

Sicherheitsmechanismen und Proxifizierung in Verteilten Systemen

Tim Paehler

12. Mai 1998

Inhaltsverzeichnis

1	Grundlagen: TCP/IP-Netzwerke	2
1.1	Das Schichten-Modell in TCP/IP-Netzwerken	2
1.2	Netzwerk- und Transportschichtprotokolle: IP, TCP	3
1.2.1	IP	3
1.2.2	TCP	4
1.3	Protokolle der Anwendungsschicht: HTTP, IIOP	4
1.3.1	HTTP	5
1.3.2	IIOP	5
1.4	Aufbau und Funktion eines Proxyservers	7
2	Sicherheitsmechanismen in verteilten Systemen	8
2.1	Was ist Sicherheit?	8
2.2	Sicherheitsmechanismen in TCP/IP-Netzwerken	9
2.2.1	Sicherheitsmechanismen auf Netzwerk- und Transportschicht	9
2.2.2	Sicherheitsmechanismen auf Anwendungsebene	10
2.3	Das Sicherheitsmodell von Java	11
2.3.1	Einschränkungen für Applets	11
2.3.2	Umgehung und Aufhebung der Einschränkungen	13
2.4	Sicherheitsmodelle auf CORBA-Ebene	14
2.4.1	Der CORBA Security Service	14
2.4.2	Orbix Wonderwall	16

1 Grundlagen: TCP/IP-Netzwerke

Verteilte Systeme benötigen für ihre Implementation ein funktionstüchtiges Netzwerk. Da wir uns im folgenden mit verteilten Systemen beschäftigen wollen, die auf TCP/IP-Protokollen basieren (sämtliche Internet-Anwendungen etwa tun dies), ist es sinnvoll, zunächst einen Blick auf die zugrundeliegende Technologie zu werfen.

1.1 Das Schichten-Modell in TCP/IP-Netzwerken

Die Komplexität moderner Computersysteme macht es nahezu unmöglich, jede Ebene von der Anwendungssoftware bis zu den in integrierten Schaltkreisen fließenden Strömen verständnismäßig zu durchdringen. Es hat sich deshalb durchgesetzt, die in einem Computer ablaufenden Prozesse in verschiedene Schichten einzuteilen und diese zu weitgehend separat zu betrachten. Das gleiche Maß an Komplexität gilt es in Netzwerken von Computern zu bewältigen, wo der Datenaustausch ebenfalls von der Programmebene bis zur physikalischen Verbindung reicht.

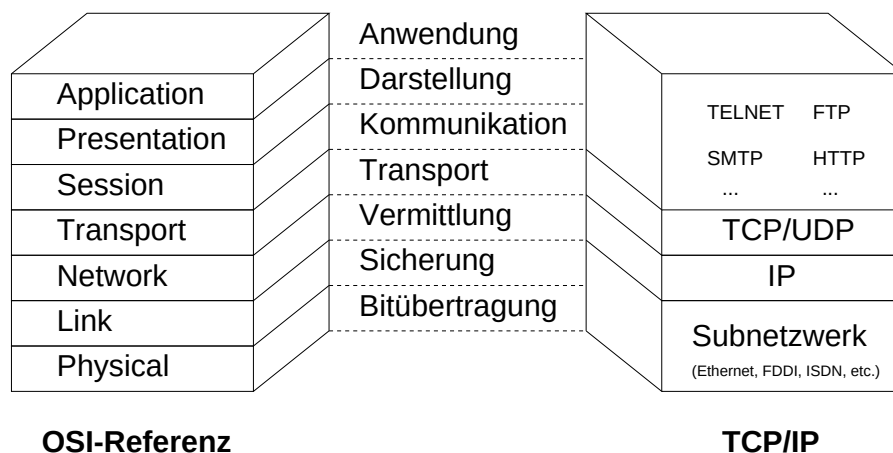


Abbildung 1: Vergleich zwischen OSI-Referenzmodell und TCP/IP-Architektur

Da der Datenaustausch zwischen möglicherweise völlig verschiedenen Rechnern einer strenger Standardisierung bedarf, als dies bei lokalen Systemen der Fall ist, hat die Internationale Standardisierungs-Organisation (ISO) frühzeitig in einem Referenz-Modell festgelegt, wie diese Einteilung in Schichten (Layers) idealerweise aussehen sollte [San93]. Diese Referenzimplementation ist als das OSI/ISO-Schichten-Modell bekannt und hat in der Entwicklung von Netzwerkprotokollen weite Beachtung erlangt. Die bekannteste Protokoll-Familie, die sich

an diesem Modell orientiert, ist TCP/IP. Genaugenommen sind - wie die Gegenüberstellung zum Referenzmodell (Abb.1) zeigt - TCP und IP nur zwei Protokolle, die die Verbindung auf Netzwerk- und Transport-Ebene regulieren; im allgemeinen bezeichnet man mit TCP/IP aber auch die Protokolle der darüberliegenden Anwendungsschicht z.B. HTTP, Telnet, FTP etc., da diese mit ihnen in engem Entwicklungszusammenhang stehen.

Zum Verständnis der Frage, wo (also in welcher Schicht) Netzwerksicherheit realisiert werden soll, ist es notwendig, den grundlegenden Aufbau des TCP/IP-Protokollstapels zu verstehen. Betrachten wir ihn nun (in unvollständiger, da exemplarischer Weise) von der Netzwerkschicht an bis zur Anwendungsschicht.

1.2 Netzwerk- und Transportschichtprotokolle: IP, TCP

1.2.1 IP

Das Internet Protocol (IP) bildet die Basis der TCP/IP-Familie. Es ist dabei vor allem für die Datenübermittlung zwischen zwei beliebig voneinander entfernten vernetzten Rechnern zuständig. Die Basiseinheit der Übermittlung ist dabei das Datagramm oder auch Paket (Packet), in dessen Kopfzeile (Abb.2) sich unter anderen die Information über Quell- und Zieladresse befinden. Die bekannte 4-stellige Form der IP-Adresse (z.B. 128.176.183.193) ergibt sich dabei aus dem Zusammensetzen einer Netzwerkadresse (im Beispiel 128.176.0.0) und einer Rechneradresse (0.0.183.193) in diesem Netzwerk, wobei die Grenze zwischen Netzwerkadresse und Rechneradresse je nach Netzwerkgröße variieren kann. Diese IP-Adresse erlaubt es nun, Rechnern oder anderen Knotenpunkten, die sich

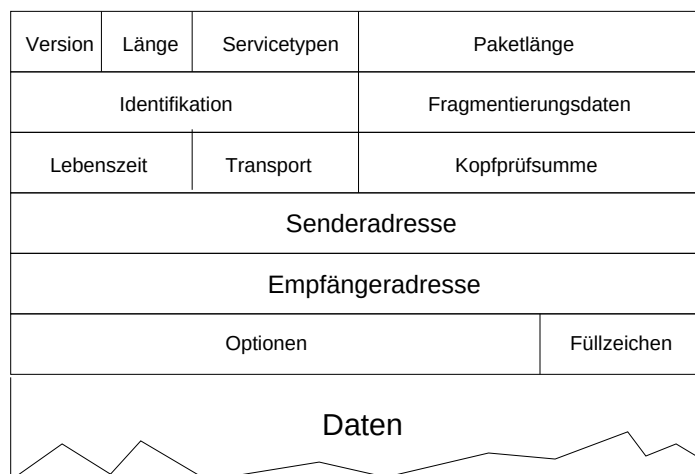


Abbildung 2: IP-Protokollkopf

auf der Strecke zwischen Ursprung und Ziel befinden, das Paket an die richtige

Adresse weiterzuleiten. Dieser Prozeß, den man als Routing bezeichnet, ist eine Besonderheit von IP gegenüber anderen Protokollen, die es besonders für große Netzwerke prädestinieren, wo Rechner selten in direktem physikalischen Kontakt zueinanderstehen. Darüberhinaus ist es möglich, bei “Staus” oder Ausfällen dynamisch eine Umleitung über andere Knoten zu empfehlen, was die Robustheit des Transfers erhöht. Die Reisepunkte (die ‘Hops’) eines IP-Pakets kann man sich z.B. mit dem Systemprogramm `traceroute` ansehen.

