
Entwurf und Implementierung eines Authentifikations-Proxys für das World Wide Web

Diplomarbeit

von

Thilo-Alexander Ginkel

Betreut von Tobias Straub



Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Theoretische Informatik
Prof. Dr. Johannes Buchmann

Juli 2004

Vorwort

„Sesam, öffne dich!“

Ali Baba im Märchen „Die Geschichte des Ali Baba und der vierzig Räuber“

Bereits Ali Baba war in einem der Märchen aus tausendundeiner Nacht regelmäßiger Nutzer eines Authentifikationssystems. Bei ihm war es eine Tür, die sich nur mit Kenntnis des oben zitierten Ausspruchs öffnen ließ. Wer wie Ali Babas Bruder Casim das zum Öffnen nötige Geheimnis vergaß, dem blieb die Tür verschlossen.

Damit beschreibt der Autor dieses Märchens ohne wissenschaftlichen Hintergrund ein auch heute weit verbreitetes Verfahren zur Legitimation: Authentifikation durch Wissen. Diese Legitimation erfolgt heute zwar nur selten durch Sprache, sondern in Form von Kennworteingaben oder Zertifikaten (Authentifikation durch Besitz), aber gerade im World Wide Web nimmt die Authentifikation zum Zugriff auf geschützte Dienste nicht zuletzt mit der fast schon exponentiell steigenden Verbreitung von E-Commerce, Webmail, Internet Banking sowie anderer webbasierter Anwendungen stetig zu. Selbst bei reinen Content-Providern geht der Trend dahin, dass Informationen erst nach vorheriger Anmeldung zugänglich sind.

Was bei der Nutzung einzelner Dienste für den Endbenutzer noch relativ problemlos erscheint, führt bei einer wachsenden Zahl von Diensteanbietern schnell zu Problemen: Entweder nutzt der Anwender für viele Dienste dasselbe Kennwort oder er geht dazu über, Kennwörter niederzuschreiben, damit es ihm nicht wie Casim im oben zitierten Märchen ergeht.

Schon im kleinen Rahmen der Einzelplatznutzung eines typischen Endbenutzer-Szenarios treten also offenbar Schwierigkeiten auf, die sich aber zumindest teilweise durch die Kennwortverwaltungsmechanismen moderner Webbrowser eindämmen lassen. Im Mehrbenutzereinsatz tun sich dagegen völlig neue Problemfelder auf. Diese Problemfelder sollen in dieser Arbeit herausgearbeitet und in einem zweiten Schritt mittels einer entsprechenden Authentifikationslösung angegangen werden.

Mein besonderer Dank gilt Tobias Straub für die erstklassige Betreuung bei der Erstellung dieser Arbeit. Prof. Johannes Buchmann danke ich dafür, dass er zuerst mein Interesse an und kurze Zeit später meine Begeisterung für die Kryptografie geweckt hat. Schließlich gilt mein Dank noch Dr. Alwine Edelmann-Ginkel und Jutta Netzel, die sich die Mühe gemacht haben, diese Arbeit Korrektur zu lesen, und damit einen entscheidenden Beitrag zur hoffentlich niedrigen Fehlerquote geleistet haben.

Darmstadt, im Juli 2004

Thilo-Alexander Ginkel

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 20. Juli 2004

Thilo-Alexander Ginkel

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.1.1	Szenario	1
1.1.2	Ziel dieser Arbeit	2
1.2	Gliederung	2
2	Grundlagen	4
2.1	Authentifikation	4
2.1.1	Definition	4
2.1.2	Credentials	6
2.1.3	Delegation	6
2.1.4	Gruppennutzung	6
2.2	Authentifikationsmechanismen im WWW	7
2.2.1	Basic- und Digest Authentication	7
2.2.1.1	Basic Authentication	8
2.2.1.2	Digest Authentication	9
2.2.2	TLS Client Authentication	11
2.2.3	Formularbasierte Ansätze	11
2.3	Transport Layer Security	13
2.3.1	Übersicht	14
2.3.2	Kryptografische Verfahren	15
2.3.2.1	Verschlüsselung	15
2.3.2.2	Authentifikation	15
2.3.2.3	Schlüsselaustausch	16
2.3.2.4	Integritätssicherung	17
2.3.3	TLS Record Protocol	17
2.3.3.1	Fragmentation Layer	18
2.3.3.2	Record Compression and Decompression Layer	18
2.3.3.3	Record Payload Protection Layer	19
2.3.4	TLS Handshake Protocol	20
2.3.4.1	Handshake Protocol	20
2.3.4.2	Change Cipher Spec Protocol	24
2.3.4.3	Alert Protocol	24
2.3.5	Anmerkungen	25
2.4	Tunneln von TLS-Verbindungen über HTTP-Proxy-Server	25

3	Entwurf	27
3.1	Einführung in die Problematik	27
3.2	Anforderungsanalyse	27
3.3	Architekturvarianten	28
3.3.1	Stellvertreter-Architektur	28
3.3.1.1	HTTP-Gateway	29
3.3.1.2	HTTP-Proxy	31
3.3.1.3	Application-Server	34
3.3.2	Clientbasierte Architektur	35
3.4	Vergleich und Bewertung	36
3.4.1	Bewertungskriterien	36
3.4.2	Kompatibilität	36
3.4.3	Integration	37
3.4.4	Benutzbarkeit	38
3.4.5	Sicherheit	39
3.4.6	Deployment	41
3.4.7	Realisierungsaufwand	42
3.4.8	Zusammenfassende Gegenüberstellung	43
3.5	Resümee	43
4	Implementierung	44
4.1	Modulare Architektur	44
4.2	Networking	46
4.2.1	Listener	46
4.2.2	Request-Verarbeitung	46
4.2.3	Method-Handler	46
4.2.4	HTTP-Engine	47
4.2.5	Proxy-Auto-Configuration	47
4.3	Authentifikationsarchitektur	49
4.3.1	Public-Key-Verfahren	49
4.3.2	Konventionelle HTTP-Authentifikation	49
4.4	Backend	51
4.4.1	Certificate-Authority	51
4.4.2	Credential-Store	51
4.4.3	Credentials	52
4.4.3.1	HTTP-Kennwort-Credentials	52
4.4.3.2	X.509-Credentials	53
4.4.3.3	Formularbasierte Credentials	53
4.4.3.4	Admin-Credential	54
4.4.3.5	Serialisierung	54
4.4.4	Benutzerverwaltung	55
4.4.5	Benutzerauthentifikation	55
4.4.6	Zugriffskontrolle	56
4.4.7	Konfigurationsverwaltung	57

4.4.8	Logging	57
4.5	Administrations-GUI	58
4.6	Interna	59
4.6.1	Funktionsweise der Man-in-the-Middle-Komponente	59
4.6.2	Abruf von Server-Zertifikaten: <code>SSLUtil.getSiteCertificate</code>	63
4.6.3	Fälschen von Server-Zertifikaten: <code>CertificateManager.fakeCertificate</code>	66
4.7	Mögliche Erweiterungen	67
4.7.1	Kryptographisch gesicherte Speicherung der Credentials	67
4.7.2	TLS-Upgrade statt Redirect für HTTP-Verbindungen	68
5	Fazit und Ausblick	69
A	Installation und Erstkonfiguration	71
A.1	Installation	71
A.1.1	Proxy	71
A.1.2	PostgreSQL-Server	71
A.1.3	PostgreSQL-Web-Administrations-GUI: <code>phpPgAdmin</code>	73
A.2	Erstkonfiguration des Proxys	74
B	Administration	76
B.1	Globale Einstellungen	76
B.2	Nutzerverwaltung	77
B.2.1	Übersicht	78
B.2.2	Benutzerkonten erstellen	78
B.2.3	Benutzerkonten verwalten	80
B.3	Zugriffsrechte verwalten	80
B.3.1	Zugriffsrecht erstellen	81
B.3.2	Zugriffsrecht löschen	82
B.4	Credential-Verwaltung	82
B.4.1	Credential-Eigenschaften	83
B.4.2	Kennwortbasierte Credentials	85
B.4.3	X.509-Credentials	86
B.4.4	Formularbasierte Credentials	86
B.5	Delegation	89
B.6	Credential-Liste	90
C	Deployment	92
C.1	Rollout der Zertifikate	92
C.2	Konfiguration der Proxy-Einstellungen	92
C.2.1	Manuelle Proxy-Konfiguration	93
C.2.2	Automatische Proxy-Konfiguration per WPAD-Protokoll	94

D Konfiguration der Testumgebung	96
D.1 Installation der Komponenten	96
D.2 Konfiguration des HTTP-Servers	96
D.2.1 Basic Authentication	96
D.2.2 Digest Authentication	97
D.3 Konfiguration des HTTPS-Servers	98
D.3.1 Zertifikatserzeugung	98
D.3.2 Konfiguration von mod_ssl	99
E Datenbank-Schema	101
E.1 Certificate-Authority: Tabelle »ca«	101
E.2 Benutzerverwaltung: Tabelle »users«	101
E.3 Credential-Store: Tabelle »credentials«	102
E.4 Zugriffskontrolle: Tabelle »access«	102
E.5 Konfigurationsverwaltung: Tabelle »config«	102
Abbildungsverzeichnis	103
Tabellenverzeichnis	105
Literaturverzeichnis	106

1.1. Motivation

Die Nutzung des World Wide Web für unternehmenskritische Dienste – sei es für die Durchführung des Zahlungsverkehrs, die Nutzung von B2B-Handelsportalen oder auch allgemein für die Nutzung von webbasierten ASP¹-Lösungen – hat in den vergangenen Jahren kontinuierlich zugenommen. Den meisten dieser Lösungen ist gemeinsam, dass bei ihnen sowohl schützenswerte Daten übertragen werden als auch eine sichere Authentifikation des Nutzers gegenüber dem Diensteanbieter erforderlich ist.

Die folgenden Betrachtungen sollen die sich in diesem Umfeld ergebenden Szenarien und Probleme bei der authentifizierten Nutzung von WWW-Diensten etwas näher beleuchten. Sie gehen dabei beispielhaft von einem kleinen Unternehmen aus, in dem Mitarbeiter auf externe geschützte Web-Inhalte zugreifen und sich dazu authentifizieren müssen.

1.1.1. Szenario

Alice ist Geschäftsführerin dieses Unternehmens, ihr Prokurist heißt Bob. Alice führt den Zahlungsverkehr ihres Unternehmens mit dem per SSL/TLS² [Die99] gesicherten Internet-Banking ihrer Hausbank durch. Die Authentifikation gegenüber dem Banking-Server erfolgt über ein SSL-Client-Zertifikat. Alice plant für den Sommer eine zweiwöchige Urlaubsreise. Da sie an ihrem Urlaubsort keinerlei Möglichkeit haben wird, die laufenden Bankgeschäfte Ihres Unternehmens zu managen, möchte sie ihrem Prokuristen Bob während ihrer Abwesenheit die Möglichkeit geben, den Zahlungsverkehr stellvertretend für sie abzuwickeln. Hierzu müsste Bob sich gegenüber der Bank mit dem SSL-Client-Zertifikat von Alice authentifizieren. Alice möchte es jedoch vermeiden, das Zertifikat direkt an Bob weiterzugeben, denn nach ihrer Rückkehr möchte sie die Bankgeschäfte wieder vollständig in Eigenregie führen. Ebenfalls wünscht sie sich eine Möglichkeit, nach Ende ihres Urlaubs in einem Protokoll einsehen zu können, wann Bob Transaktionen im Internet-Banking durchgeführt hat.

Im Unternehmen von Alice sind in der IT-Entwicklungsabteilung weitere Mitarbeiter beschäftigt: Carl und Deborah. Beide benötigen für ihre Tätigkeit Zugang zu einem Portal, das ebenfalls mit einem Authentifikationsmechanismus gesichert ist. Jedoch hat Alice vom Portal-Betreiber lediglich ein Benutzerkonto für ihr gesamtes Unternehmen erhalten. Da Zugriffe auf das Portal teilweise kostenpflichtig sind, möchte sie einen

¹Application Service Provider

²Die Begriffe *SSL* und *TLS* bezeichnen verschiedene Versionen desselben Protokolls und werden im weiteren Verlauf synonym verwendet.

Überblick darüber, wie oft Carl und Deborah das Portal jeweils in Anspruch nehmen. Der Portal-Betreiber kann diese Aufschlüsselung nicht bieten, da er für das gesamte Unternehmen nur einen Account zur Verfügung stellt.

Schließlich erhält jeder Mitarbeiter der Firma von einem weiteren Diensteanbieter einen geschützten Zugang zu dessen Informationsangebot. Die IT-Abteilung des Unternehmens hat jedoch bereits ein weiteres Projekt in Planung, in dessen Rahmen jeder Mitarbeiter erneut ein zusätzliches Zertifikat für die SSL-Client-Authentifikation erhalten würde. Da die IT-Abteilung bereits chronisch mit Arbeit überlastet ist, sucht sie nach einer Möglichkeit, den Aufwand für die Schlüsselverteilung an die Mitarbeiter sowie den zukünftigen Wartungsaufwand zu minimieren.

1.1.2. Ziel dieser Arbeit

Im Rahmen dieser Diplomarbeit soll eine Authentifikationslösung entworfen und prototypisch implementiert werden, welche die beim Zugriff einer geschlossenen Benutzergruppe auf externe geschützte WWW-Inhalte auftretenden und weiter oben anhand des fiktiven Beispiels des Unternehmens von Alice charakterisierten Probleme löst: Mitglieder der Gruppe sollen zeitlich begrenzt ihre Identität an einen Stellvertreter weitergeben können, mehrere Gruppenmitglieder sollen dasselbe Geheimnis zur Authentifikation gegenüber einem Server verwenden können, ohne dass diese Teilnehmer Kenntnis vom Geheimnis haben müssen, und schließlich soll das Deployment so weit vereinfacht werden, dass bei jedem Mitglied der geschlossenen Benutzergruppe nur noch ein einziger Schlüssel zur Authentifikation installiert werden muss.

1.2. Gliederung

Diese Arbeit beginnt in Kapitel 2 mit einer grundlegenden Einführung in die Authentifikationsthematik (Abschnitt 2.1). Es folgt eine präzise Definition von Credentials und deren Delegation und Gruppennutzung, die bereits im Rahmen der Motivation anhand des Stellvertreter-Szenarios eingeführt wurden. Im Anschluss folgt eine Übersicht über die gängigen Authentifikationsmechanismen im WWW (Abschnitt 2.2). Daran schließt sich eine Einführung in die sichere Kommunikation im Internet mittels SSL/TLS (Abschnitt 2.3) an. Das Kapitel endet mit einem Abschnitt über das Tunneln von TLS-Verbindungen über HTTP-Proxy-Server (Abschnitt 2.4).

Im Anschluss erfolgt in Kapitel 3 der konzeptionelle Entwurf des Authentifikationsproxys. Das Kapitel beginnt dabei mit einer Einführung in die Problematik (Abschnitt 3.1), die sich an die Problembeschreibung aus der Einleitung anschließt, und stellt nach einer Anforderungsanalyse (Abschnitt 3.2) unterschiedliche Realisierungsansätze vor (Abschnitt 3.3). Diese Realisierungsvarianten werden im Anschluss auf Vor- und Nachteile untersucht (Abschnitt 3.4) und einer gegenüberstellenden Bewertung unterzogen. Ausgehend von dieser Bewertung wird der HTTP-Proxy als Architektur für die spätere Implementierung ausgewählt (Abschnitt 3.5).

Kapitel 4 behandelt die Details einer prototypischen Implementierung des HTTP-Proxy.

Kapitel 5 rundet die Arbeit mit einem Resümee ab und zeigt einige zukünftige Erweiterungsmöglichkeiten für den realisierten Prototyp auf.

Die Installation und Konfiguration dieses Prototyps werden ebenso wie seine Administration im Anhang dokumentiert. Das Deployment auf den Client-Systemen ist in einem weiteren Kapitel beschrieben. Eine Anleitung zur Einrichtung einer Testumgebung für den Proxy und ein Auszug aus dessen Datenbankschema schließen den Anhang ab.

Im Rahmen dieses Kapitels sollen grundlegende, zum Verständnis des späteren Entwurfs einer Authentifikationslösung notwendige und hilfreiche Konzepte, Protokolle und Mechanismen näher erläutert werden.

Begonnen wird hierbei mit einer allgemeinen Einführung in die Authentifikationsthematik. Daran schließt sich eine Definition von Credentials sowie eine Erläuterung von deren verschiedenen Nutzungsszenarien an, gefolgt von einem Überblick über aktuell im WWW eingesetzte Authentifikationsmechanismen. Im Anschluss erfolgt eine Einführung in das zur kryptografischen Absicherung von TCP/IP-Verbindungen eingesetzte SSL/TLS-Protokoll. Den Abschluss des Kapitels bildet eine Erläuterung der als Basis für HTTPS-Proxys genutzten »CONNECT«-Methode zum Tunneln von TLS-Verbindungen über HTTP-Proxys.

2.1. Authentifikation

Authentifikation nimmt in dieser Arbeit eine zentrale Rolle ein. Daher soll zunächst der Begriff näher definiert werden. Im Anschluss werden klassische Problembereiche bei der Authentifikation in Mehrbenutzerumgebungen dargestellt.

2.1.1. Definition

Eine Authentifikation ist ein Verfahren, um einem Kommunikationspartner die eigene Identität nachzuweisen, sich ihm gegenüber zu legitimieren. Anhand der Art, auf die dieser Nachweis im Einzelnen erfolgt, lassen sich Authentifikationsschemata in verschiedene Klassen einteilen: Authentifikation durch Wissen, Authentifikation durch Besitz sowie Authentifikation mittels einer Eigenschaft.

Authentifikation durch Wissen

Diese Klasse von Authentifikationsverfahren ist die im IT-Umfeld momentan wohl am weitesten verbreitete. Ein Benutzer authentifiziert sich dabei gegenüber einem System, indem er diesem die Kenntnis eines geteilten Geheimnisses durch Preisgabe desselben nachweist. Gängige Verfahren verwenden hierzu eine geheime Ziffernfolge (PIN), ein geheimes Wort (Kennwort) oder einen geheimen Satz (*Passphrase*), häufig in Verbindung mit einem öffentlich bekannten Parameter, z. B. dem Namen eines Benutzerkontos.

Problematisch ist bei der wissensbasierten Authentifikation die Tatsache, dass sich das zur Authentifikation notwendige Geheimnis – unbemerkt vom berechtigten Nutzer – z. B. während der Eingabe ausspähen und später unberechtigt nutzen oder gar vervielfältigen lässt.

Eine Möglichkeit zur Steigerung der Sicherheit gegen Ausspähen des Geheimnisses könnte z. B. die Abfrage lediglich eines zufällig gewählten Teils des Geheimnisses sein, so dass selbst ein Angreifer, dem das Ausspähen der kompletten Benutzereingabe gelingt, nicht Kenntnis vom vollständigen Geheimnis erlangt.

Authentifikation durch Besitz

Authentifikation durch Besitz beschreibt eine Klasse von Authentifikationsverfahren, die im Alltag ständig gegenwärtig ist. Die Authentifikation erfolgt hierbei dadurch, dass ein Nutzer seine Identität oder Berechtigung nachweist, indem er einem System gegenüber den Besitz dieses Gegenstandes belegt. Das wohl am häufigsten anzutreffende Beispiel für diese Verfahrensklasse ist das klassische Schloss, das sich nur mit Hilfe des passenden Schlüssels öffnen lässt. Auch Public-Key-Verfahren zählen zu dieser Kategorie. Bei diesen stellt der (häufig nur als Soft-Token vorhandene) private Schlüssel den Besitz dar, den es durch Erstellen einer gültigen Signatur oder Entschlüsseln eines Kryptotextes zu beweisen gilt.

Problem dieser Authentifikationsklasse ist, dass sich die zur Authentifikation eingesetzten Gegenstände möglicherweise unberechtigterweise vervielfältigen lassen. Im Falle von klassischen Schlüsseln versucht man dieses Problem durch eine Steigerung der Komplexität einzudämmen. Bei Public-Key-Verfahren stellt der Einsatz von Signaturkarten mit integriertem Kryptoprozessor das Mittel der Wahl zum Schutz des privaten Schlüssels dar. Doch auch bei Smartcards ist es beispielsweise mit so genannten Side-Channel-Attacks möglich, den privaten Schlüssel an den Schutzmechanismen der Smartcard vorbei auszulesen.

Authentifikation mittels Eigenschaften und Fähigkeiten

Die Authentifikation anhand von Eigenschaften und Fähigkeiten kommt häufig in der Biometrie zum Einsatz. Beispiele sind Fingerabdruck- oder Iris-Scanner (Eigenschaften) sowie Tastaturanschlag (Fähigkeit). Um eine qualitativ hochwertige Erkennung zu ermöglichen, ist es erforderlich, dass die gewählten Eigenschaften weitgehend unveränderlich sind, also auch keinem Alterungsprozess unterliegen. Biometrische Eigenschaften und Fähigkeiten sind nicht übertragbar und können daher weder gewollt noch ungewollt weitergegeben werden.

Die Betrachtung dieser Authentifikationsklasse soll an dieser Stelle nicht weiter vertieft werden, da sie im Rahmen dieser Arbeit keine Rolle spielt. Der interessierte Leser findet z. B. in [Eck01] eine ausführliche Abhandlung der Thematik.

Hybride Verfahren

Alle genannten Authentifikationsklassen haben sowohl Stärken als auch Schwächen. Daher bietet es sich an, Verfahren aus verschiedenen Klassen zu kombinieren, um die Gesamtsicherheit der Authentifikation zu erhöhen. Gängige Varianten sind die Kombination von Authentifikation durch Wissen und Besitz, z. B. beim Schutz einer Smartcard oder eines Soft-Tokens mit einer PIN oder einem Kennwort, und die Kombination von Authentifikation durch Besitz und mittels einer Eigenschaft, z. B. bei der Kopplung eines biometrischen Merkmals an eine Smartcard.

2.1.2. Credentials

Ein im weiteren Verlauf dieser Arbeit häufig wiederkehrender Begriff ist der des *Credentials*. Dieser soll daher zunächst klar definiert werden.

Das Oxford Advanced Learner's Dictionary [Cow89] definiert Credentials als: »documents showing that a person is what he claims to be, is trustworthy, etc.«

Damit umreißt es auch bereits sehr präzise die Bedeutung, die dem Begriff im Authentifikationskontext zukommt: Ein Credential dient dem Nachweis einer Identität, der eine bestimmte Berechtigung zusteht, Zugang zu einer geschützten Ressource zu erhalten.

Bezogen auf die im vorangegangenen Abschnitt vorgestellten Authentifikationsklassen kann ein Credential also sowohl ein geteiltes Geheimnis (Wissen), der Besitz eines Gegenstandes oder auch Soft-Tokens oder eine bestimmte Eigenschaft oder Fähigkeit sein. Von diesen drei Credential-Klassen spielen jedoch im Rahmen dieser Arbeit nur die beiden erstgenannten eine Rolle: Wissen in Form von Benutzer/Kennwort-Tupeln sowie der Besitz des privaten Schlüssels zu einem X.509-Zertifikat. Eigenschaften finden dagegen schwerpunktmäßig bei biometrischen Verfahren Verwendung, die momentan im Bereich der Authentifikation im World Wide Web allerdings noch so gut wie nicht zum Einsatz kommen.

Weitere Details zur Thematik im Allgemeinen und zu den in den beiden Folgeabschnitten beschriebenen Anwendungsfällen finden sich in [Gin04].

2.1.3. Delegation

Soll nun das Nutzungsrecht an einem Credential von seinem Eigentümer A temporär an einen weiteren Nutzer B übertragen werden, spricht man von einer so genannten Delegation des Credentials.

Ein naiver Ansatz zur Realisierung einer Delegation wäre die simple Weitergabe des dem Credential zu Grunde liegenden Geheimnisses an den neuen Nutzer B . Hier ist es jedoch im Anschluss an die erwünschte Nutzung quasi unmöglich, eine wirkliche Beendigung der Nutzung durch B durchzusetzen. Auch wenn dieser vorgibt, das Credential vernichtet zu haben, ist dies selbst im Falle von an Massenspeicher gebundenen Credentials, wie X.509-Zertifikaten, nur schwer nachweisbar: Zu leicht lässt sich das Credential kopieren. Im Falle von Credentials, die sich vollständig im Gedächtnis halten lassen, z. B. Benutzer/Kennwort-Tupel, ist eine nachhaltige und nachweisbare Rücknahme der Delegation prinzipbedingt nicht realisierbar. Auch lässt sich mit diesem naiven Ansatz nicht verhindern oder kontrollieren, ob B das an ihn delegierte Credential an Dritte weitergibt, da er zur Nutzung des Credentials in dessen Besitz sein muss und dieses bei den gängigen Credential-Typen somit problemlos und für den ursprünglichen Inhaber nicht feststellbar duplizieren kann.

2.1.4. Gruppennutzung

Die Gruppennutzung eines Credentials beschreibt dessen gleichzeitige Nutzung durch mehrere Mitglieder einer geschlossenen Benutzergruppe. Sie lässt sich damit als Spezia-

lisierung der Delegation auffassen: Ein Credential wird von A nicht mehr an nur eine Person B , sondern an eine Menge von Personen C_i delegiert.

Auch hier ist der naive Ansatz nicht geeignet, um z. B. im Falle des Ausscheidens eines Gruppenmitglieds C_j eine nachhaltige Rücknahme der mit dem Credential verbundenen Berechtigungen durchzusetzen, genauso wenig, wie die im vorherigen Abschnitt bereits dargestellte Weitergabe des Credentials an eine dritte Person D zu verhindern.

2.2. Authentifikationsmechanismen im WWW

Eine wichtige Grundlage für den späteren Entwurf des Proxys stellen die im World Wide Web zum Einsatz kommenden Authentifikationsmechanismen dar.

In diesem Abschnitt sollen dabei die gebräuchlichsten Lösungen betrachtet und erläutert werden. Dies sind im einzelnen die *Basic*- und die *Digest Authentication*, die *TLS Client Authentication* sowie formularbasierte Ansätze. Nischenlösungen, z. B. Authentifikationsmechanismen basierend auf proprietären Java-Applets und ActiveX-Controls, werden bewusst ausgelassen.

2.2.1. Basic- und Digest Authentication

Das Ziel der *Basic*- und *Digest Authentication* [Fra99] ist die Integration eines Authentifikationsmechanismus in das HTTP-Protokoll [Fie99]. Erreicht wird diese Integration durch eine Erweiterung der Header des HTTP-Protokolls um authentifikationsspezifische Header-Elemente und Status-Codes, auf die später noch genauer eingegangen wird.

Während bei der *Basic Authentication* das zur Authentifikation eingesetzte Geheimnis, also das Kennwort, quasi im Klartext zum Server übertragen wird, sofern keine TLS-gesicherte Verbindung zum Einsatz kommt, verwendet die *Digest Authentication* kryptografische Hashes in einem *Challenge-Response*-Verfahren und vermeidet damit die unsichere Übertragung des Kennworts im Klartext.

Beiden Verfahren gemeinsam ist die Art der Integration in das HTTP-Protokoll: Möchte der Server nach Anforderung einer Ressource durch einen Client diese dem Client nur nach dessen erfolgreicher Authentifikation zur Verfügung stellen, antwortet der Server auf die Anforderung des Clients (*Request*) anstatt mit der Ressource mit einem speziell für diesen Zweck vorgesehenen Status-Code »401 Authorization Required« (*Challenge*). Die genaue Form der Antwort hängt davon ab, über welche Authentifikationsverfahren der Server die Authentifikation durchführen möchte; in jedem Fall gibt jedoch der »WWW-Authenticate«-Header die unterstützten Authentifikationsverfahren sowie den so genannten *Authentication Realm* an, der zusammen mit der Basisadresse der angefragten Ressource den Schutzbereich der Authentifikation vorgibt. Auf diese *Challenge* des Servers antwortet der Client nun mit seinen Credentials (*Response*). Auf den genauen Aufbau dieser verfahrensspezifischen Antwort wird in den folgenden Abschnitten noch näher eingegangen.

Der schematische Ablauf der Authentifikation ist in der Grafik in Abbildung 2.1 visualisiert.

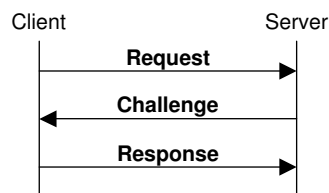


Abbildung 2.1: Vereinfachter Ablauf der Authentifikation

```
GET / HTTP/1.0
Host: www.example.com
```

Abbildung 2.2: Initialer Zugriff des Clients auf den Server

2.2.1.1. Basic Authentication

Für die *Basic Authentication* ist der erste Teil des Dialogs, die *Challenge*, die der Server als Antwort auf eine initiale Anfrage des Clients (Abbildung 2.2) an diesen übermittelt, exemplarisch in Abbildung 2.3 dargestellt.

Wie bereits für den allgemeinen Fall beschrieben, zeigt auch hier der Status-Code »401 Authorization Required« die Notwendigkeit einer Authentifikation an. Der »WWW-Authenticate«-Header gibt in diesem Fall die *Basic Authentication* als Authentifikationsverfahren vor. Als Antwort auf seine *Challenge* erwartet der Server in diesem Fall eine speziell codierte Version von Benutzernamen und Kennwort. Dazu werden Benutzername und Kennwort durch einen Doppelpunkt getrennt aneinandergelängt und anschließend einer Base64-Codierung¹ [Fre96] unterzogen.

Im Folgenden soll die Erstellung dieser Antwort kurz an einem Beispiel erläutert werden. Dabei gehen wir von der *Challenge* in Abbildung 2.3 als Authentifikationsgrundlage aus. Für diese soll nun eine Authentifikationsantwort für den Benutzernamen »Alice« und das Kennwort »secret« erstellt werden. Dazu konkatenieren wir zuerst beide Werte getrennt durch einen Doppelpunkt und erhalten »Alice:secret«. Im Anschluss

¹Allgemein handelt es sich hierbei um eine Basistransformation zur Basis 64, die es ermöglicht, binäre Daten ausschließlich mit Hilfe druckbarer Zeichen darzustellen. Ziel ist es, durch diese Transformation eventuelle Zeichensatzkonvertierungsprobleme auf dem Transportweg zwischen Client und Server zu vermeiden.

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="Auth_Realm"

[...]
```

Abbildung 2.3: *Challenge* des Servers bei der *Basic Authentication*

```
GET / HTTP/1.0
Host: www.example.com
Authorization: Basic QWxpY2U6c2VjcmV0
```

Abbildung 2.4: *Response* des Clients bei der *Basic Authentication*

führen wir eine entsprechende Base64-Codierung² dieser Zeichenkette durch und erhalten als Ergebnis »QWxpY2U6c2VjcmV0«.

Mit Hilfe dieser Information lässt sich nun eine Antwortnachricht auf die *Challenge* des Servers konstruieren, die exemplarisch in Abbildung 2.4 dargestellt ist. Die Authentifikationsnachricht wird dabei nach der Bezeichnung des Authentifikationsverfahrens in den »Authorization«-Header eingebettet.

Der Server empfängt diese Informationen, macht die Base64-Codierung rückgängig und prüft die Credentials auf Gültigkeit. Im Erfolgsfall liefert er die angeforderte Ressource aus, im Fehlerfall fordert er den Client erneut zur korrekten Authentifikation auf.

Probleme und Gegenmaßnahmen

Das wohl offensichtlichste und auch schwerwiegendste Problem der *Basic Authentication* ist seine Verwundbarkeit gegen Abhören: Fängt ein potentieller Angreifer die Antwort des Clients ab, so gelangt er in den Besitz des kompletten, zur Authentifikation notwendigen Credentials, das er durch Rückgängigmachen der Base64-Codierung mit vernachlässigbarem Aufwand erhält. Das im Anschluss als Klartext vorliegende Credential lässt sich dann direkt zur Authentifikation per Webbrowser nutzen.

Eine mögliche Maßnahme gegen diesen Angriff ist das Tunneln des HTTP-Datenstroms in einer TLS-Verbindung, wodurch sich das Abhören der Credentials nachhaltig verhindern lässt.

Eine weitere Möglichkeit, die Sicherheit der Authentifikation gegenüber dem Server zu verbessern, soll im folgenden Abschnitt erläutert werden.

2.2.1.2. Digest Authentication

Ziel der hier vorgestellten *Digest Authentication* ist es, die gravierenden Sicherheitsmängel der im vorherigen Abschnitt behandelten *Basic Authentication* unter Beibehaltung der *Challenge/Response*-Architektur zu beheben.

Die Idee hinter dem veränderten Verfahren ist, dass der Client auf die *Challenge* des Servers hin nicht mehr das unverschlüsselte Kennwort an diesen zurücksendet, sondern auf dieses vorher eine kryptografische Hashfunktion³ H anwendet und deren Ergebnis $H(\text{Kennwort})$, aus dem sich nun keine Rückschlüsse mehr auf das ursprüngliche Kennwort ziehen lassen, an den Server sendet.

²Hierzu existieren für fast jede Programmiersprache entsprechende Bibliotheken. Ferner gibt es für einfache Tests auch kommandozeilen- oder webbasierte Tools, z. B. unter der URL <http://www.fourmilab.ch/webtools/base64/>.

³Bei der *Digest Authentication* ist dies in der überwiegenden Zahl der Fälle der MD5-Algorithmus.

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Digest realm="Auth_Realm", nonce="1071105641"
[...]
```

Abbildung 2.5: *Challenge* des Servers bei der *Digest Authentication*

Während sich hiermit das Kennwort zwar sehr effektiv gegen Ausspähen schützen lässt, wäre diese Maßnahme alleine in ihrer Schutzwirkung äußerst beschränkt, da sie keinerlei Schutz vor *Replay*-Angriffen, d. h. dem Wiedereinspielen abgefangener Authentifikationsdaten, bietet.

Die *Basic Access Authentication* löst dieses Problem elegant dadurch, dass sie mit der *Challenge* des Servers eine Zufallszahl *nonce* an den Client überträgt, welche dieser beim Hashen des Kennworts in seine Berechnungen einbeziehen muss. Dazu ist es notwendig, die übertragenen Header-Elemente gegenüber der *Basic Authentication* zu modifizieren: Der »*WWW-Authenticate*«-Header gibt als Authentifikationsverfahren nun »*Digest*« an und erhält einen zusätzlichen Parameter »*nonce*«, mit dem eine vom Server generierte Zufallszahl⁴ übertragen wird.

Für das Beispiel aus Abschnitt 2.2.1.1 ist die entsprechend an die nun genutzte *Digest Authentication* angepasste *Challenge* des Servers in Abbildung 2.5 exemplarisch dargestellt.

Nach dem Empfang dieser *Challenge* berechnet der Client im Anschluss die *Response*, mit der er gegenüber dem Server nachweist, im Besitz des geheimen Kennworts zu sein. Die Berechnung erfolgt dabei wie folgt: Zu Beginn berechnet der Client zwei Werte *A1* und *A2*. Die beiden zusätzlichen Parameter *method* und *uri* bezeichnen hierbei die HTTP-Request-Methode der Anfragenachricht und die URI der Ressource, auf die zugegriffen wird.

$$A1 = H(\textit{username} + \text{,,:} + \textit{realm} + \text{,,:} + \textit{password})$$
$$A2 = H(\textit{method} + \text{,,:} + \textit{uri})$$

Im Anschluss daran berechnet er aus diesen beiden Werten und der vom Server übermittelten Zufallszahl *nonce* die an den Server zu übermittelnde Antwort:

$$\textit{response} = H(A1 + \text{,,:} + \textit{nonce} + \text{,,:} + A2)$$

Dieses Ergebnis übermittelt er nun als Teil des »*Authorization*«-Headers zur Authentifikation an den Server (siehe Abbildung 2.6).

Zusätzlich zur sicheren Authentifikation erlaubt die *Digest Authentication* optional den Schutz der Integrität der übertragenen Daten sowie die Nutzung eines Sitzungskonzepts zur Reduzierung des Berechnungs- und Kommunikationsaufwands. In der Praxis

⁴Die Implementierung des weit verbreiteten Apache-Webservers überträgt stattdessen einfach das aktuelle Datum im Unix-Format (Sekunden seit dem 01.01.1970).

```
GET / HTTP/1.1
Host: www.example.com
Authorization: Digest username="Alice", realm="Auth_Realm", \
    nonce="1071105641", uri="/", \
    response="916e7680d559f8dbfc9558635bef7619"
```

Abbildung 2.6: *Response* des Clients bei der *Digest Authentication*

scheinen diese Erweiterungen allerdings momentan nicht im Einsatz zu sein: Weder die aktuelle Version des Internet Explorer noch Mozilla Firefox machen bei der Kommunikation mit einem mit aktivierter *Digest Authentication* konfigurierten Apache 1.3 von diesen Möglichkeiten Gebrauch.

Weitere Details würden den Rahmen dieses Kapitels sprengen. Daher sei an dieser Stelle auf die Originalliteratur [Fra99] verwiesen.

2.2.2. TLS Client Authentication

Die *TLS Client Authentication* nutzt die später in den Abschnitten 2.3.2.2 und 2.3.4.1 dargestellten Mechanismen des TLS-Protokolls zur clientseitigen Authentifikation im Rahmen einer TLS-Verbindung.

Der Client weist dem Server dabei während des so genannten Handshakes am Anfang einer TLS-Verbindung nach, im Besitz des privaten Schlüssels zu seinem X.509-basierten PKI-Zertifikat zu sein. Wie dieser Nachweis wiederum vom Server bewertet wird, ist einzig und allein dessen Entscheidung. Gängige Prüfungskriterien sind Attribute des X.509-Zertifikats des Clients, z. B. dessen Aussteller oder der *Distinguished Name* des Inhabers. Auch die Prüfung auf die Existenz einer Vertrauenskette bis zu einer vertrauenswürdigen Certificate-Authority ist ein mögliches Kriterium.

2.2.3. Formularbasierte Ansätze

Formularbasierte Varianten der Benutzerauthentifikation gehören wohl zusammen mit der *Basic Authentication* zu den am weitesten verbreiteten Authentifikationsmechanismen im World Wide Web. Jedoch handelt es sich allgemein bei den als Grundlage verwendeten HTML-Formularen um ein Konzept, das ursprünglich nie für eine sichere Authentifikation entwickelt wurde.

Das Verfahren baut auf den so genannten HTML-Formularen [Rag97] auf, die allgemein dazu genutzt werden können, durch den Benutzer erfasste Daten mittels eines standardisierten Verfahrens an einen HTTP-Server zu übertragen. Diese Formulare erlauben das Platzieren von verschiedenen Eingabeelementen über den HTML-Tag »INPUT« im HTML-Quellcode, so z. B. für die Erfassung von Text (Datenfeldtyp »TEXT«) oder die maskierte Erfassung von Kennwörtern (Datenfeldtyp »PASSWORD«).

