
- [Schä] G. Schäfer: Netzwerksicherheit, Algorithmische Grundlagen und Protokolle, dpunkt-Verlag 2003
- [Schm] R. Schmitz, R. Kiefer et al., Kompendium Medieninformatik – Mediennetze, Springer-Verlag 2006
- [Schn01] B. Schneier: Secrets & Lies, IT-Sicherheit in einer vernetzten Welt, dpunkt-Verlag 2001
- [Schn04] B. Schneier, Practical Cryptography, Wiley 2004
- [Schn06] B. Schneier: Risk of Loosing Portable Devices, CRYPTO – GRAM Februar 2006, <http://www.schneier.com/crypto-gram-0602.html>
- [Schn07] B. Schneier, The Psychology of Security, Draft February 28, 2007, <http://www.schneier.com/essay-155.pdf>
- [Schn96] B. Schneier, Applied Cryptography, Second Edition, Wiley 1996
- [Schn99] B. Schneier, Attack Trees – Modelling Security Threats <http://www.schneier.com/paper-attacktrees-ddj-ft.html>
- [Schw] J. Schwenk: Sicherheit und Kryptographie im Internet, Vieweg-Verlag 2002
- [SESAME] SESAME Homepage, <https://www.cosic.esat.kuleuven.ac.be/sesame>
- [SigG] Gesetz über Rahmenbedingungen für elektronische Signaturen (Signaturgesetz), <http://www.bundesnetzagentur.de/media/archive/2247.pdf>
- [SOAP] SOAP Version 1.2 Part 1: Messaging Framework, <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>
- [SOAP-Sig] SOAP Security Extensions: Digital Signatures, <http://www.w3.org/TR/SOAP-dsig>
- [SOX] Sarbanes-Oxley Act 2002, <http://www.sarbanes-oxley.com/section.php>
- [Spie] B. Spiegel, Die obere Hälfte des Motorrads – Über die Einheit von Mensch und Maschine, Motorbuch-Verlag 2006
- [Viega] J. Viega, M. Messier, P. Chandra, Network Security with OpenSSL, O'Reilly 2002
- [Vivi] Vivisimo, Restricted Access: Entering the world of secure search. White Paper, <http://vivisimo.com/html/download-security>
- [Wind] P. J. Windley, Digital Identity – unmasking Identity Management Architecture (IMA), O'Reilly 2005
- [WS-FED] H. Lockhart et al., Web Services Federation Language Version 1.1 (2006), <http://www.ibm.com/developerworks/library/specification/ws-fed/>
- [WS-SEC] B. Atkinson et al.: Web Services Security (WS-Security), <http://www.ibm.com/developerworks/library/ws-secure>
- [WT] A. Whitten, J.D. Tygar: Why Johnny Can't Encrypt, in: L. F. Cranor, S. Garfinkel (Ed.): Security and Usability – Designing Secure Systems that People Can Use, O'Reilly, 2005, S. 669–692
- [WYY] X. Wang, Y. Yin, H. Yu, Finding Collisions in the Full SHA-1, Proc. Advances in Cryptology – Crypto 05, Springer-Verlag, 2005, also available at <http://www.infosec.sdu.edu.cn/paper/sha1-crypto-auth-new-2-yao.pdf>
- [X.509] ISO/IEC 9594-8, Information Technology – Open Systems Interconnection – The Directory: Authentication Framework, CCITT/ITU Recommendation X.509, 1993

- [XML] D. Beech, N. Mendelsohn, M. Maloney, H. S. Thompson: XML Schema, Part 1,
<http://www.w3.org/TR/xmlschema-1/>
- [XML-Enc] D. Eastlake, J. Reagle: XML Encryption Syntax and Processing, <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/>
- [XML-Sig] D. Eastlake, J. Reagle, D. Solo: XML-Signature Syntax and Processing,
<http://www.ietf.org/rfc/rfc3275.txt>
- [ZCC] E. D. Zwicky, S. Cooper, B. Chapman: Building Internet Firewalls
- [Zhu] L. Zhu, B. Tung, Public Key Cryptography for Initial Authentication in Kerberos,
<http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-init-33.txt>