Wichtig zu erwähnen bleibt, daß IP selbst ein *verbindungsloses* Protokoll ist; Ein Rechner, der IP-Pakete aussendet, kann innerhalb dieser Schicht nicht feststellen, ob sie ihr Ziel erreichen. (Es ist in etwa, als würde er seine Daten per Post verschicken.) Wie wir sehen, wird die Verbindung allerdings in der Regel von der nächsthöheren Schicht sichergestellt.

1.2.2 TCP

Das Transmission Control Protocol (TCP) übernimmt die Aufgabe der Herstellung einer Verbindung. Eine TCP-Verbindung kann man sich ähnlich einer Telefonverbindung vorstellen, bei der die Partner sofort bemerken, wenn die Leitung unterbrochen ist. Da ein Rechner in der Lage sein muß, mehrere Verbindungen (unter Umständen zum selben Rechner) gleichzeitig offenzuhalten, kommt neben der IP-Adresse ein zusätzlicher Parameter hinzu, der das Ziel und den Ursprung eines Datenaustauschs eindeutig festlegt: der Port. Ein Rechner hat dem Standard nach 65536 nummerierte Ports, von denen einige (sogenannte well known ports) für spezielle Dienste reserviert sind. So ist ein Rechner normalerweise über Port 23 per Telnet erreichbar, auf Port 25 gehen E-Mails per SMTP ein und HTTP-Anfragen werden auf Port 80 gestellt (dies sind wie erwähnt alles Protokolle aus der TCP/IP-Familie, die im Schichtenmodell oberhalb von TCP liegen, und ihre Port-Bindung nutzen). Die Implementierung von TCP-Unterstützung auf einem Rechner findet in der Regel durch die komplette oder teilweise Übernahme der BSD-Socket-Bibliothek statt, in der eine TCP-Verbindung einfach wie der Datenstrom aus einer oder in eine Datei behandelt wird, was die Programmierung von Netzwerkanwendungen nach dem Verbindungsaufbau auf reine Ein- und Ausgabeoperationen reduziert.

1.3 Protokolle der Anwendungsschicht: HTTP, IIOP

Als Beispiele für Protokolle der Anwendungsschicht und mit Blick auf die folgenden Betrachtungen zur Sicherheit wollen wir nun noch HTTP, das Protokoll des WWW, und IIOP, das Protokoll des CORBA-Standards, betrachten.

1.3.1 HTTP

Das Hypertext Transfer Protocol (HTTP) [Rfc1945] ist trotz seiner enormen informationstechnischen Relevanz ein denkbar einfaches Protokoll, das in der ersten Version lediglich 3 Grundbefehle (GET, HEAD und POST) umfaßte.

Obwohl HTTP zum Verbindungsaufbau TCP benutzt und damit zunächst per definitionem ein verbindungsorientiertes Protokoll ist, wird die Verbindung für gewöhnlich nur so lange aufrechterhalten, wie ein Client eine Anfrage stellt, und darauf eine Antwort vom Server erhält (oder ein Timeout auftritt). Dies erklärt die statische Natur von HTML-Seiten, die sich lediglich durch Speicherung auf Client- oder Serverseite (etwa durch JavaScript oder CGI-Programme) umgehen läßt. HTTP wird deshalb auch als *zustandsloses* Protokoll bezeichnet.

Um eine Ressource (i.a. ein HTML-Dokument) innerhalb von HTTP zu adressieren verwendet man den für mehrere Protokolle benutzten sog. Uniform Resource Locator (URL) (z.B. `http://wwwmath.uni-muenster.de/index.html`), der sich aus der Benennung eines Protokolls (`http`), einer IP-Adresse bzw. einem Namen, der zu einer solchen aufgelöst werden kann, (`wwwmath.uni-muenster.de`) und einem Dateipfad (`/index.html`) besteht. Ein einfacher HTTP-Request kann z.B. durch das Systemprogramm `telnet` emuliert werden:

```
# Eingabe des Nutzers:
>telnet wwwmath.uni-muenster.de 80

# Verbindungsaufbau und Meldung des Erfolgs:
Trying 128.176.181.16...
Connected to RUTHERFORD.UNI-MUENSTER.DE.
Escape character is '^]'.

# Anforderung der Seite wwwmath.uni-muenster.de/index.html durch den Nutzer:
>GET /index.html

# Ausgabe der Seite durch den Server:
<TITLE>Fachbereich Mathematik, Universität Münster</TITLE>
<frameset cols="165,*" border=1 frameborder=0 framespacing=0>
...

# Schließen der Verbindung durch den Server:
Connection closed by foreign host.
```

1.3.2 IIOP

Das Internet Inter-Orb Protocol (IIOP) [Omg98] ist die Implementation des von der OMG festgelegten Standardprotokolls zur Kommunikation zwischen ORBs (GIOP) auf Internet-Ebene.

IIOP ist ein weiteres Protokoll der Anwendungsschicht und ähnelt im Aufbau in gewisser Weise HTTP [Ix97]. Auch IIOP hat einen kleinen Satz an Nachrichten (Messages), die für die Kommunikation zwischen Clients und Servern sorgen und basiert auf dem Request/Reply-Paradigma (anders als unter HTTP, wo der Browser Anfragen stellt, die der Server beantwortet, kann aber jedes Objekt gleichzeitig Client und Server sein, wodurch, die Kommunikationsmöglichkeiten prinzipiell symmetrisch sind). Die wichtigsten Nachrichten sind dabei die vom Typ Request und Reply. Beide Nachrichten bestehen aus einem Protokoll-Kopf, der neben weiterer Meta-Information den “Adressaten” enthält (beim Request einen Objekt-Schlüssel und die aufzurufende Operation, beim Reply den Schlüssel des aufrufenden Requests) und einem Protokoll-Körper (Body), in dem die übergebenen Parameter enthalten sind. Da die Request-Nachricht unter den Gesichtspunkten Sicherheit und Proxifizierung sicherlich von Interesse ist (schließlich muß anhand von ihr entschieden werden, ob eine Anfrage erlaubt ist und an wen sie weitergeleitet werden muß), wollen wir sie uns genauer ansehen (Abb.3): Eine Request-Nachricht beginnt zunächst mit dem GIOP-Header, der die Anfrage als eine GIOP-Nachricht identifiziert und neben der Versionsnummer ihren Typ (in diesem Fall also “Request”) festlegt. Es folgt der eigentliche Request-Header, in dem die interessanten Felder die folgenden sind:

