Arbeitsbereich
Anwendungen der Informatik in
Geistes- und Naturwissenschaften
Universität Hamburg
Vogt-Kölln-Straße 30
22527 Hamburg

Diplomarbeit

Risikoanalyse von Internet-Agenten und Maßnahmen zu Ihrer Beschränkung

Marco Martienssen

15. Januar 1999

Erklärung

Ich erkläre, daß ich diese Arbeit selbständig durchgeführt und ausschließlich die angegebenen Hilfsmittel und Quellen benutzt habe.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Vorwort

„*One of the most dangerous things that you can do with a computer that is connected to the Internet is to download a program and run it*".
– Garfinkel & Spafford

Das Internet – bis vor kurzem noch ein reines Forschungsnetz – befindet sich auf dem Weg, das Medium der „Informationsgesellschaft" zu werden. Tiefgreifende gesellschaftliche, kulturelle und wirtschaftliche Veränderungen werden damit einhergehen.

Das Internet bietet bereits jetzt eine kaum zu überschauende Anzahl von Diensten und in immer kürzer werdenden Abständen kommen neue Entwicklungen hinzu. Eines der aktuellen Schlagworte ist das des „Agenten". Agenten sind nur schwierig zu definieren. Es existieren eine Vielzahl von Definitionen. Darüberhinaus existieren neben dem Begriff des Agenten eine Vielzahl weiterer Begriffe, die gleiches zu beschreiben versuchen – *Active Content*, *Extensions*, *Mobile Code*. Allgemein akzeptiert ist der mit Agenten einhergehende Paradigmenwechsel. Agenten sind im Gegensatz zu den derzeitigen Softwareprodukten auf heterogene Umgebungen ausgerichtet, sie sind kommunikativ und möglicherweise mobil.

Neue Entwicklungen, wie die des Agenten, bieten viele Chancen. Aufgrund der mit ihnen einhergehenden tiefgreifenden Veränderungen entstehen gleichzeitig neue Risiken, denen mit geeigneten Maßnahmen zu begegnen ist.

In der Vergangenheit wurde der Sicherheitsaspekt in IT-Systemen oftmals vernachlässigt. Die Notwendigkeit von Sicherheit in der Informationstechnologie wird häufig gesehen; die mit ihr verbundenen Kosten werden jedoch nicht akzeptiert, so daß Lösungen zum Einsatz kommen, von denen bekannt ist, daß sie aufgrund von Sicherheitsmängeln nur bedingt geeignet sind. Erst erfolgreiche Angriffe führen dazu, daß Sicherheit kurzzeitig in das Bewußtsein der Beteiligten gelangt.

Die Fragestellung nach der Sicherheit einer Technologie wird darüberhinaus oftmals als Hindernis für Innovationen betrachtet. Tatsächlich aber kann gerade die Sicherheit einer Dienstleistung im IT-Bereich der ausschlaggebende Punkt sein, um deren Akzeptanz zu gewährleisten. Sicherheit stellt nur dann ein Hindernis für neue Entwicklungen dar, wenn die verwendeten Mechanismen zuviel Aufmerksamkeit – insbesondere auf Seiten des Dienstleistungsnehmers – für den laufenden Betrieb erfordern.

Für das neue Software-Paradigma des Agenten kann die Sicherheit der entscheidende Punkt in Hinblick auf die weitere Entwicklung sein. Netzwerk-Agenten, die sich mobil in verteilten Systemen bewegen, erfordern völlig neue Sicherheitskonzepte. In den derzeitigen Sicherheitskonzepten gilt es das System, auf dem die Software zur Ausführung gebracht wird, zu schützen. Mit Einzug der mobilen Agenten ist die Anforderung an die Sicherheitskonzepte zu stellen, den Agenten vor der Umgebung zu schützen, von der er zur Ausführung gebracht wird. Die Mehrheit der Arbeiten, die sich mit dem Thema Sicherheit und Agenten befassen, haben sich mit dem Schutz der Umgebung vor dem Agenten beschäftigt. Erst im Rahmen des „Foundations for Secure Mobile Code Workshop“ im Jahre 1997 wurde deutlich, daß der Schutz des Agenten vor der Umgebung das weitaus schwierigere Problem darstellt.

ROBERT GRIMM stellt in diesem Zusammenhang fest, daß „ ... *the authentication of extensions (and principals), the auditing of security relevant system events, how to counter denial of service attack, a formal model of security in extensible systems, or the assertion and certification that a given implementation actually and correctly implements a given security mode, are only touched upon or not discussed at all* [GB97].“

Ähnliches stellen DAVID CHESS et al. fest: „*It is impossible to prevent agent tampering unless trusted (and tamper-resistant) hardware is available* [...] [CGH+95].“

Ziel dieser Arbeit ist es deshalb festzustellen, welche Bedrohungen gegenüber Agentensystemen existieren, welche Risiken daraus resultieren und wie diesen zu begegnen ist. Dazu werden eine Reihe von Sicherheitsarchitekturen untersucht, die in Hinblick auf das neue Software-Paradigma des Agenten entworfen wurden. Abschließend erfolgt eine exemplarische Darstellung von integrierten Sicherheitsarchitekturen, die den Schutz bestimmter Anwendungsgebiete zum Ziel haben.

Der Schwerpunkt der Arbeit liegt auf den besonderen Gefahren, die aus dem Agenten-Paradigma resultieren. Entsprechend werden Sicherheitsbereiche, die bereits weitreichend erforscht sind, nicht angesprochen. Höhere Gewalt und organisatorische Mängel werden aus diesem Grunde nicht diskutiert. Agenten tragen hier – aus Sicht des Autors – keine neue Risiken bei.

2. Methodische Grundlagen

„*Only persons that are known to the system can execute programs on the system.*“
– CHESS

Das vorliegende Kapitel beinhaltet die methodischen Grundlagen, die in der Arbeit angewendet werden. Der erste Abschnitt führt eine Reihe von Begriffsdefinitionen ein, die im Bereich der IT-Sicherheit von Bedeutung sind. Daran schließt sich eine Darstellung der Sicherheitskriterien an, die von Agentensystemen erfüllt werden sollten. Aus den Sicherheitskriterien lassen sich Anforderungen ableiten, die im nachfolgendem Abschnitt erörtert werden. Die Abschnitte „Bedrohungsanalyse“ und „Risikoanalyse“ stellen die Vorgehensweise bezüglich der Sicherheitsanalyse des Untersuchungsgegenstandes dar.

2.1. Begriffsdefinitionen

IT-Sicherheit beinhaltet sowohl den Schutz vor unbeabsichtigten Ereignissen (engl. *safety*) als auch den Schutz vor beabsichtigten Angriffen (engl. *security*). Klassisch wird IT-Sicherheit in Computersicherheit und Kommunikationssicherheit getrennt. Eine getrennte Untersuchung beider Aspekte unterbleibt in dieser Arbeit, da beide in Agentensystemen miteinander verschmelzen. Eine nähere Darstellung des Sicherheitsbegriffs, wie er in dieser Arbeit verwendet wird, erfolgt im Abschnitt 2.2.

IT-Systeme sind *Bedrohungen* (engl. *threads*) ausgesetzt, d. h., möglichen Verletzungen der Sicherheit der Systeme. Bedrohungen können unbeabsichtigter (versehentliches Senden vertraulicher Informationen) oder vorsätzlicher (Manipulation von Informationen) Natur sein. Vorsätzliche Angriffe können aktiv (Manipulation von Informationen) oder passiv (Belauschung von Leitungen) erfolgen. Bedrohungen resultieren aus *Verwundbarkeiten* (engl. *vulnerabilities*). Eine Verwundbarkeit ist „eine Schwäche in einem System, die technisch bedingt sein oder aus dem Einsatz des Systems entstehen kann [Opp92].“ Das *Risiko* ist das Maß für die enstehenden Kosten, die sich durch eine erkannte Verwundbarkeit ergeben, die zu einem erfolgreichen Angriff genutzt werden kann. Ein Angriff ist dabei die Ausnutzung einer Verwundbarkeit um ein Computersystem zu schädigen. Risiken ergeben sich, wenn Bedrohungen auf Verwundbarkeiten stoßen. Den Risiken ist mit Gegenmaßnahmen zu begegnen. Eine Formalisierung der

Gegenmaßnahmen erfolgt durch Sicherheitsmodelle. Ein *Sicherheitsmodell* beschreibt in präziser und allgemeingültiger Form das Verhalten der sicherheitsrelevanten Funktionen.

Ein weiterer wichtiger Begriff, der sich nur schwer definieren läßt, ist Vertrauen. Es handelt sich dabei eher um eine soziale als um eine technische Komponente. Vertrauen vermischt die Ziele, die eine Entität äußert, mit seinem Verhalten bezüglich der definierten Ziele [WBS98].

2.2. Sicherheitsbegriff

Der Begriff der Sicherheit bedarf einer Definition. Die verbreiteste Definition des Begriffs besteht in der Unterteilung der Sicherheit in die Komponenten Integrität, Verfügbarkeit und Vertraulichkeit (engl. integrity, availability und confidentiality). Diese Darstellung des Sicherheitsbegriffs ist auch in den Information Technology Security Evaluation Criteria (ITSEC) und den Common Criteria (CC) wiederzufinden. Die Form der Differenzierung ist für die in der Arbeit untersuchten Agentensysteme nicht ausreichend. Im Abschnitt 2.3 werden aus diesem Grunde weitere Sicherheitskriterien herangezogen, deren Erfüllung für die Sicherheit von Agentensystemen von Bedeutung ist.

Um den besonderen Belangen, die mit Agentensystemen verbunden sind, gerecht zu werden, wird der Sicherheitsbegriff in dieser Arbeit anders definiert. Im vorherigen Abschnitt wurde auf die Unterteilung des Sicherheitsbegriffs in Computer- und Kommunikationssicherheit hingewiesen. In Agentensystemen ist eine derartige Unterteilung nicht länger möglich. Eine Unterscheidung zwischen lokalem und entferntem Code ist mit einfachen Mitteln nicht zu leisten. Die Grenzen zwischen den Kategorien verschwinden. Damit geht einher, daß eine Reihe von Annahmen, die in Hinblick auf die Bewertung der Sicherheit in IT-Systemen gemacht wurden, nicht länger haltbar sind. CHESS hat die nicht länger gültigen Annahmen in einem Artikel vorgestellt:

- ▷ Die Ausführung eines Programmes kann eindeutig einer Person zugeordnet werden und die Ausführung liegt in deren Intention.
- ▷ Ausschließlich dem System bekannte Personen können Programme innerhalb des Systems ausführen.
- ▷ Software wird von einfach zu identifizierenden Quellen bezogen.
- ▷ Programme überschreiten nur selten administrative Grenzen.
- ▷ Sicherheit wird durch das Betriebssystem durchgesetzt [Che98].

Bislang gilt es in IT-Systemen die Prozesse *unterschiedlicher* Anwendern voreinander zu schützen. Ein wesentliches Werkzeug hierbei ist die virtuelle Speicherverwaltung in ihren unterschiedlichen Ausprägungen. Mit der Einführung von Agentensystemen erwächst hingegen die Anforderung, die Prozesse *eines* Anwenders gegeneinander zu schützen.

Für Agenten, die aus unterschiedlichen Quellen bezogen und unter der Kontrolle eines Anwenders ausgeführt werden, ist es nicht länger akzeptabel, daß diese in ein und derselben Sicherheitsdomäne ausgeführt werden. Prozesse eines Anwenders müssen somit in unterschiedlichen Sicherheitsdomänen zur Ausführung gelangen.

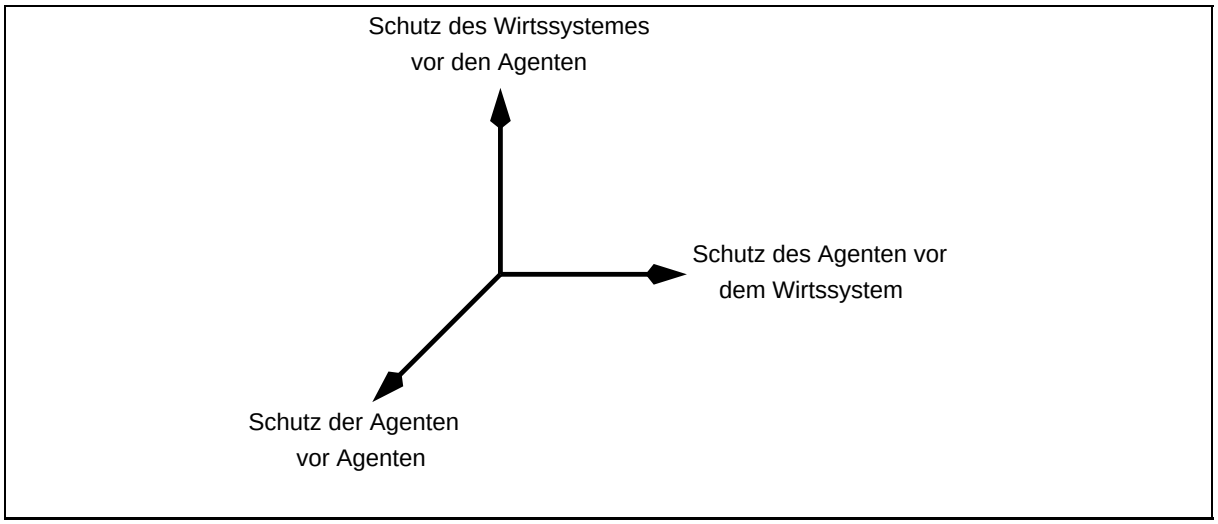

Abbildung 2.1.: Der in der Arbeit benutzte Sicherheitsbegriff für Agentensysteme zeichnet sich durch drei Dimensionen aus: Schutz des Wirtssystems vor Agenten, Schutz des Agenten vor anderen Agenten und dem Schutz des Agenten vor dem Wirtssystem.

Wesentlich schwerwiegender ist eine weitere, neue Sicherheitsanforderung. In IT-Systemen galt es bislang das ausführende System vor den Prozessen zu schützen. In Agentensystemen haben Agenten die Möglichkeit, mitsamt ihrer transitiven Hülle auf ein entferntes Wirtssystem zu migrieren, um dort die Ausführung fortzusetzen. Daraus ergibt sich für Agenten die Notwendigkeit des Schutzes vor der ausführenden Umgebung. So wie in heutigen IT-Systemen der Schutz vor maliziösen Prozessen gewährleistet werden muß, ist in Agentensystemen auch der Schutz der Agenten vor maliziösen Agenten sicherzustellen.

Aus den angeführten Anforderungen ergibt sich die Definition des Sicherheitsbegriffs, wie er in dieser Arbeit Anwendung findet:

Definition 1:

Ein Agentensystem kann als sicher bezeichnet werden, wenn

1. der Schutz des Wirtssystems vor Agenten,
2. der Schutz des Agenten vor anderen Agenten und
3. der Schutz des Agenten vor den Wirtsystemen gewährleistet ist.

Die Anforderungen können auch als die drei Dimensionen des Sicherheitsbegriffs in Agentensystemen bezeichnet werden (siehe Abbildung 2.1).

2.3. Sicherheitskriterien

Neben den bereits im obigen Abschnitt angeführten Sicherheitskriterien Integrität, Verfügbarkeit und Vertraulichkeit werden weitere Kriterien benötigt, die im folgenden vorgestellt werden. Abhängig von der gewählten Sicherheitspolitik gilt es die Kriterien in unterschiedlichem Maße zu erfüllen.

2.3.1. Funktionalität

Ziel eines Softwareentwicklungsprozesses ist die Erfüllung einer spezifierten Funktionalität. Entspricht eine Software nicht der spezifizierten Funktionalität ist ihr Verhalten nicht vorhersagbar, weshalb sie auch nicht als sicher eingestuft werden kann. In Agentensystemen ist die Funktionalität ein wesentliches Kriterium, daß es für die Entscheidung der Ausführung eines Agenten herangezogen werden muß. In Anbetracht, daß fremde bzw. unbekannte Agenten auf lokalen Systemen ausgeführt werden, ist es für eine Sicherheitsentscheidung erforderlich, daß die Funktionalität bekannt ist und erfüllt wird.

2.3.2. Integrität

Integrität ist die Qualität, die sicherstellt, daß eine Nachricht oder ein Datum nicht verändert wurde. Integrität gewährleistet Datenkonsistenz und schützt vor unbefugten Veränderungen von Informationen. Das umfaßt das Einfügen, Umstellen und Löschen von Teilinformationen als auch dem Erzeugen und Vernichten ganzer Datensätze.

Integrität in Agentensystemen bedeutet, daß sowohl der Agent wie auch seine Daten, als auch die ihn ausführende Umgebung vor Manipulationen geschützt sind.

2.3.3. Korrektheit

Korrektheit ist ein Begriff der theoretischen Informatik, der auch in der IT-Sicherheit von Bedeutung ist. Unbeabsichtigte Ereignisse werden nur dann vermieden, wenn die beteiligtem Komponenten in einem IT-System korrekt implementiert worden sind. Auf die untersuchten Agentensysteme bezogen bedeutet das, daß Agenten maliziöses Verhalten aufweisen können (z. B. durch unzulässigen Ressourcenverbrauch resultierend aus Implementationsfehlern), auch wenn dieses nicht Intention des Entwicklers war. Korrektheit ist somit ein Sicherheitskriterium für Agentensysteme, das der Erfüllung bedarf.

2.3.4. Originalität

Originalität tritt erst seit kurzem im Zusammenhang mit der Ausbreitung der Kommunikationsnetze als Sicherheitskriterium in Erscheinung. In Netzwerken ist zu gewährleisten, daß Daten transportiert, nicht aber kopiert werden. Softwareobjekte wie Agenten lassen sich leicht klonen, woraus folgt, daß Agentensysteme sicherstellen müssen, daß ein Klon vom Original unterscheidbar ist.

2.3.5. Privatheit

Die Privatsphäre verliert im „Informationszeitalter“ zunehmend an Bedeutung. Dies ist ein wesentliches Problem der heutigen IT-Systeme. Privatheit setzt sich aus den Komponenten Anonymität, Pseudonymität, Unbeobachtbarkeit und Unverkettbarkeit zusammen. Anonymität gewährleistet, daß die Identität einer Entität nicht ermittelt werden kann. Im Falle der Pseudonymität tritt eine Entität unter einer anderen Identität auf, die keinen Rückschluß auf die wahre Identität zuläßt. Alleine der Entität, oder einem vertrauenswürdigen Dritten, ist es möglich, den Zusammenhang – beispielsweise zu Beweisgründen – herzustellen. Unverkettbarkeit stellt sicher, daß unterschiedliche Aktivitäten einer Entität nicht miteinander in Bezug gesetzt werden können. Ist die Kommunikation zwischen Entitäten nicht beobachtbar, wird von Unbeobachtbarkeit gesprochen.

2.3.6. Verbindlichkeit

Verbindlichkeit ist ein Aspekt der Kommunikationssicherheit. Kommunikationspartner sollten nach einem stattgefundenen Kommunikationsaustausch nicht in der Lage sein, diesen zu leugnen. Sowohl die Leugnung des Ursprungs, als auch die Leugnung der Ankunft sind zu verhindern. Transaktionen, die mit Hilfe von Agenten vollzogen werden, müssen dem Kriterium der Verbindlichkeit genügen. Keine der beteiligten Parteien darf ihre Teilnahme leugnen können.

2.3.7. Verfügbarkeit

Verfügbarkeit impliziert, daß legitimierte Subjekte jederzeit auf jene Betriebsmittel zugreifen können, für deren Zugriff sie autorisiert sind. In IT-Systemen, in denen die Verfügbarkeit gewährleistet ist, darf es nicht zu einer unbefugten Vorenthaltung von Informationen oder Betriebsmitteln kommen.

2.3.8. Vertraulichkeit

Vertraulichkeit bedeutet, daß auf Informationen lediglich autorisierte Subjekte Zugriff erhalten. Vertraulichkeit beruht auf der korrekten Identifizierung eines Subjektes durch

einen Authentifikationsprozeß und der damit verbundenen Autorisierung. Vertraulichkeit umfaßt sowohl die Datenvertraulichkeit als auch die Vertraulichkeit des Verkehrsflusses. Die Einführung der Agentensysteme erweitert die Anforderungen in Bezug auf die Vertraulichkeit dahingehend, daß hier auch die Vertraulichkeit der Ausführung zu gewährleisten ist.

Das Beobachten der Existenz oder Abwesenheit von Daten, das Feststellen der Größe einer Datenmenge und das Beobachten des dynamischen Verhaltens der Charakteristika einer Datenmenge ist in vertraulichen IT-Systemen nicht möglich [Fle97].

2.3.9. Zurechenbarkeit

Die Durchführung sicherheitsrelevanter Aktionen in IT-Systemen macht es erforderlich, eine Verknüpfung einer Identität mit den durchgeführten Aktionen herzustellen. Im Falle der Verletzung der Sicherheitspolitik sind Beweise erforderlich, die es erlauben, Verantwortlichkeiten zu schaffen. Hier wird deutlich, daß es für Agentensysteme nicht erforderlich bzw. nicht möglich ist, alle genannten Sicherheitskriterien zu erfüllen. Sich widersprechende Kriterien – wie beispielsweise Anonymität und Zurechenbarkeit lassen deutlich werden, daß es für Anwendungen erforderlich ist, zu spezifizieren, welche Sicherheitskriterien erforderlich sind. Entsprechend ist eine Sicherheitspolitik einzuführen und eine Sicherheitsarchitektur aufzustellen, die die Sicherheitspolitik durchsetzt.

2.4. Sicherheitsanforderungen an Agentensysteme

Die Sicherheitsanforderungen an Agentensysteme verlangen, daß die in einer Sicherheitspolitik aufgestellen Sicherheitskriterien durch das Sicherheitskonzept des Agentensystemes erfüllt werden. Bereits 1975 haben SALTZER und SCHROEDER eine Reihe von Richtlinien ausgearbeitet [SS75], die in Bezug auf das Sicherheitskonzept von Multics Anwendung fanden. Darüberhinausgehend besitzen die aufgestellten Prinzipien eine Allgemeingültigkeit, so daß sie in dieser Arbeit als Entwurfsprinzipien für die Systemkonzepte von Agentensystemen herangezogen werden:

- ▷ Der Entwurf eines Systems, insbesondere das Sicherheitskonzept, sollte öffentlich zugänglich sein. *Security by Obscurity* hat ausschließlich zur Folge, daß die rechtmäßigen Nutzer eines Systems in der Bewertung der Systemsicherheit im Unklaren gelassen werden.

- ▷ Zugriffe auf Objekte sind in der Standardeinstellung zu verweigern. Dieses schränkt die Offenheit eines Agentensystemes erheblich ein, erhöht die Sicherheit jedoch im erheblichem Maße.

- ▷ Die Menge der autorisierten Personen und ihre Zugriffsrechte sollten in regelmäßigen Abständen dahingehend überprüft werden, ob sie mit den derzeitigen Anforderungen im Einklang stehen.

- ▷ Die Teilnehmer in einem Agentensystem sollten nur mit den notwendigen Rechten ausgestattet werden. Das entspricht dem *least privilege principle*. Der Schaden, der von fehlerhafter oder maliziöser Software ausgehen kann, minimiert sich, wenn dem Nutzer nur die minimal notwendigen Rechten zugeteilt werden. Jedes überflüssige Recht ermöglicht Software, wie z.B. trojanischen Pferden, die Rechte auszunutzen, um maliziöse Aktionen durchzuführen. Einsatz findet das Prinzip auch in der Softwareentwicklung, wenn die Spracheigenschaft der Kapselung genutzt wird, um den Zugriff auf Datenstrukturen auf eine definierte Schnittstelle zu beschränken.

- ▷ Die Schutzmechanismen eines Agentensystemes müssen in alle Schichten einfließen, wobei die Grundsätze der Einheitlichkeit (Orthogonalität) und Einfachheit einzuhalten sind.

- ▷ Die psychologische Akzeptanz eines Sicherheitsmodells durch den Benutzer ist zu gewährleisten. Nur wenn den Nutzern eines Agentensystems bewußt ist, welche Werte in diesem System enthalten sind, sind die damit verbundenen Sicherheitsmaßnahmen durchzusetzen.

Der Schutz eines Rechensystems sollte weiterhin nicht von einem einzigen Sicherheitsmechanismus abhängen. Ein Durchbruch durch einen einzigen Sicherheitsmechanismus hat zur Folge, daß das gesamte System für den Angreifer offensteht. Wünschenswert ist ein gestaffelte Sicherheitsarchitektur (engl. *defense in depth*), in der mehrere Sicherheitsmechanismen durchbrochen werden müßten, um Zugriff auf das System zu erhalten (Bastionskonzept). Nicht erwünscht sind Abhängigkeiten zwischen den Sicherheitsmechanismen, da dann die Komplexität des Sicherheitsmodell steigt und es wahrscheinlicher wird, daß das Modell fehlerhaft implementiert wird.

2.5. Bedrohungsanalyse

Die Bedrohungsanalyse könnte darauf abzielen, einzig die drei Grundbedrohungen als Gefahrenpotential für Agentensysteme darzustellen. Bereits in Abschnitt 2.3 wurde darauf hingewiesen, daß ein derartiges Vorgehen nicht ausreichend ist. Entsprechend den dort aufgestellten Kriterien existieren weitaus mehr Bedrohungen. Eine Darstellung der potentiellen Risiken erfolgt in Kapitel 4.

Die in dieser Arbeit durchgeführte Bedrohungsanalyse folgt keinem formalen Vorgehen im Sinne des Abarbeitens eines Katalogs von Vorgehensweisen. Stattdessen erfolgt einleitend die Aufstellung von exemplarischen Angriffsszenarien. Dem schließt sich eine Darstellung der Werte innerhalb von Agentensystemen an. Den Werten stehen Bedrohungen gegenüber, die teilweise bereits den Szenarien zu entnehmen sind. Darüberhinaus existieren weitere Angriffstechniken, die im Kapitel 4 erläutert werden.

2.6. Risikoanalyse

Die Arbeit vollzieht eine qualitative Risikoanalyse. Mittels einer Risikoanalyse wird festgestellt, welche Verluste aus den in der Bedrohungsanalyse erkannten Bedrohungen entstehen können. Eine Aufstellung entsprechender Verluste für Agentensysteme im Detail ist aufgrund fehlender Zahlen nicht möglich.

Als weiterer Aspekt der Risikoanalyse erfolgt in den Kapiteln 5 und 6 eine detaillierte Untersuchung der Sicherheit der Basiskomponenten und der erweiterten Sicherheitsarchitekturen. Im Rahmen der Sicherheitsbeurteilung der Basiskomponenten erfolgt gleichzeitig die Untersuchung der Sicherheitsarchitekturen der in Kapitel 3 vorgestellten Agentensysteme, da deren Sicherheitsarchitektur im wesentlichen auf die der Basiskomponenten aufbaut. In den jeweiligen Abschnitten wird auf die Erweiterungen eingegangen, die in den Agentensystemen bezüglich der Sicherheitsarchitektur vorgenommen wurden.

Ausgehend von den in den Kapitel 4–6 erzielten Ergebnissen wird in Kapitel 7 für unterschiedliche Anwendungsgebiete dargelegt, welche Sicherheitskriterien wesentlich sind. Darauf aufbauend werden integrierte Sicherheitsarchitekturen angedeutet, die den anwendungsspezifischen Sicherheitsanforderungen gerecht werden sollen.

2.7. Klassifikation der Sicherheitsziele

FARMER et al. haben eine Klassifikation von Sicherheitszielen erstellt, in der sie darstellen, welche Ziele einfach, schwierig bzw. überhaupt nicht erreicht werden können [FGS96]. Berichte zum Thema Sicherheit in Bezug auf mobile Agenten beziehen sich häufig auf diesen Forschungsbericht, so daß die dort erzielten Ergebnisse auch als Grundlage für diese Arbeit verwendet werden.

Folgende Sicherheitsziele bezeichnen FARMER et al. als nicht erreichbar:

- ▷ Ist ein Interpreter unverfälscht?
- ▷ Wird ein Interpreter einen Agenten korrekt ausführen?
- ▷ Wird eine Umgebung einen Agenten vollständig ausführen?
- ▷ Wird eine Umgebung einen Agenten wie angefordert übertragen?
- ▷ Können Code und Daten eines Agenten privat gehalten werden?
- ▷ Kann ein Agent einen Schlüssel mit sich führen?
- ▷ Kann die Kommunikation zwischen Agenten privat erfolgen?
- ▷ Kann ein Agent von einem Klon unterschieden werden?

Darüberhinaus ermittelten sie Sicherheitsziele, die nur mit großem Aufwand erreicht werden können, da für ihre Erreichung nicht auf Sicherheitslösungen zurückgreifen werden kann, die bereits für verteilte Systeme entwickelt worden sind. Zur Erreichung der nachfolgenden Sicherheitsziele ist es erforderlich, daß neue Sicherheitsmechanismen entwickelt werden, die den geforderten Zielen gerecht werden:

- ▷ Kann eine Sprache eingesetzt werden, in der alle Agenten sicher sind?
- ▷ Kann der Sender eines Agenten dessen Flexibilität einschränken?
- ▷ Kann ein Interpreter feststellen, ob ein Agent sich in einem sicheren Zustand befindet?
- ▷ Kann der Sender eines Agenten festlegen, welche Interpreter daß Recht haben, seinen Agenten auszuführen?

In den Kapiteln 5 und 6 werden die mit Agenten in Verbindung stehenden Sicherheitsarchitekturen untersucht. Es wird sich zeigen, in welchem Maße die dort untersuchten Sicherheitsarchitekturen, den von FARMER et al. erzielten Ergebnisse entsprechen.

Schließlich wurden aber auch Sicherheitsziele ermittelt, die einfach zu erreichen sein sollten, da sie auf bereits eingeführte Sicherheitsmechanismen zurückgreifen könnten. Zu eben diesen Zielen zählen FARMER et al.:

- ▷ Kann der Autor und der Sender eines Agenten authentifiziert werden?
- ▷ Kann die Integrität des Codes eines Agenten verifiziert werden?
- ▷ Kann ein Interpreter die Privatheit eines Agenten während der Übertragung sicherstellen?
- ▷ Können Interpreter sich selbst gegen Agenten schützen [FGS96]?

Das Beispiel der Sprache Java, die für die Mehrheit der implementierten Agentensysteme eingesetzt wird, zeigt, daß der letzte Punkt, der Schutz der Umgebung vor Agenten, theoretisch erreichbar, praktisch bislang jedoch nicht hinreichend verwirklicht worden ist.

3. Internet-Agenten

„*A distributed system is one in which the failure of a computer you didn´t even know existed can render your own computer unusable.*“
– Leslie Lamport

3.1. Agenten

Globalisierung drückt sich in der Informationstechnologie in der zunehmenden Nutzung des Internets und der firmeneigenen Intranets aus. Anwendungen in diesem Umfeld müssen dem verteilten Charakter der Informationssysteme gerecht werden. Gleichzeitig müssen bestehende Anwendungen (engl. legacy applications) einbezogen werden. Ein Weg, diesem Problem gerecht zu werden, ist der Einsatz von Middleware-Plattformen – wie beispielsweise Common Object Request Broker Architecture (CORBA). Ein anderer Lösungsweg soll mit Agenten beschritten werden.

Vor allem in Bereichen der elektronischen Dienstemärkte werden Dienstleistungen angeboten, in denen Agententechnologien bereits zum Einsatz kommen. Katalogbasierte Einkaufsmöglichkeiten im Internet lassen nur wenig Raum für dynamische Interaktion. Es kommen im wesentlichen einseitige Pull- oder Push-Technologien zum Einsatz. Die Folge ist, daß der potentielle Kunde viel Zeit damit verbringen muß, Anbieter für ein gewünschtes Produkt zu finden und deren Angebote zu vergleichen. Agenten sollen wesentlich bessere Interaktionsmöglichkeiten bieten können. Darüberhinaus könnten sich mit ihnen die geschilderten Prozesse automatisieren lassen, so daß der Anwender mit nur wenig Zeitaufwand sein gewünschtes Ergebnis erhält. Zukünftig könnten Agenten in der Lage sein, Verträge auszuhandeln und Geschäfte abschließend zu tätigen. Agenten könnten den elektronischen Dienstemärkten zu jenem Durchbruch verhelfen, der bereits seit einigen Jahren postuliert wird.

Daneben werden eine Reihe weiterer Vorteile mit dem Einsatz von Agenten in verteilten Umgebungen verbunden:

- ▷ Reduzierung der Netzwerkbelastung,
- ▷ Reduzierung der Ressourcenbelastung auf Seiten des Clients,

- ▷ asynchrone Arbeitsweise und
- ▷ dezentrale Struktur.

Den Vorteilen stehen eine Reihe von Nachteilen gegenüber, von denen der Sicherheitsaspekt am schwersten wiegt:

- ▷ Bislang existieren nur kleine Agentensysteme,
- ▷ hohe Anforderungen an die Infrastruktur und
- ▷ Probleme der Sicherheit.

Im Anschluß wird die Schwierigkeit der Definition des Agentenbegriffs verdeutlicht. Agenten werden eine Vielzahl von Eigenschaften zugeordnet, eine Ursache für die Probleme der Begriffsbildung. Im Abschnitt 3.2 werden die wichtigsten Eigenschaften der Agenten aufgeführt. Im Anschluß erfolgt die Zerlegung der Agentenmodelle in ihre Einzelkomponenten. Abschließend werden die bekanntesten Agentensysteme vorgestellt.

3.1.1. Definitionen

Es existieren eine Vielzahl von Definitionen des Agentenbegriffs, die jeweils eine oder mehrere Eigenschaften in den Vordergrund stellen. Hierzu zählen u. a. *Robots*, *Intelligent Agents*, *Interface Agents*, *Knowbots*, *Mobile Agents*, *Personal Agents* und *Softbots*. Begriffserklärungen, die versuchen eine umfassendere Erklärung zu liefern, sind derart allgemein, daß es nur sehr schwer fällt, Anwendungen anhand dieser Definitionen zu kategorisieren. Als Beispiel sei hierzu eine Definition von DAVID EICHMANN angeführt:

> **Definition 2:**
>
> Ein Agent ist ein Programm, das mit dem Benutzer interagiert und ihm assistiert [Eic94].

Es stellt sich die Frage, ob es grundsätzlich möglich ist, eine allgemeine und gleichzeitig präzise Umschreibung für den Begriff des Agenten aufzustellen. In Hinblick auf die unterschiedlichen Definitionen, die sich im Laufe der Zeit innerhalb der verschiedenen Forschungsbereiche etabliert haben, erscheint dieses eher unwahrscheinlich.

Für die vorliegende Arbeit sind mobile Agenten von entscheidender Bedeutung, da deren Fähigkeiten besonders in Netzwerken zum Tragen kommen. Der Schwerpunkt liegt somit bei mobilen Agentensystemen, wobei auch Systeme wie Channels, Controls, Plugins und Servlets in die Betrachtung einbezogen werden. Zusammengefaßt werden die untersuchten Agentensysteme unter dem Begriff des Internet-Agenten. Hauptanliegen ist die gemeinsame Betrachtung unterschiedlicher Softwaresysteme, die im Internet zum Einsatz kommen und einen mobilen Charakter aufweisen. Bereits einleitend wurde darauf

verwiesen, daß eine Reihe anderer Begriffe existieren (Active Content, Mobiler Code), die ebenfalls zur Umschreibung der in dieser Arbeit untersuchten Technologien hätten herangezogen werden können.

Insgesamt weist das Untersuchungsgebiet das Problem auf, daß die betrachteten Technologien bislang nur unzureichend verstanden sind. Weitere Forschungen sind erforderlich, um eine abschließende Kategorisierung der Technologien vornehmen zu können.

Internet-Agenten

Im weiteren Verlauf der Arbeit wird der Begriff des Agenten unter der folgenden Definition verwendet:

> **Definition 3:**
>
> Internet-Agenten (allg. Netz-Agenten) sind kommunikative Programme, die sich Netztechniken zu eigen machen, um in dem Medium des Netzes Aufgaben erledigen zu können. Sie besitzen Eigenschaften wie Autonomität, Intelligenz und Mobilität, wobei die Eigenschaften in verschiedenen Systemen unterschiedlich ausgeprägt sind.

Die Definition soll hervorheben, welche Bedeutung Netzwerke im Zusammenhang mit Agenten gewonnen haben. Ausgehend von den Entwicklungen im Bereich der Künstliche Intelligenz (KI) haben sich die Arbeiten im Agentenumfeld immer stärker zu den Netzwerktechniken hin orientiert. Bekannte Projekte, die sich mit Agenten beschäftigen, wie *Ahoy*[1], *BargainFinder*[2], *Firefly*[3], *MetaCrawler*[4] und *Mole*[5]. stehen alle im Zusammenhang mit dem Internet bzw. wurden in Hinblick darauf ergänzt. Neue Agenten-Modelle wie *Applets*, *Castanet*, *Controls*, *Plug-ins* und *Servlets* sind im Zuge der Entwicklung des Internets in den letzten Jahren entstanden. Aufgabenfelder, die heute von Agenten abgedeckt werden sollen, weisen ebenfalls in die Richtung von Netzwerken.

3.2. Eigenschaften

Die Untersuchung von Agentensystemen wird durch die Begriffsproblematik erschwert. Von Vorteil ist die Gliederung der Agentensysteme anhand der in ihnen verwirklichten Eigenschaften. Es existieren bereits eine Vielzahl von Eigenschaften, die in unterschiedlicher Kombination und Gewichtung in verschiedenen Modellen/Projekten zur Definition von Agenten herangezogen werden. Je nachdem, welcher Forschungszweig für die

[1] `http://ahoy.cs.washington.edu:6060`
[2] `http://bf.cstar.ac.com/bf/`
[3] `http://www.firefly.com`
[4] `http://www.metacrawler.com`
[5] `http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole`

Definition verantwortlich ist, werden unterschiedliche Schwerpunkte hinsichtlich der notwendigen Eigenschaften/Fähigkeiten von Agenten gesetzt. Definitionen aus dem Gebiet der KI, wie glaubwürdige oder emotionale Agenten [Bat94], werden mit anthropogenen Attributen versehen und sind durch die den Agenten zugebilligte Intelligenz geprägt. Die Robotik, ein spezielles Forschungsgebiet der KI, stellt die Autonomität in den Vordergrund, während Architekturen im Zusammenhang mit Netzwerken die Mobilität als wichtigste Eigenschaft anführen.

Die nachfolgende Aufstellung beinhaltet die vom Autor als wesentlich angesehenen Eigenschaften von Agentensystemen. Eine Vielzahl weiterer Eigenschaften wird in der Literatur mit Agenten in Zusammenhang gebracht. Exemplarisch sei hier auf [CGH+95] und [Her96] verwiesen.

3.2.1. Autonomität

Kennzeichnend für Agenten ist die Erledigung von Aufgaben ohne unmittelbare Eingriffe seitens des Anwenders. Der Benutzer spezifiziert eine Aufgabe und der Agent entscheidet, wann und wie sie erledigt wird. Weiterentwickelte Systeme reagieren selbständig auf das Eintreten bestimmter Ereignisse. Über diese Eigenschaft der *Autonomie* verfügen alle Agenten, wenn auch in unterschiedlicher Ausprägung. Ihre Autonomie läßt Agenten dynamisch auf ihre Umwelt reagieren. Zum einen lassen Agenten die Dynamik von komplexen Medien wie beispielsweise dem Internet für den Anwender beherrschbar werden. Auf der anderen Seite erhöhen Agenten selbst die Dynamik von Netzwerken, gewährleisten aber gleichzeitig, daß die Komplexität für den Nutzer nur in geringem Maße ansteigt.

3.2.2. Intelligenz

Intelligenz ist der Grad des Schlußfolgerns und des gelernten Verhaltens: die Fähigkeit des Agenten, Aussagen über die Ziele des Benutzers entgegenzunehmen und die ihm anvertrauten Aufgaben auszuführen. Minimum ist eine Art der Darstellung von Präferenzen, beispielsweise in der Form von Regeln, mit einer Interferenzmaschine oder einem anderen Mechanismus zur Ziehung von Schlußfolgerungen basierend auf den Präferenzen. Ein höherer Grad der Intelligenz schließt ein Benutzermodell oder eine andere Form des Verstehens und Schlußfolgerns über jenes, was der Benutzer erledigt haben möchte, ein. Hinzu kommt die Planung der Mittel, um dieses Ziel zu erreichen. Am Ende der Intelligenz-Skala stehen Systeme, die in der Lage sind, zu lernen und sich ihrer Umgebung anzupassen. Beides geschieht in Bezug auf die Vorstellungen des Anwenders und den für die Agenten verfügbaren Ressourcen [GAA+95].

3.2.3. Kommunikativität

Kommunikation ist wohl die zwingendste Eigenschaft von Agenten. Kommunikation erfolgt in Agentensystemen in vielfältigster Form. Zunächst gibt es eine Vielzahl unterschiedlicher Subjekte, mit denen kommuniziert werden muß. Dazu gehören Agenten, Anwender, Umgebungen und Gruppen von Subjekten. Kommunikation kann lokal erfolgen, mit der Folge, daß ein Agent zunächst zu seinem Kommunikationspartner migrieren muß. Kommunikation kann aber auch global innerhalb von Netzwerken erfolgen. Der Nachrichtenversand kann asynchron oder synchron erfolgen. Desweiteren existieren eine Reihe von Standardprotokollen, die zur Kommunikation genutzt werden können. Daneben ist der Einsatz von speziellen, für bestimmte Aufgaben adaptierter Protokolle in Agentensystemen von großem Vorteil.

3.2.4. Mobilität

Mobile Agenten bewegen sich autonom im Netz, d. h., sie verfolgen einen nicht fest definierten Plan, sondern sie können ad hoc Entscheidungen treffen und ihre Pläne dahingehend modifizieren. Es gibt keine Anwendungsfelder, für deren Erfüllung Agenten unabdingbar wären. Agenten stellen lediglich eine Technik dar, mit dem viele Probleme in verteilten Umgebungen auf elegante Weise gelöst werden können.

Der Begriff der Mobilität umfaßt eine Vielzahl von Techniken. Eine Differenzierung des Begriffes Mobilität ist deshalb erforderlich. Oft anzutreffen ist die Unterscheidung in starke und schwache Mobilität [CGPV97]. In Systemen, die *starke Mobilität* (engl. *strong mobility*) unterstützen, sind Agenten in der Lage, mit ihrer transitiven Hülle, bestehend aus Code, Daten und Referenzen, auf verschiedene Wirtssysteme zu migrieren. Voyager ist ein Beispiel für ein solches Agentensystem. *Schwache Mobilität* (engl. *weak mobility*) erlaubt Agenten hingegen lediglich dynamisch zur Laufzeit Code aus beliebigen Quellen an sich zu binden. Diese Form der Mobilität wird von Applets unterstützt.

Die Vorteile von Agenten beruhen auf ihrer Fähigkeit der Migration. Migration bezeichnet die Fähigkeit, sich von einem Ort zu einem anderen bewegen zu können. Es ist zu unterscheiden, ob der Agent oder die Umgebung, in der der Agent ausgeführt wird, über die Migration entscheidet. Der Zeitpunkt der Migration und die Menge der Informationen, die vom Agenten mitführt werden können, sind in vielen Systemen unterschiedlich gelöst. Als weitere Frage schließt sich die Anzahl der Migrationssprünge (Hops) an. In Systemen mit Applets oder ActiveX-Agenten erfolgt die Migration einmalig, Systeme wie Aglets oder Voyager schränken die Anzahl der Hops nicht ein.

3.3. Architektur

Für die Implementation und den Einsatz von Agenten wird Unterstützung auf unterschiedlichen Ebenen benötigt – auf der Betriebssystem-, Interaktions-, Protokoll- und der Sprachebene. Desweiteren gibt es die umfassende Komponente der Umgebung die

das jeweilige Agentensystem kapselt. Die in Frage kommenden Anwendungsgebiete der Agenten stellen unterschiedliche Ansprüche an die Architektur. Suchmaschinen benötigen den schnellen Zugriff auf verteilte Informationssysteme; für Anwendungen aus dem Bereich der elektronischen Dienstemärkte muß die Transaktionssicherheit gewährleistet sein, Team-Arbeit benötigt Unterstützung bei der Kommunikation auf allen Ebenen, während beim Mobile Computing die asynchrone Kommunikation im Vordergrund steht.

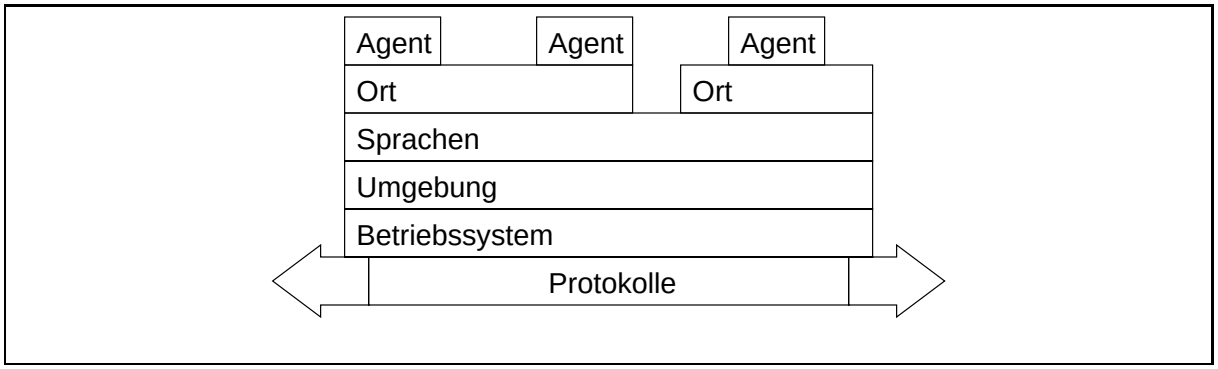

Abbildung 3.1.: Die Architektur eines Agentensystemes läßt sich durch ein Schichtenmodell darstellen. Zu den bekannten Schichten Betriebssysteme und Protokolle kommen die Schichten der Umgebungen, Sprachen, Orte und der Agenten hinzu.

3.3.1. Betriebssysteme

Eine wesentliche Anforderung an Agenten-Systeme ist die Unabhängigkeit von Software- und Hardwareplattformen. Dementsprechend sollte die Wahl des Betriebsystems keinen wesentlichen Einfluß auf die Architektur von Agentensystemen haben. Insbesondere da es unterhalb der kapselnden Umgebung liegt, wie Abbildung 3.1 zeigt. OUSTERHOUT stellte bereits auf der USENIX '95 Konferenz fest, daß Skriptsprachen viele Elemente enthalten, die heute von Betriebssystemen verwaltet werden: Ausführungsumgebung, Benutzerschnittstelle, Kommunikation und Sicherheit [Ous95]. Gleiches stellen MILOJIČIĆ et al. in [MGW97] fest. Im Forschungsprojekt „Betriebssysteme" der OPEN GROUP wird dargelegt, wie Technologien heutiger Betriebssysteme auf Anwendungsebene zum Einsatz kommen können. Als Beispiel führen MILOJIČIĆ et al. die Fehlertoleranz an.

Trotz der zunehmenden Unabhängigkeit der Applikationen von der Betriebssystemebene werden deren Grundfunktionalitäten benötigt. Bereits in Betriebssystemen enthaltene Mechanismen müssen nicht mehr durch die Umgebung implementiert, sondern nur noch für die unterschiedlichen Agentensysteme abstrahiert werden.[6]

[6] Siehe hierzu beispielsweise [JRS95] und [CMRB96].

3.3.2. Interaktionskomponente

Ein wesentlicher Unterschied zwischen Agenten und Standard-Software besteht darin, daß Agenten persönliche Software darstellen, die den Vorlieben der Anwender Rechnung tragen und Entscheidungen anhand des Wissens über den Benutzer treffen können. Dementsprechend spielt die Interaktion zwischen Agent und Anwender eine tragende Rolle. Die Interaktion zwischen Anwender und Agent kann über den Versand von Nachrichten oder durch Meldungen in Fenstersystemen erfolgen.[7] Im wesentlichen ist heute allerdings der Browser die Interaktionskomponente zwischen Benutzer und Agent. Der Browser stellt ein Werkzeug dar, daß eine strukturierte und uniforme Darstellung der in Netzwerken zugänglichen Informationen erlaubt. Der Browser ist ein vertrautes Werkzeug, so daß sein Einsatz als Interaktionskomponente in Agentensystemen mit nur wenig Aufwand verbunden ist.

3.3.3. Protokolle

Die Entwicklung von Agentensystemen erfordert die Unterstützung einer Reihe von Standard-Protokollen. Dies ist insbesondere in Bezug auf die in dieser Arbeit untersuchten Internet-Agenten der Fall. Sowohl für die Kommunikation, Migration aber auch für die Abwicklung elektronischer Zahlungsströme ist die Unterstützung zahlreicher Protokolle notwendig. Ein wesentlicher Vorteil der Agentensysteme ist die Möglichkeit, über Agenten spezielle Protokolle anzuwenden, die von herkömmlicher Softwareprodukten nicht unterstützt werden.

3.3.4. Sprachen

Die für die Implementation eines Agenten verwendete Programmiersprache spielt eine tragende Rolle in der Architektur von Agentensystemen. Agenten sind hochgradig verteilte Anwendungen, ein Aspekt, der sich in der Programmiersprache niederschlagen sollte.

Anhand des nachfolgenden Szenarios lassen sich eine Reihe von Aspekten ableiten, die für die Auswahl der Implementationssprache entscheidend sind.

[7] Siehe hierzu auch [Mar97d].

Szenario 1:

ShopAgent ist ein Agent, der es dem Benutzer ermöglicht, anhand der Angabe eines Produkts und einer Menge gewünschter Eigenschaften, den günstigsten Anbieter im Netz zu finden. In Abstimmung mit dem Anwender ist der Agent in der Lage, das entsprechende Produkt bei dem gefundenen Anbieter zu kaufen. Damit der Anwender nicht die ganze Zeit mit dem Netzwerk verbunden sein muß, migriert der Agent zu den jeweiligen Systemen der Anbieter. Nach Abschluß der Aufgabe kehrt er auf das System des Anwenders zurück.

Die Sprache muß Kommunikation und Migration unterstützen, unabhängig von den verwendeten Plattformen sein, die Arbeit im Netzwerk unterstützen, und ihren Beitrag dazu leisten, daß die Möglichkeit maliziösen Verhaltens seitens des Agenten bzw. der Umgebung eingeschränkt wird. Hierzu gehört auch, daß Agent und Umgebung unerwartet auftretende Fehler behandeln können. Weitere wichtige Aspekte sind Mächtigkeit, Erweiterbarkeit, Einbettbarkeit und Performanz.

Anforderungen an Programmiersprachen [BZW98]:

Kommunikationsfähigkeit Agenten als hochkommunikative Systeme erfordern Implementationssprachen, die Konstrukte zur Verfügung stellen, mit denen Kommunikation auf allen Ebenen unterstützt wird.

Objekt-Orientierung Agenten sind Objekte. Sie bestehen aus Methoden und innerem Zustand. Wünschenswert ist deshalb eine Sprache, die das objekt-orientierte Programmiermodell unterstützt.

Plattformunabhängigkeit Agenten beziehen ihre wesentlichen Vorteile aus dem Umstand, daß sie in offenen, heterogenen Umgebungen agieren. Sie operieren auf den unterschiedlichsten Software- und Plattformplattformen. Das Ziel läßt sich mit Hilfe von Sprachen erreichen, die interpretiert oder in einen Zwischencode übersetzt werden, der von einer virtuellen Maschine ausgeführt wird.

Robustheit Die Implementationssprache des Agenten muß Mittel zur Verfügung stellen, mit denen Ausnahmesituationen (z. B. unerwartete Fehler, Dateiende, etc.) begegnet werden kann. Ohne entsprechende Mechanismen sind Agenten nicht sinnvoll einsetzbar, da das Auftreten von Ausnahmesituationen dann jeweils zur Termination des Agenten führt. Die Robustheit einer Sprache wird u. a. durch Typsysteme und die Behandlung von Ausnahmen (Exceptions) gefördert.