Index

- .NET 171
 - Code Access Security (CAS) 172
 - Policy Levels 173
 - Role Based Security (RAS) 174
 - Security 171
 - Stack Walk 174
 - StrongName 172
 - Unmanaged Code 174
 - A
 - ACL *siehe* Access Control List
 - ACM *siehe* Access Control Matrix
 - AES *siehe* Advanced Encryption Standard
 - Absicherung der Infrastruktur 226
 - Access Control 193
 - dynamisch 193
 - Access Control List 73, 187, 218
 - Access Control Matrix 74
 - Administration von Applikationen 18, 233
 - Administrationszone 234
 - Advanced Encryption Standard 79, 80
 - Analyse des Ausgangsszenarios 55
 - Angriffsfläche 248
 - Designfehler 252
 - Internet 248
 - Intranet 250
 - Application-level-proxy 249
 - Applikationsarchitektur und Rechte 198
 - Applikationsdesign für Firewall-Umgebungen 241
 - Archetypen von Sicherheitsproblemen 59
 - Asymmetrische Algorithmen 85
 - Asymmetrische Kryptografie 102
 - Attack-Tree 37
 - Attacken, semantische 24, 44
 - Attributzertifikate 72, 92
 - Aufbau von SSL 140
 - Authentication 155
 - Frameworks 155
 - Step-Up 270
 - unterschiedliche Nutzergruppen 208
 - Authentifikation 19, 65
 - durch Passwörter 66
 - Authentifikationsprotokolle nach X509 95
 - Authentisierung 19, 65
 - in verteilten Systemen 107
 - mit Passwörtern 109
 - versus Identifizierung 107
 - Komponenten und Knoten 238
 - Authorisierung 72, 198
 - zentrale 198
 - Sonderfälle u. Probleme 204
 - Authorization Identity 161
 - Authorization Token, CORBA 167–170
 - Autorisierungsvektor 245
 - Autor vs. Sender 121
 - Autorität 151
 - Autoritäten 93, 110, 263
- B**
 - BBS *siehe* Benutzer-Berechtigungs-Systeme
 - bahn.de Fallstudie 9
 - Basic Authentication 128
 - Basisarchitektur 40
 - Basisprotokolle 127
 - Bastion Host 227
 - Bedrohungsmodelle 39, 42, 220
 - für bahn.de 44
 - Benutzer-Berechtigungs-Systeme (BBS) 196
 - Berechtigungen 197
 - Funktion vs. Daten 197
 - Organisatorische Änderungen 197
 - Applikationsarchitektur 197

Betriebsmodi für Blockchiffren 80
 Bindings (Protokoll) 290
 Blockchiffren 80
 Browser, Bedrohungen 48
 Buffer Overflows 24, 45, 230

C

CMS *siehe* Content Management Systeme
 CRL *siehe* Certificate Revocation List
 CBC *siehe* Cipher Block Chaining
 Certificate Authority 89, 90
 Certificate Revocation List (CRL) 94
 Challenge-Response-Verfahren 68
 Challenges 99
 Ciphersuites 145
 Ciphersuite Rollback Attack, SSL 144
 Compliance, SOX 180
 Confused Deputy Problem, Stellvertretung 206
 Content Management Systeme 179
 Architektur 181
 Berechtigungen 188
 Infrastruktur 182
 Sicherheitsarchitektur 185
 Sicherheitsmodell 188
 Trends 180
 Content Security 210
 Content-Level Security 179
 Cookie (*siehe* Sessions) 245, 246, 258
 CORBA 165
 Credential 31
 Vault 268
 forwarding 267
 Sicherheit 31, 66, 151
 Cross-Domain-Authentisierung 136
 Aliasnamen 289
 Cross-Site-Scripting 31, 50, 298
 CSIV2 und SAS 167
 Cipher Block Chaining 80
 Certificate Revocation List 94