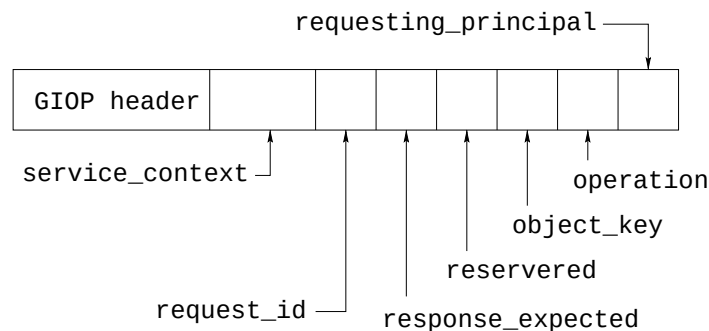


Abbildung 3: Der IIOP-Request Header

- Das **service_context**-Feld erlaubt es CORBA-Services, dem Request zusätzliche Informationen beizulegen. Diese werden allerdings von der CORBA-Kernspezifikation nicht benutzt.
- Die **request_id**, die den Request mit einer eindeutigen Nummer identifiziert. Dies ist wichtig für die richtige Zuordnung der Antwort des Servers.
- Das Feld **response_expected**, das angibt, ob der Client eine Antwort erwartet, dies ist etwa beim Aufruf einer **oneway**-Operation nicht der Fall.
- Der **object_key**, mit dem das Zielobjekt eindeutig ermittelt werden kann.

- Das `operation`-Feld, das die aufzurufende Operation enthält.
- Das `requesting_principal`-Feld, das die Benutzerkennung der Person enthält, die den Client gestartet hat.

Der daraufhin folgende Request-Body enthält die zur Ausführung der Operation nötigen Parameter.

Um ein CORBA-Objekt unter IIOP weltweit eindeutig zu adressieren wird ein ähnliches Konstrukt wie ein URL verwendet: die Interoperable Object Reference (IOR). Da IIOP im Gegensatz zu HTTP keine textbasierte, sondern eine numerische Darstellung wählt, sieht die verwendete URL weniger intuitiv aus. Hier ein Beispiel:

```
IOR:010000000e00000049444c3a48656c6c6f3a312e300000000100000000000003e00
000001010040130000006d6f7365732e6d75656e737465722e636f6d0000891300001a00
00004f422f49442b4e554d0049444c3a48656c6c6f3a312e30003000
```

Betrachtet man diese Datei jedoch mit einem Hilfsprogramm (z.B. `iortool` von Iona, oder `iordump` von OOC), das die Zahlen - wo möglich - in die zugehörigen ASCII-Zeichen umwandelt, so erhält man die folgenden Informationen [Ow97]:

```
RXR:_____%0eIDL:Hello:1.0_____%01_____>_%01_____%13moses.muenster.c
om__%13%89_____%1a0B/ID+NUM_IDL:Hello:1.0_0_
```

Auf die einzelnen Felder soll hier nicht im Detail eingegangen werden; wichtig zu wissen ist jedoch, daß neben der TCP-Adressierung (Rechnername/IP-Adresse und Port) noch weitere Informationen übergeben werden, darunter auch eine Zeichenkette (der `object_key`), die von dem zugehörigen ORB lokal interpretiert wird. Diese Zeichenkette wird im Standard nicht spezifiziert und ist damit für das Protokoll undurchsichtig ("opaque").

1.4 Aufbau und Funktion eines Proxyserver

Ein Proxyserver (lat. *proximus*: der Nächste, engl. *Proxy*: Stellvertreter) bildet in der Regel die Funktionalität eines entfernten Servers auf sich ab und leitet Anfragen gegebenenfalls an diesen weiter. Die Funktion eines Proxies läßt sich am besten am konkreten Beispiel eines HTTP-Proxies erklären, wie er in vielen Netzen eingesetzt wird (Abb.4):

Will ein Benutzer in seinem Browser eine WWW-Seite laden, so richtet er seine Anfrage nicht direkt an die Adresse, unter der sich die Seite befindet, sondern er teilt dem Proxyserver mit, daß er von ihm die Seite gerne hätte. Der Proxyserver nimmt nun die Anfrage entgegen und beantwortet sie entweder direkt aus seinem

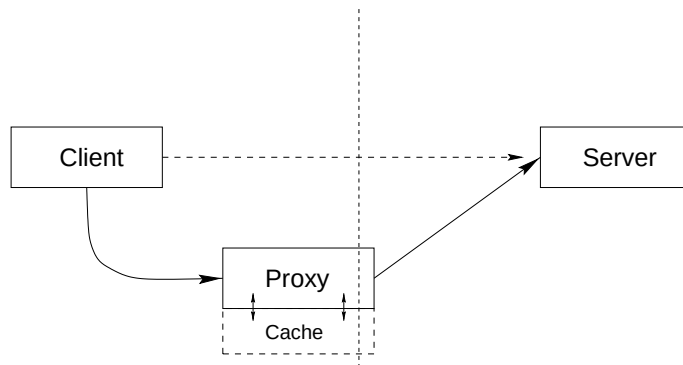


Abbildung 4: Die Funktion eines Proxy-Servers

Zwischenspeicher oder er kontaktiert die originale Adresse und reicht das Dokument dann an den Client weiter. Die Abbildung der entfernten Daten auf eine lokale Entsprechung wollen wir im folgenden als “Proxifizierung” (engl. Proxification) bezeichnen

Ein Vorteil dieses Verfahrens liegt auf der Hand: Wollen mehrere Clients auf ein statisches Dokument zugreifen, so werden die Anfragen im Durchschnitt schneller beantwortet. Der zweite Vorteil ist, daß der Rechner des Clients nicht notwendigerweise über eine eigene Verbindung zum Internet verfügen muß, was aus aufwands- wie sicherheitstechnischen Gründen sinnvoll sein könnte. Mit dem zweiten Aspekt werden wir uns in den folgenden Kapiteln gründlicher auseinandersetzen.

2 Sicherheitsmechanismen in verteilten Systemen

2.1 Was ist Sicherheit?

Die üblichen Definitionen der Computer-Sicherheit unterteilen den Begriff “Sicherheit” im allgemeinen in die folgenden Unterpunkte, die in einem sicheren System erfüllt sein müssen[Gar96][Os96]:

- **Vertraulichkeit** (Confidentiality):
Informationen dürfen nicht in die Hände Unbefugter gelangen.
- **Datenintegrität** (Data integrity):
Daten und Informationen dürfen nicht ohne Zustimmung des Besitzers geändert oder gelöscht werden.
- **Verantwortlichkeit** (Accountability):
Zu jeder getätigten Aktion muß es einen Verantwortlichen geben, der für die

Folgen belangt werden kann. Ein für kommerzielle Systeme wichtiger Spezialfall ist die *Unwiderlegbarkeit* (Non-Repudiation), d.h. das Stattfinden eines Vorgangs muß bewiesen werden können.

- **Verfügbarkeit** (Availability):

Daten und Informationen müssen autorisierten Benutzern jederzeit zugänglich sein. Aktionen, die dies unterbinden, werden als “Denial of Service”-Angriffe bezeichnet.