In [Mar97d] wurden eine Reihe von Programmiersprachen in Hinblick auf die oben angeführten Eigenschaften untersucht. Das Ergebnis stellte Java, Tcl und Telescript als

geeigneteste Programmiersprachen für die Implementierung von Agentensystemen fest. Die Sprache Telescript wird von GenMagic mittlerweile nicht mehr weiterentwickelt. Gegenüber Tcl setzt Java sich in immer mehr Agentensystemen durch. Die Mehrzahl der Agentensysteme wird mittlerweile in Java entwickelt.

3.3.5. Umgebung

Ein Agent ist nicht vollständig autonom in dem Sinne, daß er alleine existieren kann. Agenten sind stets auf ein Wirtsprogramm – eine Umgebung – angewiesen. Umgebungen können in unterschiedlichster Form implementiert werden, sie reichen von Servern (Agent Tcl[8]) bzw. Engines (Telescript) über Browser bis hin zu sog. Active Documents (Applets, Active Mail) [LD96]. Die unterschiedlichen Modelle werden für die Arbeit unter dem Begriff *Umgebung* zusammengefaßt.

Definition 4:

Eine Umgebung bildet einen Raum in dem Agenten empfangen, dekodiert und authentifiziert werden. Anschließend werden sie an einen innerhalb der Umgebung liegenden Ort weitergeleitet, an dem die Agenten zur Ausführung gelangen, Dienste in Anspruch nehmen und von wo aus sie nach Abschluß der Aufgabe weiter migrieren können.

Für die Durchführung der in der obigen Definition angeführten Schritte sind eine Reihe von Randbedingungen zu erfüllen. Es beginnt mit der Bereitschaft des Systems, einen Agenten zu empfangen. Nach dem Empfang des Agenten wird dessen Identität verifiziert, anhand derer entschieden wird, ob er zur Ausführung gebracht werden darf. Vor Beginn der Ausführung muß schließlich noch festgestellt werden, ob die vom Agenten benötigten Dienste zur Verfügung stehen, d. h., ob sie in ausreichendem Maße vorhanden sind. Umgebungen sollen gewährleisten, daß Agenten kontrolliert ausgeführt werden, also weder die Umgebung noch andere Agenten beeinträchtigen können. In Verbund mit der weitgehenden Nutzung von Java als Programmiersprache für Agentensysteme werden im wesentlichen Browser und agentensystemeigene Server als Umgebungen von Agenten eingesetzt.

3.4. Ausgewählte Agentenmodelle

Im folgenden werden Agentenmodelle vorgestellt, die ein Gerüst beinhalten, mit denen Agenten generiert werden können. Die Mehrzahl der Modelle beruht auf der Programmiersprache Java, die gleichzeitig die Ausführungsumgebung der Agenten bildet. Der

[8] Wurde in D´Agents umbenannt.

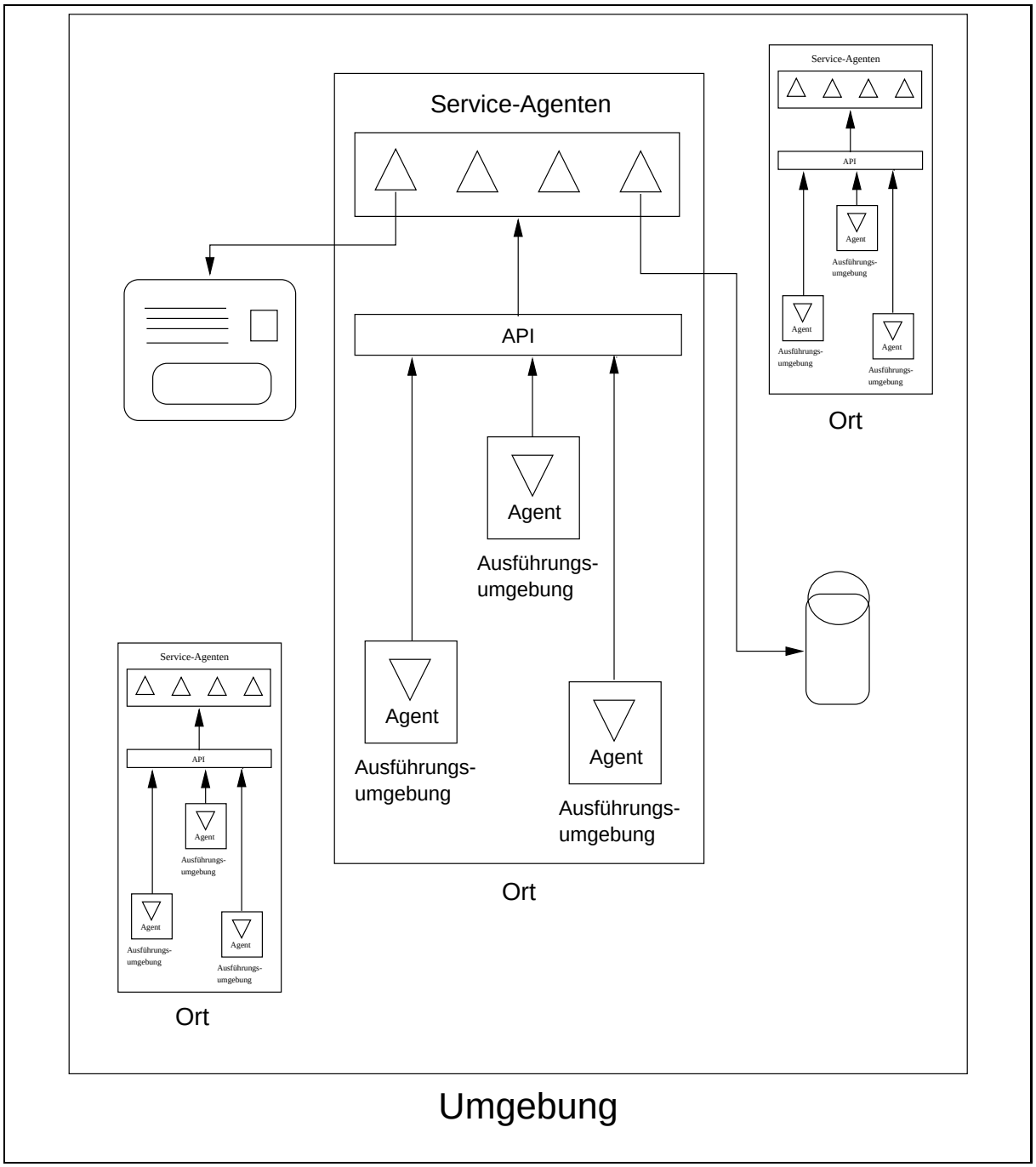

Abbildung 3.2.: Schematische Darstellung eines Umgebungskonzepts. Umgebungen können mehrere Orte anbieten, die unterschiedliche Dienste zur Verfügung stellen. Innerhalb der Orte erlauben Ausführungssysteme die Ausführung von Agenten, die in unterschiedlichen Sprachen implementiert wurden. Über eine API können die Dienste von Service-Agenten in Anspruch genommen werden, die einen kontrollierten Zugriff auf die System-Ressourcen ermöglichen.

Entwicklungsstand und damit auch die verfügbare Dokumentation der einzelnen Agentensysteme ist sehr unterschiedlich, so daß die inhaltliche Tiefe der folgenden Darstellungen ebenfalls differiert.

3.4.1. Active Networks

Agenten sind Objekte der Anwendungsebene. Mit dem von TENNENHOUSE und WETHERALL entwickelten *Active Network* können Agententechnologien auch auf niederen Schichten zum Einsatz kommen. Als Active Network werden Netzwerke bezeichnet, innerhalb derer die Möglichkeit besteht, Programme in Netzwerkknoten zu injizieren, so daß die Verarbeitung von Nachrichtenpaketen innerhalb der Netzknoten benutzer- und anwendungsspezifisch erfolgen kann. Desweiteren können Netzknoten in aktiven Netzwerken Berechnungen auf den Nutzdaten der Nachrichtenpakete durchführen. Dazu sind Fragmente mobilen Codes in den Paketen enthalten. Die Ausführung des Codes erfolgt in jedem Netzknoten, den ein Paket auf seinem Weg von der Quelle bis zum Ziel durchquert. Mit dem Paradigma des aktiven Netzwerks vollzieht sich der Wechsel vom statischen Übertragungsmedium zum programmierbaren Netzwerk.

Vorteile aktiver Netzwerke sehen TENNENHOUSE und WETHERALL in der Möglichkeit, adaptive Protokolle realisieren zu können, die umfangreichere Interaktion ermöglichen. Desweiteren können Berechnungen in die Netzwerke verlagert werden. Fernsehen und Radio über das Internet, Internet-Telefonie sowie Videokonferenzen sollen davon in erheblichem Maße profitieren können. Begründet wird dieses mit der verbesserten Unterstützung von Broad- und Multicasting. Als weiterer Vorteil des Agentenmodells wird angeführt, daß neue Mechanismen und Technologien schneller genutzt werden können, da es nicht mehr eines Herstellers bedarf, der sich bereiterklärt, in einer nachfolgenden Version seines Produkts eine Neuentwicklung zu unterstützen.

Aktive Netzwerke gibt es in zwei Ausprägungen, wobei sich beide Typen nicht ausschließen. Im einfachen Fall, enthalten die Pakete keine Programmfragemente. Stattdessen enthält der Header eines jeden Pakets Informationen darüber, welche Programme im Netzknoten auf den Paketen anzuwenden sind.

Im zweiten Verfahren ist es gestattet, aktiv Agenten (Programme) in Netzwerkknoten zu injizieren. Dazu werden sog. *Kapseln* eingesetzt, die Paketen entsprechen, mit dem Unterschied, daß Kapseln Nachrichten sind, die stets Programmfragmente enthalten [TW96]. Programmfragmente bestehen wenigstens aus einer Instruktion. Erreicht eine Kapseln einen Netzwerkknoten, wird das in ihr enthaltene Programmfragment ausgeführt. Das Ergebnis der Ausführung einer Kapsel sind null oder mehr Kapseln, die vom aktuellen Knoten weiterzuleiten sind. Kapseln können persistente Informationen auf den Netzknoten für nachfolgende Kapseln hinterlassen. Durch vorzeitige Injizierung von Kapseln in Netzwerkknoten können nachfolgende Kapseln Programmfragmente enthalten, die nur noch aus wenigen Instruktionen bestehen. Desweiteren ist ein dynamisches „load-on-demand"-Schema denkbar. Im wesentlichen bestehen die Programmfragmente der Kapseln aus primitiven Instruktionen. Sind komplexere Berechnungen erforderlich,

können aus den Programmfragmenten externe Programme aufgerufen werden.

Mittlerweile existieren mehrere Forschungsprojekte an unterschiedlichen Universitäten, die sich mit dem Einsatz von Active Networks beschäftigen. So wird im SwitchWare-Projekt[9] der Universität von Pennsylvania die Sprache Caml – ein ML-Dialekt – für die Entwicklung von Agenten in Active Networks eingesetzt [Ale97]. Die Forschungsgruppe um TENNENHOUSE und WETHERALL arbeitet im Projekt *Activeware* am MIT daran, Active Network-Technologien in die Internet-Protokolle zu integrieren.[10]

3.4.2. ActiveX-Controls

ActiveX-Controls (-Agenten) sind Kernkomponenten der von Microsoft entwickelten ActiveX-Technologie. ActiveX ist nach Definition von Microsoft eine Sammlung von Technologien, die interaktive Inhalte im World-Wide Web (WWW) ermöglichen. Das bedeutet, daß es möglich werden soll, Programme auf fremden Rechner auszuführen.

ActiveX stellt im wesentlichen einen Baukasten bestehender Microsoft Technologien dar, die angepaßt wurden, um internetfähig zu werden. Komponenten des Baukastens sind Object Linking and Embedding (OLE), Component Object Model (COM) und Object Component Extension (OCX). Seit der Veröffentlichung im Jahre 1996 ist ActiveX ein bewegliches Ziel geblieben. Darüberhinaus gibt es nur wenige öffentlich zugängliche Dokumente, die die Interna von ActiveX näher darlegen.[11]

ActiveX wurden von Microsoft zunächst als Konkurrenz zu Java positioniert. Im wesentlichen handelt es sich bei ActiveX-Agenten jedoch um Plug-ins, die lokal installiert und bei Bedarf automatisch aktiviert werden. Derzeit dienen ActiveX-Agenten hauptsächlich der Erweiterung der Funktionalität des Internet Explorers. Im Unterschied zu Plug-ins werden ActiveX-Agenten automatisch installiert, wohingegen die ersteren i. d. R. manuell installiert werden.

ActiveX-Agenten sind direkt ausführbare Programme, die in Intel Binärcode kompiliert werden. Im Gegensatz zu Java-Programmen, die in einen plattformunabhängigen Bytecode übersetzt werden, sind ActiveX-Agenten damit an die Windows-Plattform gebunden. Als Code in Binärform können sie können auf Systemfunktionen des Betriebssystems zugreifen. Gleichzeitig sind Active-Agenten – wiederum im Gegensatz zu Java-Programmen – unabhängig von der eingesetzten Programmiersprache. Die Sprache muß lediglich der Schnittstellenanforderung für ActiveX-Agenten genügen. ActiveX-Agenten können somit auch mit Java entwickelt werden, erben damit aber nicht die Plattformunabhängigkeit der Java-Programme. Ein *Wrapper* kapselt den Java-Code und stellt so die Notwendigkeit einer Windows-Plattform wieder her. In Java implementierte ActiveX-Agenten werden ähnlich wie Applets behandelt. Derartige Agenten werden nicht wie ihre nativen Pendants lokal installiert.

[9] http://www.cis.upenn.edu/~switchware/

[10] http://www.sds.lcs.mit.edu/activeware/

[11] Auch die Abgabe der ActiveX-Technologien an eine Arbeitsgruppe unter der Obhut der OPEN GROUP hat an diesem Umstand nichts wesentliches verändert.

Die folgenden Technologien können als ActiveX-Agenten eingesetzt und mit Authenticode signiert werden:

- ▷ COM-Controls,
- ▷ ausführbare Windowsprogramme,
- ▷ Installationsprogramme mit dem Typ `INF` und
- ▷ Archivdateien vom Typ `CAB` [Hop97a].

ActiveX-Agenten lassen sich – wie Applets – in Hypertext Markup Language (HTML)-Dokumente einbinden. Stößt ein Browser in einer HTML-Seite auf eine Referenz, die auf ein ActiveX-Agenten verweist, wird abhängig von der angegebenen Klassen-ID (CLSID) der entsprechende Agent geladen. Ist der Agent noch nicht auf dem Rechner vorhanden, wird er anhand der im HTML-Dokument angegebenen Uniform Resource Locator (URL) geladen und installiert. Nachfolgende Referenzierungen des gleichen Agenten instantiieren ein Objekt des vorher installierten Agenten. ActiveX-Agenten unterliegen keinerlei Beschränkungen hinsichtlich der vom System zur Verfügung gestellten Ressourcen. Der Browser setzt dem Agenten lediglich hinsichtlich des Lebenszykluses eine Grenze.

Wie im Falle von Applets, werden auch ActiveX-Agenten mit Beenden des Browsers aus dem Speicher entfernt. Sobald einem Agent der Zugang auf einen Rechner gewährt wurde, hat dieser den vollen Zugriff auf alle Ressourcen. Dem Anwender soll es jedoch mittels der von Microsoft als *Authenticode* entwickelten Technologie möglich sein, Einfluß auf die Ausführung von ActiveX-Agenten zu nehmen. Nähere Ausführungen zu Authenticode erfolgen im Abschnitt 5.2.1.

Derzeit werden ActiveX-Agenten lediglich durch den Internet Explorer unterstützt, eine Folge des Umstands, daß Agenten an die Windows-Plattform gebunden sind. Pläne, die entsprechenden Techniken auf andere Plattformen zu portieren, befinden sich noch in der Entwicklungsphase. Es ist nicht abzusehen, ob nach deren Verwirklichung ActiveX-Agenten plattformübergreifend einsetzbar sein werden.

Die Migration von ActiveX-Agenten erfolgt vom Server zum Client. Der Migrationszeitpunkt beschränkt sich auf einen Zeitpunkt vor der Ausführung des Agenten. Nach Beginn der Ausführung ist eine Migration nicht mehr möglich. Daraus folgt, daß die ActiveX-Technologien auch keine Methoden zur Verfügung stellen, die eine transparente Migration unterstützen.

Im Gegensatz zu Applets können ActiveX-Agenten wie Servlets auch auf Seiten des Servers zum Einsatz kommen. Notwendige Voraussetzung hierfür ist, daß der Server ActiveX-Agenten unterstützt.

Der Sammlung der ActiveX-Technologien stehen keine Werkzeuge zur Seite, mit denen die Ausführung der ActiveX-Agenten kontrolliert und mit denen Einfluß auf den Lebenszyklus eines Agenten genommen werden kann. Der Zugang zu den Agenten erfolgt ausschließlich über die Umgebung, in der sie ablaufen. Mögliche Umgebungen bilden der Internet Explorer oder Anwendungen aus dem Microsoft Office-Paket.

3.4.3. Aglets

Ziel der Entwicklung von Aglets ist die Bereitstellung einer Bibliothek, die die Sprache Java um das Paradigma des Agenten erweitert. Mit den Techniken *Remote Method Invocation (RMI)* und *Serialization* stehen mittlerweile im Java Kernpaket Klassen und Methoden zur Unterstützung von mobilen Objekten zur Verfügung. Es bedarf jedoch weiterer Abstraktionen, um komfortabel Agenten entwickeln zu können. Sowohl Aglets als auch Voyager verfolgen den Ansatz, Java um die fehlenden Mechanismen zu erweitern. Ohne die Erweiterung durch Bibliotheken ist Java lediglich ein schwach mobiles Agentensystem.

Das Aglets-System (Aglets Workbench) beinhaltet eine Entwicklungsumgebung sowie eine Umgebung zur Ausführung von Aglets-Agenten. Das Aglets Objektmodell besteht aus den Komponenten *Aglets*, *Context* und *Messages*. Weitere Komponenten sind das Agent Transfer Protocol (ATP) und die Aglet-API. Der Begriff *Aglet* soll Bezug auf die Zusammenfassung von Applets und Agenten zu neuen Java-Objekten nehmen[12] [LC96]. Context ist ein stationäres Objekt, daß die Umgebung für Agenten darstellt. Eine Message ist ein Objekt, daß zwischen Agenten ausgetauscht wird.

Agenten werden innerhalb eines Context erzeugt. Mit ihrer Erzeugung erhalten sie einen eindeutigen Bezeichner.[13] Als Rückgabewert nach der Erzeugung eines Agenten wird ein *Proxy-Objekt* geliefert. Die Kommunikation mit einem Agenten erfolgt vollständig über Proxy-Objekte. Methoden eines anderen Agenten können nur über die Schnittstelle des Proxy-Objekt aufgerufen werden. Damit soll sichergestellt werden, daß der Aufruf öffentlicher Methoden an einem Agenten kontrolliert erfolgen kann. Das Proxy-Objekt eines Agenten konsultiert hierzu den Security-Manager, um festzustellen, ob der Zugriff durch ein anderes Proxy-Objekt gestattet ist. Proxy-Objekte dienen weiterhin der Transparenz, wenn der zugehörige Agent an einen entfernten Ort migriert ist. Nachrichten, die an den entfernten Agenten gerichtet sind, werden vom Proxy-Objekt an den aktuellen Aufenthaltsort des Agenten weitergeleitet. So können Objekte ohne Wissen über den konkreten Aufenthaltsort eines Agenten auf diesen zugreifen.

Die Migration von Agenten erfolgt in Form serialisierter Java-Objekte. Ziel sind Rechensysteme, die einen Context zur Verfügung stellen [KLO97]. Neben Programmcode und Daten werden bei der Migration auch die Stati der Objekte mitführt. Dazu werden die Daten des Heaps serialisiert, an den Agenten gebunden und zusammen mit diesem übertragen. Die Migration eines Agenten erfolgt über die Methode `dispatch(URL url)`. Damit ist eine transparente Migration gewährleistet. Die Ausführung des Agenten wird unterbrochen und auf dem entfernten System mit Aufruf der Methode `onArrival` fortgesetzt. Entwickler müssen sich nicht um die Zusammenstellung der transitiven Hülle eines Agenten, dessen Serialisierung und anschließender Übertragung zu einem entfernten Rechner kümmern. Agenten verfügen über Methoden, die vor bzw. nach dem Auf-

[12] Es muß jedoch darauf hingewiesen werden, daß Aglets eine Mischung aus Java Applications und dem Konzept des Agenten ist.

[13] Der Bezeichner beinhaltet den Klassennamen eines jeden Aglets.

tritt eines Ereignisses aufgerufen werden. Zu den Methoden gehören u. a. `onCreation`, `onArrival` und `onDisposing`.

Die Kommunikation zwischen Agenten erfolgt mit Nachrichten-Objekten. Es werden keine Methoden an anderen Agenten aufgerufen. Die Kommunikation mit der Umgebung erfolgt über die `AgletContext`-Schnittstelle.

Bestandteil der Aglets Entwicklungsumgebung ist das Administrationswerkzeug *Tahiti*. Mittels Tahiti kann der Lebenszykluses von Agenten beeinflußt werden. Agenten können mit der Oberfläche erzeugt, deaktiviert oder terminiert werden. Desweiteren können Agenten mittels Tahiti auf entfernte Systeme migrieren. Für die Kommunikation mit Agenten stellt die Oberfläche ebenfalls eine Methode zur Verfügung (Siehe Abbildung 3.3).

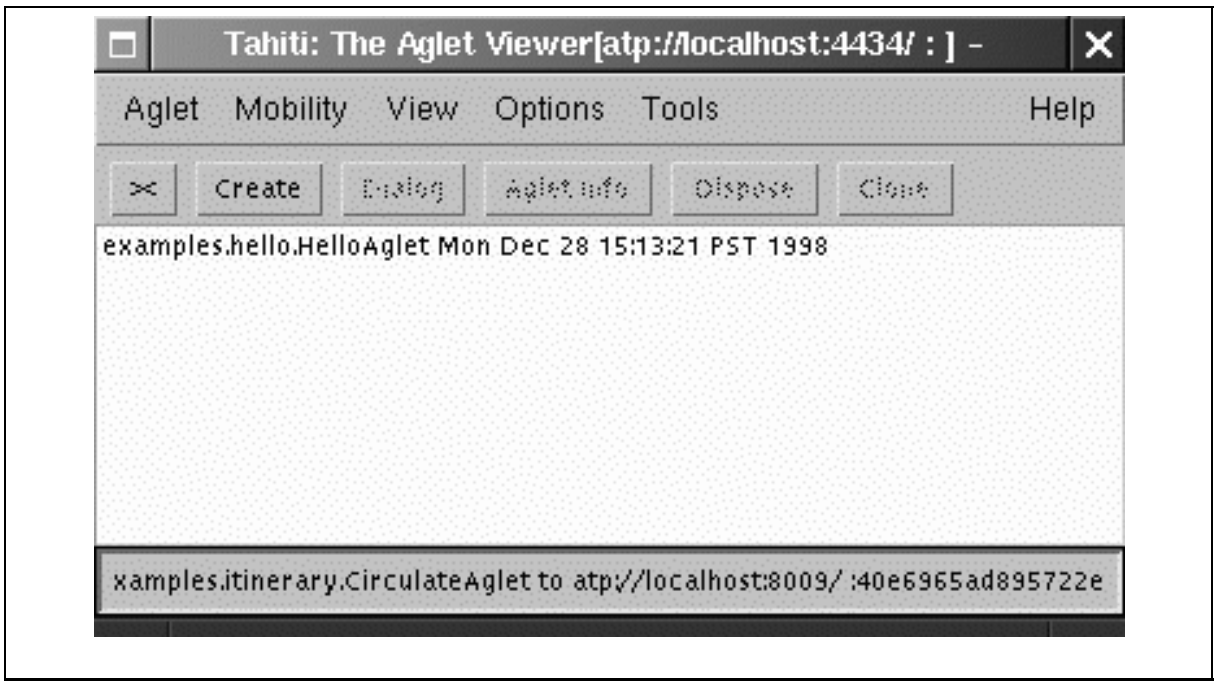

Abbildung 3.3.: Tahiti ist die administrative Oberfläche für das Aglets System.

Aglets unterstützt die Persistierung von Agenten. Auf Seiten der jeweiligen Server können Datenbanken bereitstehen, in denen Agenten auf Anfrage abgelegt werden. Die Persistierung erfolgt in Aglets durch die Zuweisung des Zustands „Deaktiviert“ zu einem Agenten. Der Agent wird dann temporär aus dem Context entfernt und im Hintergrundspeicher abgelegt. Eine nachfolgende Referenzierung des Agenten hat zur Folge, daß er durch die Umgebung reaktiviert wird.

Die abstrakte Basisklasse `Aglet` enthält finale Methoden zur Steuerung des Lebenszyklus. Die Entwicklung von Agenten folgt einem ereignisorientiertem Schema [KLO97].

3.4.4. Applets

Applets sind mobile Agenten, die dynamisch mittels eines Java-fähigen Browsers über ein Netzwerk bezogen werden. Applets existieren nur im Verbund mit HTML-Dokumenten, die eine Referenz auf eine Java-Klasse enthalten. Alle von Browsern erzeugten Java-Objekte werden als Applet bezeichnet. Objekte, die unabhängig von einem Browser instantiiert werden können, werden in der Sun-Terminologie *Applications* genannt. Applications unterliegen keinen Beschränkungen bezüglich der von ihnen durchgeführten Aktionen.

Beim Abfragen eines HTML-Dokumentes mit einer Referenz auf einer Java-Klasse wird die Klasse, wenn sie nicht bereits auf dem Ausgangsrechner vorhanden ist, übertragen. Anschließend erzeugt der verwendete Browser ein Objekt der Klasse, welches dann zur Ausführung kommt. Wird die Seite verlassen, werden alle im Dokument enthaltenen Agenten gestoppt.[14] Beim erneuten Aufruf des HTML-Dokuments werden alle enthaltenen Agenten wieder aktiviert. Erst das Beenden des Browsers führt zur Entfernung der Agenten aus dem Speicher. Agenten können deshalb nur während der Laufzeit des Browsers existieren. Sie haben nur Zugriff auf Ressourcen, die ihnen durch den Browser zur Verfügung gestellt werden. Der Browser stellt für die Agenten einen *Sandkasten* (engl. *sandbox*) dar, aus dem sie nicht ausbrechen können sollten.

Applets und Applications liegen in einem Zwischencode, dem sog. *Bytecode*, vor. Dieser ist plattformunabhängig und erfordert lediglich das Vorhandensein einer *virtuellen Maschine* – der Java Virtual Machine (JVM) – auf der der Code ausgeführt werden kann. Mittlerweile gibt es für viele Betriebssysteme eine entsprechende Implementation. Somit erfüllen Applets die notwendigen Bedingungen, um in einem heterogenen Netzwerk als Agenten agieren zu können.

Nachteilig in Hinblick auf mobile Agenten wirkte sich allerdings aus, daß Applets bis vor kurzem nur einmal, vom Server zum Client, migrieren konnten. Im Zuge der Erweiterung des Java Development Kit (JDK) (Version 1.1.1) wurden jedoch die neuen Techniken *RMI* und *Serialization* entwickelt. Mit Hilfe dieser beiden Techniken wurden Bibliotheken entwickelt, die es erlauben, daß Agenten selbständig innerhalb eines Netzes migrieren können, ohne daß sich die Migration auf die Richtung Server → Client beschränkt.

3.4.5. Castanet

Die Nutzung von Diensten im World-Wide Web (WWW) erfolgte bislang ausschließlich im *Pull*-Verfahren, d. h., der Anwender bezieht aktiv Dokumente oder Programme (Agenten) aus dem Internet. Neue Entwicklungen wenden sich stattdessen der *Push-Metapher* zu. Über Push-Technologien lassen sich Dokumente bzw. Software automatisch

[14] Das Verhalten beim Verlassen eines HTML-Dokuments kann vom Agenten mit Hilfe der Methode `stop()` beeinflußt werden. Die Methode wird durch den Browser – vor Aufruf eines neuen Dokuments – am Agenten aufgerufen.

aus dem Netz beziehen, installieren, aktualisieren und überflüssige Versionen automatisch löschen. Mit diesem Publish-and-Subscribe Paradigma ist es zunächst weiterhin notwendig, Informationen im Netz selbständig zu finden. Die sich anschließenden Tätigkeiten lassen sich dann jedoch automatisieren.

Das von Marimba entwickelte System Castanet ist eines der ersten Produkte, daß der Push-Metapher vor dem vorherrschenden Pull-Paradigma den Vorzug gibt. Castanet ist ein Broadcasting-System zur Verteilung von Software und Dokumenten. Das System Castanet ist zwar unabhängig vom transportierten Inhalt konzipiert, derzeit können mit ihm jedoch nur ausführbare Programme verteilt werden, die auf der JVM ausgeführt werden können. Castanet besteht im wesentlichen aus drei Komponenten – *Transmitter*, *Channel*, *Tuner* – deren begriffliche Bezeichnungen den schon bestehenden Broadcastingsystemen, wie beispielsweise dem Fernsehen, entliehen sind.

Transmitter sind Prozesse, die auf Seiten des Anbieters ablaufen. Sie dienen der Verteilung und der Pflege der Channels. Dabei steht die Metapher des Abonnierens eines Fernsehkanals (engl. channel) im Hintergrund. Die Dienste eines Transmitters werden von den Clients in Anspruch genommen, in dem sie sich für einen Dienst registrieren. Sie abonnieren einen Channel. Allen Abonnenten wird automatisch ein aktualisierter Channel übermittelt, so vom zuständigen Entwickler eine neue Version freigegeben wurde. Ziel der Aktualisierungen sind die Tuner der Abonnenten. Transmitter sind in der Lage, gleichzeitig verschiedene Channels zu verwalten.

Channel ist die Bezeichnung für ein Dokument oder eine Applikation. Diese werden durch den Transmitter über ein Intranet bzw. das Internet verteilt. Channels werden auf dem lokalen Speichermedium des Client gespeichert. Sie sind *self-contained*, d. h., sie verhalten sich wie normale Anwendungen.

Derzeit existieren vier unterschiedliche Kategorien von Channels:

- ▷ Ein HTML-Channel ist eine Sammlung von HTML-Dokumenten, mit einer definierten Eingangsseite.[15]

- ▷ Ein Präsentations-Channel ist eine skript-fähige Benutzungsschnittstelle, die mit dem von Marimba entwickelten *Bongo Presentation Builder* erzeugt wird.

- ▷ Ein Applet-Channel ist ein Java Applet, das optional nativen Code enthalten kann.

- ▷ Ein Application-Channel ist eine Java Application, die optional nativen Code enthalten kann [Mar97b].

Channels arbeiten bidirektional, d. h., der im wesentlichen vom Transmitter zum Tuner erfolgende Informationsfluß kann abhängig von der Anwendung auch in der umgekehrten

[15] Die HTML-Dokumente können darüberhinaus weitere eingebettete Objekte enthalten.

Richtung erfolgen. Channels können durch den Entwickler für den Anwender personalisiert werden. Hierzu werden in den Transmitter *Plug-ins*[16] eingebunden, die abhängig vom Anwender verschiedene Tätigkeiten ausführen und unterschiedliche Informationen weiterleiten.

Channels besitzen Eigenschaften von Java-Applets und von normalen Anwendungen:

- ▷ Sie sind auf verschiedenen Plattformen lauffähig;
- ▷ sie werden aus einem Netzwerk bezogen;
- ▷ sie unterliegen nahezu den gleichen Beschränkungen wie Applets;
- ▷ sie laufen in einem eigenen Fenster ab und
- ▷ sie werden auf der Festplatte abgelegt, von wo aus sie zugreifbar sind, wenn das Netzwerk nicht verfügbar ist.

Im Unterschied zu Applets haben Channels Lese- und Schreibzugriff auf ein spezielles Verzeichnis, das vom Tuner angelegt wird.

Ein Channel kann derart aufgesetzt werden, daß das Nutzungsverhalten (z. B. Verfolgung von genutzten Links in HTML-Dokumenten oder genutzte Funktionen eines Applets) des Anwenders protokolliert und an den Transmitter übermittelt wird. Hierzu existiert ein Rückkanal (BackChannel) zum Transmitter. Anwender können über eine Option den Versand von Logging-Informationen an den Transmitter unterbinden. Der Rückkanal dient allerdings gleichzeitig dazu, Channels dynamisch an die Wünsche des Anwenders anzupassen.

Die Einbindung des Anwenders in das System erfolgt über den *Tuner*. Entsprechend dem Prinzip eines Fernsehers oder Radios dient er als Empfangskomponente. Die Push-Technologie von Castanet wird durch ein periodisches Pollen des Transmitter durch den Tuner realisiert. Das scheinbare Push-Verhalten des Servers wird somit durch ein initierendes Pull des Clients angestoßen. Mittels des Tuners werden Channels installiert, aktualisiert und bei Bedarf werden Teile gelöscht. Der Anwender hat die Möglichkeit, aus der Menge der Angebote durch die Abonnierung eines oder mehrerer Channels die für ihn wichtigen Informationen auszuwählen. Nach dem Abonnieren eines Channels werden in der Folge automatisch Dokumente bzw. Programme auf dem Rechner lokal abgelegt bzw. installiert werden. Dort stehen sie dann dem Anwender zur Verfügung, der diese mit einem Browser betrachten bzw. ausführen kann. Neben der Abonnierung der Channels erfolgt über den Tuner auch deren Start und Aktualisierung. Die Aktualisierung von Channels erfolgt differentiell. Aktualisierungen großer Dateien erfolgen als Editierangaben, die vom Tuner durchgeführt werden. Schließlich dient der Tuner ebenso

[16] Diese als Plug-ins bezeichneten Komponenten stehen in keinem Zusammenhang zu jenen aus Abschnitt 3.4.6.

dazu, den Channels bestimmte Systemressourcen zuzuordnen. Der Tuner wiederum ist selbst ein Channel und wird wie die übrigen Komponenten automatisch aktualisiert.

Repeater werden es in Zukunft ermöglichen, Software von einer administrativen Site gleichzeitig an tausende oder Millionen von Anwendern zu verteilen.

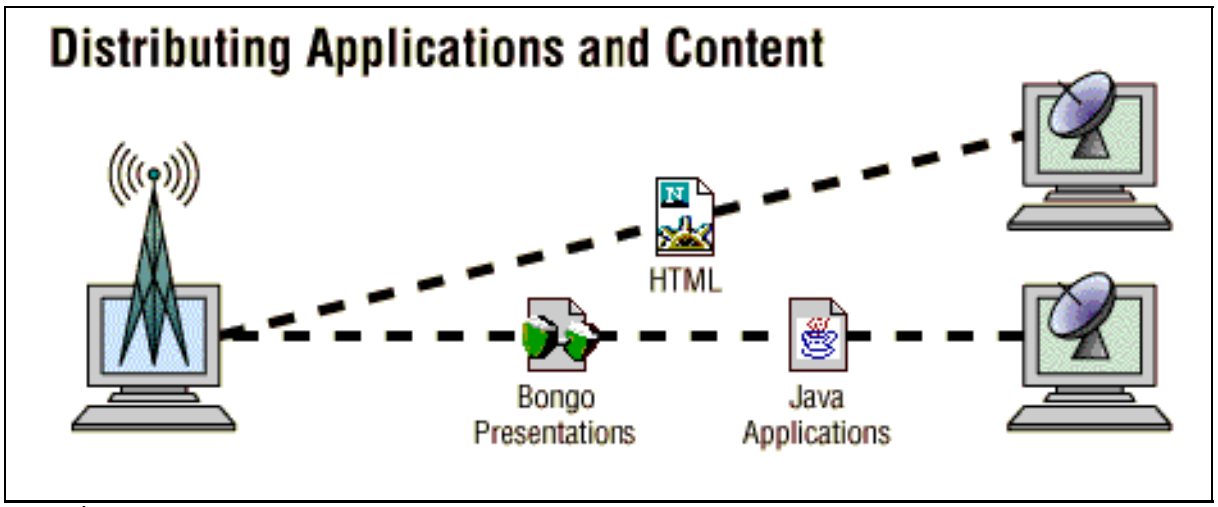

Abbildung 3.4.: Castanet entspricht dem *Push*-Paradigma, das sich durch den automatischen Bezug von Inhalten von dem *Pull*-Paradigma unterscheidet (`Quelle: Marimba`).

3.4.6. Plug-ins

Plug-ins sind Software-Komponenten, die sich in den Browser Communicator einklinken. Ähnliche Verfahren werden bereits von Bildverarbeitungssoftware und DTP-Anwendungen eingesetzt. Auf diesem Wege läßt sich die Funktionalität einer Anwendung erweitern, ohne daß die bestehenden Funktionen beeinträchtigt werden. In den meisten Fällen handelt es sich nicht um Erweiterungen, die durch Netscape erstellt, sondern um Produkte, die von Drittanbietern entwickelt wurden.

Mittlerweile gibt es eine Vielzahl von derartigen Plug-ins, die sich in den Communicator einbinden lassen. Plug-ins dienten zunächst dazu, die sog. Helper Applications abzulösen. Der Einsatz von Helper Applications zur Darstellung bzw. Wiedergabe eines MIME-Objektes erfordert, daß dieses zunächst auf einem lokalen Hintergrundspeicher abgelegt und anschließend die Helper-Applikation gestartet wird. Wird stattdessen ein Plug-in eingesetzt, verbleibt das MIME-Objekt im Hauptspeichersegment des Browsers, von wo aus es direkt durch das aufgerufene Plug-in verarbeitet wird.

Plug-ins werden ebenso wie ActiveX-Agenten auf den Hintergrundmedien eines Rechensystems installiert. Hierfür steht ein extra ausgewiesenes Verzeichnis bereit, in dem die Installation der Plug-ins erfolgt. Die Installation eines Plug-ins wird dem Anwender automatisch angeboten, wenn ein HTML-Dokument ein MIME-Objekt referenziert, für das sich weder eine Helper Application noch ein Plug-in registriert haben. In diesem

Fall wird der Server des Browser-Herstellers kontaktiert. Steht hier ein entsprechendes Plug-in bereit, wird dem Anwender die Möglichkeit geboten, dieses zu beziehen und zu installieren. Zulässig ist aber auch ein Verweis innerhalb des Dokuments auf einen Server, der das entsprechende Plug-in bereithält.

Erfolgt die Installation eines Plug-ins, so wird sich das Plug-in am Browser für ein oder mehrere MIME-Objekte registrieren.[17] Wird ein MIME-Objekt empfangen, für daß eine Registrierung eines Plug-ins existiert, wird das Plug-in geladen und eine Instanz erzeugt, die direkt in den Adreßraum des Browsers geladen wird. Der Instanz wird anschließend das Datenobjekt übergeben.

Wird der Browser beendet, bzw. das HTML-Dokument mit dem MIME-Objekt verlassen, wird die Instanz des Plug-ins terminiert. Die Termination wird dem Plug-in durch den Aufruf der Methode `NPP_Destroy` mitgeteilt. Das Plug-in wird aus dem Speicher entfernt, wenn die letzte Instanz des Plug-ins terminiert wurde. Im Gegensatz zu Java-Applets werden Plug-in-Instanzen aus dem Speicher entfernt, ohne daß der Entwickler des Plug-ins den Vorgang unterbinden kann. Die Instanz eines Plug-ins kann somit nicht weiterexistieren, wenn der Anwender das HTML-Dokument verläßt, das die Instantiierung des Plug-ins auslöste.

Neben der impliziten Kontrolle des Lebenszykluses eines Plug-ins bietet der Browser keine Möglichkeiten, direkten Einfluß auf den Lebenszyklus eines Plug-ins zu nehmen. Zur Laufzeit ist dem Communicator nicht zu entnehmen, ob ein Plug-in aktiv ist. Im Hilfe-Menue des Browser läßt sich lediglich feststellen, welche Plug-ins für den Browser installiert sind (Siehe Abbildung 3.5). Nicht entnehmbar ist die Funktionalität, die ein Plug-in beinhaltet. Bis vor kurzem dienten die Erweiterungen ausschließlich der Darstellung von Multimedia-Objekten (Animationen, Grafiken, Videos). Inzwischen werden aber bereits Plug-ins entwickelt, mit denen sich Skripte innerhalb eines Browser ausführen lassen. Bespiele hierfür sind das Tcl-Plug-in von Sun und das Shockwave-Plug-in von Macromedia. Auf diese Weise wird es möglich, HTML-Dokumente zu entwerfen, die Skripte der angeführten Sprachen enthalten. Mittels des Plug-in können die Skripte dann auf der Client-Seite ausgeführt werden.

Der Communicator bietet Plug-ins eine API, über die die Kommunikation zwischen Browser und Plug-in erfolgen kann. Folgende Aktionen werden durch die Schnittstelle unterstützt:

- ▷ die Registrierung des Plug-ins für ein oder mehrere MIME-Objekte,
- ▷ Zeichnen in das Browser-Fenster,
- ▷ Empfang von Tastatur- und Maus-Events,
- ▷ Bezug von Daten über URLs und

[17] Die auf einem Rechner installierten Applikationen können vom Benutzer ebenfalls für bestimmte Dateiformate am Browser registriert werden. Sie werden dann als Helper Application eingebunden.

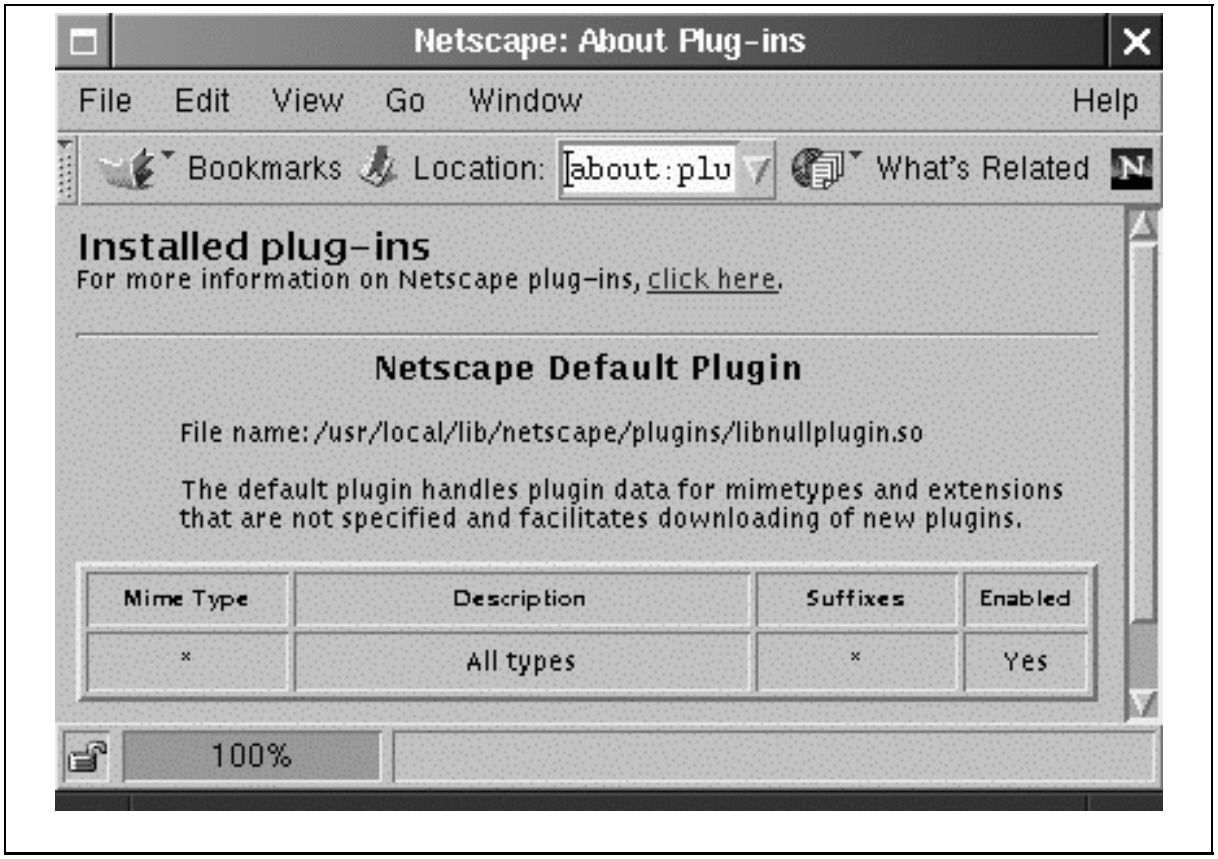

Abbildung 3.5.: Der Netscape Communicator stellt einen Menuepunkt bereit, mit dem Einblick auf die installierten Plug-in-Agenten genommen werden kann.

▷ Versand von Daten an URLs.

Ausgaben, die durch ein Plug-in produziert werden, können sowohl innerhalb des Browsers als auch außerhalb, in einem eigenen Fensters, dargestellt werden.

Bereits in Abschnitt 3.4.2 wurde darauf hingewiesen, daß ActiveX-Agenten und Plug-ins verwandte Technologien darstellen. Ebenso wie Controls sind Plug-ins plattformabhängig. Die Mehrzahl der Plug-ins liegt im Intel Binärformat vor. Zusätzlich gibt es einige Plug-ins auch für andere Communicator-Plattformen.

Zusammen mit ActiveX-Controls können Plug-ins als schwache Agentensysteme bezeichnet werden. Mit Ausnahme der Kommunikativität sind alle anderen Kerneigenschaften nur schwach ausgeprägt.

3.4.7. Servlets

Servlets stellen das Gegenstück zu Applets dar; sie können als serverseitige Applets betrachtet werden. Servlets eröffnen die Möglichkeit, Java-Code vom Client zum Server

migrieren zu lassen. Mit den Servlets liegt damit ein weiteres plattformunabhängiges Agentensystem vor, daß auf Java basiert.

Servlets sind protokoll- und plattformunabhängige serverseitige Komponenten, geschrieben in Java, welche dynamisch javafähige Server erweitern. Sie bilden ein allgemeines Gerüst für Dienste, die der Frage–Antwort-Metapher gehorchen [SUN96].

Servlets können aus dem lokalen `CLASSPATH`, einem lokalen `servlet`-Verzeichnis, welches nicht im `CLASSPATH` enthalten ist, oder von einem entfernten Rechner geladen werden. Ihrem Grundsatz der dynamischen Erweiterbarkeit entsprechend, können sie so im Stile eines Plug-ins, das die Funktionalität eines Browsers erweitert, die Funktionalität eines Servers erweitern. Auf diese Weise läßt sich beispielsweise ein Server mit Hilfe von Servlets um die Funktionalität einer Suchmaschine erweitern. Ausgehend von einem Servlet-Server mit einigen Clients kann so eine vielstufige verteilte Applikation entwickelt werden.

Servlets werden am Server für bestimmte Dienstleistungen registriert. Bei Anforderung einer Dienstleistung wird das zugehörige Servlet instantiiert, so sich noch keines im Speicher befand. Anschließend wird an dem Servlet die Methode `service()`[18] aufgerufen, die den Dienst des Servlets implementiert. Das Servlet verbleibt im Speicher, bis der Web-Server terminiert wird. Solange durch den Server keine administrativen Funktionen zur Terminierung von Servlets bereitgestellt werden, stellt die Terminierung des Servers den einzigen Weg dar, um Servlets aus dem Speicher zu entfernen.

Servlets können auf unterschiedliche Weise aufgerufen werden:

1. Der Client ruft ein Dokument von einem Web-Server ab, das durch ein Servlet bedient wird. I. d. R. fungiert das Servlet in diesen Fällen als Filter.

2. Ein Servlet kann direkt unter Angabe der konkreten URL[19] aufgerufen werden.

3. Der Client fordert ein Server Side Include (SSI) Dokument an. SSI-Dokumente erhalten einen Verweis auf ein Servlet. Sie müssen zunächst vom Server geparst werden, bevor sie an den Client geschickt werden. Auf diese Weise lassen sich in einem Zwischenschritt Inhalte dynamisch hinzufügen.

Die Kommunikation eines Servlets mit seiner Umgebung erfolgt über den Servlet-Context. So können beispielsweise Umgebungsvariablen ausgelesen werden.

Servlets sollen eine erhebliche Steigerung der Effizienz und Performanz auf Serverseite gestatten. Serverseitige Anwendungen, die auf dem Common Gateway Interface (CGI) aufbauen, erfordern, daß für jede Anfrage ein Subprozeß gestartet wird, der möglicherweise eine Datenbankanfrage stellt und anschließend nach Übersendung des Ergebnisses terminiert. Ein derartiges Vorgehen ist nicht tragbar, wenn Web-Server mit tausenden Anfragen innerhalb kürzester Zeit konfrontiert werden. Im Falle von CGI-Anwendungen,

[18] Die Methode `service()` muß von jedem Servlet implementiert werden.
[19] z. B. `http://server_host/servlet/<servlet name>?arguments`

die mittels Skriptsprachen implementiert sind, muß für jeden Subprozeß ein Interpreter instantiiert werden. Servlets bieten hier den Vorteil, daß sie einmalig instantiiert werden und anschließend für eine beliebige Anzahl von Anfragen bereitstehen. Alle Servlets operieren in einer JVM, so daß auch auf diese Weise ein großer Teil des Overheads vermieden wird. Servlets sind darüberhinaus plattformunabhängig, im Gegensatz zu perfomanten CGI-Anwendungen auf Basis von C. Aus diesen Gründen positioniert Sun Servlets als Basis für die Entwicklung von Anwendungen, die auf Web-Servern angesiedelt sind.

Unterstützt werden Servlets durch den Web-Server JavaServer[20] von Sun. Daneben existieren eine Reihe von Erweiterungen für andere Server, die diese in die Lage versetzen, Servlets zu unterstützen. Die Erweiterung beinhaltet die Unterstützung der Servlet-API. Darüberhinaus muß der Server auch eine JVM beinhalten, so daß die Servlets ausgeführt werden können. Mit Java 2 ist die Unterstützung von Servlets Teil der Entwicklungsumgebung. Die Schnittstelle für Servlets wird seitdem vom Basispaket unterstützt.

3.4.8. Voyager

Voyager ist eine von Objectspace entwickelte Bibliothek, die – ebenso wie Aglets – Java um das Paradigma des mobilen autonomen Agenten erweitert. Voyager-Programme sind Java-Applications, die mit dem von Objectspace entwickelten Virtual Class Creator (VCC) vorkompiliert werden.[21] Der VCC erfüllt die Funktion eines Präprozessors. Er erzeugt zusätzlich zu jedem übergebenen Klassentext eine weitere Klasse, die sich nach außen durch ein vorangestelltes 'V' von der ursprünglichen unterscheidet.[22] Die neu erzeugte Klasse dient als Proxy-Objekt[23] für den Zugang zur ursprünglichen Klasse. Über ein Proxy-Objekt ist es möglich, auf entfernte Objekte zuzugreifen. Beide Klassen werden abschließend mit einem herkömmlichen Java-Compiler kompiliert.

Wie aus Abbildung 3.6 zu ersehen ist, findet die gesamte Kommunikation zwischen einem Aufrufer und einem entferntem Objekt vollständig lokal über das Proxy-Objekt statt. Proxy-Objekte übernehmen die lokalen Nachrichten und senden diese an das entfernte Objekt. Ebenso werden Ergebnisse des entfernten Objekts zunächst an dessen Proxy-Objekt weitergeleitet und anschließend von diesem an den Aufrufer übergeben. Zu beachten ist, daß einem entfernten Objekt eine beliebige Anzahl von Proxy-Objekten zugeordnet sein können, jedem Proxy-Objekt entspricht dagegen genau ein entferntes Objekt ($1 : n$-Beziehung). Verbindungen zu entfernten Objekten werden in Voyager aufgebaut, indem die Umgebung – der Voyager-Server – angewiesen wird, für ein bestimmtes entferntes Objekt ein Proxy-Objekt zu erstellen (`VObject.forObjectAt(GUID)`). Alle Methoden, in denen auf entfernte Objekte zugegriffen wird, müssen Blöcke zur Behandlung der möglichen Ausnahmen implementieren. Klassen werden auf entfernte Rechner

[20] http://jserv.javasoft.com

[21] Mit der zweiten Betaversion des Voyager Kernpakets 2.0 ist die Verwendung des VCC nicht mehr notwendig.