Die Kommunikation zwischen Client und Server wird dabei über das bereits an anderer Stelle erwähnte HTTP-Protokoll abgewickelt. Je nach Konfiguration des Formu-

```

<HTML>
  <HEAD>
    <TITLE>Authentifikation</TITLE>
  </HEAD>
  <BODY>
    Bitte authentifizieren Sie sich:<P>

    <FORM ACTION="http://www.example.com/authenticate.cgi"
      METHOD="GET">
      Benutzer: <INPUT TYPE="TEXT" NAME="benutzer"><BR>
      Kennwort: <INPUT TYPE="PASSWORD" NAME="kennwort"><P>

      <INPUT TYPE="SUBMIT" VALUE="Anmelden">
    </FORM>
  </BODY>
</HTML>

```

Abbildung 2.7: HTML-Dokument zur formularbasierten Authentifikation

```

GET /authenticate.cgi?benutzer=Alice&kennwort=secret HTTP/1.1
Host: www.example.com

```

Abbildung 2.8: Formularbasierte Authentifikation bei Nutzung der »GET«-Methode

lars erfolgt die Übertragung der Nutzerdaten entweder nach der so genannten »GET«-, oder der »POST«-Methode. Erstgenannte überträgt dabei die Daten als Teil des *Query Strings* im Header der HTTP-Anfrage durch ein Fragezeichen getrennt von der URL (siehe Abbildung 2.8), letztgenannte als Teil des *Request Bodys* der Anfrage (siehe Abbildung 2.8).

Auf der Serverseite werden die so übertragenen Informationen z. B. mittels des CGI-Protokolls [Coa99] an eine Authentifikationsanwendung weitergeleitet, welche die übermittelten Informationen prüft, bei erfolgreicher Authentifikation eine Sitzung für den Client erstellt und diese anschließend anfrageübergreifend im globalen Zustand des Servers speichert. Wie dies im Einzelfall geschieht, ist vollständig implementierungsabhängig.

```

POST /authenticate.cgi HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 30

benutzer=Alice&kennwort=secret

```

Abbildung 2.9: Formularbasierte Authentifikation bei Nutzung der »POST«-Methode

Als mögliche Implementierungsvarianten seien an dieser Stelle jedoch HTTP-Cookies [Kri00], eine Zugriffssteuerung basierend auf der authentifizierten IP-Adresse sowie die Verwendung von *Session-IDs* als Teil der URL genannt. Häufig kommt zur Verwaltung der Sitzungsdaten eine Datenbank zum Einsatz.

Probleme

Wie auch bei der *Basic Authentication* ist bei formularbasierten Authentifikationsverfahren keinerlei Schutz gegen das Abhören der übertragenen Daten vorhanden, da die Authentifikationsdaten direkt und unverschlüsselt im Datenpaket des HTTP-Requests verschickt werden. Erfolgt die Nutzung des Verfahrens mit der »GET«-Methode, so werden sensible Informationen, wie oben erläutert, als Teil des *Request Strings* übertragen. Dies hat den negativen Effekt, dass so die Authentifikationsdaten möglicherweise mehrfach auf Systemen, z. B. in der Browser-History des Anwenders, in den Logs zwischen-geschalteter (transparenter) Proxy-Server und schließlich im Log des angesprochenen Servers, gespeichert werden.

Ohne den Einsatz von zusätzlichen Sicherungsmaßnahmen zur Gewährleistung von Vertraulichkeit und Integrität, wie sie z. B. TLS bietet, ist daher vom Einsatz dieses Verfahrens in sicherheitskritischen Bereichen dringend abzuraten. Vor dem Hintergrund, dass Anwender häufig identische oder zumindest ähnliche Credentials für Dienste mit unterschiedlicher Schutzbedürftigkeit verwenden, empfiehlt sich darüber hinaus auch die Sicherung wenig sicherheitskritischer Authentifikationsprozesse, um das Risiko für den Anwender gering zu halten.

Ein weiteres Problem formularbasierter Authentifikationslösungen, das besonders dann sichtbar wird, wenn der formularbasierte Mechanismus automatisiert genutzt werden soll, ist, dass weder die Feldnamen, noch die Anzahl der zur Authentifikation genutzten Felder standardisiert sind. Im schlimmsten Fall ändern sich Feldnamen sogar dynamisch, da sie z. B. an eine *Session-ID* gebunden sind, wie dies bis vor kurzem⁵ beim Webmail-Angebot von T-Online der Fall war.

2.3. Transport Layer Security

Das in diesem Abschnitt vorgestellte SSL/TLS-Protokoll geht auf einen Entwurf [Hic94] von Hickman im Rahmen seiner Tätigkeit bei Netscape Communications aus dem Jahr 1994 zurück. Es kam (damals noch als SSL 2.0) erstmalig in Netscapes WWW-Browser *Netscape Navigator* zum verschlüsselten Zugriff auf HTTP-Ressourcen zum Einsatz. Im Folgenden soll die aktuelle Version dieses Protokolls, TLS 1.0 [Die99], näher erläutert werden.

Im weiteren Verlauf dieser Abhandlung wird dabei mehrfach Bezug auf verschiedene kryptografische Konzepte und Algorithmen genommen; sie in dieser Diplomarbeit erschöpfend zu erläutern, würde allerdings den Rahmen erheblich sprengen. Stattdessen erfolgt im Kontext jeweils ein Verweis auf die entsprechende Originalliteratur.

⁵Bei der bandbreitenschonenden Version für Modem-Nutzer ist dies auch heute noch der Fall.

URL: <https://modem.webmail.t-online.de/>

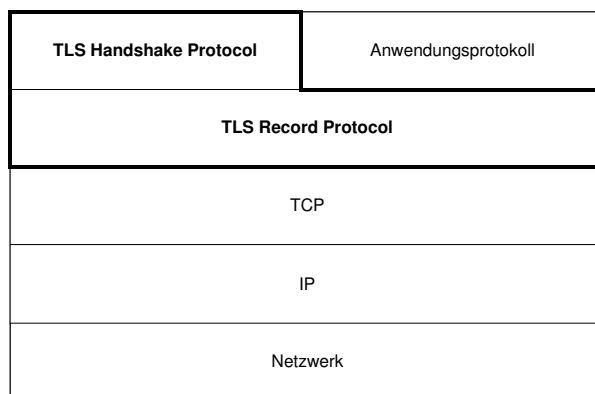


Abbildung 2.10: Einordnung von TLS in die TCP/IP-Schichtenarchitektur

2.3.1. Übersicht

Das TLS-Protokoll hat das Ziel, eine vertrauliche, authentische und integre Kommunikation zwischen zwei Kommunikationspartnern über einen unsicheren Kanal wie das Internet zu gewährleisten. TLS ist dabei in der Lage, ein beliebiges TCP/IP-basiertes Protokoll in einem Tunnel zu transportieren. Die protokollspezifische API wurde von Grund auf so konzipiert, dass sie mit minimalen Änderungen kompatibel zu der Socket-API in C/C++ bzw. dem Socket-Factory-Konzept in Java ist und dadurch eine Erweiterung von Anwendungsprotokollen um eine Schicht zur Gewährleistung von Vertraulichkeit, Integrität und Authentizität mit geringstmöglichem Aufwand ermöglicht.

Erreicht werden die obigen Schutzziele durch den kombinierten Einsatz von gängigen Verschlüsselungs-, Authentifikations- und Integritätsmechanismen, die beim TLS-Protokoll auf zwei Komponenten verteilt sind: Das *TLS Record Protocol* sowie das *TLS Handshake Protocol*.

Das *TLS Record Protocol* übernimmt die Verschlüsselung und Kompression der Nutzdaten sowie die Integritätssicherung, während das *TLS Handshake Protocol* für Verwaltungsaufgaben wie das Aushandeln von kryptografischen Algorithmen, die Authentifikation der Kommunikationspartner und die Kommunikation von Fehlerzuständen verantwortlich ist. Die logische Einordnung dieser beiden Protokollkomponenten in das vereinfachte TCP/IP-Schichtenmodell [Tan03] ist aus Abbildung 2.10 ersichtlich: Sowohl das *TLS Handshake Protocol* als auch das abzusichernde Anwendungsprotokoll setzen auf dem *TLS Record Protocol* auf, das sich gegenüber den beiden erstgenannten Protokollen wie ein Protokoll der Transportschicht, der Transportschicht gegenüber dagegen wie ein Protokoll der Anwendungsschicht verhält.

Zur Reduzierung des Overheads beim Verbindungsaufbau nutzt TLS ein Sitzungskonzept. Dieses sorgt dafür, dass globale Parameter wie anzuwendende Verschlüsselungs-, MAC-⁶ und Datenkompressionsverfahren sowie zur Nutzung der jeweiligen Verfahren notwendige Geheimnisse nicht vor jeder Verbindung zwischen denselben Kommunikationspartnern neu ausgehandelt werden müssen. Stattdessen werden diese Parameter für

⁶Message Authentication Code, eine parametrisierte Hashfunktion [Buc01, S. 175-176]

eine gewisse Zeit zur erneuten Nutzung gecached. Sowohl das *TLS Record Protocol* als auch das *TLS Handshake Protocol* greifen auf diesen sitzungsspezifischen globalen Zustand zu.

2.3.2. Kryptografische Verfahren

Bevor im weiteren Verlauf dieses Kapitels die einzelnen TLS-Teilprotokolle erläutert werden, soll dieser Abschnitt vorab einen kurzen Überblick über die beteiligten kryptografischen Verfahren und Konzepte geben.

2.3.2.1. Verschlüsselung

TLS unterstützt zum Verschlüsseln der Nutzdaten sowohl den Einsatz von symmetrischen Blockchiffren im so genannten *Cipher Block Chaining Mode* [Buc01, S. 65-68] als auch Stromchiffren [Buc01, S. 72-74]. Im ursprünglichen Standard unterstützte Blockchiffren sind RC2, IDEA, DES und TripeDES, die einzige unterstützte Stromchiffre ist RC4. Details zu diesen Verschlüsselungsalgorithmen finden sich in [Sch96].

RFC 3268 [Cho02] beschreibt schließlich eine Erweiterung der unterstützten Chiffren um AES als patentfreies und starkes symmetrisches Verfahren, das mit der Implementierung in aktuellen Versionen von OpenSSL⁷ bereits eine signifikante Verbreitung gewonnen haben sollte.

2.3.2.2. Authentifikation

Für die optionale Authentifikation der Kommunikationspartner während des später erläuterten *Handshakes* sieht der TLS-Standard im Rahmen des in Abschnitt 2.3.2.3 dargestellten Schlüsselaustauschs verschiedene Public-Key-Signaturverfahren vor, nämlich RSA [Riv78] und DSS [Nis94]. Es kommt dabei zur Authentifikation immer das Signaturverfahren zum Einsatz, welches auch zum Signieren des als Authentifikationsnachweis genutzten Zertifikats verwendet wurde.

X.509-Zertifikate

Unabhängig vom gewählten Signaturverfahren kommen dabei so genannte X.509-Zertifikate zum Einsatz. Diese werden von einer vertrauenswürdigen Certificate-Authority (kurz: CA) als Herausgeber des Zertifikats mit einer Public-Key-Signatur versehen, nachdem die CA sich basierend auf im Voraus festgelegten Richtlinien, der so genannten *Certification Policy*, von der Authentizität des Inhabers überzeugt hat. Mit ihrer Signatur bestätigt die CA also, dass der Inhaber des Zertifikats derjenige ist, der er vorgibt zu sein. Jeder, dem die Zertifizierungsrichtlinien der entsprechenden CA als ausreichend erscheinen und der damit der CA sein Vertrauen schenkt, kann dadurch auch auf die Authentizität des Zertifikatsinhabers vertrauen.

Zur Realisierung dieses Ansatzes sind im Zertifikat bestimmte Angaben hinterlegt, z. B. der Aussteller und Inhaber des Zertifikats, dessen Gültigkeitsdauer sowie die zuge-

⁷URL: <http://www.openssl.org/>

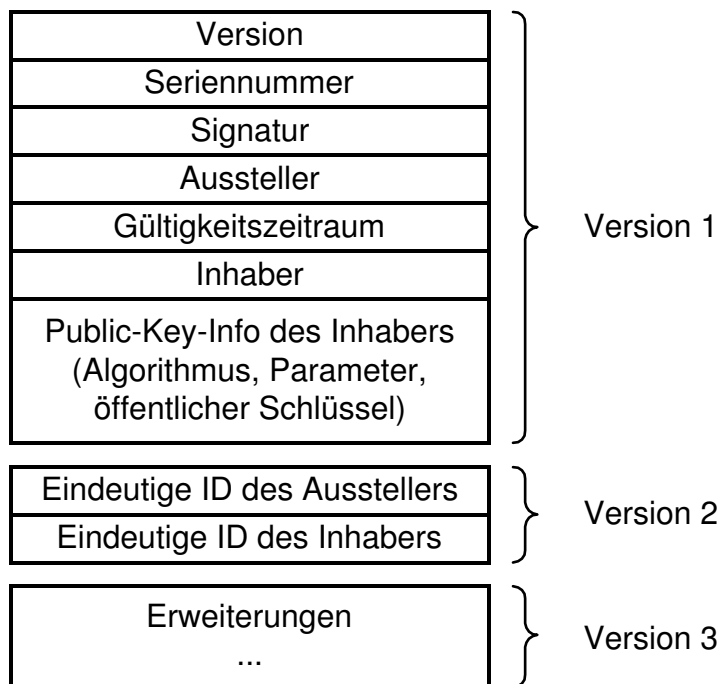


Abbildung 2.11: X.509-Zertifikat

hörigen Public-Key-relevanten Informationen, wie der öffentliche Schlüssel des Inhabers und die vom Herausgeber angefertigte Signatur des Zertifikats.

Abbildung 2.11 gibt einen vollständigen Überblick über die im X.509-Standard für Zertifikate definierten Datenfelder. Die unterschiedlichen Versionsangaben beziehen sich dabei auf unterschiedliche Revisionen der Datenstruktur, wobei TLS diese bis einschließlich Version 3 unterstützt. Weiterführende Informationen zum X.509-Standard finden sich in [Cci88].

2.3.2.3. Schlüsselaustausch

TLS sieht zum Schlüsselaustausch verschiedene Verfahren vor. Ziel jedes dieser Verfahren ist es, einen gemeinsamen geheimen Schlüssel für die symmetrische Verschlüsselung zwischen beiden Kommunikationspartnern über einen unsicheren Kommunikationskanal zu etablieren. Die vier im TLS-Standard hierfür vorgesehenen Verfahren sollen im Folgenden kurz vorgestellt werden.

RSA

Hier wird der geheime Schlüssel zufällig generiert, mit dem öffentlichen RSA-Schlüssel des Kommunikationspartners verschlüsselt und im Anschluss an den Kommunikationspartner übermittelt.

Anonymous Diffie-Hellman

Das Diffie-Hellman-Verfahren (kurz: DH) zum Schlüsselaustausch basiert auf dem Problem diskreter Logarithmen. Es erlaubt den beiden Kommunikationspartnern, einen geheimen symmetrischen Schlüssel sicher über ein unsicheres Kommunikationsmedium auszutauschen. Eine Authentifikation der beiden Kommunikationspartner findet allerdings durch das DH-Verfahren nicht statt, so dass dieses anfällig gegen Man-in-the-Middle-Angriffe ist. Zu deren Abwehr sind stattdessen zusätzlich geeignete Maßnahmen zu ergreifen. Für weitere Details zum Verfahren sei auf [Buc01, S. 129-133] verwiesen.

Fixed Diffie-Hellman

Das *Fixed Diffie-Hellman*-Verfahren (kurz: fDH) löst die Man-in-the-Middle-Problematik des DH-Verfahrens dadurch, dass die öffentlichen Diffie-Hellman-Parameter fest vorgegeben und von einer vertrauenswürdigen CA signiert werden, z. B. als Teil des X.509-Zertifikats des Servers. Allerdings besteht nun die Gefahr, dass das diskrete Logarithmusproblem in diesem Fall auf Grund der für einen langen Zeitraum konstanten DH-Parameter mit Hilfe eines *Brute-Force*-Angriffs gebrochen werden kann.

Ephemeral Diffie-Hellman

Ähnlich wie beim fDH-Verfahren ist es auch Ziel des Ephemeral Diffie-Hellman-Verfahrens (kurz: eDH), einen Man-in-the-Middle-Angriff auf den Schlüsselaustausch a priori unmöglich zu machen. Der hierzu gewählte Ansatz unterscheidet sich jedoch vom fDH-Verfahren: Statt fest vorgegebener DH-Parameter werden diese wie beim DH-Verfahren bei Bedarf neu generiert. Um jedoch die Authentizität der Daten zu gewährleisten, erweitert man das Verfahren um eine Public-Key-Komponente, d. h. die DH-Parameter werden vor dem Versand an den Kommunikationspartner mit einem Public-Key-Signaturverfahren signiert. Durch ein entsprechendes, von einer vertrauenswürdigen CA signiertes Zertifikat lässt sich so die Authentizität der DH-Parameter feststellen.

2.3.2.4. Integritätssicherung

Als Algorithmus zur Integritätssicherung kommt beim TLS-Protokoll das HMAC-Verfahren [Kra97] zum Einsatz, das eine von einem speziellen Hash-Verfahren unabhängige Realisierung eines parametrisierten Hash-Verfahrens zur Integritätssicherung beschreibt. Im Rahmen des im folgenden Abschnitt beschriebenen *TLS Record Protocol* ist das zu verwendende Hash-Verfahren dabei während des Handshakes (siehe Abschnitt 2.3.4) zwischen den Kommunikationspartnern frei verhandelbar, während für die Handshake-eigenen Integritätssicherungsmaßnahmen immer MD5 [Riv92] und SHA-1 [Nis93] zum Einsatz kommen.

2.3.3. TLS Record Protocol

Jegliche Kommunikation im Rahmen des TLS-Protokolls erfolgt über das *TLS Record Protocol*, welches sowohl die Nutzdaten des transportierten Anwendungsprotokolls als auch die im Rahmen des in Abschnitt 2.3.4 behandelten *TLS Handshake Protocol* über-

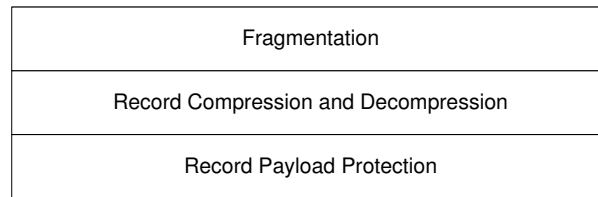


Abbildung 2.12: Schichtenarchitektur des TLS Record Protocol

tragenen Daten auf ihrem Weg zwischen den beiden Kommunikationsendpunkten passieren.

Ähnlich wie TCP/IP lässt sich auch das TLS Record Protocol anhand eines Schichtenmodells⁸ darstellen, das Abbildung 2.12 zeigt. Die folgenden Ausführungen beschreiben dabei die beim Senden eines Datenpakets ausgeführten Operationen. Beim Empfang durchläuft ein Datenpaket die drei Schichten in umgekehrter Reihenfolge. Dabei werden jeweils die inversen Operationen gegenüber dem Sendevorgang ausgeführt. Die grundlegenden Aufgaben der drei Schichten sind dabei folgende:

- *Fragmentation Layer*: Teilt den Datenstrom in Pakete mit einer Länge von maximal 2^{14} Byte auf.
- *Record Compression and Decompression Layer*: Komprimiert die Nutzdaten.
- *Record Payload Protection Layer*: Verschlüsselt die Nutzdaten und schützt sie mittels eines MACs vor Veränderungen.

2.3.3.1. Fragmentation Layer

Die mit *Fragmentation Layer* bezeichnete Schicht nimmt aus der darüber liegenden Schicht Datenpakete entgegen und teilt diese in Datenpakete mit einer Nutzdatenlänge von maximal 2^{14} Byte auf. Die Nutzdaten werden dabei um einen Header erweitert, der Informationen über den Typ der Nutzdaten, die Protokollversion sowie die Länge der Nutzdaten enthält. Der Aufbau eines solchen Fragments ist beispielhaft in Abbildung 2.13 für ein Datenpaket der Länge $n \leq 2^{14}$ Byte dargestellt. Sofern ein Datenpaket vor der Verarbeitung größer als diese Beschränkung war, wird es entsprechend auf mehrere Fragmente aufgeteilt.

2.3.3.2. Record Compression and Decompression Layer

Im Anschluss erfolgt in der darauf folgenden Schicht eine verlustfreie Komprimierung der Nutzdaten jedes Fragments. Der zur Komprimierung zu verwendende Algorithmus

⁸Der TLS-Standard dokumentiert das *TLS Record Protocol* zwar anhand einer Schichtenarchitektur, verfügbare Realisierungen, z. B. OpenSSL, folgen diesem Ansatz bei der Implementierung jedoch nur bedingt. Aufgrund dessen klarer Struktur wird im Folgenden jedoch trotzdem der im Standard vorgesehene Schichtenansatz behandelt.

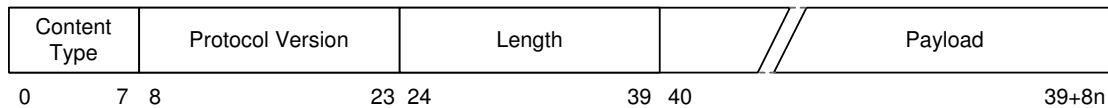


Abbildung 2.13: Datenpaket nach dem Passieren der *Fragmentation*-Schicht

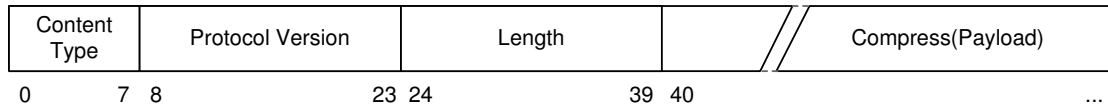


Abbildung 2.14: Datenpaket nach der Komprimierung

ist dabei für jede TLS-Session mittels eines globalen Parameters *Compression Method* festgelegt. Im Anschluss an die Komprimierung wird im Header die Länge der Nutzdaten entsprechend deren Größe nach der Kompression aktualisiert. Sofern keine Komprimierung erwünscht ist oder zwischen den Kommunikationspartnern noch kein Komprimierungsverfahren ausgehandelt wurde, kommt ein spezieller Algorithmus zum Einsatz, der so genannte *Null*-Algorithmus, der als Identitätsfunktion realisiert ist, die Nutzdaten also unverändert lässt.

2.3.3.3. Record Payload Protection Layer

Aufgabe dieser Schicht ist die eigentliche kryptografische Absicherung der übertragenen Nutzdaten, über die vor der Verschlüsselung eine MAC-Berechnung zur Integritätssicherung erfolgt. Dieser MAC wird an die komprimierten Nutzdaten aus der vorangegangenen Schicht angehängt.

Die weitere Behandlung des Pakets hängt nun davon ab, ob für die eigentliche Verschlüsselung eine Stromchiffre oder eine Blockchiffre zum Einsatz kommt: Im Falle einer Stromchiffre⁹ werden Nutzdaten und MAC ohne weitere Vorkehrungen per Stromchiffre verschlüsselt; im Anschluss wird bei Bedarf das Längenfeld des Pakets an die Größe der Nutzdaten nach der Verschlüsselung angepasst. Soll jedoch eine Blockchiffre im *Cipher Block Chaining Mode* zum Einsatz kommen, muss vor der Chiffrierung sichergestellt werden, dass die Länge der zu verschlüsselnden Daten ein Vielfaches der Blockgröße der eingesetzten Blockchiffre ist. Bei einer Blockgröße k und zu verschlüsselnden Daten der Länge n werden daher als so genanntes *Padding* noch $(n \bmod k) - 1$ Füll-Bytes sowie ein Längen-Byte, das die Anzahl der Füll-Bytes zur späteren Rekonstruktion der Originaldaten angibt, an die Nutzdaten und den MAC angehängt. Die Füll-Bytes bekommen dabei denselben Wert wie das Längen-Byte zugewiesen. Der gesamte Datenblock wird im Anschluss verschlüsselt, was zu der in Abbildung 2.16 dargestellten Datenanordnung führt.

⁹Hierzu zählt auch die spezielle *Null*-Chiffre, die zum Einsatz kommt, wenn keine Verschlüsselung erwünscht ist oder noch keine Verschlüsselungsparameter ausgehandelt wurden. Genau wie die *Null*-Komprimierung wird diese durch die Identität realisiert.

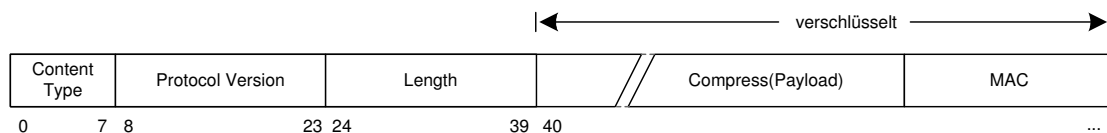


Abbildung 2.15: Datenpaket nach Verschlüsselung mittels Stromchiffre

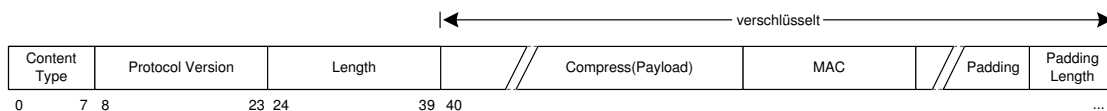


Abbildung 2.16: Datenpaket nach Verschlüsselung im CBC-Modus

Im Anschluss an diesen Schritt ist die Kodierung der Daten abgeschlossen und diese werden über den *Transport Layer* des Netzwerkprotokollstacks an den Empfänger versandt. Zur Rekonstruktion der Original-Daten führt dieser den im vorangegangenen Abschnitt beschriebenen Prozess in umgekehrter Reihenfolge mit den jeweils inversen Funktionen für Verschlüsselung und Datenkompression aus.

2.3.4. TLS Handshake Protocol

Das *TLS Handshake Protocol* ist aus drei Teilprotokollen aufgebaut, mit Hilfe derer TLS sowohl die Authentifikation, das Aushandeln der Verschlüsselungsparameter als auch die Benachrichtigung des Kommunikationspartners bei Problem- oder Fehlersituationen realisiert. Diese drei Protokolle sind im Einzelnen das *Handshake*-, *Change Cipher Spec*- sowie das *Alert Protocol*, die in den folgenden drei Abschnitten näher erläutert werden.

2.3.4.1. Handshake Protocol

Das *Handshake Protocol* wird dazu eingesetzt, den sitzungsglobalen Zustand einer TLS-Sitzung zwischen beiden Kommunikationspartnern auszuhandeln sowie deren optionale Authentifikation durchzuführen. Der so ausgehandelte Sitzungszustand besteht aus den folgenden Komponenten:

- einem vom Server vergebenen *Identifier*, der einen Sitzungszustand eindeutig identifiziert
- dem optionalen X.509-Zertifikat des jeweiligen Kommunikationspartners
- dem vom *TLS Record Protocol* verwendeten Datenkomprimierungsalgorithmus
- den kryptografischen Parametern, der so genannten *Cipher Spec*, d. h. den zur Verschlüsselung und MAC-Berechnung anzuwendenden Algorithmen

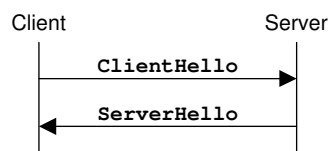


Abbildung 2.17: TLS-Handshake: Phase 1

- dem so genannten *Master Secret*, einem zwischen beiden Kommunikationspartnern geteilten Geheimnis, von dem die zur Verschlüsselung und Integritätsprüfung notwendigen Parameter abgeleitet werden
- einem Flag »*is resumable*«, das angibt, ob die Sitzung zum Aufbau neuer Verbindungen genutzt werden kann

Das *Handshake Protocol* führt das Aushandeln dieser Parameter in vier Phasen durch, die im Folgenden näher erläutert werden sollen.

Phase 1: Ermitteln der Fähigkeiten der Kommunikationspartner

Zu Beginn des Handshakes tauschen die beiden Kommunikationspartner, im Folgenden Client und Server genannt, so genannte *Hello*-Nachrichten aus. Ziel dieses Nachrichtenaustauschs ist es, sozusagen die Schnittmenge der von beiden Partnern unterstützten Algorithmen sowie die maximal unterstützte Version des TLS/SSL-Protokolls zu ermitteln. Dazu sendet der Client dem Server zunächst eine *ClientHello*-Nachricht, welche die höchste unterstützte Protokollversion, eine vom Client generierte Zufallszahl, eine priorisierte Liste der unterstützten Kompressionsverfahren und so genannten *Cipher Suites*¹⁰ sowie optional die *Session-ID* einer bereits bestehenden Sitzung enthält.

Auf diese Nachricht antwortet der Server mit einer *ServerHello*-Nachricht. In dieser teilt er dem Client die höchste von beiden Partnern unterstützte Protokollversion mit, ebenfalls eine Zufallszahl sowie aus der vom Client übermittelten Liste von Kompressionsalgorithmen und *Cipher Suites* jeweils ein ausgewähltes, zur späteren Nutzung vorgesehenes Verfahren. Findet der Server in der Liste der vom Client übermittelten kryptografischen Verfahren keine kompatiblen Algorithmen vor, bricht er stattdessen den Handshake an dieser Stelle ab.

Der stark vereinfachte Ablauf dieser ersten Handshake-Phase geht aus Abbildung 2.17 hervor.

Phase 2: Server-Authentifikation und Schlüsselaustausch

Abhängig vom in der ersten Phase des Handshakes ausgewählten Verfahren zum Schlüsselaustausch werden nun in der zweiten Phase die in Abbildung 2.18 dargestellten Nachrichten vom Server zum Client gesendet.

¹⁰Eine *Cipher Suite* spezifiziert die kryptografischen Algorithmen für Verschlüsselung der Nutzdaten (symmetrische Block- oder Stromchiffre), Schlüsselaustausch und Integritätssicherung (MAC).

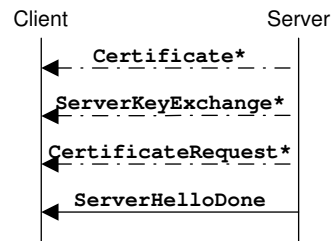


Abbildung 2.18: TLS-Handshake: Phase 2

Certificate Der Server verschickt mit dieser Nachricht sein X.509-Zertifikat incl. seines zugehörigen öffentlichen Schlüssels, sofern es sich bei der ausgewählten Methode zum Schlüsselaustausch um ein nicht-anonymes Verfahren¹¹ handelt.

ServerKeyExchange Im Anschluss verschickt der Server eine *ServerKeyExchange*-Nachricht, falls die *Certificate*-Nachricht (sofern sie gesendet wurde) nicht ausreichend Informationen enthält, um den Kommunikationspartnern den Austausch eines symmetrischen Schlüssels zu ermöglichen. Dies ist bei DH, eDH und einer Variante¹² des RSA-Verfahrens der Fall. Die *ServerKeyExchange*-Nachricht übermittelt abhängig vom gewählten Verfahren entweder einen temporären öffentlichen RSA-Schlüssel oder die öffentlichen DH-Parameter sowie bei nicht-anonymen Verfahren eine kryptografische Signatur der übermittelten Parameter.

CertificateRequest Wünscht der Server eine Authentifikation des Clients, so teilt er ihm dies direkt im Anschluss an das Versenden der *ServerKeyExchange*-Nachricht mittels einer *CertificateRequest*-Nachricht mit. Diese Nachricht enthält den gewünschten Zertifikatstyp¹³ sowie eine Liste der *Distinguished Names* zulässiger Certificate-Authoritys, die er als Aussteller von Client-Zertifikaten akzeptiert.

ServerHelloDone Mit der *ServerHelloDone*-Nachricht beendet der Server die zweite Phase des Handshakes, deren Ablauf in Abbildung 2.18 noch einmal zusammenfassend dargestellt ist, und wartet im Anschluss die Antworten des Clients in der dritten Phase ab.

Phase 3: Client-Authentifikation und Schlüsselaustausch

Die dritte Phase des Handshakes beinhaltet die Client-Authentifikation sowie den clientseitigen Teil des Schlüsselaustauschs.

¹¹Nicht-anonyme Verfahren sind RSA, eDH und fdH.

¹²In der Spezifikation wird diese mit `RSA_EXPORT` bezeichnet. Sie kommt zum Einsatz, falls der öffentliche RSA-Schlüssel des Server-Zertifikats länger als 512 Bit ist, da aktuelle US-Exportbestimmungen keine Nutzung von Moduln länger als 512 Bit für den Schlüsselaustausch in Software zulassen.

¹³RSA, DSS, RSA für fdH oder DSS für fdH.

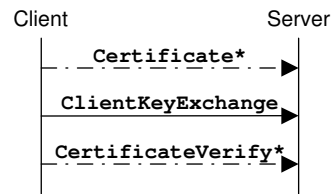


Abbildung 2.19: TLS-Handshake: Phase 3

Certificate Den Anfang macht dabei eine optional vom Client an den Server gesendete *Certificate*-Nachricht, falls diese zuvor vom Server per *CertificateRequest*-Nachricht angefordert wurde. Sie enthält das zur Authentifikation genutzte X.509-Zertifikat des Clients. Der weitere Aufbau der Nachricht entspricht der in der zweiten Phase vom Server verschickten *Certificate*-Nachricht.

ClientKeyExchange Die *ClientKeyExchange*-Nachricht schließt den Austausch des geheimen Schlüssels im Rahmen des Schlüsselaustauschs ab. Die dabei übertragenen Informationen hängen stark vom ausgewählten Schlüsselaustauschverfahren ab: Kommt ein RSA-basiertes Verfahren zum Einsatz, so erzeugt¹⁴ der Client den geheimen Schlüssel, chiffriert diesen mit dem öffentlichen Schlüssel des Servers und schickt den nun geschützten symmetrischen Schlüssel an den Server. Bei einem DH-basierten Verfahren übermittelt der Client dagegen seinen Teil der DH-Parameter.

CertificateVerify Die optionale *CertificateVerify*-Nachricht schließt die vorletzte Phase des Handshakes ab. Sie wird gesendet, falls das Client-Zertifikat zur Signatur genutzt werden kann¹⁵. Die Nachricht enthält eine Signatur über alle bis zum Zeitpunkt vor dem Versenden der Nachricht verschickten Handshake-Nachrichten und dient dem Schutz vor Man-in-the-Middle-Angriffen. Bis einschließlich SSL 2.0 war dies die einzige Möglichkeit, sich vor solchen Angriffen zu schützen. Wie wir sehen werden, existiert seit SSL 3.0 jedoch eine zusätzliche Sicherungsmaßnahme im vierten und letzten Teil des Handshakes.

Zur besseren Übersicht zeigt Abbildung 2.19 noch einmal die während der dritten Phase des Handshakes versendeten Nachrichten.

Phase 4: Abschluss des Handshakes

Am Ende des Handshakes aktiviert zuerst der Server mit Hilfe des im folgenden Abschnitt beschriebenen *Change Cipher Spec Protocol* die während der Handshake-Phase ausgehandelten Parameter. Im Anschluss sendet er (bereits unter Nutzung der neuen Parameter) eine *Finished*-Nachricht an den Client, die eine kryptografische Prüfsumme

¹⁴z. B. mit Hilfe eines kryptografisch sicheren Zufallszahlengenerators

¹⁵Dies trifft auf alle Zertifikate zu außer solchen, die zur Beglaubigung von fDH-Parametern genutzt werden.

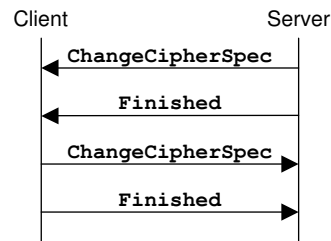


Abbildung 2.20: TLS-Handshake: Phase 4

me¹⁶ über die bisherige Datenkommunikation¹⁷ im Rahmen des *Handshake Protocol* an den Kommunikationspartner übermittelt. Der Empfänger auf der Gegenseite prüft diese auf Korrektheit und bricht die Verbindung im Falle eines Fehlers nach dem Senden eines Alerts (siehe Kapitel 2.3.4.3) ab. Ziel dieser Prüfung ist es, sicherzustellen, dass beide Kommunikationspartner während des kompletten (ungeschützten) Handshakes zu jedem Zeitpunkt mit derselben Gegenstelle kommuniziert haben. Ist dies nicht gegeben, stimmt die auf der Gegenseite zu Kontrollzwecken berechnete Prüfsumme nicht mit der von der Gegenstelle versendeten überein, da Server und Client zumindest in Teilen unterschiedliche Handshake-Nachrichten „gesehen“ haben. Analog versendet der Client im Anschluss beide Nachrichten in umgekehrter Richtung.

2.3.4.2. Change Cipher Spec Protocol

Das *Change Cipher Spec Protocol* hat die Aufgabe, einen mit Hilfe des *Handshake Protocol* neu ausgehandelten globalen Sitzungszustand für die Verarbeitung zukünftiger Nachrichten durch das *Record Protocol* zu aktivieren. Eine *Change Cipher Spec*-Nachricht besteht aus exakt einem Byte mit dem Wert 01_{hex} .

2.3.4.3. Alert Protocol

Das *Alert Protocol* erlaubt das Versenden von Fehlernachrichten und Warnungen an den Kommunikationspartner. Eine solche Nachricht besteht jeweils aus einem so genannten *Alert Level*, das die Schwere¹⁸ der folgenden Nachricht angibt, sowie der eigentlichen Nachricht, der so genannten *Alert Description*. Für beide Informationen ist im Datenpaket, dessen Aufbau aus Abbildung 2.21 hervorgeht, jeweils ein Byte reserviert.

Beispiele für mögliche Nachrichten sind Fehlerbedingungen wie unerwartete Nachrichtentypen, fehlerhafte MACs, ein Fehlschlagen der Entschlüsselung oder Dekomprimierung der Nutzdaten, Fehler bei der Zertifikatsprüfung im Rahmen der Authentifizierung während des Handshakes oder auch allgemeine Protokollfehler. Eine Ausnahme stellt die

¹⁶Diese Prüfsumme wird im speziellen Fall mit Hilfe eines parametrisierten Pseudozufallszahlengenerators erzeugt. Details finden sich in den Kapiteln 5 und 7.4.9 der TLS-Spezifikation.

¹⁷*ChangeCipherSpec*-Nachrichten gehören ausdrücklich nicht hierzu.

¹⁸Im aktuellen Standard definierte *Alert Level* sind »Warnung« und »fataler Fehler«, wobei es bei Warnungen der Implementierung obliegt, über das weitere Vorgehen zu entscheiden, während fatale Fehler in jedem Fall zu einem Abbruch der Verbindung führen.

Alert Level	Alert Description
0	7 8 15

Abbildung 2.21: Aufbau des *Alert*-Datenpakets

vor dem Schließen einer Verbindung verschickte *Alert Description* »close_notify« dar, die keine Fehlerbedingung kommuniziert, sondern das reguläre Verbindungsende anzeigt.

Für eine vollständige Liste der möglichen *Alert Descriptions* sei an dieser Stelle auf die TLS-Spezifikation [Die99, S. 24-25] verwiesen.

2.3.5. Anmerkungen

TLS ist mit seinem Vorgänger SSL der wohl am weitesten verbreitete Standard zur vertraulichen und authentischen Kommunikation über sonst ungesicherte TCP/IP-Verbindungen.

Allerdings werden in der Literatur mehrere Angriffe auf das Verfahren beschrieben: So beschreiben Klíma, Pokorný und Rosa in [Kli03] einen Angriff auf TLS-Sitzungen, die im Rahmen des Schlüsselaustauschs RSA-Schlüssel verwenden. In [Cho02] wird ebenfalls ein nicht näher erläutertes Angriff auf alle CBC-basierten TLS-*Cipher-Suites* erwähnt.

Unabhängig davon ist TLS aktuell das einzige praktikabel einsetzbare Verfahren zur sicheren Nutzung des HTTP-Protokolls und stellt insofern das Mittel der Wahl dar.