Während die letzten beiden Punkte dieser Aufzählung eher technische Problemstellungen sind, beziehen sie die beiden ersten Punkte auf die zentrale Frage “Wer darf was?”. Ein sicheres System erkennt man demzufolge im Gegensatz zu einem offenen System daran, daß der Grundsatz hier nicht “erlaubt ist, was nicht ausdrücklich verboten ist” lautet, sondern “verboten ist, was nicht ausdrücklich erlaubt ist”. In dem speziellen Fall des Zugriff auf spezielle Ressourcen werden dabei die Regeln, was wem erlaubt ist, häufig in sogenannten “Access Control Lists” (ACLs) festgehalten. Eine solche werden wir später noch in einem Beispiel behandeln.

2.2 Sicherheitsmechanismen in TCP/IP-Netzwerken

2.2.1 Sicherheitsmechanismen auf Netzwerk- und Transportschicht

Da auf der Netzwerkschicht nur Rechner miteinander verbunden werden, können Sicherheitsmechanismen auf dieser Ebene nur die Frage beantworten: “Welcher Rechner darf mit welchem Rechner in Kontakt treten?”, weshalb wenige Sicherheitsmechanismen auf IP-Ebene eingreifen. Interessanter wird dies schon auf Transportebene, da über TCP-Verbindungen konkrete, an well known Ports gebundene Dienste angefordert werden können. So kann z.B. die Forderung “Kein Rechner soll E-Mail an einen anderen Rechner als unseren Mail-Host schicken” bereits auf TCP-Ebene realisiert werden: Es wird einfach auf allen Rechnern außer dem Mail-Host der Port 25 dem Zugriff von außen verweigert.

Auf UNIX-Systemen kann dies etwa durch den TCP-Wrapper `tcpd` erfolgen, der alle eingehenden Verbindungen überwacht und mit einer Liste von erlaubten Verbindungen vergleicht, bevor er sie an die eigentlichen Dienstprogramme weitergibt. Man spricht bei dieser Art der Zugriffskontrolle im allgemeinen auch von einer ‘Firewall’.

Nicht möglich sind auf dieser Ebene allerdings Verschlüsselung von Daten (im Prinzip ist es jedem Rechner, der auf IP-Datagramme über darunterliegende Protokolle zugreifen kann, möglich, ihren Inhalt auszulesen) und Authentisierung von Benutzern (die Protokolle berücksichtigen nur Rechner und Ports, keine Benutzer). Die Implementation dieser wichtigen Sicherheitsmerkmale muß also auf höheren Schichten stattfinden.

2.2.2 Sicherheitsmechanismen auf Anwendungsebene

Zur Verschlüsselung werden im allgemeinen zwei verschiedene Mechanismen verwendet: secret-key-Mechanismen und public-key-Mechanismen. Beim erstgenannten Verfahren verfügen sowohl Sender als auch Empfänger über einen identischen, geheimgehaltenen Schlüssel (man spricht auch von einem symmetrischen Ansatz), der die Ver- und Entschlüsselung von Daten erlaubt.

Populärer sind in letzter Zeit die Mechanismen zur Verschlüsselung mittels Public-Key-Cryptography-Verfahren geworden. Dabei wird in einem hier nicht weiter erklärten kryptografischen Verfahren ein öffentlicher Schlüssel mit einem dazugehörigen privaten (geheimen) Schlüssel erzeugt. Der public key kann dabei Daten so verschlüsseln, daß sie nur mit dem dazugehörigen private key entschlüsselt werden können. Umgekehrt können mit dem private key Daten verschlüsselt werden, deren Entschlüsselung nur mit dem public key möglich ist. Der öffentliche Schlüssel läßt dabei keine direkten Rückschlüsse auf den zugehörigen Privat-Schlüssel zu, so daß seiner Öffentlichmachung nichts im Wege steht. Praktischerweise läßt sich durch diesen Mechanismus sowohl ein sicheres Ver-

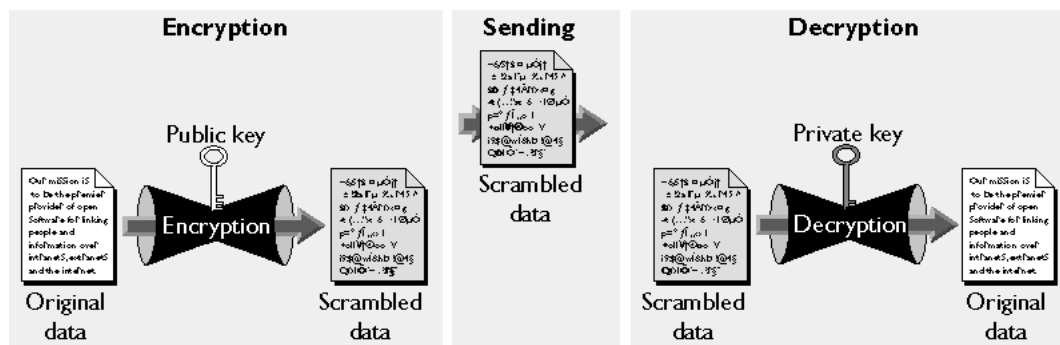


Abbildung 5: Verschlüsselte Übertragung einer Nachricht

fahren zur verschlüsselten Übermittlung von Daten als auch ein Verfahren zur Authentisierung entwickeln (Betrachtung am Beispiel des RSA-Verschlüsselungsmechanismus):

1. **Verschlüsselte Übertragung (Abb.5):** Die originalen Daten werden zunächst durch den öffentlichen Schlüssel verschlüsselt. Dann werden sie dem Besitzer des privaten Schlüssels übermittelt, welcher sie mit diesem entschlüsselt.
2. **Authentisierung von Nachrichten (Abb.6):** Der Nachricht wird durch einen öffentlich zugänglichen Algorithmus eine eindeutige Prüfsumme zugeordnet, diese Prüfsumme wird dann mit dem privaten Schlüssel verschlüsselt, wobei durch die Einzigartigkeit der Kombination aus Nachricht

und Schlüssel daraus eine “digitale Unterschrift” wird. Diese Signatur wird an die Nachricht angehängt und dem Empfänger übermittelt, welcher durch Zugriff auf denselben Algorithmus und den öffentlichen Schlüssel die Gültigkeit der Signatur überprüfen kann.