[22] Zu einer Klasse `Object` erzeugt der VCC die Klasse `VObject`.

[23] In der Objectspace-Terminologie als virtuelle Referenz bezeichnet.

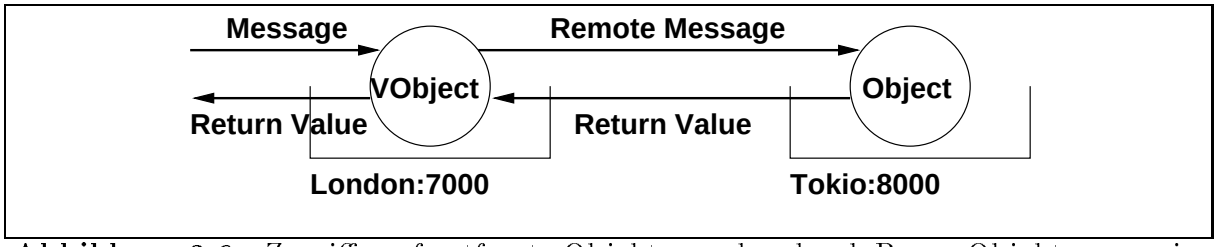

Abbildung 3.6.: Zugriffe auf entfernte Objekte werden durch Proxy-Objekte, sog. virtuelle Referenzen, gekapselt. Durch den Einsatz von Proxy-Objekten kann auf entfernte Objekte zugegriffen werden, so wie dies auf lokale Objekte erfolgen würde.

nach dem *Load on demand*-Prinzip transportiert. D. h., kann ein Objekt nicht instantiiert werden, weil der Klassentext auf dem entfernten System nicht vorhanden ist, wird dieser dynamisch vom Quellsystem bezogen und es wird eine Instanz der Klasse angelegt.

Voyager Objekte können zwischen Rechnern bewegt werden. Die Migration der Objekte erfolgt über die Methode `moveTo()`, wobei das Ziel ein Voyager Server ist. Agenten sind Voyager Objekte, die selbstständig migrieren können. Agenten können als Ziel der Migration neben einem Server ein Objekt oder einen anderen Voyager Agenten angeben. Der Migrationswunsch eines Agenten veranlaßt die Umgebung dazu, Kopien der vom Agenten referenzierten Objekte zu erzeugen und diese gemeinsam mit dem Agenten zu serialisieren.

Agenten und Objekte werden erst dann durch die Umgebung serialisiert und übertragen, wenn alle laufenden Nachrichten an die Agenten bzw. Objekte verarbeitet worden sind. Für Nachrichten, die erst während des Serialisierungsprozesses oder später eintreffen, werden *Forwarder* eingerichtet, die auf den neuen Aufenthaltsort verweisen und die Nachricht dorthin weiterleiten. Als Ergebnis liefert der Forwarder die neue Adresse des entfernten Objektes bzw. des Agenten an den Aufrufer weiter, mit der Folge, daß dessen virtuelle Referenz automatisch aktualisiert wird. Alle in Voyager implementierten Agenten erben von der Klasse `com.objectspace.voyager.agent.Agent`. Die Klasse `Agent` bildet die Oberklasse aller Agentklassen.

Jedes Voyager Objekt wird mit einem Globally Unique Identifier (GUID) versehen, einem 16-Byte langen, global eindeutigen Bezeichner [Obj97]. Der GUID oder ein dem Objekt zugewiesener Alias wird benötigt, um ein entferntes Objekt zu lokalisieren. Unterstützung findet der Prozeß der Objektlokalisierung durch einen rudimentären Verzeichnisdienst, der in Voyager bereits integriert ist.

Voyager verfügt über einen Persistierungsmechanismus, der es erlaubt, beliebige Objekte in eine Datenbank zu schreiben. Der Aufruf einer Methode an einem Objekt, daß sich nicht im Hauptspeicher befindet, von dem aber eine persistente Kopie existiert, hat zur Folge, daß das Objekt automatisch aus der Datenbank geladen und instantiiert wird.

4. Sicherheitsrisiken in Netzumgebungen

> „*The style of computing being pushed out to consumers is inherently risky and must be implemented with substantial controls if it is to be used safely*“
> – Fred Cohen

Agenten, insbesondere mobile Agenten, operieren in offenen Systemen. In derartigen Systemen existieren eine Vielzahl von Bedrohungen, die inhärent im System begründet sind. Neben den im System liegenden Bedrohungen werden zusätzliche von außen in das System hineingetragen. Für eine Risikoanalyse ist es erforderlich, daß zunächst die Bedrohungen eines IT-Systems identifiziertwerden, so daß auf deren Grundlagen die Risiken beurteilt werden können. Im Anschluß folgen die Empfehlungen für Gegenmaßnahmen.

Ein wesentliches Merkmal von Agentensystemen ist, daß von ihnen nicht nur Bedrohungen ausgehen, sondern sich Bedrohungen ebenso gegen Agenten wenden können. Zu beachten ist, daß Bedrohungen von Innen- und Außentätern ausgehen können und ein besonderes Gefahrenpotential dann gegeben ist, wenn Außen- und Innentäter kooperieren.

4.1. Szenarien

In Form von drei Szenarien soll einführend deutlich gemacht werden, welche Bedrohungen der Einsatz von Agenten mit sich bringt bzw. welchen möglichen Bedrohungen sie unterliegen. Die Szenarien nehmen Bezug auf die Anwendungsfelder von Agenten, wie sie in Kapitel 7 untersucht werden.

4.1.1. Shopping Agent

ShopAgent ist ein Agent, der es dem Benutzer ermöglicht, anhand der Angabe eines Produkts und einer Reihe gewünschter Aspekte, den günstigsten Anbieter im Netz zu finden. In Abstimmung mit dem Anwender ist der Agent in der Lage, das entsprechende Produkt bei dem günstigsten Anbieter zu kaufen. Damit der Anwender nicht die ganze

Zeit mit dem Netzwerk verbunden sein muß, migriert der Agent zu den jeweiligen Systemen der Anbieter und agiert dort als legitimer Prozeß. Nach Abschluß der Aufgabe kehrt er auf das System des Anwenders zurück.

Dieses Szenario offenbart eine Reihe von Gefahren, die bei derartigen Transaktionen auftreten können. Dazu gehört die Möglichkeit, daß die Angaben des Anbieters nicht der Realität entsprechen oder der Agent von diesem dahingehend manipuliert wird, daß alle anderen Angebote über dem eigenen liegen. Schließlich ist es dem Anbietersystem möglich, den Agenten auf dem eigenen System festzuhalten. Eine Gefahr, die insbesondere in Hinblick darauf, daß Agenten elektronische Zahlungsmittel mit sich führen können, eine ernstzunehmende Bedrohung darstellt.

Das Szenario des Shopping-Agents ist ein Beispiel dafür, daß Angriffe sich nunmehr nicht nur gegen Rechensysteme wenden, sondern in Zukunft auf Agenten ausgerichtet werden können. Damit eröffnet sich eine neue Dimension der Angriffsziele.

4.1.2. Information Agent

InfoAgent versteht die Vielzahl der unterschiedlichen Syntax, wie sie von Suchmaschinen zur Erstellung von Anfragen verwendet werden. Der Anwender nutzt InfoAgent, um Anfragen zu stellen, die von InfoAgent in die jeweilige Syntax der Suchmaschinen übersetzt werden. Anschließend kontaktiert der Agent die von ihm unterstützten Suchmaschinen, bewertet die Antworten der Suchmaschinen und stellt dem Benutzer die Ergebnisse dar.

InfoAgent erzielt seine Leistungsfähigkeit aus der Tatsache, daß er Profile der Anwender erstellt, die er bei der Bewertung der durch die Suchmaschinen gelieferten Ergebnisse verwendet. Die von InfoAgent erstellten Profile stellen für Marketingunternehmen eine Quelle von qualifizierten Informationen dar. Leitet der InfoAgent ohne Kenntnis des Anwenders die Profildaten an Dritte weiter, handelt es sich bei ihm um einen Trojaner (siehe hierzu Abschnitt 4.3.3), dessen Schadensfunktion in der Aufhebung der Vertraulichkeit durch die Weiterleitung der Profildaten besteht. Interessent für derartige persönliche Daten sind Wirtschaftsunternehmen, sowie Behörden; auch Krankenkassen und Geheimdienste könnten entsprechende Daten verwenden.

4.1.3. Mobile Computing Agent

MobileAgent unterstützt die Besitzer eines mobilen Computers (Laptop, Personal Digital Assistant (PDA)). Mit MobileAgent werden die Nutzer in die Lage versetzt, einen Agenten zu kreieren, der in ein Netz entsendet wird, um dort eine Aufgabe zu erledigen, während das mobile Gerät nicht mehr zwingend mit dem Netz verbunden ist. Verbindet sich der Nutzer nachfolgend wieder mit dem Netzwerk, kehrt der Agent selbständig auf das mobile Gerät zurück, sofern er seine Aufgabe bereits erledigt hat.

Während MobileAgent das Netzwerk durchquert und in unterschiedlichen Umgebungen die an ihn gestellte Aufgabe erfüllt, wird er nicht mehr unter der Kontrolle seines Besitzer, sondern unter der des ausführenden Systems ausgeführt. Dementsprechend hat

dieses die volle Kontrolle über die Daten und den Programmcode des Agenten. Der Manipulation des Agenten, in Form von Veränderungen an dem Code oder an den Daten, sind keine Grenzen gesetzt. Eine Trojanisierung oder die Infizierung des Agenten mit einem Virus kann nachfolgend dazu führen, daß auf dem mobilen Gerät des Anwenders eine Schadensfunktion ausgeführt wird.

4.2. Werte der Agentensysteme

Im ersten Schritt einer Risikoanalyse gilt es zunächst festzustellen, welche Werte in IT-Systemen vorhanden sind. Hierbei sind alle Werte aufzuführen, deren Verlust zu Nachteilen führen könnte.

4.2.1. Agenten

Agenten stellen in ihrer statischer Form, d.h. in Form des Programms, das sie repräsentiert, ein Wirtschaftsgut dar, das für das herstellende Unternehmen ein Wert ist. Programme, die von einem Unternehmen entwickelt werden, sind Eigenentwicklungen, deren Quelltext für ein Softwareunternehmen den gleichen Wert darstellt, wie die Baupläne eines Fahrzeugs für ein Automobilkonzern. Ein Wert bildet das Programm „Agent“ natürlich auch für die Kunden, die das Produkt in ihren IT-Prozessen einsetzen. Wird der Einsatz von Agenten hingegen verhindert, werden die Prozesse, in denen die Agenten eingesetzt wurden, behindert bis verhindert. Darüberhinaus stellt aber auch die dynamische Form des Agenten, der sich in Ausführung befindliche Agent, einen Wert dar, dessen Wiederherstellung mit dem Einsatz von Kapital und Zeit verbunden ist. Eine Terminierung eines Agenten, die vom Besitzer nicht intendiert ist, hat zur Folge, daß potentiell alle durch ihn gesammelten Informationen verloren gehen. Ihre Wiedergewinnung kann mit der erneuten Verrichtung einer Gebühr verbunden sein, wenn Informationen beispielsweise aus einem kommerziellen Informationssystem gewonnen wurden.

Der Prozeß Agent kann aus einem statischem Programmtext oder dynamisch zur Laufzeit durch andere Entitäten im Quellsystem generiert worden sein. Sein Verlust stellt damit die Anforderung auf, einen identischen Prozeß zu generieren.

4.2.2. Daten

Daten sind ein wesentlicher Bestandteile eines jeden IT-Systems. Demzufolge stellen sie in Agentensystemen einen der wesentlichen Werte dar. Der Inhalt von IT-Systemen ist die Verarbeitung von Daten. Außer dem Entzug der Maschine (Agent), wie er oben dargestellt wurde, können Prozesse beeinflußt werden, wenn falsche oder keine Daten zur Verfügung stehen. Daten befinden sich in den Rechensystemen, die die Umgebungen der Agenten darstellen und, wie bereits oben angeführt, in den Agenten selbst. Zu den

Daten in Agentensystemen gehören u.a. Geschäftsdaten, persönliche Daten, digitales Geld und geheime Schlüssel.

4.2.3. Ressourcen

Die Verarbeitung von Daten in IT-Systemen erfordert neben den Daten und der Vorschrift, wie diese zu Verarbeiten sind, Komponenten, die als Bestandteil des Verarbeitungsprozesses Daten darstellen, speichern und bearbeiten. Die Komponenten lassen sich zu dem Begriff Ressourcen zusammenfassen. Der Ausfall einer Komponente behindert oder verhindert gar die Ausführung eines Prozesses. Ein Schutz der entsprechenden Ressourcen ist für ein Agentensystem somit von immenser Bedeutung.

4.2.4. Reputation

Bislang wurden ausschließlich technische Aspekte als Werte in Agentensystemen vorgestellt. Darüberhinaus gibt es auch eine soziale Komponente, die als Reputation bezeichnet wird. In Prozessen wird ein Teilnehmer einer andere Organisation, eines anderen Unternehmens, als Repräsentant dessen aufgefaßt. Positive oder negative Leistungen des Repräsentanten werden auf das Repräsentationsobjekt abgebildet. Übertragen auf Agentensysteme bedeutet das, daß Agenten die Entität repräsentieren, die sie veröffentlicht hat. Der zunehmende Einsatz von Agenten als virtuelle Repräsentation in verteilten Umgebungen hat zur Folge, daß die Reputation einer Entität zunehmend von der virtuellen Repräsentanz in Form des Agenten abhängig ist. Angriffe gegen die virtuelle Repräsentanz können einen Verlust der Reputation nach sich ziehen. Dies ist insbesondere dann gegeben, wenn ein Fehlverhalten der virtuellen Repräsentanz dem Systembetreiber zugeschrieben wird. Gleiches gilt für den Besitzer eines Agenten, wenn der Agent durch implementationsbedingtes oder durch Manipulationen des Servers bedingtes Fehlverhalten auffällt. Ein Verlust der Reputation im Umfeld netzbasierter Dienste ist von entscheidender Bedeutung, da Kontakte und Transaktionen zunehmend virtuell erfolgen.

4.2.5. Umgebungen

Die einzelnen Bausteine eines Agentensystems – Agenten, Daten und Ressourcen – lassen sich logisch zu einer Umgebung zusammenfassen. Der logische Zusammenschluß stellt einen Mehrwert dar, denn erst die Bereitstellung von Schnittstellen und die Kooperation der verschiedenen Komponenten läßt Agenten produktiv werden. Damit ist neben der Existenz und der Sicherheit der Einzelkomponenten auch deren Zusammenwirken in Form des Konzepts der Umgebung ein Wert, den es zu wahren gilt.

4.3. Einpflanzungen

Eine wesentliche Gefahr droht IT-Systemen durch Einpflanzungen. Mittels Einpflanzungen werden Programme auf eine Weise manipuliert, so daß sie weiterhin ausführbar sind, ihre Funktionalität jedoch nicht mehr der ursprünglichen entspricht. Wie auch die in Abschnitt 4.4 aufgeführten Bedrohungen, können sich Einpflanzungen gegen Umgebungen wie auch gegen Agenten richten. Einpflanzungen jeglicher Natur können bereits in Agenten enthalten sein und während der Ausführung auf dem Umgebungssystem eine Schadensfunktion ausführen. Einpflanzungen können sich aber ebenso gegen Agenten richten, die ein Umgebungssystem aufsuchen. Ein Beispiel hierfür ist die Infizierung eines Agenten mit einem Virus.

Agenten und Umgebungssysteme können aktiv durch Einpflanzungen eine Schadensfunktion ausführen bzw. diese in andere Agenten oder Umgebungssysteme einpflanzen. Auf diese Weise kann gutartige Software zu maliziöser Software mutieren, ohne daß das in der Intention des Autors der Software lag.

Einpflanzungen können alle Sicherheitskriterien, wie sie in Abschnitt 2.3 definiert wurden, aufheben.

4.3.1. Falltüren

Falltüren (engl. *trapdoors*) oder auch *Hintertüren* (engl. *backdoors*) erlauben den Zugriff auf Programme unter Umgehung der normalen Authentifikationsmechanismen. Hintertüren werden i. d. R. von Entwicklern in Programme implantiert, um Debugging- oder Monitoring-Prozesse zu unterstützen. In der Entwicklung ist es hinderlich, längliche Authentikationsprozeduren oder Initialisierungsphasen zu durchlaufen. Verbleiben nach Abschluß der Entwicklung, derartige für die Entwicklung sinnvolle Hintertüren im Programmcode, eröffnen sich für Angreifer einfache Eintrittstüren.[1] Das wesentliche Problem von Falltüren besteht darin, daß mit ihnen Teile oder die gesamte Sicherheitspolitik umgangen werden kann.

Eine Aktivierung des Falltüren-Codes erfolgt entweder durch die Eingabe bestimmter Werte oder durch definierte Benutzerkennungen. Schlecht entworfene Programme sind eine weitere Quelle für Falltüren. So können inakzeptable Eingaben akzeptiert werden, anstatt daß eine Ausnahme generiert wird. GARFINKEL und SPAFFORD geben in diesem Zusammenhang den Hinweis, Software nur von vertrauenswürdigen Quellen zu akzeptieren, um so eine höhere Sicherheit zu haben, daß Falltüren zumindest nicht absichtlich in Softwarepaketen verblieben sind [GS97]. Falltüren machen ein System verwundbar, da sie Möglichkeiten eröffnen, Modifikationen innerhalb eines Systems durchzuführen. Hintertüren erlauben vielfältige Aktivitäten durch Angreifer: Installierung von manipulierten Systemprogrammen, Manipulation von Daten, Änderung von Zugriffsrechten.

[1] Eine der berühmtesten Hintertüren war in `sendmail` verlieben. Dort war es möglich, in einen Debug-Modus zu schalten, um so anschließend beliebige Kommandos ausführen zu können.

Die Entdeckung von Hintertüren ist kompliziert [GS97]. Es läßt sich die Frage aufwerfen, ob nicht Agenten ein großes Gefahrenpotential in diesem Zusammenhang darstellen.

4.3.2. Softwarebomben

Eine Softwarebombe besteht aus einer Menge von Instruktionen und wird durch bestimmte Bedingungen bzw. Zustände ausgelöst. Zu unterscheiden sind Logik- und Zeitbomben. Zeitbomben erfordern temporale, Logikbomben logische Auslösebedingungen. Eine logische Auslösebedingung kann in der Existenz einer bestimmten Datei bestehen, während eine temporale Auslösebedingung das Erreichen eines definierten Datums sein kann. Ein Programm, das eine Softwarebombe enthält, ist ein lauffähiges Programm und verhält sich solange normal, bis durch die Erfüllung einer Auslösebedingung ein Fehlverhalten initiiert wird. Das Fehlverhalten besteht i. d. R. aus einer unautorisierten, maliziösen Schadensfunktion.

4.3.3. Trojanische Pferde

Ein *Trojanisches Pferd* (engl. *trojan horse*) ist – in Anlehnung an die griechische Sage – ein Programm, welches über die dokumentierte Funktionalität hinaus eine nicht dokumentierte Funktionalität maliziöser Art beinhaltet. Die Schadensfunktion kann mit Eintritt eines bestimmten Ereignisses oder schleichend erfolgen.

Trojanische Pferde werden von autorisierten Benutzern ausgeführt. Sie machen sich den Umstand zu nutze, daß sie die Rechte des Anwenders erben, unter dessen Kontrolle sie ausgeführt werden. Die Schadensfunktion eines Trojaners muß nicht notwendigerweise Verwundbarkeiten innerhalb des Systems entdecken. Stattdessen nutzen Trojaner im wesentlichen wohl dokumentierte und wünschenswerte Funktionen. Dementsprechend verfügen sie über alle Möglichkeiten im System zu agieren, die dem Nutzer übertragen wurden. Lediglich in Systemen mit mandatorischer Zugriffskontrolle lassen sich Trojaner einschränken. In den gängigen Betriebssystemen ist hingegen meist nur eine diskrete Zugriffskontrolle implementiert. Gleiches gilt für die Sicherheitsarchitekturen, die in den Basistechnologien von Agentensystemen angewendet werden. Die Schwierigkeit, trojanischen Pferden zu begegnen, resultiert somit aus der Tatsache, daß sie nicht gegen die Sicherheitspolitik des Systems verstoßen.

Trojanische Pferde stellen auch dann ein Sicherheitsrisiko dar, wenn das auszuführende Programm ausschließlich oder zusätzlich in Form des Quelltextes vorliegt. Die Komplexität größerer Programme macht es unmöglich, diese eingehend auf das Vorhandensein eines Trojaners zu überprüfen. Werden von Angreifern zusätzlich Techniken eingesetzt, die die Existenz eines Trojaners im Programmtext verschleiern,[2] wäre der Einsatz von Analyse-Werkzeugen unerläßlich. Es ist unrealistisch anzunehmen, daß eine

[2] Entsprechende Methoden werden im Abschnitt 6.8 vorgestellt.

derartige Vorgehensweise von vielen Entitäten eingeschlagen wird, wenn Programme eingesetzt werden, die im Quelltext vorliegen und aus einem offenem Netz bezogen wurden.

THOMPSON hat bereits 1984 darauf hingewiesen, daß die Trojanisierung einer Software bereits während der Entwicklung erfolgen kann, ohne daß diese vom Autor der Software intendiert ist [Tho84]. Ausgehend von trojanisierten Entwicklungswerkzeugen kann jedes entwickelte Produkt trojanisiert werden. Damit weist THOMPSON eindrücklich darauf hin, daß bereits in der Entwicklung Sicherheitsmechanismen anzuwenden sind, so daß die Sicherheitsanforderungen der sich in Entwicklung befindlichen Anwendung erfüllen lassen. Insbesondere hat damit auch die Entwicklungsumgebung definierte Sicherheitskriterien zu erfüllen.

Für den Nutzer eines Agentensystems ist es heute mit verhältnismäßigem Aufwand nicht möglich, die Funktionalität eines Agenten zu ermitteln. Damit ist es Angreifern ohne Schwierigkeiten möglich, der informal versprochenen Leistung eines Agenten weitere Funktionalität hinzuzufügen. Eine Trojanisierung von Agenten ist aus Sicht des Autors eine der wesentlichen Gefahren, die für Agentensysteme besteht. Schon heute ist die Trojanisierung eine häufig zu findende Methode, wenn Betrug unter Einsatz von IT-Systemen erfolgt [Tha93]. In Hinblick darauf, daß das Problem der Entdeckung trojanischer Pferde sehr wahrscheinlich unentscheidbar ist,[3] kann nur verstärkt auf die Problematik der Trojaner hingewiesen werden.

4.3.4. Viren

Mit dem Begriff Virus werden Befehlssequenzen bezeichnet, die in der Lage sind, eine Kopie bzw. eine modifizierte Version der Befehlsfolge in einen Speicherbereich zu schreiben, der die Befehlsfolge bislang nicht enthält. Der Replikationsvorgang wird als *Infektion* bezeichnet. Computerviren können mit einer Schadensfunktion (engl. *payload*) ausgestattet sein, die über den aus dem Replikationsvorgang entstehenden Schaden hinaus, weitere Beeinträchtigungen erzeugen kann.

Viren benötigen ein Wirtsprogramm, von dem sie aus operieren. Ohne ein Wirtsprogramm sind Viren nicht lauffähig. Anhand der Wirtsumgebung lassen sich Viren in unterschiedliche Kategorien einteilen. *Bootviren* machen sich den Umstand zu nutze, daß beim Start eines Rechensystems zunächst der Bootsektor ausgelesen und der dort enthaltene Programmcode ausgeführt wird. *Makroviren* operieren innerhalb eines Wirtsprogramms, daß eine Makrosprache beinhaltet. Das in der PC-Welt weit verbreitete Office-Paket von Microsoft beinhaltet derartige Makrosprachen, die mit der neuesten Version des Office-Paketes vereinheitlicht wurden, so daß die Makroviren in diesem Fall nicht mehr an eine bestimmte Applikation – wie die Textverarbeitung oder Tabellenkalkulation – gebunden sind. Sie können damit applikationsübergreifend agieren. *Programmviren* infizieren Programme, die in Maschinencode vorliegen. Der Programmcode des Virus wird dabei entweder an den Anfang, in die Mitte oder an das Ende des zu infizierenden Programmes

[3] Bislang ist die Unentscheidbarkeit nicht formal bewiesen.

geschrieben. Der Programmanfang wird mittels eines Sprungbefehls dahingehend modifiziert, daß zunächst der Viruscode ausgeführt wird. Anschließend erfolgt ein Rücksprung zu dem eigentlichen Programm.

Die Bedrohung der Agententechnologie durch Viren ist in wesentlichem Maße gestiegen, seitdem mit *Strange Brew* der erste Virus existiert, der in Java entwickelt wurde und Java-Klassen infiziert [NCT98]. Bereits 1997 hatte LaDue theoretisch dargelegt, daß die Entwicklung von Viren unter Java möglich ist [LaD97]. Neben den Makroviren existieren mit den Java-Viren jetzt eine weitere Klasse von plattformübergreifenden Viren. Damit besteht die Gefahr, daß eine neue Generation von Viren entwickelt werden wird.

Viren stellen eine ähnliche große Gefahr für Agentensysteme dar, wie dieses bereits für Trojaner dargelegt wurde. Hinzu kommt, daß Cohen formal bewiesen hat, daß die Entdeckung von Viren im allgemeinen ein nicht-entscheidbares und damit nicht lösbares Problem ist.

4.3.5. Würmer

„Ein Wurmprogramm ist [...] Software, die so gestaltet wurde, daß Segmente dieser Software auf verschiedenen Rechnern eines Netzwerks residieren und alle Einzelstücke überlebens- und ablauffähig sind [Tha93]."

Eingeführt wurde der Begriff des Wurms von Shoch und Hupp, die 1982 einen Wurm folgendermaßen definierten:

Definition 5:

„A worm is simply a computation which lives on one or more machines. The programs on individual computers are described as the *segments* of a worm [...]."

Die Notwendigkeit der kritischen Betrachtung heutiger Agentensysteme resultiert zu einem beträchtlichen Teil aus dem Umstand, daß die den Agenten verwandten Würmer in der Vergangenheit vorwiegend negative Effekte auf ihre Umgebung hatten. Würmer waren häufig ohne Wissen des Systembetreibers auf Rechensystemen aktiv. Parasitisch wurden System-Ressourcen für eine Tätigkeit verbraucht, die nur dem Wurm diente und dem Wirtssystem nur Schaden zufügte.

Erkannt haben dies bereits Shoch und Hupp in ihrer Arbeit [SH82]. Ein unstabiler Wurm führte zu einer unkontrollierten Verbreitung desselben. Die Folge waren hunderte abgestürzte Rechner. Mit Hilfe eines Kontrollprogramms war es allerdings möglich, die weitere Ausbreitung des Wurmes zu stoppen.

Der bisher bekannteste Wurm war der sog. „Internet-Wurm" [Spa88, Spa91]. Dieser, von einem amerikanischen Studenten in Umlauf gebrachte Wurm,[4] begann seine Reise

[4] Bis heute ist ungeklärt geblieben, ob die Verbreitung des Wurms in das Internet geplant war, oder ob

durch das Internet am 2. November 1988. Obwohl dieser Wurm mit keiner Schadensfunktion ausgestattet war,[5] setzte er, aufgrund eines logischen Fehlers im Programm, innerhalb kürzester Zeit eine Vielzahl von Servern im Internet dadurch lahm, daß er sich unzählige Male replizierte und durch den damit verbundenen übermäßigen Konsum an System-Ressourcen die Rechensysteme unnutzbar machte. Ein eindrucksvolles Beispiel dafür, welche Gefahren von Denial-of-Service-Angriffen ausgehen.[6]

Die Erfahrungen, die mit dem Internet-Wurm gemacht wurden, sollten in die Entwicklung heutiger Agenten-Systeme einfließen. Würmer weisen alle Merkmale auf, die auch heutige mobile Agenten besitzen.

4.4. Bedrohungen

Agenten operieren nur sinnvoll in offenen Umgebungen. Daraus resultiert, daß sie insbesondere den Gefahren der Netzangriffe unterliegen. Die Offenheit der Kommunikationskanäle ermöglicht eine Vielzahl von Angriffspunkten. Gleiches gilt für die Umgebung.

4.4.1. Datengesteuerte Angriffe

Softwarebomben, Trojanische Pferde und Viren stehen für Programme, die maliziöse Operationen durchführen. Die Programme werden entsprechend als maliziöse Software kategorisiert. Daraus ist nicht der Schluß zu ziehen, daß die Kategorie der gutartigen Software keine Schäden in IT-Systemen verursachen kann. Angreifer können gutartige Software mit Eingaben beliefern, die zu unerwünschten Ergebnissen oder einen Prozeß in einen inkonsisten Zustand führen. Datengesteuerte Angriffe treten seit 1996 vermehrt auf. I. d. R. werden hierbei Programme angegriffen, die bei entsprechenden Daten einen Pufferüberlauf erfahren. Auch hier tritt `sendmail` erneut als unrühmliches Beispiel für Programme auf, die eine Vielzahl von Angriffspunkten für datengesteuerte Angriffe lieferten.

Nicht immer führen unangemessene Eingaben dazu, daß beliebige Instruktionen ausgeführt werden können. Nahezu immer ist jedoch die Möglichkeit gegeben, daß nicht berücksichtigte Eingaben zu unerwarteten Änderungen im Kontrollfluß eines Prozesses führen. Datengesteuerte Angriffe können somit insbesondere dazu genutzt werden, um Denial-of-Service-Angriffe durchzuführen.

Datengesteuerte Angriffe finden dann ausreichend Angriffspunkte, wenn Programme unter Mißachtung von Erkenntnissen der Softwaretechnik entwickelt werden. Programme, die sich auf diese Weise angreifen lassen, entsprechen nicht der Spezifikation und sind somit nicht korrekt. Für Agentensysteme werden datengesteuerte Angriffe wahrscheinlich eine häufig anzutreffende Angriffsart werden, da das Testen von Agentensystemen

es sich um einen „Betriebsunfall“ handelte.

[5] Alle Voraussetzungen für eine derartige Erweiterung waren allerdings gegeben.

[6] Näheres zu diesen Angriffen erfolgt im Abschnitt 4.4.3.

sehr aufwendig ist. Hinzu kommt, daß der Entwicklungszyklus von Techniken, die aktuell einen großen Schub erleben, sehr kurz ist. Eine klares softwaretechnisches Design mit anschließender Validierung und Verifikation wird leider nur selten durchgeführt. Die daraus folgenden Konsequenzen zeigen sich sehr deutlich in den Sicherheitsproblemen der Basistechnologien von Agentensystemen.[7]

Jedes Programm kann aufgrund bestimmter Eingaben Fehlverhalten produzieren, so nicht alle Eingaben auf ihre Richtigkeit und Angemessenheit überprüft werden. Der Internet-Wurm aus dem Jahre 1988 machte sich erstmals den Umstand zu nutze, einen internen Pufferüberlauf innerhalb eines Programms zu produzieren, um so die Gelegenheit zu erhalten, überschüssige Daten als Instruktionen interpretieren zu lassen. Datengesteuerte Angriffe können dazu führen, daß das angegriffene Programm veranlaßt wird, beliebige Instruktionen auszuführen [Sib96].

Kein Programm ist vollständig fehlerfrei, so seine Korrektheit nicht bewiesen ist. Hieraus ergibt sich die potentielle Gefahr, daß nahezu jedes Programm mittels eines datengesteuerten Angriffes kompromittiert werden kann.

Die Analyse, die für die Durchführung datengesteuerter Angriffe erforderlich ist, verlangt vom Angreifer ein hohes Verständnis für die Strukturen ausführbarer Programme, deren Quelltext nicht vorliegt. Mit dem Wandel von Programmen, die in Maschinencode vorliegen, zu jenen, die durch eine abstrakte Maschine interpretiert werden, hat sich die Analyse stark vereinfacht. Für eine Sprache wie Java existieren eine Vielzahl von Decompilern,[8] die es ermöglichen, aus Bytecode den Quelltext wiederherzustellen. Es gilt jedoch zu beachten, daß Java eine robuste Sprache ist, in der Pufferüberläufe laut der Spezifikation der Sprache nicht möglich sein sollen.

Datengesteuerte Angriffe können die Sicherheitskriterien Integrität, Originalität, Verbindlichkeit, Verfügbarkeit und Vertraulichkeit verletzen.

4.4.2. Datenmanipulationen

Daten sind eines der wichtigsten Güter in der heutigen Gesellschaft. Eine wesentliche Bedrohung stellt somit die Manipulation dieses Wirtschaftsguts dar. Datenmanipulationen können bei deren Erfassung, Be- oder Verarbeitung erfolgen. Manipulationen können temporär oder permanent erfolgen.

Eine besondere Gefahr stellen Veränderungen dar, die nicht offensichtlich erfolgen und erst nach längerer Zeit entdeckt werden. Derartige Manipulationen werden als schleichende Veränderungen (engl. *data diddling*) bezeichnet. Datenmanipulationen stellen insbesondere dann eine Gefahr dar, wenn die durch die Manipulationen ausgelösten Folgen nicht rückgängig gemacht werden können. So zum Beispiel in einer Transaktion, in der eine Überweisung manipuliert und der Betrag anschließend sofort abgehoben wird.

[7] Siehe hierzu Kapitel 5.

[8] Ein Beispiel für einen solchen Decompiler ist der auf Mocha aufbauende Decompiler Jasmine (http://members.tripod.com/~SourceTec/jasmine.htm).

Datenmanipulationen können von allen beteiligten Subjekten in Agentensystemen vorgenommen werden, wobei hierbei das Risiko für Agenten besonders groß ist, da sie unter der Kontrolle eines fremden Umgebungssystems ausgeführt werden.

Die Manipulation von Daten zieht eine Verletzung der Sicherheitskriterien Integrität und Originalität nach sich.

4.4.3. Denial-of-Service (Diensteverweigerung)

Denial-of-Service-Angriffe machen sich zu nutze, daß in IT-Systemen gemeinsame Ressourcen angeboten werden. Durch unangemessene Nutzung der gemeinsamen Ressourcen durch eine Entität werden diese den anderen Teilnehmern entzogen. Im einfachsten Fall, wie den SYN-Flooding[9]-Angriffen, werden Dienste einfach mit Daten *überflutet*; mit der Folge, daß sie von den rechtmäßigen Nutzern nicht mehr genutzt werden können. Intelligentere Angriffe unterbinden Dienste, leiten diese um und ersetzen sie durch andere [CZ95].

Es ist davon auszugehen, daß Denial-of-Service-Angriffe nicht vollständig vermeidbar sein werden. Das wesentliche Problem besteht darin, daß eine übermäßige Nutzung nur schwerlich durch das Rechensystem als legal oder illegal eingestuft werden kann. Eine hohe Last kann auf einem System durch die legitime Berechnung von Fibonacci-Zahlen erzeugt werden. Gleiches kann auch von einem Angreifer durchgeführt werden. Die Aufstellung von typischen Anwendungsprofilen wie finanztechnische, mathematische oder multimediale Anwendungen kann eine mögliche Lösung darstellen. Den Anwendungsprofilen sind typische Werte für den Ressourcenverbrauch zuzuordnen, die von den Instanzen der Anwendungsprofile nicht überschritten werden dürfen. Ein Angriff auf *einen* Dienst darf nicht zur Folge haben, daß die weiteren angebotenen Dienste in Mitleidenschaft gezogen werden.

Agentensysteme könnten zukünftig besonders von diesen Angriffen betroffen sein, da in der Konzeption der Basiskomponente Java die Betrachtung dieser Bedrohung bewußt nicht einbezogen wurde [GS97].

In Agentensystemen können Denial-of-Service-Angriffe durch Agenten dazu genutzt werden, andere Agenten an der Ausführung zu hindern oder die Dienste eines Umgebungssystems für die Nutzung durch andere Agenten zu blockieren. Desweiteren besteht die Gefahr, daß Agenten durch andere Agenten oder durch das Umgebungssystem terminiert werden.

Denial-of-Service-Angriffe verletzen das Sicherheitskriterium der Verfügbarkeit und möglicherweise auch das der Integrität, wenn es zu einem Zusammenbruch des Rechensystems kommt, ohne daß alle Daten persistiert wurden.

[9] Ein SYN-Flooding-Angriff bringt die interne Tabellen im TCP/IP-Stack zum Überlauf. Damit können keine weiteren Verbindungen vom System, gegen das der Angriff gerichtet ist, entgegengenommen werden. Siehe hierzu auch [SKK+96].

4.4.4. Hijacking (Entführung)

Das neue Softwareparadigma des Agenten führt eine neue Bedrohung in Rechensysteme ein, die es in dieser Form bislang nicht gab. Mobile Agenten bewegen sich innerhalb von Netzwerken und unterliegen dabei der Bedrohung auf einem dieser Systeme festgehalten zu werden. Die Ausführung von Agenten erfolgt vollständig unter der Kontrolle des Umgebungssystems, so daß eine Bedrohung in Form von Entführung für Agenten äußerst real ist. Es sei darauf hingewiesen, daß das Fallbeispiel *Big-Brother* in Anhang A.1 eine derartige Bedrohung implementiert und gleichzeitig aufzeigt, daß diese Gefahr nicht nur von dem Umgebungssystem, sondern auch von anderen Agenten ausgeht.

Sobald Agenten Daten mit sich führen, die für den Besitzer des Agenten von Bedeutung sind, stellt die Bedrohung der Entführung eine nicht zu unterschätzende Gefahr dar. Es sei darauf verwiesen, daß die Bedrohung der Entführung als besondere Form eines Denial-of-Service-Angriffes betrachtet werden kann, da der Agent weiterhin existiert, ohne daß seine Integrität verletzt wurde, er für seinen Besitzer jedoch nicht mehr verfügbar ist.

4.4.5. Maskerade

Mit Maskerade (engl. *masquerading* oder *spoofing*) wird das Vortäuschen einer Identität bezeichnet. In Agentensystemen, wie auch in anderen IT-Systemen, muß gewährleistet sein, daß jeder Teilnehmer eine eindeutige Identität besitzt und die Zuordnung einer Identität zu einem Subjekt in einem Authentifikationsprozeß bewiesen werden kann.

Gelingt es einem Subjekt, dem Authentifikationsprozeß eine andere Identität als die eigene vorzutäuschen, agiert er nachfolgend unter der vorgetäuschten Identität. Die Folge ist, daß die Sicherheitskriterien, wie sie im Abschnitt 2.3 definiert wurden, nicht erfüllt werden können.

4.4.6. Piggybacking

Unter Piggybacking versteht man, daß ein Täter sich an eine andere Person hängt. In der Realwelt kann dieses in dem Folgen einer autorisierten Person in einen Hochsicherheitstrakt bedeuten, einschließlich der Passierung von Sicherheitsschleusen. Auf Agenten übertragen passiert ein Agent die Authentikationsschranke eines Systems, indem er sich an einen anderen Agenten anhängt. Dieses kann bereits auf dem System erfolgen, auf dem der Opferagent erzeugt wurde. Auf diese Weise kann der Wirtsagent signiert werden, während er gleichzeitig den maliziösen Agenten enthält. Passiert der Wirtsagent die Authentikationsschranke, gelangt er zur Ausführung und „brütet“ zunächst den maliziösen Agenten aus, der den Wirtsagent dahingehend manipuliert hat.

4.4.7. Replaying (Wiedereinspielen)

Eine weitere Angriffsmethode, die dem Bereich der Kommunikationssicherheit zuzurechnen ist, ist das Wiedereinspielen von Nachrichten bzw. Nachrichtenpaketen. Ein Wiedereinspielen kann zur Folge haben, daß Transaktionen erneut veranlaßt oder kostenpflichtige Dienst wiederholt genutzt werden. Das Wiedereinspielen von Nachrichten ist eine Bedrohung, die auch in Agentensystemen existiert. Mit der Entwicklung der Active Networks können Agenten in Vermittlungsrechner geladen werden, um dort Nachrichten zu empfangen und bei Bedarf wiedereinzuspielen. Gleiches gilt für Nachrichten, die in Umgebungssystemen empfangen bzw. versandt werden. Umgebungssysteme können ohne weiteres Nachrichten erneut versenden. Agenten können dies, wenn sie auf unautorisierte Weise in den Besitz der Nachrichten gelangen.

4.4.8. Salami Angriffe

Die Vielzahl der zu verarbeitenden Informationen macht es in komplexen Prozessen nahezu unmöglich, jeden Vorgang zu kontrollieren. Entsprechend besteht die Gefahr, daß geringfügige Manipulationen in komplexen Prozessen nicht auffallen. Ein Aspekt, den sich Angreifer mit dem sog. Salami-Angriff zu nutze machen, in dem sie beispielsweise in Finanztransaktionen Rundungsdifferenzen auf eigene Konten umleiten.

Aufgrund der Endlichkeit von Rechensystemen sind Ungenauigkeiten eine inhärente Systemeigenschaft von IT-Systemen. Dementsprechend fallen Rundungsdifferenzen nicht sofort auf, da sie als selbstverständlich erscheinen.

4.4.9. Sniffing (Schnüffeln)

Manipulierende bzw. störende Angriffe lassen dem Opfer häufig deutlich werden, daß ein Angriff stattfindet. Im Falle des passiven Mitlesens (engl. *sniffing*) von Informationen, werden keine zustandsverändernden Aktionen an den Daten vorgenommen, so daß diese Form eines Angriffs nur schwer zu entdecken ist. Das Mitlesen der versandten Daten ermöglicht ein späteres Wiedereinspielen, bei dem einmal zur Authentisierung benutzte Daten, wie z. B. verschlüsselte Paßwörter, von einem Angreifer im Zuge eines späteren Zugangsversuch wieder eingespielt werden können.

Auch wenn die von einem Agenten „erschnüffelten“ Daten nicht außerhalb des Umgebungssystems transportiert werden können, ist es häufig ausreichend, daß er Zugriff auf die sensitiven Daten erhält. Im anderen Fall, indem die Informationen aus dem Umgebungssystem transportiert werden müssen, kann der Weg einer direkten Netzwerkverbindung eingeschlagen werden. Auch die Sicherheitspolitik der JVM, die die Möglichkeiten der Java-Agenten stark einschränkt, erlaubt zumindest eine Netzwerkverbindung zum Quellsystem des Agenten. Ist auch diese nicht möglich, können verdeckte Kanäle genutzt werden, um die Informationen aus einem Umgebungssystem zu transportieren. Bellovin hat bereits 1995 gezeigt, daß ein solcher Kanal beispielsweise über DNS-Anfragen

eröffnet werden kann [Bel95].

Das Umgebungssystem hat keine Probleme, erlauschte Informationen weiterzuleiten, da es dieses unter eigener Kontrolle tut und dabei keinen Einschränkungen unterliegt. Alle Daten der Agenten liegen i. d. R. in Klartextform vor.

Dekompilierung ist in Agentensystemen eine besondere Form des Schnüffelns. Es ermöglicht Angreifern Einblick in den Quelltext von Agenten zu erhalten. Damit werden zum einen die Intellectual Properties des Agentenprogrammierers verletzt. Zum anderen können Agenten so leicht auf Schwachstellen im Programmcode analysiert werden. Im letzteren Fall entfällt für datengesteuerte Angriffe die aufwendige Analyse des Maschinencodes, die für die Bestimmung der maliziösen Eingabedaten erforderlich ist.

Eine besondere Form der Datenspionage ist *Scavening*. Hier werden ausschließlich Daten temporärer Natur ausspioniert, die über längere Zeit im Haupt- und Hintergrundspeicher verbleiben. Dazu gehören Daten in Warteschlangen, Puffern und temporären Verzeichnissen. Häufig handelt es sich um temporäre Informationen, die nach Beendigung einer Verarbeitung übrig geblieben sind bzw. nur unzureichend entfernt wurden.

4.4.10. Verdeckte Kanäle

Verdeckte Kanäle (engl. *Covert Channels*) erlauben einem Programm, auf verdecktem Wege Informationen zu übertragen. Es werden zwei Arten von verdeckten Kanälen unterschieden: Speicherkanäle und Zeitkanäle. Von einem Speicherkanal wird dann gesprochen, wenn es einem Prozeß gelingt zu beobachten, wie von einem anderen Prozeß Objekte gespeichert werden. Verdeckte Informationsflüsse können bei Speicherkanälen über die Objektattribute (Dateinamen, Dateiattribute) oder der Objektexistenz (Dateiexistenz) fließen. Speicherkanäle können durch die Rate, in der Objekte manipuliert werden, entdeckt werden. Zeitkanäle existieren, wenn ein Prozeß aufeinanderfolgende Ereignisse beobachten und die Zeit zwischen diesen messen kann. Die Messung setzt voraus, daß der Prozeß Zugriff auf einen Zeitgeber hat, wie z. B. eine Echtzeituhr. Aktionen anderer Prozesse führen zu Rauschen auf dem Zeitkanal. Entsprechend lassen sich nur geringe Datenraten über einen Zeitkanal übertragen. Die Übertragungsrate kann allerdings ausreichen, um kurze Informationen, wie z. B. geheime Schlüssel, zu übertragen.

Die Beschneidung der Bandweite zur Vermeidung verdeckter Kanäle ist nicht ausreichend. Auch kleine Bandbreiten sind kein Schutz, da es wichtige Informationen gibt, wie das Beispiel der Schlüssel zeigt, die nur wenige Byte umfassen.

Allgemein gilt es zu beachten, daß je mehr Dinge ein Programm beobachten kann, desto mehr Möglichkeiten sind vorhanden, vertrauliche Informationen nach außen lecken zu lassen.

Verdeckte Kanäle verletzen die Sicherheitskriterien Anonymität, Originalität und Vertraulichkeit.

4.4.11. Verkehrsflußanalyse

Mit der Belauschung wurde bereits eine Bedrohung aufgezeigt, die sich gegen die Vertraulichkeit der Daten richtet. Eine Verkehrsflußanalyse ist ebenfalls eine Bedrohung, die sich gegen die Vertraulichkeit von Daten richtet. Objekt der Bedrohung sind in diesem Fall jedoch nicht die Nutzdaten, wie sie in Rechensystemen und Kommunikationskanälen existieren. Bedrohungsgegenstand sind Verkehrsdaten bzw. Protokolldaten, wie sie beispielsweise in den Kopfinformationen von Paketen enthalten sind, wenn diese über ein paketvermittelndes Netz übertragen werden.

Aus den Protokolldaten lassen sich die Kommunikationsdaten ermitteln. Ein Angreifer wird durch die Informationen in die Lage versetzt, zu erkennen, zwischen welchen Subjekten Kommunikation stattfindet, zu welchem Zeitpunkt und in welchen Abständen dies geschieht und wie groß die Menge der ausgetauschten Informationen ist. Ohne daß dabei die Vertraulichkeit der Nutzinformationen verletzt wird, reichen die ermittelten Daten aus, um einen für den Angreifer erforderlichen Informationsgewinn zu erzielen

Verkehrsflußanalyse erfolgt sinnvollerweise dort, wo viele Pakete ein Rechensystem durchlaufen und weitergeleitet werden. Ein Profil, das auf die in Netzwerken enthaltenen Vermittlungsrechner zutrifft. In Abschnitt 3.4.1 wurde ein Agentensystem vorgestellt, das genau derartige Rechner als Umgebungssystem nutzt und entsprechend potentiell über die Mittel verfügt, eine Verkehrsflußanalyse durchzuführen. Desweiteren sind die Vermittlungsrechner selbst in der Lage, ausgiebige Verkehrsflußanalysen durchzuführen.

4.5. Verwundbarkeiten

Im Kapitel 2 wurden Verwundbarkeiten als inhärente Systemschwächen definiert, die mittels Bedrohungen ausgenutzt werden können. Inhärente Systemschwächen sind vor allem dort vorhanden, wo Implementationsmängel über die Auslieferung des Produkts hinausgehend bestehen bleiben.

Die Entwicklung des Internets erfolgt seit 1994 – Auslöser war die Entwicklung des WWW – in einer exponentiellen Geschwindigkeit. Neben der stetig steigenden Anzahl von angeschlossenen Rechensystemen und den damit unüberschaubar werdenden Netzen, werden nahezu täglich neue Produkte vorgestellt und neue Techniken entwickelt. Dieses wird nicht zuletzt deutlich an der rasanten Entwicklung der Agentensysteme und deren Basiskomponenten.

Eine Entwicklung von qualitativ hochwertigen Softwareprodukten nach den Regeln der Softwaretechnik erfordert nicht zuletzt Zeit, damit der Entwicklungszyklus vollständig durchlaufen werden kann [Som92]. Das große öffentliche Interesse an den Basiskomponenten von Agentensystemen hat zur Folge, daß viele Implementationsmängel entdeckt und behoben wurden.[10] Es ließe sich der Schluß ziehen, daß dieses als positive Entwicklung zu bewerten sei. Dabei würde allerdings übersehen werden, daß die Menge der

[10] Siehe hierzu u. a. [GS97, DW95, Kok97].

Fehler offenbart, daß die Regeln der Softwareentwicklung nicht ausreichend Beachtung finden. Der Entwicklungsprozeß eines Softwareprodukts ist auch für die Systemsicherheit von entscheidender Bedeutung [Pfl89].

Implementationsmängel stellen insbesondere den Sicherheitsaspekt der Funktionalität und der Korrektheit in Frage. Mangelhafte Implementationen können Systemzustände zur Folge haben, die die Konsistenz des gesamten Rechensystems bedrohen. In derartigen Fällen kann das Rechensystem möglicherweise zum Zusammenbruch geführt werden, obwohl dieses auf der höheren Ausführungsebene der Agenten – wie z.B. im Falle der Ausführung von Agenten in der JVM – nicht möglich sein sollte. Desweiteren stellen Implementationsmängel nur zu häufig das Eingangstor für die in den obigen Abschnitten aufgeführten Bedrohungen dar. Eine fehlerhafte Implementation und die damit verbundene Beschränkung der Funktionalität kann unkalkulierbare Kosten nach sich ziehen. Zu beachten ist vor allem, daß die direkteste Art eines maliziösen Angriffs auf ein Rechensystem die Ausnutzung von Implementationsfehlern ist.

4.6. Zusammenfassung

Es zeigt sich, daß in Agentensystemen Bedrohungen insbesondere entsprechend der drei Dimensionen des Sicherheitsbegriffes wirken. Die Aufstellung der Bedrohungen für Agentensysteme zeigt, daß ein weites Spektrum von Gefahren für Agentensysteme besteht. Die Bedrohungen richten sich gegen alle Objekte eines Systems. Eine grafische Zusammenfassung der Situation läßt sich wie in Abbildung 4.1 darstellen.

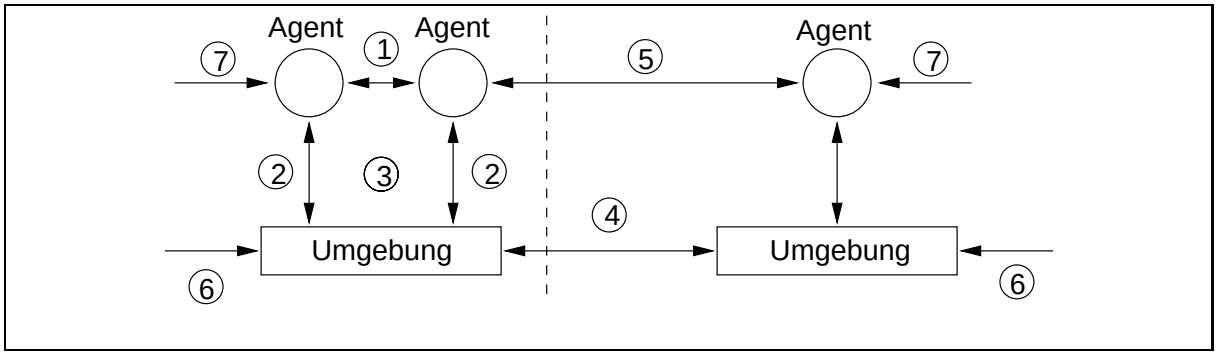

Abbildung 4.1.: In Agentensysteme existieren vielfältige Bedrohungen: 1. Angriffe von Agenten gegen andere Agenten, 2. Angriffe von Agenten gegen Umgebungen, 3. Angriffe von Umgebungen gegen Agenten, 4. Angriffe von Umgebungen gegen andere Umgebungen, 5. Angriffe von Agenten gegen Agenten in anderen Umgebungen, 6. Angriffe von Dritte gegen Umgebungen und 7, Angriffe von Dritte gegen Agenten.