D

DAC *siehe* Discretionary Access Control
 DES *siehe* Data Encryption Standard
 DMZ *siehe* Demilitarized Zone
 DRM *siehe* Digital Rights Management
 Data Encryption Standard 79
 Data-Tagging (Klassifizierung) 184
 Deep-Linking 253
 Delegation 69, 120, 123
 der Identität im Portal 261
 in GSS-API 160

mit Kerberos 134
 und Impersonation 120
 von Aufträgen 123
 traced 267
 Demilitarized Zone 226–230
 Denial-of-Service Attacke 70
 Deployment, Sicherheitsfragen 224
 Dictionary-Attack 67, 123
 Diffie-Hellman-Protokoll 86
 Digest Authentication 128
 Digital Rights Management 212
 Digitale Signaturen 88
 Digitale Wasserzeichen 212
 Discretionary Access Control 72, 213
 Diskrete Logarithmen 86
 Distributed Cache 245
 Distributed Trusted Computing Base 105
 DoS *siehe* Denial-of-Service Attacke
 Domain, Rechtezone 275

E

ECB-Mode *siehe* Electronic Codebook Mode
 EPRL *siehe* End-Entity Public Key Certificate Revocation List
 Electronic Codebook Mode 80
 ebay Fallstudie 24
 Elliptische Kurven 87
 End-Entity Public Key Certificate Revocation List 94
 Erzwungener Protokollwechsel im Firewall 229

F

FCRA *siehe* First-Cut-Risk-Analysis
 Faktorisierungsproblem 85
 Filetransfer, DMZ 241
 Firewall-Architekturen 227
 Firewalls (*siehe* auch DMZ) 226, 241
 Firewall-Control-Protocol 229
 Firmenportal 266, 267
 First-Cut-Risk-Analysis 36
 Föderative Sicherheit 275
 Organisation, Kreditkarte 278
 im Portal 281
 Analyse 296
 Föderatives Identitätsmanagement 276, 290
 Standards 290
 Vorteile 280
 Föderatives Trust Management 278
 Functional User 109, 114, 205

G

Geburtstags-Paradoxon 83
Gegenseitige Authentisierung 238
Generic Security Service Application Programming Interface 155
Geschäftsmodell 11, 25
Geschäftsprozesse 203
Geschäftsvorgänge 13, 14
Global-Sign-Off, Problem 298
GSS-API *siehe* Generic Security Service Application Programming Interface
GSS-API Message Flow 157
GSS-API Mechanismen 158
GSS-API und Kerberos 159

H

HSM *siehe* Hardware-Security-Module
Hardware-Security-Module 58, 117, 151
Hashfunktionen 81, 84
Hashfunktionen und MACs 102
Heuristiken für Softwareentwickler 265
Holder-of-Key 125
Host-BedrohungsmodeLL 43
Http Authentication 127

I

Identity 161
 Authentication 161
 Authorization 161
 Provider 278
 Token 265
Identitätsproblem des Zertifikats 92
Identität, verschiedene 281
Identitätstoken 265
Impersonation 69
Impersonifizieren 66, 123–124
Infrastruktur 225
 mobile 271
Initialisierungsvektor 81
Integration Legacy Systeme 267
Integritätsschutz 70
Intermediate, SOAP 176
Intermediate vs. Originator 121
Internet-BedrohungsmodeLL 43, 58, 152
Internet-Publishing Architektur 185
Intranet-BedrohungsmodeLL 43, 59

J

JCA *siehe* Java Cryptography Architecture
Java.crypto.* Klassen 100, 102

Java.security.* Klassen 100, 103
Java Cryptography Architecture (JCA) 100
Java Kryptomodule 104

K

Kanalbasierte Sicherheit 106, 152, 259
 SSL 259
Kerberos 129
 Attacken auf 133
 Delegation 134
 Funktionsweise 129
 Erweiterungen 137
 und Windows 137
Key Agreement, TTP 127
Key-File 239
Key-Store 239
Kollisionen 81
Kollisionsresistenz 83
Kommunikation 49
 mit Kunden 50
 Datenbank-gestützt 52
Kontext-Weitergabe 69
Kreditkarte 278
Kryptografie mit Java 99
Kryptografische Algorithmen 77
Kundensicht 12, 26

L

Labels, Security 214, 215
Legacy Systeme, Integration von 267
Liberty Alliance 293
Lokale Sicherheit 106