2.4. Tunneln von TLS-Verbindungen über HTTP-Proxy-Server

Während sich klassische HTTP-Verbindungen problemlos über einen HTTP-Proxy abwickeln lassen [Fie99], besteht bei TLS-gesicherten HTTP-Verbindungen das Problem, dass selbst der *HTTP Request* bereits durch das TLS-Protokoll gesichert ist. Somit besteht für einen HTTP-Proxy ohne zusätzliche Protokollerweiterungen keine Möglichkeit, den Request an den eigentlichen Ziel-Server weiterzugeben, da ihm dieser nicht bekannt ist.

Aus diesem Grund etablierte sich im Lauf der 90er Jahre eine Erweiterung der von HTTP-Proxys unterstützten Request-Methoden um die so genannte »CONNECT«-Methode [Luo98]. Ursprünglich war diese auf das Tunneln von SSL-Verbindungen ausgelegt, effektiv erlaubt sie aber ein Tunneln beliebiger TCP/IP-basierter Protokolle durch einen HTTP-Proxy.

Die Möglichkeiten der »CONNECT«-Methode lehnen sich dabei an den gewöhnlichen TCP/IP-Verbindungsaufbau zum Öffnen eines bidirektionalen Kommunikationskanals an: Sie ermöglicht vom Proxy aus den Verbindungsaufbau zu einem wählbaren Port auf einem Ziel-System.

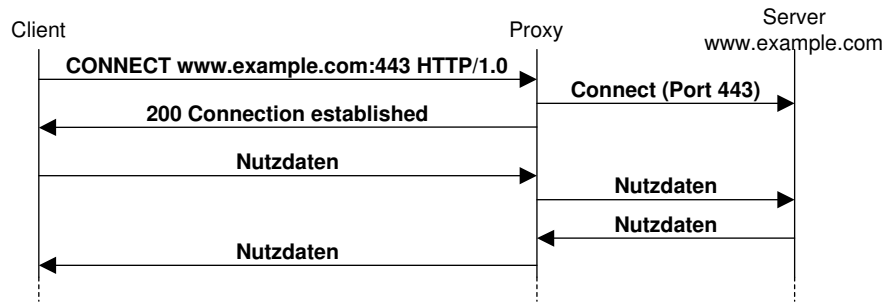


Abbildung 2.22: Schematischer Kommunikationsablauf der »CONNECT«-Methode

Dies wird wie folgt realisiert: Nachdem der Client eine Verbindung zum Proxy aufgebaut hat, teilt er dem Proxy in HTTP-Request-Notation seinen Wunsch mit, eine Verbindung zum Ziel-System aufzubauen: »CONNECT <ziel-host>:<port> HTTP/1.x«. Darauf folgen optional jeweils in einer eigenen Zeile zusätzliche HTTP-Header; eine leere Zeile schließt den Request ab. Den erfolgreichen Verbindungsaufbau teilt der Proxy dem Client mittels des Status-Codes »200 Connection Established« mit. Im Fehlerfall erfolgt analog die Rückgabe eines entsprechenden Fehlercodes, z. B. falls der Ziel-Host nicht erreichbar ist oder den Versuch eines Verbindungsaufbaus abweist.

Der zusammengefasste Ablauf der Kommunikation ergibt sich aus Abbildung 2.22.

In diesem Kapitel soll ein Konzept für eine im nächsten Kapitel als Prototyp implementierte Authentifikationslösung für das World Wide Web entworfen werden, die eine effiziente Verwaltung von Credentials in einer Mehrbenutzerumgebung sowie deren Delegation und Gruppennutzung ermöglicht.

3.1. Einführung in die Problematik

Die sicherste und für den Einzelplatznutzer auch am einfachsten zu realisierende Methode zur sicheren Kommunikation im World Wide Web ist die Nutzung von Diensten, die durch Verwendung des TLS-Protokolls eine Ende-zu-Ende-Sicherheit gewährleisten. Auch die Authentifikation gegenüber diesen Diensten lässt sich im Einzelplatzeinsatz effizient mit Hilfe von im Credential-Store¹ des genutzten Webbrowsers gespeicherten Credentials realisieren.

Sobald jedoch keine Einzelplatz-, sondern eine Mehrplatzanwendung vorliegt, oder ein Credential temporär an einen Dritten delegiert werden soll, treten die bereits in Kapitel 2.1 erläuterten Probleme auf: Weder lässt sich mit diesem Ansatz eine einmal durchgeführte Delegation revozieren, noch besteht eine Kontrolle darüber, wann Credentials genutzt und an wen sie weitergegeben werden. Darüber hinaus steigt der administrative Aufwand durch die Verteilung, Verwaltung und Wartung der nötigen Credentials unverhältnismäßig stark an: Jeder Nutzer ist ständig mit dezentral zu installierenden aktuellen Zertifikaten und Benutzer/Kennwort-Tupeln für die durch ihn genutzten Dienste zu versorgen.

Diese Probleme des dezentralen Ansatzes machen die Entwicklung einer Alternative unumgänglich, die im weiteren Verlauf dieses Kapitels entworfen wird.

3.2. Anforderungsanalyse

Die im Rahmen dieses Abschnitts folgende Anforderungsanalyse soll die bereits in der Einleitung in Kapitel 1.1 erfolgte grobe Abgrenzung des geforderten Funktionsumfangs einer Lösung weiter konkretisieren.

Ziel ist es, eine Authentifikationslösung für das World Wide Web zu entwerfen und als Prototyp zu implementieren. Diese soll sich beim Zugriff auf Ressourcen im WWW, die eine Authentifikation erfordern, stellvertretend und automatisch für den Nutzer gegenüber dem jeweiligen Server authentifizieren. Neben einer effizienten Administration

¹Im Fall des Microsoft Internet Explorer kommt als Credential-Store die so genannte *Windows Protected Storage* zum Einsatz, bei Mozilla nennt sich das entsprechende Äquivalent *Password Manager*.

von Credentials und Benutzerkonten sollte diese Lösung ebenfalls zur Realisierung einer Gruppennutzung von Credentials sowie deren revozierbarer Delegation in der Lage sein. Eine Unterstützung gängiger Authentifikationsverfahren (siehe Kapitel 2.2) ist dabei ebenso anzustreben wie eine transparente Realisierung des Dienstes gegenüber dem Endbenutzer. Da der Aufwand beim Deployment wesentlich an den Kosten eines solchen Gesamtsystems beteiligt ist, sollte ein weiteres Augenmerk auf ein effizientes Deployment der Authentifikationslösung gerichtet sein. Während Administratoren über eine Verwaltungsschnittstelle Benutzer, Credentials und Zugriffsrechte verwalten können, sollten Endbenutzer in der Lage sein, Delegationen, zu denen sie berechtigt sind, eigenverantwortlich durchzuführen, ohne einen Administrator hinzuziehen zu müssen. Da es sich bei den vom Authentifikationsdienst verwalteten Credentials um stark schützenswerte Daten handelt, ist zur Legitimation gegenüber dem Dienst sowohl bei der Credential-Nutzung als auch bei der Administration ein starkes Authentifikationsverfahren vorzusehen. Ein weiterer wichtiger Aspekt ist die Sicherheit der Credentials, die sowohl vor dem Zugriff durch berechtigte Benutzer als auch durch unberechtigte Dritte zu schützen sind.

3.3. Architekturvarianten

Für den im vorangegangenen Abschnitt geforderten Funktionsumfang der Authentifikationslösung sind verschiedene Implementierungsansätze denkbar. Einige dieser Architekturansätze sollen nun im weiteren Verlauf des Kapitels betrachtet und im Anschluss bewertet und verglichen werden.

Allen im Folgenden dargestellten Architekturen gemeinsam ist eine zentrale Komponente zur Speicherung der Credentials in einem so genannten Credential-Store sowie zur Verwaltung von Nutzerkonten und -berechtigungen (ACLs). Benutzer authentifizieren sich selbst gegenüber dieser zentralen Instanz, um sich für die Nutzung bestimmter Credentials zu legitimieren.

3.3.1. Stellvertreter-Architektur

Eine Klasse von Realisierungen lässt sich allgemein als Stellvertreter-Architektur charakterisieren. Bezeichnend für diese Art von Architektur ist der Einsatz einer Komponente, die als eine Art Middleware zwischen Client und Server zum Einsatz kommt und – angebunden an Credential-Store und ACL-Verwaltung – sich stellvertretend für den Nutzer gegenüber dem jeweiligen Ziel-Server authentifiziert (siehe Abbildung 3.1). Die Benutzerauthentifikation erfolgt bei dieser Architekturklasse gegenüber der jeweiligen Middleware, das dabei zum Einsatz kommende Verfahren ist in bestimmten Grenzen frei wählbar.

Im Folgenden werden drei Vertreter dieser Stellvertreter-Architektur betrachtet.

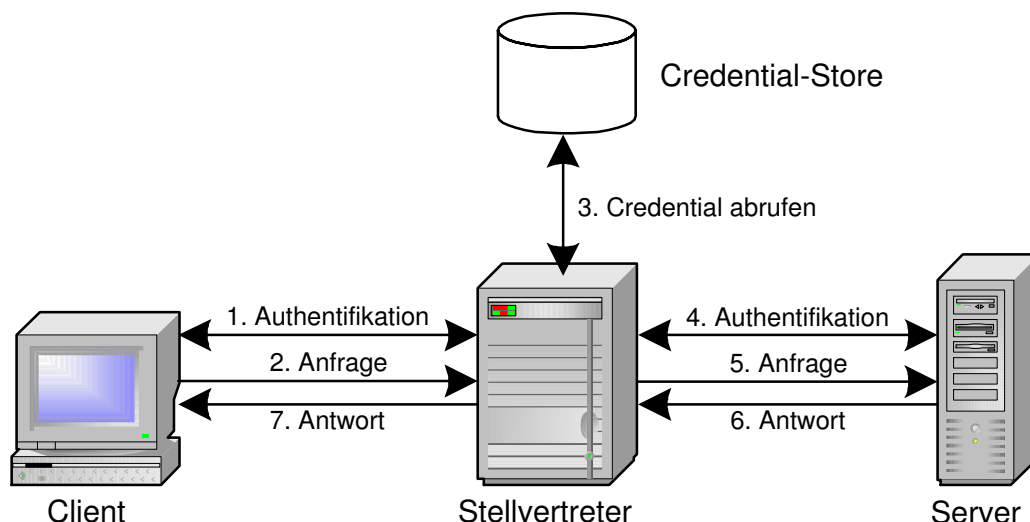


Abbildung 3.1: Schematischer Ablauf der Authentifikation bei einer Stellvertreter-Architektur

3.3.1.1. HTTP-Gateway

Die HTTP-Gateway genannte Variante realisiert einen Authentifikationsdienst auf der Anwendungsschicht des vereinfachten TCP/IP-Schichtenmodells als HTTP- oder HTTPS-Server. Auf diesen kann vom Client ohne besondere Zusatzmaßnahmen über einen konventionellen WWW-Browser zugegriffen werden.

Um die Authentifikationsdienste des Gateways zu nutzen, greift ein Client auf eine spezielle URL auf dem Gateway zu. Diese URL konstruiert der Client entweder selbstständig² oder nutzt alternativ ein vom Gateway abrufbares HTML-Eingabeformular oder auch Navigationsformular, das beim Abschicken zu einem entsprechenden URL-Aufruf auf dem Gateway führt. Das Gateway leitet aus dieser URL die eigentlich gewünschte URL auf dem Ziel-Server ab und baut zu diesem eine Verbindung auf. Dann ruft es die angeforderten Inhalte vom Server ab, führt dabei eventuell notwendige Authentifikationsvorgänge durch und übermittelt im Anschluss die abgerufenen Inhalte an den Client. Dieser hat sich bereits zu Beginn mit einem der in Kapitel 2.2 dargestellten Authentifikationsverfahren, vorzugsweise aber per X.509-Client-Authentifikation, gegenüber dem Gateway legitimiert.

Dem Client gegenüber tritt das HTTP-Gateway ausschließlich unter einer einzigen Basis-URL auf, die z. B. auf dem Host-Namen³ des HTTP-Gateways basiert. Damit lässt sich die Verbindung zum Gateway sehr leicht per TLS sichern, da ein einziges, von einer vertrauenswürdigen Certificate-Authority unterschriebenes Zertifikat zur Absiche-

²Denkbar ist beispielsweise ein Schema, bei dem an die Basis-URL des Gateways (hier im Beispiel »https://gateway.example.com/«) die URL der gewünschten Seite angehängt wird. Für einen Zugriff auf die URL »https://banking.example.com/« ergäbe sich dann die URL »https://gateway.example.com/https%3A//banking.example.com/«

³Im Beispiel »gateway.example.com«.

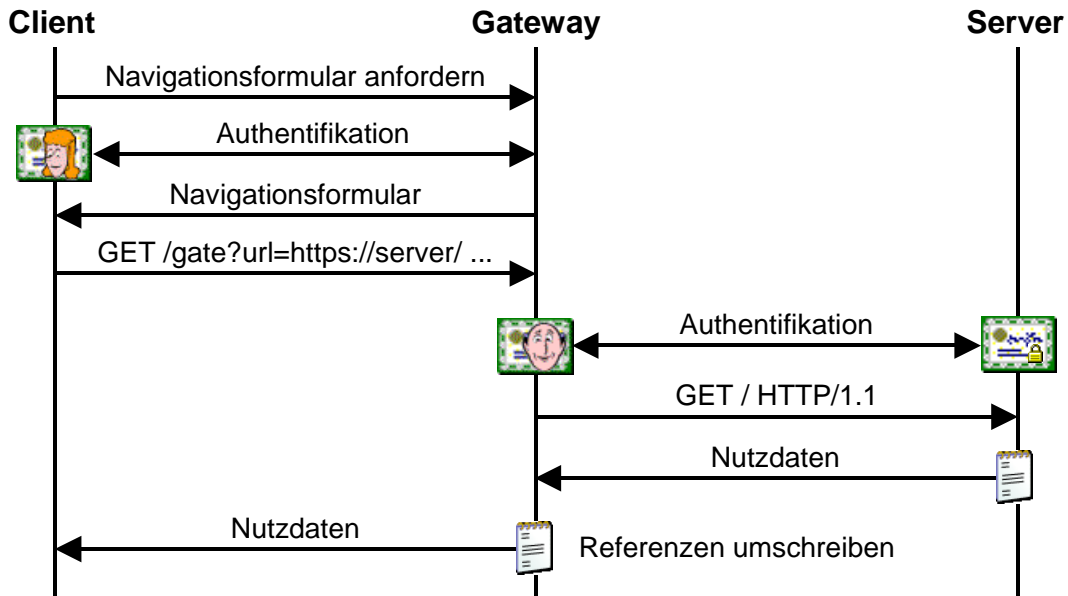


Abbildung 3.2: Kommunikationsablauf bei Nutzung des HTTP-Gateways

ung des kompletten Gateways genügt. Sofern das Wurzelzertifikat dieser CA bereits als Vertrauensanker in den eingesetzten Webbrowsern installiert ist, minimiert dies den Aufwand beim Deployment auf nahezu Null. Alternativ ist natürlich auch der Betrieb mit einem selbst-signierten Zertifikat oder einem von einer organisationsinternen CA ausgestellten Zertifikat möglich. Beim Gateway wird alle Kommunikation über *einen* Host geleitet. Dadurch erfordert diese Lösung allerdings zwingend das automatische Nachbearbeiten der an den Client zurückgelieferten HTML-Seiten und JavaScript-Dateien, um eventuell darin enthaltene Hyperlinks, Host-Namen, Bildverweise und sonstige Referenzen so zu modifizieren, dass diese ebenfalls über das HTTP-Gateway und nicht direkt vom ursprünglichen Server abgerufen werden.

Als Authentifikationsmechanismus des Endbenutzers gegenüber dem Gateway sind alle in Abschnitt 2.2 dargestellten Verfahren denkbar, wobei starken Verfahren wie der TLS-Client-Authentifikation aus verständlichen Gründen der Vorzug zu geben ist.

Abbildung 3.2 zeigt einen möglichen Kommunikationsablauf unter Einbeziehung von Client, Gateway und Server.

Probleme

Jedoch ist dieser Ansatz nicht völlig frei von Problemen: Gerade im Falle von aktiven Inhalten wie JavaScript gelingt das Umschreiben von Hyperlinks und Referenzen auf die Basis-URL des Gateways nur unvollständig, binäre Inhalte wie ActiveX, Java oder Flash bleiben hier vollständig außen vor. Mögliche Konsequenz ist, dass das Gateway für Seiten, bei denen diese Kompatibilitätsprobleme auftreten, keine entsprechende stellvertretende Authentifikation für alle Zugriffe durchführen kann, da manche Zugriffe am Gateway vorbei direkt an den Server geleitet werden.

Ein weiteres Problem betrifft die Nutzung von Java-Applets: Alle authentifizierten Zugriffe auf externe Inhalte erfolgen aus Sicht des Clients über die Adresse des Gateways. Java-Applets, die bedingt durch die Java-Sandbox-Architektur nur TCP/IP-Verbindungen zu dem Host aufbauen dürfen, von dem der Client zuvor das entsprechende Java-Applet heruntergeladen hat, können nun keine Verbindungen mehr zu dem ursprünglichen Server aufbauen, da das Applet aus Client-Sicht vom HTTP-Gateway und nicht vom Server stammt.

3.3.1.2. HTTP-Proxy

Der HTTP-Proxy basiert auf dem in [Fie99] definierten HTTP-Proxy-Standard, der Erweiterungen des HTTP-Protokolls zum stellvertretenden Abruf von HTTP-Ressourcen durch einen so genannten HTTP-Proxy-Server beinhaltet, sowie dem in Abschnitt 2.4 vorgestellten Verfahren zum Tunneln von TLS-Verbindungen über HTTP-Proxys. Allerdings sind gegenüber einem klassischen Proxy verschiedene Modifikationen nötig, damit der Proxy als zentrale Authentifikationsinstanz zwischen Client und Server agieren kann.

Verbindungsaufbau

Nach der Entgegennahme einer eingehenden Verbindung muss der Proxy prüfen, ob er für den Host-Namen des Ziel-Servers, den der Proxy vom Client als Teil der HTTP-Anfragenachricht mitgeteilt bekommen hat, im Credential-Store ein Credential gespeichert hat.

Ist dies nicht der Fall – handelt es sich also um den Abruf einer Ressource, für die der Proxy keine Authentifikation durchführen muss – leitet der Proxy die Verbindung im Falle einer HTTPS-basierten Verbindung wie ein gewöhnlicher HTTP-Proxy nach dem in Abschnitt 2.4 beschriebenen Verfahren an den Server weiter. Handelt es sich dagegen um eine HTTP-Anfrage, so wird die entsprechende URL vom Server stellvertretend für den Client abgerufen und an diesen weitergeleitet. Eingriffe in die übertragenen Daten sind in beiden Fällen nicht nötig.

Sind dagegen passende Credentials im Credential-Store hinterlegt, ist eine Authentifikation des Proxys gegenüber dem Server stellvertretend für den Client erforderlich.

Authentifikation bei geschützten Server-Ressourcen

Das Vorgehen bei einer stellvertretenden Authentifikation gegenüber dem Server unterscheidet sich in Abhängigkeit davon, ob die ursprüngliche Verbindungsanfrage des Clients eine TLS-geschützte HTTPS- oder eine konventionelle HTTP-Verbindung ist.

TLS-Verbindungen

Handelt es sich um eine TLS-gesicherte Verbindung, muss der Proxy dazu in den TLS-Authentifikationsvorgang eingreifen, da er sonst keine Gelegenheit hat, den Nutzdatenstrom für eine HTTP-basierte Authentifikation (*Basic/Digest Authentication*, formularbasierte Authentifikation) zu verändern oder eine TLS-Client-Authentifikation gegenüber dem

Server durchzuführen. Den Datenstrom einfach mitlesen kann er nicht, da TLS eine Ende-zu-Ende-Sicherheit gewährleistet und dem Proxy zum Entschlüsseln des Datenstroms der zum Zertifikat des Servers gehörende private Schlüssel fehlt⁴, welcher sich nach dem aktuellen Stand der Technik glücklicherweise⁵ nicht aus dem öffentlichen Schlüssel ableiten lässt.

Folglich muss der Proxy die Ende-zu-Ende-Sicherheit der TLS-Verbindung am Proxy aufbrechen, was er durch Eingreifen in den TLS-Handshake realisiert. Während dieses Handshakes gibt sich der Proxy als eigentlicher Ziel-Server aus, damit er die Kommunikation mit dem Client entschlüsseln kann. Dazu liefert er an den Client statt des originalen Server-Zertifikats ein modifiziertes Zertifikat aus. Dieses veränderte Zertifikat entsteht durch Duplizieren des originalen Server-Zertifikats. In der Kopie tauscht der Proxy den darin gespeicherten öffentlichen Schlüssel gegen einen neuen öffentlichen Schlüssel aus, zu dem ihm der private Schlüssel bekannt ist. Hierdurch wird sichergestellt, dass der Proxy diesen Handshake durch Kenntnis des passenden privaten Schlüssels zum Zertifikat erfolgreich durchführen kann, wenn er sich während des Handshakes dem Client gegenüber als Server ausgibt. Nach dem Austausch des öffentlichen Schlüssels und einiger weiterer Attribute⁶ stimmt nun allerdings die vom Herausgeber unter dem Zertifikat angebrachte Signatur nicht mehr. Daher ist es erforderlich, dass der Proxy das Zertifikat neu signiert. Da er jedoch keinen Zugang zum Signaturschlüssel des ursprünglichen Herausgebers des Zertifikats hat, muss vor dem Erstellen einer neuen Signatur der Herausgeber des Zertifikats durch eine in den Proxy integrierte Certificate-Authority⁷ ersetzt werden. Diese signiert anschließend das Zertifikat mit ihrem eigenen Signaturschlüssel. Im Weiteren wird der gesamte Vorgang dieser Zertifikatsveränderung (siehe Abbildung 3.3) mit »Fälschen des Zertifikats« bezeichnet.

Nachdem der Proxy sich also während des Handshakes mit dem gefälschten Zertifikat dem Client gegenüber authentifiziert und als Ziel-Server ausgegeben hat, baut der Proxy nun eine Verbindung zum eigentlichen Ziel-Server auf. Es existieren nun – unbemerkt⁸ vom Client – zwei eigenständige Kommunikationsverbindungen, eine zwischen Client und Proxy, der gegenüber dem Client die Rolle des Servers einnimmt, und eine zwischen Proxy und dem eigentlichen Ziel-Server.

⁴Genau genommen fehlt ihm primär das so genannte *Master Secret*, das zu Beginn des Handshakes zwischen den beiden Kommunikationspartnern vereinbart wird. Um dort jedoch die Rolle des Servers einnehmen zu können, müsste der Proxy dessen privaten Schlüssel kennen.

⁵Andernfalls wären gängige Public-Key-Kryptoverfahren unbrauchbar.

⁶Dies sind z. B. die Seriennummer und optional der Gültigkeitszeitraum des Zertifikats. Auch machen nicht alle X.509-Erweiterungen im Kontext des neuen Herausgebers Sinn. Eine Liste von Erweiterungen, die nicht in das gefälschte Zertifikat übernommen werden sollten, findet sich in der Dokumentation der Realisierung des HTTP-Proxy in Abschnitt 4.6.1.

⁷Diesem neuen Aussteller muss der Client zumindest indirekt vertrauen, wenn der Nutzer beim Verbindungsaufbau per Webbrowser keine Warnung über eine unterbrochene Vertrauenskette erhalten soll. Sofern die Nutzergruppe, die auf den Proxy zugreifen soll, also nicht bereits z. B. in eine organisationsweite Public-Key-Infrastruktur eingebunden ist, ist die einmalige Installation eines entsprechenden Root-Zertifikats als Vertrauensanker auf den Clients unumgänglich.

⁸Aus Sicht des Clients stellt sich die Verbindung – abgesehen vom veränderten Herausgeber des Server-Zertifikats – wie eine über den Proxy getunnelte direkte TLS-Verbindung zum Ziel-Server dar.

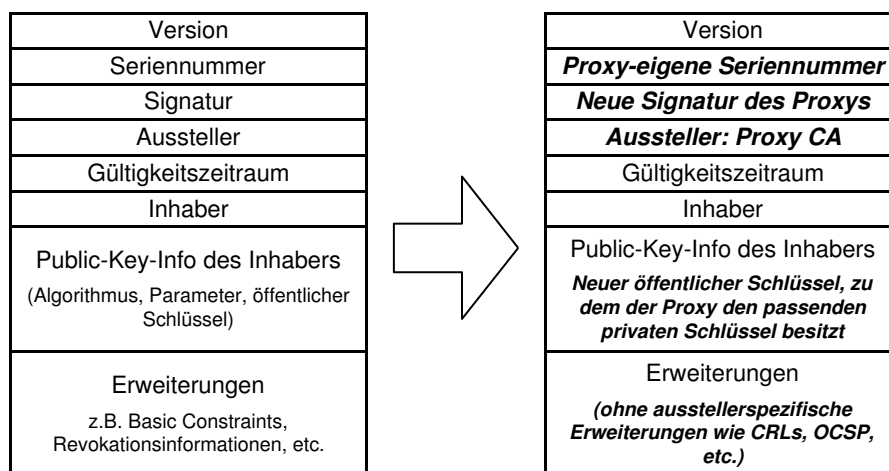


Abbildung 3.3: „Fälschen“ eines X.509-Serverzertifikats durch den Proxy

Anschließend bekommt der Proxy vom Client die relative URL der abzurufenden Ressource mittels einer in der TLS-Verbindung getunnelten konventionellen HTTP-Anfragenachricht übermittelt.

HTTP-Verbindungen

Handelt es sich dagegen um eine HTTP-Verbindung, bekommt der Proxy die diesmal absolute URL der abzurufenden Ressource direkt nach Annahme der Verbindungsanfrage des Clients in einer HTTP-Anfragenachricht übermittelt.

Abruf der Ressource

Daraufhin authentifiziert sich der Proxy dem Ziel-Server gegenüber bei Bedarf mit dem passenden Credential, das er aus dem Certificate-Store abrufen, fordert im Anschluss die Ressource beim Ziel-Server an und leitet sie über die bestehende Verbindung zwischen Client und Proxy an den Client weiter, womit der authentifizierte Abruf der Ressource beendet ist.

Authentifikation des Clients

Als Authentifikationsmechanismus des Endbenutzers gegenüber dem Proxy sind wie beim HTTP Gateway ebenfalls alle in Abschnitt 2.2 dargestellten Verfahren denkbar, wobei bei Nutzung von *Basic*- oder *Digest Authentication* eine Abwandlung, die so genannte Proxy Authentication [Fra99, Abschnitt 3.6] zum Einsatz käme. Bei formularbasierter Authentifikation wäre ein geeignetes Session-Tracking zu implementieren, das es dem Nutzer ermöglicht, nach möglichst einmaliger Authentifikation die Dienste des Proxys bis zum Ablauf eines Zeitlimits zu nutzen, ohne ständig erneut das Authentifikationsformular ausfüllen zu müssen. Die TLS-Client-Authentifikation ist für TLS-basierte HTTPS-Verbindungen unproblematisch umzusetzen, erfordert allerdings

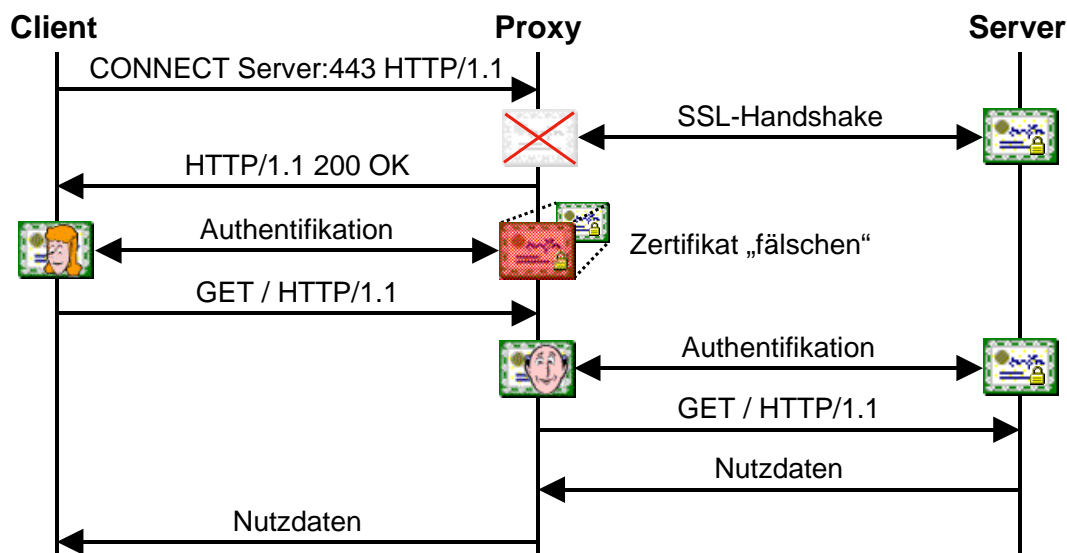


Abbildung 3.4: Kommunikationsablauf bei Nutzung des HTTP-Proxy

für die Authentifikation bei eigentlich HTTP-basierten Verbindungen etwas Zusatzaufwand durch Umleiten der HTTP-Verbindung auf eine TLS-gesicherte Verbindung, auf der sich eine zertifikatsbasierte Client-Authentifikation durchführen lässt.

Zusammenfassung

Abbildung 3.4 zeigt einen möglichen Kommunikationsablauf unter Einbeziehung von Client, Proxy und Server.

3.3.1.3. Application-Server

Der Application-Server soll an dieser Stelle lediglich der Vollständigkeit halber erwähnt werden. Weiterführende Details zu diesem Ansatz finden sich in [Gin04].

Die Application-Server-Architektur basiert auf einem zentralen Serversystem, auf dem ein Webbrowser installiert ist, der über native Mechanismen⁹ die zur Authentifikation notwendigen Credentials verwaltet. Der Endbenutzer meldet sich per Remote Access¹⁰ bei diesem System an und kann im Anschluss durch Nutzung der dort installierten Browser-Umgebung die vorkonfigurierten Credentials nutzen.

Um eine effiziente Administration der Credentials durchzuführen, bedarf es spezieller Tools, die zum jetzigen Zeitpunkt allerdings nicht existieren.

Probleme

Problemfelder bei dieser Architektur sind die Realisierung eines zuverlässigen Schutzes der Credentials, deren effiziente Administration sowie die für den Endbenutzer ungewohn-

⁹z. B. Windows Protected Storage oder Mozilla Password Manager

¹⁰z. B. Windows Terminal Services, X-Forwarding, VNC, NoMachine NX

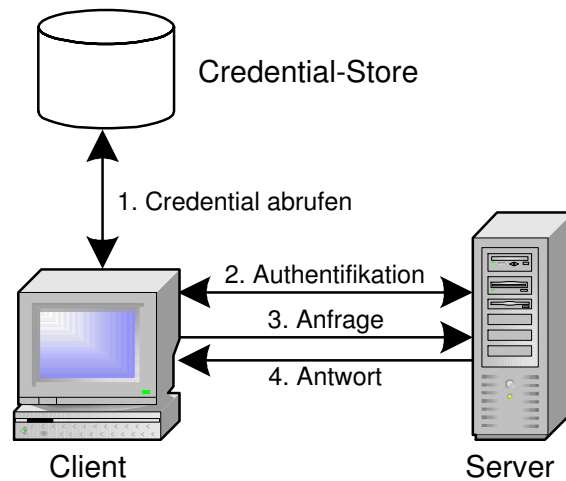


Abbildung 3.5: Schematischer Ablauf der Authentifikation bei einer clientbasierten Architektur

te Remote-Arbeitsumgebung. Letztgenannte verfügt einerseits möglicherweise über ein anderes *Look-and-Feel* als die primäre Arbeitsumgebung des Anwenders. Andererseits führt die eingesetzte Remote-Technologie zu Zeitverzögerungen bei der Annahme von Benutzereingaben durch den Server sowie der Übertragung von veränderten Bildschirm-inhalten zum Client.

3.3.2. Clientbasierte Architektur

Ein clientbasierter Realisierungsansatz ist ebenfalls ausführlich in [Gin04] dokumentiert. Da es sich durchaus um eine interessante Architektur handelt, sollen deren charakteristische Eigenschaften an dieser Stelle jedoch noch einmal herausgearbeitet werden.

Im Gegensatz zu den im vorangegangenen Abschnitt vorgestellten Stellvertreter-Architekturen kommt der clientbasierte Ansatz (siehe Abbildung 3.5) ohne eine Middleware-Komponente aus und bietet damit echte Ende-zu-Ende-Sicherheit.

Dabei kommt auf Seiten des Clients ein speziell angepasster Webbrowser zum Einsatz, der statt der üblicherweise lokalen Verwaltungsmöglichkeit¹¹ für Credentials über eine Anbindung an einen zentralen Credential-Store verfügt.

Erfolgt nun ein Zugriff auf eine geschützte WWW-Ressource, fordert der Browser das zugehörige Credential vom zentralen Credential-Store an und fährt mit der Authentifikation – transparent für den Nutzer – fort. Selbstverständlich ist eine starke Authentifikation des Clients gegenüber dem Credential-Store notwendig. Da ebenfalls ein hoher Schutzbedarf für die vom Credential-Store zum Client übertragenen Credentials besteht, bietet sich hier eine Absicherung der Verbindung mittels TLS an. Die Authentifikation des Clients kann dann per X.509-Zertifikat erfolgen.

Zur Vermeidung einer kompletten Neuentwicklung ließe sich die dargestellte Architek-

¹¹Windows Protected Storage oder Mozilla Password Manager

tur als Browser-Plugin oder *Browser Helper Object* für den Microsoft Internet Explorer oder mit Hilfe einer angepassten Version des Mozilla Web Browsers realisieren. Ein alternativer Realisierungsansatz könnte auf einem Wrapper um den TCP/IP-Stack oder unter Windows auf einem Wrapper um die WININET.DLL¹² auf dem Client aufbauen.

Einer der offensichtlichen Vorteile dieses Ansatzes ist der hohe clientseitige Integrationsgrad mit dem Webbrowser.

Jedoch steht diesem Vorteil ein gravierendes Problem gegenüber: Die Credentials verlassen den Schutzbereich des Credential-Stores und sind damit auf dem Client angreifbar. Leider stellt dieser Schwachpunkt sowohl die Revozierbarkeit einer Delegation als auch die Verhinderung einer unkontrollierten Weitergabe von Credentials in Frage.

3.4. Vergleich und Bewertung

Im Folgenden sollen die Vor- und Nachteile der in den vorangegangenen Abschnitten vorgestellten Architekturvarianten herausgestellt und bewertet werden. Um diese Bewertung vergleichbar und nachvollziehbar zu gestalten, erfolgt anschließend eine Gegenüberstellung der einzelnen Architekturen in einem tabellarischen Bewertungsschema. Abschließend wird auf Basis der Bewertung eine Architektur für die spätere prototypische Realisierung ausgewählt.

3.4.1. Bewertungskriterien

Die Bewertung der Realisierungsvarianten erfolgt in verschiedenen Kategorien: Kompatibilität, Integration, Benutzbarkeit, Sicherheit, Deployment, sowie Realisierungsaufwand. Die Grundlage für die Bewertung in den Kategorien Sicherheit und Benutzbarkeit entstammt [Gin04].

Für jede der Kategorien erfolgt nun eine kurze einführende Begriffsklärung gefolgt von einer Bewertung der einzelnen Architekturen.

3.4.2. Kompatibilität

Kompatibilität bezeichnet in diesem Kontext die Unterstützung von (Industrie-)Standards sowie die reibungslose Zusammenarbeit mit bestehenden (heterogenen) Client-Umgebungen.

Application-Server

Bei der auf dem Application-Server installierten Software handelt es sich um einen gewöhnlichen Webbrowser, der als solcher gängige Standards unterstützt. Abstriche bei der Kompatibilität zu Plugins oder ActiveX-Controls sind dann denkbar, wenn diese

¹²Bei der WININET.DLL handelt es sich um eine Komponente, die eine High-Level-API für den Zugriff auf HTTP(S)- sowie FTP-Ressourcen zur Verfügung stellt und auf der viele Windows-Anwendungen aufsetzen, so auch der Microsoft Internet Explorer.

administrative Rechte benötigen, die dem Endbenutzer auf dem Application-Server aus offensichtlichen Gründen nicht zur Verfügung stehen.

Weiterhin ist die Nutzung der auf dem Application-Server verwalteten Credentials grundsätzlich auf Anwendungen beschränkt, die direkt auf dem Application-Server ausgeführt werden. Eine entfernte Nutzung direkt auf einem Client-System ist nicht möglich.

HTTP-Gateway

Die Kompatibilität des HTTP-Gateways wird primär durch das nötige Umschreiben von Hyperlinks und Referenzen sowie die Nutzung eines nicht-standardisierten Verfahrens zum Aufrufen der gewünschten Website eingeschränkt. Neben allgemeinen Problemen beim Umschreiben der Seiteninhalte führt dies ebenfalls zu Inkompatibilitäten mit Java-Applets, die aus ihrer Sandbox lediglich Socket-Verbindungen zu dem Server aufbauen können, von dem der Client sie abgerufen hat. Weitere Probleme durch dieses Verfahren beim Einsatz von Scripting oder binären aktiven Inhalten sind denkbar.

HTTP-Proxy

Der HTTP-Proxy stellt in dieser Hinsicht einen deutlichen Fortschritt dar. Er unterstützt transparent den Aufbau von HTTP- sowie HTTPS-Verbindungen, solange im Client die Konfiguration eines Proxy-Servers vorgesehen ist. Dies ist für nahezu alle gängigen Clients gegeben. Plugins, Java-Applets oder Anwendungen von Drittanbietern stehen die Dienste des Proxys ebenfalls über ein standardisiertes Protokoll zur Verfügung.

Clientbasierte Architektur

Das vom clientseitigen Ansatz erreichbare Kompatibilitätsniveau ist stark von der konkret gewählten Realisierung abhängig. Eine auf einem modifizierten oder erweiterten Webbrowser aufbauende Lösung wird hier schlechter abschneiden als ein Wrapper für den systemeigenen TCP/IP-Stack oder z. B. die Windows-Bibliothek WININET.DLL, da sie ihre Dienste nicht Plugins oder Anwendungen von Drittanbietern zur Verfügung stellen kann.

3.4.3. Integration

Dieser Abschnitt befasst sich mit der Integration der verschiedenen Architekturen in die Arbeitsumgebung des Benutzers. Je nahtloser die Integration möglich ist, desto besser ist eine Lösung in Bezug auf ihre Anwenderfreundlichkeit zu bewerten. Damit hat diese Kategorie auch direkte Auswirkungen auf das im nächsten Abschnitt untersuchte Kriterium der Benutzbarkeit.

Application-Server

Auf Grund der vielfältigen Realisierungsmöglichkeiten für den Application-Server ist eine allgemeine Aussage über den erzielbaren Integrationsgrad nur schwer möglich. Kommt beispielsweise eine Remote-Desktop-Lösung wie VNC¹³ zum Einsatz, bei der der Benut-

¹³URL: <http://www.realvnc.com/>

zer auf einem vollständigen neuen Desktop arbeiten muss, ist die Qualität der Integration sicherlich schlechter zu bewerten als bei einer Lösung, die einzelne Fenster der entfernten Arbeitsumgebung auf dem Desktop des Nutzers darstellen kann, wie dies z. B. das X11-Forwarding beherrscht.