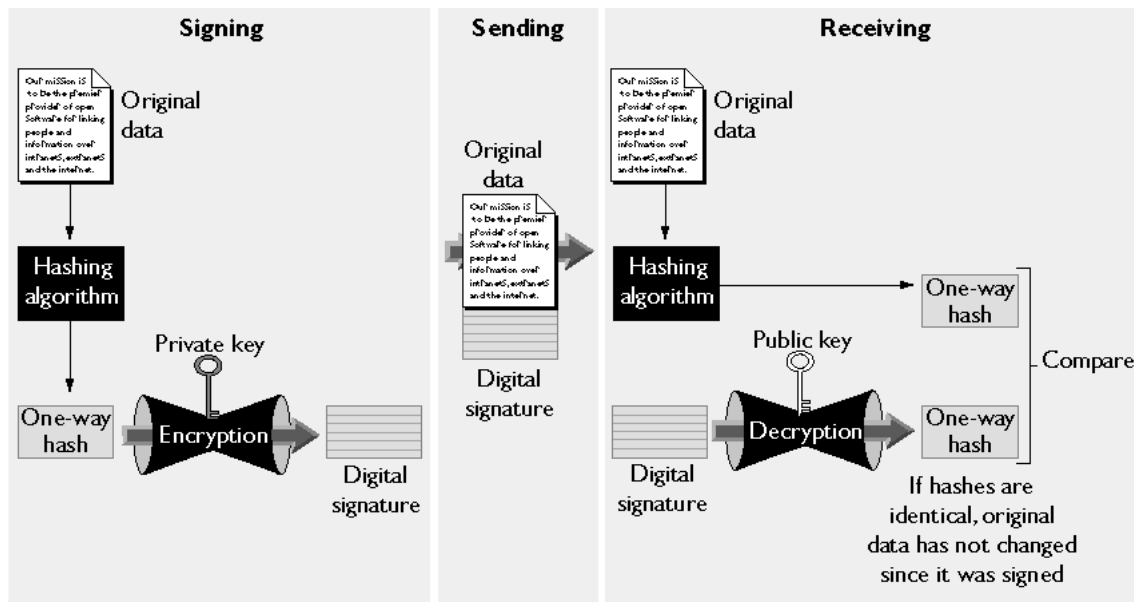


Abbildung 6: Digitales Signieren einer Nachricht

Die Anwendung dieser Verfahren sind vielfältig: E-Mails, Programm-Code-Archive und auch Login-Verbindungen (`ssh`) benutzen Public-Key-Verfahren, um Daten zu verschlüsseln oder zu signieren. Letzteres wird uns im folgenden Kapitel begegnen.

2.3 Das Sicherheitsmodell von Java

2.3.1 Einschränkungen für Applets

Java nimmt in verteilten Netzwerkumgebungen insofern eine herausragende Rolle ein, da es durch den hardwareunabhängigen Bytecode möglich ist, fremde Programme über das Netzwerk zu laden und auf der lokalen Maschine auszuführen. Daß diese Möglichkeit des Zugriffs in einem sicheren System mit äußerst strengen Einschränkungen einhergehen muß, dürfte klar werden, wenn man sich vorstellt, ein Java-Applet unbekannter Herkunft und Funktionalität hätte freien Zugriff auf die Dateien eines Anwenders und könnte diese nach Belieben auslesen oder verändern. Deshalb unterliegen über das Netz (im allgemeinen über HTTP) heruntergeladene Applets gegenüber lokalen Applikationen starken Beschränkungen. Folgendes ist ihnen unmöglich [Hen97]:

- Dateien auf dem lokalen Rechner lesen oder schreiben. Dateien oder Verzeichnisse auf dem lokalen Rechner umbenennen, löschen oder anlegen.
- Netzwerkverbindungen zu Rechnern außer dem Applet-Server selbst aufbauen.
- Netzwerkverbindungen auf dem lokalen Rechner mithören oder neue Netzwerkverbindungen annehmen.
- Informationen über den Benutzer (Name, Login-Name, Home-Directory) oder die folgenden System-Attribute ermitteln: `user.name`, `user.home`, `user.dir`, `java.home`, `java.classpath`.
- Irgendwelche Programme auf dem lokalen Rechner starten. Insbesondere haben Applets keinen Zugriff auf die Methoden `Runtime.exec()`.
- Den Java-Interpreter beenden. Ein Applet kann weder `System.exit()` noch `Runtime.exit()` aufrufen.
- Dynamische Link-Libraries nachladen. Ein Applet darf also nicht `System.load()` oder `loadLibrary()` aufrufen.
- Ein Applet darf niemals einen `ClassLoader` oder einen `SecurityManager` erzeugen.
- Applets dürfen keine Klassen außerhalb der *Core-API*-Klassen - den Packages `java.*` zugreifen oder laden.
- Applets dürfen keine Klassen definieren, wenn diese bereits auf dem lokalen System vorhanden sind. Dies verhindert, daß ein Applet seine eigenen Klassen anstelle der (sicheren) Systemklassen benutzt.
- Applets dürfen keine `Threads` außerhalb ihrer eigenen `ThreadGroup` erzeugen oder beeinflussen.

All diese Sicherheitsbestimmungen werden dabei im Kontext des Applets (z.B. über `applet.getAppletContext()` erreichbar), in dem das Applet abläuft von zwei zentralen Instanzen kontrolliert: Der `ClassLoader` [Js] und der `SecurityManager`. Der `ClassLoader` überwacht das Laden des Bytecodes der Klassen und die Zuordnung zu den jeweiligen Namensräumen. Da ein Applet nicht in der Lage sein soll, Systemklassen zu überschreiben oder die Klassen anderer Applets zu manipulieren, wird jedem Applet ein eigener separater Namensraum zugeordnet. Zusätzlich überprüft der sog. Verifier des `ClassLoaders` die Gültigkeit des Bytecodes, da es ansonsten einem absichtlich fehlerhaften Bytecode erlaubt wäre, durch Erzwingen eines Stacküberlaufs oder -unterlaufs die Virtual Machine in einen undefinierten und damit gefährdeten Zustand zu bringen. Erst wenn der

heruntergeladene Code den Verifier durchlaufen hat, wird er an die Virtual Machine weitergereicht und gelangt zur Ausführung.

Um potentiell gefährliche Aktionen während der Ausführung zu überwachen, wird der `SecurityManager` verwendet. Er kontrolliert Aktionen wie z.B. Datei- und Netzwerkzugriff und verbietet diese bei fehlender Erlaubnis. Wird ein `SecurityManager` in einem Java-Programm mittels `System.setSecurityManager()` installiert, so wird etwa bei jedem lesenden Dateizugriff zunächst die Methode `SecurityManager.checkRead()` aufgerufen, welche die Gültigkeit des Zugriffs anhand verschiedener Parameter überprüft. Bei fehlender Erlaubnis (was bei einem Applet standardmäßig der Fall ist) wird eine `SecurityException` ausgelöst, was den Vorgang unterbindet. Ein Programm-Beispiel hierzu findet sich in [Cam96].

Interessant im Hinblick auf verteilte Systeme ist sicherlich die Einschränkung der erlaubten Netzwerkverbindungen eines Applets auf den Server, von dem es heruntergeladen wurde, sowie das Verbot, eingehende Nachrichten entgegenzunehmen. Ersteres bedeutet, daß Nachrichten an andere Server über den Ursprungsserver umgeleitet werden müßten. Die zweite Einschränkung bedeutet, daß ein Applet nur Nachrichten vom Server über eine Verbindung empfangen kann, die es selbst aufgebaut hat. Damit sind zum Beispiel IIOP-Requests, die eine neue Verbindung aufbauen, für ein Applet nicht entgegennehmbar, was zur Folge hat, daß ein solches Applet keine CORBA-Server-Funktionalität implementieren kann. Glücklicherweise gibt es aber Möglichkeiten, diese Einschränkungen zu umgehen bzw. aufzuheben, ohne die Sicherheit des Gesamtsystems zu beeinträchtigen.

2.3.2 Umgehung und Aufhebung der Einschränkungen

Da der `SecurityManager` auf Browser-Seite die oben angegebenen Regeln für aus dem Netz geladene Applets bestimmt, ist es ihm möglich, die Einschränkungen individuell zu lockern. Dies macht etwa dann Sinn, wenn ein Applet von einer vertrauenswürdigen Organisation oder aus einem Intranet stammt.