M

MAC *siehe* Mandatory Access Control
MAC *siehe* Message Authentication Code
MLS *siehe* Multi-Level Security
Mandatory Access Control 73
Mandatory Data Classification 34
Mandatory Data Ownership 34
Masquerading 121
Master Secret, Speicherung 151
Meet-in-the-Middle Attacke, SSL 146
Message Authentication Code 82
Microsoft Passport 138
Middleware Security 165
Mobile Dokumente 211
Mobile Infrastruktur 271
Mobile Geräte 272
Mobile Security 271–274
Multi-Level Security (MLS) 75, 213

N

Nachrichtenbasierte Sicherheit (*siehe auch*
Objektbasierte Sicherheit) 106
Name vs. Principal 108
Need-to-know Prinzip und BBS 196
Need-to-do Prinzip und BBS 196
Nevis Web 234
Nicht-Abstreichbarkeit 54, 71, 152
Non-Repudiation *siehe*
 Nicht-Abstreichbarkeit
Nonce (Number used once) 128

O

OBSOC *siehe* Online Business Solution
 Operation Center
OCSP *siehe* Online Certificate Status
 Protocol
Objektbasierte Sicherheit 15, 106
Online Business Solution Operation
 Center – Sicherheitslöcher 1
One-Pass Authentication 95
One-Way-Eigenschaft 83
Online Certificate Status Protocol (OCSP) 94
Originator 125
Originator vs. Intermediate 121

P

PAC *siehe* Privilege Attribute Certificate
PBE *siehe* Password-based Encryption
PNRG *siehe* Pseudozufallsgenerator
POLA *siehe* Principle-Of-Least-Authority
PayPal 31
Passphrase 110
Password-Based Encryption (PBE) 84
Passwörter 109
 Sicherheitsrelevante Eigenschaften
 110–113
 Verwaltungskosten 113
People Problem 61
Perfect Forward Secrecy, 147
Performance-Fragen 89, 148, 196
Perimeter-Security 182, 225
Phishing 15, 32, 63
Plattform-Bedrohungsmodell 54, 55
Plattform-Sicherheit 223
Point-of-Contact 231
Portal Access Control 192
Portal 9
 SSO 262
 Komponenten 42
 Sicherheitsfragen 13

Pseudozufallsgenerator (PNRG) 97
Preauthentication Data (Kerberos) 134
PreMasterSecret, SSL 143
Principal 69
Principals und Domänen 132
Principle-Of-Least-Authority (POLA)
 53, 125
Privilege Attribute Certificate (PAC)
 93, 267
Privilege Elevation 206
Programmer-Conceptual-Model 254
Propagation von Metadaten 263

R

RBAC *siehe* Role Based Access Control
RMI *siehe* Remote Method Invocation
Realm 275
Records Management 210
Reduktion von Autorität 251
Registry 109, 118, 269, 276
Remote Method Invocation (RMI) 170
Replay-Attack 68
Reverse Proxy 231–237, 257
Risikoanalyse der Geschäftsvorgänge 35
Risiko 62
 Einschätzung 62
 Integral 64
 Priorisierung 37
Role Based Access Control 74
 Hierarchisch 187
Rollenmodellierung 189, 192
Rollenwechsel 188
RSA-Verfahren 85

S

S/MIME 51
SAML *siehe* Security-Association-Markup-
 Language
SASL *siehe* Simple Authentication and
 Security Layer
SECIOP 166
SESAME *siehe* Secure European System for
 Applications in a Multi-vendor Env.
SOAP Aufbau 174
SOAP Security 176
SPKM *siehe* Simple Public Key Mechanism
SPNEGO *siehe* Simple and Protected
 Negotiation Mechanism
SQL Injection 24, 223, 234
SSL 139
 Aufbau 140
 Handshake 142