HTTP-Gateway

Im Gegensatz dazu erlaubt es das HTTP-Gateway dem Nutzer in jedem Fall, mit seiner gewohnten Arbeitsumgebung weiterzuarbeiten. Der Integrationsgüte abträglich ist allerdings die zum Zugriff auf authentifizierte Websites notwendige Nutzung eines HTML-Formulars im Gegensatz zum üblichen Verfahren über die Adressleiste des Webbrowsers. Auch die notwendigen Veränderungen der URLs abgerufener Websites führen hier zu einer Abwertung.

HTTP-Proxy

Der HTTP-Proxy arbeitet, einmal konfiguriert, weitgehend unbemerkt vom Anwender. Lediglich eine Authentifikation des Clients gegenüber dem Proxy ist erforderlich. Kommt als Authentifikationsverfahren eine Authentifikation per X.509-Zertifikat zum Einsatz, lassen sich die meisten Browser jedoch so konfigurieren, dass diese Authentifikation ebenfalls unbemerkt durch den Anwender vollautomatisch erfolgt.

Clientbasierte Architektur

Gelingt es bei der clientbasierten Architektur, die Authentifikationslösung in den gewöhnlich vom Anwender eingesetzten Webbrowser zu integrieren, stellt diese eine optimale Lösung in Bezug auf die Integration in die Client-Umgebung dar. Kommt dagegen ein anderer Browser zum Einsatz, führt dies subjektiv aus Sicht des Anwenders zu Abstrichen in Bezug auf die Güte der Integration, auch wenn das System objektiv betrachtet alle Anforderungen an eine gute Integration erfüllt.

3.4.4. Benutzbarkeit

Benutzbarkeit wird in der Literatur [Din02] häufig formal als Zusammenwirken von verschiedenen Eigenschaften eines Systems definiert: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlertoleranz, Individualisierbarkeit, Lernförderlichkeit sowie Datensicherheit.

Ein gegebenes System erschöpfend in Bezug auf diese Eigenschaften zu analysieren ergibt bereits Stoff für eine eigenständige Arbeit, so dass an dieser Stelle ein weniger formaler Ansatz zur Bewertung der Benutzbarkeit der verschiedenen Architekturen gewählt wurde. Dabei werden bei der Untersuchung der Architekturvarianten jedoch durchaus einige der obigen Eigenschaften aufgegriffen.

Application-Server

Die Benutzbarkeit des Application-Servers steht und fällt mit der dort vorzufindenden Arbeitsumgebung. Entspricht diese den Erwartungen des Anwenders in Bezug auf die installierte Web-Browsing-Umgebung sowie das *Look-and-Feel*, so ist von einer höheren Benutzbarkeit auszugehen, als wenn der Anwender eine völlig fremde Arbeitsumgebung

vorfindet.

Ein weiterer Faktor mit Einfluss auf die Benutzbarkeit dieser Architektur ist die Latenz bei Benutzereingaben und Systemausgaben. Da auf den Application-Server in der überwiegenden Zahl der Fälle über ein Netzwerk und nicht lokal zugegriffen wird, wirkt sich die Netzwerklatenz negativ auf das flüssige Arbeiten des Anwenders mit der auf dem Server laufenden Anwendung aus.

HTTP-Gateway

Das HTTP-Gateway weist zwar nicht die Latenzprobleme der Application-Server-Lösung auf, da hier die Anwendung, mit welcher der Benutzer direkt interagiert, konventionell auf dem lokalen Endbenutzersystem installiert ist, jedoch konfrontiert dieser Ansatz den Benutzer in anderer Hinsicht mit einem ungewohnten Verfahren zum Zugriff auf geschützte Websites: Statt wie gewohnt über die Adressleiste des Webbrowsers erfolgt hier die Angabe der abzurufenden Seite über ein HTML-Formular. Die Qualität des Feedbacks nimmt ebenfalls dadurch ab, dass aus der Adressleiste des Browsers später nicht mehr auf Anrieb erkennbar ist, auf welcher Website sich der Anwender befindet. Gleiches gilt für einen Indikator für sichere Verbindungen (*Padlock Icon*), der je nach eingesetzter Authentifikationstechnologie gegenüber dem Gateway immer aktiv ist und daher keine Rückschlüsse mehr auf die Vertraulichkeit der Datenübertragung zum eigentlichen Server zulässt.

HTTP-Proxy

Der HTTP-Proxy bietet hier Verbesserungen gegenüber dem HTTP-Gateway derart, dass die Navigation – wie üblich – über Adresszeile des Webbrowsers erfolgen kann, die auch später die korrekte URL der aufgerufenen Seite reflektiert. Der Benutzer kann seine gewohnte Arbeitsumgebung einsetzen. Die obigen Einschränkungen in Bezug auf die Aussagekraft des Sicherheitsindikators bleiben jedoch bestehen.

Clientbasierte Architektur

Bei der Benutzbarkeit der clientbasierten Architektur handelt es sich um eine zweiseitige Angelegenheit: Sofern eine Realisierung auf Basis des vom Endbenutzer gewohnten Webbrowsers erfolgt, ist von einer sehr guten Benutzbarkeit auszugehen, da der Nutzer seine gewohnte Arbeitsumgebung vorfindet, die völlig transparent die zusätzlichen Authentifikationsaufgaben ausführt.

Kommt stattdessen jedoch ein anderer Browser zum Einsatz, nimmt der Abstand zwischen den Erwartungen des Nutzers und der vorliegenden Arbeitsumgebung zu, was sich als Folge negativ auf die Benutzbarkeit auswirkt.

3.4.5. Sicherheit

Der Sicherheitsbegriff umfasst bei den folgenden Betrachtungen sowohl den Schutz der Credentials vor unberechtigter Nutzung und dem Ausspähen durch Unberechtigte als auch die Vertraulichkeit der im Rahmen der Authentifikationslösung durchgeführten Datenkommunikation.

Gemeinsamkeiten

Der Credential-Store ist eine allen vier Varianten gemeinsame Komponente. Durch seine zentrale Realisierung stellt er ein attraktives Angriffsziel dar, denn dort ließen sich durch einen einzigen Angriff alle gespeicherten Credentials in Erfahrung bringen. Jedoch bietet eine zentrale Speicherung der Credentials auch den Vorteil, dass sich daraus im Gegensatz zu einer verteilten Verwaltung der Credentials in einer heterogenen Client-Umgebung ein klar abgegrenzter Schutzbereich ergibt, der sich mit einfacheren Mitteln absichern lässt als eine verteilte Datenhaltung.

Gemeinsamkeiten der Stellvertreter-Architekturen

Application-Server, HTTP-Gateway und HTTP-Proxy haben alle das gemeinsame Sicherheitsproblem, dass sie SSL-geschützte Kommunikationsverbindungen auf der Middleware-Komponente entschlüsseln müssen. Dies macht neben der unverschlüsselten Kommunikation auch jegliche über diese Komponente abgewickelte verschlüsselte Kommunikation verwundbar durch das Abhören durch externe Angreifer oder den Betreiber der Middleware.

Während gegen externe Angriffe in gewissem Umfang eine Absicherung durch technische Maßnahmen möglich scheint, bleibt als zentraler Schwachpunkt der Stellvertreter-Architekturen die auch aus Sicht des Datenschutzes äußerst bedenkliche Abhörmöglichkeit durch den Betreiber des Systems bestehen.

Application-Server

Einige gängige Application-Server-Lösungen verschlüsseln die Kommunikation zwischen Client und Application-Server nicht. Abhängig von der gewählten Plattform lässt sich dieses Problem jedoch relativ unkompliziert lösen, z. B. durch Tunneln der Verbindung per SSH-Port-Forwarding.

Weiterhin ist eine Absicherung der Laufzeitumgebung des Application-Servers gegen Angriffe durch berechtigte Nutzer des Systems auf die gespeicherten Credentials essenziell. Bedenkt man, dass auf dem Application-Server gängige Webbrowser zum Einsatz kommen, erscheint dieses Ziel nur schwer oder gar nicht zu erreichbar.

HTTP-Gateway

Das HTTP-Gateway ermöglicht keine Prüfung der ursprünglichen Eigenschaften des SSL-Zertifikats einer Website durch den Anwender, da es für die Kommunikation ein eigenes Zertifikat verwendet, das mit dem ursprünglichen keine Attribute gemeinsam hat. Die Realisierung einer verschlüsselten Kommunikation zwischen Client und Gateway ist dagegen möglich.

HTTP-Proxy

Auch beim HTTP-Proxy ist eine überwiegend verschlüsselte Kommunikation zwischen Client und Proxy realisierbar. Damit diese Sicherheit jedoch nicht nur für HTTPS-, sondern auch für HTTP-Verbindungen zur Verfügung steht, sind einige technische Tricks zum Umleiten der jeweiligen HTTP- auf eine per TLS geschützte HTTPS-Verbindung nötig.

Clientbasierte Architektur

Die Sicherheit der clientbasierten Architektur ist zugleich Stärke und Schwäche: Während dieser Ansatz echte Ende-zu-Ende-Sicherheit ohne zwischengeschaltete Middleware bietet, sind die Credentials bei dieser Variante deutlich schwächer geschützt als bei den anderen drei Realisierungsarten. Hier müssen nämlich die Credentials zur Nutzung vom Credential-Store auf den Client übertragen werden. Auch wenn sich diese Übertragung kryptografisch, z. B. per TLS, absichern lässt, so liegen die Credentials bei ihrer Nutzung auf dem Client doch unverschlüsselt im Arbeitsspeicher vor. Auch ist der Einsatz eines bössartigen Clients denkbar, um an die Daten der Credentials zu gelangen.

3.4.6. Deployment

Der Deployment-Aufwand umfasst den Aufwand für alle Arbeiten, die notwendig sind, um die jeweilige Architektur zu installieren und einzurichten sowie eine bestehende Nutzerbasis in das System zu migrieren. Der Aufwand zur Migration bestehender Benutzerkonten hängt dabei jedoch stark vom Integrationsgrad der eingesetzten Lösung mit der Systemumgebung ab und kann daher nicht allgemein abgehandelt und bewertet werden.

Application-Server

Der Deployment-Aufwand für den Application-Server ist dann minimal, wenn sich die zum Zugriff notwendige Client-Software bereits vorinstalliert auf den Clients befindet. Dies ist z. B. bei Windows-XP-Systemen mit dem Client des Windows-Terminal-Server und bei allen X-basierten Systemen durch das integrierte X-Forwarding gegeben. Kann auf keinen vorinstallierten Client zurückgegriffen werden, steigt der Aufwand signifikant an.

HTTP-Gateway

Abhängig vom eingesetzten Authentifikationsverfahren ist der Aufwand beim Deployment des HTTP-Gateways ebenfalls minimal, sofern beispielsweise bereits vorhandene Benutzerkonten integriert werden können oder bereits eine Public-Key-Infrastruktur zur Authentifikation per X.509-Zertifikat vorhanden ist.

Soll die Verbindung zwischen Client und Gateway per TLS geschützt werden, muss das X.509-Zertifikat weiterhin von einer Certificate-Authority ausgestellt werden, der der Client vertraut. Dies kann einerseits durch eines der vielen kommerziellen Trust-Center geschehen, deren Root-Zertifikate bereits standardmäßig in den verbreiteten Webbrowsern als Vertrauensanker konfiguriert sind. Andererseits ist auch die Nutzung eines selbst-signierten Zertifikats oder eines von einer organisationsinternen CA signierten Zertifikats denkbar. In den beiden letzten Fällen steigt der Deployment-Aufwand dabei leicht an, da das Zertifikat des Ausstellers auf den Clients möglicherweise erst als Vertrauensanker installiert werden muss.

HTTP-Proxy

Der HTTP-Proxy setzt zur korrekten Funktion eine Konfiguration der Proxy-Einstellungen auf dem Client-System voraus. Beim Einsatz des Microsoft Internet Explorer kann diese Konfiguration automatisch mit Hilfe des so genannten *Web Proxy Auto-Discovery*

Protocol (WPAD) [Gau99] erfolgen. Webbrowser, die auf Mozilla aufbauen, unterstützen diese Art der automatischen Konfiguration noch nicht. Stattdessen muss bei diesen Browsern manuell eine URL für die *Proxy Auto-Configuration* (PAC) [Net96] erfasst oder der Proxy-Server von Hand in den Einstellungen des Browsers eingetragen werden. Inzwischen ist jedoch eine Vorabversion eines Patches¹⁴ für Mozilla verfügbar, der den Browser um die Unterstützung von WPAD erweitert.

Da der HTTP-Proxy zum Eingreifen in TLS-Verbindungen X.509-Zertifikate „fälschen“ und signieren muss, ist es erforderlich, das Root-Zertifikat seiner Certificate-Authority als Vertrauensanker auf den Client-Systemen zu installieren. Sofern die Authentifikation der Nutzer per TLS-Client-Authentifikation erfolgt, bietet es sich an, das CA-Zertifikat zusammen mit dem Client-Zertifikat des Nutzers und dessen privaten Schlüssel in einem PKCS#12-Container [Rsa99] an die Nutzer zu verteilen. Sowohl der Internet Explorer als auch Mozilla importieren dieses dann bei der Installation des Benutzerzertifikats als Vertrauensanker.

Clientbasierte Architektur

Die clientbasierte Architektur hat im Vergleich zu den übrigen Lösungen den höchsten Aufwand beim Deployment, da hier die Installation einer angepassten Anwendung oder Systemkomponente auf dem Client-System erforderlich ist. Es existieren jedoch Deployment-Lösungen zum Automatisieren dieses Schrittes, die – sofern vorhanden – genutzt werden können.

3.4.7. Realisierungsaufwand

Aussagen über den Realisierungsaufwand eines Projekts, ohne dieses zuvor vollständig spezifiziert und konzipiert zu haben, sind immer mit Vorsicht zu genießen. Trotzdem soll an dieser Stelle eine grobe Abschätzung des relativen Aufwands einer Realisierung der verschiedenen Architekturen erfolgen.

Application-Server

Der Application-Server baut auf bereits verfügbarer Standard-Software auf, eine Realisierung stellt daher primär eine Integrationsaufgabe dar. Einzig die Verwaltungsfunktionalität für die Credentials erfordert hier eine Neuentwicklung. Daher ist der Realisierungsaufwand für diese Architektur als relativ gering einzuschätzen.

HTTP-Gateway und HTTP-Proxy

HTTP-Gateway und HTTP-Proxy weisen in ihrer Architektur diverse Gemeinsamkeiten auf, die einen ähnlichen Realisierungsaufwand bei beiden nahe legen. Das HTTP-Gateway benötigt bei der Implementierung des Umschreibens von Hyperlinks und Referenzen einen erhöhten Aufwand, während dieser beim HTTP-Proxy bei der Implementierung des Man-in-the-Middle-Angriffs auf TLS-Verbindungen anfällt. Insgesamt erfordern diese beiden Architekturen mit Sicherheit einen erheblich höheren Aufwand als die Application-Server-Variante, da es gilt, von Grund auf ein neues Softwaresystem zu entwickeln. Begrenzt wird der Aufwand aber wiederum dadurch, dass lediglich eine Serverlösung zu

¹⁴URL: http://bugzilla.mozilla.org/show_bug.cgi?id=28998

entwickeln ist, während auf Seiten der Clients Standardsoftware zum Einsatz kommen kann.

Clientbasierte Architektur

Die clientbasierte Architektur erfordert hingegen die Entwicklung sowohl einer Client-Komponente zur Durchführung der Authentifikation als auch die Entwicklung eines serverseitigen Credential-Stores. Daher ist hier von einem sehr hohen Entwicklungsaufwand auszugehen.

3.4.8. Zusammenfassende Gegenüberstellung

Zur besseren Vergleichbarkeit der Bewertungen der einzelnen Architekturen erfolgt in Tabelle 3.1 eine zusammenfassende, schematisierte Gegenüberstellung der Vor- und Nachteile der verschiedenen Realisierungsvarianten. Dabei kommt ein fünfstufiges, schulnotenartiges Bewertungssystem¹⁵ zur Anwendung.

	Gateway	Proxy	Application-Server	Client-Architektur
Kompatibilität	○	+	○	++/○*
Integration	○	+	+/- -*	++/-*
Benutzbarkeit	○	+	○	++
Sicherheit	+	+	-	-
Deployment	++/○*	+/○*	+/- -*	--
Realisierungsaufwand	○	○	+	--

Tabelle 3.1: Vergleich der verschiedenen Realisierungsvarianten

*abhängig von der gewählten Realisierung und Systemumgebung; siehe Text

3.5. Resümee

Alle beschriebenen Implementierungsvarianten haben offensichtlich ihre Vor- und Nachteile. Trotzdem soll im Folgenden lediglich einer der Ansätze beim weitergehenden Entwurf und der Implementierung eines Prototyps berücksichtigt werden. Die Wahl fiel dabei auf den HTTP-Proxy, der nach Ansicht des Autors einen guten Kompromiss darstellt. Er weist eine hohe Kompatibilität mit nahezu beliebigen Anwendungen auf, gewährleistet ein hohes Sicherheitsniveau und hat im Vergleich zu einem clientbasierten Ansatz einen noch recht moderaten Realisierungsaufwand. Dies macht Abstriche im Bereich des Deployment-Aufwands im Vergleich zum HTTP-Gateway und die etwas schlechtere Benutzbarkeit als bei einem clientbasierten Ansatz verschmerzbar.

¹⁵Die Bewertungssymbole haben folgende Entsprechungen: ++ sehr gut, + gut, ○ zufriedenstellend, - ausreichend, -- mangelhaft.

Implementierung

Im Rahmen dieser Arbeit wurde eine prototypische Implementierung für den im vorangegangenen Kapitel konzipierten HTTP-Proxy realisiert, die in diesem Kapitel näher erläutert werden soll.

Die Implementierung trägt den Namen *TLS Authentication Proxy* und erfolgte mit Hilfe der Java¹ 2 SE/EE 1.4, die im Gegensatz zu älteren Versionen direkt die *Java Secure Socket Extension*² (JSSE) und damit das SSL/TLS-Protokoll unterstützt.

Die kryptografischen Algorithmen stellt der FlexiProvider³ des Fachgebiets Theoretische Informatik der TU Darmstadt zur Verfügung, für das Handling von ASN.1-Datenstrukturen wird das CODEC-Paket⁴ der Fraunhofer-Gesellschaft genutzt. Für das Datenbank-Management-System (DBMS) wiederum fiel die Wahl auf PostgreSQL⁵, dessen JDBC-Treiber im Vergleich zum weiter verbreiteten MySQL unter einer deutlich weniger restriktiven Lizenz erhältlich ist. Als Servlet-Container für das Web-GUI kommt der J2EE-konforme jetty://-Webserver⁶ zum Einsatz.

Die Dokumentation der Klassen und Schnittstellen des Proxys liegt vollständig als Javadoc im Quellcode sowie als HTML vor. Daher soll in den folgenden Abschnitten nicht penibel jedes kleinste Detail, das bereits per Javadoc ausführlich dokumentiert wurde, erneut in epischer Breite erläutert werden. Vielmehr gibt der Rest des Kapitels eine Übersicht über die Interaktion der einzelnen Module sowie die der Implementierung zu Grunde liegenden Prinzipien. Für Details zu den implementierten APIs sei daher auf die Javadoc-Dokumentation verwiesen.

4.1. Modulare Architektur

Um eine einfache Wartbarkeit und Erweiterbarkeit des Codes zu gewährleisten, wurde der vorliegende Prototyp des Proxys als modulare Architektur realisiert. Abbildung 4.1 gibt einen ersten Überblick über die verschiedenen Modulgruppen, die nun im Folgenden näher erläutert werden.

¹URL: <http://java.sun.com/>

²URL: <http://java.sun.com/products/jsse/>

³URL: <http://www.flexiprovider.de/>

⁴URL: <http://www.semoa.org/misc/codec.html>

⁵URL: <http://www.postgresql.org/>

⁶URL: <http://jetty.mortbay.com/>

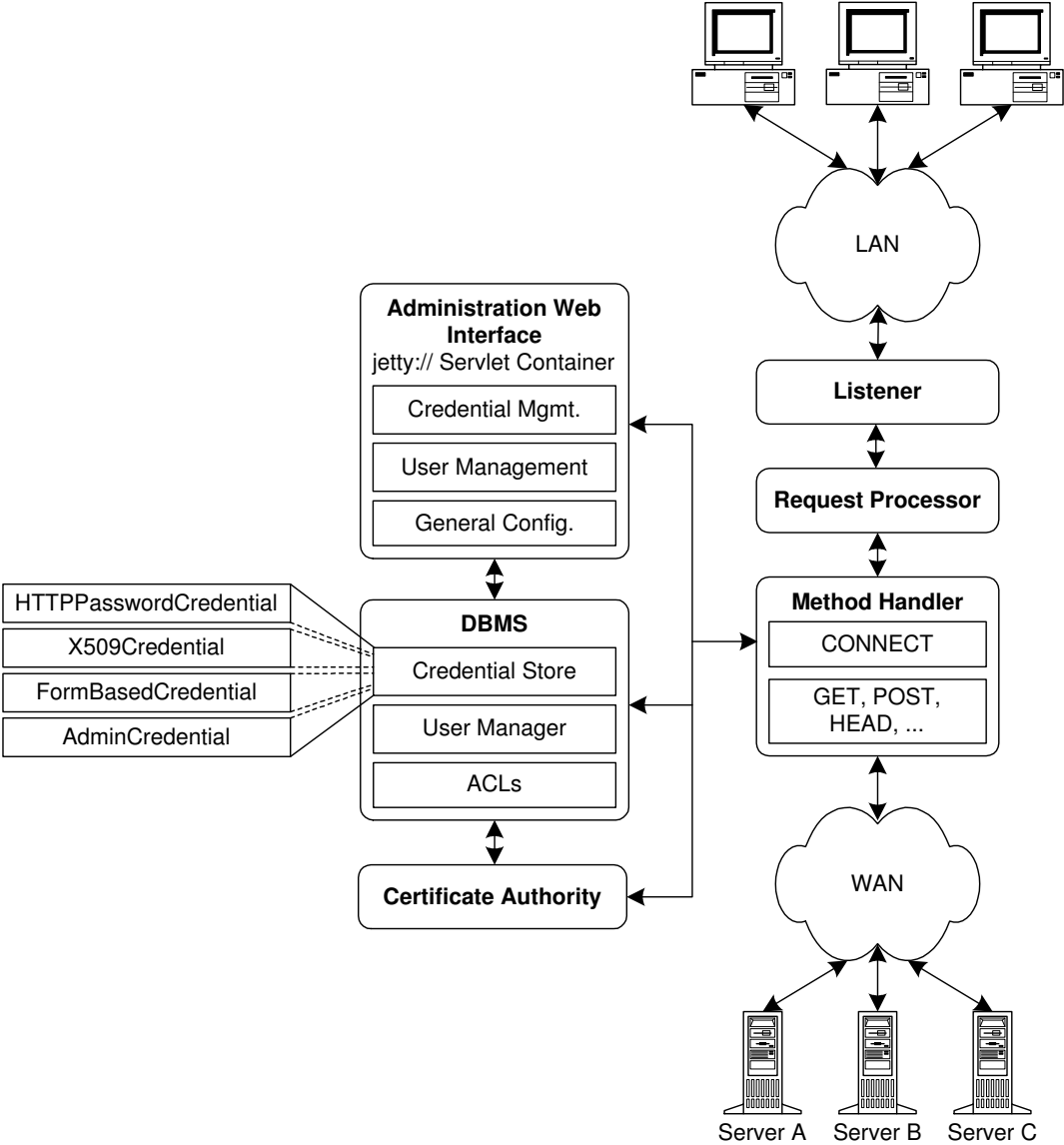


Abbildung 4.1: Modulare Architektur des Proxys

4.2. Networking

Diese Kategorie umfasst Komponenten, die für die Funktionalität des Proxys auf der Netzwerkebene verantwortlich sind.

4.2.1. Listener

Der so genannte Listener ist in der Klasse `tg.tlsauthproxy.Listener` realisiert. Seine Aufgabe besteht darin, auf einem vorgegebenen Port auf eingehende Verbindungen zu warten. Beim Eintreffen einer neuen Verbindungsanfrage nimmt der Listener diese entgegen, erstellt ein neues `RequestProcessor`-Objekt und übergibt diesem die Verbindung. Im Anschluss ist der Listener wieder zur Annahme der nächsten Verbindung bereit.

4.2.2. Request-Verarbeitung

Direkt bei der Erstellung einer neuen Instanz der Klasse `tg.tlsauthproxy.RequestProcessor` erstellt diese in ihrem Konstruktor einen neuen Thread. Hierdurch wird erreicht, dass der Konstruktor die Kontrolle so schnell wie möglich wieder an den Aufrufer abgeben kann, hier also den Listener.

Wenn der neu erstellte Thread des Request-Processors kurze Zeit später wieder die Kontrolle erhält, liest dieser über den zuvor vom Listener übergebenen `Socket` die HTTP-Anfragenachricht des Clients. Hierzu bedient er sich der Klasse `tg.tlsauthproxy.http.HTTPRequest`. Schließlich bestimmt er die für die Anfrage verwendete HTTP-Request-Methode, aus der er im Anschluss den Klassennamen des Method-Handlers ableitet, der daraufhin die Verarbeitung der Anfrage übernimmt. Findet der Request-Processor keinen spezialisierten Method-Handler für eine Request-Methode, kommt der Handler `tg.tlsauthproxy.methodhandler.Default` zum Einsatz.

4.2.3. Method-Handler

Bis auf den Default-Handler implementiert jeder Method-Handler genau eine HTTP-Request-Methode.

Connect-Handler

Der in der Klasse `tg.tlsauthproxy.methodhandler.Connect` implementierte Handler für die »CONNECT«-Methode realisiert das Man-in-the-Middle-Verfahren zum Eingreifen in SSL-Verbindungen. Nähere Details zu den Interna dieses Handlers liefert Abschnitt 4.6.1.

Default-Handler

Der Default-Handler verarbeitet alle übrigen Request-Methoden und kommt damit primär bei konventionellen HTTP-Anfragen zum Einsatz. Auch übernimmt er die Aus-

lieferung der für die automatische Proxy-Konfiguration notwendigen PAC-Datei (siehe Abschnitt 4.2.5).

HTTP-Proxy-Anfragen, also HTTP-Anfragen mit absoluter URL, für deren Server kein Credential im Credential-Store vorliegt, leitet der Default-Handler wie ein gewöhnlicher HTTP-Proxy-Server direkt an den Ziel-Server weiter, nachdem er die Zulässigkeit der Anfrage anhand deren Quelladresse überprüft hat.

Bei HTTP-Proxy-Anfragen, für deren Server hingegen Credentials vorhanden sind, ist zur Authentifikation des Benutzers eine Umleitung der Anfrage nach dem in Abschnitt 4.4.5 beschriebenen Verfahren auf eine HTTPS-URL nötig. Hier führt der Default-Handler also lediglich die Umleitung durch, die eigentliche Behandlung der Anfrage übernimmt im Anschluss der für HTTPS-Verbindungen zuständige Connect-Handler.

Der Default-Handler verarbeitet ebenfalls lokale HTTP-Anfragen. Über diese stellt er den Clients die inhaltsgleichen Dateien »/wpad.dat« und »/proxy.pac« zur automatischen Proxy-Konfiguration (vgl. Abschnitt 4.2.5) zur Verfügung. Ebenfalls bietet er unter der URL »/robots.txt« eine Datei zum Unterbinden der Indizierung durch Suchmaschinen gemäß dem Robots Exclusion Protocol [Kos04] an.

4.2.4. HTTP-Engine

Ein nicht unwesentlicher Teil des implementierten Codes dient der Umsetzung des HTTP-Protokolls. Ein Großteil dieser Implementierung befindet sich im Paket `tg.tlsauthproxy.http`. Die dort realisierten Klassen implementieren überwiegend HTTP-Anfrage (`tg.tlsauthproxy.http.HTTPRequest`) und -Antwort-Nachrichten (`tg.tlsauthproxy.http.HTTPResponse`), die selbst wiederum Hilfsklassen wie `tg.tlsauthproxy.http.HTTPHeader` zur internen Repräsentation der Datenstruktur der jeweiligen Nachricht nutzen. Auch die Umsetzung der Cookie-Unterstützung auf Header-Ebene ist hier realisiert, nämlich in der Klasse `tg.tlsauthproxy.http.HTTPCookie`.

Auf einer höheren Abstraktionsebene befindet sich der so genannte HTTP-Socket-Connector (für Anfragenachrichten in der Klasse `tg.tlsauthproxy.util.HTTPRequestSocketConnector`, für Antwortnachrichten in der Klasse `tg.tlsauthproxy.http.HTTPResponseSocketConnector` realisiert). Dieser arbeitet als eine Art Datenpumpe zwischen den Verbindungen zu Client und Server. Dabei werden die Daten jedoch nicht unverändert weitergeleitet, sondern es werden vor dem Weiterversand an den übermittelten HTTP-Headern die in Abschnitt 4.3.2 beschriebenen Modifikationen zur Authentifikation vorgenommen.

4.2.5. Proxy-Auto-Configuration

Um den administrativen Aufwand zur Konfiguration der Client-PCs möglichst gering zu halten, unterstützt der Proxy das so genannte *Web Proxy Auto-Discovery Protocol* (WPAD) [Gau99]. Dieses Protokoll ermöglicht eine automatische Konfiguration der Proxy-Server-Einstellungen auf dem Client, die momentan jedoch nur vom Microsoft

Internet Explorer unterstützt wird. Mozilla benötigt hingegen einen Patch, um WPAD zu unterstützen. Alternativ lassen sich beide Browser auch manuell mit der Adresse des Proxy-Servers oder der URL der *Proxy Auto-Configuration*-Datei (PAC) [Net96] konfigurieren, auf der auch das WPAD-Verfahren basiert. Bei der PAC-Datei handelt es sich prinzipiell um eine JavaScript-Datei, die dem Webbrowser abhängig von der aufgerufenen URL mitteilt, ob dieser die URL direkt vom Ziel-Server oder über einen Proxy-Server abrufen soll. Die Erstellung der PAC-Datei übernimmt dabei die Klasse `tg.tlsauthproxy.util.PAC`. Die Bearbeitung von Anfragen nach der PAC-Datei durch die Clients übernimmt der Standard-Method-Handler `tg.tlsauthproxy.methodhandler.Default`, der bei Anfragen nach den URLs »/wpad.dat« sowie »/proxy.pac« dynamisch eine PAC-Datei generiert und diese an den Client ausliefert.

Da der PAC-Datei während der Laufzeit eine nahezu unbeschränkte JavaScript-Laufzeitumgebung zur Verfügung steht, sind auch über die statische Vorgabe eines Proxy-Servers hinausgehende Szenarien denkbar. So nutzt die vorliegende Implementierung die vorhandenen Möglichkeiten, um bei entsprechender Konfiguration (siehe Anhang B.1) nur *die* Anfragen über den Proxy zu leiten, für die auch Credentials hinterlegt sind. Alle anderen Anfragen werden stattdessen direkt an den Ziel-Server gesendet. Primär führt dies zu einer Reduzierung der Last auf dem Proxy. Des Weiteren entsteht hierdurch der besonders aus datenschutzrechtlicher Sicht positive Nebeneffekt, dass für diese Verbindungen sowohl die Abhörmöglichkeit auf dem Proxy entfällt als auch keine Protokollierung der Anfragedaten auf dem Proxy mehr erfolgen kann.

Jedoch ergibt sich durch diese selektive Steuerung der Proxy-Nutzung ein Informationsleck: Damit das PAC-JavaScript ermitteln kann, welche Anfragen über den Proxy geleitet werden sollen, muss dieses prinzipiell Kenntnis davon haben, für welche Hosts der Proxy Credentials gespeichert hat. Da es sich bei JavaScript um normale Textdateien handelt, liegt diese Information nach dem Abruf der PAC-Datei im Klartext vor. Jedoch lässt sich diese Datei ohne vorherige Authentifikation abrufen, so dass die Host-Liste damit praktisch öffentlich verfügbar ist. Dies ist sicherlich nicht immer wünschenswert, so dass der Proxy ebenfalls eine Möglichkeit vorsieht, diese Informationen mit Hilfe einer kryptografischen Hashfunktion zu schützen.

Schutz der Host-Namen in der PAC-Datei

Die Idee hinter dem realisierten Verfahren ist, statt dem jeweiligen Host-Namen eines Servers einen Wert in der PAC-Datei zu speichern, der mittels einer Einwegfunktion berechnet wird. Als Einwegfunktion kommen z. B. gängige Hashverfahren wie der MD5-Algorithmus [Riv92] in Frage.

Sei S_i der Host-Name eines Servers, für den im Credential-Store ein Credential hinterlegt ist, und h eine kryptografische Hashfunktion, so berechnet der Proxy also bei der Erzeugung der PAC-Datei zuerst

$$H_i = h(S_i) \quad (4.1)$$

und schreibt diesen Wert H_i anschließend statt des Host-Namens S_i in die PAC-Datei. Da es sich bei der Hashfunktion um eine Einwegfunktion handelt, ist es aus Sicht

der Komplexitätstheorie schwer⁷, zu H_i wieder den ursprünglichen Server-Namen S_i zu berechnen.

Zur Laufzeit in der JavaScript-Umgebung des Webbrowsers kann das PAC-JavaScript nun den Namen eines Servers S , für den festgestellt werden soll, ob und über welchen Proxy der Zugriff gerouted werden soll, natürlich nicht mehr einfach mit den in der PAC-Datei gespeicherten Daten vergleichen. Diese enthält nämlich den ursprünglichen Host-Namen nicht mehr. Stattdessen muss die Funktion für jeden gespeicherten Hash-Wert prüfen, ob die Bedingung

$$H_i \stackrel{?}{=} h(S) \quad (4.2)$$

erfüllt ist. Sofern dies der Fall ist, wird die Verbindung im Anschluss über den Proxy geleitet, da ein entsprechendes Credential im Credential-Store existiert.

Implementiert wurde dieses Verfahren in der Klasse `tg.tlsauthproxy.util.PAC` unter Verwendung des MD5-Hashverfahrens. Für dieses existiert von Paul Andrew Johnston eine Portierung⁸ nach JavaScript, die als Teil der durch den Proxy generierten PAC-Datei zum Einsatz kommt.

4.3. Authentifikationsarchitektur

Die im Proxy implementierte Authentifikationsarchitektur unterstützt sowohl Public-Key-basierte Mechanismen als auch konventionelle Verfahren. Diese beiden Kategorien spiegeln sich auch in einer entsprechend unterschiedlichen Realisierung der verschiedenen Verfahren wieder.

In diesem Abschnitt werden ausschließlich die Authentifikationsmechanismen gegenüber anderen Servern betrachtet. Eine Erläuterung der Benutzerauthentifikation folgt in Abschnitt 4.4.5.

4.3.1. Public-Key-Verfahren

Als Public-Key-Verfahren unterstützt der Prototyp die TLS-Client-Authentifikation mittels X.509-Zertifikaten. Zur Realisierung einer Authentifikation während des TLS-Handshakes kommt ein spezieller Key-Manager zum Einsatz.

Weitere Details zur eigentlichen Umsetzung finden sich in der Dokumentation des X.509-Credentials im Abschnitt 4.4.3.

4.3.2. Konventionelle HTTP-Authentifikation

Für die Durchführung der Authentifikation bei konventionellen Verfahren, also der HTTP-Kennwort-Authentifikation und der Formular-Authentifikation, ist ein anderer

⁷und nach dem aktuellen Stand der Technik sogar praktisch unmöglich

⁸URL: <http://pajhome.org.uk/crypt/md5/>

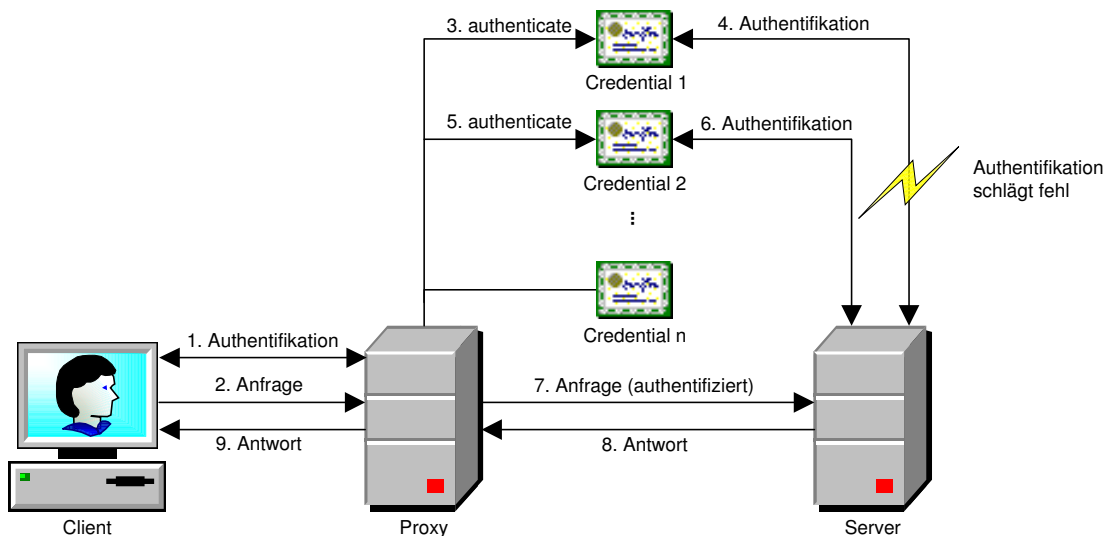


Abbildung 4.2: Schematischer Ablauf der HTTP-Authentifikation

Ansatz notwendig. Diese Verfahren arbeiten allgemein auf der Ebene des HTTP-Protokolls, z. B. durch Veränderung der Header von Anfrage- und Antwortnachrichten oder durch Modifikation der URL.

Um dies zu realisieren, ist während des Abrufs einer Ressource vom Ziel-Server sowohl vor dem Senden der Anfrage an den Server als auch nach dem Empfangen der Antwort eine Authentifikationsmöglichkeit vorgesehen. Zu beiden Zeitpunkten ruft der Proxy dazu der Reihe nach die Methode `authenticate` aller für den Ziel-Server hinterlegten Credentials auf. Ist ein Credential nicht in der Lage, für die angeforderte Ressource eine Authentifikation durchzuführen, zeigt es dies durch Werfen einer `AuthenticationFailureException` an. Der Proxy fährt dann analog mit dem nächsten Credential fort. Wirft der Methodenaufruf hingegen keine Exception, betrachtet der Proxy die Authentifikation als erfolgreich und fährt mit dem Abruf der Ressource fort, ohne für weitere Credentials eine Authentifikation zu versuchen.

Ein exemplarischer vereinfachter Ablauf der Authentifikation vor dem Versenden der Anfrage ist in Abbildung 4.2 dargestellt. Nach dem Empfang der Antwort vom Server wird die Authentifikationsfunktion der Credentials nach dem obigen Schema erneut aufgerufen, so dass die Credentials ihren Zustand bei Bedarf unter Berücksichtigung der Antwortnachricht aktualisieren können. Im Hinblick auf eine übersichtliche Darstellung ist dieser Teil der Authentifikation jedoch nicht abgebildet; in der Grafik wäre er zwischen den Ereignissen 8 und 9 einzufügen.