Wie wird aber die Unterscheidung zwischen vertrauenswürdigen und nicht vertrauenswürdigen Applets vorgenommen? Das Verfahren bedient sich des in 2.2.2 vorgestellten Public-Key-Mechanismus: Zunächst wird der Applet-Bytecode oder ein `jar`-Archiv von seinem Erzeuger mittels eines privaten Schlüssels signiert. Daraufhin wird der zugehörige öffentliche Schlüssel einer vertrauenswürdigen Organisation (einer Certificate Authority, CA) übergeben, die nach hinreichender Überprüfung den Schlüssel unter dem Namen des Erzeugers speichert und auf Anfrage zugänglich macht. Vertraut man nun der CA, bzw. ist man der Meinung, sie bei Inkorrektheit der gelieferten Information rechtlich belangen zu können, so kann man sicher sein, daß der Code von dem angegebenen Erzeuger stammt. Vertraut man nun auch diesem Erzeuger, so kann man seinem Browser erlauben, Applets dieser Herkunft - einmalig oder dauerhaft - zusätzliche Privilegien einzuräumen. Genaues dazu findet sich unter [Net]. Da die Verteilung dieser Pri-

vilegien äußerst feinkörnig ist, besteht auch die Möglichkeit, einem Applet “ein wenig” zu vertrauen, ihm also z.B. lediglich lesenden Datei-Zugriff einzuräumen. In dem für unsere Betrachtungen interessanten Spezialfall des CORBA-Applets auf einer WWW-Seite ergibt sich noch eine weitere Möglichkeit, zumindest Zugriffe auf andere Server zu tätigen: Der Ursprungsserver fungiert als Proxyserver für IIOP-Requests. Dazu im folgenden Unterkapitel mehr.

2.4 Sicherheitsmodelle auf CORBA-Ebene

Da die CORBA-Ebene im Abstraktionsgrad weit über der Schicht der Netzwerkverbindungen liegt, ist es nicht sinnvoll, Sicherheit bezüglich CORBA-Objekten auf Netzwerkebene zu implementieren:

- Die Spezifikation verpflichtet die ORB-Hersteller nicht zur Kommunikation über TCP/IP-Netzwerke. So kann z.B. zur Optimierung, etwa wenn Client und Server auf demselben Rechner laufen, ein anderer Weg zur Übermittlung gewählt werden.
- CORBA-Objekte haben von sich aus keine Möglichkeit, sich auf einen speziellen (‘well-known’) Port festzulegen. Die meisten ORBs empfangen eingehende Requests auf dynamisch vergebenen Ports.
- Da das IIOP-Protokoll über die IOR (Interoperable Object Reference) einem CORBA-Objekt genau einen Port zuweist, erlaubt ein Port-basiertes Sicherheitssystem lediglich, den Zugriff auf ein Objekt entweder komplett zu erlauben oder zu verbieten. Häufig möchte man allerdings die Zugriffsrechte auf Methodenbasis verteilen.

Es besteht also der Bedarf an einem Konzept, das Sicherheit auf Objekt- und Methodenebene realisiert. Zwei Ansätze, die dies bewerkstelligen, sollen hier vorgestellt werden.

2.4.1 Der CORBA Security Service

Auf das sehr umfangreiche Sicherheitsmodell des CORBA-Standards kann nur kurz eingegangen werden. Die genaue Spezifikation findet sich in [Os96], ein Kurzüberblick wird in [Or97] gegeben.

Der CORBA Security Service definiert einen kompletten Rahmen, in dem die Sicherheit verteilter Systeme realisiert werden soll. Konkret handelt es sich dabei um eine Sammlung von CORBA-Schnittstellen, die entweder von ORB-Herstellern oder den Anwendungen selbst implementiert werden müssen. Die Ziele, die durch die Verwendung des Security Services erreicht werden sollen, sind die unter 2.1 genannten. Dies kann dabei auf zwei verschiedenen Ebenen erreicht werden: Die System-Ebene, wo der ORB die Verwaltung der Sicherheit übernimmt, und die

Anwendungsebene, in der die Anwendung selbst die notwendigen Schritte zur Sicherheit unternimmt. Der Hintergrund dieser Aufteilung ist, daß einerseits ORBs auch Anwendungen sicher machen sollen, die selbst über keinerlei Sicherheitsmechanismen verfügen (sog. *sicherheitsunbewußte* Anwendungen - security unaware applications); andererseits sollen Anwendungen die Möglichkeit haben, die Sicherheitsmöglichkeiten mit ihrer Semantik zu verknüpfen (etwa ein Bank-Server, der Transaktionen ab einem bestimmten Geldbetrag besonderen Sicherheitsbestimmungen unterstellt).

Die Integration des Security Service in den ORB findet wie bei allen anderen CORBA-Services durch zusätzliche Objekte statt, die sich aufgrund ihrer Darstellung in (Pseudo-)IDL als normale CORBA-Objekte ansteuern lassen. Als Beispiel und im Hinblick auf die bisher vorrangig betrachtete Frage "Wer darf was?", wollen wir schrittweise ein Szenario betrachten, in dem ein Client eine Methode in einem entfernten Server aufruft. Beide Objekte sollen dabei sicherheitsunbewußt sein, d. h. es soll lediglich die Sicherheit auf System-Ebene betrachtet werden (Abb.7):

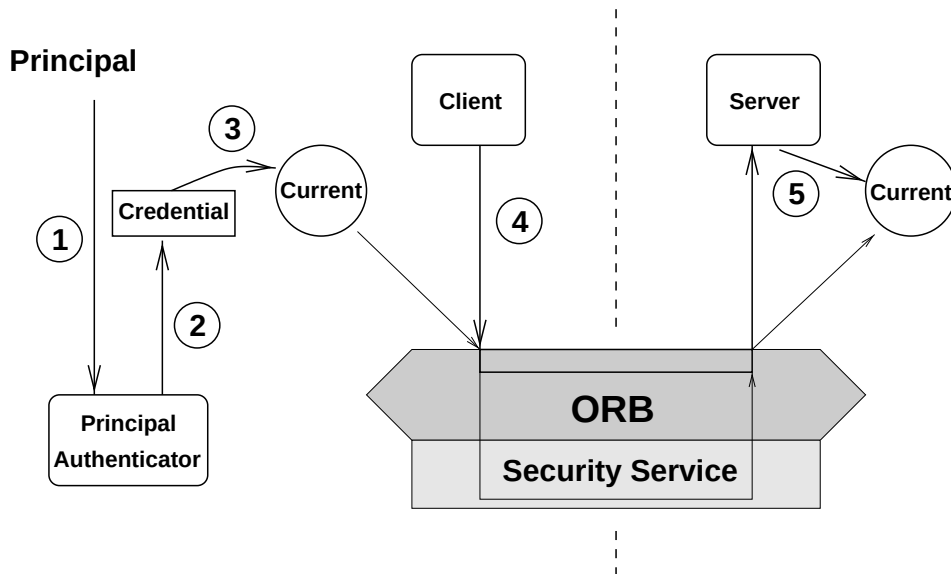


Abbildung 7: Objektsicherheit beim entfernten Methodenaufruf

1. **Anmeldung und Authentisierung des Benutzers:** Durch ein System-Programm wird der Benutzer (Principal) aufgefordert, sich - z.B. durch Eingabe eines Paßworts - beim ORB zu authentisieren.
2. **Erzeugung eines Sicherheitstickets:** Der PrincipalAuthenticator gibt ein oder mehrere beglaubigte Sicherheitstickets in Form eines Credentials-

Objekt aus, das die Identität und die Rechte (Privileges) als Attribute enthält. Dies ist der "Ausweis" des Principals.