- Security 150
 Grenzen 150, 152
- SSL/TLS 139
- SSLIOP 166
- SSO *siehe* Single-Sign-On
- Salt, Salting 67
- Schlüssel 78
- Schlüsselabhängige Hashfunktionen 83
- Schlüsselaustausch-Protokoll 86
- Search Engine Security 216–222
- Secure European System for Applications in a Multi-vendor Env. 137
- Secure-Associations-Markup-Language 136, 289, 291
- Security Context 39, 264
- Diagramm 39, 220
 Ausbau 270
- Security Policies 33
- Security Token Issuer *siehe auch* Trusted-Third-Party
- Seed 98
- Sender vs. Autor 121
- Server und Knoten 238
- Service Management Delegation 190–191
- SessionIDs 99, 244
- Sessionkonzept 242
- Sessions 108, 242
- Wiederaufnahme 149
 Key 99, 131
 Sticky 245
 Timeouts 45, 246, 247
 unterschiedliche Timeouts 247
- Sicherheit auf Clientseite 45, 47
- Sicherheitsanalyse 53
- Szenario 53, 265
 Beispiel 52, 219, 296
 Föderatives SSO, 296
- Sicherheitsanforderungen 14, 29, 221, 222
- Search Engine 221, 222
- Sicherheitsdienste 65
- Sicherheitsframework, Firma 203
- Sicherheitskonzepte und Analysen 33
- Sicherheitsmassnahmen, Einschätzung 63
- Sicherheitszonen 41
- Sichtbarkeit von Servern am Internet 256
- Simple Authentication and Security Layer 161
- Simple and Protected Negotiation Mechanism 159
- Simple Object Access Protocol (SOAP) 175
- Single-Point-Of-Failure (Kerberos) 134
- Single-Sign-On 19, 116, 270
- für Portale 61, 262
 Fehler 119
- Software-Architektur der Authentisierung 115
- Simple Public Key Mechanism 159
- Spezialapplikationen, Bedrohungsmodell 45
- Spider Mechanismus, CMS 183
- State, halten von 243
- Stellvertreter, Proxy 121
- Symmetrische Verschlüsselung 101
- Symmetrische Verschlüsselungsalgorithmen 79
- Synchronisierte Registries 269
- System Context 38
- System Context Diagramm 38
- T**
- TGT *siehe* Ticket-Granting-Ticket
- Testkonzept 24
- Ticket-Granting-Ticket 130
- Ticket-Granting-Server 131
- Three Pass Mutual Authentication 97
- Three-Pass Authentication 96
- Token, sichere 237
- Trichter, Angriffsfläche 248
- Transformationen, räumlich moralisch 60
- Trust Management, föderativ 278
- Trust-Store 239
- Trusted-Computing-Base 249
- Trusted-Third-Party 30, 90, 110, 262, 283
- U**
- Ursachen von Sicherheitsproblemen, 2
- Usability, 3
- User-Bedrohungsmodell 43
- User-Conceptual-Model 63
- V**
- Verantwortung des Servers 49
- Übertragung 49
- Cookies 50
- Mail 51
- Verbindungsgraph, Webanwendung 41
- Verfügbarkeit 71
- Verkleinerung der Angriffsfläche 248
- Verteilte Sicherheit 106
- Verteilte Systeme 105
- Vertrauen 29
- Kunde, Anbieter 29
- Delegation 121
- föderativ 278
- Identität 129

Vertraulichkeit 70
Vier-Augen-Prinzip, Geschäftsprozesse 204
Virtuelle Organisation 226, 275
Vollmacht, Proxy 121–123
Vorgehensweise bei Risikoanalyse 34
Vulnerability Analysis 265

W

WAYF *siehe* Where-are-you-from
WS-Federation, Beispiel 294
WS-Trust Service 283
Wahrnehmungsproblem 2
Web Services Federation 294
Web-SSO zwischen Domänen 283, 284
Web-SSO, Implementation 285
Web-SSO, Push vs. Pull 288
Weitergabe der Identität *siehe* Delegation
Where-are-you-from, Identity Provider 286

Wörterbuch-Attacke 67, 145
Workflow und Realität 203

X

XSS *siehe* Cross-Site-Scripting

Z

Zahlungsmethoden 30
Zeitstempel, Problem 134
Zertifikate 89, 103
 Ablauf 240
 Erstellung 89
 nach X509 90
 Zurückziehen 93
Zonenkonzept 41
Zufallswerte 98
Zufallszahlen, Erzeugung 103
Zugriffskontrolle 207