Für die formularbasierten Credentials ist zusätzlich die Implementierung eines Spezialfalls erforderlich: Da einige formularbasierte Authentifikationssysteme nach dem erfolgreichen Login eine Umleitung des Webbrowsers auf eine neue URL vornehmen, ist ein Mechanismus erforderlich, der es einem Credential ermöglicht, den Proxy zur Weiterleitung einer während des Logins empfangenen Umleitung an den Client zu veranlassen. Dies erfolgt durch das Werfen einer speziellen Exception: `Authentication-`

`RedirectionException`.

4.4. Backend

In der Kategorie *Backend* wurden alle Komponenten zusammengefasst, die zwar keine direkte Schnittstelle nach außen bereitstellen, jedoch ein integraler Bestandteil für die Funktion des Proxys sind.

4.4.1. Certificate-Authority

Da der Proxy bei SSL-Verbindungen Zertifikate fälschen und mit seinem eigenen privaten Schlüssel und Zertifikat signieren muss, verfügt er über eine kleine Certificate-Authority. Diese CA ist im Interface `tg.tlsauthproxy.pki.CertificateManager` spezifiziert und beispielhaft für PostgreSQL in der Klasse `tg.tlsauthproxy.pki.PostgreSQLCertificateManager` realisiert.

Der Proxy speichert während seiner Laufzeit die gefälschten Zertifikate im RAM zur erneuten Verwendung. Eine persistente Speicherung in der Datenbank erfolgt auf Grund der dynamischen Natur der Daten nicht. Das jeweilige Zertifikat wird nach einem Neustart des Proxys beim ersten Zugriff auf einen Host einfach neu erstellt. Das CA-Zertifikat und der zugehörige private Schlüssel werden dagegen persistent in der Datenbank vorgehalten.

Darüber hinaus stellt die CA ebenfalls Methoden zum Erstellen und Verlängern von Benutzerzertifikaten sowie dem Export derselben als PKCS#12-SafeBag zur Verfügung.

Datenbank-Realisierung

Die Certificate-Authority speichert in der Datenbank-Tabelle »ca« lediglich ihr Wurzelzertifikat unter dem Schlüssel »cert« sowie den zugehörigen privaten Schlüssel im Eintrag »key«.

Ein Auszug aus dem entsprechenden Datenbankschema findet sich in Abschnitt E.1.

4.4.2. Credential-Store

Im Credential-Store werden die zur Authentifikation benötigten Credentials zentral verwaltet. Er stellt die Schnittstelle zwischen den übrigen Modulen des Proxys und dem eigentlichen Datenspeicher, z. B. einem DBMS, dar.

Der Credential-Store stellt allgemeine Methoden zum Abrufen, Speichern und Löschen von Credentials zur Verfügung, genauso wie eine Methode, um für eine HTTP-Anfrage eine Authentifikation für alle zutreffenden Credentials durchzuführen. Diese zuletzt genannte Methode kommt u. a. für den bereits in Abschnitt 4.3.2 beschriebenen Authentifikationsmechanismus zum Einsatz.

Die Schnittstelle des Credential-Stores ist durch das Interface `tg.tlsauthproxy.credentials.CredentialStore` vorgegeben, eine Realisierung für PostgreSQL befindet sich in der Klasse `tg.tlsauthproxy.credentials.PostgreSQLCredentialStore`.

Datenbank-Realisierung

Die Realisierung des Credential-Stores speichert in der Tabelle »`credentials`« neben einer eindeutigen ID für jedes Credential dessen Host, Port, HTTP-Schema und Namen. Weitere Credential-spezifische Attribute werden in einer zusätzlichen Spalte mit Hilfe des im folgenden Abschnitt 4.4.3 beschriebenen Serialisierungsmechanismus gespeichert.

Ein Auszug aus dem Datenbankschema des Credential-Stores findet sich in Abschnitt E.3.

4.4.3. Credentials

Die vorliegende Implementierung unterstützt drei verschiedene Credential-Typen: HTTP-Kennwort-Credentials, X.509-Credentials sowie formularbasierte Credentials. Während die Eigenschaften der zu Grunde liegenden Authentifikationsmechanismen bereits in Abschnitt 2.2 behandelt wurden, gehen die folgenden Absätze auf spezifische Realisierungsdetails der unterschiedlichen Credential-Typen ein.

4.4.3.1. HTTP-Kennwort-Credentials

Die kennwortbasierte Authentifikation im Rahmen des HTTP-Protokolls basiert überwiegend auf den Mechanismen der *HTTP Basic Authentication* sowie der *HTTP Digest Authentication*. Beide Verfahren bauen in ähnlicher Weise auf Veränderungen der HTTP-Anfrage- und -Antwort-Header auf, so dass es sich anbietet, diese innerhalb derselben Credential-Klasse zu implementieren, sowie den in Abschnitt 4.3.2 erläuterten generischen Authentifikationsmechanismus zu nutzen.

Das Credential wird jeweils bei Antwortnachrichten des Servers aktiv, die mit dem Status-Code »401 Authorization Required« die Notwendigkeit einer Authentifikation anzeigen. Wird seine Authentifikationsfunktion mit einer entsprechenden Antwortnachricht aufgerufen, so ergänzt es die ursprüngliche HTTP-Anfragenachricht um einen »Authorization«-Header mit den Authentifikationsdaten und veranlasst im Anschluss einen erneuten Abruf der Ressource durch den Proxy.

Um bei der *Basic Authentication* nicht für jeden Seitenzugriff das vollständige Challenge/Response-Verfahren zur Durchführung der Authentifikation durchlaufen zu müssen, speichert das Credential zur Beschleunigung der Zugriffe in einem Cache, gegenüber welchen so genannten *Authentication Realms* es sich bereits erfolgreich authentifiziert hat. Bei nachfolgenden Zugriffen fügt es dann bereits vor dem Versand der Anfragenachricht an den Server automatisch die nötigen Authentifikations-Header in die Anfragenachricht ein, ohne erst die Antwort »401 Authentication Required« des Servers abwarten zu müssen.

Für die *Digest Authentication* lässt sich diese Optimierung nicht realisieren, da dort die Antwort des Clients aus einer veränderlichen (und nicht wie bei der *Basic Authentication* konstanten) Challenge abgeleitet wird, die dem Client erst bekannt ist, nachdem er sie vom Server empfangen hat.

4.4.3.2. X.509-Credentials

X.509-Credentials bestehen aus einem X.509-SSL-Client-Zertifikat sowie dem zugehörigen privaten RSA- bzw. DSA-Schlüssel. Da beide üblicherweise lediglich in einem maschinenlesbaren Format vorliegen und extern generiert werden, verfügt die Klasse `tg.tlsauthproxy.credentials.x509Credential` über eine Methode zum Importieren von PKCS#12-Containern, dem Industriestandard zum sicheren Transport von Benutzerzertifikaten und Schlüsseln.

Da die X.509-basierte SSL-Authentifikation während des SSL-Handshakes stattfindet, kann der weiter oben beschriebene generische Authentifikationsmechanismus, der bestimmte HTTP-Header nutzt, für diesen Credential-Typ nicht eingesetzt werden. Stattdessen kommen eine spezielle `SSLContextFactory` sowie ein spezieller `x509KeyManager` zum Einsatz, die dafür sorgen, dass der Proxy während des SSL-Handshakes auf eine Certificate-Request-Nachricht mit dem korrekten Client-Zertifikat antworten kann. Aufgabe der `UserSSLContextFactory` ist es dabei, einen `SSLContext` zu erstellen, der mit einem `ServerKeyManager` ausgestattet ist, der exakt Zugriff auf die Client-Zertifikate und Schlüssel eines Benutzers hat.

4.4.3.3. Formularbasierte Credentials

Formularbasierte Credentials ähneln den HTTP-Kennwort-Credentials in der Hinsicht, dass bei beiden Verfahren ähnlich geartete Geheimnisse, z. B. Benutzer/Kennwort-Tupel, zum Einsatz kommen. Die Realisierung setzt ebenfalls auf dem in Abschnitt 4.3.2 beschriebenen Authentifikationsmechanismus auf.

An dieser Stelle enden die Gemeinsamkeiten jedoch schon, denn während die Authentifikation mittels HTTP-Kennwort-Credentials verbindungsbasiert arbeitet, ist für die formularbasierte Authentifikation die Erweiterung des HTTP-Protokolls um ein Sitzungskonzept notwendig. Andernfalls müsste sich der Client mit jeder neuen Verbindung zum Server erneut authentifizieren, im schlimmsten Fall also mit jeder Anfragenachricht.

Diese Sitzungsverwaltung lässt sich auf unterschiedliche Weise realisieren; der vorliegende Prototyp implementiert davon zwei verschiedene Alternativen: Das Einbetten derselben eindeutigen *Session-ID* in die URLs aufeinanderfolgender Anfragenachrichten und die so genannten HTTP-Cookies [Kri00].

Das URL-basierte Verfahren lässt sich wiederum in zwei weit verbreitete Realisierungsvarianten aufteilen: Die eine Variante sendet nach der Übermittlung der Formulardaten und erfolgreicher Authentifikation per Status-Code »302 Found« in Verbindung mit einem »Location«-Header eine Umleitung an den Client. Die Ziel-URL der Umleitung enthält dabei eine *Session-ID*, welche die Sitzung eindeutig identifiziert. Die andere Variante führt nach der erfolgreichen Authentifikation keine Umleitung durch, sondern übermittelt an den Nutzer direkt die angefragten Inhalte. Damit Aufrufe von Folgeseiten ebenfalls der Sitzung zugeordnet werden können, bettet der Server in die Hyperlinks auf der an den Nutzer zurückgelieferten Seite eine *Session-ID* ein.

Der vorliegende Prototyp unterstützt das Sitzungsmanagement mittels Cookies sowie die umleitungsbasierte URL-Variante. Neue Sitzungen können entweder automatisch

beim ersten Zugriff eines Clients auf eine Ressource angelegt werden, bzw. wenn eine vorhandene Session inzwischen abgelaufen ist, sowie manuell beim Zugriff auf eine so genannte *Trigger-URL*⁹ aus dem Webbrowser des Clients.

Für das Sitzungsmanagement durch Cookies benötigt der Proxy einen Sitzungs-Cache. Vor jedem Seitenabruf auf einem Ziel-Server prüft der Proxy, ob im Sitzungs-Cache bereits ein Eintrag einer gültigen Session für das Ziel-System existiert. Ist dies nicht der Fall, legt der Proxy wie folgt eine neue Sitzung an, die er für nachfolgende Zugriffe im Sitzungs-Cache ablegt: Nachdem der Proxy eine Anfrage mit den übermittelten Formulardaten an den Server geschickt hat, antwortet dieser nach erfolgreicher Authentifikation mit einem Cookie. Diesen Cookie speichert der Proxy zusammen mit weiteren Sitzungsinformationen, um ihn bei allen weiteren Folgeanfragen im HTTP-Header wieder an den Server zu übermitteln, welcher anhand der in den Cookie eingebetteten Sitzungsinformationen die Sitzung wiedererkennt.

Da Sitzungen serverseitig gewöhnlich nur eine endliche Lebensdauer haben, lässt sich am Proxy für jedes formularbasierte Credential ein individueller Inaktivitäts-Timeout konfigurieren, nach dessen Ablauf auch der Proxy die Sitzung verwirft und bei Folgezugriffen automatisch eine neue Sitzung mit dem Server etabliert.

Die für die Authentifikation notwendigen Zusatzparameter¹⁰ speichert das Credential dabei zusammen mit den übrigen Authentifikationsdaten.

URL-basierte Sitzungen können in der vorliegenden Implementierung nicht automatisch beim ersten Zugriff auf einen Server erstellt werden, sondern erfordern zwingend die Nutzung einer Trigger-URL.

4.4.3.4. Admin-Credential

Das `AdminCredential` nimmt eine Sonderstellung ein: Im Gegensatz zu den übrigen Credentials kapselt es keine Authentifikationsgeheimnisse, sondern verleiht jedem Nutzer des Proxys, der auf das Credential zugreifen darf, Administratorrechte auf dem Proxy. Entsprechend gibt es auch nur genau eine Instanz dieses Credentials für den Host-Namen »proxyadmin«.

4.4.3.5. Serialisierung

Um eine Erweiterung um zusätzliche Credential-Typen ohne Änderungen am Datenbankschema zu ermöglichen, kommt für die Serialisierung der Credential-spezifischen Attribute in der Datenbank ein textbasiertes Format zum Einsatz. Ein neuer Credential-Typ muss dazu lediglich die von `tg.tlsauthproxy.credential.Credential` geerbte

⁹Bei der Trigger-URL handelt es sich um einen beliebigen Pfad auf dem Ziel-Server. Für die Website `www.example.com` könnte dies z. B. die URL `http://www.example.com/login` sein. Welche URL exakt genutzt wird, spielt dabei keine Rolle, der Pfad muss nicht einmal auf dem Ziel-Server existieren. Wichtig ist lediglich, dass der Nutzer Kenntnis von der Trigger-URL zum Anstoßen der Authentifikation hat.

¹⁰z. B. die für den automatischen Login aufzurufende URL sowie weitere üblicherweise im Login-Formular hinterlegte Parameter

Methode `getCredentialData` überschreiben und in dieser die neuen Attribute in Textform kodiert zusätzlich zu den Attributen der Superklasse zurückliefern.

4.4.4. Benutzerverwaltung

Die Benutzerverwaltung des Proxys stellt Methoden zur Verfügung, um Benutzerkonten mit den zugehörigen X.509-Zertifikaten zu erstellen und zu verwalten. Ebenfalls stellt die Benutzerverwaltung eine nutzerseitige Sicht auf die im Abschnitt 4.4.6 dargestellte Zugriffskontrolle bereit. Sie ist durch das Interface `tg.tlsauthproxy.backend.UserManager` spezifiziert und wird von der Klasse `tg.tlsauthproxy.backend.PostgreSQLUserManager` implementiert.

Die einzelnen Benutzer des Proxys sind dabei als Objekte der Klasse `tg.tlsauthproxy.backend.UserPrincipal` modelliert, die primär das X.509-Zertifikat des Nutzers als Java-Principal kapselt und um unterstützende Methoden zur Datenbank-Serialisierung und Zugriffskontrolle erweitert.

Datenbank-Realisierung

In der Tabelle »users« der Datenbank speichert die Benutzerverwaltung neben dem Namen eines Benutzers, einer eindeutigen ID und einem *Flag*, das anzeigt, ob das jeweilige Benutzerkonto aktiv ist, das X.509-Zertifikat des Nutzers.

Ein Auszug aus dem zugehörigen Datenbankschema findet sich in Abschnitt E.2.

4.4.5. Benutzerauthentifikation

Wie bereits zuvor erwähnt, authentifiziert der Proxy seine Benutzer per SSL-Client-Authentifikation, also anhand von X.509-Zertifikaten.

Dazu ist es erforderlich, dass der Benutzer sein Zertifikat im Certificate-Store seines Webbrowsers installiert. Wenn der Nutzer nun auf eine Website zugreift, für die der Proxy ein oder mehrere Credentials verwaltet, fordert der Proxy den Browser des Nutzers auf, sich per X.509-Zertifikat zu authentifizieren. Abhängig vom eingesetzten Browser und dessen Konfiguration erfolgt diese Authentifikation transparent oder der Browser fordert den Benutzer auf, die Nutzung des Zertifikats zu bestätigen. Falls mehrere Zertifikate für eine Authentifikation in Frage kommen, besteht in den meisten Fällen ebenfalls eine Auswahlmöglichkeit, welches Zertifikat zur Authentifikation zum Einsatz kommen soll.

Diese Authentifikation ist so lange ohne Schwierigkeiten möglich, wie für die Verbindung zum Ziel-Server eine SSL-geschützte HTTPS-Verbindung zum Einsatz kommt. Jedoch soll der Proxy seine Authentifikationsdienste auch für normale HTTP-Verbindungen anbieten. Das HTTP-Protokoll unterstützt aber keine X.509-basierte Client-Authentifikation, so dass an dieser Stelle – will man nicht auf ein anderes Authentifikationsverfahren ausweichen – ein Workaround nötig ist: Hierzu leitet der Proxy HTTP-Verbindungen, für deren Ziel-Server er über mindestens ein Credential verfügt, per Status-Code »302 Moved« auf eine speziell aufgebaute HTTPS-URL um. Dabei ersetzt er das ursprüngliche Protokoll-Schema »http« durch das entsprechende Schema für

SSL-Verbindungen »https« und fügt einen speziellen Port¹¹ in die URL ein¹², damit der Proxy bei folgenden Zugriffen diese als umgeleitete Zugriffe erkennen kann.

Wenn der Proxy schließlich eine Anfrage anhand des Ports als umgeleitet erkannt hat, baut er eine normale HTTP-Verbindung zum Ziel-Server auf, während zwischen Client und Proxy eine HTTPS-Verbindung zum Einsatz kommt. Da der Ziel-Port auf dem Server während der Umleitung verloren geht, ermittelt der Proxy den korrekten Ziel-Port anhand der jeweils im Credential hinterlegten Daten. Als Zertifikat sendet der Proxy an den Client ein spezielles Dummy-Zertifikat, das neben dem korrekten Host-Namen des Servers im *Distinguished Name* einen Warnhinweis darauf beinhaltet, dass die eigentliche Verbindung zum Server nicht per TLS gesichert ist.

4.4.6. Zugriffskontrolle

Der Proxy realisiert für die Nutzung der Credentials eine Zugriffskontrolle. Diese stellt einerseits sicher, dass nur Benutzer ein Credential zur Authentifikation gegenüber einem Server einsetzen können, denen der Zugriff auf dieses entweder durch einen Administrator oder mittels Delegation durch einen anderen Nutzer erlaubt wurde. Andererseits ermöglicht sie die weitere Einschränkung dieser berechtigten Nutzung durch so genannte Constraints. Die vorliegende Realisierung implementiert ein Constraint ohne Nutzungsbeschränkungen sowie ein Constraint, das die Nutzung eines Credentials nur im Zeitraum zwischen einem Start- und einem Enddatum zulässt.

Formal betrachtet basiert das für die Zugriffskontrolle genutzte Modell auf einer Zugriffsmatrix [Eck01] M , deren Spalten durch die Menge \mathcal{C} der Credentials und deren Zeilen durch die Menge \mathcal{B} der Benutzer des Proxys festgelegt werden. Die Menge \mathcal{E} beschreibt die möglichen Einschränkungen oder auch Constraints auf den Zugriffsrechten. M lässt sich dabei als Wertetabelle der Abbildung $\varphi : \mathcal{B} \times \mathcal{C} \mapsto \mathcal{E}$ interpretieren. Für einen Benutzer $b \in \mathcal{B}$ und ein Credential $c \in \mathcal{C}$ liefert $\varphi(b, c)$ das jeweilige Constraint, gibt also an, ob und mit welchen Rechten ein Benutzer auf ein Credential zugreifen darf.

Da Zugriffsmatrizen häufig nur dünn besetzt sind, verwenden praktische Umsetzungen häufig statt einer Matrixdarstellung eine zeilenweise (Zugriffskontrolllisten oder auch ACLs) oder spaltenweise Darstellung (Zugriffsausweise oder auch Capabilities). Auf den Proxy übertragen gibt eine ACL an, auf welche Credentials ein Nutzer zugreifen darf, eine Capability ordnet dagegen einem Credential die berechtigten Nutzer zu. Die APIs der Zugriffskontrolle des Proxys sind an die ACL-Darstellung angelehnt.

Die Zugriffskontrolle ist als Teil der Benutzerverwaltung (siehe Abschnitt 4.4.4) in das Interface `tg.tlsauthproxy.backend.UserManager` bzw. die Implementierung `tg.tlsauthproxy.backend.PostgreSQLUserManager` integriert. Die beiden Constraints sind von der abstrakten Basisklasse `tg.tlsauthproxy.credentials.ValidityConstraint` abgeleitet und in den Klassen `tg.tlsauthproxy.constraints.UnrestrictedValidityConstraint` und `tg.tlsauthproxy.constraints.DateValidityConstraint` implementiert.

¹¹Port 9443

¹²z.B. wird aus der URL `http://www.example.com/login` die neue URL `https://www.example.com:9443/login`.

Datenbank-Realisierung

Da das relationale Datenbankmodell als Teil einer Relation keine Listenstrukturen unterstützt, die für eine zeilen- oder spaltenweise Realisierung der Zugriffsmatrix jedoch nötig wären, erfolgte die Umsetzung der Zugriffsrechte als eine eigene Tabelle mit dem Namen »access«, in der die nicht-leeren Einträge der Zugriffsmatrix gespeichert sind. Eine Zeile dieser Tabelle enthält neben einem Verweis auf den Benutzer b und das Credential c die Nutzungseinschränkungen $\varphi(b, c)$. Daneben wird ein zusätzlicher Verweis auf den Besitzer des Zugriffsrechts gespeichert, der für die Delegation eines Credentials sowie deren Revokation relevant ist.

Ein Auszug aus dem Datenbankschema der Zugriffskontrolle findet sich in Abschnitt E.4.

4.4.7. Konfigurationsverwaltung

Für die Konfigurationsverwaltung nutzt der Proxy einen zweigleisigen Ansatz: Konfigurationsvariablen, die den Datenbankzugriff selbst oder lokale Pfade betreffen, sind im Basisverzeichnis der Proxy-Installation in der Datei »Proxy.ini« abgelegt, die automatisch bei der Erstkonfiguration des Proxys (siehe Anhang A) angelegt und interaktiv mit Werten gefüllt wird. Zum Zugriff auf diese Datei kommt die Klasse Java-Standardklasse `Properties` zum Einsatz.

Die Verwaltung aller übrigen Konfigurationsparameter erfolgt dagegen datenbankbasiert mit Hilfe der Klasse `tg.tlsauthproxy.admin.PostgreSQLConfigurationManager`, die das Interface `tg.tlsauthproxy.admin.ConfigurationManager` implementiert.

Datenbank-Realisierung

Für einen Konfigurationseintrag speichert die Konfigurationsverwaltung neben einem frei wählbaren Namen als Schlüssel die eigentlichen Konfigurationsdaten in einer Textrepräsentation sowie den ursprünglichen Datentyp¹³. Darüber hinaus wird eine kurze Beschreibung des Konfigurationseintrags gespeichert, der vom webbasierten Administrations-GUI zusammen mit den Konfigurationsdaten angezeigt wird. Ein weiteres *Flag* legt fest, ob die Konfigurationsvariable bei der Erstkonfiguration des Proxys erfasst werden soll.

Ein Auszug aus dem Datenbankschema der Konfigurationsverwaltung findet sich in Abschnitt E.5.

4.4.8. Logging

Der Proxy protokolliert während des Betriebs verschiedene Ereignisse: Allgemeine Statusnachrichten, Zugriffe auf die Administrationsschnittstelle sowie Zugriffe auf die eigentliche Proxy-Komponente.

¹³Integer, String oder Boolean

Allgemeine Statusnachrichten

Allgemeine Statusnachrichten verarbeitet der Prototyp über das Interface `tg.tls-authproxy.Logger`. Die konkret im vorliegenden Prototyp für das Logging eingesetzte Klasse `tg.tlsauthproxy.ConsoleLogger` schreibt die zu protokollierende Meldung zusammen mit der aktuellen Uhrzeit und dem Namen des betroffenen Moduls auf die Konsole.

Administrative Zugriffe

Zugriffe auf die Administrationsschnittstelle protokolliert der Proxy über die im verwendeten `jetty://`-Servlet-Container integrierte Logging-Funktionalität. Diese protokolliert Zugriffe in der Datei `»/var/log/admin.log«` unter dem Basisverzeichnis der Proxy-Installation. Als Format für die Log-Datei kommt das *NCSA Extended/Combined Log Format*¹⁴ zum Einsatz. In diesem werden neben der jeweils angeforderten Ressource, dem authentifizierten Nutzer, Datum und Uhrzeit der Anfrage, Status-Code und Größe der Antwort auch der Referrer, also die verweisende URL, sowie der *User-Agent* des Clients protokolliert.

Proxy-Zugriffe

Für die Protokollierung von Proxy-Zugriffen kommt ebenfalls das *NCSA Extended/Combined Log Format* zum Einsatz. Die entsprechende Log-Datei `»access.log«` befindet sich im Verzeichnis `»/var/log/«` unter dem Basisverzeichnis der Proxy-Installation. Wie bei administrativen Zugriffen wird für jede authentifizierte Anfrage der Name des Benutzers protokolliert, der die Anfrage initiiert hat. Die Logging-Schnittstelle ist im Interface `tg.tlsauthproxy.backend.AccessLog` spezifiziert, die Implementierung für das *NCSA Extended/Combined Log Format* befindet sich in der Klasse `tg.tls-authproxy.backend.CommonAccessLog`.

4.5. Administrations-GUI

Die Administrationsschnittstelle des Proxys wurde im Hinblick auf eine möglichst hohe Portabilität als webbasierter Ansatz realisiert; alle administrativen Aufgaben lassen sich also per Webbrowser über eine HTML-Oberfläche durchführen. Die Authentifikation gegenüber diesem Administrations-GUI erfolgt dabei mit Hilfe derselben Mechanismen, wie sie auch bei der Nutzung eines Credentials zum Zugriff auf eine geschützte Website zum Einsatz kommen; der Benutzer oder Administrator authentifiziert sich also auch hier mit seinem X.509-Zertifikat. Um das Administrations-GUI zur Verwaltung von Nutzern, Credentials und Zugriffsrechten einsetzen zu dürfen, benötigt ein Nutzer das Zugriffsrecht auf das spezielle Credential `»Administrative Privileges (proxyadmin)«`. Besteht kein Recht zur Verwendung dieses Credentials, darf ein Nutzer im Administrations-GUI

¹⁴URL: <http://hoohoo.ncsa.uiuc.edu/docs/setup/httpd/LogOptions.html>

lediglich einsehen, auf welche Credentials er Zugriff hat, und – unter den entsprechenden Voraussetzungen – selbst Delegationen durchführen. Der Zugang zum Administrations-GUI ist nur direkt über den Proxy unter der URL `https://proxyadmin/` möglich.

Erkennt der bereits in Abschnitt 4.2.3 beschriebene Connect-Handler einen Zugriff auf einen Host mit dem Namen »proxyadmin«, so gibt er die Verbindung an den Method-Handler des Administrations-GUIs, `tg.tlsauthproxy.admin.AdminMethodHandler` weiter. Dieser authentifiziert im Anschluss den Client und übergibt die Verbindung schließlich an die Klasse `tg.tlsauthproxy.admin.AdminServer`, welche die Schnittstelle zum eingebetteten `jetty://`-Webserver bereitstellt. Dieser übergibt die Kontrolle anschließend – je nach aufgerufener URL – an eines der Servlets aus dem Paket `tg.tlsauthproxy.admin.servlet`. Das jeweilige Servlet führt die eigentliche Interaktion mit dem Nutzer durch Ausgabe der Administrationsseiten und die Entgegennahme und Umsetzung von Formulardaten durch. Die Servlets greifen dabei wiederum auf zentrale Komponenten des Proxys wie den Credential-Store, die Benutzerverwaltung, die Zugriffskontrolle und die Konfigurationsverwaltung zu, um ihre Verwaltungsaufgaben umzusetzen.

4.6. Interna

Dieser Abschnitt soll die Implementierungsdetails einiger für die Funktion des Proxys besonders wichtiger Komponenten beleuchten. Besonderheiten werden jeweils direkt am Quellcode aufgezeigt.

4.6.1. Funktionsweise der Man-in-the-Middle-Komponente

Die Man-in-the-Middle-Komponente, die ein aktives Eingreifen des Proxys auch in eigentlich vertrauliche, Ende-zu-Ende-gesicherte SSL/TLS-Verbindungen ermöglicht, nimmt eine zentrale Rolle für die Funktion des Proxys ein. Die Funktionsweise dieser wichtigen Komponente soll daher im Folgenden detailliert erläutert werden.

Die Kernfunktionalität dieser Komponente ist in der Klasse `tg.tlsauthproxy.methodhandler.Connect` realisiert. Deren Methode `handleRequest` wird bei jeder am Proxy eintreffenden »CONNECT«-Anfrage aufgerufen und führt dann den Man-in-the-Middle-basierten Verbindungsaufbau durch.

Zu Beginn wertet die Methode die Anfrage-URI aus und teilt diese in ihre Komponenten, Ziel-Host und Port, auf. Im Anschluss prüft die Methode anhand des gerade ermittelten Host-Namens, ob es sich um eine Anfrage für die Administrationskomponente handelt. Diese Anfragen werden in einer separaten Klasse `tg.tlsauthproxy.AdminMethodHandler` verarbeitet:

```
if ("proxyadmin".equalsIgnoreCase(host)) {
    AdminMethodHandler handler = new AdminMethodHandler();
    handler.handleRequest(request, socket);
    return;
}
```

Nachfolgend ermittelt der Proxy, ob für den angefragten Host Credentials im Credential-Store gespeichert sind:

```
Credential[] cred = Proxy.getCredentialStore().
    getCredentials(host, lookupPort);
```

Ist dies nicht der Fall, prüft der Proxy ob die Verbindungsanfrage des Clients aus einem zulässigen IP-Bereich stammt. Gegebenenfalls weist er die Verbindungsanforderung ab.

```
boolean passed = Netmask.addressWithinNetList(
    socket.getInetAddress().getHostAddress(),
    Proxy.getConfigManager().
        getStr("allowed_networks").split(",_")
);

if (!passed) {
    ServerError.returnServerError(socket, request,
        HTTPStatus.StatusCodes.FORBIDDEN);
    return;
}
```

Dann baut er eine Verbindung zum Ziel-Server auf und verbindet den Client-Socket direkt mit dem Server. Dazu kommen in der Methode `connectClientToServer` zwei Instanzen der Klasse `tg.tlsauthproxy.util.RawSocketConnector` zum Einsatz, welche die vom Client bzw. Server empfangenen Daten an den jeweiligen Kommunikationspartner weiterleiten.

```
SocketFactory socketFactory = SocketFactory.getDefault();
Socket          targetSocket = socketFactory.createSocket(host,
                                                         port);

HTTPResponse proxyResponse = new HTTPResponse();
proxyResponse.setProtocolVersion(request.getProtocolVersion());
proxyResponse.setStatus(HTTPStatus.StatusCodes.OK,
    "Connection_Established");
proxyResponse.writeHeader(socket.getOutputStream());

connectClientToServer(socket, targetSocket, request, proxyResponse);
```

Der eigentlich interessante Teil des Connect-Method-Handlers kommt dagegen zum Einsatz, wenn für den Ziel-Host mindestens ein Credential gefunden wurde.

Zuerst bringt die Methode hier das Zertifikat des Servers in Erfahrung. Handelt es sich um eine umgeleitete¹⁵ HTTP-Verbindung, so erstellt sie dazu ein komplett neues Zertifikat. Andernfalls kommt die Methode `getSiteCertificate` aus der Klasse `tg.tlsauthproxy.util.SSLUtil` zum Ermitteln des Server-Zertifikats zum Einsatz. Details zur Realisierung dieser Methode folgen in Abschnitt 4.6.2.

¹⁵Erkennbar am Port »HTTP_REDIRECT_PORT«.

```
X509Certificate cert = null;

if (port == HTTP_REDIRECT_PORT)
    cert = createHttpRedirectCert(host);
else
    cert = SSLUtil.getSiteCertificate(host, port);
```

Im Anschluss wird das vom Server abgerufene Zertifikat dupliziert, mit dem CA-Zertifikat des Proxys als Aussteller versehen und signiert. Nähere Details hierzu folgen in Abschnitt 4.6.3.

```
X509Certificate fakeCert =
    Proxy.getCertManager().fakeCertificate(cert, keyPair);
```

Die bis zu diesem Zeitpunkt noch ungesicherte Verbindung zum Client wird im Anschluss zur SSL-Verbindung aufgewertet. Dazu kommt eine `createSocket`-Methode der `SSLConnectionFactory` zum Einsatz, der ein bestehender `Socket` als Basis übergeben werden kann.

Vor dem Upgrade der Verbindung sendet der Proxy eine entsprechende positive Rückmeldung an den Client. Um dem Client-Socket das Server-Zertifikat zuordnen zu können, wird vor dem Beginn des Handshakes mit Hilfe der Klasse `tg.tlsauthproxy.util.SocketToHostMap` eine entsprechende Zuordnung erstellt. Die Prüfung des Nutzerkontos des Clients übernimmt die Klasse `tg.tlsauthproxy.pki.ClientTrustManager`, die Verwaltung der gefälschten Server-Zertifikate die Klasse `tg.tlsauthproxy.pki.FakeServerKeyManager`. Sowohl der Key-Manager als auch der Trust-Manager sind dem globalen Client-SSL-Kontext zugeordnet, den die Methode über den Aufruf von `Proxy.getClientSSLContext()` abrufen.

```
SSLConnectionFactory socketFactory = Proxy.getClientSSLContext().
    getSocketFactory();
SSLSocket clientSocket = (SSLSocket) socketFactory.createSocket(
    socket, socket.getInetAddress().getCanonicalHostName(),
    socket.getPort(), true);
clientSocket.setUseClientMode(false);
clientSocket.setNeedClientAuth(true);

HTTPResponse proxyResponse = new HTTPResponse();
proxyResponse.setProtocolVersion(request.getProtocolVersion());
proxyResponse.setStatus(HTTPStatus.StatusCodes.OK,
    "Connection_Established");
proxyResponse.writeHeader(socket.getOutputStream());

DistinguishedName serverDN = new DistinguishedName(cert.
    getSubjectDN());
SocketToHostMap.add(clientSocket, serverDN.getCN());
clientSocket.startHandshake();
```

Nach erfolgreichem Ende des Handshakes wird im Anschluss das vom Client übermittelte Zertifikat ausgelesen und mit Hilfe des User-Managers in ein Objekt vom Typ `tg.tls-authproxy.backend.UserPrincipal` gewandelt. Schließlich folgt die Erstellung einer entsprechend personalisierten Sitzung für die Client-Verbindung.

```
X509Certificate[] clientChain = CertificateConverter.convert(
    clientSocket.getSession().getPeerCertificateChain());
UserPrincipal clientPrincipal = null;
if (clientChain != null && clientChain.length > 0)
    clientPrincipal = Proxy.getUserManager().getUser(clientChain[0]);
clientSession = Proxy.getSessionManager().
    createClientSession(clientSocket, clientPrincipal, host);
```

Nun beginnt der eigentliche Verbindungsaufbau zum Ziel-Server. Dazu wird zuerst ein benutzerspezifischer `SSLContext` erstellt, der bei einer eventuell erforderlichen SSL-Client-Authentifikation Zugriff auf alle für den authentifizierten Benutzer verfügbaren Credentials hat.

```
SSLContext context =
    UserSSLContextFactory.getDefault().getContext(clientPrincipal,
        "TLS");
SocketFactory sf = null;
Session serverSession = null;
```

Daraufhin erfolgt erneut eine Unterscheidung anhand des Ports, ob es sich um eine zu Authentifikationszwecken umgeleitete¹⁶ HTTP-Verbindung oder um eine Standard-SSL-Verbindung (alle übrigen Ports) handelt.

Für den Umleitungsfall muss zunächst der eigentliche Ziel-Port ermittelt werden, der aus den gespeicherten Credentials abgeleitet wird. Im Anschluss erfolgt der Aufbau einer konventionellen (d. h. nicht SSL-gesicherten) TCP/IP-Verbindung zum Ziel-Server.

```
if (port == HTTP_REDIRECT_PORT) {
    sf = new ConnectTimeoutSocketFactory(
        Proxy.getConfigManager().getInt("timeout", 1500),
        SocketFactory.getDefault());
    RestrictedCredential[] creds = Proxy.getCredentialStore().
        getRestrictedCredentials(host, 0, clientPrincipal);

    int serverPort = -1;
    for (int n = 0; n < creds.length; n++)
        if (!Credential.SCHEME_HTTPS.equals(creds[n].toCredential().
            getScheme())) {
            serverPort = creds[n].toCredential().getPort();
        }

    if (serverPort == -1)
```

¹⁶Erkennbar am Port »HTTP_REDIRECT_PORT«.

```
serverPort = 80;

serverSession = Proxy.getSessionManager().
    createServerSession(host, serverPort, sf,
        ServerSessionInitHandler.getInstance());
}
```

Für SSL-geschützte Verbindungen beschränkt sich der Verbindungsaufbau auf das Erstellen eines neuen `SSLSocket` und der zugehörigen Server-Session.

```
else {
    sf = new ConnectTimeoutSocketFactory(
        Proxy.getConfigManager().getInt("timeout", 1500),
        context.getSocketFactory());
    serverSession = Proxy.getSessionManager().
        createServerSession(host, port, sf,
            ServerSessionInitHandler.getInstance());
}
```

Im Anschluss werden Client-Socket und Server-Socket bis zum Schließen der Client-Verbindung miteinander verbunden. Die eigentliche Arbeit übernehmen dabei in der Methode `connectSSLClientToServer` die Klassen `tg.tlsauthproxy.util.HTTPRequestSocketConnector` sowie `tg.tlsauthproxy.util.HTTPResponseSocketConnector`.

```
do {
    connectSSLClientToServer(clientSession, serverSession);
} while (!clientSocket.isClosed() && !socket.isClosed());
```

Der exemplarische Ablauf einer SSL-Verbindung mit Client-Authentifikation ist zur Veranschaulichung in Abbildung 4.3, der einer umgeleiteten HTTP-Verbindung mit Basic-Authentication in Abbildung 4.4 dargestellt.

4.6.2. Abruf von Server-Zertifikaten: `SSLUtil.getSiteCertificate`

Während man vermuten könnte, dass die Ermittlung des zu einem SSL-Server gehörenden X.509-Zertifikats unproblematisch über die Methode `getPeerCertificateChain()` des einer SSL-Verbindung zu Grunde liegenden SSL-Kontexts erfolgen kann, stellt sich in der Praxis heraus, dass dieser Ansatz nicht in allen Fällen zum gewünschten Ergebnis führt.