3. **Erzeugung und Konfiguration eines Ausführungskontextes:** Durch den Aufruf von `ORB::get_current()` enthält das System eine Referenz auf den aktuellen Ausführungskontext `Current`. Diesem werden dann mittels `set_credentials()` die Sicherheitstickets übergeben.
4. **Aufruf der Methode im entfernten Objekt:** Der Methodenaufruf erfolgt normal. Neben der üblichen Information über Parameter etc. wird dem Request vom Security Service ein `service_context` (s. 1.3.2) beigefügt, der auf zusätzliche Informationen verweist. Anhand dieser Informationen extrahiert der Security Service auf Serverseite das `Current`-Objekt mit den zugehörigen `Credentials`.
5. **Ausführung der Methode durch das entfernte Objekt:** Es werden zunächst die Attribute aus den `Credentials` ausgelesen. Daraufhin wird durch das `AccessDecision`-Objekt überprüft, ob der Client auf die entsprechende Servermethode zugreifen darf. Die Überprüfung wird anhand von ORB-internen Access Control Listen vorgenommen, für deren Einrichtung ebenfalls ein Interface existiert.

Die Spezifikation des Security Service wurde im Hinblick auf die Wichtigkeit der Sicherheit verteilter Objekte weitreichend gefaßt, was jedoch zur Folge hat, daß Implementationen seitens der ORB-Hersteller zur Zeit noch auf sich warten lassen. Der nun folgende Ansatz beschreibt eine bereits bestehende, vergleichsweise einfache Alternative der Firma Iona Technologies.

2.4.2 Orbix Wonderwall

Orbix Wonderwall [Ow97] ist ein auf IIOP-Ebene arbeitender Proxyserver und Firewall. Durch Auslesen und Verändern der Felder im IIOP-Protokollkopf ist es dem Programm möglich, den Zugriff auf Objekte in einfacher Art zu kontrollieren und zu manipulieren. Wir betrachten dabei die beiden Komponenten zunächst getrennt, bevor wir auf ihre gemeinsame Behandlung beim Erstellen einer Konfigurationsdatei eingehen.

Proxifizierung

Zum besseren Verständnis zunächst also der Ablauf der Proxifizierung an einem einfachen Beispiel (Abb.8):

1. **Initialisierung des Servers und Erzeugung der IOR:** Der Server wird gestartet und erzeugt mittels `object_to_string()` eine IOR.

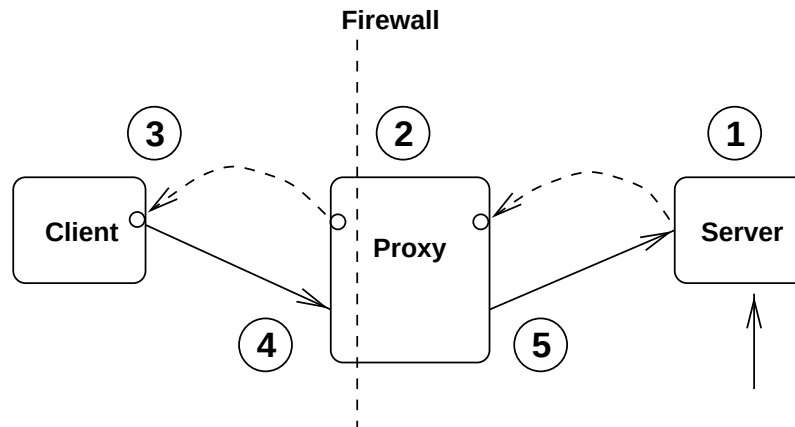


Abbildung 8: Proxifizierung durch Orbix Wonderwall

2. **Maskierung der Server-IOR:** Bevor nun die IOR an den Client übergeben wird, wandelt ein Hilfsprogramm einen Teil der IOR (in der Regel das Adreßfeld Rechnername/Port) um, indem es an die Stelle des Rechners, auf dem das Objekt abläuft, die Adresse des Proxy-Servers hinterlegt. Zusätzlich muß der Proxy wissen, wo sich das Objekt befindet.
3. **Übermittlung zum Client und Umwandlung der IOR in ein Objekt:** Die IOR wird daraufhin, wie normalerweise üblich, dem Client über eine Netzwerkverbindung übermittelt - etwa via HTTP als Parameter eines Applets oder via NFS als Datei, und dieser wandelt sie durch `string_to_object()` wieder in eine Referenz auf ein CORBA-Objekt um.
4. **Entfernter Methodenaufruf:** Will nun der Client in dem entfernten Objekt eine Methode aufrufen, so tut er dies, indem er eine TCP-Verbindung zu dem Proxy-Server öffnet, im Glauben, das Objekt befinde sich dort.
5. **Aufnahme und Weiterleitung des Aufrufs durch den Proxy:** Stattdessen "lauscht" aber ein Programm auf diesem Port, das den Request-Header (s. 1.3.2) der IIOP-Nachricht ausliest, das originale Objekt über eine Datenbank ausfindig macht und die Nachricht an den richtigen Adressaten weiterleitet.

Interessant ist sicherlich der Fall, in dem das entfernte Objekt eine sog. Factory ist, was soviel heißt, daß es eine Referenz auf ein weiteres CORBA-Objekt zurückliefert. Dies ist etwa bei einem Objekt der Fall, das die im folgenden Beispiel skizzierte Schnittstelle `Kernel::Context` des Insekten-Simulators implementiert:

```
// IDL
interface Insect {
```

```

...
};

interface Context {
    ...
    Insect createInsect(...);
};

```

In diesem Fall stellt sich bei der Proxifizierung folgendes Problem: Der Client fordert vom Server durch Aufruf von `createInsect()` ein neues Insekt an; dieser weiß jedoch nicht, daß er hinter einer Firewall abläuft und gibt die Referenz auf das Insekt in Form einer IOR zurück, die höchstwahrscheinlich auf einen Rechner verweist, auf den der Client keinen Zugriff hat.

Hier muß also erneut der Proxy eingreifen und die zur Weiterleitung eingehende Referenz auf `Insect` “on the fly” so verändern, daß sie sich ebenfalls scheinbar auf dem Proxyserver befindet. Danach übernimmt der Proxy für das Insekten-Objekt genauso die Vermittlung, wie vorher schon für das `Context`-Objekt.

Zugriffssicherung

Die Proxy-Funktionalität der Wonderwall ist zugleich auch der Ansatzpunkt für ihre Sicherheitskomponente: Es liegt nahe, den Proxy die IIOP-Requests nicht nur weiterleiten, sondern auch anhand verschiedener Kriterien filtern zu lassen. Dabei kann allein durch Auslesen der im Protokollkopf vorhandenen Felder (s. 1.3.2) bereits eine Zugriffsbeschränkung auf Objekt- und Methodenebene erreicht werden, wogegen das `principal`-Feld im Vergleich zu dem in 2.4.1 beschriebenen Ansatz aufgrund fehlender sicherer Authentisierung ein nur unzuverlässiges Auswahlkriterium darstellt.

Die Zugriffsbestimmungen werden - wie im allgemeinen üblich - in einer Access Control List festgehalten. Eine ACL besteht aus einer oder mehreren Regelsätzen (Rules), die sich auf bestimmte Einzelaspekte beziehen (s. folgendes Beispiel).