Dies ist eine Darstellung, wie sie ähnlich auch von HOHL gewählt wird. In Hinblick auf die besonderen Gefahren, die aus dem Agenten-Paradigma erwachsen und in Reflektion des Sicherheitsbegriffes, wie er in der Arbeit verwendet wird, lassen sich die Gefahren für

Agentensysteme zu einer Darstellung komprimieren, wie sie in Abbildung 4.2 zu sehen ist.

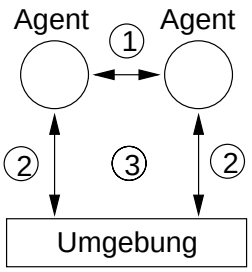

Abbildung 4.2.: Die mit Agentensystemen verbundenen Gefahren resultieren aus den besonderen Bedrohungen, die in diesen Systemen bestehen: 1. Agenten bedrohen andere Agenten, 2. Agenten bedrohen Umgebungen und 3. Umgebungen bedrohen Agenten.

5. Sicherheitsbeurteilung der Basiskomponenten von Agentensystemen

„*The goal of an attacker is to be able to run a program of his choice on your computer without your knowledge.*"
– Garfinkel & Spafford

Das vorhergehende Kapitel hat gezeigt, daß eine Vielzahl von Bedrohungen existieren, die genutzt werden können, um die Objekte innerhalb eines Agentensystems anzugreifen. Es bedarf geeigneter Sicherheitsmechanismen, die den Bedrohungen erfolgreich begegnen und damit einhergehend den in Abschnitt 2.3 aufgestellten Sicherheitskriterien gerecht werden.

Nötig ist der Entwurf und die Umsetzung einer Sicherheitsarchitektur, die den drei Dimensionen des Sicherheitsbegriffes gerecht wird. Agentensysteme sind komplexe Softwaresysteme, deren Gefahrenpotential aufgrund der vielen Angriffspunkte sehr groß ist. Die Ausführung von unbekannten Prozessen ist die größte Bedrohung, die einem Rechensystem widerfahren kann. Dabei ist es nahezu ausgeschlossen, vor der Ausführung eines Programmes festzustellen, welchem Zwecke es dient. Garfinkel und Spafford weisen daraufhin, daß selbst nach der Ausführung eines Programmes häufig nicht identifiziert werden kann, welche Aktionen durchgeführt worden sind.

Die Komplexität von Agentensystemen resultiert insbesondere aus der Möglichkeit, Agenten zwischen unterschiedlichen Umgebungen migrieren zu lassen. Lassen sich die Sicherheitsprobleme bei einfacher Migration, d. h. der einmaligen Migration vom Server zum Client, in den Griff bekommen, so steigt die Komplexität der Sicherheitsanforderungen bei mehrfacher Migration um ein Vielfaches. Moderne Betriebssysteme wie AIX, MacOS oder Windows NT bieten keinen Schutz auf Betriebssystemebene vor Agenten, die aus dem Internet bzw. einem beliebigem Netzwerk bezogen werden. Agenten erben die Rechte des Benutzers, unter dessen Kontrolle sie zum Einsatz kommen. Auf Betriebssystemebene gibt es kein der Sandbox aus Java entsprechendes Konzept.

Es wird deutlich, daß Sicherheitskonzepte erforderlich sind, die dem Paradigmenwechsel, der mit der Agententechnologie einhergeht, Rechnung tragen. Alle Agentensysteme

nutzen Basiskomponenten, und ihre Sicherheitsmechanismen beruhen im wesentlichen auf den dort enthaltenen Techniken. Aus diesem Grund werden in den folgenden Abschnitten die Komponenten Internet, Browser und Java auf die Eignung ihrer Sicherheitsmechanismen für Agentensysteme untersucht. Die drei Komponenten sind wesentliche Bestandteile nahezu jedes Agentensystems. Das Internet stellt das Transportmedium für Agenten dar, Browser dienen als Wirtssysteme, in denen mit der JVM eine Ausführungsumgebung zur Exekution der Agenten bereit steht.

Die Beurteilung der Sicherheitskonzepte der unten aufgeführten Agentensysteme mangelt daran, daß die angekündigten Sicherheitsmechanismen sich in der Mehrheit der Fälle noch in der Entwicklungsphase befinden. Sind die Entwicklungen abgeschlossen, fehlt es häufig an ausreichender Dokumentation. Mit der unzureichenden Dokumentation geht einher, daß die Nutzung der Sicherheitskomponenten nur sehr selten erfolgt und eine Beurteilung der Komponenten nur eingeschränkt möglich ist.

Viele Fehler, die in der Vergangenheit in Bezug auf die Sicherheit von Rechensystemen begangen wurden, wiederholen sich heute. Entwickler verzichten – zum Teil gezwungen durch den Marktdruck – auf die Studie existierender, langeingeführter Systeme. Heute scheint der wichtigste Schritt bei der Entwicklung eines Softwareproduktes dessen schnelle Markteinführung zu sein. In Anbetracht der Entwicklung der bisherigen Internettechnologien erscheint es nahezu unausweichlich, daß die Einführung der Agententechnologien mit ähnlichen, altbekannten Sicherheitsproblemen zu kämpfen haben wird.

Augrund der zu erwartenden Neuerungen, die mit der Verbreitung von Agenten-Technologien Einzug halten werden, ist darüberhinaus davon auszugehen, daß weitere, bisher nicht aufgetretene Sicherheitsprobleme auftauchen werden.

5.1. Internet

Die Entwicklung des Internets fand zunächst im wesentlichen im wissenschaftlichen Umfeld statt. Hier wurde dem freien Informationsaustausch der Vorrang vor allen anderen Belangen gegeben. Mit der Entwicklung des Netzes vom wissenschaftlichen Informationsnetz hin zum kommerziellen Multimedianetz treten Probleme auf, die in der Historie und dem ehemaligem Auftrag des Netzes begründet liegen.

Spätestens mit der zunehmenden Kommerzialisierung und der damit einhergehenden Nutzung von Internet-Technologien in Intranets wurde deutlich, daß die Internet-Technologien den veränderten Sicherheitsansprüchen nicht gerecht werden können. Es zeigt sich heute deutlich, daß Sicherheitsaspekte bei der Entwicklung des Internets eine untergeordnete Rolle spielten. Sicherheit und Internet-Technologien ist ein komplexes Thema, dem diese Arbeit mit einem Abschnitt nicht gerecht werden kann. Die folgenden Ausführungen beschränken sich aus diesem Grund auf die wesentlichen Punkte. Für weitergehende Analysen sei auf die entsprechende Literatur verwiesen.

5.1.1. IPv4

Derzeit befindet sich weltweit die 1981 spezifizierte Version 4 des Internet Protocols (IPv4) im Einsatz. Das Protokoll wurde für eine verbindungslose, unzuverlässige Paketvermittlung entwickelt. Es bietet keine integrierten Sicherheitsmechanismen, TCP/IP-Pakete werden in unverschlüsselter Form übertragen.

Das einzig optionale Sicherheitsmerkmal des IP-Protokolls sind Sicherheitsmarken mit denen eine Klassifikation der Nachrichtenpakete vorgenommen werden kann [Ken91]. Die Sicherheitsmarken können lediglich als Hilfsmittel für die Zugriffskontrolle in Multilevel-Security Systemen dienen. Entsprechend selten kommen die Sicherheitsmarken zum Einsatz [Fle97].

Zur Familie der Internet-Protokolle gehören weiterhin DNS, ICMP, TCP und UDP. Auch für diese Protokolle gilt das bereits für das Internet-Protokoll gesagte, es existieren keine integrierten Sicherheitsmechanismen.

DNS bildet Rechnernamen auf IP-Adressen und umgekehrt ab. Der weltweite Namensraum ist in einer verteilten, baumförmigen Datenbank implementiert, wobei Teilbäume von DNS-Servern verwaltet werden. Für die Abbildung der Namen in beide Richtungen werden zwei unabhängige Bäume genutzt, zwischen denen kein semantischer Zusammenhang besteht. ICMP dient der Kommunikation zwischen Rechnern und Gateways. Dabei werden sie weitestgehend eingesetzt, um die Netzwerkverbindungen zwischen den angeschlossenen Systemen zu testen. TCP baut auf IP auf und gleicht dessen Unzuverlässigkeit aus. Es sorgt für eine geordnete, zuverlässige bidirektionale Kommunikation zwischen Kommunikationspartnern, die sich auf einem oder auf verschiedenen Rechensystemen befinden können. UDP ist ebenso wie TCP ein Protokoll der Transportschicht, gewährleistet allerdings im Gegensatz zu TCP keine zuverlässige Übertragung. Vorteil des Protokolls ist der geringere Overhead.

5.1.2. IPv6

Derzeit befindet sich die neue Version des Internet-Protokolls IPv6 in einem Revisionsprozeß. Zukünftig wird es die aktuelle Version des IPs ablösen. Ein wesentliches Problem, das sich mit der rasenden Entwicklung des Internets ergab, ist der zunehmende Bedarf an Adressen. Neben der Lösung dieses Problems ist ein weiterer Aspekt des neuen Protokolls die Bereitstellung von Sicherheitsmechanismen, die den Bedürfnissen der veränderten Anwendungen, die über das Netz genutzt werden, gerecht werden.

Zur Realisierung der neuen Sicherheitsmechanismen werden mit IPv6 Authentifizierungs- und Chiffrierungs-Header eingeführt. Authentikationsheader (AH) dienen der Wahrung der Authentizität und der Integrität. Wird ein asymmetrisches Signaturverfahren verwendet, kann die Unwiderrufbarkeit gewährleistet werden. Desweiteren ist ein Schutz vor Wiedereinspielungsangriffen gegeben [Fle97, Atk95]. Mit AH werden alle statischen Daten des Nachrichtenpaketes geschützt. Eine Chiffrierung der Nutzdaten inklusive der Header der höheren Protokollschichten erfolgt mittels des Chiffrierungs-

Headers. Das Ergebnis der Chiffrierung wird als Encapsulating Security Payload (ESP) bezeichnet. Die Verwendung der Chiffrierung gewährleistet auf IP-Ebene Vertraulichkeit und Integrität. Ein Schutz vor Wiedereinspielungsangriffen ist ebenfalls gegeben. Die Chiffrierung kann in zwei unterschiedlichen Modi betrieben werden. Im Transport-Modus werden lediglich die Nutzinformationen zusammen mit den Headern der höheren Protokollschichten verschlüsselt, während im Tunnel-Modus das gesamte Datagramm verschlüsselt wird.

Bestandteil der Sicherheitsarchitektur des neuen IP-Protokolls sind Sicherheitsgateways, die zwischen externen, nicht vertrauenswürdigen Rechnern und internen, vertrauenswürdigen Rechnern vermitteln. Das Gateway übernimmt für die internen Rechner die Bearbeitung der Sicherungsheader. Gateways erlauben den Aufbau sicherer Tunnel zwischen Kommunikationspartnern, die über das Internet miteinander verbunden sind.

IPv6 abstrahiert von den konkreten Mechanismen, die zur Umsetzung der Sicherheitsmerkmale erforderlich sind, durch den Begriff der Sicherheitsassoziation. Eine Sicherheitsassoziation ist eine Beziehung zwischen Kommunikationsteilnehmern. Sie beschreibt die Nutzung von Sicherheitsdiensten, die beim Verbindungsaufbau zwischen den Teilnehmern durch eine auszuhandelnde Parametermenge definiert werden [Fle97]. Die eine Kommunikationssituation beschreibenden Parameter sind u. a. der Authentifizierungsalgorithmus mitsamt Schlüssel, der Chiffrierungsalgorithmus mitsamt Schlüssel, die Lebensdauer der Assoziation und der Sensitivitätsgrad der Daten. Sicherheitsassoziationen sind unidirektional, zur Sicherung des Kommunikationsflusses in beide Richtung ist somit der Aufbau von zwei Sicherheitsassoziationen erforderlich. Die Etablierung einer Sicherheitsassoziation setzt zwingend eine Authentifizierung und einen Schlüsselaustausch voraus. Die Authentifizierung kann auf Benutzer- und auf Maschinenebene erfolgen. Im Falle von Mehrbenutzersystemen ist die Authentifizierung auf Benutzerebene unerläßlich.

Die anderen Protokolle sind mit Ausnahme von DNS von keinen Änderungen betroffen. Sie profitieren allerdings von den Sicherheitsmechanismen, die durch IP zur Verfügung gestellt werden. DNS-Server sind zukünftig in der Lage, ihre Nachrichten digital zu signieren, so daß der Client die Authenzität der Nachricht verifizieren kann. Durch die verteilte Struktur des DNS ist bereits implizit eine Infrastruktur für öffentliche Schlüssel vorhanden. So lassen sich die erforderlichen Schlüssel zur Verifikation von digitalen Signaturen ebenfalls über das DNS verfügbar machen. Die Antwort des Servers umfaßt zusätzlich die Anfrage des Clients, so daß sichergestellt ist, daß die Antwort mit der intendierten Anfrage korrespondiert.

5.1.3. World-Wide Web

Die wesentliche Anwendung des Internets bildet seit Mitte der 90er Jahre das World-Wide Web. Das WWW wurde entwickelt, um Dokumente öffentlich zugänglich zu machen, die über sog. Hyperlinks miteinander verknüpft sind. Die Dokumente werden auf Web-Servern zur Verfügung gestellt. Der Zugriff der Clients auf die Dokumente erfolgt

i. d. R. durch einen Browser, deren Sicherheitsmerkmale im Abschnitt 5.2 untersucht werden.

Web-Server sind dahingehend entworfen, daß sie anonyme Anfragen von nicht authentisierten Clients erhalten und die Antworten in einer schnellen und effizienten Weise liefern [GS96]. Die Entwicklung der Web-Server ging davon aus, daß auf diesen lediglich öffentliche Dokumente abgelegt werden würden. Sensitive Dokumente sollten nicht auf den Systemen plaziert werden. In zunehmenden Maße werden über Web-Server Dienste angeboten, die eine Authentifizierung der Clients erforderlich werden läßt. Entsprechend sind Web-Server mittlerweile mit Verfahren versehen, die eine Zugriffskontrolle auf Benutzer- oder Maschinenebene erlauben.

Abhängig vom Sensitivitätsgrad der bereitliegenden Dokumente werden eine Reihe von Zugriffskontrollmechanismen durch den Web-Server angeboten:

- ▷ Versteckte URLs,
- ▷ Maschinenbasierte Einschränkungen und
- ▷ Identitätsbasierende Zugriffskontrollen [GS97].

Die durch den Web-Server angebotene Funktionalität kann durch CGI-Programme oder Module erweitert werden. CGI-Programme sind Subprozesse des Servers und unterliegen nicht den Sicherheitseinschränkungen, die durch den WWW-Server durchgesetzt werden. Module sind Erweiterungen des Servers, die eine vom Server definierte API implementieren.

Für eine Reihe der Web-Server ist der Quelltext frei zugänglich. Der derzeit verbreiteste Server Apache[1] verdankt seine weite Verbreitung u. a. diesem Aspekt. Die freie Verfügbarkeit des Quelltextes für ein Produkt kann einen Vorteil für die Systemsicherheit des Web-Servers darstellen. Administratoren haben so die Möglichkeit, auftretenden Sicherheitsproblemen im Quelltext auf den Grund zu gehen. Bei Bedarf können die Mängel im Quelltext direkt behoben werden.

5.1.4. Beurteilung

Die nachfolgende Beurteilung folgt dem Ansatz, die wesentlichen Probleme der Internet-Technologien darzustellen. Für tiefergehende Ausführungen sei auf [Bel89, Bel93, Bel96, SKK+96] und vor allem auf [Fle97] verwiesen.

IPv4

Das Internet-Protokoll beinhaltet keine Sicherheitsmechanismen für die Nutzdaten. Daraus folgt, daß keine der in Kapitel 2 angeführten Sicherheitsaspekte durchgesetzt werden können. IP unterstützt nicht die Aspekte Integrität, Verfügbarkeit und Vertraulichkeit.

[1] `http://www.apache.org`

Wesentliche Ursache für die Sicherheitsprobleme im Rahmen der Nutzung des Internets ist der Umstand, daß das unterliegende TCP/IP nicht unter Sicherheitsgesichtspunkten entwickelt worden ist. Resultat dieser Entwicklung ist die Abwesenheit nahezu aller Sicherheitsmerkmale, so daß keine der Sicherheitsanforderungen erfüllt werden kann. Der Einsatz von Internet-Technologien zur Kommunikation erfordert die Existenz zusätzlicher Mechanismen, mit denen eine den Sicherheitsanforderungen genügende Kommunikation möglich wird.

Die Abwesenheit von Sicherheitsmechanismen wird deutlich an der Vielzahl von möglichen Angriffen, die sich gegen TCP/IP-Dienste und -Verbindungen richten können:

Daten-Spoofing Daten-Spoofing erfolgt im Rahmen eines „Man-in-the-middle"-Angriffs. Der Angreifer ist in der Lage, in eine bestehende Verbindung Daten zu injizieren.

Denial-of-Service Server halten für Verbindungen eine Reihe von Puffern bereit. Das gezielte Füllen der Puffer durch Angreifer hat zur Folge, daß andere Clients keine Verbindung zum angegriffenen Server etablieren können.

IP-Spoofing ist ein Verfahren, mittels dessen der Gegenseite eine Identität vorgetäuscht wird. Genutzt wird das Verfahren zur Vortäuschung einer vertrauenswürdigen Identität.

Sniffing Es existieren eine Vielzahl von Programmen, die als Netzwerk-Schnüffler (engl. network sniffers) dienen. Sie gestatten es, die IP-Pakete, die durch ihren Knoten gehen, zu analysieren und auszulesen. Angreifer können mit Hilfe von Analyse-Werkzeugen in den Besitz von Paßwörtern gelangen, die ohne Verschlüsselung in einem IP-Netzwerk übertragen werden.

Verbindungs-Hijacking Die mangelnden Sicherheitsmerkmale des TCP/IP-Protokolls versetzt Angreifer in die Lage, Verbindungen übernehmen zu können.

Der `SYN`-Flooding Angriff aus dem Jahre 1996 hat gezeigt, daß auch Diensteverweigerungs-Angriffe im Zusammenhang mit TCP möglich sind. Ein `SYN`-Flooding macht sich zu Nutze, daß der Server eine Warteliste mit gewünschten Verbindungen hält, die durch gezielt manipulierte Pakete zum Überlauf gebracht werden kann. Die Folge ist, daß legitime Nutzer keinen Zugriff mehr erhalten. TCP erlaubt es Angreifern, die Verbindungsetablierung zu verhindern und – ebenso wie in IP – bestehende Verbindungen abzubrechen oder zu drosseln. Spoofing-Verfahren sind im Zusammenhang von TCP bereits seit 1985 bekannt [Mor85]. Das von MORRIS beschriebene Verfahren erstellt eine Sequenznummer-Vorhersage und erlaubt so die Generierung von authentischen Netzwerkpaketen.

FLEGEL weist darauf hin, daß geschützte Kanäle, die mittels auf IP aufbauender Authentifizierungsverfahren, wie beispielsweise Kerberos, etabliert werden, weiterhin angreifbar bleiben [Fle97]. Angreifern steht es weiterhin offen, Pakete einzufügen, zu

entfernen oder zu modifizieren. Erst kombinierte Authentikations- und Integritätsverfahren bieten wirksame Gegenmittel. Unzureichende Implementationen des TCP/IP-Protokollstacks bereiten weitere Sicherheitsprobleme.

Neben IP und TCP bereiten auch die anderen Protokolle des Internets Sicherheitsprobleme. So ist der wesentliche Mangel des DNS-Konzeptes der nicht vorhandene semantische Zusammenhang der Bäume, die jeweils eine Abbildungsrichtung abdecken. TCP-Dienste nutzen die inverse Abbildung, um eine IP-Adresse in einen Namen aufzulösen und diesen gegen die Liste der vertrauenswürdigen Rechensysteme abzugleichen. Ziel des Angreifers ist es somit, Kontrolle über einen DNS-Server zu erlangen, so daß er in der Lage ist, Anfragen auf inverse Abbildungen mit gefälschten Informationen zu beantworten. Das anfragende System hat keine Möglichkeit zu verifizieren, ob es korrekte Informationen erhält oder nicht. Eine weiteres Problem des DNS ist die Domain-Umleitung [Fle97].

Im DNS besteht grundsätzlich das Problem, daß unabhängig vom Anfrager jede Anfrage beantwortet wird. Es findet keine Zugriffskontrolle auf Basis der Identität des Anfragers statt. Damit ist es Angreifern gestattet, ganze Teilbäume auszulesen und aus den erhaltenen Informationen neue Angriffsziele zu bestimmen. Wie FLEGEL ausführt, spiegeln die DNS-Bäume von Netzwerken häufig hierarchische Strukturen der Organisation wider. Mit dem Wissen können Angreifer wiederrum gezielte Angriffe planen [Fle97].

Ein als „Ping-of-Death“ bekannt gewordener Angriff aus dem Jahre 1997 nutzte übergroße ICMP-Datagramme, um die Zielsysteme in undefinierte Systemzustände zu überführen. Eine häufige Folge der Angriffe war der Zusammenbruch des angegriffenen Systems. Mit Hilfe geeigneter ICMP-Nachrichtenpakete ist es möglich, Pakete umzuleiten, Verbindungen zu unterbrechen und den Durchsatz von Verbindungen zu beschneiden. Abschließend ist es mit ICMP möglich, verdeckte Kanäle aufzubauen. Weitergehende Informationen zum Thema DNS sind [Bel95] und [Sch93] zu entnehmen.

IPv6

IPv6 ist mit Sicherheitsmechanismen ausgestattet, die es zukünftig gestatten, die Integrität und Authentizität von Nachrichtenpaketen, die über ein IP-Netzwerk verteilt werden, zu gewährleisten. Damit wird ein großer Fortschritt gegenüber einem IP ohne Sicherheitsmechanismen erzielt. Es muß allerdings darauf verwiesen werden, daß IPv6 nicht frei von Sicherheitsproblemen ist.

Generell stellt IP keine Mechanismen zu Verfügung, mit denen sich Verkehrsflußanalysen verhindern lassen. Ohne daß Angreifer Einblick in die transportierten Informationen erhalten, haben sie zumindest die Gelegenheit festzustellen, zwischen welchen Teilnehmern Kommunikation stattfindet.

IP ist auch in der neuen Version 6 weiterhin für Diensteverweigerungs-Angriffe anfällig. FLEGEL hat einen möglichen Angriff vorgestellt. Im Zuge des Angriffes wird das Zielsystem mit Fragmenten – Paketen, die aufgrund ihrer Länge unterteilt werden mußten – versorgt, so daß der entsprechende Puffer stets gefüllt ist. Damit kann das angegriffene

System keine weiteren Verbindungen aufnehmen. Die Authentikation des Senders erfolgt erst nach dem Zusammensetzen der Pakete, so daß die Identität des Angreifers verborgen bleibt. IPv6 erlaubt Angreifern ebenfalls, die Filterregeln von Paketfiltern durch kleine Fragmente zu unterlaufen. Möglicherweise kann dies auch durch überlappende Fragmente erfolgen [Fle97]. Auch ICMP kann in der neuen Version zu Angriffszwecken mißbraucht werden. Die für IPv4 gültigen Verfahren der Paket-Umleitung, Abbruch bestehender Verbindungen und das Drosseln der Datenübertragung behalten für IPv6 ihre Gültigkeit. Lediglich das Drosseln der Datenübertragung ist schwieriger geworden. Durch die minimale Paketlänge von 576 Oktetten werden mit jedem Paket stets Nutzdaten übertragen. Die Nutzung von ICMP-Paketen für verdeckte Kanäle setzt den Zugriff auf das interne System voraus. Damit ist ein Angreifer gegebenenfalls auch in der Lage, Zugriff auf die kryptografischen Komponenten zu erhalten, so daß verschlüsselte Pakete keinen Schutz mehr bieten.

Angriffe auf Basis von TCP erfordern die Fälschung der Absenderadressen. Die Angriffe können unterbunden werden, wenn ausschließlich Sicherungsheader verwendet werden, die eine Authentifizierung des Absenders gestatten.

Das DNS kann seine Antworten zukünftig mit digitalen Signaturen sichern. Das Problem der verfälschten Antworten besteht jedoch weiterhin, wenn der Angreifer Zugriff auf einen Teilbaum des DNS und zudem Zugriff auf die privaten Schlüssel hat. In diesem Fall ist der Angreifer wie unter IPv4 in der Lage, inkorrekte Angaben zu produzieren und diese zusätzlich noch zu authentisieren. Der Einsatz von Sicherungsheadern mindert jedoch den Nutzen derartiger Angriffe [Fle97]. Daneben existiert allerdings weiterhin das Problem, daß keine Zugriffskontrolle auf die vom DNS bereitgestellten Informationen existiert.

Abschließend sei darauf verwiesen, daß IPv6 bislang nur zu Experimenten in Netzwerken verwendet wird. Es bliebt zunächst abzuwarten, ob sich IPv6 in naher Zukunft etablieren wird. Solange nicht mindestens die Endteilnehmer eine Unterstützung von IPv6 aufweisen, bleiben die erzielten Sicherheitsverbesserungen des Protokolls ungenutzt.

World-Wide Web

Web-Server sind das Aushängeschild vieler Organisationen und Unternehmen im Internet. Dieses macht sie zum vorrangigen Ziel zahlreicher Angriffe.[2] In der Mehrzahl der Fälle richteten sich die bisherigen Angriffe gegen die Inhalte der Server, die in Form von HTML-Dokumente angeboten werden. Gelingt der Einbruch in einen Web-Server, hat ein Angreifer aber ebenso die Möglichkeit, Programme bzw. Agenten auf dem Web-Server zu installieren. Derartige Einbrüche und die von ihnen ausgehenden Konsequenzen können wesentlich länger verborgen bleiben, als der Austausch von HTML-Dokumenten, der i. d. R. innerhalb kürzester Zeit bemerkt wird.

[2] Eine Liste von Einbrüchen auf Web-Server bekannter Organisationen und Unternehmen ist unter `http://www.nearlive.com/archive/` zu finden.

Wie bereits in einem vorherigen Abschnitt angesprochen wurde, liegt für eine Reihe der Web-Server der Quelltext im Internet vor. Das Internet ist aus diesem Grunde eine der wesentlichen Bezugsquellen für Web-Server. Ein bekanntes Beispiel hierfür ist der derzeit weitverbreiteste WWW-Server Apache. Für die Systemsicherheit bildet die freie Verfügbarkeit des Quelltextes damit auch ein Problem, da nahezu ausschließlich nicht authentifizierte Kopien des Quelltextes aus Netzwerken wie dem Internet bezogen werden. Angreifer haben die Möglichkeit, manipulierte Versionen des Quelltextes auf FTP-Servern zur Verfügung zu stellen. Zwei Aspekte sind dabei wesentlich. Zum einen kann der Angreifer eine bereits vorkompilierte Version des Servers anbieten, so daß der Nutzer keine Möglichkeit hat, etwaige Änderungen im Quelltext aufzuspüren. Zum anderen sind Web-Server mittlerweile sehr komplexe Programme, die zudem durch eine Vielzahl von externen Modulen erweitert werden können. Es wird damit für den Betreuer eines Web-Servers nahezu unmöglich, nachzuvollziehen, ob der Quelltext, der aus einem Netzwerk bezogen wurde, Sicherheitsprobleme bereiten kann.

In der Vergangenheit waren Mängel in der Implementation von Web-Servern und CGI-Programmen eine häufige Ursache für Sicherheitsprobleme im Umfeld von Web-Servern. Insbesondere CGI-Programme, die die Funktionalität von Web-Servern beliebig erweitern können, haben häufig nicht direkt ersichtliche Implikationen auf die Systemsicherheit des Servers. Einschränkungen, die durch die Konfiguration des Web-Servers vorgegeben wurden, können durch den Einsatz von CGI-Programmen unbeabsichtigt wieder aufgehoben werden.

Die mangelhafte Konfiguration eines Web-Servers ist ein weiteres Problem. Eine vollständige Trennung von sensitiven und nicht sensitiven Dokumenten ist häufig nicht gegeben. Bereits das Wissen über die Verzeichnisstruktur, die von vielen Servern automatisch generiert wird, verschafft den Angreifern Informationen, die zu Angriffszwecken mißbraucht werden können. CGI-Programme werden i. d. R. in Form von Skripten implementiert, so daß zu ihrer Ausführung Interpreter benötigt werden. Der Einfachheit halber wird der Interpreter derart plaziert, daß er im selben Verzeichnis wie die Skripte liegt. Dieser schwerwiegende Konfigurationsfehler erlaubt es Angreifern, dem Interpreter beliebige Skripte zu übergeben, die von diesem ohne Einschränkungen ausgeführt werden.

Obwohl Webserver als öffentliche Informationsdienste ausgewiesen sind, wurde ein Standard entwickelt, mittels derer der Web-Server festlegen kann, welche Verzeichnisse durch Agenten ausgewertet werden dürfen.[3] In einem Artikel des Risk-Digest hat Joe Dellinger bereits 1998 darauf hingewiesen, daß Web-Server eine Vielzahl von Informationen bereithalten, die nicht für die Öffentlichkeit bestimmt sind [Del96]. Eine Möglichkeit diesem zu begegnen bietet der Robot Exclusion Standard [Kos95b]. Nach diesem Standard kann jeder Web-Server im Wurzelverzeichnis eine Datei `robots.txt` anlegen, die dem jeweiligen Robot mitteilt, welche Verzeichnisse nicht auszuwerten sind. Wie Betrand Meyer in seinem Artikel „Here is what I am not telling you“ richtig fest-

[3] Siehe hierzu [Kos95a]

stellt, wird gerade hierdurch jedem maliziösen Agenten mitgeteilt, welche Informationen vom Server-Betreiber als vertraulich eingestuft werden [Mey98].[4] Die Datei `robots.txt` ist für jeden lesbar, nur auf diese Weise kann sie von Agenten ausgewertet werden.

Agenten können so eingesetzt werden, um in einem ersten Schritt Hinweise auf die Orte vertraulicher Informationen zu erhalten. In nachfolgenden Schritten geht der Angreifer dann dazu über, weitere Sicherheitslücken des Systems zu ermitteln, um so Zugriff auf die vertraulichen Information zu erhalten. Der Robots Exclusion Standard ist sinnvoll um die Indizierung von HTML-Dokumenten zu vermeiden, er bietet jedoch keine Möglichkeit um den Zugriff auf vertrauliche Dokumente zu unterbinden. Vertrauliche Dokumente sind allerdings auch nicht auf einem öffentlich zugänglichen Web-Server unterzubringen. Es sei darauf hingewiesen, daß dieses nicht die Intention des Standards war, er aber in dieser mißbräuchlichen Form von Organisationen und Unternehmen genutzt wird. Die Abbildung 5.1 zeigt deutlich, daß der mißbräuchliche Einsatz des Standards zu Sicherheitsproblemen führen kann.

```
User-agent: *
Disallow: /bug-navigator # Bug Data
Disallow: /warp/customer # Registered Users
Disallow: /kobayashi # Navigation for registered
Disallow: /cgi-bin # no programs
Disallow: /pcgi-bin # no programs
Disallow: /univ-src/ccden # will get content through /univercd
Disallow: /cpropub/univercd # obsolete
```

Abbildung 5.1.: Das Beispiel zeigt eine Datei `robot.txt`, die eine Reihe von Verzeichnissen anführt, die von Agenten nicht besucht werden sollen. Den Namen der Verzeichnissen ist zu entnehmen, daß dort zum Teil sensitive Informationen enthalten sein dürften (Quelle: [Mey98]).

5.2. Browser

Seit der Entwicklung des ersten Browsers am Institut CERN hat deren Funktionalität eine rasante Verwandlung erfahren. Die ersten Versionen von Mosaic[5] ermöglichten lediglich die Darstellung von HTML-Dokumenten sowie die Darstellung von Grafiken. Heute enthält ein Web-Browser einen Mail- und News-Client, unterstützt die Sprachen Java und JavaScript, ermöglicht Web-Conferencing und Internet-Telefonie. Zusätzlich ist die

[4] Die sich seinem Artikel anschließende Diskussion hat Meyer unter `http://www.eiffel.com/private/meyer/robots.html` zusammengefaßt.

[5] Mosaic war der erste Web-Browser und wurde am CERN von Tim Berners-Lee entwickelt.

Funktionalität eines Browsers durch die Installation von Helper Applications bzw. Plugins erweiterbar. Sehr deutlich wird die gestiegene Komplexität auch in dem benötigtem Hintergrundspeicher. Waren anfänglich weniger als ein Megabyte Speicherplatz für einen Browser erforderlich, werden heute zwischen 15 und 30 Megabyte benötigt.

Mit dem Internet Explorer von Microsoft und dem Communicator von Netscape existieren zwei Browser, die für eine Vielzahl von Agentensystemen als Umgebungssystem geeignet sind. Als Umgebungssystem bilden sie damit eine der Basiskomponenten von Agentensystemen.

In den nachfolgenden Abschnitten werden die Browser Communicator und Internet Explorer vorgestellt. Es erfolgt eine Analyse der Sicherheitsarchitekturen der Produkte und eine Aufstellung der spezifischen Sicherheitsprobleme. Abschließend erfolgt eine zusammenfassende Darstellung, die die übergreifenden Sicherheitsmängel beinhaltet.

5.2.1. Internet Explorer

Der Internet Explorer[6] kann als Umgebungssystem für ActiveX- und Java-Agenten dienen. Für beide Systeme bietet er eine Ausführungsumgebung an. In den Browser sind mit Authenticode, Java Capabilities und dem Zonenkonzept drei Mechanismen integriert, die der Sicherheit der Systeme dienen sollen, auf denen der Browser installiert ist. Die Verfahren sollen laut Microsoft gewährleisten, daß den aus dem Einsatz von Agenten resultierenden Bedrohungen begegnet werden kann.

Im Anschluß erfolgt zunächst die Vorstellung der Sicherheitsmechanismen. Darauf aufbauend wird analysiert, in welchem Maße die Mechanismen geeignet sind, den Gefahren der Agentensysteme zu begegnen.

Authenticode

Authenticode ist die Komponente der Sicherheitsarchitektur des Internet Explorers, die die Entscheidungsgrundlage schaffen soll, anhand derer beurteilt wird, ob die Ausführung eines Agenten gestattet werden kann. Mit Hilfe von Authenticode soll die Identifikation des Autors eines Programmes erfolgen können. Weiterhin soll das Sicherheitskriterium Integrität für einen Agenten (Programm) verifizierbar sein [Mic96].

Authenticode basiert im wesentlichen auf der Technologie der digitalen Signaturen. Mit Authenticode hat der Produzent eines Agenten die Möglichkeiten, signierte Agenten zu generieren. Zu dem zu signierenden Code wird zunächst ein *Fingerabdruck* (engl. *fingerprint*) mittels Hashverfahren gebildet. Der Hashwert hat dabei eine Länge von 128 oder 160 Bit. Anschließend verschlüsselt der Produzent des Agenten den Hashwert mit seinem privaten Schlüssel und fügt die damit erzeugte digitale Signatur dem Agenten bei.

Empfängt der Internet Explorer einen digital signierten Agenten, wird zunächst der Hashwert des Agenten berechnet. Anschließend wird der dem Agenten beigefügte

[6] Grundlage sind die Versionen 3 und 4 des Internet Explorers.

Hashwert extrahiert und mit dem vorher berechneten Wert verglichen. Stimmen beide Hashwerte überein, ist die Authentizität des Agenten und seine Integrität bewiesen. Im nächsten Schritt wird das Zertifikat des Agenten präsentiert. Dem Zertifikat ist der Unterzeichner des Agenten und die Certification Authority (CA), die das Zertifikat ausgestellt hat, zu entnehmen. Wird das Zertifikat eines Agenten akzeptiert, wird dieser anschließend auf dem lokalen System installiert. In der Folge gilt der Agent entsprechend dem Authenticode-Verfahren als vertrauenswürdig und es finden keine weiteren Überprüfungen statt, wenn der Agent zur Ausführung gebracht wird.

Stellt der Browser im Zuge der Überprüfung der digitalen Signatur fest, daß die Signatur ungültig ist – der Agent also manipuliert wurde – wird dieses dem Anwender mittels einer entsprechenden Meldung mitgeteilt. Damit besteht weiterhin die Möglichkeit, den Agenten zu installieren. Im Falle eines nichtsignierten Agenten wird der Anwender auf diesen Umstand hingewiesen und befragt, ob der Agent installiert werden soll.

Zertifikate werden von unabhängigen CAs wie VeriSign herausgegeben. Microsoft nutzt für die Identitäts- und Schlüsselinformationen dem Standard X.509 v3 konforme Zertifikate. Signaturen genügen dem Standard PKCS #7 und #10 [Mic96]. Der Internet Explorer ist bereits mit einer Anzahl von Zertifikaten ausgestattet. Die Zertifikate stammen im wesentlichen von CAs, die sicherstellen sollen, daß neue, von einer CA signierten Zertifikate, überprüft werden können

Authenticode unterscheidet zwei Klassen von Zertifikaten. Ein Zertifikat der Klasse 2 ist für Individualpersonen gedacht. Für die Ausstellung des Zertifikats sind Angaben über den Namen, Adresse, Email-Adresse, Geburtsdatum und der Sozialversicherungsnummer erforderlich. Im Anschluß an die Verifizierung der Informationen wird das Zertifikat herausgegeben. Für ein Zertifikat der Klasse 3, das für kommerzielle Softwareproduzenten gedacht ist, wird darüberhinaus ein Dun-and-Bradstreet Rating benötigt.

Der Besitzer eines Zertifikates muß sich verpflichten, daß er keine Software vertreibt, von der ihm bekannt ist bzw. von der ihm hätte bekannt sein müssen, daß sie Viren oder andere maliziöse Anweisungen enthält, die das Rechensystem des Anwenders beeinträchtigen könnten. Die Verpflichtung des Zertifikatsbesitzers verlangt weiterhin, daß Entwicklungs- und Vertriebspraktiken angewandt werden, die den vorherrschenden Industriestandards entsprechen [Mic96]. Verstößt ein Softwareproduzent gegen die Auflagen, kann sein Zertifikat entzogen werden, mit der Folge, daß auch alle anderen vom Softwareproduzenten unterzeichneten Agenten eine nicht länger gültige Signatur aufweisen.

Die Authenticode Technologie erlaubt den Entwicklern weiterhin, ActiveX-Agenten auf spezielle Weise zu kennzeichnen. Mit einem als „safe for initializing“ gekennzeichnetem Agent bestätigt der Autor, daß der von ihm entwickelte Agent sich unabhängig von den Eingabewerten stets gutartig verhält. Die Markierung „safe for scripting“ ist eine Bestätigung des Autors, daß der entsprechende Agent unabhängig davon, wie seine Methoden aufgerufen und die Eigenschaften manipuliert werden, keinen Schaden auf dem ausführenden System bewirken kann [Joh96]. Die Entscheidung über eine Kennzeichnung trifft der Autor des Agenten. Die anschließende Entscheidung des Internet Explorers basiert ausschließlich auf der Kennzeichnung durch den Autor. Semantisch

Abbildung 5.2.: Nach dem Empfang eines Agenten wird dem Anwender ein Zertifikat präsentiert, daß die Identität des Produzenten des Agenten enthält. Neben der Akzeptierung des Agenten besteht die Wahl, alle folgenden Objekte – somit auch alle Agenten, die vom gleichen Produzenten stammen und/oder von der gleichen CA unterzeichnet worden, zukünftig ohne Intervention des Anwenders zu akzeptieren.

bestätigt der Autor mit der Kennzeichnung seines Agenten, daß dieser sich unabhängig von dem Dokument, in das er eingebettet ist, stets gutartig verhält.

Die vom Agenten benötigten Privilegien werden im Authenticode-Verfahren in der Signatur versteckt. Die Granularität der Rechtevergabe ist sehr grobkörnig. Es wird lediglich zwischen den Sicherheitsstufen Hoch, Mittel und Niedrig unterschieden. Die Privilegien beziehen sich im Falle von Authenticode auf die gesamte signierte CAB-Datei.[7]

Java-Klassen können ebenso wie die in Abschnitt 3.4.2 angeführten ActiveX-Agenten

[7] Netscape erlaubt dagegen die Signierung einzelner Klassen und darauf basierend die Erteilung einzelner Privilegien je Klasse. Damit handelt es sich um ein wesentlich feinkörnigeres Verfahren, als jenes, welches von Microsoft entwickelt worden ist.

Abbildung 5.3.: Im Falle, daß Authenticode im Laufe der Signaturverifikation feststellt, daß der Agent zwischenzeitlich modifiziert worden ist, wird dem Anwender dieser Warnhinweis gegeben und die Installation des Agenten wird unterbunden.

mittels Authenticode signiert werden. Ein Java-Agent wird dabei mit allen notwendigen Klassen zu einer CAB-Datei zusammengefaßt. Die dort enthaltenen Daten werden anschließend komprimiert und signiert. Das von Netscape und Sun genutzte Format Java Archive (JAR) ist zu dem von Microsoft verwendetem Verfahren nicht kompatibel. Somit muß für jeden Browser ein explizit signiertes Archiv zur Verfügung gestellt werden, so ein signierter Agent von beiden Browsern unterstützt werden soll.

Abbildung 5.4.: Empfängt der Internet Explorer einen Agenten, der nicht signiert worden ist, wird dieses dem Anwender durch diese Meldung mitgeteilt.

Mit Authenticode in der Version 2.0, die in den Internet Explorer 4.0 integriert ist und als Erweiterung für die Version 3.0 aus dem Internet bezogen werden kann, haben Entwickler die Möglichkeit, ihren signierten Agenten mit einem Zeitstempel zu versehen. Der Zeitstempel attestiert, daß der Agent während der Gültigkeit des korrespondierenden Zertifikates signiert worden ist. Mit einem Zeitstempel versehene signierte Agenten erhalten damit eine nicht limitierte Lebenszeit. Desweiteren erlaubt die neue Version Authenticodes dem Internet Explorer vor dem Bezug eines entsprechend signierten Agenten zu überprüfen, ob das Zertifikat, das die Gültigkeit der Signatur attestiert, in der Zwi-

schenzeit zurückgezogen worden ist.

Zonenkonzept

Mit Version 4.0 beinhaltet der Internet Explorer eine neue Sicherheitskomponente, die unter dem Begriff der *Sicherheitszone* (engl. *Security Zone*) eingeführt wurde. Das neue Konzept der Sicherheitszonen soll dem Anwender bei der Entscheidung sicherheitskritischer Fragen behilflich sein. Über Sicherheitszonen werden die Netzwerke bzw. Rechner in unterschiedliche Domänen eingeordnet. Resultierend aus der Zuordnung von Servern zu Sicherheitszonen, werden diese entsprechend der für die Sicherheitszone definierten Sicherheitspolitik behandelt, wenn der Anwender auf diesen Dokumente aufruft. Für alle Server, die einer Sicherheitszone zugeteilt wurden, unterbleiben Befragungen des Nutzers, da die Sicherheitsentscheidungen aufgrund der der Sicherheitszone zugeteilten Sicherheitspolitik erfolgen kann. Allen Servern einer Sicherheitszone wird das gleiche Vertrauen entgegengebracht. Das Zonenkonzept umfaßt mit Auslieferung vier vordefinierte Zonen: Internet, lokales Intranet, vertrauenswürdige Server und eingeschränkte Server. Für jede einzelne Zone können Sicherheitsmerkmale gesetzt werden. Entsprechend den zugeordneten Sicherheitsmerkmalen setzt jede Zone eine andere Sicherheitspolitik durch. Jeder Zone kann eine der drei vordefinierten Sicherheitsstufen Hoch, Mittel oder Niedrig zugeordnet werden.

Die folgenden Regeln gelten für die unterschiedlichen Sicherheitszonen:

Hoch: Java-Agenten werden ohne Konsultation des Anwenders in der Sandbox ausgeführt. Alle Methodenaufrufe, die aus der Sandbox hinausgehen, erfordern der Zustimmung des Benutzers. ActiveX-Agenten werden nicht ausgeführt.

Mittel: Java-Agenten können ein Megabyte Speicherplatz auf dem lokalen Rechner benutzen. Anfragen hierüber hinaus erfordern die Zustimmung des Anwenders. Zugriffe auf das lokale Dateisystem als auch die Ausführung von ActiveX-Agenten sind ebenfalls mit Zustimmung des Anwenders möglich.

Niedrig: Java-Agenten werden ohne Einschränkungen ausgeführt. Es erfolgen keine Konsultationen des Anwenders. Die Ausführung von ActiveX Agenten erfordert die Zustimmung des Anwenders.

Benutzerdefiniert: Benutzerdefinierte Sicherheitsstufen können erstellt werden. Es ist möglich, ActiveX-Agenten ohne Zustimmung des Anwenders ausführen zu lassen [Mic98a].

In der Standardeinstellung ist der Internetzone die Sicherheitsstufe „Mittel“ zugeordnet, während alle Rechner, die in der Sicherheitszone der vertrauenswürdigen Rechner eingeordnet sind, lediglich der Sicherheitsstufe „Niedrig“ unterliegen, d. h. Rechner innerhalb des Intranets werden ohne Unterscheidung als vertrauenswürdig behandelt.

Abbildung 5.5.: Das Zonenkonzept des Internet Explorers erlaubt die Einteilung der Dienste, die in einem Netzwerk in Anspruch genommen werden, in unterschiedliche Zonen. Für jede einzelne Zone wird festgelegt, welcher Sicherheitsstufe sie unterliegt. Zur Auswahl steht, neben den drei vordefinierten Stufen Hoch, Mittel und Niedrig, eine benutzerdefinierte Sicherheitsstufe.

Daneben existiert eine weitere, benutzerdefinierte Sicherheitsstufe, in der der Anwender feinkörniger bestimmen kann, welche Rechte Servern erteilt werden, die dieser Stufe zugeordnet werden. Zu den dort einstellbaren Sicherheitsoptionen zählen:

- ▷ der Zugriff auf Dateien, ActiveX-Agenten und Skripte;
- ▷ Identifizierung von Servern mittels Secure Sockets Layer (SSL);
- ▷ Festlegung der Capabilities für Java Applets.

Der Internet Explorer informiert Benutzer in der Sicherheitsstufe „Mittel“ über potentiell unsichere (unsignierte) Controls und erlaubt ihnen deren Ausführung [Mic96].

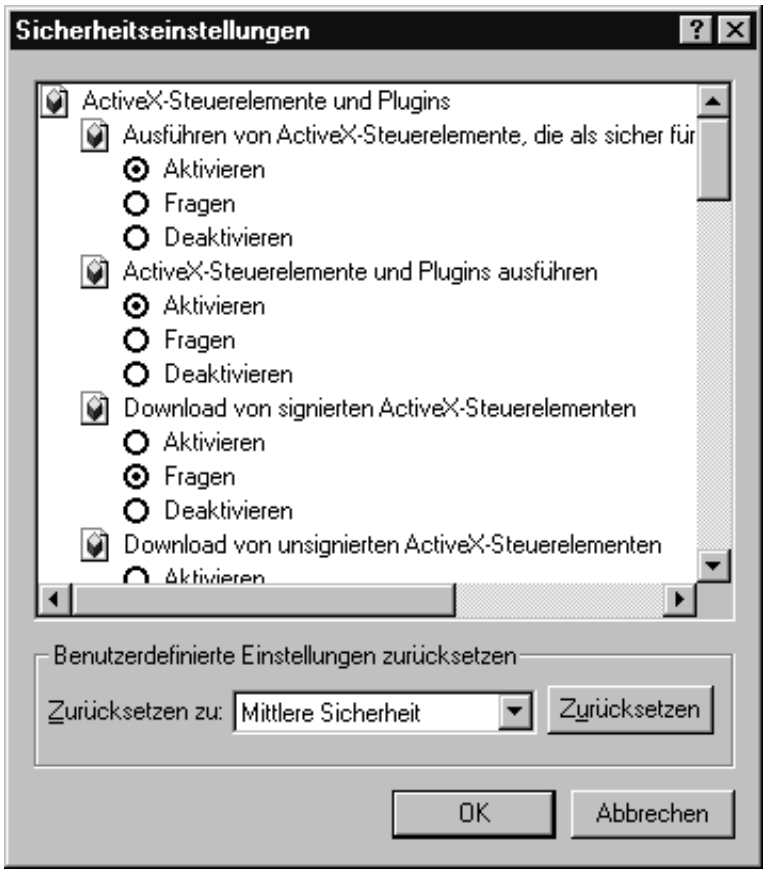

Abbildung 5.6.: Im Falle der benutzerdefinierten Sicherheitseinstellungen bietet der Internet Explorer die Möglichkeit, auf einzelne Aspekte der Sicherheitspolitik des Browsers Einfluß zu nehmen.

Java Capabilities

Mit dem Internet Explorer in der Version 4.0 wird für Java-Agenten eine Sicherheitsarchitektur eingeführt, die auf Capabilities beruht. Das von Microsoft entwickelte Capability-System erlaubt die Zuteilung von Rechten bevor der Agent ausgeführt wird [Mic97b]. In Verbindung mit dem Zonenkonzept sind jeder Zone eine Reihe von Rechten zugeordnet, die von Java-Agenten in Anspruch genommen werden können. Die den jeweiligen Zonen zugeordneten Rechte für Java-Agenten können durch den Anwender verändert werden.

Die den jeweiligen Zonen zugeordneten Capabilities werden in drei Klassen eingeteilt. In der ersten Klassen sind jene Capabilities enthalten, die Java-Agenten zugeteilt werden können, ohne daß der Anwender konsultiert wird. Die zweite Klasse enthält jene Capabilities, deren Zuteilung eine explizite Zustimmung des Nutzers erfordert. In dieser Klasse nicht enthaltene Capabilities werden den Agenten nicht zugeteilt. In der letzten Klasse sind schließlich jene Capabilities aufgelistet, welche kein Agent der Zone in Anspruch

nehmen kann [Mic98b].

Java-Agenten, die Methoden aufrufen, für die sie nicht die notwendigen Capabilities erhalten haben, lösen innerhalb der JVM eine Sicherheitsausnahme aus, die durch den Agenten aufgefangen werden kann.

Unterschieden wird zwischen parametrisierten und nichtparametrisierten Capabilities. Ein Beispiel für eine parametrisierte Capability ist `NetIOPermission`. Einem Agenten kann diese Capability mit der Parametrisierung, daß lediglich eine Netzwerkverbindung zu einem bestimmten Rechner aufgebaut werden kann, erteilt werden.

Java-Agenten können mit dem Tool `signcode` durch den Entwickler signiert werden. Im Zuge der Signierung kann der Agent in unterschiedliche Sicherheitsstufen eingeordnet werden. Anhand dieser Informationen ist der empfangende Internet Explorer in der Lage zu bestimmen, welche Rechte der Agent für seine Ausführung benötigt. Signierte Java-Agenten werden in Form von CAB-Dateien übertragen. Diese sind mit einer Signatur versehen, in der die für die Ausführung des Agenten erforderlichen Rechte kodiert sind. Die Signierung bezieht sich auf alle im Archiv enthaltenen Agenten. D. h., es werden nicht einzelne Agenten, sondern wie durch die Authenticode-Technologie vorgegeben, Archive digital signiert.