So treten dann Probleme auf, wenn der Server den Proxy zu einer SSL-Client-Authentifikation auffordert. Da der Proxy zu diesem Zeitpunkt nämlich noch nicht die Identität des Clients kennt, und damit nicht weiß, auf welche Credentials der Client zugreifen darf, kann er nicht das für die Authentifikation zu verwendende Zertifikat ermitteln. Ohne entsprechendes Client-Zertifikat bricht der Handshake jedoch mit einer Exception ab. Um den Client wiederum zur Authentifikation aufzufordern, muss der Proxy dagegen erst das Zertifikat des Servers fälschen, um sich gegenüber dem Client auszuweisen. Das

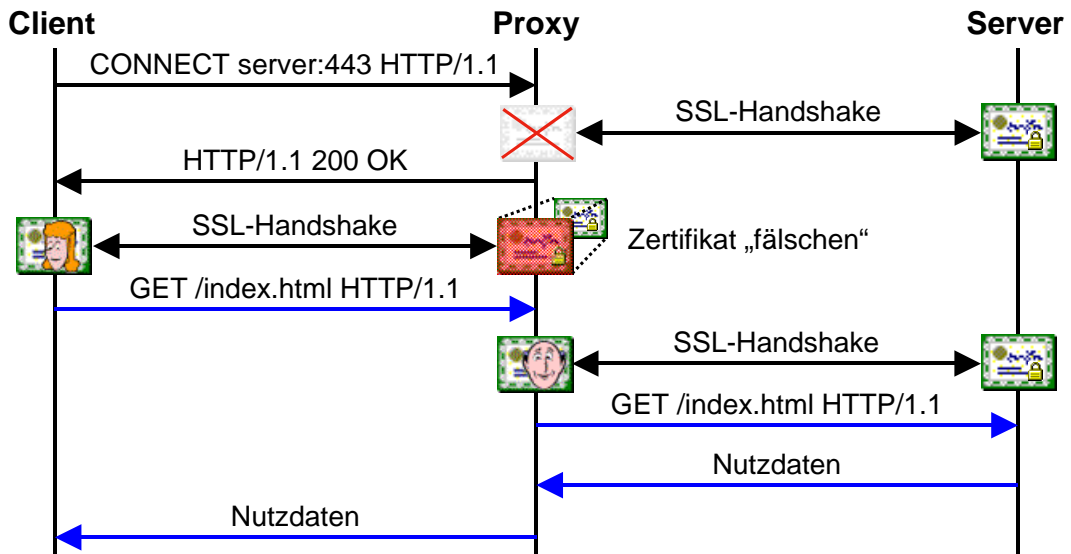


Abbildung 4.3: Ablauf einer HTTPS-Verbindung

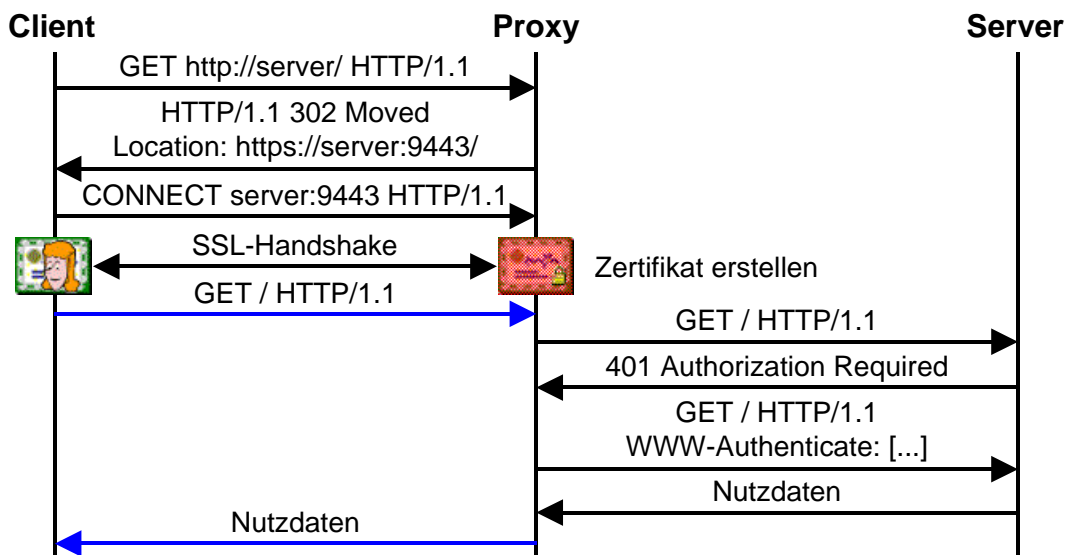


Abbildung 4.4: Ablauf einer HTTP-Verbindung

entsprechende Server-Zertifikat ist ihm zu diesem Zeitpunkt jedoch noch nicht bekannt, usw.

Das realisierte Verfahren macht sich zu Nutze, dass JSSE direkt nach dem Empfang der *Certificate*-Nachricht des Servers das enthaltene X.509-Server-Zertifikat an den Trust-Manager des SSL-Kontexts zur Prüfung weitergibt – unabhängig davon, ob der Handshake im weiteren Verlauf durch eine Exception abgebrochen wird.

So implementiert die Lösung einen eigenen Trust-Manager, `tg.tlsauthproxy.util.SSLUtil.CertificatePeeker`, der mittels der Hilfsklasse `tg.tlsauthproxy.util.SSLUtil.CertificateInfo` das gewünschte Server-Zertifikat auch im Falle einer Exception an die Methode `getSiteCertificate` weiterreicht.

Die Methode erstellt zu Beginn eine neue Instanz der Klasse `CertificateInfo` sowie des Trust-Managers `CertificatePeeker`, um dann einen neuen SSL-Kontext mit diesem Trust-Manager zu initialisieren:

```
CertificateInfo ci = INSTANCE.new CertificateInfo();
TrustManager[] tm = new TrustManager[] {
    INSTANCE.new CertificatePeeker(ci)};
SSLContext ctx = SSLContext.getInstance("TLS");
ctx.init(null, tm, null);
```

Im Anschluss finden der klassische Socket-basierte Verbindungsaufbau zum Ziel-Server sowie die Initialisierung verschiedener SSL-Parameter statt:

```
SocketFactory sf = new ConnectTimeoutSocketFactory(
    Proxy.getConfigManager().getInt("timeout", 1500),
    ctx.getSocketFactory());
SSLSocket sock = (SSLSocket) sf.createSocket(host, port);

sock.setUseClientMode(true);
sock.setEnableSessionCreation(true);
```

Schließlich folgt der für den eigentlichen Abruf des Zertifikats verantwortliche Code. Dieser ermittelt zuerst, welches SSL-Session-Objekt dem Socket zugeordnet ist, um im Anschluss mit dem Handshake zu beginnen. Tritt währenddessen eine Exception auf, versucht die Methode über den zuvor registrierten Trust-Manager an das Server-Zertifikat zu gelangen. Dies geschieht über die Methode `CertificateInfo.getChain()`. Ansonsten kommt der dokumentierte Weg über die Methode `SSLContext.getPeerCertificateChain()` zum Einsatz.

```
SSLSession sess = sock.getSession();
X509Certificate[] chain = null;

try {
    sock.startHandshake();
    if (sess.getPeerCertificateChain() != null)
        chain = CertificateConverter.convert(
            sess.getPeerCertificateChain());
```

```
} catch (IOException e) {  
    chain = ci.getChain();  
}
```

4.6.3. Fälschen von Server-Zertifikaten: CertificateManager. fakeCertificate

Essentiell für die Funktionsweise des Proxys ist das Fälschen von X.509-Zertifikaten. Dieses ist in der Methode `fakeCertificate` der Klasse `tg.tlsauthproxy.pki.PostgreSQLCertificateManager` realisiert.

Ein Großteil des entsprechenden Codes dieser Methode kopiert verschiedene Attribute des originalen Zertifikats in das gefälschte Zertifikat. Da dieser Code nicht sonderlich interessant zu lesen ist, sollen im Folgenden lediglich *die* Teile der Methode betrachtet werden, die jeweils eine Besonderheit in der Implementierung darstellen.

Die erste dieser Besonderheiten ist die Bestimmung der Seriennummer des neuen Zertifikats. Um die Notwendigkeit eines streng monoton steigenden, globalen Zählers für die Seriennummer zu vermeiden, setzt die vorliegende Implementierung als Seriennummer die Anzahl der Millisekunden seit dem 01.01.1970, 0:00 Uhr.

```
cert.setSerialNumber(BigInteger.valueOf(Calendar.getInstance().  
    getTimeInMillis()));
```

Weiterhin muss das gefälschte Zertifikat einen neuen öffentlichen Schlüssel, zu dem der Proxy ebenfalls den privaten Schlüssel kennt, sowie einen neuen *Issuer Distinguished Name* erhalten.

```
cert.setSubjectPublicKey(keyPair.getPublic());  
cert.setIssuerDN(myCACert_.getSubjectDN());
```

Damit nachgehende Uhren auf den Clients zu keinen Fehlern bei der Überprüfung des Zertifikats führen, verschiebt die Methode den Gültigkeitsbeginn des Zertifikats zehn Minuten in die Vergangenheit.

```
Calendar notBefore = Calendar.getInstance();  
notBefore.add(Calendar.MINUTE, -10);  
cert.setNotBefore(notBefore);
```

Nicht alle im originalen Zertifikat enthaltenen X.509-Erweiterungen machen im Kontext des neuen Herausgebers noch Sinn. Daher verwendet die Klasse eine Blacklist, die verhindert, dass bestimmte Erweiterungen in des neue Zertifikat übernommen werden. Die in dieser Blacklist enthaltenen Erweiterungen sind in Tabelle 4.1 aufgeführt. Ergänzt werden um neue Einträge kann die Blacklist im Konstruktor von `tg.tlsauthproxy.pki.PostgreSQLCertificateManager`.

Schließlich folgt die eigentliche Signaturerstellung. `cert.sign` ist hierbei eine in der Klasse `tg.tlsauthproxy.pki.X509Certificate` implementierte Erweiterung, die den bei JSSE etwas umständlichen Signaturvorgang über eine vereinfachte Schnittstelle zur Verfügung stellt.

OID	Beschreibung
1.3.6.1.5.5.7.1.1	CRL
2.5.29.31	CRL Distribution Points
2.5.29.32	Certificate Policies
2.16.840.1.113733.1.6.15	VeriSign ServerID
2.16.840.1.113730.1.3	Netscape Revocation URL
2.16.840.1.113730.1.8	Netscape CA Policy URL

Tabelle 4.1: Blacklist für nicht zu kopierende X.509-Erweiterungen

```
Signature mySig = Signature.getInstance("SHA1withRSA");
mySig.initSign(myCAPrivateKey_);
cert.sign(mySig, myCACert_.getPublicKey());
```

Das nun mit einer gültigen Signatur des Proxys versehene neue Zertifikat wird schließlich an die aufrufende Methode als gefälschtes Zertifikat zurückgeliefert.

4.7. Mögliche Erweiterungen

4.7.1. Kryptographisch gesicherte Speicherung der Credentials

Während die vorliegende Implementierung eine unverschlüsselte Speicherung der Credentials im Credential-Store vorsieht, ist folgende Erweiterung denkbar: Um die im Credential-Store abgelegten Credentials gegen unbefugten Zugriff nach einer Kompromittierung des Proxys zu schützen, können diese derart verschlüsselt abgelegt werden, dass eine Wiederherstellung eines Credentials ohne das Mitwirken des jeweils berechtigten Clients nicht möglich ist. Dazu kommt ein hybrides Verschlüsselungsverfahren zum Einsatz.

Verfahrensbeschreibung

Für jedes Credential M wird ein symmetrischer Schlüssel K_S erzeugt, mit dem M verschlüsselt gespeichert wird (Gleichung 4.3).

$$C = E(M, K_S) \quad (4.3)$$

Für jeden berechtigten Nutzer i eines Credentials wird K_S nun mit dem Public Key K_{E_i} des Nutzers verschlüsselt (Gleichung 4.4) und ebenfalls in der Datenbank abgelegt.

$$K_{S_i} = E(K_S, K_{E_i}) \quad (4.4)$$

Soll nun ein Credential im Namen eines berechtigten Nutzers zum Einsatz kommen, sendet der Proxy K_{S_i} an den Client, der im Besitz des privaten Schlüssels K_{D_i} ist und mit diesem den symmetrischen Schlüssel entschlüsseln kann (Gleichung 4.5).

$$K_S = D(K_{S_i}, K_{D_i}) \quad (4.5)$$

Der Client sendet K_S nun zurück an den Proxy, der damit das zur Authentifikation notwendige Credential M berechnen kann (Gleichung 4.6).

$$K_S = D(K_{S_i}, K_{D_i}) \quad (4.6)$$

Die Verwendung eines hybriden Verfahrens ist allgemein notwendig, damit der Client beim Entschlüsseln des Credentials für den Proxy nicht in dessen Besitz gelangt.

Einschränkungen

Der im vorangehenden Abschnitt beschriebene Datenaustausch lässt sich mit der üblicherweise auf dem Client installierten Standard-Browsersoftware leider nicht realisieren. Jedoch ist die Implementierung einer clientseitigen Hilfsanwendung als Java-Application¹⁷ denkbar, welcher der private Schlüssel des Clients bekannt ist. Zum Austausch der zum Zugriff auf die Credentials notwendigen symmetrischen Schlüssel kommuniziert diese mit dem Proxy über einen separaten, verschlüsselten Kanal.

4.7.2. TLS-Upgrade statt Redirect für HTTP-Verbindungen

Momentan ist zur korrekten Authentifikation des Clients bei HTTP-Verbindungen eine Umleitung des clientseitigen Browsers auf eine HTTPS-URL notwendig, so dass eine TLS-basierte Client-Authentifikation durchgeführt werden kann, die prinzipbedingt zwingend eine TLS-geschützte Verbindung voraussetzt. Da jedoch die Umleitung auf eine neue URL nicht immer völlig problemlos ist, wäre die Nutzung eines weniger invasiven Ansatzes wünschenswert.

Hier bietet das in RFC 2817 [Kha00] beschriebene Verfahren zum Upgrade einer HTTP-Verbindung auf TLS eine interessante Möglichkeit. Durch Nutzung eines speziellen HTTP-Status-Codes und Einfügen eines »Upgrade«-Headers in die Antwort des Proxys wird bei dieser Methode durch eine Folge von HTTP-Nachrichten eine TLS-gesicherte Verbindung zwischen Client und Proxy aufgebaut. Auf dieser TLS-Verbindung ist im Anschluss die übliche Client-Authentifikation als integraler Bestandteil des Handshakes möglich.

Ein Schwachpunkt dieses Ansatzes soll jedoch nicht verschwiegen werden: Er wird momentan noch von keinem Browser unterstützt, weshalb dieses Verfahren zum jetzigen Zeitpunkt eher akademischen Wert hat. Dies ist auch der Grund, warum das Verfahren lediglich als mögliche Erweiterung vorgesehen ist und bei der Implementierung des Prototypen keine Berücksichtigung fand.

¹⁷alternativ auch als Java-Applet

Fazit und Ausblick

Motivation für diese Diplomarbeit war die zunehmende Verbreitung von Authentifikationstechniken im World Wide Web, verbunden mit den dabei auftretenden Problemfeldern.

Diese Problemfelder wurden, verbunden mit einer allgemeinen Einführung in die Authentifikationsthematik, im einleitenden Teil dieser Arbeit herausgearbeitet, nämlich die effiziente Verwaltung, Delegation und Gruppennutzung von Credentials.

Darauf aufbauend wurden zwei Architekturen vorgestellt, die sich beide mit einer zentralen Komponente zur Verwaltung der Credentials dieser Probleme annehmen: Ein Middleware-basierter Ansatz, die so genannte Stellvertreter-Architektur, sowie ein client-seitiger Ansatz. Für die Stellvertreter-Architektur wurden drei Realisierungsvarianten präsentiert: Eine auf der Anwendungsebene als HTTP-Server realisierte Variante, das HTTP-Gateway, eine Realisierung auf Protokollebene als HTTP-Proxy sowie eine Variante auf der Basis eines Application-Servers.

Eine anschließende Bewertung und ein Vergleich aller vier Realisierungsvarianten führte zum Entwurf einer Authentifikationslösung auf der Basis der HTTP-Proxy-Architektur.

Die im weiteren Verlauf der Arbeit durchgeführte Implementierung dieser Architektur hat gezeigt, dass die dargestellten Konzepte auch praktisch umsetzbar sind. Die realisierte Architektur ermöglicht im Vergleich zu den in gängigen Webbrowsern realisierten Credential-Stores sowohl eine effiziente zentrale Administration von Credentials in einer Mehrbenutzerumgebung als auch deren Delegation und Gruppennutzung. Dabei setzt sie auf standardisierten Protokollen auf und erreicht dadurch eine hohe Kompatibilität zu bestehenden Client-Installationen. Sie realisiert sogar eine Art *Pseudo-Single-Sign-On* [Pas03], der im Gegensatz zu echten *Single-Sign-On*-Lösungen, wie Kerberos, Microsoft Passport oder Liberty Alliance, nicht auf die Kooperation der jeweiligen Serverbetreiber angewiesen ist.

Gleichwohl handelt es sich bei der Implementierung um einen Prototyp oder eine Machbarkeitsstudie mit Verbesserungspotential. Die folgenden Verbesserungs- und Erweiterungsmöglichkeiten sollen aufzeigen, an welchen Stellen eine zukünftige Weiterentwicklung der Software ansetzen könnte:

- Wünschenswert wären Verbesserungen im Bereich der Benutzbarkeit, insbesondere bei der Erstellung neuer Credentials. Hier wäre ein Assistent denkbar, der dem Anwender die mühsame Konfiguration der Authentifikationsparameter bei der Erstellung eines formularbasierten Credentials abnimmt. Dieser Assistent könnte z. B. die Parameter automatisch bestimmen, indem er bei einer Browser-basierten Authentifikation die übermittelten Formularinhalte mitliest.
- Zwar protokolliert der Proxy, welcher authentifizierte Nutzer zu welchem Zeitpunkt

auf eine Ressource zugegriffen hat, jedoch ist momentan keine Protokollierung der übertragenen Inhalte möglich. So bedenklich eine solche Protokollierung aus Sicht des Datenschutzes auch ist, so nötig ist sie in manchen Szenarien, in denen der Nachweis erbracht werden muss, welche Aktionen ein Nutzer beim Gebrauch eines Credentials auf einem Server durchgeführt hat.

Die entsprechende Konfiguration einer Logging-Möglichkeit ist dabei in der aktuellen Realisierung bereits vorgesehen, es fehlt lediglich die Implementierung der eigentlichen Protokollierung.

- Neben dem momentan implementierten datumsbasierten Constraint für die Nutzungsbeschränkung eines Credentials sind weitere Constraint-Typen denkbar, z. B. für die Einschränkung der Nutzung auf bestimmte Tageszeiten oder Wochentage.
- Die vorliegende Realisierung ermöglicht bei der Delegation eines Credentials keine Einschränkung des Constraints, das einer Nutzungsberechtigung zu Grunde liegt. Eine Möglichkeit, dieses Constraint z. B. durch Definition oder Verkürzung eines Nutzungszeitraums weiter zu beschränken, wäre daher wünschenswert.
- Zwei zusätzliche Erweiterungen, welche die Sicherheit der zentral verwalteten Credentials erhöhen sowie eine alternative Realisierung der Authentifikation per X.509-Zertifikat bei HTTP-Verbindungen umsetzen, wurden bereits in Abschnitt 4.7 dargestellt.

Zusammenfassend stellt die realisierte Proxy-Architektur also eine durchaus in der Praxis einsetzbare Lösung dar, die jedoch auch noch Spielraum für zukünftige Erweiterungen bietet.

Alice genießt zwischenzeitlich übrigens ihren Urlaub auf Barbados, während Bob mit Hilfe des an ihn delegierten Credentials für die Dauer ihrer Abwesenheit die Bankgeschäfte des Unternehmens erledigt.

Installation und Erstkonfiguration

Während die Installation des Proxys selbst weitestgehend menügeführt erfolgt, ist für die Konfiguration der Datenbank ein deutlich höherer Aufwand nötig. Diese soll daher im Folgenden detailliert erläutert werden. In einem weiteren Abschnitt wird abschließend ebenfalls die Erstkonfiguration des Proxys dargestellt.

A.1. Installation

A.1.1. Proxy

Die Installation des Proxys beschränkt sich auf das Entpacken des Installationsarchivs¹ des Proxys in ein beliebiges leeres Verzeichnis, das im Folgenden »Basisverzeichnis« genannt wird.

A.1.2. PostgreSQL-Server

Kommt als Systemplattform eine Linux-Distribution zum Einsatz, so dürfte die reine Installation in der Mehrheit der Fälle mit der Installation des PostgreSQL-Pakets per Paketverwaltung abgeschlossen sein.

Im Folgenden soll dagegen die deutlich weniger intuitive Installation von PostgreSQL auf einem Microsoft-Windows-System unter Cygwin², einer POSIX-Implementierung für die Microsoft-Windows-Plattform, erläutert werden. Dazu sind nach der Installation der Pakete »`postgres`«, »`ipc-daemon2`« und »`cygrunsrv`« über die Cygwin-Paketverwaltung noch einige Installationsschritte vorzunehmen. Diese werden im Folgenden auf der Cygwin-Konsole (üblicherweise die „Bash“-Shell) ausgeführt.

Erstellen eines Benutzerkontos für den PostgreSQL-Daemon

Um die Sicherheit des Systems für den Fall einer Kompromittierung des PostgreSQL-Daemons, z. B. durch einen zukünftigen Buffer-Overflow, zu schützen, ist es empfehlenswert, den PostgreSQL-Daemon unter einem eigenen Benutzerkonto, z. B. mit dem Namen »`PostgreSQL`«, auszuführen. Dieses Benutzerkonto wird mit Hilfe der Nutzerverwaltung von Windows erstellt³:

```
cmd /c lusrmgr.msc
```

¹`tlsauthproxy-x.y.zip` bzw. `tlsauthproxy-x.y.tar.bz2`, wobei »`x.y`« für die Versionsnummer der im Archiv enthaltenen Distribution des *TLS Authentication Proxy* steht.

²URL: <http://www.cygwin.com/>

³Das genaue Vorgehen ist – je nach eingesetzter Windows-Version – leicht unterschiedlich.

Wichtig dabei ist, dass dem Nutzer durch Entzug aller Gruppenmitgliedschaften nur minimale Zugriffsrechte zur Verfügung stehen.

Im Anschluss muss das Benutzerkonto der Kategorie »Als Dienst anmelden« unter »Lokale Richtlinien« | »Zuweisen von Benutzerrechten« hinzugefügt werden:

```
cmd /c secpol.msc
```

Mit Hilfe des Befehls

```
mkpasswd -l > /etc/passwd
```

wird abschließend die Cygwin-Nutzerdatenbank mit ihrem Windows-Pendant synchronisiert.

ipc-daemon2

Nachdem die Vorarbeiten nun abgeschlossen sind, wird als nächstes der für die Interprozesskommunikation zuständige »ipc-daemon2« installiert und anschließend direkt gestartet:

```
ipc-daemon2 --install-as-service  
cygrunsrv -S ipc-daemon2
```

Konfiguration des Datenverzeichnisses

Im Anschluss legt man nun das Datenverzeichnis an, in dem PostgreSQL die eigentliche Datenbank speichern soll:

```
mkdir /var/lib  
mkdir /var/lib/postgres  
mkdir /var/lib/postgres/data  
chown -R PostgreSQL /var/lib/postgres/
```

Nun wird – eingeloggt als Nutzer »PostgreSQL« – die PostgreSQL-Datenbank initialisiert:

```
initdb -D /var/lib/postgres/data
```

postmaster

Darauf folgt Installation und Start des PostgreSQL-Daemons, der den Namen »postmaster« trägt:

```
cygrunsrv --install postmaster --path /usr/bin/postmaster --args \  
  "-D_/var/lib/postgres/data_i" --dep ipc-daemon2 --termsig INT \  
  --user PostgreSQL --shutdown  
net start postmaster
```

#	TYPE	DATABASE	USER	IP-ADDRESS	IP-MASK	METHOD
local	all		all			md5
host	all		all	127.0.0.1	255.255.255.255	md5

Abbildung A.1: Konfigurationsdatei »pg_hba.conf«

Einspielen des Datenbank-Schemas

Die Datenbank des Proxys wird nun – wieder als Benutzer »PostgreSQL« – wie folgt angelegt und mit ihrem initialen Datenbestand gefüllt. Das dazu verwendete Datenbankschema »tlsauthproxy.schema« befindet sich im Unterverzeichnis »/schema« des Basisverzeichnisses.

```
psql template1 < schema/tlsauthproxy.schema
```

Dabei werden zwei neue Datenbanknutzer erstellt: »root« mit dem Kennwort »changeme« und »tlsauthproxy« mit dem Kennwort »secret«. Diese müssen im Anschluss an die Installation unbedingt durch sichere Kennwörter ersetzt werden!

Absicherung von lokalen Zugriffen

Standardmäßig ist eine neue PostgreSQL-Installation so konfiguriert, dass lokale Zugriffe und Zugriffe über das Loopback-Interface ohne weitere Authentifikation erlaubt sind.

Um diese potentielle Sicherheitslücke zu beheben, empfiehlt es sich, in der Konfigurationsdatei »/var/lib/postgres/data/pg_hba.conf« die Authentifikations-Methode »trust« durch eine sichere Methode, z. B. »md5« zu ersetzen. Eine entsprechend angepasste Konfigurationsdatei ist in Abbildung A.1 dargestellt.

A.1.3. PostgreSQL-Web-Administrations-GUI: phpPgAdmin

Für die Konfiguration des DBMS und die manuelle Wartung des Datenbestands hat sich der Einsatz des Web-Frontends phpPgAdmin⁴ als äußerst hilfreich erwiesen. Dessen Installation und Konfiguration soll im Folgenden kurz beschrieben werden.

Die Installation beginnt mit dem Entpacken des Installationsarchivs in ein vom Apache-Webserver publiziertes Verzeichnis. Möglich wäre z. B. das Verzeichnis »phpPgAdmin« direkt unter dem DocumentRoot-Verzeichnis des Apache-Servers, das üblicherweise unter der URL <http://localhost/phpPgAdmin/> erreichbar ist.

Im Anschluss sollte noch eine optionale Konfigurationsänderung mit Hilfe einer neu zu erstellenden ».htaccess«-Datei (Abbildung A.2) im »phpPgAdmin«-Verzeichnis vorgenommen werden: Durch die Konfigurationsänderung erfolgt beim Aufruf der URL von phpPgAdmin direkt die Anzeige des Login-Bildschirms anstatt der standardmäßig dargestellten Dateiliste.

⁴URL: <http://phpPgAdmin.sourceforge.net/>

```
Options -Indexes
DirectoryIndex index.php
```

Abbildung A.2: Konfigurationsdatei ».htaccess« zur Änderung des Standard-Dokuments beim Verzeichniszugriff

Läuft der zu administrierende PostgreSQL-Server auf demselben System, auf dem auch Apache und phpPgAdmin installiert wurden, ist die Konfiguration an dieser Stelle bereits abgeschlossen. Andernfalls muss die Konfigurationsdatei »phpPgAdmin/conf/config.inc.php« entsprechend um die Angabe des Host-Namens des Servers ergänzt werden.

A.2. Erstkonfiguration des Proxys

Neben dem Entpacken des Installationsarchivs ist vor der ersten Inbetriebnahme des Proxys die Konfiguration einiger grundlegender Parameter erforderlich. Diese Konfigurationsparameter fragt der Proxy automatisch beim ersten Start interaktiv auf der Konsole ab. Später lässt sich dieser Konfigurationsmodus jederzeit durch Anhängen der Option »--config« an die Kommandozeile des Proxys erneut aufrufen.

Der Start des Proxys erfolgt über den Aufruf der Batch-Datei »tlsauthproxy.bat« (Windows) bzw. des Shell-Skripts »tlsauthproxy.sh« (Unix), die sich beide im Basisverzeichnis des Proxys befinden.

Nach dem Start beginnt der Proxy mit der Konfiguration. Die während dieser Konfiguration abgefragten Optionen werden im Folgenden anhand einer beispielhaften Konfigurationssitzung erläutert.

Zu Beginn ist es erforderlich, dem Proxy das Basisverzeichnis mitzuteilen, in das er entpackt wurde:

```
TLS Authentication Proxy Version 0.x

It seems that this is the first time you run the proxy. Please take
a few minutes to complete the initial configuration.

First, you will need to configure some basic settings:
  Base directory [.]: C:\TLSEAuthProxy
```

Im Anschluss wird der Datenbank-Zugriff konfiguriert. Host-Name, Datenbank und Benutzername sind dabei mit Standardwerten für eine Installation der Datenbank auf dem lokalen Rechner vorbelegt. Lediglich das Datenbank-Kennwort muss in jedem Fall eingegeben werden.

```
Next, you will need to configure the PostgreSQL database server to
use:
  Host name [localhost]: stargazer
```

```
Database [tlsauthproxy]: tlsauthproxy
User [tlsauthproxy]: tlsauthproxy
Password: secret
```

Im nächsten Schritt erfolgt die Konfiguration der TCP/IP-Ports, auf denen der Proxy Verbindungen entgegennimmt:

```
The database has been configured successfully, so the setup will now
continue with the configuration of additional general settings:
Ports [8080]: 80, 8080
```

Abschließend werden die Daten des *Distinguished Name* des Proxy-eigenen CA-Zertifikats erfasst:

```
Finally, it is required to generate an X.509 certificate for the
proxy.
```

```
Please provide the certificate attributes:
```

```
Common Name (CN): Acme, Inc. Proxy CA
Organizational Unit (OU): Proxy CA
Organization (O): Acme, Inc.
Locality (L): Darmstadt
State (ST): Hessen
Country (CN): DE
Valid For (years) [5]: 10
```

Zu diesem Zeitpunkt ist die grundlegende Konfiguration abgeschlossen, der Proxy startet.

Webbasierte Konfiguration

Weitere Konfigurationsänderungen sowie die Verwaltung von Credentials und Benutzern sind nun über das webbasierte Administrations-Interface möglich. Um auf diese Schnittstelle zuzugreifen, ist es notwendig, im Webbrowser auf dem lokalen Rechner den HTTPS-Proxy als localhost:8080 (bzw. anstatt des Ports 8080 mit dem in der vorangegangenen Konfiguration alternativ eingestellten Port) zu konfigurieren. Die dazu im Browser durchzuführenden Einstellungen sind in Abschnitt C.2.1 dokumentiert.

Im Anschluss kann der Proxy unter der URL `https://proxyadmin/` administriert werden. Der Zugang zu dieser Administrationsschnittstelle ist von dem Rechner, auf dem der Proxy installiert ist, zu diesem Zeitpunkt noch unauthentifiziert möglich. Als nächsten Schritt empfiehlt es sich daher, einen neuen Nutzer auf dem Proxy anzulegen und diesen mit administrativen Privilegien auszustatten, so dass der Proxy nur noch authentifizierte Zugriffe auf das Administrations-Interface zulässt.

Weiterführende Informationen zur webbasierten Administration des Proxys enthält der folgende Anhang B.

Administration

In diesem Teil des Anhangs werden die Grundlagen der webbasierten Administration des Proxys erläutert. Es erfolgt dabei eine Darstellung von allgemeinen Einstellungen, Benutzerverwaltung, Credential-Verwaltung sowie Delegation von Credentials.

Die webbasierte Administrationsschnittstelle ist, sofern der Proxy als solcher im Webbrowser konfiguriert wurde, über die URL `https://proxyadmin/` erreichbar. Bis auf die Erstkonfiguration (siehe Abschnitt A.2) benötigt der Nutzer Zugriffsrechte auf das Credential »Administrative Privileges (proxyadmin)«, um auf die Administrationsfunktionalität zugreifen zu dürfen. Andernfalls besteht lediglich Zugang zum Delegationsmenü sowie zur Liste der verfügbaren Credentials.

B.1. Globale Einstellungen

Die Seite »General« (siehe Abbildung B.1) ermöglicht die Veränderung globaler Parameter des Proxys. Nach Veränderung einer oder mehrerer Optionen werden die Änderungen durch Betätigen der Schaltfläche »Save« gespeichert.

Address Ranges Allowed To Connect

Diese Option legt fest, welche Clients auf die anonym nutzbaren Dienste des Proxys zugreifen dürfen. Die Angabe mehrerer durch Kommata getrennter Tupel *IP-Adresse/Netzmaske* ist zulässig.

Beispiel: Um dem lokalen Rechner (localhost) sowie allen Rechnern aus dem Netz der TU Darmstadt den Zugriff auf den Proxy zu erlauben, weist man der Einstellung den Wert »127.0.0.1/255.255.255.255, 130.83.0.0/255.255.0.0« zu.

Client Certificate: Default Attribute

Mit diesen Einstellungen lassen sich die verschiedenen Attribute des *Distinguished Name* für die Erstellung neuer Nutzer mit Standardwerten vorbelegen.

Network Timeout

Diese Option gibt den Timeout (in Sekunden) für den Aufbau neuer TCP/IP-Verbindungen vom Proxy zu einem Ziel-Server vor.

PAC: Selectively Proxy Requests

Ist diese Option aktiviert, wird die Datei zur automatischen Proxy-Konfiguration so verändert, dass nur Verbindungen zu Servern, für die im Proxy Credentials hinterlegt wurden, über diesen geleitet werden. Alle anderen Verbindungen erfolgen dagegen auf direktem Weg ohne zwischengeschalteten Proxy.

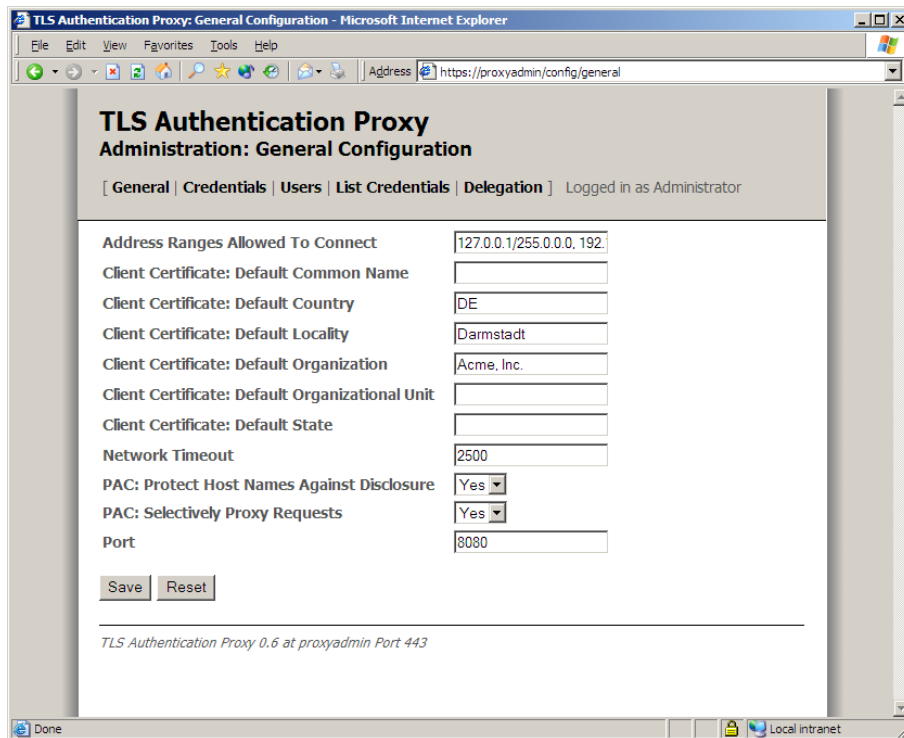


Abbildung B.1: Administration: Allgemeine Einstellungen

PAC: Protect Host Names Against Disclosure

Da bei Aktivierung der vorangegangenen Option in Form der PAC-Datei eine vollständige Liste von Servern abrufbar ist, für die der Proxy Credentials speichert, besteht mit dieser Option die Möglichkeit, diese Liste mit Hilfe einer kryptografischen Hashfunktion zu verschleiern, so dass keine Rückschlüsse auf die beteiligten Server mehr möglich sind. Nähere Details zum eingesetzten Verfahren finden sich in Abschnitt 4.2.5.

Ports

Diese Einstellung gibt an, auf welchen Netzwerk-Ports der Proxy Verbindungsanfragen entgegennimmt. Mehrere Ports werden durch Kommata getrennt angegeben.

Hinweis: Hier wird eine Änderung erst nach einem Neustart des Proxys wirksam.

B.2. Nutzerverwaltung

Der Menüpunkt »Users« stellt Funktionen zur Verwaltung von Nutzern und Zugriffsrechten auf Credentials (ACLs) zur Verfügung.

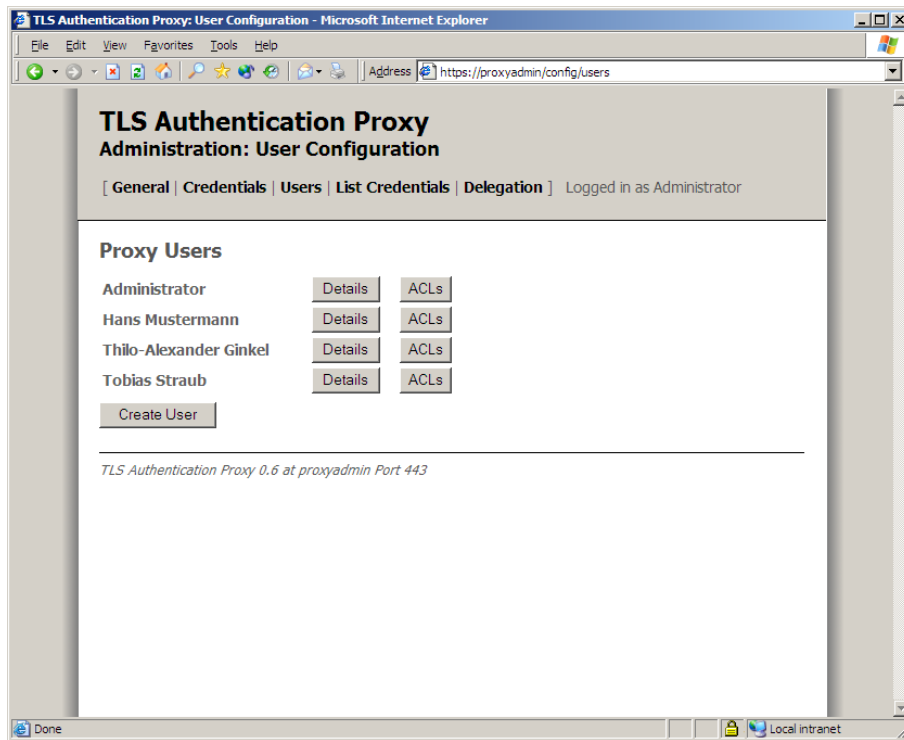


Abbildung B.2: Administration: Benutzerverwaltung: Übersicht

B.2.1. Übersicht

Auf der Übersichtsseite der Nutzerverwaltung (siehe Abbildung B.2) steht eine Liste der dem System bekannten Nutzer zur Verfügung. Neben jedem Nutzer befindet sich eine Schaltfläche zum Betrachten der Details des Nutzerkontos (»Details«) sowie ein Button zum Einsehen und Bearbeiten der Credential-Nutzungsberechtigungen des Benutzers (»ACLs«).

B.2.2. Benutzerkonten erstellen

Bei der Erstellung eines neuen Nutzers (siehe Abbildung B.3) muss neben den üblichen Attributen eines *Distinguished Names*, von denen lediglich der *Common Name* verpflichtend ist, noch die Gültigkeit des Zertifikats festgelegt werden. Diese kann im Bereich von 30 Tagen bis 2 Jahren über die Combo-Box »Validity« gewählt werden.

Durch Anwählen der Schaltfläche »Create« wird ein Benutzerkonto für den neuen Nutzer angelegt sowie ein Public-Key-Schlüsselpaar und ein Nutzerzertifikat erstellt. Der private Schlüssel und das Nutzerzertifikat werden in einem PKCS#12-Container verpackt, den der Administrator anschließend durch Klicken auf die Schaltfläche »Download PKCS#12« herunterladen kann (siehe Abbildung B.4). Der PKCS#12-Container ist durch eine zufällig gewählte, achstellige alphanumerische Transport-PIN geschützt, die dem Administrator auf der Download-Seite angezeigt wird.