Konfiguration

Da die Wonderwall einige der in den vorherigen Kapiteln vorgestellten Konzepte implementiert, wollen wir sie auch auf technischer Ebene betrachten, um einen Überblick über die konkreten Möglichkeiten bei der Konfiguration zu gewinnen. Dazu betrachten wir die im Insekten-Simulator-Projekt benutzte Konfigurationsdatei als Beispiel:

```

#####
# file iioproxy.cf - a sample WonderWall configuration file

# general options
domain          uni-muenster.de
log requests replies

```

```

# iiop-port listening for incoming calls:
port          1670

# used by the built-in http-server:
http-port     8080
http-files    /u/paehler/public/InsectSim.2.0

# Object Database
#
# {Kernel,Sim}_Context is declared as server, so objects produced by it apply
# to the same rules as {Kernel,Sim}_Context itself:

server Kernel_Context \
    /u/paehler/public/InsectSim.2.0/tmp/Kernel_server.ior
server Sim_Context \
    /u/paehler/public/InsectSim.2.0/tmp/Sim_server.ior

# ... other objects might follow

# Access Control List
# anything not stated below is access denied by default

# we do not need any IIOP Service Contexts:
deny servicecontexts *

# control-object should only be available to a secure host:
allow object Sim_Context op getControl \
    ipaddr 128.176.182.64/255.255.255.192 \
    principal paehler
deny object Sim_Context op getControl

# all other operations for {Sim,Kernel}_Context (createInsect(),
# getStatistic() etc.) are available to the outside world:
allow object Sim_Context
allow object Kernel_Context

# if the message has not matched a rule until now, it will be
# denied automatically

```

Der sicherheitsrelevante Teil der Konfigurationsdatei ist in zwei Teile aufgespalten: Die Objekt-Datenbank, in der verschiedene IORs mit einem Namen und Eigenschaften versehen werden, und der ACL, in der den benannten Objekten die entsprechenden Zugriffsrechte zugeordnet werden. Die Einträge in der Objektdatenbank bestehen lediglich aus einem Schlüsselwort (**object**, **server**), einem Namen (**Sim.Context**, **Kernel.Context**) und einem Dateipfad, unter dem die IOR

des Objektes gespeichert ist. Das mit `Kernel_Context` benannte Objekt verweist in unserem Beispiel auf ein Programm, das das Interface `Kernel::Context` implementiert und beim Aufruf von `createInsect()` eine Referenz auf ein `Insect`-Objekt zurückgibt. Damit eine solche Referenz ebenfalls von der Wonderwall entsprechenden Zugriffs- und Weiterleitungsbestimmungen zugeordnet werden kann, wird `Kernel_Context` als `server` deklariert, was bedeutet, daß alle Objekte, die von ihm erzeugt werden den gleichen Bestimmungen unterliegen wie das Server-Objekt selbst. Für die Schnittstelle `Sim::Context`, der der symbolische Name `Sim_Context` zugeordnet wird, gilt dies ebenfalls, da sie z.B. die Referenz auf das `Control`-Objekt zurückgeben soll. Einfache CORBA-Objekte würden dagegen als `object` deklariert.

Die ACL ist im Prinzip selbsterklärend: einem `allow` oder `deny` folgt der in der Objektdatenbank festgelegte Bezeichner sowie eine Anzahl an Auswahlkriterien, die zur Anwendung der Regel erfüllt sein müssen. Bei mehreren Regeln gelten diese als mit einem logischen UND verknüpft, so kann z.B. `Sim::Context::getControl()` nur von dem Benutzer `paehler` auf einem Rechner mit der Adresse `128.176.182.(65 - 127)` aufgerufen werden.

Wird nun wie im obigen Beispiel eine Anfrage an die Wonderwall gestellt, so werden zusätzlich zu dem bereits beschriebenen Proxifizierungsprozeß die Zugriffsrechte auf das spezifizierte Objekt überprüft. Dabei wird - wie im allgemeinen üblich - die ACL von oben nach unten durchgearbeitet, bis sich eine Regel findet, die den Zugriff auf das in Frage kommende Objekt explizit erlaubt oder verbietet. Findet sich eine solche Regel, wird der Abarbeitungsvorgang abgebrochen. Das bedeutet, daß bei sich widersprechenden Regeln diejenige in Kraft tritt, die in der Datei weiter oben steht. Im konkreten Beispiel heißt das, daß die Regel `deny object Sim_Context op getControl` nur dann zum Verbot des Zugriffs führt, wenn die vorangehende Regel nicht zur Anwendung gekommen ist.

Fazit

Die Orbix Wonderwall stellt für Anwendungen mit durchschnittlichem Sicherheitsbedarf sicherlich eine einfache Alternative zum CORBA Security Service dar. Dies ist jedoch in praktischer Hinsicht mit einigen Einschränkungen versehen:

- Die Wonderwall ist eigentlich als rein serverseitige Lösung konzipiert. In der Praxis benötigt man allerdings als ORB für die Client-Klassen OrbixWeb mit Proxy-Unterstützung, da ansonsten nur die Basisfunktionalität (z.B. keine Unterstützung von Factory-Objekten) bereitsteht.
- Die Liste der erlaubten IORs wird nur beim Start des Proxy-Servers eingelesen. Das bedeutet, daß bei einer Änderung der IOR (dies ist z.B. durch die dynamische Portvergabe beim Serverneustart der Fall), der Proxy neu

gestartet werden muß. Ein solches Vorgehen ist bei der im Beispiel verwendeten Konfiguration ein erheblicher Mehraufwand gegenüber einer dynamischen Ermittlung der erlaubten IORs, da der Proxy-Prozess auf einem anderen Rechner und unter einer anderen Kennung gestartet wird als der Server.

Da die Sicherheit für CORBA-basierte Anwendungen weiterhin eine wichtige Rolle spielen wird, ist zu erwarten, daß diese Probleme entweder durch zukünftige Versionen der Wonderwall oder durch entsprechende Konkurrenzprodukte gelöst werden.

Literatur

- [San93] Michael Santifaller. TCP/IP und ONC/NFS in Theorie und Praxis. Addison-Wesley 1993
- [Rfc1945] RFC1945 HTTP/1.0-Spezifikation. Mai 1996.
<http://andrew2.andrew.cmu.edu/rfc/rfc1945.html>
- [Omg98] OMG. CORBA/IIOP Specification v2.2 Februar 1998
<http://www.omg.org/corba/>
- [Ix97] IIOP: Alternative zu HTTP. Bernhard Merkle. iX 7/97. heise-Verlag
- [Gar96] Simson Garfinkel, Gene Spafford. Practical UNIX & Internet Security. O'Reilly & Associates, Inc. 1996
- [Net] Netscape Object Signing - Establishing Trust for Downloaded Software,
<http://developer.netscape.com/docs/manuals/signedobj/trust/index.htm>
- [Hen97] Norman Hendrich. Java für Fortgeschrittene. Springer Verlag 1997
- [Js] The Java Security FAQ <http://java.sun.com/sfaq/index.html>
- [Cam96] The Java Tutorial - Object-Oriented Programming for the Internet. Addison-Wesley 1996. HTML-Version:
<http://java.sun.com/docs/books/tutorial/index.html>
- [Os96] OMG. Security Service Specification v1.0. November 1996
<http://www.omg.org/corba/>
- [Or97] Robert Orfali, Dan Harkey, Jeri Edwards. Instant CORBA. John Wiley and Sons 1997
- [Ow97] Wonderwall Administrators Guide. IONA Technologies Ltd. 1997