Ist die JVM des Internet Explorers nicht in der Lage, die Capabilities automatisch zu gewähren, wird dem Anwender eine Dialog-Box präsentiert, in der die Frage gestellt wird, ob der Java-Agent ausgeführt werden soll. Dieses ist dann der Fall, wenn die von ihm benötigten Capabilities nicht mit denen der Sicherheitsstufe, in die er eingeordnet wurde, übereinstimmen. Die Befragung des Anwenders unterbleibt, wenn das zum Agenten gehörige Dokument in einer Zone liegt, deren Rechte denen durch den Agenten geforderten entsprechen oder darüberhinaus gehen.

ActiveX-Controls

ActiveX-Agenten unterliegen während ihrer Ausführung keinerlei Einschränkungen. Nach der erfolgreichen Installation eines Agenten auf einem lokalem System erfolgen keine weiteren Sicherheitsüberprüfungen bezüglich der Integrität und der Konformität eines ActiveX-Agenten. Das Sicherheitskonzept der ActiveX-Agenten fußt ausschließlich auf vertrauensbasierenden Prinzipien, die mittels der Authenticode-Technologie vor der Installation des Agenten sichergestellt werden. Die ActiveX-Technologie beinhaltet somit keine Sicherheitsarchitektur. Es stehen keine weiteren Sicherheitskomponenten/-mechanismen zur Verfügung, die über die Mittel hinausgehen, die das Umgebungssystem anbietet.

Nach der Installation eines ActiveX-Agenten kann dieser wie eine normale Windows Applikation agieren. ActiveX-Agenten haben uneingeschränkten Zugriff auf den gesamten Arbeitsspeicher, alle Betriebssystemfunktionen, alle Dateien des Dateisystems sowie Zugriff auf die Netzverbindungen.

Beurteilung

Die von Microsoft entwickelte Sicherheitsarchitektur des Internet Explorers ist für die Sicherung von Agentensystemen, insbesondere wenn ActiveX-Agenten zum Einsatz kommen, nicht ausreichend. Das Authenticode-Verfahren überträgt dem Anwender die Entscheidung, ob Agenten zur Ausführung kommen sollen, ohne daß es ausreichende Informationen beiträgt, aufgrund derer der Anwender seine Entscheidung treffen kann. Die Sicherheitsarchitektur wird dem mit der Agententechnologie einhergehenden Paradigmenwechsel nicht gerecht. Die Sicherheitsdimensionen werden durch die verwendeten Technologien nicht adressiert, eine Durchsetzung der in Abschnitt 2.3 aufgestellten Sicherheitskriterien unterbleibt.

Mit Authenticode hat Microsoft ein binäres Vertrauensmodell entwickelt. Agenten, denen der Anwender aufgrund des präsentierten Zertifikates vertraut, werden in seinem Rechensystem installiert. Agenten, denen das Vertrauen verweigert wurde, werden nicht installiert und können damit nicht ausgeführt werden. Eine differenzierte Entscheidung ist dem Anwender damit nicht möglich. Dieses ist ein äußerst problematischer Umstand, insbesondere in Hinblick auf die Mächtigkeit der ActiveX-Agenten.

In der Vergangenheit wurden eine Reihe von Implementationsfehlern innerhalb des Browsers entdeckt, die es Agenten erlaubten, unter Umgehung aller Sicherheitskontrollen Zugriff auf die lokalen Systemressourcen zu erhalten. So zeigte ein Implementationsfehler, der Ende 1997 im Internet Explorer entdeckt worden ist, daß auch Browser anfällig für datengesteuerte Angriffe sind. Ein Student am Massachusetts Institute of Technology (MIT) entdeckte zu dieser Zeit, daß URLs zu einem Pufferüberlauf führten, wenn sie länger als 256 Zeichen waren. Der Mangel konnte dazu ausgenutzt werden, um einen an den URL angehängten binären Code auszuführen.

Entdeckte Sicherheitslücken führen nicht zu einer Überarbeitung bzw. der Modellierung eines neuen Sicherheitskonzeptes. Entdeckte Sicherheitslücken werden stets nur mit einem Patch behoben und nicht mittels einer generellen Lösung. Die Geschwindigkeit der Produktion von Patches in Verbindung mit der zunehmenden Integration von Browser und Betriebssystem birgt potentiell eine nicht abschätzbare Menge neuer Seiteneffekte.

Im Verhältnis zur vorhergehenden Version bietet der Internet Explorer in der Version 4.0 wesentlich mehr Möglichkeiten, auf die Sicherheitseinstellungen des Browsers Einfluß zu nehmen. Die Vorteile, die aus dieser Einflußnahme durch den Nutzer erwachsen, werden allerdings durch die Unübersichtlichkeit und die unzureichende Dokumentation derselbigen kompensiert. Die Optionen, mit denen Einfluß auf die Sicherheitsmerkmale genommen werden kann, sind über verschiedene Menues verteilt, die wiederum mehrere unterschiedliche Auswahlmerkmale aufweisen. Entsprechend ist auch die Dokumentation verteilt. Eine detaillierte Schritt-für-Schritt Beschreibung der Sicherheitsmerkmale des Browsers ist nicht vorhanden. Eine Einführung in das Thema Sicherheit, die gerade für Arbeiten in offenen Netzen erforderlich ist, ist in der Dokumentation nicht enthalten. Eine Dokumentation der vielen Sicherheitsoptionen, die die durch den Browser durchgesetzte Sicherheitspolitik beeinflußt, ist nicht vorhanden.

Desweiteren sind eine Reihe von Sicherheitsmerkmalen deaktiviert, die von Bedeutung für die Sicherheit eines Umgebungssystemes sind. So werden in der Standardeinstellung Zertifikate neuer Agenten nicht dahingehend überprüft, ob das Zertifikat in der Zwischenzeit zurückgezogen worden ist. Die entsprechende Option deaktiviert ist. Ebenso ist die rudimentäre Protokollierung der Aktivitäten von Java-Agenten in der Standardeinstellung deaktiviert. Eine Protokollierung der durch ActiveX-Agenten durchgeführten Operationen erfolgt überhaupt nicht. Dagegen ist selbst in der höchsten voreingestellten Sicherheitsstufe „Hoch" die Option aktiviert, die die Ausführung von ActiveX-Agenten gestattet, wenn diese als „safe for scripting" gekennzeichnet sind. Es wird kein Hinweis darauf gegeben, daß diese Kennzeichnung alleinig durch den Produzenten des Agenten erfolgt.

Der Internet Explorer beinhaltet, wie auch der im nächsten Abschnitt vorgestellte Communicator, lediglich Sicherheitskomponenten, die dem Schutz des Umgebungssystemes vor Agenten dienen sollen. Angriffe auf Agenten sind, wie das Beispiel eines Angriffes des „Chaos Computer Club" (CCC) auf einen Agenten der Firma Brokat gezeigt hat, jedoch nicht nur eine theoretische Möglichkeit [MM97].

Nach der allgemeinen Sicherheitsbeurteilung des Internet Explorers werden in den folgenden Abschnitten die einzelnen Komponenten detailliert beurteilt.

Authenticode Authenticode ist eine von Microsoft entwickelte Technologie, die eine Reihe von Sicherheitsproblemen aufweist. Die Annahme, daß von Unternehmen über das Internet verteilte Software a priori vertrauenswürdig ist, ist ein Trugschluß. Diese Art des Bezugs von Software unterscheidet sich wesentlich vom herkömmlichen Modell des Kaufes der Software in Form einer CD-ROM. Eine Reihe von Unterschieden seien hier angeführt:

- ▷ Software, die in Form einer CD-ROM erworben wird, wird durch den Anwender bewußt installiert und ausgeführt. Software, die in Form von Agenten auf ein System gelangt, wird durch eine einzelne Bestätigung auf dem Wirtsrechner installiert und ausgeführt. Durch fehlerhafte Parametrisierung der Umgebung (Internet Explorer) kann dieses für den Anwender vollständig transparent erfolgen. Darüberhinaus wird die Quelle des Agenten – der Web-Server – nicht authentifiziert, so daß der Anwender nicht sicher sein kann, von wem er den Agenten bezieht.[8]

- ▷ Das Internet bietet jedem die Möglichkeit, Software zu vertreiben und damit insbesondere auch maliziöse Software. Der Vertrieb maliziöser Software über Distributoren, Einzelhandelsgeschäfte oder Kaufhäuser ist hingegen wesentlich aufwendiger.

- ▷ Software aus dem Geschäft muß bewußt gekauft werden. Dieses erfolgt i. d. R. nach eingehender Überlegung. Das Internet bietet Anwendern hingegen die Möglichkeit, Software zu testen, ohne einen Betrag dafür zu entrichten. Das hat zur Folge, daß

[8] Siehe hierzu auch [Hop97b]

eine Vielzahl von Produkten installiert wird, ohne daß die damit verbundenen Gefahren gesehen werden. Anwender werden ein Programm nutzen wollen, wenn es ihnen eine ansprechende Funktionalität verspricht. Dem „normalen“ Anwender sind die Implikationen seines Handelns auf die Systemsicherheit seines Rechners dabei nicht bewußt.

Authenticode basiert damit auf der Annahme, daß der Nutzer zwischen seriösen und nicht seriösen Anbietern unterscheiden kann. Im Hinblick auf die Darstellungsmöglichkeiten, die im Internet mit wenigen Mitteln zu erreichen sind, muß diese Annahme zurückgewiesen werden. Es ist für Nutzer nicht möglich, gesichert zwischen seriösen und nicht seriösen Anbietern zu unterscheiden. Dieses gilt auch dann, wenn dem Nutzer ein Zertifikat präsentiert wird, dem er die Identität des Control-Signierers und die ausstellende Zertifizierungsinstanz entnehmen kann.

Eine digitale Signatur ist ebenfalls kein ausreichendes Sicherheitsmerkmal einer Software bzw. eines Agenten, da nicht zu ersehen ist, welche Semantik mit ihr verbunden ist. Der Signatur eines Agenten ist nicht zu entnehmen, welchem Zweck die Signatur dient, was mit ihr ausgedrückt werden soll und welche Personen im Besitz des Signierungsschlüssels sind [DS97].

Software läßt sich aus Sicht der Sicherheit in drei Kategorien einteilen: vertrauenswürdige, gutartige und maliziöse Software [Gas88]. Ein Programm fällt in die Kategorie der vertrauenswürdigen Software, wenn es von vertrauenswürdiger Seite nach strikten Standards entwickelt wurde und gezeigt wurde, daß sie korrekt im Sinne einer formalen Modellierung und Verifikation ist. Vertrauenswürdig ist nur die sicherheitsrelevante Software, diese liegt innerhalb des Sicherheitsperimeters.

Demgegenüber stehen die als vertrauenswürdig bezeichneten Agenten, wie z.B. signierte Java- oder ActiveX-Agenten. Dem Anschein nach sind diese Agenten der Kategorie der vertrauenswürdigen Software zuzurechnen. Dabei wird jedoch übersehen, daß mit der Signierung nicht dem Agenten, sondern dem Unterzeichner Vertrauen entgegengebracht werden soll. Derartige Agenten können im besten Falle als gutartig bezeichnet werden. Dem Anwender stellt sich durch die Begrifflichkeit allerdings ein anderes Bild dar, weshalb „vertrauenswürdige“ Agenten existieren, die maliziöse Operationen durchführen.

Ein großes Problem des Ansatzes der „vertrauenswürdigen“ Software tritt dann ein, wenn ein Agent unberechtigt bzw. irrtümlicherweise das Vertrauen der ausführenden Instanz erhält. In diesem Fall gibt es keine nachfolgende kontrollierende Instanz, die die Möglichkeiten des Agenten beschränkt.

Wie in Abschnitt 5.2.3 ausgeführt wird, ist die Signierung eines Agenten nicht ausreichend, wenn dieser in einen Kontext (z.B. ein HTML-Dokument) eingebettet ist, der nicht authentifiziert wird. Darüberhinaus können Agenten in einem vom Autor nicht intendierten Kontext ausgeführt werden.

In [Neu98] kommt die Untersuchung zu dem Schluß, daß kurz- bis mittelfristig nicht damit zu rechnen ist, daß ein größerer Teil der Agent signiert werden wird. Der damit

verbundene Aufwand, resultierend insbesondere aus dem inkompatiblen Format der beiden am häufigsten verwendeten Browsern, steht in keinem Verhältnis zu dem erzielten Nutzen. Mit einer Änderung ist somit erst dann zu rechnen, wenn die aufgetretenen Schadensfälle eine Höhe erreicht haben, die es zwingend erforderlich macht, Schutzmaßnahmen zu ergreifen.

Dem Microsoft Dokument [Mic96] ist die Aussage zu entnehmen, daß „*based on their experience with and trust in the software publisher* [...]“ Anwender entscheiden können, ob sie ein Programm ausführen sollten. Microsoft mutet dem Anwender mit seiner Sicherheitstechnologie Authenticode damit zu, in einem trial-and-error-Verfahren festzustellen, welchen Programmen vertraut werden kann. Für den Fall, daß ein Programm maliziöses Verhalten aufweist, führt das Dokument aus, daß der Anwender die Möglichkeit hat, gegen den Unterzeichner des Programms vorzugehen. Dabei wird übersehen, daß mittels von Verschleierungs- und Verzögerungstechniken sowohl der Identitäts- als auch der Zeitbezug aufgehoben werden kann. Damit ist ein zeitlicher Zusammenhang zwischen der Ausführung eines Agenten in einem Netzwerk und der Feststellung eines maliziösen Aktion auf dem Rechensystem nicht mehr gegeben. Als Beweismittel erforderliche Spuren der maliziösen Aktionen werden von „intelligenten“ Angreifern während und nach Durchführung der Aktionen entfernt. Selbst wenn sich der Produzent eines maliziösen Agenten identifizieren läßt, stehen dem Anwender damit keine Beweismittel zur Verfügung, mit deren Hilfe die Schuldfrage gerichtlich geklärt werden kann.

Die durch den Autor durchgeführte Kennzeichnung (sicher für die Initialisierung und/oder Scripting) seines Controls ist eine unzureichende Entscheidungsbasis für den Internet Explorer. Jeder Autor hat die Möglichkeit, sein Control entsprechend zu kennzeichnen, ohne daß im Anschluß eine Überprüfung erfolgt, in der festgestellt wird, ob die Kennzeichnung rechtens ist. Ein weiteres Mal obliegt es dem Anwender damit, festzustellen, welches Control maliziöse Operationen durchgeführt hat.

Wie GARFINKEL und SPAFFORD in [GS97] ausführen, kann ein signierter Code darüberhinaus zu Zwecken verwendet werden, für die er nicht implementiert wurde. Agenten können mit ungeeigneten Daten versorgt werden, die Sicherheitslücken innerhalb des Agenten offenbaren.[9] Diese können anschließend von Angreifern ausgenutzt werden. Wird der Agent innerhalb eines HTML-Dokumentes lediglich referenziert, so daß er vom Hersteller bezogen wird, kann der Angreifer in das HTML-Dokument die notwendigen Daten eintragen, die es ihm erlauben, die entdeckten Sicherheitslücken innerhalb des Agenten auszunutzen.

Authenticode 2.0 erlaubt, wie in Abschnitt 5.2.1 bereits beschrieben, die Anbringung eines Zeitstempels an einem signierten Agenten, mit der Folge, daß die Signatur zeitlich uneingeschränkt gültig bleibt. Daher, können Angreifer mit Zeitstempeln versehene Agenten beliebig lange analysieren, um Schwachstellen in der Implementation zu

[9] So wird in [GS97] darauf hingewiesen, daß viele ActiveX-Agenten kollabieren, wenn ihnen Datenstrukturen übergeben werden, die eine angenommene Länge überschreiten. In den letzten Jahren basierten eine Reihe von Angriffen im Internet auf Stack-Overflows, so daß es nicht unwahrscheinlich ist, daß anzunehmen ist, daß diese Angriffstechniken auch im Zusammenhang mit Agenten zu Einsatz kommen.

entdecken.

Eine weitere Neuerung innerhalb Authenticodes ist die Überprüfung, ob ein Zertifikat seit der Ausstellung durch die CA und dem Ende seiner Gültigkeit zurückgezogen worden ist. Die Überprüfung wird jedoch nur für Agenten durchgeführt, die bislang nicht auf dem lokalem System des Anwenders installiert sind. Bereits installierte Agenten sind von der Überprüfung nicht betroffen. Damit gelten lokal installierte Agenten weiterhin als vertrauenswürdig, auch wenn deren Zertifikat in der Zwischenzeit zurückgezogen worden ist. Ein weiteres Problem ist, daß die korrespondierende Option, wie bereits erwähnt, in der Standardeinstellung des Internet Explorers deaktiviert ist.

Von der Rücknahme eines Zertifikates sind alle mit diesem Zertifikat signierten Agenten betroffen. Dieses ist im Falle von Unternehmen, die ein Zertifikat für ihre Agenten verwenden, ein Problem. Wird nach Veröffentlichung eines Agenten festgestellt, daß dieser Implementationsfehler beinhaltet, die zu Angriffen mißbraucht werden können, reicht eine Entfernung des Agenten vom Webserver des Unternehmens nicht aus. In der Zwischenzeit kann der Agent auf vielen anderen Servern angeboten werden. Das Problem ließe sich nur mit der Rücknahme des Zertifikates lösen, was für ein Unternehmen jedoch eine inakzeptable Lösung ist. Die unzureichende Versionskontrolle der ActiveX-Agenten stellt in diesem Zusammenhang ein weiteres Problem dar. Die Angabe der Version für einen ActiveX-Agenten erfolgt innerhalb der Auszeichnung (engl. *tag*) für den Agenten im HTML-Dokument. Ein Angreifer kann somit stets eine höhere Version für das fehlerhafte Control auswählen, mit der Folge, daß die auf dem Rechensystem des Anwenders befindliche Version durch seine Version ersetzt wird. Eine feiner granulierte Sicherheitspolitik ist nur dann gegeben, wenn jedes Control mittels eines anderen Zertifikates signiert wird.

```
<OBJECT ID="Exploder1" WIDTH=86 HEIGHT=31
 CODEBASE="Exploder.ocx"
 CLASSID="CLSID:DE70D9E3-C55A-11CF-8E43-780C02C10128">
    <PARAM NAME="_Version" VALUE="65536">
    <PARAM NAME="_ExtentX" VALUE="2646">
    <PARAM NAME="_ExtentY" VALUE="1323">
    <PARAM NAME="_StockProps" VALUE="0">
</OBJECT>
```

Abbildung 5.7.: Die Auszeichnung für einen ActiveX-Agenten besteht aus einer Objekt-Id, einer Referenz auf den Code (CODEBASE) und einer Klassen-Id, die den Agenten eindeutig identifiziert. Der Autor des Dokumentes, in dem der Agent eingebettet ist, gibt weiterhin in der Auszeichnung die Versionsnummer des Agenten an. Aus dieser wird bestimmt, ob ein möglicherweise bereits lokal vorhandener Agent aktuell ist oder nicht.

Die im Browser enthaltenen Zertifikate können vom Benutzer nur unter erhöhtem Aufwand überprüft werden. Jeder Anwender geht davon aus, daß die enthaltenen Zertifikate gültig und vertrauenswürdig sind. In Anbetracht der Tatsache, daß Browser häufig aus dem Internet bezogen werden, ist davon auszugehen, daß es Angreifern möglich ist, Kopien des Internet Explorers anzubieten, die mit von ihm erstellten Zertifikaten ausgestattet sind. Anschließend empfängt der Benutzer mit dem Browser Agenten, die mit einem Schlüssel signiert worden sind, die von einer CA zertifiziert wurden.

Ein weiteres Problem der Authenticode Technologie liegt in der unzureichend vorhandenen Infrastruktur. Lediglich in den Vereinigten Staaten ist es bislang ohne Schwierigkeiten möglich, ein Zertifikat durch eine CA zu erhalten. In anderen Ländern, wie z. B. der Bundesrepublik Deutschland, befinden sich die entsprechenden Strukturen erst im Aufbau, so daß die Authenticode Technologie bereits aus strukturellen Gründen nicht eingesetzt werden kann. Der Bezug eines Zertifikates aus den Vereinigten Staaten für ein ausländisches Unternehmen ist nicht möglich.

Der Bezug von ActiveX-Agenten kann bereits zu einem Denial-of-Service-Angriff werden, ohne daß der Agent zur Ausführung gelangt. Das Agentenkonzept mit der Signierung mittels der Authenticode Technologie erfordert, daß der Agent zunächst aus dem Netz bezogen und der Anwender im folgenden Schritt befragt wird, ob der Agent installiert werden soll. Das Beispiel des von Microsoft betriebenen Servers MSNBC (`http://www.msnbc.com`) zeigt, daß wenn nahezu in jedem Dokument ein bestimmter ActiveX-Agent enthalten ist, der Anwender geradezu genötigt wird, diesen zu installieren [Smi98]. Jedes Dokument mit enthaltenem Agenten hat zur Folge, daß stets der Agent aus dem Netz geladen und der Anwender befragt wird, ob dieser lokal zu installieren ist. Authenticode stellt keine Mechanismen zur Verfügung, mittels derer festgelegt werden kann, daß bereits vormals abgelehnte ActiveX-Agenten nicht erneut aus dem Netz bezogen werden.

In die Sicherheitsbeurteilung eines ActiveX-Agenten fließt resultierend aus der Authenticode Technologie nur die Signatur des Agenten ein, nicht jedoch das System, der Web-Server, von dem das Dokument und das in ihm eingebettete Control stammt. Mit der Markierung eines ActiveX-Agenten als „safe for initializing“ und „safe for scripting“, sollte das umgebende Dokument keine Sicherheitsprobleme mit sich bringen können. Alle zum Thema Authenticode publizierten Dokumente[10] schweigen sich jedoch darüber aus, wie ein Autor sicherstellen kann, daß sein ActiveX-Agent sich in allen Umgebungen gutartig verhält. Beispiele aus der Vergangenheit zeigen, daß kommerzielle Controls in einer entsprechenden Umgebung zu maliziösen Aktionen bewegt werden können [Smi96, Lam97, Hop97b].

In einem Artikel zum Internet Explorer weist Microsoft daraufhin, daß Zertifikate nur Softwareproduzenten erteilt werden, die den definierten Graden an Integrität und Sicherheit in ihren Programmen entsprechen [Mic97a]. Das Beispiel des ActiveX-Agenten *Exploder* hat gezeigt, daß die Aussage von Microsoft falsch ist. Desweiteren geht aus den

[10] Siehe beispielsweise [Joh96, Mic96].

Dokumenten zur Authenticode-Technologie hervor, daß bei der Erteilung von Zertifikaten keine Programme des Antragstellers auf ihre Wirkung hinsichtlich Integrität und Sicherheit untersucht werden.

Grundsätzlich kann nicht impliziert werden, daß ein signierter Code sicher ist. Die Signatur kann hierüber keinen Aufschluß geben. Der Anwender kann lediglich entscheiden, ob er der Person vertraut, die eine Software signiert hat.

ActiveX Controls Ein wesentliches Problem von ActiveX-Agenten ist der Umstand, daß dem Anwender keine Informationen zur Verfügung stehen, welche Funktionalität ein Agent beinhaltet. Im Falle der ActiveX-Agenten ist dieses besonders schwerwiegend, da die Agenten ohne Einschränkungen Zugriff auf sämtliche Funktionen des Rechensystemes haben.

Die weit verbreitete Plattform Windows 95 bzw. 98 bietet keine Sicherheitsmechanismen, mit denen Angriffe aus dem Internet abgewehrt werden können. Windows NT bietet eine Sicherheitsarchitektur, die nicht verhindert, daß ActiveX-Agenten aus dem Internet mit den Rechten des Benutzers ausgestattet werden und entsprechend mit diesen Rechten Angriffe durchführen können.

ActiveX-Agenten haben keine inhärenten Schutzmechanismen gegen Pufferüberläufe, so daß datengesteuerte Angriffe, wie sie in Abschnitt 4.4.1 dargestellt wurden, leicht durchführbar sind.

ActiveX-Agenten sind damit eine höchst unsichere Technologie. KOKE kommt zu dem Schluß, daß vom „kommerziellen Gebrauch dieser Technologie im öffentlichen Internet [...] außer in Sonderfällen abgesehen werden [sollte]“ [Kok97]. COHEN kommt in [Coh97] zu einem ähnlichem Schluß. Danach sind ActiveX Agenten eine inhärent unsichere Technik, in der jeder Agent potentiell maliziös ist.

Java Capabilities Die Capabilities für Java-Agenten werden im Falle des Internet-Explorers für die gesamte Lebenszeit des Agenten erteilt. Die Gültigkeit der Rechte werden nicht auf die Codestellen begrenzt, für die die Rechte zwingend erforderlich sind. Damit verstößt das Capability-System von Microsoft gegen das „least privilege“-Prinzip, nach dem zusätzliche Rechte nur für die Zeit erteilt werden sollten, zu denen die Rechte absolut erforderlich sind. Im Code des Agenten sind keine Aufrufe an ein API enthalten, die die Rechte des Agenten erfordern.

In Verbindung mit dem Zonenkonzept des Internet Explorers können Java Agenten automatisch zur Ausführung gelangen, wenn die für ihre Ausführung erforderlichen Rechte, durch die Rechte der Zone abgedeckt werden, aus der das den Agenten enthaltene Dokument stammt. Damit ist verbunden, daß die Ausführung des Agenten für den Nutzer möglicherweise unbemerkt erfolgt. In Verbund mit dem nicht vorhandenen Auditing-Informationen ist dieses eine nicht wünschenswerte Vorgehensweise.

Zonenkonzept - Microsoft mißlingt es mit dem Internet Explorer deutlich herauszuarbeiten, welche Unterschiede zwischen den verschiedenen Sicherheitszonen bestehen. Es wird dem Anwender nicht deutlich, welche Konsequenzen für die Systemsicherheit beim Wechsel zwischen den unterschiedlichen Sicherheitszonen bestehen [PSK98].

Das Zonenkonzept erlaubt es dem Anwender, den verschiedenen Zonen Servern zuzuteilen. Die Adressen der Server können zu sog. Vertrauenslisten zusammengefaßt werden (engl. protection lists) [Neu98]. In Abhängigkeit von der Herkunft der Dokumente können diesen zusätzliche Rechte eingeräumt werden. Die Angabe der Adressen sollte in Form der numerischen IP-Adressen erfolgen. Nur auf diesem Wege ist sichergestellt, daß auch der intendierte Server angesprochen wird. Namensähnlichkeiten können zu irrtümlichen Eintragungen führen. Eine andere Gefahr stellt DNS-Spoofing dar, das eine Adresse auf eine falsche IP-Adresse abbildet.

Der Anwender wird beim Wechsel zwischen den unterschiedlichen Sicherheitsstufen für die Sicherheitszonen darauf hingewiesen, daß beispielsweise für Agenten aus dem Internet mindestens die Sicherheitsstufe „Mittel“ aktiviert sein sollte. Bei einem Wechsel der Sicherheitsstufe auf benutzerdefiniert unterbleibt diese Meldung. Dem ungeschulten Nutzer wird damit der Eindruck vermittelt, daß eine Manipulation der Internetoptionen im benutzerdefinierten Modus weniger kritisch als ein Wechsel in die Sicherheitsstufe „Niedrig“ ist.

5.2.2. Communicator

Der von Netscape entwickelte Browser Communicator unterstützt in der Standardausführung lediglich Java-Agenten. Über die Integration geeigneter Plug-ins können auch andere Agentensysteme – wie ActiveX-Agenten – unterstützt werden.

Der Communicator bietet einige wenige Sicherheitsoptionen an, die dem Anwender die Möglichkeit geben, auf die vom Browser durchgesetzte Sicherheitspolitik Einfluß zu nehmen. Zu den veränderbaren Merkmalen zählt die Aktivierung bzw. Deaktivierung von Java/JavaScript, die De-/Aktivierung Hinweismeldungen und die Möglichkeit, Zertifikate zu betrachten.

Mit Version 4.0 des Communicators ist ein Administrationstool entwickelt worden, das es Systemadministratoren erlauben soll, eine netzweit verbindliche Sicherheitspolitik für alle installierten Communicator durchzusetzen. Die Anwendung *Mission Control* soll dem Administrator ermöglichen, zentral die Festlegung der Sicherheitseigenschaften aller im Netz installierten Browser durchzuführen. Der Administrator soll definieren können, welche Zertifikate als vertrauenswürdig anerkannt und welche zusätzlichen Rechte den Agenten eingeräumt werden, die vertrauenswürdige Zertifikate referenzieren. Er soll die Unterstützung von Java-Agenten und JavaScript Dokumenten aktivieren bzw. deaktivieren. Mission Control steht für alle Windows-Plattformen, MacOS und neun Unix-Derivate zur Verfügung [Net98b].

Java-Agenten werden im Communicator innerhalb der von Sun definierten Sandbox ausgeführt. Eine detaillierte Beschreibung dieser Sicherheitsdomäne erfolgt im Ab-

schnitt 5.3.1. Darüberhinaus hat Netscape eine auf Capabilities basierende Sicherheitsarchitektur entwickelt, die im folgendem Abschnitt vorgestellt wird.

Abbildung 5.8.: Die Sicherheitseinstellungen, die innerhalb des Netscape Communicators im Menuepunkt Sicherheit zusammengefaßt sind, beziehen sich im wesentlichen auf kryptografische Aspekte. An dieser Stelle werden Paßwörter festgelegt, die Konfiguration von SSL vorgenommen und es erfolgt hier die Verwaltung der Zertifikate des Browsers.

Object Signing

„Object Signing“ ist das von Netscape entwickelte Verfahren zur Signierung von Objekten. Damit verfügt der Communicator über ein Signierungsverfahren, das vergleichbar mit der von Microsoft entwickelten Authenticode-Technologie ist. Derzeit unterstützt der Communicator die Signierung von Java-Agenten, JavaScript und Plug-ins. Object Signing soll ebenso wie Authenticode die Identifizierung des Unterzeichners eines Agenten (Objektes) erlauben und die Manipulation von Agenten offenbaren [Net97]. Eine kontrollierte Ausführung von Java-Agenten und JavaScript-Dokumenten soll hiermit erreicht werden.[11]

Object Signing wird seit der Version 3.0 des Communicators unterstützt. Mittels der Object-Signing-Technologie signierte Java-Agenten können über die durch die Sandbox definierten Rechte weitere erhalten. Auch in der Lösung von Netscape werden Capabilities genutzt, um die Vergabe zusätzlicher Rechte zu kontrollieren. Der Besitz einer Capability verschafft dem Agenten einen erweiterten Zugriff auf die von der Umgebung angebotenen Ressourcen. Object Signing nutzt Techniken, die auf dem Standard X.509 v3 für Zertifikate beruhen und folgt dem PKCS #7 Standard für die Signierung und Verschlüsselung beliebiger Daten.

[11] Es sei darauf hingewiesen, daß Plug-in Agenten in die kontrollierte Ausführung nicht einbezogen werden.

Signierte Agenten nutzen die „Netscape Capabilities API“, um über die zusätzlichen Rechte verfügen zu können. Der sich in Ausführung befindliche Agent existiert in der Netscape Sicherheitsarchitektur weiterhin in der Sandbox. Die „Netscape Capabilities API“ ermöglicht es ihm aber, über die Sandbox hinausgehende Rechte zu erhalten.

Die Sicherheitsarchitektur unterscheidet Instanzen aus drei unterschiedlichen Klassen: Prinzipale (engl. principals), Objekte (engl. targets) und Privilegien (engl. privilegies). Prinzipale sind Instanzen der Klasse `Principal` und repräsentieren Subjekte, die auf Objekte zugreifen wollen. Ein Prinzipal wird durch ein Zertifikat repräsentiert. Objekte sind Instanzen der Klasse `Target` und sind abstrakte Repräsentationen der Systemressourcen. Der Zugriff von Prinzipalen auf Objekte erfordert eine Reihe von Privilegien, die Instanzen der Klasse `Privilege` sind.

Wird ein Java-Agent aus einem Netz geladen, wird er in Form einer JAR-Datei[12] übermittelt, so er signiert ist. Nicht signierte Agenten werden in der bekannten Form der `Class`-Datei übertragen. Die JAR-Datei enthält neben der Signatur des Agenten das durch eine CA signierte Zertifikat des Produzentendes Agenten. Das Zertifikat enthält den öffentlichen Schlüssel des Agentenproduzenten. Wird ein signierter Agent empfangen, werden im ersten Schritt alle Java-Klassen aus der JAR-Datei extrahiert. Jede in dem Archiv enthaltene Klasse kann von einem anderen Prinzipal unterzeichnet sein. Innerhalb des Archives existiert ein Verzeichnis `META-INF`, daß alle korrespondierenden Zertifikate enthält. Die Datei `manifest.mf` enthält die Hashwerte der im Archiv enthaltenen Java Klassen. Ein Beispiel einer solchen Datei ist in Abbildung 5.9 zu sehen.

```
Manifest-Version: 1.0
Created-By: Zignature Shell (zigbert 0.5)
Comments: PLEASE DO NOT EDIT THIS FILE. YOU WILL BREAK IT.

Name: netscape/evang/demo/filesec/FileSecDemo.class
Digest-Algorithms: MD5 SHA1
MD5-Digest: l1xKyJAif3eK6UP6+9VHjw==
SHA1-Digest: NYZ5IRJN6AmTkoylKBUwrXgPa3A=

Name: NetscapeApplet.class
Digest-Algorithms: MD5 SHA1
MD5-Digest: Nu3mIXv5XfufcqAyuIzN5g==
SHA1-Digest: 2nabDg8tlM9HAxD5iZe2xt0xIj0=
```

Abbildung 5.9.: Ein Beispiel für eine `manifest.mf`-Datei.

Für alle signierten Agenten verifiziert der Communicator die Signatur und die Integrität des Agenten. Damit wird sichergestellt, daß der Agent vom Besitzer des Zertifikates unterzeichnet und der Agent seit der Signierung nicht modifiziert wurde. Wurde

[12] Das Format der JAR-Datei entspricht dem `ZIP`-Format.

der Agent zeitlich nach der Signierung manipuliert, wird er ebenso vollständig in der Sandbox ausgeführt wie unsignierte Java-Agenten.

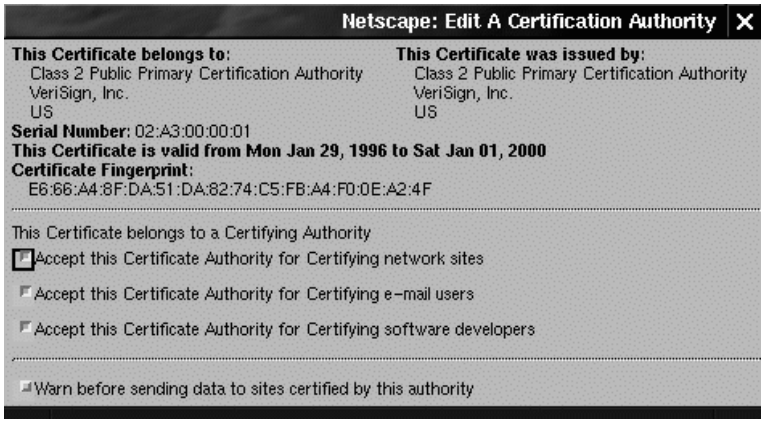

Abbildung 5.10.: Im Netscape Communicator lassen sich erweiterte Informationen zu Zertifikaten darstellen. Neben der durch das Zertifikat repräsentierten Identität sind Informationen über die CA enthalten, ein kryptografischer Fingerabdruck (engl. *fingerprint*) und die Dauer der Gültigkeit des Zertifikates.

Signierte Agenten können eine Reihe von Privilegien anfordern. Im Zuge dessen wird dem Anwender durch den Communicator eine Dialogbox präsentiert. Dieser kann der Anwender die folgenden Informationen als Grundlage seiner Vertrauensentscheidung entnehmen:

Wer hat den Java-Agenten unterzeichnet? Im Falle eines signierten Agenten ist die Identität des Unterzeichners bekannt, so daß basierend auf dieser Information eine Entscheidung getroffen werden kann. Agenten, die keine oder eine ungültige Signatur besitzen, werden ausschließlich in der Sandbox ausgeführt.

Welche Aktionen werden durchgeführt? Aufgrund einer textuellen Beschreibung der Aktionen, die der Agent durchzuführen hat, kann eine Entscheidung getroffen werden. Die Beschreibung beruht auf einer Untersuchung des Agentencodes durch den Communicator und erfordert keine durch den Produzenten bereitgestellte Beschreibung.

Wurde der Agent manipuliert? Mit der Überprüfung der Signatur eines signierten Agenten wird gleichzeitig seine Integrität überprüft. Wurde ein Agent nach Erstellung der Signatur manipuliert, wird er innerhalb der Sandbox ausgeführt und kann keine zusätzlichen Rechte erhalten [Net98c].

Der Anwender hat zu diesem Zeitpunkt die Möglichkeit, das Zertifikat des Agenten einzusehen und die Privilegien zu gewähren oder abzulehnen. Werden die Privilegien

dem Agenten gewährt, erhält er keinen automatischen Zugriff auf die Targets für die er autorisiert wurde. Für jeden Zugriff muß stattdessen explizit eine entsprechende Capability vom Privilegien-Manager angefordert werden. Dieses erfolgt mittels der Methode `enablePrivilege` und der entsprechenden Capability als Parameter. Wird eine Capability nicht gewährt, generiert der Privilegien-Manager eine Ausnahme vom Typ `ForbiddenTargetException`.

Die Erteilung einer Capability an einen Agenten erfolgt lediglich einmal während seiner Lebenszeit. In nachfolgenden Zugriffen auf Targets wird der Anwender nicht erneut nach den bereits erteilten Capabilities befragt. Weiterhin ist hingegen der Aufruf der Methode `enablePrivilege()` am Privilegien-Manager notwendig [Net98a]. Der Privilegien-Manager erkennt anhand der dem Agenten zugeordneten Listen erteilter Privilegien, daß dem Agenten die Capability bereits erteilt worden ist und übergibt dem Aufruf eine neue Instanz die Capability.

Die Erteilung der Rechte für Java-Agenten erfolgt temporär. Der Agent behält die ihm erteilten Recht bis zur Termination des Browsers. Über eine Schaltfläche hat der Benutzer zusätzlich bei der Rechteerteilung die Wahl, die Entscheidung zu persistieren. In diesem Fall wird der Anwender bei dem erneuten Laden des Agenten nicht konsultiert. Vielmehr wird der Agent sofort ausgeführt und die ihm in der Vergangenheit erteilten Rechte werden ihm erneut gewährt.

Das Privilegienmodell des Communicators ist von wesentlich feinerer Granularität, als dieses für Java-Agenten im Internet Explorer der Fall ist. Eine korrekte Implementation der Capability-Architektur gewährleistet, daß der privilegierte Zugriff auf jene Zeilen des Quelltextes begrenzt werden kann, die diese benötigen.

Jede Java-Klasse kann von mehreren Entitäten signiert werden. Es existiert eine Zugriffsmatrix, anhand der die Policy-Engine entscheiden kann, mit welchen Rechten eine Entität Zugriff auf eine Ressource erhält.[13]

Erst zukünftige Versionen des Communicators werden den von Sun mit Java 2 spezifizierten Standard für Capabilities entsprechen.

Plug-ins

Plug-ins sind native Anwendungen, die im jeweiligen Maschinencode der entsprechenden Maschine vorliegen. Sie unterliegen bezüglich ihrer Ausführung auf dem Rechensystem keinerlei Einschränkungen. Es existiert keine Sicherheitsarchitektur, die die Ausführung von Plug-ins sichert. Ebenso wie ActiveX-Agenten agieren Plug-ins wie normale Anwendungen.

Eine besondere Eigenschaft des Communicators ist die automatische Installierung neuer Plug-ins, die mit der Option „AutoInstall“ in der Standardinstallation des Browsers bereits aktiviert ist. Hat der Anwender im Zuge der Installation eines signierten Plug-ins jenes Plug-in als vertrauenswürdig erklärt, werden nachfolgende Versionen des Plug-ins

[13] Siehe weiteres in [WBDF97].

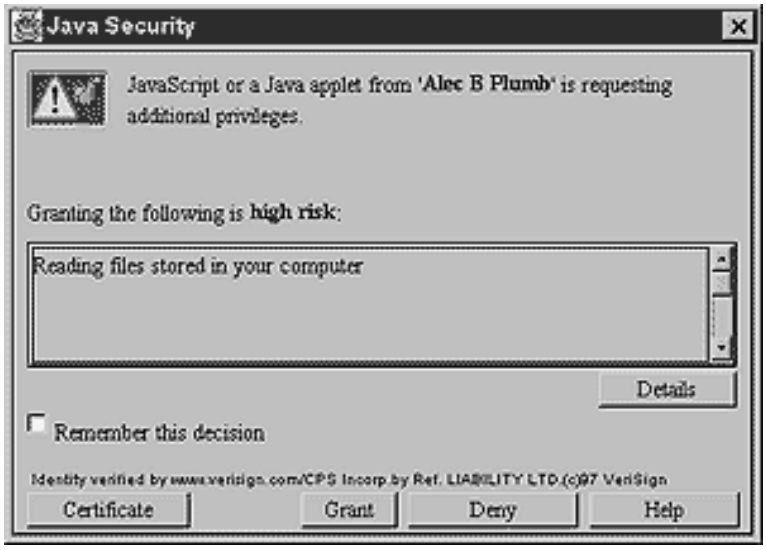

Abbildung 5.11.: Im Falle eines signierten Agenten, der erweiterte Rechte benötigt, wird eine Dialogbox präsentiert, die dazu auffordert, die Rechte zu erteilen oder zu verweigern. Die Entscheidung kann durch mittels eines Schalters persistiert werden.

automatisch ohne Einbeziehung des Anwenders installiert [Net97].

Beurteilung

Die Sicherheitsarchitektur des Communicators weist mit Object Signing einen ähnlichen Mechanismus auf, wie dieser im Sicherheitskonzept des Internet Explorers in der Kombination aus Authenticode und Java Capabilities zu finden ist. Ein Gegenstück zu den Sicherheitszonen im Explorer gibt es im Communicator nicht. In Bezug auf die Angemessenheit der Architektur folgt daraus, daß auch der Communicator die aus dem Agenten-Paradigma resultieren Gefahren mit seinen Sicherheitsmechanismen nicht angemessen reflektiert. Mit den Plug-ins unterstützt der Browser eine Technologie, die mit ActiveX-Agenten vergleichbar ist. Beide Technologien erfüllen keine der in Kapitel 2 aufgestellten Sicherheitskriterien. Eine Einbettung der Plug-ins in die Sicherheitsarchitektur des Browsers ist nicht gegeben.

Ein weiterer Mangel des Communicators besteht in der Notwendigkeit, daß für die Betrachtung der mitgelieferten Dokumentation JavaScript zu aktivieren ist. In einem Communicator, in dem der JavaScript aus Sicherheitsgründen deaktiviert ist, können keine Informationen zum Communicator eingesehen werden. Auch wenn durch die Aktivierung von JavaScripts keine Sicherheitsprobleme entstehen solange der Rechner nicht mit dem Internet verbunden ist, ist es als kritisch zu werten, daß der Aufruf der Dokumentation Technologien erfordert, die Sicherheitsrisiken mit sich bringen. Positiv ist hingegen zu werten, daß der Communicator mit lokaler Dokumentation ausgeliefert wird,

sie damit nicht länger aus dem Internet bezogen werden muß. Es gelten jedoch die gleichen Mängel wie für den Internet Explorer; auch die Dokumentation des Communicators trägt der Wichtigkeit der Sicherheit nicht im ausreichendem Maße Rechnung.

Mit Version 4.05 des Communicators beinhaltet die Implementation der JVM das Package `netscape.secfile` mit dem Java-Agenten in einem speziellen Verzeichnis Dateien lesen und schreiben können. LaDue hat mit dem Java-Agenten `DiskHog` gezeigt, daß der Communicator Agenten damit die Möglichkeit eröffnet wird, einen Denial-of-Service-Angriff durchzuführen, der das Ziel hat, die Festplatte des Zielsystemes zu füllen [LaD98].

Der Communicator weist ein weiteres wesentliches Sicherheitsproblem auf. Signierte Agenten können auf dem Weg bis zum Zielsystem manipuliert werden. Der Verifikationsmechanismus für signierte Agenten erkennt die Manipulation, lehnt den Agenten jedoch nicht ab. Stattdessen wird der Agent innerhalb der Sandbox ausgeführt, ohne daß der Agent erweiterte Rechte erhält. Einen Hinweis, daß der Agent manipuliert worden ist, wird nicht gegeben. Angreifern wird damit die Möglichkeit eröffnet, mit manipulierten Agenten zu experimentieren und die Ergebnisse auszuwerten.

Es wurde bereits dargestellt, daß die Sicherheitsmechanismen des Communicators durch Parameter beeinflußt werden können. Die Standardeinstellungen, so wie sie nach der Installation des Browsers vorzufinden sind, sind für ein sicheres System ungeeignet. Eine Reihe von Optionen wären zu de- bzw. zu aktivieren, so ein sicheres Browsen im Netz gewährleistet sein soll. Aktiviert sind die Optionen Java, JavaScript, Autoinstall, Cookies. Deaktiviert ist die Option „Warnen vor der Akzeptanz von Cookies“.

Im Zuge der Installation erfolgt kein Hinweis, daß es Aufgabe des Nutzers ist, die oben genannten Punkte zu kontrollieren und sie gegebenenfalls seinen Sicherheitsanforderungen anzupassen. Die Optionen Java, JavaScript, AutoInstall und Cookies sind im Menue „Voreinstellungen“ unter dem Punkt „Erweitert“ zu finden. Aus der Tatsache, daß es gleichzeitig im Menue „Communicator“ einen Menueeintrag „Sicherheit“ gibt, ließe sich für einen Anwender der Schluß ziehen, daß die Optionen außerhalb des Sicherheitsmenues nicht sicherheitsrelevant sind. Es ist offensichtlich, daß dieses ein falscher Schluß ist. Die Dokumentation enthält keinen Hinweis darauf, daß sicherheitsrelevante Einstellungen auch in anderen Menues vorgenommen werden müssen. Aus dem Menue Voreinstellungen gibt es keinen Zugriff auf eine kontextsensitive Hilfe, die Informationen bereithält, auf denen der Anwender seine Entscheidungen bezüglich der Optionen festlegen kann.

Mit Version 4.0 des Communicators stellt Netscape die Funktion der automatischen Aktualisierung bereit. Bei Aktivierung der Funktion werden neue Versionen des Browsers und der installierten Plug-ins automatisch aus dem Internet bezogen. Im Zuge dessen wird dem Anwender ein Zertifikat präsentiert, das für diesen als Grundlage dienen soll, ob er die Aktualisierung gestattet [PSK98].

Java Capabilities Daneben wurde mit Version 4.x des Communicators eine Sicherheitsarchitektur für Java-Agenten entwickelt, die es ermöglicht, Java-Agenten nur jene Rechte zuzuteilen, die diese für die Durchführung ihrer Aufgaben benötigen. Entscheidungsgrundlage für den Anwender ist dabei neben der digitalen Signatur des Agenten eine textuelle Beschreibung der Funktionalität, die durch den Communicator aus dem Bytecode des Agenten ermittelt wird. Die textuelle Beschreibung der Funktionalität von Java-Agenten ist ein Schritt in die richtige Richtung. Es bleibt allerdings festzuhalten, daß die Beschreibung in dieser Form wiederum nur die Entscheidung des Nutzer unterstützen kann. Sie ist keine Gewährleistung dafür, daß maliziöse Aktionen des Agenten erkannt werden. Es obliegt dem Anwender, die gelieferten Informationen zu interpretieren und entsprechend eine Entscheidung zu treffen, ob die vom Agenten durchgeführten Aktionen ein maliziöses Verhalten nach sich ziehen können. Deutlich ist bereits jetzt, daß datengesteuerte Angriffe und Diensteverweigerungs-Angriffe auf diesem Wege nicht erkannt werden können.

Das Capability-System des Communicators basiert auf digitalen Signaturen, wie sie auch in Authenticode für ActiveX-Agenten verwendet werden. Es wurde bereits in der einleitenden Beurteilung der Sicherheitsarchitektur des Communicators darauf verwiesen, daß über die digitalen Signaturen hinaus eine textuelle Beschreibung der Aktionen des Agenten als Grundlage zur Entscheidungsfindung vorhanden ist. Die generierte Beschreibung kann Hinweise enthalten, die deutlich werden lassen, daß das Verhalten des Agenten nicht seiner Beschreibung entspricht, die auf dem Quellsystem zu finden war. Es bleibt jedoch abzuwarten, inwieweit die dort enthaltenen Informationen die Semantik eines Agenten erkennen lassen.

```
<applet code="WriteFile.class" ARCHIVE="signedWriteFile.jar">
</applet>
```

Abbildung 5.12.: Dieses Auszeichnung referenziert den signierten Agenten `WriteFile` im Archiv `signedWriteFile`.

Signierte Java-Agenten durchlaufen die Verifikationskontrolle des Communicators auch dann, wenn das Zertifikat des Unterzeichners bereits ausgelaufen ist. Damit haben signierte Java-Agenten in der Sicherheitsarchitektur des Communicators eine unbegrenzte Lebensdauer. Damit trägt Netscape nicht den Untersuchungen der Kryptanalysis Rechnung, die eine ständig zunehmende Schlüssellänge fordern [BDR+96]. Angreifer können versuchen, die Signatur eines Agenten zu knacken, um anschließend Modifikationen am Agenten vorzunehmen. Anschließend wird der Agent erneut mit der ursprünglichen Signatur versehen.

Die Abbildung 5.11 weist auf ein weiteres Sicherheitsproblem im Capability-System hin. Die Dialogbox enthält den Hinweis, daß ein Java-Agent *oder* ein JavaScript weitergehende Rechte anfordert. Es bleibt somit offen, ob in dem Dokument, daß derzeit durch den Browser dargestellt wird, ein Agent oder ein Skript referenziert wird. Dem

Anwender wird die Information vorenthalten, aus welcher Klasse von Objekten das referenzierte Objekt stammt. Bereits ohne diesen weiteren Schwachpunkt ist ein fundierte Entscheidung eines Anwenders zur Ausführung eines Agenten aus Sicht der Sicherheit nicht abschließend zu treffen. Mit dem hier dargestellten Sicherheitsmangel wird eine Entscheidung für die Ausführung eines Agenten nahezu unmöglich gemacht.

Plug-ins Plug-ins sind in die Sicherheitsarchitektur des Communicators nicht eingebettet. Die von Plug-ins ausgehenden Sicherheitsrisiken werden durch die vorhandenen Sicherheitsmechanismen nicht adressiert. Plug-in-Agenten unterliegen als Agenten in Form von Maschinencode keinerlei Einschränkungen. Sie haben vollständigen Zugriff auf alle Systemressourcen und Systemfunktionen des Betriebssystems. Sie erben die Privilegien des Anwenders, unter dessen Berechtigung sie ausgeführt werden, und können diese zur Durchführung ihrer Aufgaben ausnutzen.

GARFINKEL und SPAFFORD haben in [GS97] dargelegt, welche Gefahren von Plug-In Agenten ausgehen können:

- ▷ Angriffe durch maliziöse Plug-Ins;
- ▷ Angriffe durch harmlose Plug-Ins, die vom Angreifer modifiziert wurden;
- ▷ Angriffe durch Ausnutzung von Implementationsfehler im Plug-In und
- ▷ Ausnutzung einer Programmiersprache, die durch ein Plug-In unterstützt wird.

Neben der Möglichkeit beliebigen Maschinencode ausführen zu können, können Plug-ins den Communicator beeinflussen. Einfluß kann auf die Statuszeile des Browsers ausgeübt werden, der zu entnehmen ist, welches Dokument durch einen Link referenziert wird. Damit können nicht vertrauenswürdige Server ihre Identität verschleiern.