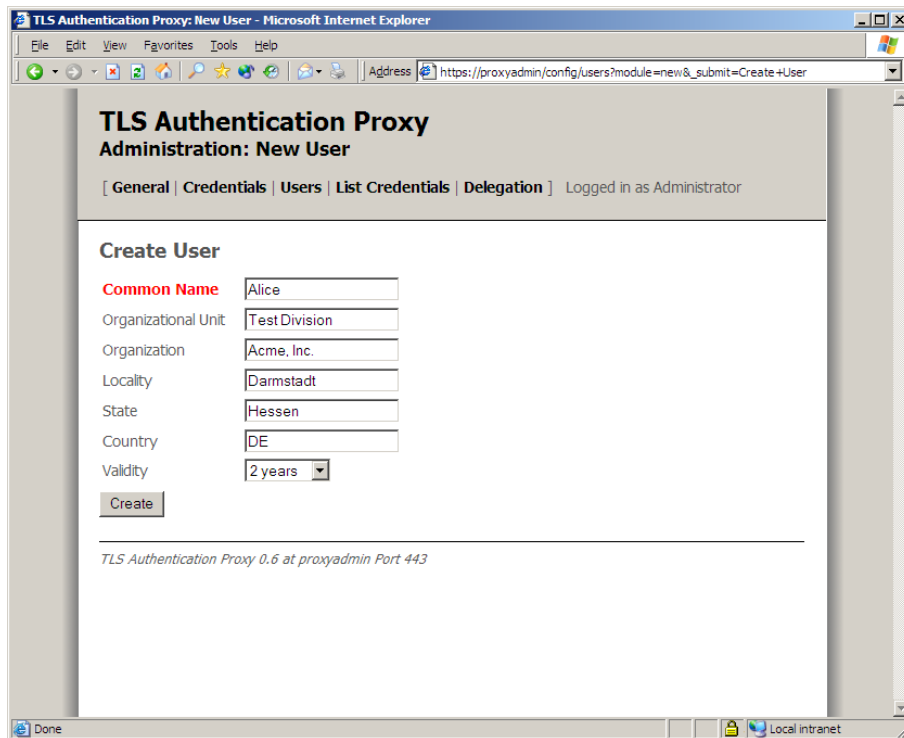


Abbildung B.3: Administration: Benutzerverwaltung: Nutzer erstellen

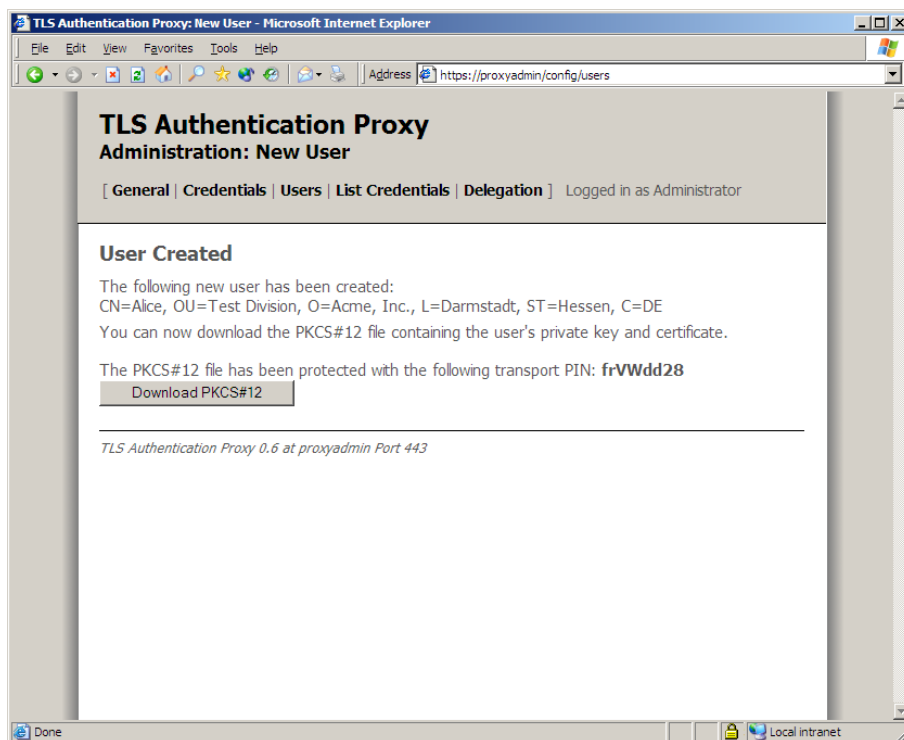


Abbildung B.4: Administration: Benutzerverwaltung: PKCS#12-Download

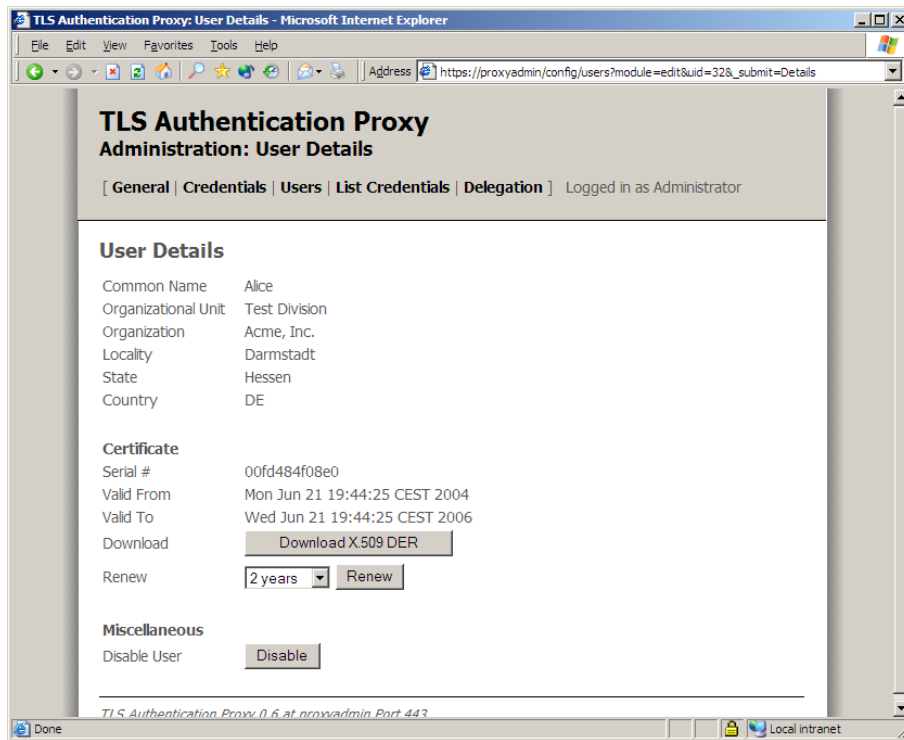


Abbildung B.5: Administration: Benutzerverwaltung: Benutzer-Eigenschaften

B.2.3. Benutzerkonten verwalten

Für bereits bestehende Benutzerkonten führt das Anklicken der Schaltfläche »Details« auf der Benutzerübersicht auf eine Seite, welche die Details des Benutzerkontos anzeigt (siehe Abbildung B.5).

Das dem Benutzerkonto zu Grunde liegende X.509-Zertifikat lässt sich in dieser Ansicht sowohl herunterladen (Schaltfläche »Download X.509 DER«) als auch erneuern (Schaltfläche »Renew«). Bei der Zertifikatserneuerung kann die neue Gültigkeitsdauer in vorgegebenen Stufen festgelegt werden.

Schließlich besteht die Möglichkeit, das Benutzerkonto durch einen Klick auf »Disable« zu deaktivieren, bzw. durch Anklicken der Schaltfläche »Enable« wieder zu aktivieren. Ein Benutzer mit deaktiviertem Konto kann keine Funktionen des Proxys nutzen, für die er sich authentifizieren muss. Er kann sich also weder mit Hilfe von Credentials authentifizieren, noch sie delegieren. Einem deaktivierten Administratorkonto bleibt dann auch der Zugriff auf die komplette Administrationsschnittstelle verwehrt.

B.3. Zugriffsrechte verwalten

Durch Anwählen des Schalters »ACLs« neben einem Benutzernamen auf der Übersichtsseite der Benutzerverwaltung gelangt man zu einer Liste der Zugriffsrechte auf Credentials, die der jeweilige Nutzer besitzt (siehe Abbildung B.6). Für jedes Zugriffsrecht

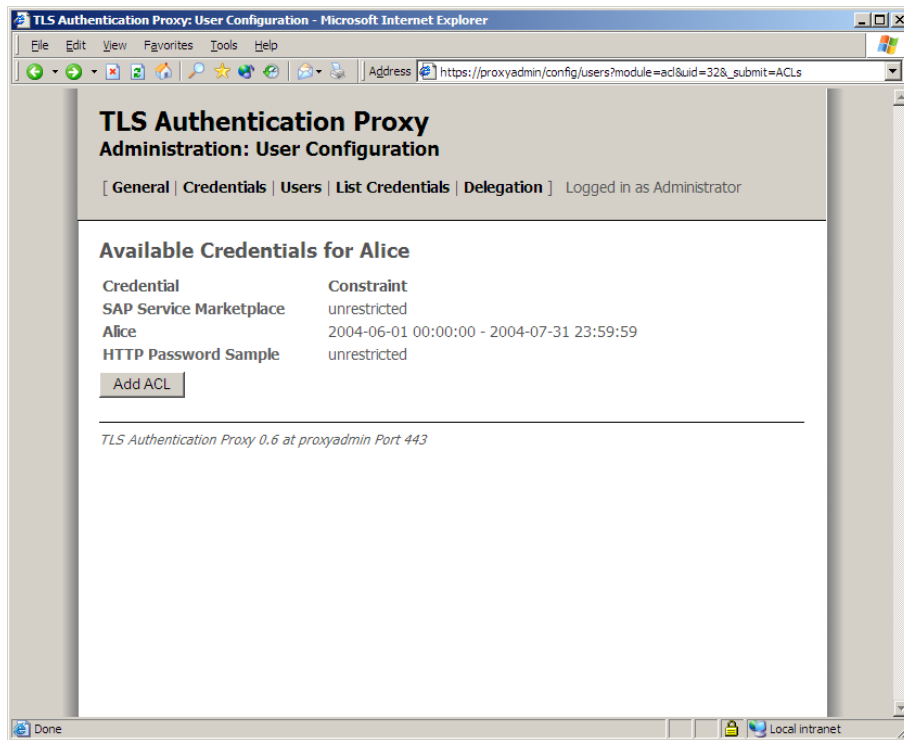


Abbildung B.6: Administration: Zugriffsrechte: Übersicht

sind in der Übersicht sowohl der Name des referenzierten Credentials als auch eventuelle Einschränkungen aufgeführt. Durch Anklicken eines Eintrags lassen sich weitere Eigenschaften des Zugriffsrechts anzeigen.

B.3.1. Zugriffsrecht erstellen

Um dem Nutzer Zugriff auf ein weiteres Credential zu gewähren, wählt man den Schalter »Add ACL« an. Im Anschluss erscheint die in Abbildung B.7 dargestellte Eingabemaske. Deren Einstellungsmöglichkeiten sollen nun im Folgenden kurz erläutert werden.

Credential

Die Auswahlliste enthält alle Credentials, auf die der ausgewählte Benutzer noch keinen Zugriff hat. In Klammern ist jeweils der Hostname des Servers aufgeführt, zu dem das Credential gehört.

Owner

Diese Einstellung gibt den Besitzer des neuen Zugriffsrechts an. Der Besitzer eines Zugriffsrechts darf dieses per Delegation an einen anderen Nutzer weitergeben. Ist eine Delegation durch einen Nutzer nicht gewünscht, sollte als Besitzer ein Administrator angegeben werden.

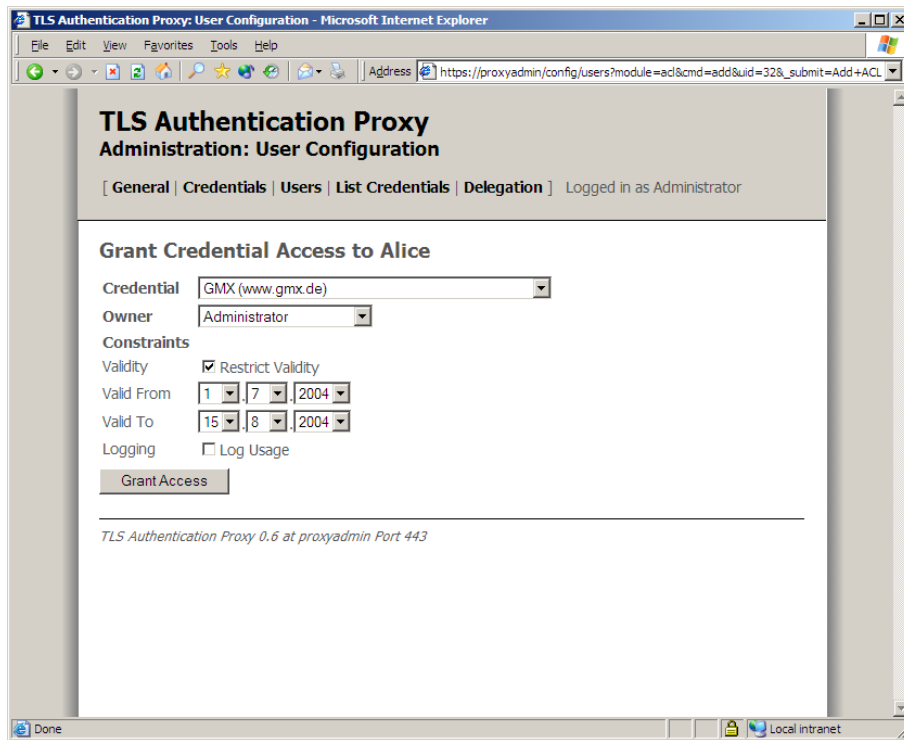


Abbildung B.7: Administration: Zugriffsrechte: Zugriffsrecht erstellen

Constraints

In diesem Abschnitt sind mögliche Einschränkungen der Nutzung des Zugriffsrechts konfigurierbar. Dazu muss die Checkbox »Restrict Validity« angekreuzt sowie unter »Valid From« und »Valid To« das Gültigkeitsdatum des Credentials vorgegeben werden.

Log Usage

Die Option »Log Usage« ist für eine zukünftige Erweiterung zum Protokollieren der Credential-Nutzung vorgesehen und erfüllt im Moment keine Funktion.

B.3.2. Zugriffsrecht löschen

Nach dem Anwählen eines bestehenden Zugriffsrechts auf der Übersichtsseite besteht auf der dann erscheinenden Detailseite die Möglichkeit, das Zugriffsrecht über die Schaltfläche »Delete« dauerhaft zu löschen.

B.4. Credential-Verwaltung

Nach Anwahl des Menüpunkts »Credentials« erhält der Nutzer eine Übersicht über die Hosts, für die aktuell Credentials hinterlegt sind (siehe Abbildung B.8). Durch Klicken auf die Schaltfläche »Details« gelangt er zu einer Liste von Credentials für den jeweils

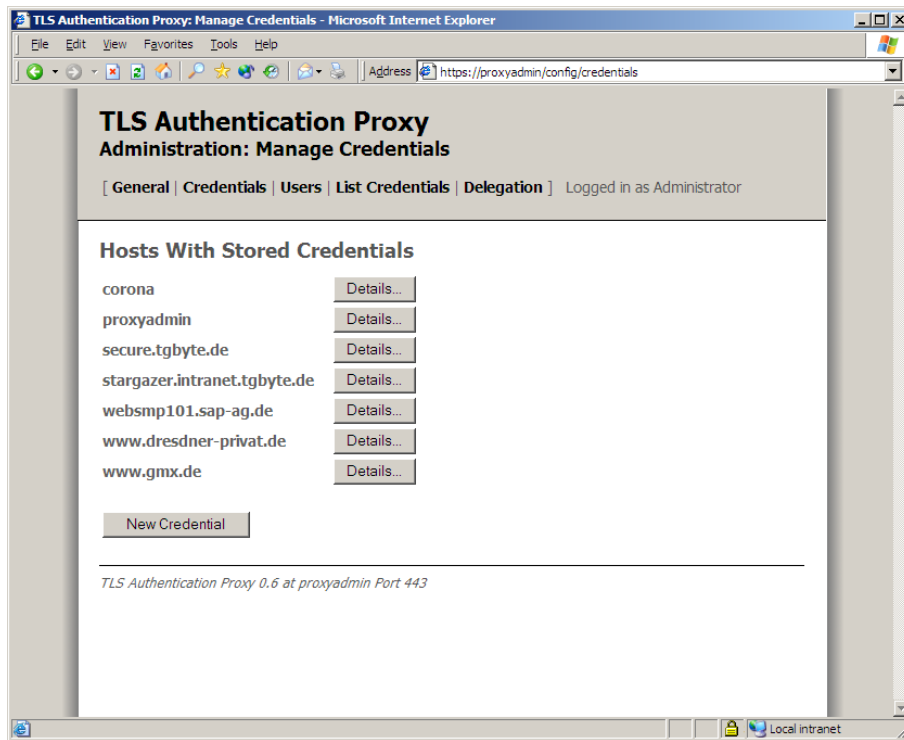


Abbildung B.8: Administration: Credentials: Übersicht (Hosts)

ausgewählten Host (Abbildung B.9). Dort lässt wiederum jedes Credential durch Klick auf den Button »Details« bearbeiten und gegebenenfalls auch löschen.

Ein neues Credential kann durch Anwählen der Schaltfläche »New Credential« erstellt werden. Nach Auswahl des jeweiligen Credential-Typs werden dann auf der nachfolgenden Seite dessen spezifische Attribute erfasst.

B.4.1. Credential-Eigenschaften

Alle Credential-Typen verfügen über einige gemeinsame Eigenschaften, die im Folgenden kurz erläutert werden. Die typspezifischen Eigenschaften folgen im Anschluss in einem separaten Abschnitt für jeden Credential-Typ.

Name

Diese Eigenschaft legt den Namen des Credentials fest. Der Name hat rein deskriptiven Charakter und kann frei gewählt werden.

Type

Das Feld »Type« gibt den Typ des Credentials an. Der Typ eines Credentials wird mit dessen Erstellung einmalig festgelegt und kann daher später nicht mehr verändert werden. Zulässige Werte sind momentan »HTTP Password (Basic/Digest Authentication)«, »X.509 SSL/TLS Client Authentication« sowie »Form-Based HTTP Authentication«.

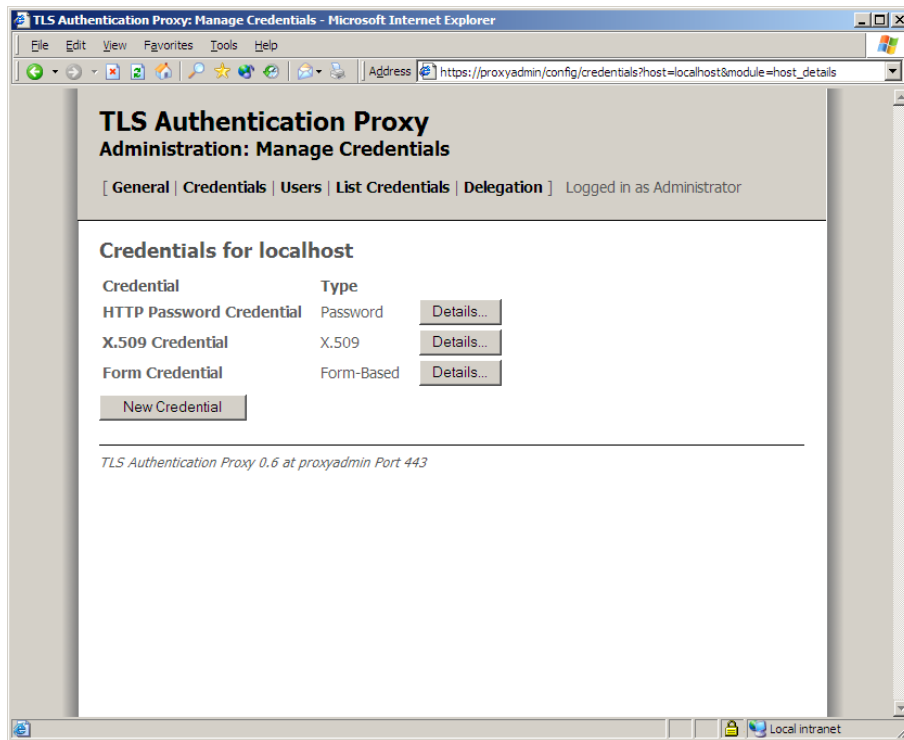


Abbildung B.9: Administration: Credentials: Übersicht (Credentials)

Host Name

Diese Eigenschaft gibt den Namen des Hosts an, zu dem das Credential gehört. Sie ermöglicht beim Zugriff auf einen Host die Zuordnung der zugehörigen Credentials.

Port

Die Einstellung »Port« gibt den Port an, für den das Credential gültig ist. Gängige Werte sind »80« (HTTP) sowie »443« (HTTPS).

Scheme

Das Schema bezeichnet den Teil einer URL vor dem ersten Doppelpunkt und dient in diesem speziellen Fall der weiteren Einschränkung der Gültigkeit eines Credentials. Je nach Einstellung beschränkt es die Nutzung eines Credentials auf HTTP- oder HTTPS-Verbindungen.

Service URL

Die Service-URL gibt einen Einstiegspunkt in die Website vor, zu deren Nutzung das Credential berechtigt. Sie wird in der Credential-Liste (siehe Abschnitt B.6) als Hyperlink für das jeweilige Credential verwendet, um dem Nutzer eine Möglichkeit zum direkten Zugriff auf die zugeordnete Ressource zu geben.

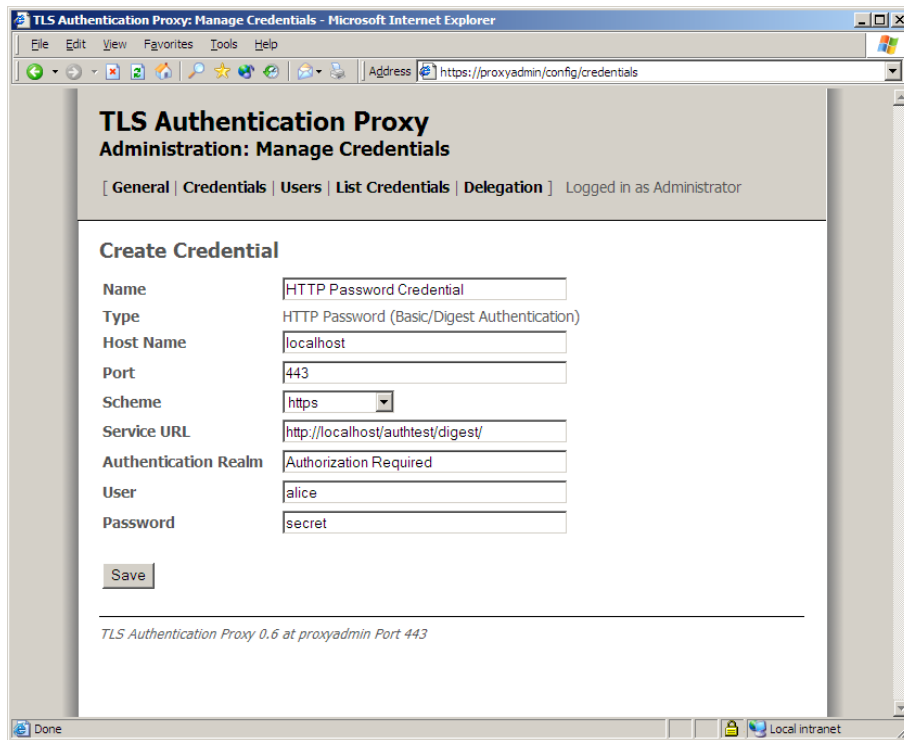


Abbildung B.10: Administration: Credentials: Kennwortbasiertes Credential erstellen

B.4.2. Kennwortbasierte Credentials

Die Erstellung eines neuen kennwortbasierten Credentials erfolgt über die in Abbildung B.10 dargestellte Maske. Deren Einstellungsmöglichkeiten sollen nun kurz beschrieben werden.

Authentication Realm

Diese Eigenschaft legt den für die *Basic*- und *Digest Authentication* notwendigen so genannten *Authentication Realm* fest (siehe Kapitel 2.2.1). Nur wenn dieser mit dem vom Server übermittelten *Authentication Realm* übereinstimmt, wird das Credential für eine Authentifikation berücksichtigt.

User

Die Einstellung »User« spezifiziert, welcher Benutzername zu Authentifikationszwecken an den Server übermittelt wird.

Password

Diese Einstellung gibt vor, welches Kennwort zur Authentifikation an den Server übermittelt wird.

Die beim späteren Verändern des Credentials möglichen Einstellungen sind identisch mit den bei der Erstellung verfügbaren Optionen.

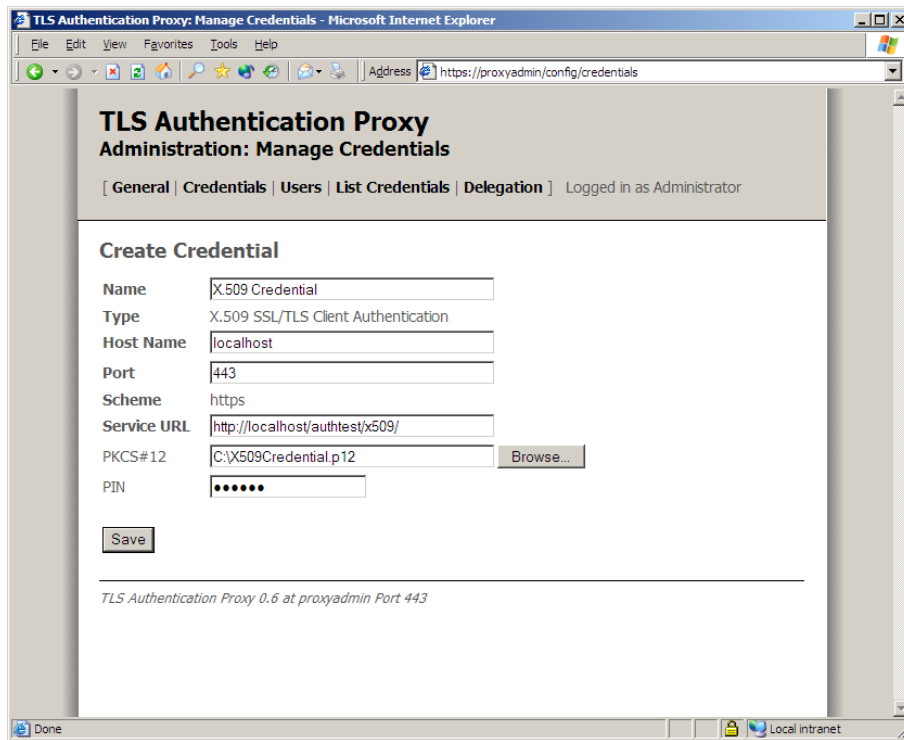


Abbildung B.11: Administration: Credentials: X.509-Credential erstellen

B.4.3. X.509-Credentials

Die Erstellung eines X.509-Credentials (siehe Abbildung B.11) erfordert an zusätzlichen Parametern lediglich die Angabe eines PKCS#12-Containers, der – geschützt durch eine Transport-PIN mittels einer symmetrischen Chiffre – das Benutzerzertifikat sowie den zugeordneten privaten Schlüssel beinhaltet.

Beim Zugriff auf ein bereits existierendes Credential (siehe Abbildung B.12) besteht analog die Möglichkeit, Benutzerzertifikat und privaten Schlüssel durch ein neues Zertifikat/Schlüssel-Paar zu ersetzen.

Weiterhin werden in diesem Zusammenhang die wichtigsten Attribute des aktuell verwendeten Zertifikats dargestellt: Inhaber, Aussteller, Gültigkeitszeitraum sowie Seriennummer.

B.4.4. Formularbasierte Credentials

Die Parameter eines formularbasierten Credentials lassen sich in zwei Kategorien aufteilen: Einstellungen, die die Übermittlung der Formulardaten beeinflussen, sowie Einstellungen, die sich auf die Art des Sitzungs-Managements auswirken.

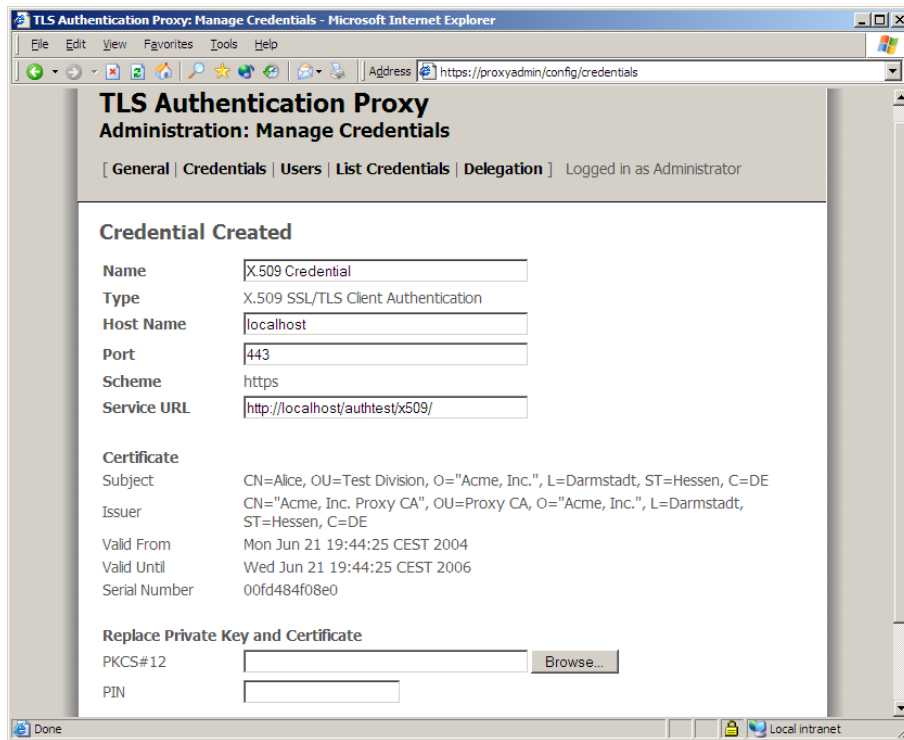


Abbildung B.12: Administration: Credentials: X.509-Credential – Eigenschaften

Formularübermittlung

Die erste Kategorie von Parametern betrifft die eigentliche Übermittlung des Authentifikationsformulars an den Server. Deren einzelne Optionen sind im Folgenden aufgeführt:

Request Method

Die *Request Method* legt fest, ob die Authentifikationsparameter als »GET«- oder als »POST«-Anfrage an den Server übermittelt werden.

Submission URL

Dieser Parameter gibt die URL vor, an welche die Authentifikationsdaten übermittelt werden. Sie entspricht dem »ACTION«-Attribut des entsprechenden HTML-Formulars.

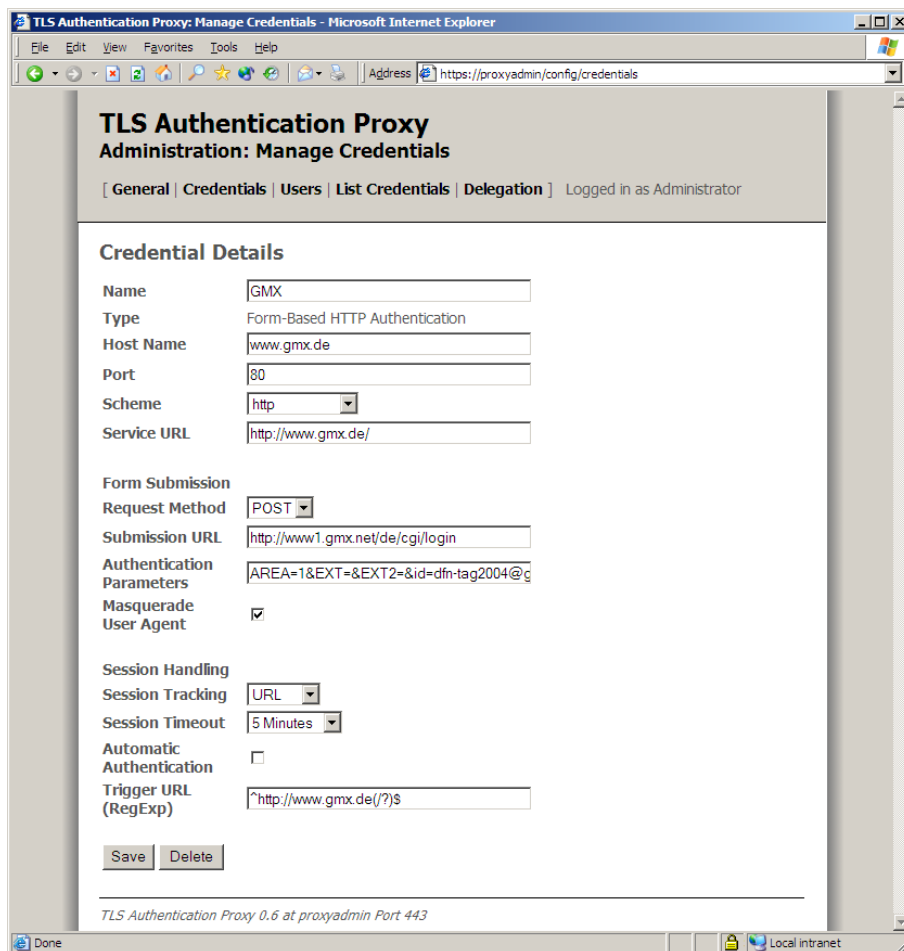
Masquerade User Agent

Ist diese Option aktiviert, übernimmt der Proxy bei Authentifikationsanfragen den vom Client übermittelten »User-Agent«-Header.

Authentication Parameters

Diese Einstellung enthält die an die *Submission URL* zu übermittelnden Authentifikationsparameter in URL-codierter Form¹.

¹Beispielsweise ergeben die Formularfelder »name« und »password« – belegt mit den bekannten Beispieldaten »alice« und »secret« – die URL-codierte Form »name=alice&password=secret«.



The screenshot shows a web browser window titled "TLS Authentication Proxy: Manage Credentials - Microsoft Internet Explorer". The address bar shows "https://proxyadmin/config/credentials". The page content is as follows:

TLS Authentication Proxy Administration: Manage Credentials

[General | Credentials | Users | List Credentials | Delegation] Logged in as Administrator

Credential Details

Name	<input type="text" value="GMX"/>
Type	Form-Based HTTP Authentication
Host Name	<input type="text" value="www.gmx.de"/>
Port	<input type="text" value="80"/>
Scheme	<input type="text" value="http"/>
Service URL	<input type="text" value="http://www.gmx.de/"/>

Form Submission

Request Method	<input type="text" value="POST"/>
Submission URL	<input type="text" value="http://www1.gmx.net/de/cg/login"/>
Authentication Parameters	<input type="text" value="AREA=1&EXT=&EXT2=&id=dfn-tag2004@g"/>
Masquerade User Agent	<input checked="" type="checkbox"/>

Session Handling

Session Tracking	<input type="text" value="URL"/>
Session Timeout	<input type="text" value="5 Minutes"/>
Automatic Authentication	<input type="checkbox"/>
Trigger URL (RegExp)	<input type="text" value="^http://www.gmx.de/(?)\$"/>

TLS Authentication Proxy 0.6 at proxyadmin Port 443

Abbildung B.13: Administration: Credentials: Formularbasiertes Credential erstellen

Sitzungs-Management

Darüber hinaus existiert eine zweite Parameterkategorie, die Einfluss auf das Sitzungs-Management für die Authentifikation nimmt:

Session Tracking

Die Einstellung »Session Tracking« legt fest, ob das Sitzungs-Management Cookie-basiert oder URL-basiert durchgeführt wird.

Session Timeout²

Das *Session Timeout* gibt an, nach welchem Inaktivitätszeitraum der Proxy eine Sitzung als abgelaufen betrachtet und bei Folgezugriffen eine erneute Authentifikation durchführt.

Automatic Authentication²

Ist diese Option aktiviert, erstellt der Proxy automatisch beim ersten Zugriff auf den Server eine neue Sitzung und authentifiziert sich gegenüber dem Server. Dasselbe gilt, falls lediglich eine bereits abgelaufene Sitzung im Sitzungs-Cache vorhanden ist.

Trigger URL

Bei der so genannten Trigger-URL handelt es sich um einen regulären Ausdruck. Alle URLs, auf die dieser reguläre Ausdruck passt, lösen die Erstellung einer neuen Sitzung mit neuem Authentifikationsvorgang aus.

Die beim späteren Bearbeiten des Credentials möglichen Einstellungen sind identisch mit den bei der Erstellung verfügbaren Optionen.

B.5. Delegation

Das Delegations-Modul (siehe Abbildung B.14) ist für alle authentifizierten Proxy-Nutzer zugänglich, nicht nur für Administratoren. Es ermöglicht dem jeweils angemeldeten Benutzer, das Nutzungsrecht für ein Credential unter Beibehaltung eventuell vorhandener Nutzungseinschränkungen an einen anderen Benutzer zu delegieren. Dazu wählt er im Abschnitt »New Delegation« unter »Credential« das zu delegierende Credential und unter »User« den Benutzer aus, an den das Nutzungsrecht delegiert werden soll. Ein anschließender Klick auf die Schaltfläche »Delegate Credential« führt die Delegation aus.

Eine Liste aktiver Delegationen wird im Abschnitt »Delegated Credentials« dargestellt. Dort sind neben dem Namen des jeweiligen Credentials dessen Host, der Name des Benutzers, an den delegiert wurde, sowie eventuelle Nutzungseinschränkungen aufgeführt. Die Rücknahme einer Delegation lässt sich durch Anwählen der Schaltfläche »Revoke« erreichen.

²Nur bei Cookie-basiertem Session-Tracking.

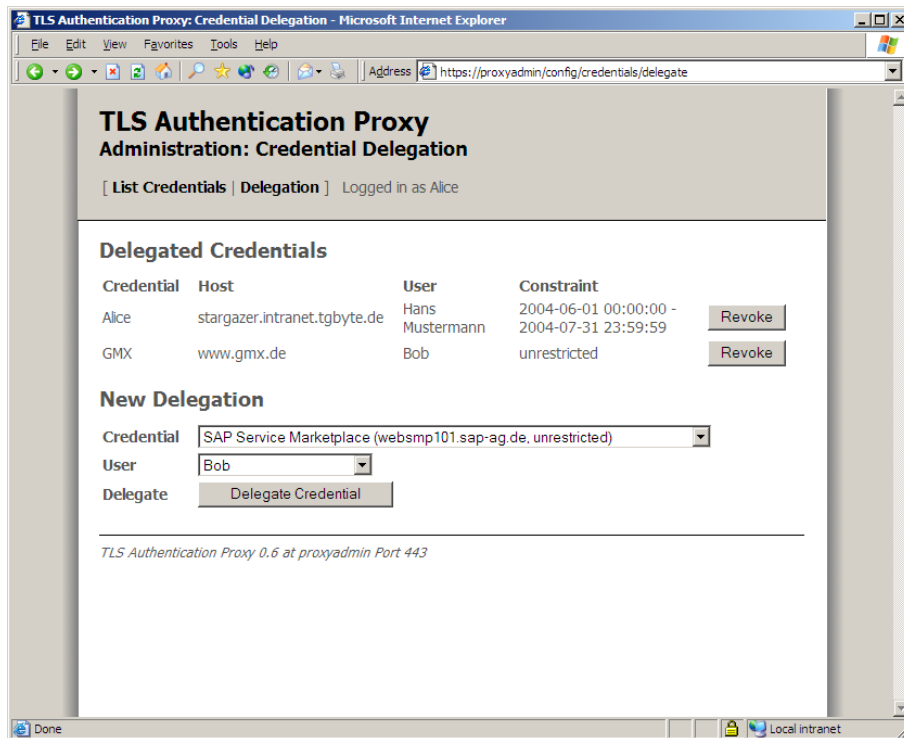


Abbildung B.14: Administration: Delegation

B.6. Credential-Liste

Auch die Credential-Liste (siehe Abbildung B.15) ist für alle authentifizierten Nutzer des Proxys zugänglich. Sie stellt eine alphabetisch sortierte Liste der Credentials dar, die dem angemeldeten Nutzer zur Verfügung stehen. Eventuell vorhandene Nutzungseinschränkungen sind hinter dem Hostnamen des Credentials in Klammern vermerkt. Ein Klick auf den Namen eines Credentials öffnet in einem neuen Fenster die jeweils für das Credential hinterlegte Service-URL.