Das Plug-in *Shockwave* von Macromedia offenbarte erstmalig, welche Möglichkeiten dem Entwickler eines Plug-ins gegeben sind. Aufgrund eines Implementationsfehlers war es den Betreibern von Web-Servern möglich, ein Shockwave-Movie zu entwickeln, welches die Mails des Anwenders auslesen konnte. Damit hat sich gezeigt, daß installierte Plug-ins auch von Außentätern genutzt werden können, um ein System anzugreifen. Ein jedes Plug-in kann zu maliziösem Verhalten verleitet werden, so es Implementations- oder Designmängel aufweist, die von Angreifern ausgenutzt werden können.

Eine Einbindung der Plug-in-Agenten in die Sicherheitsarchitektur des Communicators ist zukünftig zwingend erforderlich. Unterbleibt dieses wie in der Vergangenheit, ist eine Kompromittierung eines Systemes, auf dem Plug-in-Agenten installiert sind, eine leichte Aufgabe für Angreifer.

5.2.3. Zusammenfassung

In den vorherigen Abschnitten wurden die beiden verbreitesten Browser mit ihren Sicherheitsarchitekturen vorgestellt. Es zeigt sich, daß beide Produkte für den Einsatz von

Agentensystemen technologisch vorbereitet sind, es bestehen jedoch eine Reihe von Sicherheitsproblemen, die sowohl die Sicherheit der Umgebungssystems als auch die der Agenten beeinträchtigen. Neben den spezifischen Sicherheitsproblemen der vorgestellten Browser existieren weitere, übergreifende Probleme.

Jedes der Produkte besitzt eine spezifische Sicherheitsarchitektur, deren Aufgabe es ist, die vom Hersteller intendierte Sicherheitspolitik durchzusetzen. Unabhängig von der Qualität der Sicherheitsarchitektur ist für beide Browser das Problem gegeben, daß weder eine formale noch eine informale Spezifikation der Sicherheitsarchitektur existiert. Desweiteren wechselt die Sicherheitsarchitektur mit jeder Version. Damit variiert gleichzeitig auch stets die aktuell durchgesetzte Sicherheitspolitik. Die Parametrisierung, die am Browser vorgenommen werden kann, hat Einfluß auf die vom Browser durchgesetzte Sicherheitspolitik. Welche Auswirkungen unterschiedliche Parametrierungen auf die durchzusetzende Sicherheitspolitik haben, ist nicht transparent.

Keines der vorgestellten Browserprodukte beinhaltet eine Sicherheitspolitik, die den Schutz des Agenten vor dem Umgebungssystem umzusetzen vermag. Damit bleibt jedoch eine wichtige Sicherheitsdimension, wie in Abbildung 2.1 dargestellt, aus der Betrachtung ausgeschlossen. Im Kapitel 6 wird deutlich, daß bislang keine Sicherheitsarchitekturen existieren, die dieser Sicherheitsdimension Rechnung tragen und bereits praktisch implementiert worden sind. Den Browser-Herstellern ist hier anzulasten, daß keine Informationen verfügbar sind, denen zu entnehmen ist, daß der Schutz der Agenten vor den Umgebungssystemen kein Sicherheitsproblem ist, daß lediglich theoretischer Natur ist. Das bereits angeführte Beispiel des Brokat-Agenten weist hierauf nachdrücklich hin.

Beide vorgestellten Browser unterstützen signierte Agenten. Die dabei zum Einsatz kommenden Zertifikate sind jedoch nicht kompatibel, d. h., Zertifikate werden von der CA für einen bestimmten Browser ausgestellt und sind damit für das Konkurrenzprodukt nicht einsetzbar. Das ist ein nicht haltbarer Zustand. Zukünftige Versionen der Browser müssen einheitliche Zertifikate unterstützen, so sich der Einsatz von signierten Agenten zunehmen soll.

Ebenso wie der Internet Explorer beinhaltet das Softwarepaket Communicator eine Reihe von Zertifikaten. Hierzu zählen u. a. Zertifikate von AT&T, IBM, Thawte und VeriSign. Beachtenswert ist dabei, daß die Zertifikate von IBM und Thawte eine Gültigkeit von z. T. über 20 Jahren aufweisen (siehe Abbildung 5.13). In Hinblick auf die Entwicklung der Kryptanalysis ist das ein unverständlicher Schritt, da die heute verwendeten Schlüssellängen innerhalb kurzer Zeit nicht mehr ausreichend sind, um die damit verschlüsselten Daten zu sichern.

Ein wesentliches Problem beim Einsatz von Browsers ist der initiale Bezug. Browser befinden sich häufig bereits beim Kauf eines Computersystems auf diesem vorinstalliert. Ist das nicht der Fall oder wird eine neue Version benötigt, gibt es die Möglichkeit, einen Browser über das Internet zu beziehen. Der Bezug eines Softwareprodukts über das Internet kann jedoch nicht als sicher bezeichnet werden, solange keine Authentikation des Quellsystems und keine Überprüfung der Integrität des Softwareproduktes erfolgt. Weiterhin wäre der Aufbau eines gesicherten Kanals erforderlich, wie dieses beispielsweise

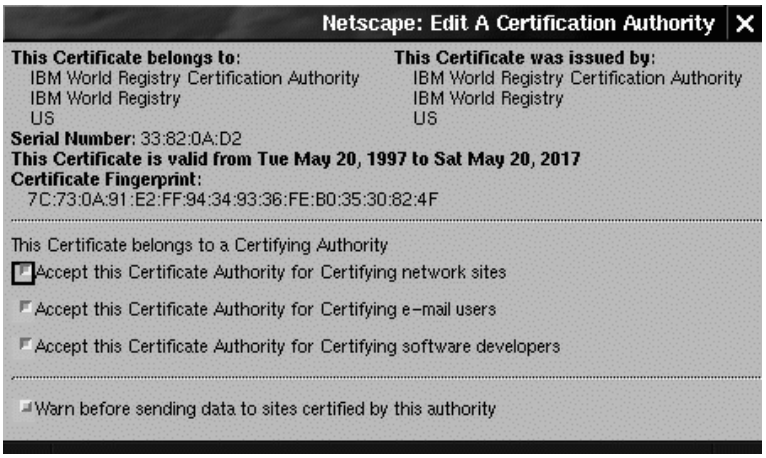

Abbildung 5.13.: Der Netscape Communicator enthält bereits eine Reihe von Zertifikaten mit unterschiedlicher Gültigkeitsdauer. Das hier dargestellte Zertifikat von IBM ist bis zum Jahre 2017 gültig.

mittels des SSL-Protokolles möglich ist. Derzeit kommen keine der erforderlichen Sicherheitmaßnahmen beim Bezug eines Browsers aus dem Internet zum Einsatz. Sollten zukünftig derartige Vertriebstechniken aufgebaut werden, besteht weiterhin das Problem, daß der Browser häufig nicht direkt von den Serversystemen des Herstellers bezogen wird. Da beide Browser unentgeltlich im Internet erhältlich sind, kann jeder Betreiber eines Servers die Browserprodukte anbieten.

Neben dem Internet besteht insbesondere für Privatanwender eine zweite Bezugsquelle in Form von CD-ROMs, die als Beilage von Zeitschriften erhältlich sind. In Anbetracht der Größe der Produkte wird dieses im Privatbereich die gängigste Quelle für den Internet Explorer und Communicator sein. Ebenso wie der Bezug über das Internet kann auch dieser Vertriebsweg nicht als sicher angesehen werden. Zwar entfällt das Problem der Authentifizierung des Quellsystems und der Sicherung der Übertragung, eine Signierung des Browsers einschließlich der Überprüfung der Browserintegrität erfolgt aber im Falle des Vetriebsweges CD-ROM ebenfalls nicht.

Der unsichere Vertriebsweg bedroht insbesondere die Sicherheitstechnologien, die in den Browsern integriert sind. Zunächst haben Angreifer die Möglichkeit, den Code der Browser dahingehend zu manipulieren, daß sicherheitsrelevante Programmteile während der Ausführung des Browsers übersprungen werden. Damit sind die Browser nicht mehr in der Lage, ihre interne Sicherheitspolitik durchzusetzen. Ein weiteres Problem resultiert aus dem Umstand, daß mit der Auslieferung des Browsers gleichzeitig die Auslieferung von Zertifikaten verbunden ist, die die wichtigsten CAs repräsentieren. Eine Manipulation der Zertifikate ist wesentlich einfacher durchzuführen, als die Manipulation des Programmcodes. Der erzielte Effekt ist jedoch ein ähnlicher. Angreifer können Zertifikate bekannter CAs gegen eigene austauschen, während dem Anwender weiterhin der

Eindruck vermittelt wird, daß die entsprechenden Zertifikate von CAs stammen.

Keiner der vorgestellten Browser hält Mechanismen bereit, mit denen eine direkte Einflußnahme auf die Ausführung der Agenten möglich ist. Eine temporäre oder permanente Terminierung eines Agenten auf kontrolliertem Wege ist nicht möglich. Wird ein Agent in einem eigenen Fenster ausgeführt, bietet sich die Möglichkeit, daß Fenster zu schließen. Grafische Benutzungsoberflächen operieren ereignisgesteuert, die Ereignisse werden an die Prozesse weitergegeben, die innerhalb eines Fensters operieren. Entsprechend obliegt es der „Entscheidung“ des Agenten, wie das Ereignis „Fenster schließen“ verarbeitet wird. Eine ähnliche Problematik stellt sich bei der Terminierung von Agenten durch das Wechseln des aktiven Dokuments, daß aktuell im Browser dargestellt wird. An Java-Agenten wird die Methode `stop()` aufgerufen, so daß das Dokument verlassen wird, in das der Agent eingebettet ist. Die Methode `stop()` kann vom Entwickler überschrieben werden, so daß auch in diesem Fall es der Entscheidung des Agenten obliegt, ob er sich terminiert.

Eine Einflußnahme auf den Kontrollfluß ist dem Anwender damit nicht möglich. Die automatische Ausführung von Agenten, die eine Dienstleistung für den Anwender erbringen, ohne daß dieser den Prozeß aktiv anstoßen muß, mag Vorteile in Hinblick auf einfach zu bedienende Systeme haben. Aus Sicht der Sicherheit ist es fatal, daß zur Laufzeit kein Einfluß auf eine Technologie genommen werden kann, deren Auswirkungen auf die Systemsicherheit derzeit nicht absehbar sind. Einzig die Termination des Browsers erlaubt dem Nutzer die Beendigung aller instantiierten Agenten.

Eine Kompromittierung der Sicherheit des Browsers als Umgebungssystem führt zu großen Problemen für ein Agentensystem. Die Ausnutzung von Sicherheitslücken innerhalb der Sicherheitsarchitektur eines Browsers kann Agenten in den Besitz von Rechten bringen, die ihnen nicht zustehen. Mangelhafte Implementierungen der JVM, die mit den Browsern ausgeliefert wird, haben in der Vergangenheit dazu geführt, daß Agenten unter Umgehung aller Sicherheitsmechanismen jede Methode der Java-API aufrufen konnten. Zuletzt ist ein derartiges Sicherheitsproblem im Juli 1998 im Netscape Communicator in der Version 4.0x entdeckt worden, der es Angreifern ermöglicht, alle Sicherheitskontrollen des Browser zu deaktivieren [Fel98].

Sicherheit ist ein umfangreiches Themengebiet und erfordert in den komplexen Strukturen heutiger Netz- und Informationsysteme ein hohes Fachwissen. In den Bemühungen der Browserhersteller, die Sicherheitsaspekte ihrer Produkte möglichst transparent werden zu lassen, erreichen sie, daß dem Anwender verborgen bleibt, welche konkrete Sicherheitspolitik in den Produkten durchgesetzt wird. Dem Anwender bietet sich die Möglichkeit, Einfluß auf eine Reihe von Sicherheitsmerkmalen zu nehmen. Welche Wirkung damit auf die Sicherheitspolitik genommen wird, bleibt jedoch unklar. Der nicht seltene Wechsel der Sicherheitspolitiken in unterschiedlichen Browserversionen stellt in diesem Zusammenhang ein nicht unwesentliches Problem dar.

Keiner der beiden vorgestellten Browser protokolliert die durch Agenten durchgeführten Operationen. Damit hat der Anwender keine Möglichkeit, Informationen darüber zu erhalten, welche Aktionen durch die Agenten durchgeführt wurden. Insbesondere für den

Internet Explorer im Verbund mit der Authenticode-Technologie ist dies ein schwerwiegender Mangel, da der Anwender im Falle maliziöser Aktionen darauf angewiesen ist, im nachhinein festzustellen, welcher Agent die entsprechenden Operationen durchgeführt hat. Das gesamte Authenticode-Konzept ist damit keine sinnvolle Sicherheitsarchitektur, da keine Protokollinformationen generiert werden, die im Falle eines Angriffes zu Beweiszwecken verwendet werden könnten.

Wie bereits ausgeführt wurde, stellt die rapide Entwicklungsgeschwindigkeit der Produkte ein Sicherheitsproblem dar. Zum einem werden damit neue Sicherheitslücken aufgetan, die aus der Implementierung neuer Funktionen resultieren. Das Funktionsspektrum der Browser erweitert sich von Version zu Version erheblich. Dabei werden nicht nur neue Funktionen implementiert, sondern auch Fremdprodukte in den jeweiligen Browser integriert. Zum anderen wirkt sich die Entwicklungsgeschwindigkeit negativ auf die Konsolidierung der Produkte aus. Die Vielzahl der in der Vergangenheit entdeckten Implementationsmängel, die nahezu alle Auswirkungen auf die Sicherheit der Produkte hatten, sind hierfür ein deutlicher Beleg.[14]

Der schnelle Ausstoß neuer Versionen hat zur Folge, daß sich eine Vielzahl unterschiedlicher Versionen eines Produkts im Einsatz befinden. Damit verbleiben fehlerhafte Versionen lange im Umlauf und eröffnen Angreifern die Möglichkeit, bereits lange bekannte und gut dokumentierte Sicherheitslücken in älteren Versionen der Browser auszunutzen, um so ein System anzugreifen. Die Hersteller versuchen diesem Problem durch die schnelle Bereitstellung eines Patches zu begegnen, der neu entdeckte Sicherheitslücken schließen soll. Das Produkt wird damit jedoch erst recht zu einem Flickwerk, zu einer Konsolidierung können diese Maßnahmen im Sinne der Softwaretechnik nicht beitragen. Darüberhinaus wird den meisten Benutzern die Entdeckung neuer Sicherheitslücken verborgen bleiben. Notwendig wäre die Fähigkeit des Browsers, einen sicheren Kanal zum Server des Herstellers aufzubauen, über den erfragt werden kann, ob neue Sicherheitsmängel entdeckt worden sind. Bei Bedarf können dann über den gleichen Kanal jene Patches übertragen werden, die die neu entdeckten Sicherheitslücken schließen sollen.

Ein weiterer wesentlicher Punkt, der sich auf die Systemsicherheit auswirkt, ist die Integration von Internet-Technologien in das Betriebssystem bzw. die Benutzungsoberfläche. Die zunehmende Integration hat das Ziel, den Zugriff auf lokale und entfernte Daten sowie die Ausführung von lokalen und entfernten Programmen vollständig transparent werden zu lassen. Diese Zielsetzung ist aus Sicht der Systemsicherheit äußerst problematisch. Eine Unterscheidung von lokalen und entfernten Programmen bzw. Daten ist wichtig für die Systemsicherheit. Eine Auflösung der Systemgrenzen löst gleichzeitig die Sicherheitsgrenzen auf. Den Anwendern wird damit jede Kontrolle über die Ausführung von Programmen genommen. Mobile Agenten kommen so vollkommen transparent und für den Anwender vollständig unsichtbar zum Einsatz. Bereits zum derzeitigen Zeitpunkt führt der Mangel an Transparenz zur Ausführung von Agenten, ohne daß der Nutzer diese wahrnimmt.

[14] Siehe hierzu auch [PSK97].

Die Einbettung eines Agenten in ein HTML-Dokument zieht das Problem nach sich, daß lediglich eine Authentifizierung des Agenten möglich ist, wenn dieser signiert ist, nicht jedoch das Dokument, in das er eingebunden ist [Hop97a]. Das hat zur Folge, daß ein eingebetteter Agent durch

- ▷ einen unsignierten Agenten,
- ▷ einen von einer anderen Entität signierten Agenten oder
- ▷ einen anderen durch die gleiche Entität signierten Agenten ersetzt werden kann [Hop97a].

Darüberhinaus können Agenten von beliebigen Dokumenten aus referenziert werden. D. h., ein Angreifer kann einen signierten Agenten in ein Dokument einbetten und eine Reihe von Parameter spezifizieren, die außerhalb des intendierten Wertebereiches liegen. Auf diese Weise können beispielsweise Pufferüberläufe provoziert werden, wenn der Agent auf dem Rechensystem eines Anwenders zum Einsatz gelangt.

Web Spoofing

FELTEN et al. haben 1997 ein Sicherheitsproblem in Bezug auf das Arbeiten mit Browsern identifiziert, dessen Lösung bis heute aussteht. Im Zusammenhang mit dem Internet ist seit langem das Problem des Spoofings bekannt. FELTEN et al. haben für das Browsen im Netz eine spezifische Form des Spoofings erkannt, das von ihnen als Web-Spoofing bezeichnet wird. Nutzern des WWW kann von Angreifern eine Scheinwelt vorgetäuscht werden, die für ein System wie die „reale“ Welt des Internets erscheint. Ein angreifendes System täuscht einem Opfersystem das reale Netz vor, in dem jeder Verweis, den das Opfersystem auswählt, zunächst jenes des Angreifers referenziert. Mittels eines Parameters bestimmt der Angreifer das eigentliche Ziel des Verweises, bezieht das ursprüngliche Dokument, manipuliert es und sendet es anschließend an das System des Opfers. Das System des Angreifers fungiert als eine Art Proxy, der für das Opfersystem vollständig transparent und damit nahezu unsichtbar ist. Der vollständige Netzverkehr des Opfersystems fließt durch das System des Angreifers, der damit als „Man-in-the-middle“ agiert. Alle vom Angreifer versandten Dokumente sind dahingehend manipuliert, daß die enthaltenen Verweise zunächst den Angreifer referenzieren. Ein Beispiel hierfür ist der Abbildung 5.14 zu entnehmen.

```
http://www.attacker.org/http://www.uni-hamburg.de
```

Abbildung 5.14.: Eine Referenz eines Dokumentes wird durch Web-Spoofing in die obige Form transformiert. Zunächst wird das angreifende Systeme kontaktiert, das die eigentliche URL als Parameter erhält.

Der initiale Verweis auf das angreifende System gelingt durch die Manipulation bestehender HTML-Dokumente. Es hat sich gezeigt, daß es möglich ist, die Web-Präsenz

bekannter Organisationen und Unternehmen anzugreifen. Die Manipulation von Verweisen bleibt wesentlich länger unentdeckt, als die Ersetzung der HTML-Dokumente, insbesondere in Hinblick darauf, daß scheinbar die gleichen Dokumente als Ergebnis der Verfolgung eines Links geliefert werden [FBDW97].

Der Angreifer, der ein Web-Spoofing durchführt, hat die vollständige Kontrolle über die projizierte Welt. SSL und andere Protokolle, die eine gesicherte Verbindung erlauben, bieten gegenüber Web-Spoofing keinen Schutz, da die Verbindung zwischen Angreifer und Opfer besteht.

Vervollständigt wird der Angriff durch JavaScript. Mittels eines Skriptes lassen sich sowohl die Status- als auch die URL-Zeile manipulieren, so daß diesen nicht zu entnehmen ist, daß das Opfer einem Spoofing-Angriffes unterliegt. Die Abschaltung aller aktiven Inhalte (ActiveX, Java, JavaScript) erschwert Angriffe, eine umfassende Lösung existiert jedoch nicht [FBDW97].

5.3. Java

Java ist eine von Sun entwickelte Programmiersprache, die 1995 veröffentlicht wurde, und seitdem verstärkt in Agentensystemen eingesetzt wird. In Java entwickelte Programme sind plattformunabhängig. Sie lassen sich auf jedem System ausführen, für das eine Implementation der JVM existiert. Java beinhaltet eine Reihe von Mechanismen, die die Entwicklung von Agenten erleichtern. Ende 1998 wurde die Version Java 2 veröffentlicht. Neben den zunehmenden Funktionen hat sich mit den Versionswechseln auch die Sicherheitsarchitektur Javas verändert. In den folgenden Abschnitten wird die Evolution der Sicherheitsarchitektur nachvollzogen und in der abschließenden Beurteilung die Implikationen auf die Sicherheit von Agentensystemen aufgezeigt.

5.3.1. Sicherheitsarchitektur JDK 1.0

Bereits die erste Version der JVM umfaßte eine Sicherheitsarchitektur, die dem Einsatz von Agenten in Netzwerken gerecht werden sollte. Der wesentliche Aspekt der Sicherheitsarchitektur ist im JDK 1.0 das Konzept der *Sandbox*. Die Sandbox ermöglicht die Ausführung nicht vertrauenswürdiger Agenten in einer vertrauenswürdigen Umgebung. Damit wird von der traditionellen Sichtweise, daß einer Software vertraut werden muß, bevor sie ausgeführt wird, abgewichen. Beim Bezug eines Java-Agenten ist keine Entscheidung zu treffen, ob dieser vertrauenswürdig ist. Die JVM entscheidet autonom anhand des Ursprungs des Agenten, ob dieser als vertrauenswürdig zu betrachten ist. Das Konzept der Sandbox soll sicherstellen, daß maliziöse Agenten keinen Schaden auf Rechensystemen erzeugen können.

Die Kernkomponenten, die der Durchsetzung des Sandbox-Modells dienen, sind:

- ▷ die Robustheit der Sprache,

- ▷ der Class-Loader,
- ▷ der Verifier und
- ▷ der Security-Manager.

Alle Komponenten der Sicherheitsarchitektur wirken zusammen und setzten gemeinsam die Sicherheitspolitik der Sandbox durch.

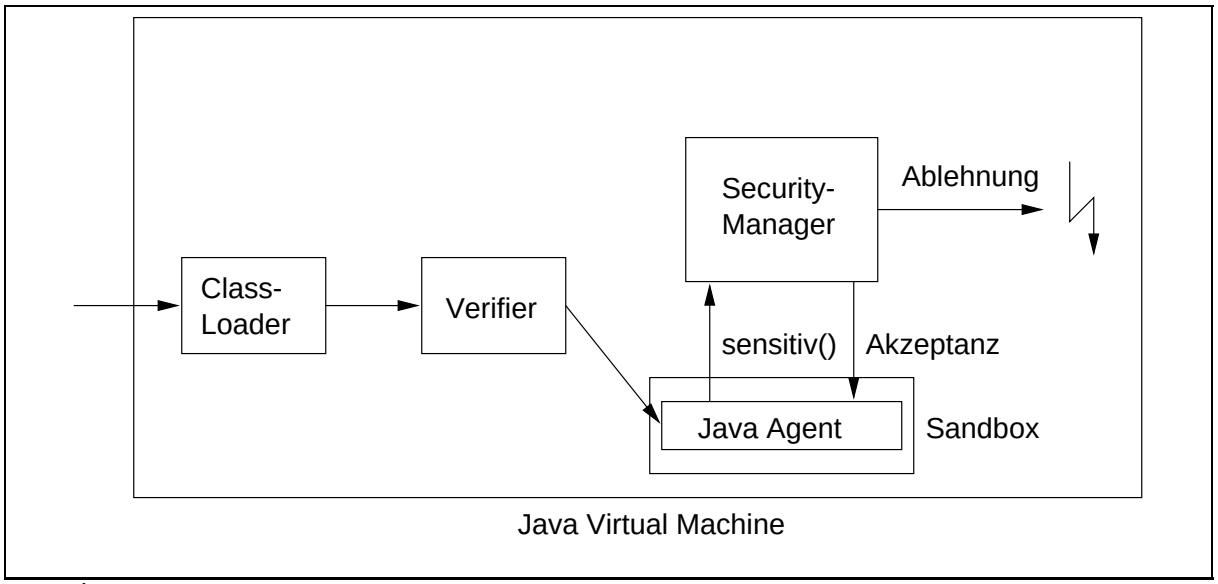

Abbildung 5.15.: Die Sicherheitsarchitektur Javas setzt sich aus mehreren Komponenten zusammen. Die erste Komponente ist das Class-Loader-Objekt, das einen Agenten bezieht. Das Class-Loader-Objekt leitet den Agenten an den Verifier weiter. Im Anschluß an dessen Überprüfungen wird der Agent in der Sandbox zur Ausführung gebracht. Im Falle eines aktivierten Security-Managers hat der Aufruf sensitiver Methoden die Konsultation des Security-Managers zur Folge. Wird der Aufruf nicht genehmigt, generiert der Security-Manager eine Ausnahme.

Sandbox

Das von Sun entwickelte Konzept der Sandbox beinhaltet, daß alle lokalen Klassen als vertrauenswürdig gelten, während alle externen Klassen als nicht vertrauenswürdig eingestuft werden. Eine Klasse gilt als lokal, wenn sie entweder Bestandteil der Systemklassen der JVM oder über die definierte Variable `CLASSPATH` zugreifbar ist. Während vertrauenswürdige Klassen vollen Zugriff auf alle Ressourcen des lokalen Systemes haben, sind die Möglichkeiten der nicht vertrauenswürdigen Klassen stark eingeschränkt. Sie laufen innerhalb der Sandbox ab, die gewährleisten soll, daß die in ihr ablaufenden Agenten keinen Zugriff auf Methoden erhalten, die zu maliziösen Aktionen genutzt

werden können. Damit werden zwei Sicherheitsdomänen anhand der unterschiedlichen Vertrauensgrade aufgebaut. Innerhalb einer Domäne haben alle Agenten die gleichen Rechte.

Zu den Beschränkungen, denen Agenten innerhalb der Sandbox unterliegen, zählen:

- ▷ kein Zugriff auf das Dateisystem,
- ▷ kein direkter Zugriff auf Speicherzellen,
- ▷ keine Erzeugung neuer Prozesse durch Agenten,
- ▷ Netzwerkverbindungen dürfen nur zum Quellsystem des Agenten aufgebaut werden,
- ▷ Interaktion mit anderen Applets ist nur innerhalb eines HTML-Dokumentes möglich und
- ▷ Agenten innerhalb der Sandbox können keine nativen Programme ausführen.[15]

Das Sandbox-Konzept soll gewährleisten, daß nicht vertrauenswürdige Software auf einem Wirtsrechner ohne Beeinträchtigung dessen ausgeführt werden kann. Maliziöse Agenten sollen keine Möglichkeit erhalten, Schaden innerhalb des Wirtssystems anzurichten. Agenten innerhalb der Sandbox haben deshalb lediglich Zugriff auf Ressourcen, die ihnen durch die Sandbox bereitgestellt werden. Zu beachten ist, daß Sun eine Unterscheidung von Java-Agenten vornimmt, die in Browsern ausgeführt werden und solchen, deren Ausführungsumgebung lediglich aus der JVM besteht. Im letzteren Fall spricht Sun von *Java Applications* und diese unterliegen nicht den Einschränkungen des Sandbox-Modells, wenn sie in einer JVM ausgeführt werden.

Robustheit

Java wurde zunächst als Sprache für den Einsatz in Geräten aus dem Bereich der Konsumentenelektronik entwickelt. In diesem Einsatzgebiet ist es von wesentlicher Bedeutung, daß die in den Geräten agierenden Programme äußerst stabil laufen. Instabile Programme resultieren im wesentlichen aus Implementationsmängeln. Diesem Problem ist Sun durch die Entwicklung einer robusten Sprache begegnet. Dem ist jedoch nicht der Schluß zu entnehmen, daß eine robuste Programmiersprache die Lösung für alle Sicherheitsprobleme darstellt. Zu beachten ist stets, daß derartige Programmierhilfen auch gleichzeitig den Programmierer maliziöser Software davor bewahrt, Implementationsfehler zu begehen, mit der Folge, daß maliziöse Aktivitäten schwerer zu entdecken sind.

[15] Dieses Verbot ist essentiell für die Sandbox. Alle Sicherheitsmerkmale, die für Java-Agenten gelten, sind für native Programme nicht anwendbar, da die Sicherheitspolitik für native Applikationen keine Anwendung findet.

Der robuste Entwurf der Sprache zeichnet sich durch die Entfernung von Sprachkonzepten aus, die sich in der Vergangenheit als gefährlich für die Stabilität erwiesen haben, sowie durch Konstrukte, die die Erzeugung stabilerer Programme erlauben. Folgende Merkmale zeichnen jede Implementierung der JVM aus:

Ausnahmebehandlung Die Sprache stellt Konstrukte zur Verfügung, mit denen eine strukturierte Fehlerbehandlung möglich ist. Der Programmierer kann im Programmcode die Behandlung „unerwarteter" Ereignisse, wie beispielsweise den Zusammenbruch einer Netzwerkverbindung, vorsehen. Die Möglichkeit der Behandlung von Ausnahmefehlern gewährleistet, daß der Auftritt eines Ausnahmefehlers nicht zwangsweise einen Abbruch des Prozesses nach sich zieht.

Automatische Speicherverwaltung Das Java-Laufzeitsystem in Form der JVM stellt eine automatische Speicherverwaltung bereit, die den Programmierer von der Bürde befreit, selbst für die Speicherfreigabe nicht weiter referenzierter Objekte zu sorgen. Damit werden Probleme vermieden, die in Sprachen wie C und C++ für unzuverlässige Applikationen sorgen. Hierzu gehören der Zugriff auf Speicherbereiche, die noch von anderen Datenstrukturen genutzt werden; Zugriffe auf Speicherbereiche, die bereits freigegeben wurden oder der Verbrauch von Speicher durch nicht freigegebene Objekte.

Bounds Checking Sowohl für Arrays als auch für Strings erfolgt eine automatische Überprüfung der Grenzen. Damit werden Überläufe von Puffern und die damit verbundenen Fehler und Sicherheitlücken vermieden.

Einfach-Vererbung Java erlaubt im Gegensatz zu anderen objektorientierten Programmiersprachen, wie z. B. Eiffel, lediglich Einfachvererbung. Einfachvererbung erlaubt eine wesentlich deutlichere Darstellung von Vererbungs-Beziehungen, da eine eindeutigere Begriffsbildung erfolgt. Es bedarf keiner gesonderten Behandlung von Methoden, die in mehreren übergeordneten Klassen definiert sind.

Keine Pointer Java verfügt über keine Zeiger im Sinne von C oder C++. Stattdessen können nur strukturierte Speicherzugriffe erfolgen. Der Bytecode enthält lediglich symbolische Referenzen, die erst zur Laufzeit in reale Speicheradressen durch den Java Interpreter umgesetzt werden [GM95].[16] Programmierer können dementsprechend keine Zeigerarithmetik durchführen und insbesondere keine Zeiger verfälschen, da die Zeigerzuteilung transparent für den Programmierer erfolgt und vollständig durch das unterliegende Laufzeitsystem kontrolliert wird. Sowohl die Sprache Java als auch der unterliegende Bytecode bieten keine Möglichkeit, um unstrukturierte Speicherzugriffe auszudrücken [Ven97b].

[16] Der Java Bytecode enthält keine Hinweise, an welcher Stelle des Speichers spezifische Datenstrukturen abgelegt werden. Die ist die Speicherverteilung ist abhängig von der Hard- und Software-Plattform, d.h., von der spezifischen Implementation der JVM [GM95]. Sie ist nicht Bestandteil der Spezifikation der JVM.

Streng typisiert Java ist streng typisiert, d. h. jedem Bezeichner ist ein Typ zugeordnet. Damit ist eindeutig definiert, wie die mit den Bezeichnern verbundenen Daten zu interpretieren sind. Die Typüberprüfung erfolgt jedoch nicht vollständig vor der Ausführung. Statische Typüberprüfung zum Zeitpunkt der Kompilation werden im Bytecode reflektiert. Die statische Typüberprüfung erfolgt in Java aus Geschwindigkeitsaspekten. Sie erfolgt zum Zeitpunkt des Linkens der Klassen. Die gesamte Sicherheitsarchitektur hängt fundamental von der Typsicherheit der Sprache ab [WF98].

Class-Loader

Java Anwendungen können dynamisch zur Laufzeit erweitert werden. In verteilten Systemen ist damit ein großer Vorteil verbunden, denn so muß bei einer Migration eines Agenten nur ein Bruchteil des Codes mitgeführt werden. Werden zur Laufzeit am Migrationsziel des Agenten weitere Klassen benötigt, werden diese durch einen Class-Loader bezogen. Klassen werden in ein Array von Bytes geladen und anschließend zu Instanzen der Klasse `Class` konvertiert. Erst die Class-Loader-Technologie stellt sicher, daß zur Zeit der Kompilierung nicht alle Klassen eines Agenten vorhanden sein müssen.

Es sind zwei Arten von Class-Loadern zu unterscheiden: ein „primordial“ Class-Loader und Class-Loader-Objekte. Der primordial Class-Loader ist Bestandteil der JVM und ebenso wie diese in einer nativen Sprache implementiert. Class-Loader-Objekte sind hingegen Objekte der in Java zu implementierenden abstrakten Klasse `ClassLoader` bzw. deren Subklassen. Die JVM betrachtet alle Klassen, die durch den primordial Class-Loader geladen werden als vertrauenswürdig, unabhängig davon, ob sie Bestandteil der Klassen der JVM sind. Semantisch bedeutet der Begriff „vertrauenswürdig“ in diesem Zusammenhang erneut, daß durch den primordial Class-Loader ausschließlich Systemklassen oder Klassen, die über die Variable `CLASSPATH` zugreifbar sind, geladen werden. Agenten, die durch Class-Loader Objekte geladen werden, werden als nicht vertrauenswürdig betrachtet [Ven97c].

Die Bedeutung des Class-Loaders für die Sicherheit resultiert aus dem Umstand, daß die JVM für jedes instantiierte Objekt verzeichnet, durch welchen Class-Loader das Objekt bezogen wurde. Versucht eine Klasse A eine Klasse B zu referenzieren, fordert die JVM den Class-Loader der Klasse A auf, die Referenz aufzulösen. Diese Vorgehensweise hat zur Folge, daß jeder Class-Loader einen Namensraum bildet, so daß sich Klassen nur sehen können, wenn sie durch den gleichen Class-Loader geladen wurden.

Die Namen, die innerhalb eines durch einen Class-Loader gebildeten Namensraum existieren, sind eindeutig. Class-Loadern ist es unmöglich, zwei unterschiedliche Klassen mit gleichen Namen zu laden. Damit ist es maliziösen Klassen nicht möglich, sich als vertrauenswürdige Klassen auszugeben, die bereits innerhalb eines Namensraums existieren.

Die JVM kontrolliert über den Class-Loader-Mechanismus Zugriffe von Agenten untereinander und unterbindet Zugriffe zwischen Klassen, die sich in verschiedenen Na-

mensräumen befinden. Zugriffe auf Klassen außerhalb des eigenen Namensraumes sind für Java Objekte nur dann möglich, wenn dieses durch die umgebende Anwendung unterstützt wird. Ohne die Unterstützung sind Java Objekte nicht einmal in der Lage, festzustellen, daß andere Namensräume vorhanden sind, in denen weitere Java Objekte agieren. Generell ist nur die Interaktion mit Objekten innerhalb des gleichen Namensraums möglich.

Im Falle der in den Abschnitten 5.2.1 und 5.2.2 vorgestellten Browser, wird für Java Agenten, die von unterschiedlichen Servern bezogen werden, jeweils eine Instanz der Klasse `ClassLoader` erzeugt.

Java Klassen werden durch den Class-Loader geladen, damit bildet er die erste „Verteidigungslinie“ gegen maliziöse Java-Agenten [Ven97c]. Der Class-Loader verhindert durch den Mechanismus der Namensräume, daß maliziöser Code gutartigen Code beeinflußt. Weiterhin wird die Möglichkeit unterbunden, daß sich maliziöser Code als Teil der vertrauenswürdigen Systemklassen ausgibt. Könnte ein maliziöser Agent sich als vertrauenswürdige Systemklasse maskieren, würde damit das Konzept der Sandbox durchbrochen. Der maliziöse Code hätte dann Zugriff auf alle Methoden und Ressourcen, die außerhalb der Sandbox liegen. Die Sicherheit der JVM wäre kompromittiert.

Der Algorithmus zum Bezug einer Klasse folgt diesem Schema:

1. Im ersten Schritt stellt das Class-Loader-Objekt fest, ob eine Klasse referenziert wird, die durch den Class-Loader nicht geladen werden darf. Ist dieses der Fall, wird eine Sicherheitsausnahme generiert, sonst wird mit Schritt zwei fortgefahren.

2. Das Class-Loader-Objekt leitet die Anfrage an den primordial Class-Loader weiter. Kann dieser eine Referenz auf die gewünschte Klasse zurückliefern, liefert das Class-Loader-Objekt die Klasse zurück. Ansonsten wird mit Schritt drei fortgefahren.

3. Existieren vertrauenswürdige Pakete (z. B. `java.lang`), denen das Class-Loader-Objekt keine Klassen zufügen darf, überprüft das Objekt, ob die referenzierte Klasse dort existiert. Ist das der Fall, erzeugt das Class-Loader-Objekt eine Sicherheitsausnahme. Im anderen Fall wird mit Schritt vier fortgefahren.

4. Wird dieser Punkt erreicht, ist die referenzierte Klasse im lokalen System nicht vorhanden. Das Class-Loader-Objekt versucht nun, die erforderliche Klasse aus dem Netzwerk zu laden. Gelingt dieses nicht, wird eine Ausnahme generiert, die beinhaltet, daß die Klasse im Netz nicht gefunden werden konnte [Ven97c].

Jeder Programmierer, der eine Java Anwendung implementiert, die als Umgebung für Java Objekte agiert, kann einen oder mehrere Class-Loader implementieren. Microsoft und Netscape haben für ihren jeweiligen Browser eine entsprechende Implementation des Class-Loaders entwickelt. Der Class-Loader bildet damit zusammen mit dem nachfolgend

beschriebenen Security-Manager den anpaßbaren Teil der Sicherheitskomponenten Javas. Entsprechend den Sicherheitsanforderungen der Anwendung können so unterschiedliche Ausprägungen des Sandbox-Konzeptes implementiert werden.

Security-Manager

Der Security-Manager definiert die äußere Grenze der Sandbox. Mittels des Security-Managers können unterschiedliche Sicherheitspolitiken durchgesetzt werden. Der Security-Manager der JVM-Spezifikation ist lediglich eine abstrakte Klasse, die nicht instantiiert werden kann. Es obliegt dem Entwickler, die Klasse zu implementieren und damit die Sicherheitspolitik der Anwendung festzulegen. Java-fähige WWW-Browser verfügen bereits über einen eigenen Security-Manager. Bis zur Version 3.x des Navigators bestand die Implementation des Security-Managers von Netscape darin, daß alle Aufrufe, die von nicht vertrauenswürdigen Agenten ausgingen, die Erzeugung einer `SecurityException` zur Folge hatten. In den nachfolgenden Versionen wurde eine differenziertere Sicherheitspolitik implementiert. Der Security-Manager beinhaltet für jede „gefährliche" Methode eine korrespondierende Methode,[17] die durch die „gefährliche" Methode aufgerufen wird. Anhand der Class-Loader-Informationen kann der Security-Manager feststellen, ob der Aufruf durch eine vertrauenswürdige bzw. durch eine nicht vertrauenswürdige Klasse, d.h., durch einen Java-Agenten, der in der Sandbox ausgeführt wird, erfolgt ist. Wurde der Aufruf von einem Agenten aus der Sandbox initiiert, generiert der Security-Manager eine Ausnahme (`SecurityException`).

Zu den Methoden, deren Aufruf eine Konsultation des Security-Managers nach sich zieht, zählen u. a. die folgenden Aktionen:

- ▷ Modifikation eines Threads;
- ▷ Erzeugung eines neuen Class-Loader-Objektes;
- ▷ Kreierung eines neuen Prozesses;
- ▷ Beendigung der umgebenden Anwendung;
- ▷ Lese- und Schreibzugriffe auf Dateien;
- ▷ Akzeptieren beliebiger Netzwerkverbindungen [Ven97a].

Der Security-Manager ist eine optionale Komponente der Sicherheitsarchitektur Javas. Den Entwicklern einer Java Anwendung steht es frei, die abstrakte Klasse der Java-Spezifikation zu implementieren. Wird ein Security-Manager implementiert, wird er einmalig zu Beginn der Applikation instantiiert. Anschließend ist es während der gesamten Lebenszeit der Applikation nicht möglich, das instantiierte Security-Manager-Objekt zu erweitern, zu überschreiben oder gar zu ersetzen. [FM96].

[17] Ein Aufruf der Methode `exit()`, die die Ausführung der JVM beendet, wird durch einen Aufruf der Methode `checkexit()` am Security-Manager auf ihre Zulässigkeit überprüft.

Verifier

Der Quelltext eines Javaprogrammes wird durch einen Java Compiler in Bytecode überführt. Eine JVM hat ohne Unterstützung durch den Bytecode-Verifier keine Möglichkeit festzustellen, ob der Bytecode durch einen maliziösen Compiler erzeugt oder der generierte Bytecode im nachhinein manipuliert worden ist. Aufgabe des Verifier ist es, sicherzustellen, daß der Bytecode einer Reihe von Regeln genügt. Stößt der Verifier dabei auf eine Regelverletzung, generiert er eine Ausnahme. Der Verifier ist Bestandteil der JVM-Spezifikation und somit Teil jeder Implementation der JVM. Daneben trägt der Verifier auch zur Robustheit der Sprache bei, indem er den Bytecode erkennt, der durch einen fehlerhaften Compiler erzeugt wurde.

In der Regel führt der Verifier die Überprüfung in zwei Phasen durch. In der ersten Phase, vor der Ausführung des Bytecodes wird dessen Struktur überprüft. Die zweite Phase beginnt mit Ausführung des Bytecodes und stellt sicher, daß alle symbolischen Referenzen von Feldern, Klassen und Methoden korrekt sind [Ven97d].

Erste Phase Alle Überprüfungen innerhalb der ersten Phase finden ausschließlich direkt auf dem geladenen Bytecode statt. Die erste Überprüfung betrifft die Magic-Number der geladenen Datei. Handelt es sich um eine `Class`-Datei, muß diese `0xCAFEBABE` lauten. Nachfolgend wird überprüft, ob die Länge der Datei korrekt ist, d. h., es wird sichergestellt, daß der Datei keine Informationen zugefügt oder abgeschnitten wurden. Jede Komponente einer `Class`-Datei ist mit ihrer Länge und ihrem Typ versehen, so daß festgestellt werden kann, ob die externe Länge mit der Summe der internen Längenangaben korrespondiert.

Daneben überprüft der Verifier eine Reihe von Invarianten der Sprache, die jede Klasse erfüllen muß. Dazu gehört beispielsweise, daß jede Klasse, mit Ausnahme der Klasse `Object`, eine Oberklasse besitzen muß. Damit überprüft der Verifier zur Laufzeit, daß die Regeln, die der Compiler zur Kompilationszeit durchgesetzt haben sollte, eingehalten werden.

Im Anschluß an die globale Überprüfung der `Class`-Datei erfolgen Kontrollen, die sicherstellen, daß jede Komponente eine wohlgeformte Instanz ihres Typs ist.

Im letzten Teil der ersten Phase führt der Verifier eine Datenflußanalyse auf den Bytecodeströmen durch, die die Methoden der Klasse repräsentieren [Ven97d]. Der Bytecodestrom besteht aus einer Folge von 1-Byte langen Opcodes, die möglicherweise ein oder zwei Operanden besitzen. Im Rahmen der Datenflußanalyse stellt der Verifier sicher, daß

- ▷ kein Zugriff auf lokale Variablen erfolgt, bevor diese einen gültigen Wert besitzen;
- ▷ Attributen der Klasse stets Werte des korrekten Typs zugewiesen werden;
- ▷ Methoden stets mit der korrekten Anzahl und den richtigen Typen aufgerufen werden.

Zweite Phase In der zweiten Phase während der Ausführung der Klasse stellt der Verifier sicher, daß die symbolisch referenzierten Felder, Klassen und Methoden existieren. Symbolische Referenzen erhalten ausreichende Informationen, so daß das referenzierte Objekt eindeutig identifiziert werden kann. Der Verifier folgt den Referenzen, um so festzustellen, daß diese korrekt sind. Dieses ist ein Teilaspekt des „dynamic linking", d. h., der Auflösung von symbolischen Referenzen in direkte. Im Falle, daß die symbolische Referenz nicht aufgelöst werden kann, weil die geforderte Klasse nicht zugreifbar ist oder nicht die erforderliche Methode enthält, generiert der Verifier eine Ausnahme.

In der Standardeinstellung der Java-Implementation Suns wird der Verifier nur für nicht-lokale Klassen aktiviert. Dieses gilt für die Versionen 1.0 und 1.1 des JDK.

5.3.2. Sicherheitsarchitektur JDK 1.1

Das JDK 1.1 baut auf der bekannten Sicherheitsarchitektur auf und erweitert diese um das Konzept des digital signierten Agenten. Digital signierte Agenten können die Grenze der Sandbox überschreiten, wenn die Verifikation der Signatur ergibt, daß diese korrekt ist.

Die Unterzeichnung von Agenten erfolgt mittels des Tools `jarsigner`. Vor der Signierung eines Java-Agenten werden zunächst alle erforderlichen Dateien zu einer JAR-Datei zusammengefaßt. In diesem Zustand wird dann das gesamte Archiv erzeugt. Dabei werden die Dateien `mykey.sf` und `mykey.dsa` dem Archiv zugefügt.[18] Die Datei `mykey.sf` enthält die Hash-Informationen, während `mykey.dsa` die eigentliche Signatur zusammen mit dem Zertifikat enthält. Die JVM auf der empfangenden Seite extrahiert den Agenten mit den erforderlichen Dateien aus dem Archiv und verifiziert die Signatur des Agenten. Daraufhin hat ein signierter Agent vollen Zugriff auf alle Systemressourcen, d. h., er unterliegt nicht länger den Einschränkungen des Sandbox-Modells. Führt die Verifikation des Agenten zu keinem erfolgreichen Ergebnis, wurde der Agent also zwischenzeitlich manipuliert, erfolgt die Ausführung des Agenten, wie im Falle eines unsignierten Agenten, in den durch die Sandbox aufgestellten Grenzen.

Das JDK 1.1 umfaßt mit den Schnittstellen für digitale Signaturen und Message Digests neue Klassen, die kryptografische Anwendungen unterstützen. Darüberhinaus existieren abstrakte Schnittstellen für die Handhabung von Schlüsseln, Zertifikaten und eine Schnittstelle für die Zugriffskontrolle.

Das Sicherheitskonzept des JDK 1.1 ist vergleichbar mit der von Microsoft entwickelten Authenticode-Technologie. Im Gegensatz zu dieser werden signierte Java-Agenten weiterhin in einer virtuellen Maschine ausgeführt und sind in einer robusten Sprache implementiert, so daß die von mangelhaft implementierten Agenten ausgehenden Gefahren nicht die gleichen sind, wie im Falle der mit Authenticode signierten ActiveX-Agenten. Desweiteren wird die Ausführung der signierten Java-Agenten durch die Sicherheitskom-

[18] Das entspricht einer Berechnung des Hashwertes, Entpackung des Archives, zufügen der `mykey.*` Dateien und der anschließenden erneuten Zusammenfassung der Dateien zum Archiv.

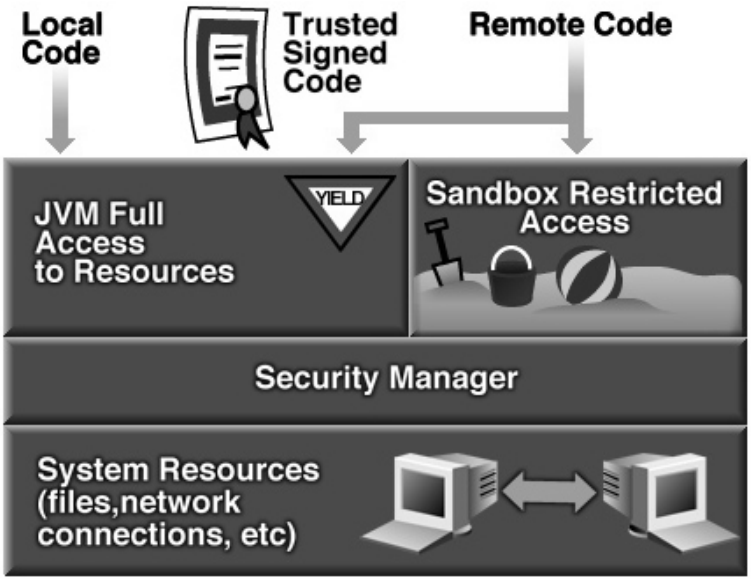

Abbildung 5.16.: Die Sicherheitsarchitektur im JDK 1.1 unterscheidet den entfernten Code in vertrauenswürdige Agenten und nicht vertrauenswürdige Agenten. Vertrauenswürdige Agenten sind signiert. Ihnen werden die gleichen Rechte zugestanden, die lokale Klassen erhalten (`Quelle: [CW97]`).

ponenten Class-Loader und Verifier überwacht, so daß die Implementierung maliziöser Agenten unter Java aufwendiger ist, als dieses für ActiveX-Agenten der Fall ist.

Das für das JDK 1.1 erweiterte Sicherheitskonzept ist eine Reaktion auf die – insbesondere von Unternehmen – geforderte Rücknahme der für Java-Agenten geltenden Restriktionen. Mit dem bekannten Sandbox-Modell konnten für Agenten, die aus einem Netz bezogen wurden, viele Anwendungen nicht verwirklicht werden, da die Restriktionen beispielsweise den Zugriff auf das lokale Dateisystem verweigerten.

Aus lizenzrechtlichen Gründen und aufgrund der Exportbeschränkungen in den Vereinigten Staaten werden mit dem JDK keine Klassen ausgeliefert, die die kryptografischen Klassen implementieren. Stattdessen können Implementation der „Cryptographic API“ von unterschiedlichen Herstellern bezogen werden. Die JVM ermittelt aus den Einträgen in der Datei `java.security`, welcher „Provider“ bei Nutzung einer bestimmten kryptografischen Klasse genutzt werden soll. Ein Eintrag in der `java.security` hat die Form:

Insgesamt stellt die Java Laufzeit-Umgebung eine Schnittstelle zur Verfügung, die es dem Sicherheits-Entscheidungsmechanismus erlaubt, den Laufzeit-Stack nach nicht vertrauenswürdigen Code zu untersuchen und darauf basierende Entscheidungen zu treffen. Der Java Sicherheitsmechanismus basiert auf der Inspektion des Laufzeitstacks.

Die mit dem aktuellen JDK vollzogene Entwicklung ist lediglich ein Zwischenschritt hin zu einer neuen Sicherheitsarchitektur, die von dem Sicherheitsarchitekten Javas – Li

```
security.provider.n = classname
```

Abbildung 5.17.: Mit einem derartigen Eintrag in der `java.security` kann die JVM ermitteln, welche Implementation einer kryptografischen Funktion (`classname`) genutzt werden soll. Eine Unterscheidung mehrerer Hersteller (Provider) erfolgt durch die Priorisierung, die durch `n` gegeben ist.

GONG – entworfen wurde, und mit dem aktuellen Java 2 abschließend umgesetzt worden sein soll.

5.3.3. Sicherheitsarchitektur Java 2

Mit dem Versionswechsel zu Java 2 vollzieht sich ein Wechsel der Sicherheitsarchitektur Javas. Dieser besteht im wesentlichen darin, daß es kein binäres Vertrauensmodell mehr gibt, das lediglich zwischen vertrauenswürdigen und nicht vertrauenswürdigen Java-Klassen unterscheidet. Stattdessen wird eine Sicherheitsarchitektur eingeführt, die auf einem Capability-System beruht. Die neue Architektur erlaubt Java-Agenten genau die Rechte zuzuteilen, die sie für die Durchführung ihrer Aufgaben benötigen. In den bisherigen Versionen gab es bislang lediglich die Möglichkeit, den Zugriff auf lokale Ressourcen stark zu beschränken oder diesen ungehindert zu gewähren. Es sei darauf hingewiesen, daß darüberhinaus spezifische Sicherheitspolitiken durchgesetzt werden konnten. Dazu war es jedoch erforderlich, für jede Sicherheitspolitik eine Subklasse der abstrakten Klasse des Security-Managers zu implementieren. Dem Nutzer eines Agentensystems, der einen Browser als Umgebungssystem einsetzte, war es damit nicht möglich, Einfluß auf die Sicherheitspolitik der JVM zu nehmen.

Die nun neu eingeführte Sicherheitsarchitektur Javas ist der Abbildung 5.18 zu entnehmen. Zusammengefaßt zeichnet sich die neue Sicherheitsarchitektur Javas durch folgende Punkte aus:

- ▷ feinkörnige Zugriffskontrolle,
- ▷ einfach konfigurierbare Sicherheitspolitik,
- ▷ einfach erweiterbare Zugriffkontrollstruktur,
- ▷ Erweiterung von Sicherheitsüberprüfungen in allen Javaprogrammen [GMPS97].

Mit der neuen Sicherheitsarchitektur vollzieht sich der größte Sprung in der Evolution der Sicherheitsarchitektur Javas. Unterstützte die erste Version lediglich die Unterscheidung zwischen nicht vertrauenswürdigem, entfernten Code und vertrauenswürdigem, lokalen Code, erlaubt die neue Sicherheitsarchitektur fein-granulare Zugriffsentscheidungen auf Basis eines durch den Systemadministrator definierten Policy-Objektes. Der Wandel der Sicherheitsarchitektur ist in einer Übersicht in Tabelle 5.1 zusammengefaßt.

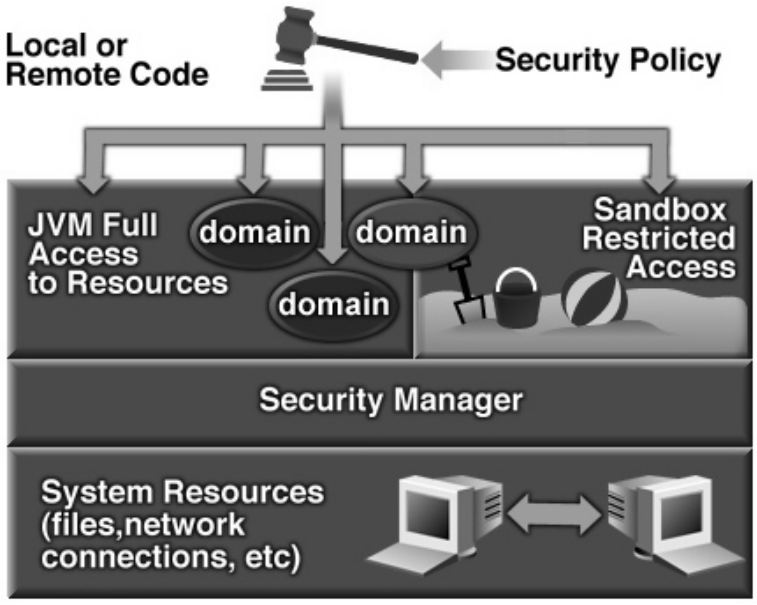

Abbildung 5.18.: Mit der Version 2 hat Java eine feinkörnige Sicherheitsarchitektur erhalten, in der anhand einer Sicherheitspolitik für jede Klasse entschieden wird, welche Rechte ihr zugestanden werden können. Es hält ein erweitertes Domänenkonzept Einzug, das nicht länger aus lediglich zwei Domänen für lokalen und entfernten Code besteht (`Quelle: [CW97]`).

Die seit dem JDK 1.0 erfolgten Änderungen der Sicherheitsarchitektur Javas erforderten keine Änderungen der JVM.

Access-Controller

Die neue Sicherheitsarchitektur sieht eine Trennung zwischen der Sicherheitspolitik und der Komponente, die diese durchsetzt, vor. Im Gegensatz zum alten Konzept des Security-Managers, der neben der Durchsetzung der Sicherheitspolitik diese auch beinhaltete, setzt der Access-Controller nur noch die von außen vorgegebene Sicherheitspolitik durch. Die Sicherheitspolitik der Anwendung wird zur Laufzeit durch ein Objekt der Klasse `Policy` repräsentiert. Die Klasse des Security-Managers ist weiterhin im JDK enthalten, dient jedoch nur noch der Abwärtskompatibilität. Alle Methodenaufrufe am Security-Manager werden an den Access-Controller weitergeleitet. Neu entwickelte Anwendungen nutzen ausschließlich ein Objekt der Klasse `AccessController` zur Durchsetzung der Sicherheitspolitik.

Der Access-Controller entscheidet, basierend auf der durch ein Policy-Objekt repräsentierten Sicherheitspolitik, ob der Zugriff auf eine kritische Systemressource gewährt oder verweigert wird. Alle Systemklassen konsultieren hierfür den Access-Controller durch den Aufruf der Methode `checkPermission(Permission)`. Ein Beispiel ist in Ab-

Tabelle 5.1.: Evolution der Sicherheitsarchitektur Javas [IBM98]

	JDK 1.0	JDK 1.1	Java 2
Zugriff lokaler, nicht signierter Agenten	Uneingeschränkt	Uneingeschränkt	Sicherheitspolitik
Zugriff lokaler, signierter Agenten	nicht verfügbar	uneingeschränkt	Sicherheitspolitik
Zugriff entfernter, nicht signierter Agenten	Sandbox	Sandbox	Sicherheitspolitik
Zugriff von Applikationen	Uneingeschränkt	Uneingeschränkt	Sicherheitspolitik

bildung 5.19 dargestellt.

Eine Entscheidung fällt der Access-Controller auf Basis der Aufrufkette. Der Access-Controller traversiert die Kette der Aufrufer rückwärts und bestimmt für jede Klasse, ob die Rechte für die Durchführung der Methode vorhanden sind. Der Algorithmus terminiert, wenn der Anfang der Aufrufkette erreicht wurde, d. h., alle Klassen in der Aufrufkette verfügen über die notwendigen Rechte. Trifft der Access-Controller auf eine Klasse, die nicht über die erforderlichen Rechte verfügt, wird der Algorithmus unterbrochen und eine Ausnahme vom Typ `AccessControlException` generiert. Stößt der Access-Controller innerhalb der Aufruferkette auf eine Klasse, die als privilegiert markiert ist und die über die notwendigen Rechte verfügt, terminiert der Algorithmus frühzeitig.

```
FilePermission perm = new FilePermission("/tmp/xxx", "read");
AccessController.checkPermission(perm):
```

Abbildung 5.19.: Systemklassen erzeugen eine Capability, die sie dem Access-Controller übergeben. Dieser überprüft, ob das Recht erteilt werden kann.

Eine Klasse kann sich durch den Aufruf der Methode `doPrivileged()` am Access-Controller als privilegiert kennzeichnen. Die Klasse definiert hierzu eine anonyme Inner-Class, die das Interface `PrivilegedAction` implementiert (siehe Abbildung 5.20). Beim konkreten Aufruf der Methode wird eine Instanz der `PrivilegedAction`-Implementierung übergeben. Nach der Erteilung der Rechte ruft die Methode `doPrivileged` die Methode `run()` an der `PrivilegedAction`-Instanz auf.

Security-Manager

Der Security-Manager, der bislang die Sicherheitspolitik und den sie durchsetzenden Mechanismus vereinigte, wird mit dem neuen JDK nicht länger benötigt. Die Aufgaben des Security-Managers wurden auf den Access-Controller und das Policy-Objekt aufgeteilt. Aus Gründen der Abwärtskompabilität ist der Security-Manager noch Bestandteil des

```
AccessController.doPrivileged(new PrivilegedAction() {
  public Object run() {
    System.loadLibrary("awt");
    return null;
  }
});
```

Abbildung 5.20.: Das Code-Fragment stellt die Übergabe einer Instanz der PrivilegedAction-Implementierung dar.

JDKs. Ein Aufruf der Methode `checkPermission` am Security-Manager wird durch diesen an den Access-Controller weitergeleitet. Das Verhalten des Security-Managers kann allerdings durch eine neue Implementierung desselbigen verändert werden. Weiterhin ist die Klasse des Security-Managers nicht länger eine abstrakte Klasse, die der Implementation bedarf. Sie kann nun direkt instantiiert werden.

Policy-Objekt

Die Zugriffsentscheidungen in der zukünftigen Sicherheitsarchitektur werden durch den Access-Controller aufgrund eines Policy-Objektes getroffen. Ein Policy-Objekt ist die Laufzeitrepräsentation der Sicherheitspolitik eines Systemes. Die Sicherheitspolitik wird durch textuelle Beschreibungen in Form von ASCII-Dateien definiert.

Im Systemverzeichnis einer jeden JDK-Installation befindet sich eine Datei namens `java.security`. In dieser Datei werden Verweise auf Dateien definiert, die unterschiedliche Sicherheitspolitiken spezifizieren.

In der Abbildung 5.22 wird mit der Zeile `policy.url.1= ...` auf die Datei verwiesen, die die systemweite Sicherheitspolitik spezifiert. Damit hat die Systemadministration eines Netzwerks als weitere Neuerung die Möglichkeit, eine systemweite Sicherheitspolitik für Java-Agenten zu definieren. Darüberhinaus kann den Nutzern des Netzwerkes das Recht eingeräumt werden, eigene Sicherheitspolitiken zu definieren, die die systemweite ergänzen. Die jeweiligen Dateien, in denen Sicherheitspolitiken für Java-Agenten spezifiziert werden, enthalten lediglich Einträge, in denen Rechte erteilt werden. Damit können Nutzer des Netzwerkes die systemweite Sicherheitspolitik lediglich erweitern, nicht jedoch Rechte zurücknehmen. Die Syntax für die Definition von Sicherheitspolitiken hält für die Verweigerung von Rechten keine Konstrukte bereit. Alle Rechte, die in den Dateien, die die jeweiligen Sicherheitspolitiken definieren, nicht angegeben sind, werden nicht gewährt. Java-Agenten, die nicht definierte Rechte in Anspruch nehmen wollen, lösen eine Ausnahme vom Typ `SecurityException` aus. Ein Eintrag in einer Policy-Datei, die ebenso wie die obige Datei im ASCII-Format kodiert wird, ist in der Abbildung 5.23 zu sehen [Gre98].

Die in einer Sicherheitspolitik definierten Rechte können darüberhinaus auf bestimm-

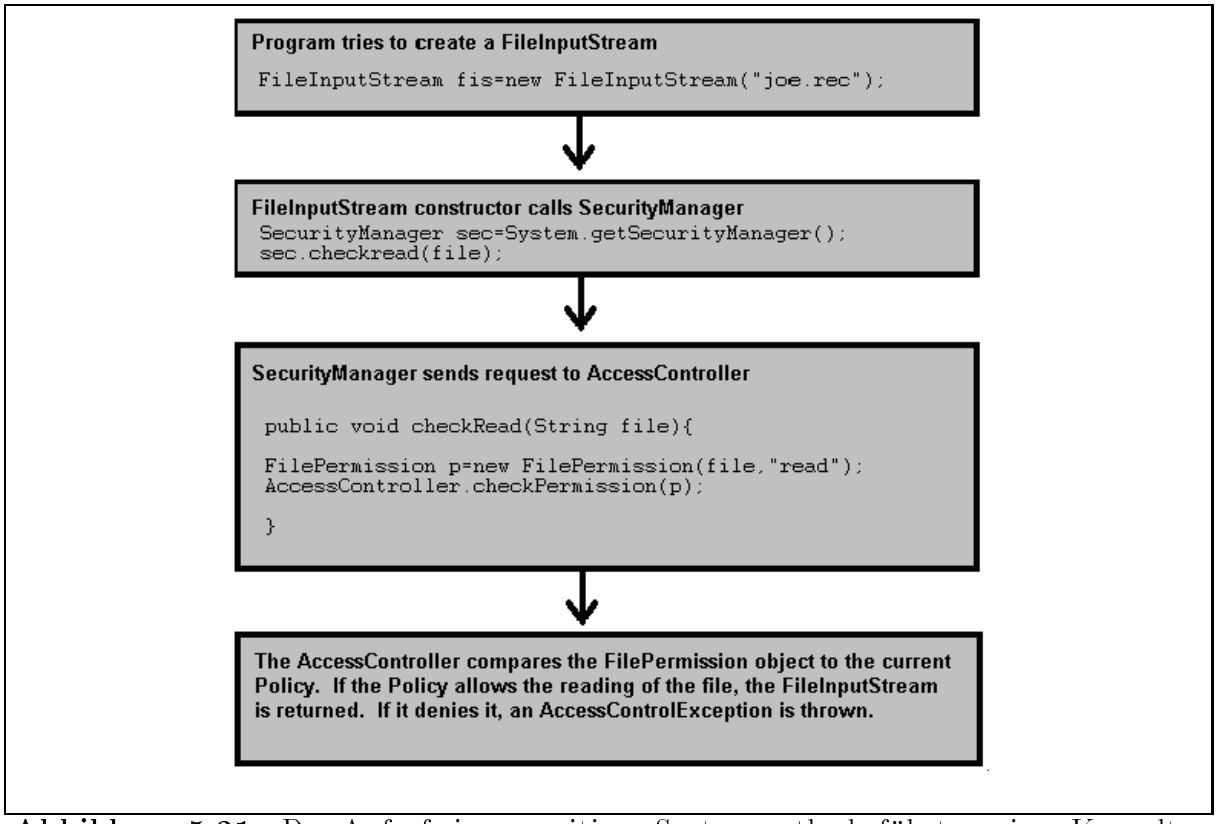

Abbildung 5.21.: Der Aufruf einer sensitiven Systemmethode führt zu einer Konsultation des Access-Controllers. Der Security-Manager wird weiterhin unterstützt, er leitet die Anfrage jedoch lediglich an den Access-Controller weiter. Dieser entscheidet aufgrund der durch das Policy-Objekt repräsentierten Sicherheitspolitik, ob die Rechteerteilung gestattet wird.

te Agenten oder Agentengruppen beschränkt werden. Hierzu bietet die Definitionssprache Konstrukte an, mit denen die Identität eines Subjektes und das Quellsystem eines Agenten spezifiziert werden kann. Erfordert das Policy-Objekt, daß ein Agent für den Zugriff auf eine geschützte Ressource signiert ist, wird der Zugriff verweigert, wenn ein Agent diese Bedingung nicht erfüllen kann.

Aus allen Dateien, die Sicherheitspolitiken definieren und auf die aus der Datei `java.security` verwiesen wird, generiert die JVM das bereits oben erwähnte Policy-Objekt, das zur Laufzeit der JVM die Sicherheitspolitik eines Netzes beinhaltet.

Die neue Sicherheitsarchitektur ermöglicht es damit, für jede Anwendung eine speziell angepaßte Sicherheitspolitik zu konstruieren, die den Sicherheitsanforderungen der Anwendung gerecht wird [Gre98]. Damit die neuen Eigenschaften der Sicherheitsarchitektur auch genutzt werden, ist dem JDK mit dem `policytool` ein grafisches Tool beigefügt, das es auf einfache Weise ermöglicht, eine Sicherheitspolitik zu erstellen.

Der Bezug eines Objektes über den Class-Loader zieht eine Konsultation des Policy-

```
# The default is to have a single systemwide policy file,
# and a policy file in the user's home directory.
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy
```

Abbildung 5.22.: Dies ist ein Beispiel für eine `java.security`-Datei. Es wird eine systemweite Datei `$java.home/lib/security/java.policy` und eine jeweils anwenderspezifische Datei `$user.home/.java.policy` referenziert.

```
grant codeBase "http://www.irgendwo.de", signedBy "Mustermann, Gaby" {
  permission java.lang.RuntimePermission "print.queueJob";
  permission java.io.FilePermission "/tmp/file", "write,delete,execute";
  permission java.net.SocketPermission "9.24.104.51:1345", "connect";
};
```

Abbildung 5.23.: Innerhalb einer Policy-Datei werden die Rechte für definierte Entitäten spezifiert. In diesem Fall werden allen Agenten, die von Gaby Mustermann signiert und vom Server `http://www.irgendwo.de` stammen, die Rechte eingeräumt, zu Drucken, die Datei `/tmp/file` zu lesen, zu schreiben und auszuführen, sowie eine Verbindung zum System `9.24.104.51` über den Port `1345` aufzubauen.

Objekts nach sich. Wenn die Sicherheitspolitik einen zutreffenden Eintrag enthält, wird dynamisch eine Sicherheitsdomäne erzeugt, der die mit dem Eintrag verbundenen Rechte zugeteilt werden. Anschließend wird das Objekt in die neu erzeugte Domäne eingeordnet. Nachfolgende Objekte zum gleichen Eintrag existieren in der gleichen Domäne. Basis für Zugriffsentscheidungen wird zukünftig die „CodeSource“ sein. Die CodeSource besteht aus der Kombination der URL-Adresse des Agenten und dem Unterzeichner des Agenten. Weiteres hierzu folgt im nächsten Abschnitt.

Mit Java 2 unterliegt jede Java Anwendung, ob Applet oder Application, lokal oder nicht-lokal, signiert oder nicht-signiert, einer Sicherheitspolitik. Zu beachten ist allerdings, daß die Komponente des Access-Controllers weiterhin eine optionale ist. Soll in einem System eine Sicherheitspolitik durchgesetzt werden, bedarf es einer expliziten Instantiierung des Access-Controllers. Für den Fall, daß für Java Applications eine Sicherheitspolitik gelten soll, bedarf es einen Aufruf der JVM mit entsprechenden Parametern. Die folgende Zeile ist hierfür ein Beispiel:

```
java -Djava.app.class.path=/home/apps java.security.Main anApplication
```

Abbildung 5.24.: Der Aufruf instantiiert für die Anwendung `anApplication` eine Sicherheitspolitik, die durch `java.security.Main` spezifiert wird.

Zur Laufzeit gibt es für jede JVM nur ein gültiges Policy-Objekt. Das aktive Policy-

Objekt kann über den Aufruf der Methode `getPolicy()` erworben werden. Klassen mit den erforderlichen Rechten können über die Methode `setPolicy` ein neues Policy-Objekt instantiieren.

Protection-Domain

Ein weiteres wesentliches Konzept der Sicherheitsarchitektur Javas in der Version 2 ist das der Sicherheitsdomäne. In dem bisherigen binären Vertrauensmodell existierten lediglich zwei Sicherheitsdomänen: die der Sandbox und die Welt außerhalb der Sandbox. Die neue Sicherheitsarchitektur unterstützt beliebig viele Sicherheitsdomänen. Das Java Sandbox-Konzept stellt eine statische Domäne dar, mit Java 2 werden Domänen dynamisch „on demand" erzeugt. Domänen werden transparent beim Laden einer Klasse generiert [GMPS97]. Die Zuteilung einer Klasse zu einer Domäne erfolgt über die Methode `setProtectionDomain`, die außerhalb des `java.security`-Packages nicht aufgerufen werden kann. Die Sicherheitsdomänen zeichnen sich dadurch aus, daß die Agenten in unterschiedlichen Domänen über verschiedene Rechte verfügen. Innerhalb einer Domäne besitzen hingegen alle Agenten die gleichen Rechte.

Sicherheitsdomänen sind ein wirkungsvoller Mechanismus, um Objekte mit gleichen Sicherheitsbedürfnissen zu gruppieren und Gruppen mit unterschiedlichen Sicherheitsbedürfnissen voneinander zu isolieren. Interaktion zwischen Domänen kann lediglich über den vertrauenswürdigen Systemcode stattfinden und durch die Zustimmung aller beteiligten Domänen.

Der Mechanismus der Sicherheitsdomäne stellt sicher, daß es Java-Agenten nicht möglich ist, in den Besitz von Rechten zu gelangen, die ihnen nicht gewährt wurden. Agenten erhalten ihre Rechte nicht direkt, sie sind nicht im Besitz der jeweiligen Capabilities, die ihnen zugestanden worden. Stattdessen ermittelt die JVM die Capabilities eines Agenten aus der Sicherheitsdomäne, in die ein Java-Agent eingeordnet wird.

Konzeptionell umfaßt eine Domäne eine Menge von Klassen, denen die gleichen Rechte erteilt wurden. Domänen werden durch eine CodeSource eindeutig identifiziert. Die CodeSource besteht aus einer URL und einer Menge von Zertifikaten. Alle Klassen einer Domäne stammen aus der gleichen Quelle und sind mit den gleichen Schlüsseln unterzeichnet. Jede Klasse gehört genau zu einer Domäne.

Die aktuelle Policy wird durch eine ProtectionDomain konsultiert, wenn diese ihre Rechte initialisiert. Die ProtectionDomain übergibt dem Policy-Objekt ein CodeSource-Objekt und die Zertifikatattribute. Anhand der Informationen, die das Policy-Objekt von der ProtectionDomain erhält, ermittelt es die Rechte der Domain. Diese werden als Rückgabewert an die Domain übergeben [Gon98].

Eine weniger „mächtige" Domäne kann keine zusätzlichen Rechte durch den Aufruf einer „mächtigeren" Domäne erhalten. Eine mächtigere Domäne verliert ihre zusätzlichen Rechte, so sie von einer weniger mächtigen Domäne aufgerufen wird [GMPS97].

Die Domäne einer Klasse wird einmalig gesetzt. Eine Migration einer Klasse in eine andere Domäne ist nach dem Setzen der Domänen-Zugehörigkeit nicht mehr möglich.

Nach der Generierung einer Domäne können die ihr zugeteilten Rechte lediglich gelesen werden. Eine nachfolgende Modifikation der Rechte durch die Domäne wird nicht unterstützt.

Class-Loader

Java 2 unterscheidet verschiedene Typen des Class-Loaders. Wird die erste Klasse einer Application geladen, instantiiert die JVM eine neue Instanz der Klasse `URLClassLoader`, die eine Subklasse der abstrakten Klasse `SecureClassLoader` ist. Für Applets wird eine Instanz der Klasse `AppletClassLoader`[19] erzeugt. Systemklassen, die mit der JVM ausgeliefert werden und damit Bestandteil der `java.lang.*` Hierarchie sind, werden durch den primordial Class-Loader geladen [Gon98].

Aufgabe des SecureClassLoader-Objektes ist es, die Schutzdomäne der zu ladenden Java Klassen zu bestimmen. Die Bestimmung erfolgt anhand der CodeSource der Klassen. Angeforderte Klassen, von denen bislang keine Instanz in der JVM existiert, unterliegen mit Java 2 einem veränderten Lademechanismus. Der Class-Loader sucht eine Klasse in der folgenden Reihenfolge:

1. im System-Classpath, der durch das Attribut `java.sys.class.path` spezifiert wird;
2. den installierten Erweiterungen der JVM durch zusätzliche Pakete;
3. den Applikations-spezifischen Classpath, der durch das Attribut `java.class.path` definiert wird [IBM98].

Das Attribut `java.sys.class.path` definiert die lokalen Systemklassen, die Bestandteil der JVM sind, wenn sie ausgeliefert wird. Ausschließlich die Systemklassen gelten als vertrauenswürdig und unterliegen damit nicht der Sicherheitspolitik, wie sie durch das Policy-Objekt definiert wird. Alle anderen lokalen Klassen, die durch das Attribut `java.class.path` definiert werden, gelten als nicht vertrauenswürdig.

5.3.4. Applets

Java Applets sind die bekanntesten Java-Agenten, die von allen Browsern unterstützt werden, in denen eine JVM implementiert ist. Die von Sun vorgesehene Sicherheitsarchitektur für Java Applets basierte vollständig auf der Sandbox. Mit dem Wechsel der Sicherheitsarchitektur findet zwischen Java Applets und Java Applications keine Unterscheidung mehr statt. D. h. insbesondere, daß es für Java-Agenten über die bereits vorgestellten Sicherheitskomponenten keine weiteren Sicherheitsmodule gibt. Die Sicherheitsarchitektur für Java-Agenten basiert vollständig auf dem bereits vorgestellten Modell.

[19] Die Klasse `AppletClassLoader` ist eine Subklasse der Klasse `URLClassLoader`.

Der Internet Explorer und Netscape Communicator unterstützen beide noch nicht die JVM in der Version 2, beinhalten damit noch nicht die neue Sicherheitsarchitektur Javas. Stattdessen haben im Hause beider Hersteller Eigenentwicklungen stattgefunden, die jeweils auf Capability-Systemen basieren.

Abschließend läßt sich feststellen, daß die Sicherheitsanforderungen von Java-Agenten und deren Umgebungssystem von der gleichen Sicherheitsarchitektur durchgesetzt wird, wie dieses für alle anderen Java Anwendungen der Fall ist.

5.3.5. Castanet

Agenten werden in der Castanet-Terminologie als Channels bezeichnet. Ausführbare Channels werden per Grundeinstellung als nicht vertrauenswürdig eingestuft. Im Gegensatz zu Java Applets erhalten nicht vertrauenswürdige Channels Zugriff auf das lokale Dateisystem. Der Tuner legt für jeden Channel ein spezielles Verzeichnis an. In diesen Verzeichnissen haben Channels Lese- und Schreiberlaubnis. Vertrauenswürdige Channels haben dagegen die gleichen Rechte, wie jede beliebige andere lokale Applikation.

Die Sicherung von Channels wird in Castanet auf zwei Arten unterstützt: Signierung und Verschlüsselung mittels SSL [Mar97c]. Für beide Techniken benötigt der Betreiber eines Channels ein Zertifikat.

Digital signierte Channels werden automatisch durch den Tuner auf ihre Integrität überprüft. Hierzu wird die digitale Signatur mittels des öffentlichen Schlüssels aus dem Zertifikat entschlüsselt. Anschließend wird ein *Message Digest* über den Channel gebildet. Stimmt der erhaltene Wert mit dem entschlüsselten Wert überein, ist der Channel authentifiziert und seine Integrität bestätigt. Schlägt die Überprüfung fehl, wird der Channel abgelehnt. Authentifizierte Channels werden in der Oberfläche des Tuners durch einen beigefügten Stift (siehe Abbildung 5.25) gekennzeichnet.

Ein Channel wird als vertrauenswürdig eingestuft, wenn der Entwickler den Channel als vertrauenswürdig proklamiert, signiert, und der Anwender den signierten Channel verifiziert und ihm die entsprechenden Rechte eines vertrauenswürdigen Channels erteilt [Mar97b]. Der Anwender kann seine Entscheidung lediglich anhand des Zertifikats des Channels treffen.

Bezieht der Tuner einen als vertrauenswürdig gekennzeichneten Channel aus einem Netzwerk, wird dieser nicht automatisch zur Ausführung gebracht. Stattdessen wird

- ▷ eine Warnung angezeigt, daß der Channel erweiterte Rechte benötigt, die nicht vertrauenswürdigen Castanet Agenten nicht zugestanden werden.
- ▷ eine Liste dargestellt, die die vom Castanet Agenten geforderten Rechte enthält.
- ▷ die Möglichkeit angeboten, Einblick in das Zertifikat des Agenten zu nehmen.
- ▷ der Anwender aufgefordert, die Rechte zu erteilen oder zu verweigern. Mit der Erteilung der Rechte wird der Agent als vertrauenswürdig eingestuft [Mar97a].

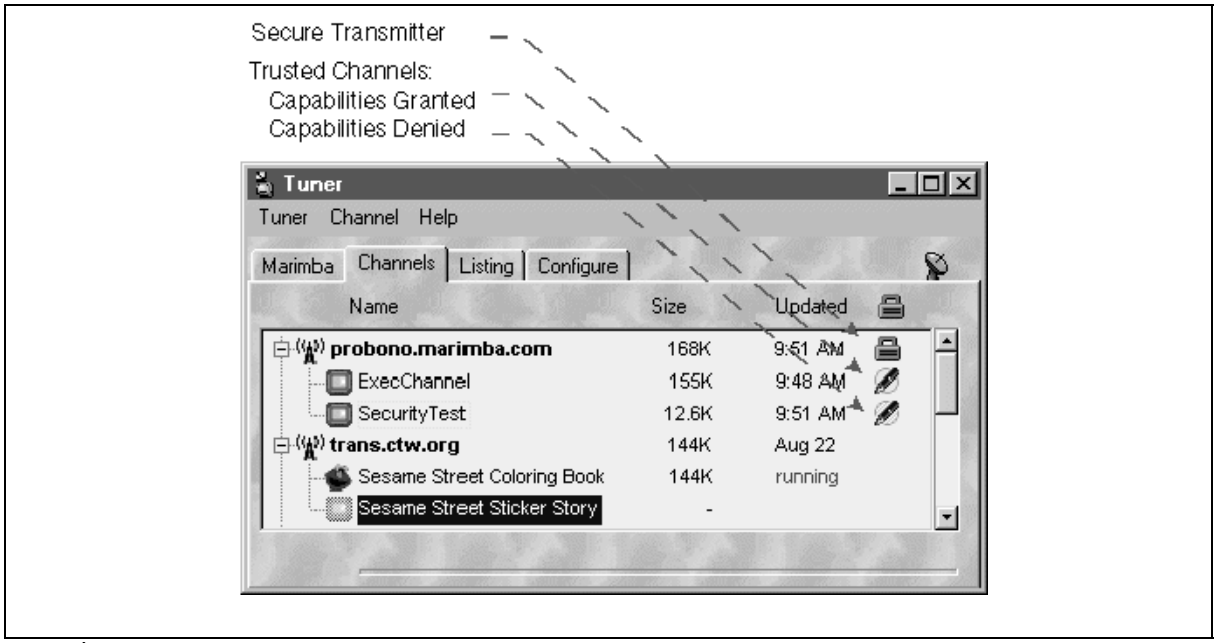

Abbildung 5.25.: Dem Castanet Tuner kann der Anwender anhand von Symbolen entnehmen, bei welchen Transmittern es sich nun sichere handelt und welchen Channels Rechte erteilt bzw. verweigert wurden.

Zertifikate für Castanet Channels und Transmitter werden bisher lediglich von VeriSign ausgestellt. Daraus folgt, daß dem Betreiber eines Agentensystemes lediglich nur eine CA zur Verfügung steht. Der Betreiber ist gezwungen, dieser CA zu vertrauen.

Die Signierung eines Channels muß sich nicht vollständig auf diesen erstrecken. Das System zur digitalen Signierung unterstützt auch die teilweise Signierung eines Channels. Auf diese Weise ist gewährleistet, daß Channels mit dynamischen Komponenten, die sich während der Lebenszeit ändern, ebenfalls signiert werden können.

Der VeriSign Produktkatalog enthält vier unterschiedliche Zertifikate. Zertifikate der ersten Klassen erfordern nahezu keine Überprüfungen durch VeriSign, so daß diese Zertifikate nur eine geringe Zusicherung beinhalten, daß die Korrespondenz der ausgewiesenen Person mit dem öffentlichem Schlüssel gegeben ist. Zertifikate der Klasse zwei erfordern vom Antragsteller Informationen, die es VeriSign erlauben, die Identität des Antragstellers eindeutig zu verifizieren. Zertifikate der Klassen drei und vier sind für kommerzielle Unternehmen gedacht, für die ein Eintrag im Handelsregister erforderlich ist. Darüberhinaus vergibt VeriSign Test-Zertifikate, die ebenfalls dazu genutzt werden können, um Channels zu signieren. Test-Zertifikate haben eine Gültigkeit von zwei Wochen.

Channels enthalten einen Rückkanal („BackChannel“) zum Transmitter. Über diesen können Daten versandt werden, die es dem Channel-Betreiber erlauben, eine Nutzungsanalyse des Channels durchzuführen. Der Rückkanal sendet Daten mittels SSL an den Transmitter, wenn der Channel von einem *sicheren Transmitter* bezogen wurde [Mar97b].

Der Zugang zu Channels kann auf zwei Arten kontrolliert werden. Die einfache Lösung besteht darin, daß der Entwickler des Channels diesen als *hidden* kennzeichnet. Der Zugriff auf derartige Channels ist dem Client dann nur durch das Wissen des Channel-Namens und des zugehörigen Transmitters möglich, da er durch den Tuner nicht angezeigt wird [Mar97b]. Die zweite Variante der Zugangskontrolle erfordert für den Zugriff auf einen Channel eine User ID sowie ein Paßwort.

Nur sichere Transmitter verschlüsseln mittels SSL und authentifizieren sich gegenüber dem Tuner bzw. Publisher. Die Authentikation eines Transmitters erfolgt über die Präsentation eines Zertifikates. Transmitter beinhalten Methoden, mit denen es ihnen möglich ist, Zertifikate von VeriSign zu beziehen und diese zu installieren [Mar97b].

Der Zugriff auf Systemressourcen wird durch einen Tuner Security-Manager geregelt, der eine Spezialisierung des Applet Security-Managers darstellt [Mar97b].

5.3.6. Servlets

Die Entwicklung Javas hat dazu geführt, daß die Sprache heute für Bereiche eingesetzt wird, für die sie nicht entwickelt worden ist. Bis 1997 wurde Java im wesentlichen auf Client-Systemen eingesetzt. Mit der Entwicklung von Servlets stehen nun auch Java-Programme auf Seiten des Servers zur Verfügung. Welche Auswirkungen sich daraus für die Sicherheit von Servern ergeben, bedarf einer detaillierten Untersuchung, als es in dieser Arbeit möglich ist. Forschungen auf diesem Gebiet wurden bislang nicht unternommen.

Servlets, die aus entfernten Quellen bezogen werden, unterliegen ähnlichen Sicherheitsauflagen, wie dieses für Applets der Fall ist. Sie werden als nicht vertrauenswürdig eingestuft und laufen innerhalb einer Sandbox des Servers ab. Darüberhinaus gibt es ACLs, die den Zugriff auf Systemressourcen reglementieren [Cha97]. Vertrauenswürdige Servlets werden über signierte JAR-Dateien realisiert.

5.3.7. Voyager

Voyager stellt Methoden zur Verfügung, die es dem Entwickler erlauben, auf einfache Weise das Paradigma des Agenten in Anwendungen umzusetzen. Die wesentliche Sicherheitskomponente ist der Voyager Security-Manager. Der Security-Manager Voyagers ist eine Subklasse der Klasse `SecurityManager` und implementiert die Sicherheitspolitik Voyagers. Damit basiert die Sicherheitsarchitektur vollständig auf der Javas und das derzeit ausschließlich in der Version 1.0. Eine Anpassung an die neue Sicherheitsarchitektur Javas ist bislang nicht erfolgt.

5.3.8. Beurteilung

Java hat sich zur Sprache des Internets entwickelt. Es gibt eine Vielzahl von Anwendungen im Inter- und Intranet, die auf Java basieren. Doch Java ist nicht zu diesem Zweck

entwickelt worden. Die Entwicklung von Java begann in Form von Oak. Ziel der Sprache Oak war die Entwicklung kompakter und zuverlässiger Programme, die in Geräten wie Fernsehern oder Toastern eingesetzt werden sollten. Es sollte auf einfache Art und Weise möglich sein, die Programme aus einem Netzwerk in ein entsprechendes Gerät zu laden. Die Zielumgebungen waren geschlossene Systeme. Das Laden von Software wäre nur durch den Hardwareproduzenten erfolgt [GS97].

Die Vergangenheit der Sprache Java ist von entscheidender Bedeutung, weil Sicherheitsbelange erst eingeflossen sind, als deutlich wurde, daß die weitere Entwicklung in Richtung des Internets erfolgen sollte. Die Sicherheitskomponenten der Sprache Java sind demnach lediglich eine Erweiterung der Sprache und sind nicht in das Design der Sprache eingeflossen.

Volpano und Smith zeigen auf, wie die Vorgehensweise für den Entwurf einer Programmiersprache erfolgen sollte: Zunächst sollte bekannt sein, daß Programmiersprachen eine fundamentale Rolle in der Computersicherheit einnehmen und sie sollten dementsprechend entworfen werden. Es sollten die gewünschten Sicherheitsanforderungen aufgestellt werden, die durch die Sprache zu erfüllen sind. Die anschließende Erweiterung der Sprache um weitere Merkmale sollte stets beachten, daß die Sicherheitsanforderungen weiterhin erfüllt bleiben [VS97]. Die Einführung von Sicherheitsmerkmalen als »Add-on« widerspricht sowohl den Entwicklungsprinzipien die in der Softwaretechnik erstellt wurden als auch den Anforderungen, die die Computersicherheit an den Entwicklungsprozeß stellt.

Für die Sprache existiert weder eine formale Semantik noch eine formale Beschreibung des Typsystems.[20] Dementsprechend ist nicht beschreibbar, was mit einem Java Programm im formalen Sinne gemeint ist. Gleiches gilt für das Typsystem, hier ist man auf eine augenscheinliche Stimmigkeit des Systems angewiesen. Die Existenz einer formalen Semantik, die in Java nicht gegeben ist, wird von Volpano und Smith als wesentliches Merkmal einer *sicheren* Programmiersprache hervorgehoben [VS97]. Bei gegebener formaler Spezifikation und Definition der Semantik der Programmiersprache ist es möglich, den Beweis zu erbringen, daß ein Programm seiner Spezifikation entspricht. Damit kann die Fehlerfreiheit des Programms verifiziert werden, vorausgesetzt, der durchgeführte Beweis ist korrekt [Som92]. Notwendig ist eine präzise Darstellung, wie sich ein wohlgetyptes Programm verhält, wenn es ausgeführt wird. Eine sichere Implementierung muß alle Fehler entdecken, die es zum Abbruch bringen könnten [VS97].

Java unterliegt einer komplizierten Sicherheitspolitik, da die Sprache zwei vollkommen unterschiedlichen Einsatzgebieten gerecht werden muß. Sie dient sowohl als Programmiersprache für den Entwurf beliebiger Programme (general purpose language), als auch der Entwicklung von Agenten, die über ein Netz auf einen lokalen Rechner geladen werden [GS97]. Entsprechend werden zwei völlig konträre Sicherheitspolitiken benötigt. In Java implementierte Browser sollen in der Lage sein, Dokumente auf der Festplatte abzulegen, Applets soll dieser Zugriff hingegen verwehrt bleiben.

[20] Drossopoulou und Eisenbach entwickeln derzeit eine formale Semantik für Java [DEK97].

Die Sicherheit des Java-Systems beruht auf drei Komponenten. Ein gutes Sicherheitskonzept faßt die Sicherheitsmodule in einer Komponente zusammen und minimiert so die Gefahr von Sicherheitslücken. Durch das Zusammenspiel mehrerer Komponenten steigt die Komplexität des Systems und damit die Gefahr von Sicherheitslücken. Der Einsatz vom mehreren Sicherheitskomponenten ist dann sinnvoll, wenn diese unabhängig voneinander agieren. Der Ausfall einer Komponente hat nicht die Kompromittierung des Systems zur Folge. Man spricht hier von einem Bastionskonzept.

Das Schichtenmodell der Java Sicherheitsarchitektur bestehend aus der Typsicherheit, dem Verifier, Classloader und Security-Manager stellt kein Bastionskonzept dar, indem erst alle Schichten der Sicherheitsarchitektur durchbrochen werden müssen. Die Sicherheitsarchitektur der Sprache Java und des Laufzeitsystems besteht aus vier Komponenten, die abhängig voneinander sind. Das heißt aber gleichzeitig, daß es einem Angreifer lediglich gelingen muß, eine der Schranken zu durchbrechen, um die Sicherheit eines Java-Systems zu kompromittieren. Die Sicherheit eines Java-Systems ist lediglich dann gewährleistet, wenn alle Bausteine der Sicherheitsarchitektur korrekt implementiert sind und die Zusammenarbeit der Komponenten wie spezifiziert erfolgt. Die Wahrscheinlichkeit, daß eine Sicherheitsarchitektur mit abhängigen Komponenten Mängel aufweist, steigt mit jeder zusätzlichen Komponente. Ließe sich die Wahrscheinlichkeit der korrekten Implementation der einzelnen Bausteine in der Wahrscheinlichkeit p ausdrücken,[21] so würde für eine Sicherheitsarchitektur mit n voneinander abhängigen Komponenten der Schluß zu ziehen sein, daß das System mit einer Wahrscheinlichkeit von p^n korrekt funktioniert. Anders ausgedrückt, mit einer Wahrscheinlichkeit von $1 - p^n$ wird eine derartige Architektur Ziel eines erfolgreichen Angriffes.

Die folgenden Punkte sind laut DEAN und FELTEN wesentlich für einen sicheren, mobilen Code:

- ▷ Das Austauschformat für mobile Agenten muß einfach zu analysieren sein.
- ▷ Wenige, gute Mechanismen sollten konsistent verwendet werden.
- ▷ Das System sollte lediglich theoretisch gut verstandene Konstrukte verwenden, die bereits implementiert wurden.
- ▷ Das System sollte klein und wohlstrukturiert sein, so daß es einfach zu verstehen ist [DF97].

Java erfüllt bislang keinen der genannten Punkte [DF97].

Ein fundamentales Problem Java-basierter Agentensysteme ist die Frage des Besitzers eines Agenten. In Java gibt es die Notation des „Besitzers“ eines Objekts nicht. Kein Java Objekt ist Besitzer einer bestimmten Referenz. Implizit kann in Java als Besitzer P eines Objektes O jener bezeichnet werden, der zu einem Zeitpunkt t alleinig das Objekt

[21] Es wird dabei davon ausgegangen, daß die Wahrscheinlichkeit für eine korrekte Implementation für alle Komponenten gleich ist.

O referenziert. Erst dann ist P in der Lage, das Objekt aus dem Speicher entfernen zu lassen. Ein weiterer Mangel der sich aus dem fehlenden Besitzer eines Objektes ergibt, ist der Umstand, daß die von einem migrierten Agenten belegten Ressourcen nicht freigegeben werden [LO98]. Es ist Aufgabe der automatischen Speicherverwaltung, Objekte aus dem Speicher zu entfernen. Kein Objekt ist in der Lage, ein anderes Objekt aus dem Speicher zu entfernen. Solange ein Objekt referenziert wird, verbleibt es im Speicher.

Die Einbeziehung des Agentenparadigmas in die Sicherheitsarchitektur ist nur unvollständig vollzogen worden. Als äußerst problematisch ist zu bewerten, daß der Schutz des Agenten vor dem Wirtssystem in der Sicherheitsarchitektur keine Beachtung findet. In keiner veröffentlichten Dokumentation geht Sun auf die Probleme von Agenten ein, die auf unbekannten Rechensystemen eingesetzt werden. Dabei sollte das Beispiel des CCC-Angriffs auf die Internet-Banking Anwendung von Brokat deutlich gemacht haben, welche Konsequenzen der Einsatz von Agententechnologien hat. Der Schutz des Wirtssystemes vor Agenten, sowie der Schutz der Agenten untereinander weisen eine Reihe von Design- als auch Implementationsmängel aufn. Die konkreten Probleme werden in den nachfolgenden Abschnitten behandelt.

Designprobleme

Die Forschungsgruppe um FELTEN hat eine Reihe von Designfehlern in Java ausgemacht:

- ▷ Das derzeit größte Problem ist die bisher nicht spezifizierte Sicherheitsarchitektur. Für eine nicht spezifizierte Sicherheitsarchitektur kann nicht bewiesen werden, daß es korrekt ist, da es keine Grundlage hierfür gibt.

- ▷ Die Sicherheit des Java Laufzeitsystems hängt davon ab, daß die Integrität des Typsystems gewahrt bleibt. Die Integrität dessen bleibt dann gewahrt, wenn der Security-Manager, Classloader und Verifier korrekt implementiert sind. Bytecode wird als gültig bezeichnet, wenn er den Bytecodeverifier durchlaufen hat. Eine formale Definition der Korrektheit eines Bytecodes existiert nicht.

- ▷ Die Sicherheit des Java-Systems hängt von der Korrektheit des Security-Managers, des Class-Loaders und des Bytecode-Verifiers ab.

Darüberhinaus existieren eine Reihe weiterer wesentlicher Designprobleme in der Sprache, die es zweifelhaft werden lassen, daß Java geeignet ist, sichere Agentensysteme zu implementieren.

Die Designer der Sprache Java haben im Hinblick auf die Entwicklung des Sicherheitskonzeptes den Denial-of-Service-Angriffen keine besondere Bedeutung beigemessen. Eine Reihe von LADUE entwickelter Applets weisen daraufhin, daß es keines besonderen Aufwands bedarf, um derartige Angriffe durchzuführen. Der unbegrenzte Verbrauch von Ressourcen, z.B. Rechenzeit, ist in verteilten Umgebungen – wie dem Internet – nicht hinnehmbar. Eine Sprache wie Java, die den Anspruch erhebt, sichere Anwendungen

in verteilten Umgebungen zu ermöglichen, muß diese Gefahr adressieren und geeignete Mittel zur Verfügung stellen, um legitimierten Anwendungen den Zugriff auf Ressourcen zu erlauben, während gleichzeitig nicht autorisierten Anwendungen dieser Zugriff zu verwehren ist.

Die Entscheidung der Entwickler, Denial-of-Service-Angriffen im Sicherheitskonzept keine besondere Rolle zuzuordnen, ist insbesondere deshalb bedenklich, weil im zunehmenden Maße Angreifer sich dieser Technik bedienen, um fremde Systeme außer Kraft zu setzen. Ein Beispiel aus dem Jahr 1996 ist eine Angriffstechnik, die als *SYN-flooding* bezeichnet wird. Kurze Zeit später erfolgten weitere Denial-of-Service-Angriffe, die mittels großer Ping-Pakete arbeiteten und das angegriffene System zum Stillstand oder zu einem Reboot veranlassen konnten.[22] GARFINKEL und SPAFFORD führen eine Reihe von Denial-of-Service-Angriffen an, die leicht mittels Java implementiert werden können [GS97].

In die Erstellung der Sicherheitsarchitektur sind keine Erkenntnisse zur Verhinderung von Zeitkanälen eingeflossen. Die Mächtigkeit der Sprache, insbesondere die sich ständig vergrößernde Standardbibliothek, erlaubt es auf einfache Art und Weise, verdeckte Zeitkanäle aufzubauen. In sensitiven Anwendungen stellt dieses ein Problem dar, da es Agenten damit möglich ist, Informationen nach außen lecken zu lassen.

Java ist abhängig von der Integrität der DNS-Anfragen. Forschungsberichte haben gezeigt, daß DNS-Anfragen genutzt werden können, um die Sicherheitsarchitektur Javas zu kompromittieren [FBDW97]. DNS-Anfragen sind weiterhin mit dem Problem behaftet, daß sie Informationen außerhalb der Sicherheitsarchitektur lecken lassen. Zu weiteren DNS-Problemen sei auf Abschnitt 5.1.4 verwiesen.

Während der Ausführung des Konstruktors können andere Methoden des zu instantiierenden Objektes aufgerufen werden. Zu diesem Zeitpunkt ist das Objekt erst teilweise initialisiert. Damit können Methoden am Objekt aufgerufen werden, die von Subklassen überschrieben worden sind, ohne daß deren Konstruktor vollständig ausgeführt worden ist [DFWB97]. Es obliegt der Aufmerksamkeit des Programmierers, derartige Situationen zu vermeiden.

Ein Sicherheitsmerkmal Javas soll der Umstand sein, daß Applets keine Informationen über die Speicheradresse der Objekte erhalten können. DEAN et al. haben jedoch in [DFW96] dargelegt, daß die in der Bibliothek enthaltene Applet-Methode `hashCode()` die interne Speicheradresse des Objektes in eine Integer-Zahl konvertiert und als Ergebnis zurückliefert.

Im Abschnitt 5.3.3 wurde bereits darauf hingewiesen, daß die Änderungen an der Sicherheitsarchitektur Javas keine Modifikationen der JVM bedurften. Daraus folgt allerdings, daß alle Sicherheitsmängel der JVM, die nicht aus Implementations- sondern auf Designmängeln bestehen, weiterhin existieren.

[22] Näheres hierzu wurde im Abschnitt 5.1.4 ausgeführt.

Typsicherheit/Typsystem

Fundamentaler Bestandteil der Sicherheitsarchitektur ist die Typsicherheit. Sie schützt die Integrität des Speichers. Die Typsicherheit garantiert, daß keine Fehlinterpretationen von Daten erfolgen. Nur solange sichergestellt werden kann, daß keine beliebigen Speicherzugriffe erfolgen können, kann die Zugriffskontrolle an wohldefinierten Einstiegspunkten gewährleisten, daß lediglich privilegierte Klassen Zugriff auf kritische Systemressourcen erhalten [WF98]. Durchläuft ein Javacode erfolgreich die Typüberprüfung besteht eine begründete Annahme, daß es keine Typ-Konfusionen gibt – eine Variable kann zur Laufzeit nicht ihren Typ ändern. Kann die Typsicherheit nicht gewährleistet werden, ist die gesamte Architektur kompromittiert. SARASWAT legt in seiner Arbeit dar, daß die Typsicherheit Javas nicht gegeben ist. Die Folge dessen ist, daß

- ▷ Java Objekte private Felder anderer Objekte lesen und modifizieren können;
- ▷ interne Datenstrukturen der JVM ausgelesen werden können;
- ▷ nicht definierte Methoden an Objekten aufgerufen werden können, mit nicht vorhersehbaren Ergebnissen [Sar97].

Möglich wird dieses durch die Ausnutzung von Mängeln im Typsystem Javas. Durch die Wahl geeigneter Klassen ist ein *Typ-Spoofing* möglich. Die von SARASWAT dokumentierte Vorgehensweise sei nachfolgend beschrieben.

Gegeben seien zwei Klassen A und A', deren Full Qualified Name (FQN) identisch ist. Die Klassen werden durch die Class-Loader-Objekte C und C' geladen. Instanzen der Klassen A und A' werden somit in unterschiedliche Namensräume geladen. Eine weitere Klasse D kann alle Methoden der Klasse A' aufrufen. D enthält eine Variable v, die jede beliebige Instanz von A' aufnehmen kann.

Darüberhinaus existiert eine weitere Klasse B, die es Agenten aus anderen Namensräumen erlaubt, Instanzen der Klasse A zu beziehen.[23] Bezieht die Klasse D nun eine Instanz der Klasse A über die Brücken-Klasse B, geht die empfangende Klasse D davon aus, daß es sich um eine Instanz der Klasse A' handelt. Dieses ist jedoch nicht der Fall, übergeben wird eine Instanz der Klasse A mit der Folge, daß sie so behandelt wird, als handele es sich um eine Instanz der Klasse A'. Die JVM benutzt die Signatur der Klasse A' um auf eine Instanz der Klasse A zu operieren.

Der Zugriff auf ein Attribut und der Aufruf einer Methode an einem Objekt haben zur Folge, daß die JVM überprüft, ob diese Operationen für das angegebene Objekt zulässig sind. Daraus wäre der Schluß zu ziehen, daß der oben beschriebene Vorgang des Type-Spoofing nicht möglich ist. Wie SARASWAT festgestellt hat, fließen in den Entscheidungsprozeß keinerlei Laufzeitinformationen ein, so daß sich die Signatur der Instanz einer Klasse von der Signatur der Klasse unterscheiden kann. Der JVM ist es

[23] Die Klasse B wird von SARASWAT als „Bridge“ bezeichnet [Sar97].