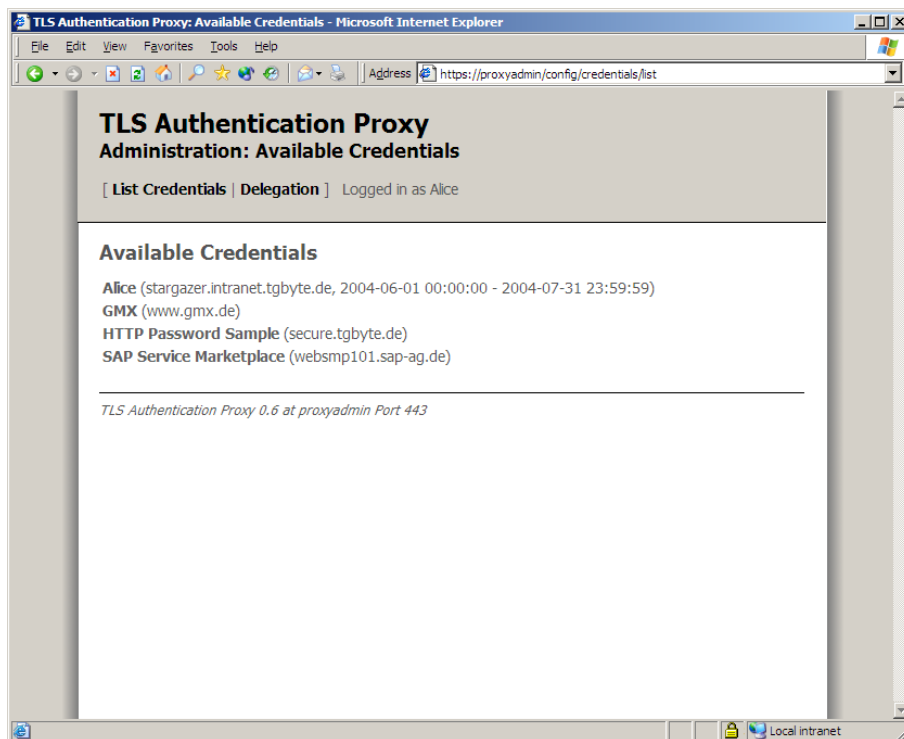


Abbildung B.15: Administration: Credential-Liste

Deployment

In diesem Kapitel werden die notwendigen Schritte beim Deployment des Proxys in einem Netzwerk dargestellt. Neben dem Erstellen, Verteilen und der Installation von Zertifikaten umfassen diese weiterhin die Konfiguration der Proxy-Einstellungen auf den beteiligten Client-Systemen.

Die folgende Dokumentation erhebt dabei keinen Anspruch darauf, diese Schritte für alle möglichen Deployment-Szenarien erschöpfend darzustellen. Vielmehr soll sie als eine Art Wegweiser grundlegende Konzepte und Strategien beim Deployment des Proxys aufzeigen.

C.1. Rollout der Zertifikate

Das Rollout der Zertifikate lässt sich in zwei Kategorien gliedern: Die Verteilung der Benutzerzertifikate an die einzelnen Nutzer sowie die Installation des Wurzelzertifikats der Proxy-eigenen Certificate-Authority als Vertrauensanker in den Certificate-Stores der clientseitigen Webbrowser.

Während der erstgenannte Punkt primär eine organisatorische Herausforderung darstellt, deren Lösung jedoch nicht Thema dieses Dokuments sein soll, lässt sich die Installation des Wurzelzertifikats des Proxys auf den Clients mit einem einfachen technischen Trick quasi automatisieren: Da jeder Nutzer des Proxys sowieso sein eigenes Benutzerzertifikat zur Authentifikation gegenüber dem Proxy im Certificate-Store des Webbrowsers hinterlegen muss, bietet es sich an, während dieses unumgänglichen Installationsschritts auch die Installation des Proxy-Wurzelzertifikats durchzuführen. Erfreulicherweise unterstützt das PKCS#12-Format, das für den Transport des privaten Schlüssels und des Benutzerzertifikats zum Einsatz kommt, nicht nur den Transport eines einzigen Zertifikat/Schlüssel-Paars, sondern ermöglicht das Einbinden zusätzlicher Zertifikatsketten. Diese werden beim Importieren der PKCS#12-Datei in den Certificate-Store des Webbrowsers auf Nachfrage automatisch installiert. Das vom Proxy so beim Anlegen eines Benutzerkontos in jede PKCS#12-Datei eingefügte Wurzelzertifikat kann mit diesem Mechanismus also zusammen mit den Benutzerzertifikaten verteilt werden.

C.2. Konfiguration der Proxy-Einstellungen

Neben dem Verteilen der Benutzerzertifikate ist die Konfiguration der Proxy-Einstellungen auf den Client-Systemen eine weitere Hürde, die es beim Deployment zu überwinden gilt. Der folgende Abschnitt dokumentiert zu Beginn deren manuelle Konfiguration und stellt im Anschluss eine automatisierte Konfigurationsmöglichkeit vor, die in der Lage ist, den nötigen Deployment-Aufwand erheblich zu verringern.

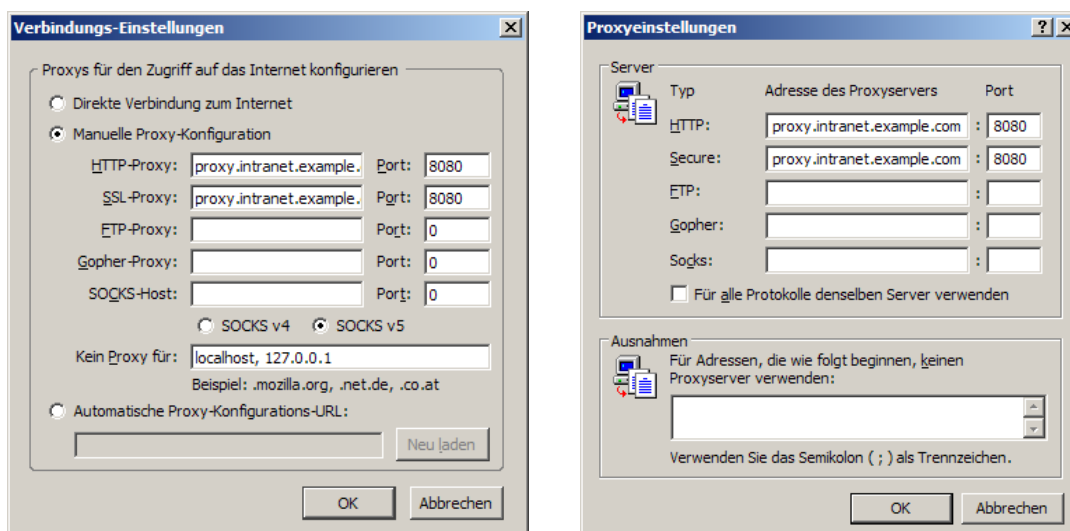


Abbildung C.1: Manuelle Konfiguration der Proxy-Einstellungen für Mozilla Firefox (links) und Internet Explorer (rechts)

C.2.1. Manuelle Proxy-Konfiguration

Bei der manuellen Konfiguration der Proxy-Einstellungen auf dem Client muss die Konfigurationsänderung direkt im jeweiligen Webbrowser vorgenommen werden. Auch diese manuelle Konfiguration lässt sich auf verschiedene Arten realisieren: Entweder gibt man die Adresse und den Port des HTTP- und HTTPS-Proxy-Servers fest vor oder man hinterlegt manuell in den Einstellungen des Webbrowsers die URL einer so genannten PAC-Datei zur automatischen Proxy-Konfiguration.

Vorgabe von Proxy-Adresse und Port

Für die manuelle Vorgabe der Adresse des Proxys sowie des zugehörigen Ports muss in den Verbindungseinstellungen¹ der Host-Name oder die Adresse des Rechners eingetragen werden, auf dem der *TLS Authentication Proxy* läuft, ebenso der Port (Standardwert: »8080«). Die Konfiguration dieser Einstellungen für den Microsoft Internet Explorer und Mozilla Firefox zeigt Abbildung C.1.

Bei dieser Art der Konfiguration werden alle HTTP- und HTTPS-Anfragen über den Proxy geroutet.

Vorgabe einer URL zur automatischen Proxy-Konfiguration

Sollen dagegen nur Verbindungen über den Proxy geleitet werden, für deren Ziel-Hosts der Proxy auch Credentials speichert, kommt ein alternatives Verfahren zum Einsatz: Die so genannte *Proxy Auto-Configuration* (PAC) [Net96]. Um diese zu nutzen, muss in

¹ Im Microsoft Internet Explorer erreicht man diese im Menü »Extras« unter »Internetoptionen...« | »Verbindungen« | »Einstellungen« | »Erweitert«, bei Mozilla Firefox ebenfalls im Menü »Extras« unter »Einstellungen...« | »Allgemein« | »Verbindungs-Einstellungen...«.

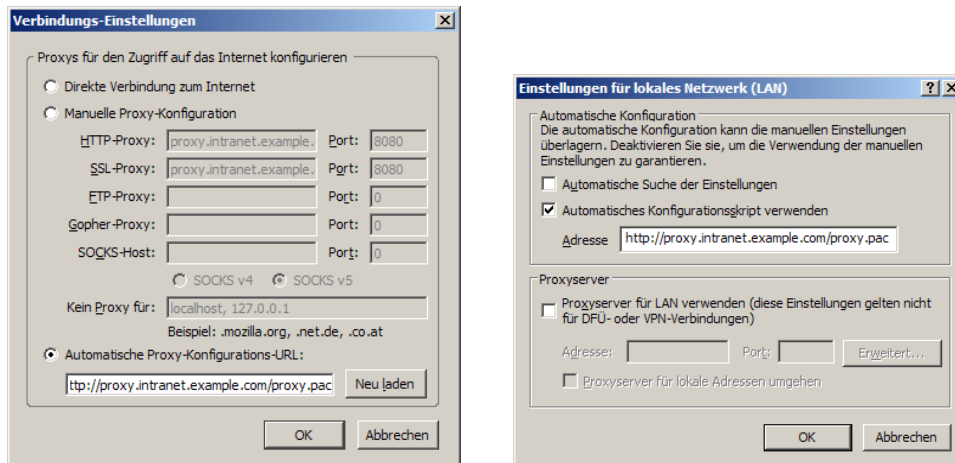


Abbildung C.2: Konfiguration der Proxy-Auto-Konfiguration für Mozilla Firefox (links) und Internet Explorer (rechts)

den Verbindungseinstellungen¹ des Webbrowsers eine URL für die automatische Proxy-Konfiguration hinterlegt werden (siehe Abbildung C.2). Diese URL ergibt sich aus dem Host-Namen des Proxy-Servers, dem Port des Proxys und dem Dateinamen der PAC-Datei »/proxy.pac«. Für den Hostnamen »proxy.intranet.example.com« und den Port »8080«, auf dem der *TLS Authentication Proxy* standardmäßig Verbindungen entgegennimmt, lautet diese URL beispielsweise:

»http://proxy.intranet.example.com:8080/proxy.pac«

C.2.2. Automatische Proxy-Konfiguration per WPAD-Protokoll

Für eine automatisierte Konfiguration der Proxy-Einstellungen auf den Clients unterstützt der Proxy das so genannte *Web Proxy Auto-Discovery Protocol* (WPAD) [Gau99].

Das WPAD-Protokoll ermöglicht in Verbindung mit dem Microsoft Internet Explorer² als Webbrowser auf dem Client eine automatische Konfiguration von dessen Proxy-Einstellungen. Es baut dabei auf dem DNS-Protokoll sowie dem Dynamic-Host-Configuration-Protokoll (DHCP) auf.

Der WPAD-Standard sieht verschiedene Konformitätsklassen vor, die jeweils eine mehr oder weniger umfangreiche Unterstützung der Discovery-Mechanismen des WPAD-Protokolls erfordern. Dieses Dokument beschreibt ausschließlich die Konfiguration der Klassen 0 (DNS-basierte Realisierung) und 1 (DNS- und DHCP-basierte Realisierung).

Welche der beiden Konformitätsklassen tatsächlich umgesetzt wird, bleibt dem Leser überlassen. In den meisten Systemumgebungen sollte jedoch bereits eine Realisierung der Klasse 0 zum gewünschten Ergebnis führen.

²Andere verbreitete Webbrowser bieten momentan noch keine WPAD-Unterstützung an. Für Mozilla existiert zwar ein entsprechender Patch, dessen Rollout auf den Clients den Aufwand jedoch weit über den Aufwand bei einer manuellen Konfiguration der Proxy-Einstellungen erhöhen würde.

Klasse 0: DNS-basierte Variante

Um die Klasse 0 zu unterstützen, genügt es, für den Host des Proxys einen »A«-Resource-Record (oder alternativ einen »CNAME«-Resource-Record) mit dem Namen »wpad« in die lokale Zone des Nameservers einzufügen, der die DNS-Anfragen der Clients beantwortet. Das folgende Beispiel geht von einer lokalen Zone mit dem Namen »intranet.example.com« aus. Sowohl dieser Zonenname als auch die IP-Adresse des Proxys sind bei der praktischen Umsetzung des Beispiels entsprechend anzupassen.

```
$ORIGIN intranet.example.com.
wpad      IN A      ip.adresse.des.proxys
```

Das Zonenformat des Beispiels ist auf den ISC-Bind-Nameserver³ ausgelegt. Andere Nameserver verwenden möglicherweise ein anderes Zonenformat; entsprechende Details finden sich in der jeweiligen Dokumentation der Nameserver-Software.

Weiterhin muss der *TLS Authentication Proxy* so konfiguriert werden, dass er Verbindungen auf Port 80 entgegennimmt. Die Konfiguration dieser Einstellung ist in Abschnitt B.1 beschrieben.

Klasse 1: DNS- und DHCP-basierte Variante

Soll die Proxy-Lösung die Konformitätsklasse 1 des WPAD-Standards unterstützen, so muss zusätzlich zu der im vorangegangenen Abschnitt beschriebenen DNS-Konfiguration eine Konfigurationsänderung im DHCP-Server vorgenommen werden, der für die Vergabe der IP-Adressen der Clients zuständig ist.

Das folgende Beispiel geht von einem ISC-DHCP-Server⁴ aus und bezieht sich auf die Konfigurationsdatei »dhcpd.conf«. Andere DHCP-Server verfügen über analoge Konfigurationsoptionen; eine Anleitung zur entsprechenden Konfiguration des Windows-eigenen DHCP-Servers ist in der Microsoft Knowledge Base verfügbar⁵.

```
option option-252    "http://wpad.intranet.example.com/proxy.pac";
```

Ab der Version 3 des ISC-DHCPD ist die obige Syntax nicht mehr zulässig. Hier muss stattdessen folgende Konfigurationsänderung vorgenommen werden:

```
option wpad-url      code 252 = text;
option wpad-url      "http://wpad.intranet.example.com/proxy.pac";
```

Auch bei diesen beiden Beispielen ist es erforderlich, analog zur Konfiguration des Nameservers den Namen der lokalen Zone in der WPAD-URL anzupassen.

³URL: <http://www.isc.org/sw/bind/>

⁴URL: <http://www.isc.org/sw/dhcp/>

⁵URL: <http://support.microsoft.com/default.aspx?scid=kb;EN-US;252898>

Konfiguration der Testumgebung

Im Rahmen dieses Kapitels sollen die zum Einrichten einer Testumgebung für den Proxy notwendigen Konfigurationsschritte näher erläutert werden.

Als Basis für diese Testumgebung kommt dabei serverseitig ein Apache 1.3 HTTP-Server¹ in Verbindung mit `mod_SSL`² zum Einsatz. Diese Plattform erlaubt den Test von *Basic*- und *Digest Authentication* sowie der Authentifikation per X.509-Zertifikat. Zum Erzeugen von Zertifikaten kommt `OpenSSL`³ zum Einsatz.

D.1. Installation der Komponenten

Für die Installation der Komponenten sei auf deren jeweilige Dokumentation verwiesen, da diese Installation stark von der eingesetzten Betriebssystem-Plattform abhängig ist. Der Autor setzte für seine Tests z. B. Debian Linux ein, unter dem eine Installation der entsprechenden Pakete »`apache`«, »`libapache-mod-ssl`« sowie »`openssl`« mit Hilfe des Paket-Managers ausreichend ist. Andere Plattformen erfordern möglicherweise eine Übersetzung des Quellcodes.

D.2. Konfiguration des HTTP-Servers

D.2.1. Basic Authentication

Die *Basic Authentication* greift zur Prüfung von Credentials auf eine Konfigurationsdatei im Textformat zu, in der die Benutzer/Kennwort-Tupel zur Berechtigungsprüfung hinterlegt sind. Darüber hinaus muss die *Basic Authentication* in der Server-Konfiguration – oder ersatzweise in der so genannten »`.htaccess`«-Datei – aktiviert werden.

Zur Bearbeitung der Benutzerdatei existiert ein mit Apache mitgeliefertes Tool, »`htpasswd`«. Um damit beispielsweise eine neue Konfigurationsdatei mit dem Namen »`.htpasswd`« und den Benutzer-/Kennwortdaten »`alice`« / »`secret`« anzulegen, verwendet man folgenden Programmaufruf:

```
htpasswd -bc .htpasswd alice secret
```

Eine ausführliche Dokumentation der übrigen verfügbaren Optionen zeigt »`htpasswd`« beim Aufruf ohne Parameter an.

Zur weiteren Konfiguration ist im Apache-Webserver die *Basic Authentication* zu aktivieren, was hier beispielhaft mit Hilfe der Konfigurationsdatei »`.htaccess`« gezeigt

¹URL: <http://httpd.apache.org/>

²URL: <http://www.modssl.org/>

³URL: <http://www.openssl.org/>

```
AuthType Basic
AuthName "Authorization_Required"
AuthUserFile /path/to/.htpasswd
Require valid-user
```

Abbildung D.1: Konfigurationdatei ».htaccess« zur Aktivierung der *Basic Authentication*

```
AuthType Digest
AuthName "Authorization_Required"
AuthDigestFile /path/to/.htdigest
Require valid-user
```

Abbildung D.2: Konfigurationdatei ».htaccess« zur Aktivierung der *Digest Authentication*

werden soll. Eine Konfiguration über die globale Konfigurationsdatei des Servers funktioniert analog. Dazu werden im zu schützenden Verzeichnis die Befehle aus Abbildung D.1 in eine Datei mit dem Namen ».htaccess« eingefügt. »AuthUserFile« zeigt dabei auf den Pfad der zuvor erzeugten ».htpasswd«-Datei, »AuthName« gibt den zur Authentifikation zu nutzenden *Authentication Realm* vor.

D.2.2. Digest Authentication

Die Konfiguration der *Digest Authentication* ähnelt der im vorangegangenen Abschnitt beschriebenen Konfiguration stark, so dass an dieser Stelle lediglich die Teile der Vorgehensweise beschrieben werden, die sich davon unterscheiden.

Auf Grund des anderen Aufbaus der Benutzerdatei für die *Digest Authentication* existiert für deren Wartung ein separates Kommandozeilenprogramm, »htdigest«. Dessen Aufrufkonvention ähnelt der von »htpasswd«. Allerdings erfordert der Aufruf zusätzlich die Angabe des *Authentication Realm* als Parameter. Dieser geht nämlich neben Benutzername und Kennwort in den Zwischenwert *A1* der Response-Berechnung ein (siehe Abschnitt 2.2.1.2). Der vollständige Aufruf für den Realm »Authorization Required« und die Datei ».htdigest« lautet dann:

```
htdigest -c .htdigest "Authorization_Required" alice
```

Das Kennwort liest das Programm von der Standardeingabe.

Die Konfigurationsdatei (siehe Abbildung D.2) für den Apache-Webserver unterscheidet sich von der *Basic Authentication* lediglich im anders konfigurierten »AuthType«, der Pfad zur ».htdigest«-Datei ist selbstverständlich entsprechend den Gegebenheiten anzupassen.

D.3. Konfiguration des HTTPS-Servers

Die Einrichtung einer Testumgebung für die TLS-Client-Authentifikation erfordert die Konfiguration eines HTTPS-Servers. Im Folgenden wird dabei beispielhaft die Konfiguration eines mittels `mod_ssl` um die TLS-Funktionalität erweiterten Apache-Webservers beschrieben.

D.3.1. Zertifikatserzeugung

Für die Konfiguration der HTTPS-Testumgebung ist die Erstellung von drei Zertifikaten notwendig:

- Wurzelzertifikat einer CA (kurz: CA-Zertifikat)
- Zertifikat des TLS-Webservers (kurz: Server-Zertifikat)
- Zertifikat eines Beispielnutzers (kurz: Client-Zertifikat)

Die dazu notwendigen OpenSSL-Befehle werden nachfolgend kurz dokumentiert.

CA-Zertifikat

Das CA-Zertifikat wird als so genanntes selbstsigniertes Zertifikat mit dem folgenden Kommando erstellt. OpenSSL erfragt im Anschluss interaktiv die Attribute des neu zu erstellenden Zertifikats sowie ein Kennwort für den privaten Schlüssel. Das dabei angeforderte Kennwort wird mit »CA-Kennwort« bezeichnet.

```
openssl req -x509 -days 3652 -newkey rsa:2048 -out ca-cert.pem \  
-keyout ca-sec-key.pem
```

Im Anschluss befindet sich das für einen Zeitraum von zehn Jahren gültige Zertifikat in der Datei »`ca-cert.pem`«, das RSA-Schlüsselpaar in der Datei »`ca-key.pem`«.

Server-Zertifikat

Die Erstellung des Server-Zertifikats beginnt mit der Erzeugung eines so genannten Certificate-Signing-Requests mittels der folgenden Kommandozeile. Auch hier erfragt OpenSSL wieder ein Kennwort sowie die Attribute des späteren Server-Zertifikats. Wichtig ist hierbei, dass als *Common Name* der vollständige Host-Name des Servers angegeben wird. Das hier vergebene Kennwort für den privaten Schlüssel wird mit »Server-Kennwort« bezeichnet. Das *Challenge Password* kann leer bleiben.

```
openssl req -newkey rsa:2048 -out server-request.pem \  
-keyout server-temp-key.pem
```

Anschließend wird dieser Certificate-Signing-Request mit dem zuvor erstellten CA-Zertifikat als Aussteller versehen und signiert. Als Kennwort muss dabei das CA-Kennwort eingegeben werden.

```
openssl x509 -in server-request.pem -CA ca-cert.pem \  
  -CAkey ca-key.pem -req -CAserial serial.dat -CAcreateserial \  
  -out server-cert.pem
```

Das von der CA signierte Server-Zertifikat befindet sich nun in der Datei »server-cert.pem«.

Damit beim Starten des Webservers nicht jedes Mal die Eingabe eines Kennworts nötig ist, wird abschließend das Kennwort des privaten Schlüssels des Servers entfernt. An der dabei erfolgenden Eingabeaufforderung ist letztmalig das Server-Kennwort einzugeben.

```
openssl rsa -in server-temp-key.pem -out server-key.pem
```

Der Server-Schlüssel liegt nun zur Verwendung durch den Webserver in der Datei »server-key.pem« vor.

Client-Zertifikat

Dieselbe Prozedur wird nun im Anschluss für das Client-Zertifikat durchgeführt. Auch für dieses wird zu Beginn ein Certificate-Signing-Request erstellt. Für den privaten Schlüssel erfragt OpenSSL erneut ein Kennwort, das hier »Client-Kennwort« genannt wird.

```
openssl req -newkey rsa:2048 -out client-request.pem \  
  -keyout client-key.pem
```

Im Anschluss wird dieses Zertifikat analog zum Server-Zertifikat von der CA signiert. Hier ist erneut die Eingabe des CA-Kennworts nötig.

```
openssl x509 -in client-request.pem -CA ca-cert.pem \  
  -CAkey ca-key.pem -req -CAserial serial.dat \  
  -out client-cert.pem
```

Das von der CA signierte Client-Zertifikat liegt nun in der Datei »client-cert.pem« vor und muss zusammen mit dem privaten Schlüssel für die Nutzung im Webbrowser noch vom textbasierten PEM-Format in das binäre PKCS#12-Format konvertiert werden. Diese Konvertierung erfordert die Eingabe des Client-Kennworts; als *Export Password* kann ebenfalls das Client-Kennwort angegeben werden.

```
openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem \  
  -out client.p12
```

Das Client-Zertifikat steht nun in der Datei »client.p12« in einem Format zur Verfügung, das sich in den Certificate-Store der gängigen Webbrowser und des *TLS Authentication Proxy* importieren lässt.

D.3.2. Konfiguration von mod_ssl

Nach der Zertifikatserstellung muss im Apache-Webserver abschließend das Server-Zertifikat konfiguriert werden und es muss festgelegt werden, dass der Server Client-Zertifikate von der im vorangegangenen Abschnitt erstellten CA akzeptiert. Dies erfolgt durch

Einfügen folgender Konfigurationsoptionen⁴ in die Konfigurationsdatei »`httpd.conf`« des Apache-Webservers. Eventuell vorhandene gleichlautende Optionen müssen selbstverständlich aus der Konfigurationsdatei entfernt werden.

```
SSLEngine on
SSLCertificateFile /path/to/server-cert.pem
SSLCertificateKeyFile /path/to/server-key.pem
SSLCACertificateFile /path/to/ca-cert.pem
SSLVerifyClient require
SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown
```

Nach einem Neustart des Apache-Webservers erfordern Zugriffe auf den SSL-Server nun die Authentifikation per X.509-Zertifikat, die z. B. mit Hilfe des zuvor erstellten Client-Zertifikats erfolgen kann.

⁴Das Pfad-Präfix »`/path/to/`« muss hierbei selbstverständlich durch den korrekten absoluten Dateipfad ersetzt werden.

Datenbank-Schema

Die in diesem Teil des Anhangs abgedruckten Auszüge aus dem vom *TLS Authentication Proxy* verwendeten Datenbank-Schema sollen einen ersten Eindruck von der Struktur der eingesetzten Datenbank geben. Das vollständige Schema incl. der von PostgreSQL als Trigger modellierten Fremdschlüsselbeziehungen befindet sich nach der Installation des Proxys (siehe Abschnitt A.1.1) im Unterordner »/schema« des Proxy-Basisverzeichnisses.

E.1. Certificate-Authority: Tabelle »ca«

```
CREATE TABLE "ca" (  
    "name" character varying(20) NOT NULL,  
    "value" bytea,  
    CONSTRAINT "ca_pkey" PRIMARY KEY ("name")  
) WITH OIDS;
```

E.2. Benutzerverwaltung: Tabelle »users«

```
CREATE TABLE "users" (  
    "name" character varying(255) NOT NULL,  
    "id" integer NOT NULL DEFAULT nextval('"users_uid_seq"'::text),  
    "enabled" boolean NOT NULL DEFAULT 't'::bool,  
    "description" character varying(255),  
    "certificate" bytea,  
    CONSTRAINT "users_name_key" UNIQUE ("name","id"),  
    CONSTRAINT "users_pkey" PRIMARY KEY ("name"),  
    CONSTRAINT "users_uid_key" UNIQUE ("id")  
) WITH OIDS;
```

E.3. Credential-Store: Tabelle »credentials«

```
CREATE TABLE "credentials" (  
    "id" integer NOT NULL  
        DEFAULT nextval('"credentials_id_seq"'::text),  
    "host" character varying(100) NOT NULL,  
    "port" integer NOT NULL,  
    "scheme" character varying(10) NOT NULL,  
    "name" character varying(50) NOT NULL,  
    "credential" text NOT NULL,  
    CONSTRAINT "credentials_pkey" PRIMARY KEY ("id")  
) WITH OIDS;
```

E.4. Zugriffskontrolle: Tabelle »access«

```
CREATE TABLE "access" (  
    "uid" integer NOT NULL,  
    "cid" integer NOT NULL,  
    "constraint" text,  
    "owner" integer,  
    CONSTRAINT "access_pkey" PRIMARY KEY ("uid","cid")  
) WITH OIDS;
```

E.5. Konfigurationsverwaltung: Tabelle »config«

```
CREATE TABLE "config" (  
    "name" character varying(30) NOT NULL,  
    "value" character varying(255) NOT NULL,  
    "type" character(1) NOT NULL,  
    "description" character varying(255),  
    "basic" boolean NOT NULL DEFAULT 'f'::bool,  
    CONSTRAINT "config_pkey" PRIMARY KEY ("name")  
) WITH OIDS;
```

Abbildungsverzeichnis

2.1	Vereinfachter Ablauf der Authentifikation	8
2.2	Initialer Zugriff des Clients auf den Server	8
2.3	<i>Challenge</i> des Servers bei der <i>Basic Authentication</i>	8
2.4	<i>Response</i> des Clients bei der <i>Basic Authentication</i>	9
2.5	<i>Challenge</i> des Servers bei der <i>Digest Authentication</i>	10
2.6	<i>Response</i> des Clients bei der <i>Digest Authentication</i>	11
2.7	HTML-Dokument zur formularbasierten Authentifikation	12
2.8	Formularbasierte Authentifikation bei Nutzung der »GET«-Methode . . .	12
2.9	Formularbasierte Authentifikation bei Nutzung der »POST«-Methode . .	12
2.10	Einordnung von TLS in die TCP/IP-Schichtenarchitektur	14
2.11	X.509-Zertifikat	16
2.12	Schichtenarchitektur des TLS Record Protocol	18
2.13	Datenpaket nach dem Passieren der <i>Fragmentation</i> -Schicht	19
2.14	Datenpaket nach der Komprimierung	19
2.15	Datenpaket nach Verschlüsselung mittels Stromchiffre	20
2.16	Datenpaket nach Verschlüsselung im CBC-Modus	20
2.17	TLS-Handshake: Phase 1	21
2.18	TLS-Handshake: Phase 2	22
2.19	TLS-Handshake: Phase 3	23
2.20	TLS-Handshake: Phase 4	24
2.21	Aufbau des <i>Alert</i> -Datenpakets	25
2.22	Schematischer Kommunikationsablauf der »CONNECT«-Methode	26
3.1	Schematischer Ablauf der Authentifikation bei einer Stellvertreter-Architektur	29
3.2	Kommunikationsablauf bei Nutzung des HTTP-Gateways	30
3.3	„Fälschen“ eines X.509-Serverzertifikats durch den Proxy	33
3.4	Kommunikationsablauf bei Nutzung des HTTP-Proxys	34
3.5	Schematischer Ablauf der Authentifikation bei einer clientbasierten Architektur	35
4.1	Modulare Architektur des Proxys	45
4.2	Schematischer Ablauf der HTTP-Authentifikation	50
4.3	Ablauf einer HTTPS-Verbindung	64
4.4	Ablauf einer HTTP-Verbindung	64
A.1	Konfigurationsdatei »pg_hba.conf«	73

A.2	Konfigurationsdatei ».htaccess« zur Änderung des Standard-Dokuments beim Verzeichniszugriff	74
B.1	Administration: Allgemeine Einstellungen	77
B.2	Administration: Benutzerverwaltung: Übersicht	78
B.3	Administration: Benutzerverwaltung: Nutzer erstellen	79
B.4	Administration: Benutzerverwaltung: PKCS#12-Download	79
B.5	Administration: Benutzerverwaltung: Benutzer-Eigenschaften	80
B.6	Administration: Zugriffsrechte: Übersicht	81
B.7	Administration: Zugriffsrechte: Zugriffsrecht erstellen	82
B.8	Administration: Credentials: Übersicht (Hosts)	83
B.9	Administration: Credentials: Übersicht (Credentials)	84
B.10	Administration: Credentials: Kennwortbasiertes Credential erstellen	85
B.11	Administration: Credentials: X.509-Credential erstellen	86
B.12	Administration: Credentials: X.509-Credential – Eigenschaften	87
B.13	Administration: Credentials: Formularbasiertes Credential erstellen	88
B.14	Administration: Delegation	90
B.15	Administration: Credential-Liste	91
C.1	Manuelle Konfiguration der Proxy-Einstellungen	93
C.2	Konfiguration der Proxy-Auto-Configuration	94
D.1	Konfigurationsdatei ».htaccess« zur Aktivierung der <i>Basic Authentication</i>	97
D.2	Konfigurationsdatei ».htaccess« zur Aktivierung der <i>Digest Authentication</i>	97

Tabellenverzeichnis

3.1	Vergleich der verschiedenen Realisierungsvarianten	43
4.1	Blacklist für nicht zu kopierende X.509-Erweiterungen	67

Literaturverzeichnis

- [Buc01] J. Buchmann: *Einführung in die Kryptographie*
2., erweiterte Auflage, Heidelberg : Springer, 2001, ISBN 3-540-41283-2
- [Buc03] J. Buchmann: *Public-Key-Infrastrukturen*
Vorlesung im Sommersemester 2003 an der TU Darmstadt
URL: <http://www.informatik.tu-darmstadt.de/TI/Lehre/SS03/Vorlesung/PKI.html>
- [Cci88] CCITT: *Recommendation X.509: The Directory - Authentication Framework*, 1988
- [Cho02] P. Chown: *Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)*
RFC 3268, URL: <http://www.ietf.org/rfc/rfc3268.txt>
- [Coa99] K. A. L. Coar, D. R. T. Robinson: *The WWW Common Gateway Interface – Version 1.1*, Internet Draft, 1999
URL: <http://cgi-spec.golux.com/draft-coar-cgi-v11-03-clean.html>
- [Cow89] A. P. Cowie: *Oxford Advanced Learner's Dictionary of Current English*
4., neubearb. Auflage, Berlin : Cornelsen und Oxford University Press, 1989, ISBN 3-464-05511-6
- [Die99] T. Dierks, C. Allen: *The TLS Protocol – Version 1.0*
RFC 2246, URL: <http://www.ietf.org/rfc/rfc2246.txt>
- [Din02] DIN Deutsches Institut für Normung e.V.: *Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten*
DIN EN ISO 9241, 2002
- [Eck01] C. Eckert: *IT-Sicherheit : Konzepte – Verfahren – Protokolle*
München : Oldenbourg, 2001, ISBN 3-486-25298-4
- [Fie99] R. Fielding u. a.: *Hypertext Transfer Protocol – HTTP/1.1*
RFC 2616, URL: <http://www.ietf.org/rfc/rfc2616.txt>
- [Fra99] J. Franks u. a.: *HTTP Authentication: Basic and Digest Access Authentication*
RFC 2617, URL: <http://www.ietf.org/rfc/rfc2617.txt>

- [Fre96] N. Freed, N. Borenstein: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*
RFC 2045, URL: <http://www.ietf.org/rfc/rfc2045.txt>
- [Gau99] P. Gauthier u. a.: *Web Proxy Auto-Discovery Protocol*
Internet Draft, URL: <http://www.web-cache.com/Writings/Internet-Drafts/draft-ietf-wrec-wpad-01.txt>
- [Gin04] T.-A. Ginkel, T. Straub: *Secure Delegation of WWW Credentials: A Practical Approach*. In (J. von Knop, W. Haverkamp, E. Jessen Hrsg.): Proceedings 18. DFN-Arbeitstagung über Kommunikationsnetze. Düsseldorf, 02.-04. Juni 2004. (Im Druck)
- [Hic94] K. Hickman: *The SSL Protocol*
URL: http://wp.netscape.com/eng/security/SSL_2.html
- [Kha00] R. Khare, S. Lawrence: *Upgrading to TLS Within HTTP/1.1*
RFC 2817, URL: <http://www.ietf.org/rfc/rfc2817.txt>
- [Kli03] V. Klíma, O. Pokorný, T. Rosa: *Attacking RSA-based Sessions in SSL/TLS*
Cryptology ePrint Archive, Report 2003/052, 2003
URL: <http://eprint.iacr.org/2003/052>
- [Kos04] M. Koster: *The Robots Exclusion Protocol*
URL: <http://www.robotstxt.org/wc/exclusion.html>
- [Kra97] H. Krawczyk, M. Bellare, R. Canetti: *HMAC: Keyed-Hashing for Message Authentication*
RFC 2104, URL: <http://www.ietf.org/rfc/rfc2104.txt>
- [Kri00] D. Kristol, L. Montulli: *HTTP State Management Mechanism*
RFC 2965, URL: <http://www.ietf.org/rfc/rfc2965.txt>
- [Luo98] A. Luotonen: *Tunneling TCP Based Protocols Through Web Proxy Servers*
Internet Draft,
URL: <http://www.web-cache.com/Writings/Internet-Drafts/draft-luotonen-web-proxy-tunneling-01.txt>
- [Net96] Netscape Communications Corp.: *Navigator Proxy Auto-Config File Format*, URL: <http://wp.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html>
- [Nis93] NIST: *Secure Hash Standard*
National Institute of Standards and Technology, U.S. Department of Commerce, 1993
URL: <http://www.itl.nist.gov/fipspubs/fip180-1.htm>

- [Nis94] NIST: *Digital Signature Standard*
National Institute of Standards and Technology, U.S. Department of
Commerce, 1994
URL: <http://www.itl.nist.gov/fipspubs/fip186.htm>
- [Pas03] A. Pashalidis, C. J. Mitchell: *A Taxonomy of Single Sign-On Systems*. In
(R. Safavi-Naini, J. Seberry Hrsg.): *Information Security and Privacy – 8th*
Australasian Conference, ACISP 2003, Wollongong, Australia, 9.-11. Juli
2003, Proceedings, Springer-Verlag (LNCS 2727), Berlin, 2003, S. 249-264
URL: <http://www.isg.rhul.ac.uk/~xrtc/cv/ssotax.pdf>
- [Rag97] D. Raggett, A. Le Hors, I. Jacobs: *HTML 4.01 Specification*
W3C Recommendation, 1997, URL: <http://www.w3.org/TR/html4/>
- [Riv78] R. L. Rivest, A. Shamir, L. Adleman: *A Method for Obtaining Digital*
Signatures and Public-Key Cryptosystems
Communications of the ACM, Vol. 21, No. 2, Feb 1978
URL: <http://doi.acm.org/10.1145/359340.359342>
- [Riv92] R. Rivest: *The MD5 Message Digest Algorithm*
RFC 1321, URL: <http://www.ietf.org/rfc/rfc1321.txt>
- [Rsa99] RSA Laboratories: *PKCS 12 v1.0: Personal Information Exchange Syntax*
URL: <http://www.rsasecurity.com/rsalabs/pkcs/pkcs-12/>
- [Sch96] B. Schneier: *Angewandte Kryptographie*
1. Auflage, Bonn : Addison-Wesley, 1996, ISBN 3-89319-854-7
- [Tan03] A. S. Tanenbaum: *Computernetzwerke*
3., überarbeitete Auflage, München : Pearson-Studium, 2003, S. 52-55