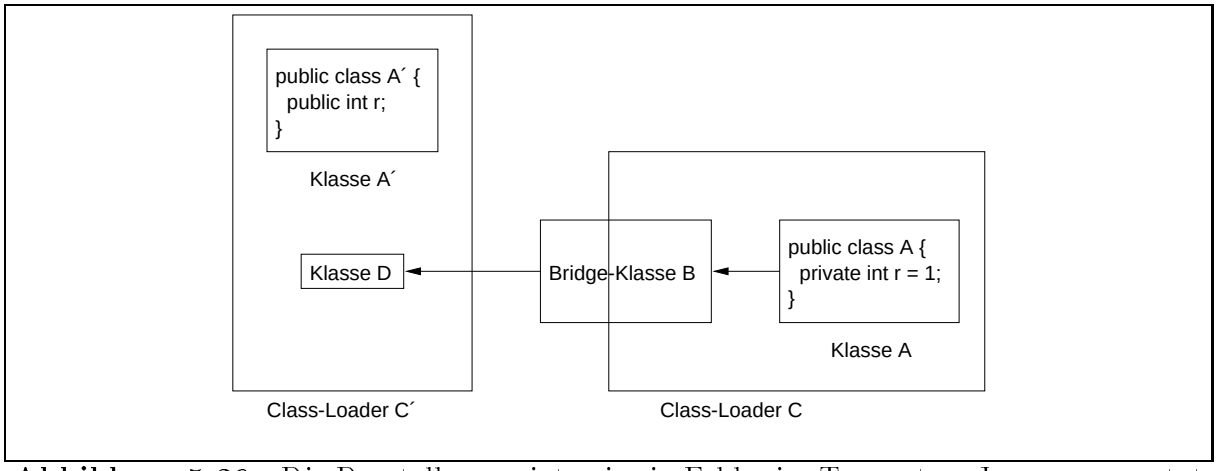

Abbildung 5.26.: Die Darstellung zeigt, wie ein Fehler im Typsystem Javas ausgenutzt werden kann. Über eine *Bridge*-Klasse kann die Klasse D eine Instanz der Klasse A beziehen und diese wird vom Typsystem behandelt, als ob es sich um eine Instanz der Klasse A' handeln würde.

zur Laufzeit nicht möglich, derartige Unterschiede festzustellen. Genau diesen Mangel macht sich das oben beschriebene Verfahren des Type-Spoofing zu Nutze.

Ist die Instanz der Klasse A kleiner als die Instanz der Klasse A', haben Referenzierungen von Attributen der Klasse A' zur Folge, daß auf Speicherbereiche zugegriffen wird, die außerhalb der Klasse A liegen. Desweiteren können in der Klasse A' Methoden definiert sein, die in der Klasse A nicht definiert sind.

Typ-Spoofing läßt sich verhindern, wenn ein Typ als Paar (Name, Class-Loader) definiert ist. In [LY97] wird eine Java-Typ entsprechend definiert, die Definition wird allerdings weder von den Java-Compilern noch von der JVM umgesetzt [Sar97].

Im Gegensatz zu der Java-Dokumentation, in der behauptet wird, daß die Typsicherheit statisch zur Ladezeit gesichert werden kann, müssen zur Laufzeit noch Typüberprüfungen stattfinden [DFW96]. Daraus ist der Schluß zu ziehen, daß der Verifier nicht die einzige Komponente des Laufzeitsystems ist, deren Korrektheit garantiert sein muß.

Implementationsmängel

Aus der Sicht einer Sicherheitsbetrachtung sind nur Systeme einsetzbar, die ein Stadium der Konsolidierung erreicht haben. Dieses trifft insbesondere auf die Agentensystem nicht zu, die auf Java basieren. Das letzte JDK hat innerhalb eines Jahres die Versionen 1.1.1 bis 1.1.7 durchlaufen. Unbeachtet bleiben bei dieser Darstellung die zusätzlichen Patches, die aufgrund von Sicherheitslücken veröffentlicht werden mußten. Innerhalb dieser Entwicklung beschränkte man sich nicht auf eine Behebung von Systemmängeln, sondern erweiterte das System um weitere Funktionalitäten. Unter diesen Voraussetzungen ist Java eine denkbar schlechte Wahl als Basis eines Agentensystemes, das in heterogenen, offenen, verteilten Umgebungen agieren wird, und damit eines Sicherheitskonzeptes

bedarf, das den damit einhergehenden Anforderungen gerecht wird.

Eine Reihe von Sicherheitslücken, die aus Implementationsfehlern resultierten, sind innerhalb kürzester Zeit nach ihrer Entdeckung behoben worden. Demgegenüber steht die Tatsache, daß mittlerweile Millionen Implementationen der JVM auf Rechnern installiert sind. Nur einer Minderheit wird die Bedeutung der Sicherheitslücken bewußt sein und noch eine kleinere Zahl von Installationen wird stets mit dem neuesten Patch aktualisiert. Darüberhinaus ist davon auszugehen, daß neue Programm-Versionen bzw. Patches die Einführung neuer Fehler bedeuten, die unter Umständen zu neuen Sicherheitslücken führen oder alte wieder eröffnen. Das derzeitige Sicherheitskonzept der Java-Implementierungen zieht nach sich, daß entdeckte Sicherheitslücken nur durch die vollständige Aktualisierung aller Clients geschlossen werden können.

Bisher ist nicht bekannt geworden, daß die Sicherheitslücken dazu genutzt worden sind, um einen Angriff mittels eines maliziösen Agenten durchzuführen.

Bytecode

Der Bytecode enthält ein strukturelles Problem, das nicht leicht zu lösen ist. Hersteller von Applikationen erwarten, den Schutz ihres geistigen Eigentums. In Java implementierte Anwendungen legen dagegen die komplette Implementation offen. Jedem Interessierten stehen eine Reihe von Decompilern zur Verfügung, mit denen der Code einer Java-Anwendung wiederhergestellt werden kann. Von Interesse ist dieser Umstand insbesondere im Zusammenhang mit den Implementationen der Sicherheitsarchitekturen. Alle Komponenten der Sicherheitsarchitekturen lassen sich auf einfache Art und Weise aus den mitgelieferten Klassendateien wiederherstellen.

Die wohldefinierte Struktur des Bytecodes und die dort enthaltenen Detailinformationen lassen das Reengineering von Bytecode zu Java-Code zu einer einfachen Aufgabe werden. Neben dem Verlust eines Wettbewerbsvorteils durch die Offenlegung des Wirtschaftsgutes Agent besteht für Angreifer die Möglichkeit, Implementationsmängel zu entdecken und diese für Angriffe auszunutzen. Schwerer wiegt die Gefahr, daß aufgrund der einfachen Struktur Bytecode manipuliert wird, indem direkt in den `class`-Dateien Methoden entfernt, hinzufügt oder manipuliert werden. Das Einfügen von Methoden in eine `class`-Datei wird durch die Unterstützung des `goto`-Befehls auf Bytecode-Ebene erleichtert. Die dabei enstehenden Inkonsistenzen bezüglich der Java-Sprachregeln werden durch den Verifier nicht erkannt [LaD97]. Wenn derartige Zugriffe auf die Klassen `SecurityManager` oder `AppletClassLoader` erfolgen können, gerät das Sicherheitskonzept Javas aus den Fugen.

Es existieren mittlerweile über 60 Sprachen, die Bytecode für die JVM erzeugen [Tol97].[24] Es ist unwahrscheinlich anzunehmen, daß alle Sprachen wohldefinierten – den Java-Sprachregeln gehorchenden – Bytecode generieren.

[24] Eine Zusammenfassung der für die JVM existierenden Sprachen ist unter `http://grunge.cs.tu-berlin.de/~tolk/vmlanguages.html` zu finden.

Tabelle 5.2.: Zugriffsrechte auf Java-Methoden [CW97]

Qualifier	Klasse	Subklasse	Package	World
private	X			
protected	X	X*	X	
public	X	X	X	X
package	X		X	

Package-Mechanismus

Der Package-Mechanismus Javas ist aus Sicht der Systemsicherheit sehr schlecht umgesetzt. Jede externe kann sich als Mitglied eines Packages deklarieren, das als zusätzliches lokales Modul im `CLASSPATH` enthalten ist. Mitglied einer Klasse zu sein, wird als optionales Attribut verwirklicht, wobei das Attribut einen beliebigen Wert annehmen kann, unabhängig von der Quelle, aus der die Klasse bezogen wurde [Bil96]. Klassen, die sich als Mitglied eines Packages ausgeben können, haben erweiterten Zugriff auf die Methoden der im Package enthaltenen Klassen. Neben den als `public` deklarierten Methoden haben Mitglieder eines Packages auch Zugriff auf alle Methoden der Klassen, die ohne Qualifizierung deklariert wurden. Die Integrität der Packages wird nicht gewährleistet.

In dem derzeitigen Java-System wird lediglich der Zugriff auf Variablen kontrolliert, nicht jedoch deren Sichtbarkeit.

Security-Manager

Der Security-Manager soll die Funktionalität eines Referenzmonitors beinhalten. Die Grundanforderungen an einen Referenzmonitor werden durch den Security-Manager nicht abschließend erfüllt:

Aktivitätsforderung Der Security Monitor muß explizit durch das Laufzeitsystem gestartet werden. Falls dies durch einen Implementierungsfehler nicht geschieht, werden alle Zugriffe prinzipiell zugelassen.

Verifizierbarkeit Der Security-Manager ist in Java geschrieben, einer Sprache, der eine formale semantische Definition fehlt.

Manipulationssicherheit Durch den *Superclass*-Angriff können private Variable des Security-Managers manipuliert werden [Kok97].

Der Aufruf des Security-Managers obliegt dem Programmierer. Eine Manipulation des Security-Managers kann mittels einer Angriffsmethode von HOPWOOD durchgeführt werden. Eine Verifikation des Security-Managers ist nicht möglich, da er in einer Sprache

implementiert ist, deren Semantik formal nicht definiert ist. Laut GASSER gibt es bislang kein größeres System, das alle drei Prinzipien eines Referenzmonitors vollständig erfüllt [Gas88]. Dementsprechend ist auch von Java nicht zu fordern, daß eine vollständige Entsprechung der Anforderung vorhanden ist. Es bleibt allerdings anzumerken, daß keines der Prinzipien vollständig erfüllt wird und es damit verschiedene Wege gibt, das Konzept des Referenzmonitors außer Kraft zu setzen.

Die Modifikationen der Sicherheitspolitik durch Anpassung der Implementation des Security-Managers war in den JDKs 1.0 und 1.1 die einzige Möglichkeit, mit der Einfluß auf die durchgesetzte Sicherheitspolitik genommen werden konnte. Es kann jedoch nicht Aufgabe eines Entwicklers sein, die Sicherheitspolitik einer Anwendung festzulegen. Vielmehr sollte die Festlegung der Sicherheitspolitik ausschließlich durch ausgebildete Sicherheitsspezialisten erfolgen. Systemadministratoren sollten beispielsweise in der Lage sein, eine der Anwendung angemessen Sicherheitspolitik aufzustellen. Die Implementation dieser in Form einer Subklasse des Security-Managers ist jedoch ein inakzeptabler Weg. Unterschiedliche Sicherheitsanforderungen verschiedener Nutzer würden erfordern, daß für jede Benutzergruppe ein eigener Security-Manager implementiert wird, der die angemessene Sicherheitspolitik durchsetzt. Das mit Java 2 neu eingeführte Konzept des Access-Controllers erlaubt eine einfachere Einflußnahme auf die durchzusetzende Sicherheitspolitik. Entscheidungen bezüglich der Sicherheitspolitik werden nun nicht mehr zum Zeitpunkt der Kompilation des Quelltextes sondern zur Laufzeit durchgeführt.

Denial-of-Service-Angriffe werden durch den Security-Manager nicht verhindert. Es können beliebig große Speicherbereiche alloziiert und unbeschränkt viele Threads erzeugt werden.

Verifier

Die Implementation des Verifiers ist nicht trivial. Die Spezifikation der JVM umfaßt 475 Seiten. In der Spezifikation sind 700 Axiome enthalten. Das AppletSecurity-System ist 500 Zeilen lang, der Bytecode Verifier ist siebenmal so groß. Somit versucht eine komplexe Softwarekomponente zu verifizieren, daß ein Bytecode ein legales Java Programm repräsentiert. Solange jedoch keine formale Beschreibung des Bytecodes existiert, kann nicht garantiert werden, daß der Verifier korrekt funktioniert.

Werden Fehler oder Schwächen im Verifier entdeckt, bedeutet das, daß alle Implementationen der JVM mit fehlerhaftem Verifier aktualisiert werden müssen. Es ist leicht einzusehen, daß ein derartiger Prozeß nur in kleinen, lokalen Gruppen konsequent durchgeführt werden kann. In großen Netzen ist davon auszugehen, daß eine durchgeführte Aktualisierung nicht alle Teilnehmer umfaßt und somit veraltete und damit mit Sicherheitslücken versehene Versionen im Einsatz bleiben.

Die Analyse des Bytecodes ist relativ schwierig, da dieser in linearer Form vorliegt. Eine globale Analyse des Datenflusses ist hier erforderlich, die durch Ausnahmen und Ausnahmebehandlungen weiter erschwert wird. Normalerweise erfolgt die Typüberprüfung durch den Compiler, der den abstrakten Syntaxbaum analysiert. Damit erhöht sich

die Komplexität des Verifiers, der sicherzustellen hat, daß der Bytecode den aufgestellten Regeln genügt.

Der Verifier beinhaltet desweiteren eine Reihe von Mängeln. So ist es dem Verifier nicht möglich zu entscheiden, ob ein Bytecode von einem Compiler generiert wurde. Bytecode kann manipuliert werden, ohne daß der Verifier die Manipulationen erkennt und dementsprechend schreitet er auch nicht ein. Der Verifier generiert keine Ausnahme. Stattdessem wird eine Core-Dump generiert. Desweiteren kann zur Laufzeit Bytecode generiert werden, gegen den statisch nur schwierig Sicherheitsgarantien geprüft werden können [VS97].

Der Verifier überprüft in seiner ersten Phase, daß einer `Class`-Datei keinerlei Code entfernt oder zugefügt wurde. Modifikationen des Bytecodes werden durch den Verifier jedoch nur dann erkannt, wenn die externe Länge nicht mit der Summe der internen Längenangaben übereinstimmt. Diese Form der Überprüfung kann damit keine Modifikationen entdecken, bei denen die Längenangaben entsprechend angepaßt werden. LaDue hat in praktischen Beispielen demonstriert, daß eine entsprechende Manipulation des Bytecodes ohne großen Aufwand durchführbar ist [LaD97].

Die Spezifikation der JVM führt drei Eigenschaften auf, die jede Ausnahmebehandlung zu erfüllen hat:

- ▷ Die Bereiche von Instruktionen, die durch Ausnahmebehandlungen geschützt werden, müssen disjunkt sein oder einander enthalten;
- ▷ Eine Ausnahmebehandlung kann nicht innerhalb eines Codes erscheinen, der durch sie geschützt wird;
- ▷ Die Programmkontrolle kann einer Ausnahmebehandlung durch keinen anderen Grund als einer Ausnahme übergeben werden [LY97].

Die angeführten Eigenschaften werden jedoch nicht durch den Verifier durchgesetzt. Sun geht davon aus, daß sie keine Bedrohung für die Integrität der JVM darstellen [LY97].

Eine weitere Aussage der JVM Spezifikation beinhaltet, daß ein beliebiger Bytecode an eine `class`-Datei angehängt werden kann, ohne daß dieses den Verifikationsprozeß beeinflußt. Angehängte Opcodes werden nicht konsistent überprüft. Nach Untersuchungen von LaDue werden sie manchmal inspiziert und manchmal nicht. LaDue hat in [LaD97] nachgewiesen, daß eine nicht dem Standard entsprechende Ausnahmebehandlung in Kombination mit der auf Bytecodeebene zugängliche `goto`-Instruktion genutzt werden kann, um das Sicherheitskonzept Javas zu unterlaufen.

Der Verifier tritt in der Standardeinstellung der JVM-Implementation Suns lediglich für nicht lokale Klassen in Aktion. Lokale Klassen werden nicht durch den Verifier überprüft. Erst der Aufruf der JVM mit der Option `-verify` veranlaßt diese dazu, für jede Klasse den Verifier zu konsultieren. Hiermit ist erneut ein Punkt vorhanden, in dem deutlich wird, daß lokale Klassen nicht nur als vertrauenswürdig sondern auch als sicher betrachtet werden. Es wurde bereits mehrfach darauf hingewiesen, daß dieses eine nicht

zulässige Annahme ist. Problematisch ist weiterhin, daß dem Anwender mit der Option `-noverify` die Möglichkeit eröffnet wird, die Verifikation von Klassen vollständig zu deaktivieren.

JDK 1.0

Die Sicherheitsarchitektur Javas in der JDK-Version 1.0 zielt auf den Schutz des Umgebungssystems vor direkten Angriffen. Kein Schutz erscheint für Klassen notwendig, die aus entfernten Quellen bezogen und in den lokalen `CLASSPATH` abgelegt werden. Alle dort enthaltenen Klassen gelten als vertrauenswürdig, so daß aus Sicht von Sun hier keine weiteren Schutzmechanismen erforderlich sind. Es findet keine Unterscheidung zwischen sicheren und vertrauenswürdigen Erweiterungen der lokalen Klassen statt [Bil96]. Vertrauenswürdige Klassen bringen beispielsweise das Problem mit sich, daß sie von Angreifern für datengesteuerte Angriffe genutzt werden können. Implementationsmängel bieten eine Reihe von Möglichkeiten, mit denen Angreifer in die Lage versetzt werden, Angriffe auf Agentensysteme durchzuführen und dabei vertrauenswürdige Klassen zu nutzen.

JDK 1.1

Die Einführung eines neuen Systemcodes für die Sicherheit beinhaltet implizit die Gefahr neuer, bislang nicht existenter Implementationsmängel.[25] Die Signierung von Agenten erweitert das Funktionalitätsspektrum und steigert *nicht* die Sicherheit [BF97]. Codesignierung bietet die Basis, feiner granulierte Zugriffskontrollen zu erlauben und dem Anwender mehr Informationen über den auszuführenden Agenten bereitzustellen. Eine Erhöhung der Systemsicherheit ist damit jedoch nicht verbunden.

Die Implementierung eines korrekten und funktionalen Algorithmus zur Erweiterung der Rechte für signierte Agenten ist kompliziert. Eine erste Implementation im JDK 1.1 weist eine Sicherheitslücke auf, die es signierten Agenten, denen nicht vertraut wird, ermöglicht, die Sandbox zu verlassen [MF97].

Es darf nicht aus dem Blickwinkel geraten, daß vertrauenswürdige Agenten nicht mit sicheren Agenten gleichzusetzen sind. Die Signierung eines Agenten gibt keine Auskunft über die Funktionalität eines Agenten [Bil96].

Java 2

Die neue entwickelte Sicherheitsarchitektur Javas, wie sie in Java 2 implementiert ist, bringt eine Reihe von Verbesserungen für die Systemsicherheit mit sich. Daneben existieren allerdings weiterhin die Designfehler und -schwächen, die in den vorangegangenen Abschnitten dargelegt wurden. Auch in der neuesten Version stellt die JVM keine un-

[25] Ebensolche wurde kurz nach Veröffentlichung des JDK 1.1 entdeckt. Siehe hierzu `http://www.cs.princeton.edu/sip/new/april29.html`.

mittelbaren Mittel bereit, mit denen ein Auditing der sicherheitsrelevanten Funktionen durchgeführt werden kann.

Es hat sich weiterhin gezeigt, daß in den neuesten Versionen noch Implementationsfehler entdeckt werden, die seit Anbeginn in der JVM enthalten sind. So entdeckten FELTEN et al. im Juli einen Designfehler im Dynamic Linking-Mechanismus, der alle Versionen des JDK, einschließlich einer Beta-Version von Java 2, betraf. Der Fehler ermöglichte es, über einen illegalen Typcast das Typsystem Javas zu durchbrechen. Damit ist es einem Java-Agenten, der den Fehler ausnutzt, möglich, beliebige Operationen auf dem Zielsystem durchzuführen [Mur98].

Die Entwicklung Javas schreitet derart schnell voran, daß der Einfluß der neuen Komponenten, die in Java integriert werden (z. B. Servlet in Java 2), auf die Sicherheit nicht abgeschätzt werden kann [VS97].

Intensivere Evaluierungen der Implementationen der Java Virtual Machine (JVM) sind erforderlich, damit zukünftig sichergestellt werden kann, daß nach Veröffentlichung einer Implementation nicht innerhalb kürzester Zeit eine Vielzahl von Sicherheitslücken gefunden werden. Dazu gehört, daß die Entwicklungsgeschwindigkeit der Produkte gedrosselt wird, bzw. zweigleisige Entwicklung betrieben wird, in dem Sinne, daß eine Konsolidierung der bestehenden Produkte durchgeführt wird, während eine „experimentelle" Version die neuen Erweiterungen enthält. Eine frühzeitige Freigabe der experimentellen Version würde Sicherheitsüberprüfungen durch externe Experten ermöglichen, die dann in die konsolidierte Fassung einfließen können. Dieses Entwicklungsmodell wird beispielsweise im Rahmen des Betriebssystems Linux eingesetzt.

DEAN et al. stellen in [DFW96] abschließend fest, daß das Java-System ohne massive Änderungen nicht zu sichern ist. Dies schließt ein Redesign der Sprache, des Bytecodes und des Laufzeit-Systems ein.

Anforderungen

BILLON hat eine Reihe von Anforderungen an folgende Implementationen der JVM gestellt, die zu erfüllen sind, wenn die Sicherheit des Javasystems erhöht werden soll. Folgende Anforderungen wurden aufgestellt:

- ▷ Kompilierte Java-Klassen im `CLASSPATH` müssen vor allen Lese- und Schreiboperationen geschützt werden. Lesezugriffe sind nur durch den Lademechanismus der JVM zu erlauben.

- ▷ Alle in Java Klassen spezifizierten Zugriffsbeschränkungen müssen respektiert werden.

- ▷ Nicht lokalen Klassen darf es nicht möglich sein, sich als Mitglied eines lokalen Packages zu definieren.

- ▷ Es muß sichergestellt sein, daß es sich bei dem Java-System um das originäre von Sun veröffentlichte System handelt und entsprechend auch um die Systemklassen, die von Sun veröffentlichten sind.
- ▷ Eine Dekompilation von Bytecode, wie es derzeit möglich ist, ist aus Sicht der Systemsicherheit und des Schutzes der Intellectual Properties nicht wünschenswert [Bil96].

BILLON stellt eine Architektur für eine sichere Java-Laufzeitumgebung vor, die den von ihm aufgestellten Sicherheitsanforderungen genügen soll.

Voyager

Mit Voyager wird das Software-Konzept des Agenten eingeführt, ohne die Sicherheitsarchitektur dahingehend zu erweitern. Die einzige Änderung besteht in der Erweiterung des Security-Managers um vier weitere Methoden.

In den bisherigen veröffenlichten Voyager Core-Paketen ist es möglich, in einem Voyager-Server `SystemListener` zu registrieren. Ein `SystemListener` implementiert die Methode `systemEvent()`, die durch den Voyager-Server jedesmal aufgerufen wird, wenn ein System-Event generiert wird. Auf diese Weise ist es Agenten und auch mobilen Objekten möglich, Kenntnis über alle System-Ereignisse zu erhalten, die auf einem Server produziert werden.[26]

In diesem Zusammenhang ist von besonderem Interesse, daß die zu einem Systemereignis gelieferten Informationen den GUID des Objekts enthalten, das das Ereignis ausgelöst hat. Mittels eines GUID und der Adresse des Voyager-Servers, auf welchem das Objekt sich momentan aufhält, kann dieses Objekt referenziert werden und seine öffentlichen Methoden können aufgerufen werden.

```
SystemEvent( sending Sync( __ack(ILcom.objectspace.voyager.Address;)V,
127.0.0.1:9000/621048253 ->
127.0.0.1:1336/91-130-128-112-2-173-145-14-78-45-129-128-128-129-133-184)
127.0.0.1:1336/91-130-128-112-2-173-145-14-78-45-129-128-128-129-133-184
(166 bytes ) at Thu Feb 19 12:41:15 PST 1998 [127.0.0.1:9000] )
```

Abbildung 5.27.: Eine Voyager-Umgebung erzeugt Systemereignisse, in denen eine Vielzahl von Informationen enthalten sind. Die für Angreifer wichtigste Information ist dabei der GUID des Objekts, das das Ereignis ausgelöst hat. In diesem Fall lautet der GUID des Objektes `91-130-128-112-2-173-145-14-78-45-129-128-128-129-133-184`.

Es sollte beachtet werden, daß auch eine Erweiterung des Sicherheitskonzeptes nur wenig Abhilfe verspricht. Sollte es zukünftig nicht mehr möglich sein, `SystemListener`

[26] Siehe hierzu auch Anhang A.1.

auf entfernten Systemen zu registrieren, bleibt die Möglichkeit offen, dieses auf dem lokalen System durchzuführen. Die Durchführung eines Methoden-Aufrufs an einem entfernten Objekt durch ein lokales Objekt generiert System-Ereignisse, die den GUID sowie die Adresse des entfernten Objektes enthalten. Es ist damit auch bei einem modifizierten Sicherheitskonzept möglich, entfernte Objekte zu referenzieren und öffentliche Methoden an diesen aufzurufen.

Dem OO-Paradigma entsprechend können an einem Objekt nur Methoden aufgerufen werden, die im Klassentext des Objektes definiert und über die Klassenhierarchie vererbt wurden. Daraus folgt, daß an einem Objekt keine Methoden aufgerufen werden können, die erst in einer abgeleiteten Klasse definiert werden. Voyager verletzt dieses OO-Paradigma durch die Möglichkeit, daß in der Methode `moveTo()` eine Callback-Methode angegeben werden kann. Durch die Registrierung eines ObjectListeners an Objekten können deren Methoden „erschnüffelt" werden und indirekt über die Methode `moveTo()` aufgerufen werden.

Mobile Agenten haben keine Möglichkeit, eine eigene Sicherheitspolitik durchzusetzen. Die öffentlichen Methoden der Agenten können von allen anderen Entitäten aufgerufen werden, solange dies nicht durch die Sicherheitspolitik des Servers unterbunden wird. Auch in Voyager gibt es keine Besitzer-Metapher. Agenten stehen keine Mittel bereit, mit denen sie beeinflussen können, wer ihre öffentlichen Methoden aufrufen darf.

ObjectListener können einem Voyager-Objekt als Assistent zugefügt werden. Migriert das Voyager-Objekt, wird es durch den Assistenten begleitet. Gleiches gilt für den Fall, daß das Objekt sich persistiert. Die Directory-Klasse erlaubt die Erstellung von Verzeichnissen. Diese können dazu genutzt werden, „Yellow Pages" von Objekten zu erstellen. Auf diese Weise können Objekte Referenzen auf andere Objekte erhalten.

Jedes Objekt, das nicht auf dem gleichen Rechner und gleichem dem Port wie der Voyager-Server läuft, wird als „Foreign" betrachtet und unterliegt damit den Restriktionen des Security-Managers, so dieser installiert ist. Es findet keine weitere Differenzierung statt. Als direkte Folge dieser Sicherheitspolitik ist es allen Entitäten möglich, eine Reihe von sicherheitsrelevanten Methoden an allen Voyager-Objekten aufzurufen.

Die Sicherheitsarchitektur kann nur als äußerst lückenhaft bezeichnet werden. Im jetzigen Entwicklungszustand sind eine Vielzahl von Methoden aufrufbar, die eine große Gefährdung der Sicherheit des Rechensystemes darstellen. Zu diesen Methoden zählen u. a.:

VObject.dieNow() Der Aufruf der Methode `dieNow()` hat zur Folge, daß das Objekt, an dem diese Methode aufgerufen wird, zerstört wird. Die Methode ist bereits für `VObject`, der Wurzel aller Voyager Objekte, implementiert. Der Aufruf führt *nicht* zur Konsultation des Security-Managers, weshalb es möglich ist, diese Methode an jedem Voyager Objekt aufzurufen und es damit zu zerstören.

VObject.moveTo() Die Methode `moveTo()` ist eine der elementaren Methoden, die für die Implementation eines Agentensystemes benötigt wird. An dieser Methode wird

gleichzeitig deutlich, welche Konsequenzen aus dem Umstand entstehen, daß das neue Software-Paradigma des Agenten nicht in das Sicherheitskonzept Voyagers einbezogen wurde.

1. Jedes Voyager-Objekt bzw. -Agent kann auf jeden Voyager-Server migrieren. `moveTo()` gehört *nicht* zu den Methoden, die vor ihrer Ausführung eine Konsultation des Security-Managers nach sich ziehen. Entsprechend ist innerhalb der Voyager-Umgebung die Migration von Agenten/Objekten auf einen Server nicht zu verhindern.
2. Ist das Ziel einer Migration ein Voyager-Objekt, hat dieses zur Folge, daß dieses Objekt solange in seiner Ausführung blockiert, solange das migrierte Objekt seine Callback-Methode nicht beendet hat. Dementsprechend können Voyager-Objekte auf ewig blockiert werden, wenn diese zum Migrationsziel von Dummy-Objekten werden, die ihre Callback-Methode nie beenden.
3. Eine weitere Folge des Voyager-Sicherheitskonzept ist der Umstand, daß die `moveTo()`-Methode eine öffentliche Methode ist und damit von jeder Instanz einer Klasse aufgerufen werden kann. Dementsprechend können Agenten/Objekte auf einen Voyager-Server „entführt“ werden.
4. Der Methode `moveTo()` kann ein optionaler Parameter übergeben werden, der die Callback-Methode angibt, die bei Ankunft des Agenten/Objektes auf einem Voyager-Server auszuführen ist. Als Parameter kann eine String-Konstante oder eine Variable eingesetzt werden. Diese Implementierung erlaubt es, an `VObject`-Objekten, dem Wurzel-Objekt des Voyager-Systems, Methoden aufzurufen, die durch abgeleitete Klassen definiert werden.

Jedes Voyager Objekt kann die Methode `moveTo()` nutzen, um auf ein entferntes System zu migrieren. Dem entfernten System ist es nicht möglich, die Migration des Objektes auf sein System zu verhindern.

VObject.removeAssistant() Voyager Objekte können sich mittels der vordefinierten Methode `addAssistant()` an einen nicht transienten Objekt-Listener registrieren. Diese Registration kann durch die Methode `removeAssistant()` aufgehoben werden.

Voyager.addSystemListener() Voyager Server produzieren eine Vielzahl von Events, die an registrierte SystemListener weitergeleitet werden. Jedem Voyager Objekt ist es möglich, einen SystemListener an einem Voyager Server zu registrieren. Auf diese Weise kann jedes Objekt u. a. Kenntnis über globale Identifier erhalten, die zur Referenzierung von Voyager Objekten benötigt werden.

Das Voyager Core-Paket unterstützt keinerlei Audit-Mechanismen. Auf Seiten des Servers werden keinerlei Log-Dateien erzeugt, die Auskunft darüber geben, welche Prozesse auf welche Ressourcen zugreifen.

Die Dokumentation, die mit Voyager ausgeliefert wird, kann im Hinblick auf die dort enthaltenen Informationen zum Thema Sicherheit nur als unzureichend bezeichnet werden. Auf lediglich sechs Seiten wird das Thema Sicherheit abgehandelt. Eine Reflektion der aus dem Agenten-Paradigma resultierenden Gefahren erfolgt in den Abschnitten nicht. Selbst eine Darstellung der Methoden, die durch die Konsultation des Security-Managers als sensitiv eingestuft werden, erfolgt nicht.

Castanet

In [Mar97a] wird die Behauptung aufgestellt, daß signierter Software gegenüber das gleiche Vertrauen aufgebracht werden kann, wie dieses für versiegelte Software aus Fachgeschäften der Fall ist. Die obigen Ausführungen haben bereits deutlich werden lassen, welche Möglichkeiten bestehen, signierte Software bereitzustellen, der der Benutzer fälschlicherweise vertraut. Im Gegensatz zum Verkauf versiegelter Produkte in Fachgeschäften, ist es im Internet jedem auf einfache Weise möglich, Software anzubieten. Mit geeigneten Mitteln der Darstellung kann die signierte Software auf einem Web-Server ebenso präsentiert werden, wie dieses durch große Softwareunternehmen wie IBM oder Microsoft erfolgt.

Castanet beinhaltet die Möglichkeit, daß nur Teile eines Channels signiert werden [Mar97a]. Damit soll sichergestellt sein, daß Channels signiert werden können. die während ihres Lebenszykluses modifiziert werden. Das teilweise Signieren eines Channels kann dazu genutzt werden, um maliziösen Code mit einem vertrauenswürdigen Channel von Dritten zu kombinieren. Auf diesem Weg kann maliziöser Code in das System gelangen, obwohl der Channel durch eine Entität signiert ist, der der Benutzer vertraut.

5.4. Zusammenfassung

Die vorhergehenden Abschnitte haben gezeigt, daß die Sicherheitsarchitekturen der Basistechnologien nicht ausreichend sind, um den Gefahren in Agentensystemen erfolgreich zu begegnen. Daraus folgt für die vorgestellten Agentensysteme, deren Sicherheitskonzepte ausschließlich auf den hier dargelegten Sicherheitsmechanismen beruhen, daß die Sicherheitsanforderungen für die Anwendungsgebiete nur unzureichend erfüllt werden können. Folglich bedarf es erweiterter Sicherheitsarchitekturen, denen es gelingt, den spezifischen Sicherheitsanforderungen von Agentensystemen gerecht zu werden.

6. Sicherheitsbetrachtung von Sicherheitsarchitekturen für Agentensysteme

„*From a security perspective, there is nothing more dangerous than a global, homogeneous general-purpose interpreter.*“
– Rubin & Geer

Die Ausführungen im vorgehenden Kapitel haben deutlich werden lassen, daß die Sicherheitsmechanismen der Basistechnologien nicht ausreichend sind, um einen umfassenden Schutz für Agentensysteme zu bieten. Insbesondere der Schutz des Agenten vor dem Wirtssystem wird von den bislang vorgestellten Sicherheitsmechanismen nicht adressiert. Zu den offenen Sicherheitsanforderungen gehören:

- ▷ Vertrauliche Kommunikation zwischen Agenten
- ▷ Schutz des Agenten vor der Umgebung
- ▷ Privatheit des Agentencodes, der Agentendaten, der Ausführung
- ▷ Integrität der Ausführung
- ▷ Schutz vor Diensteverweigerung
- ▷ Monitoring des Ressourcenverbrauchs
- ▷ Eindeutige Identifikation des Urhebers von Operationen
- ▷ Aufzeichnung sicherheitsrelevanter Operationen
- ▷ Signierung von Dokumenten durch Agenten
- ▷ Anonymität der Agenten

Neben den nicht adressierten Sicherheitsanforderungen weisen die bislang vorgestellten Sicherheitsmechanismen Mängel auf, die dazu führen, daß die erfüllten Sicherheitsmerkmale nur unzureichenden Schutz für Agentensysteme bieten. Aus diesen Gründen wurden und werden Sicherheitsarchitekturen entwickelt, die einen weitergehenden Schutz vor Bedrohungen in Agentensystemen bieten sollen. In den folgenden Abschnitten werden die Sicherheitsarchitekturen konzeptionell vorgestellt und dahingehend untersucht, welchen Sicherheitsanforderungen sie gerecht werden.

Die nachfolgenden Sicherheitsarchitekturen müssen sich insbesondere daran messen lassen, inwieweit sie der neuen Sicherheitsdimension – dem Schutz des Agenten vor der Umgebung – entsprechen. CHESS et al., FARMER et al. und ORDILLE sind in ihren Arbeiten zu dem Schluß gelangt, daß sich der Schutz des Agenten vor der Umgebung nicht gewährleisten lasse [CGH+95, FGS96, Ord96]. Andere Untersuchungen sind zu dem Ergebnis gelangt, daß es Lösungen gibt, die den Schutz sicherstellen.

Die beiden anderen Sicherheitsdimensionen lassen sich scheinbar mit bereits existierenden Sicherheitslösungen abdecken, CHESS hat jedoch richtig darauf hingewiesen, daß diese Lösungen von einer Reihe von Annahmen ausgehen, die in Agentensystemen nicht länger gültig sind.[1] Zu diesen Annahmen zählt u. a., daß „nur Personen die dem System bekannt sind, im System Programme ausführen können [Che98]." In Agentensystemen veranlassen hingegen Produzenten von Agenten legitimierte Nutzer dazu, ihre Agenten auszuführen.

Allen Sicherheitslösungen für Agentensysteme muß gemeinsam sein, daß der Mehrwert, der durch den Einsatz dieser Technologie erzielt werden soll, nicht durch die Sicherheitsarchitektur verloren geht. Sollte sich am Ende herausstellen, daß die Sicherheitsarchitekturen die Vorteile der Agententechnologie in großem Maße einschränken, verliert die neue Technologie ihre Berechtigung. Eine wesentliche Motivation von Agenten ist die Nutzung vielfältiger Dienste, die von verschiedenen, konkurrierenden Dienstleistern in offenen Netzen angeboten werden. Eine Beschränkung der Offenheit ist gleichbedeutend mit der Aufhebung der wesentlichen Vorteile der Agententechnologie.

Ein weiterer Aspekt der zu untersuchenden Sicherheitsarchitekturen ist die Frage, ob sich mit ihnen Angriffe verhindern lassen – das Sicherheitssystem reagiert proaktiv – oder ob sich im Anschluß eines Angriffes dieser lediglich identifizieren läßt (reaktiv). Reaktive Sicherheitsarchitekturen sind nicht einsetzbar, wenn der Angreifer im Anschluß an den Angriff nicht länger identifizierbar ist [ST97b].

6.1. Digitale Signaturen

Digitale Signaturen sind eine wesentliche Komponente der Sicherheitsarchitekturen der im vorangegangenen Kapitel vorgestellten Basistechnologien von Agentensystemen. In diesem Abschnitt sollen digitale Signaturen noch einmal aufgegriffen werden, da sie in

[1] Siehe hierzu Kapitel 2.

einigen der nachfolgenden Architekturen zur Wahrung der Integrität und der Authentizität herangezogen werden.

6.1.1. Konzept

Digitale Signatursysteme nutzen i. d. R. asymmetrische Verschlüsselungsverfahren, um die Authenzität und die Integrität zu gewährleisten. Digitale Signaturen können in Agentensystemen eingesetzt werden, um Teile eines Agenten oder den ganzen Agenten mit einer Signatur zu versehen. Dazu wird ein Hashwert A_{Hash} über dem Agenten berechnet. Der Hashwert wird anschließend mit dem privaten Schlüssel S_S des Senders des Agenten verschlüsselt. Damit ist der Agent digital signiert und wird als Tupel $(A, S_S(A_{Hash}))$ versandt. Um die Vertraulichkeit des Agenten während der Übertragung zu wahren, kann der Agent zusätzlich mit dem öffentlichen Schlüssel des Empfängers E_P verschlüsselt werden. Dieser erhält dann vom Sender des Agenten die folgende Nachricht:

$$E_P(A, S_S(A_{Hash}))$$

Der Empfänger entschlüsselt den Agenten zunächst mit seinem privaten Schlüssel, so dieser vom Sender mit dem öffentlichen Schlüssel des Empfängers verschlüsselt worden ist. Anschließend berechnet der Empfänger den Hashwert über demAgenten. Nach der Ermittlung des öffentlichen Schlüssels des Senders kann der Empfänger den vom Sender berechneten Hashwert dekodieren und mit dem von ihm berechneten Wert vergleichen. Stimmen beide Werte überein, ist die Authentizität und Integrität des Agenten nachgewiesen.

Neben dem Sender des Agenten kann der Agent von einer Reihe weiterer Entitäten unterzeichnet werden. Hierzu zählen Entwickler und Hersteller von Agenten, sowie die Systeme, die ein Agent bereits in der Vergangenheit besucht hat.

Voraussetzung für den Einsatz digitaler Signaturen ist eine funktionsfähige Infrastruktur für öffentliche Schlüssel (Public Key Infrastructure (PKI)). Eine Signatur kann nur dann zur Überprüfung der Authentizität dienen, wenn der Empfänger den korrespondierenden öffentlichen Schlüssel erlangen kann. Wichtigster Bestandteil einer PKI sind zertifizierende Instanzen, die mittels eines Zertifikates die Korrespondenz eines öffentlichen Schlüssels mit einer Identität herstellen. Der Nutzer von Zertifikaten muß darauf vertrauen, daß die im Zertifikat enthaltenen Angaben korrekt sind. Aufbauend auf das Vertrauen in eine CA werden digitale Signaturen als Entscheidungsgrundlage für die Rechtevergabe in Agentensysteme herangezogen. Der Empfänger vertraut im wesentlichen darauf, daß er vom Sender einen Agenten erhält, der sich gutartig verhält.

6.1.2. Beurteilung

Digitale Signaturen bilden keine Sicherheitsarchitektur, sie sind vielmehr ein Werkzeug, das in Sicherheitsarchitekturen eingesetzt wird. Trotzdem werden digitale Signaturen

in Agentensystemen, die auf ActiveX- und Java-Agenten aufzen, häufig als einzige Entscheidungsgrundlage herangezogen, anhand der entschieden wird, welche Rechte Agenten erhalten.

Keine der oben angeführten Sicherheitsanforderungen kann durch den Einsatz digitaler Signaturen entsprochen werden. Digitale Signaturen können lediglich dazu dienen, die Authentizität einer Entität und die Integrität eines Codes verifizierbar zu machen. Eine Abdeckung der der drei Dimensionen des Sicherheitsbegriffes kann durch Sicherheitsmechanismen, die alleinig auf digitalen Signaturen aufbauen, nicht erbracht werden.

Digitale Signaturen können nur dann Anwendung finden, wenn die Basis – eine funktionierende Infrastruktur für öffentliche Schlüssel – gegeben ist. Bislang existieren keine Lösungen für eine PKI, die hinreichend skalierbar sind und sich nicht auf bestimmte Gruppen beschränken [Che98, GS97].

Der Einsatz von digitalen Signaturen erlaubt lediglich den Aufbau reaktiver Sicherheitsarchitekturen. Im Schadensfall ist es zwingend erforderlich, daß die den Schaden auslösende Entität ermittelbar ist. Es muß jedoch davon ausgegangen werden, daß eine zweifelsfreie Identifizierung des Urhebers eines Schadens nicht geleistet werden kann. Selbst wenn der Urheber sich ermitteln läßt, steht der Geschädigte in der Pflicht, den Vorgang zu beweisen. Agieren Täter und Opfer in unterschiedlichen Rechtsräumen, lassen sich die Rechte des Opfers nur selten durchsetzen.

Digitale Signaturen können in geschlossenen Agentensystemen, in denen jedes beteiligte System allen anderen vertraut, dazu dienen, die Authentizität eines Agenten sicherzustellen. Agenten migrieren lediglich auf Systeme, von denen bekannt ist, daß sie eine sichere Umgebung darstellen und keine Manipulationen an Agenten vornehmen. Zwischen den Umgebungen werden kryptografische Protokolle aufgebaut, so daß ein Agent auch während der Migration nicht angreifbar ist. Agenten aller teilnehmenden Systeme gelten als vertrauenswürdig, es werden von Agenten keine maliziösen Aktionen durchgeführt.

Derartige Lösungen bilden eine in sich geschlossene Welt. Es handelt sich um Systeme, in denen nur Teilnehmer gestattet sind, die dem System bekannt sind und als vertrauenswürdig eingestuft werden. Dieser Standard-Ansatz ist zu restriktiv [ST97a]. Ein wesentliches Merkmal mobiler Agenten ist die Operation in offenen Systemen. Wird die ihnen zur Verfügung stehende Welt lediglich auf eine Reihe vertrauenswürdiger Systeme beschränkt, gehen wesentliche Vorteile des Paradigmas der mobilen Agenten verloren (Autonomie, Mobilität). Neue Anbieter können in das geschlossene System nur dann aufgenommen werden, wenn alle bisherigen Teilnehmer dem neuen Anbieter vertrauen. Desweiteren ist nicht sichergestellt, daß einer der Teilnehmer sich nicht doch maliziös verhält. Es wären Kontrollen erforderlich, mit denen ermittelt werden kann, welcher der Teilnehmer sich möglicherweise maliziös verhalten hat.

Das Vertrauen in derartigen Systemen ist häufig absolut, d. h., es findet keine Differenzierung des Grades an Vertrauen statt, daß einem Agenten bzw. einer Umgebung entgegengebracht wird. Ein Beispiel hierfür sind ActiveX-Agenten. Verhält sich in einem entsprechendem Agentensystem trotz aller Annahmen ein Agent bzw. eine Umgebung

nicht gutartig, existieren keine Schutzmechanismen, die es erlauben, den Angriffen zu begegnen. Nicht gutartige Aktionen können bereits aus Implementationsmängeln resultieren. Agenten bzw. Umgebungen können ein vom Entwickler nicht intendiertes malizöses Verhalten aufweisen. Digitale Signaturen verhindern nicht, daß mangelhafte oder maliziöse Software eingesetzt wird.

Neue, bislang unbekannte Personen, Organisationen, Unternehmen können keine Reputation aufweisen, die anderen Entitäten als Basis für eine Vertrauensentscheidung dient.

Eine Differenzierung des Vertrauens in unterschiedliche Stufen führt zu dem Problem, daß eine Entscheidungsgrundlage existieren muß, anhand der Agenten in unterschiedliche Vertrauenskategorien eingeordnet werden können. Lösungen, wie sie in den Basistechnologien wie dem Communicator, Internet Explorer und Java existieren, sind bislang vollkommen unzureichend.

Im Falle eines neuen Agenten hat in diesen Systemen der Anwender alleine auf Grundlage des präsentierten Zertifikats zu entscheiden, welche Rechte dem Agentenugeteilt werden können. Zu diesem Zeitpunkt ist dem Anwender häufig unbekannt, welche Operationen durch den Agenten durchgeführt werden. Eine eingehende Überprüfung des Zertifikats ist zwingend erforderlich, verlangt jedoch ein hohes Fachwissen, über dasnur eine Minderheit der Nutzer von Agenten verfügt. Die enthaltenen Informationen können häufig zu Fehlentscheidungen verleiten. Welche Adresse ist die richtige `nasa.gov` oder `nasa.com`? Stimmt `microsoft.com` oder `MICROSOFT.COM`? An den Beispielen wird deutlich, daß selbst mit Fachwissen nur sehr schwer eine Entscheidung bezüglich der Qualität eines Zertifikats getroffen werden kann. Heutige Zertifikate enthalten keine ausreichenden Informationen, damit das Zertifikat eindeutig einer Person zugeordnet werden kann [GS97].

Digitale Signaturen können ausschließlich dazu dienen, statische Komponenten vor Manipulationen zu schützen. Zwischenergebnisse, die der Agent während seiner Reise ermittelt, können nicht digital signiert werden, da sie dann in nachfolgenden Berechnungen nicht verändert werden dürfen. Maliziöse Server könnten weiterhin Ergebnisse entfernen. Deshalb wäre eine Numerierung für die Zwischenergebnisse einzuführen, die sicherstellt, daß Resultate nicht entfernt werden können.

Das Vertrauen in eine Person und das Vertrauen in von einer Person geschriebene Programme sind zwei völlig unterschiedliche Ebenen des Vertrauens. Vertraut Person A darauf, daß Person B gutartige Agenten implementiert, ist für Person A daraus nicht zwangsläufig der Schluß zu ziehen, daß den Programmen das gleiche Vertrauen entgegengebracht werden kann. So kann sich A nicht sicher sein, daß B angemessen mit seinen privaten Schlüssel verfährt, mit deren Hilfe die Agenten digital signiert werden. Ebenso folgt aus der Tatsache, daß Person A Person B vertraut, nicht, daß die von B entwickelten Agenten nicht maliziös sind. Bestandteil für das Vertrauen in eine Software ist der Entwicklungsprozeß, d. h.: Wie wurde die Software hergestellt? Welche Design-, Entwicklungs- und Test-Verfahren wurden angewandt? Wesentlich für das Vertrauen in eine Software sind schließlich die Eigenschaften des Produktes [DFS98]. Bislang stehen

dem Empfänger eines Agenten keine Informationsquellen bereit, mit denen er Vertrauen in einen Agenten erhalten kann. Die Signatursysteme der Basistechnologien aus Kapitel 5 ziehen aus dem Umstand, daß einer Person vertraut wird den Schluß, daß auch der Software, die von der Person entwickelt wurde, vertraut werden kann.

Es bleibt zu klären, welche Merkmale ein Benutzer bzw. System heranziehen kann, anhand derer nachweislich bestimmt werden kann, ob ein signierter Agent vertrauenswürdig ist oder nicht.

FELTEN hat in einem Artikel im Rahmen des RISK-FORUMS deutlich gemacht, daß mit digitalen Signaturen eine Reihe von Erwartungen verknüpft sind, die von diesen nicht erfüllt werden:

- ▷ Digitale Signaturen geben keine Auskunft darüber, wer ein Programm geschrieben hat. Eine digitale Signatur kann von einem Objekt entfernt und durch eine andere ersetzt werden. Eine Signatur gibt im besten Falle darüber Auskunft, daß der Unterzeichner das Programm kürzlich bestätigt hat.

- ▷ Ist ein Programm von einer vertrauenswürdigen Person unterschrieben, ist hieraus nicht der Schluß zu ziehen, daß es sicher ist, daß entsprechende Programm auszuführen. Neben der Möglichkeit, daß die Schlüssel des Unterzeichner kompromittiert wurden, besteht die Gefahr, daß das Programm in einem Dokument eingebettet ist, daß das gutartige Programm zu maliziösen Aktionen veranlaßt.[2]

- ▷ Digitale Signaturen schaffen keine Verantwortlichkeiten. Es ist bestenfalls nur mit großem Aufwand möglich, beweisbar darzulegen, daß eine maliziöse Aktion durch ein bestimmtes Programm ausgelöst worden ist [Fel97].

Den Ausführungen von FELTEN läßt sich anfügen, daß die Mehrheit der Anwendungssysteme nicht geeignet ist, sensitive Informationen wie private Schlüssel aufzubewahren. In Hinblick auf die weitverbreitete Nutzung von PCs und Apple-Rechnern ist davon auszugehen, daß es Dritten möglich ist, illegalen Zugriff auf private Schlüssel zu erhalten. VeriSign, als eine CA, sagt hierzu in ihren Zertifizierungsbestimmungen: „[...] jede Person sollte ihr Schlüsselpaar auf einem vertrauenswürdigen System generieren und die notwendigen Maßnahmen treffen, um dessen Kompromittierung, Enthüllung, Modifizierung oder unautorisierte Nutzung zu verhindern. [GS97]“ Es ist offensichtlich, daß diese Bestimmungen von den oben angeführten Anwendungssystemen nicht erfüllt werden kann und darüberhinaus die Anwender nicht über das notwendige Wissen verfügen, um die geforderte Geheimhaltung sicherzustellen.

Beim Einsatz digitaler Signatursysteme ist deshalb darauf zu achten, daß durch eine Signatur lediglich zum Ausdruck gebracht wird, daß eine Entität zum Zeitpunkt des Signierens Zugriff auf den entsprechenden privaten Schlüssel hatte. Die Semantik einer digitalen Signatur stellt keine 1:1-Beziehung zwischen einer Person und einem privaten

[2] Siehe hierzu auch Abschnitt 5.2.1.

Schlüssel her [GS97]. Die involvierte Hardware in Kombination mit den verwendeten Betriebssystemen bilden i. d. R. unsichere Systeme.

Abschließend sei noch auf das Problem zurückgezogener Zertifikate verwiesen. Die Kompromittierung eines privaten Schlüssels oder aber auch der Einsatz dessen zur Signierung maliziöser Software veranlaßt die herausgebende CA zur Rücknahme des korrespondierenden Zertifikats. Die Veröffentlichung zurückgezogener öffentlicher Schlüssel kann über Certificate Revocation List (CRL) erfolgen. Jede CA veröffentlicht in regelmäßigen Abständen eine Liste mit zurückgezogenen Schlüsseln. Problem dieser Listen ist ihre Größe und das Problem der Lücke zwischen der Kompromittierung eines Schlüssels und der Rücknahme eines Zertifikats. In jedem Fall muß der Nutzer einer PKI sich über zurückgezogene Zertifikate informieren. Das vorherige Kapitel hat anhand des Internet Explorers gezeigt, daß die genutzten Basistechnologien dies in den Standardeinstellungen nicht unbedingt unterstützen. Stattdessen kann jede CA eine Online-Datenbank bereithalten, die bei jeder Zertifikatsüberprüfung befragt wird.

6.2. Fault-tolerance

6.2.1. Konzept

SCHNEIDER hat ein Verfahren entwickelt, daß den Schutz der Agenten vor maliziösen Umgebungen gewährleisten soll. Kern des Ansatzes ist die Tolerierung maliziöser Umgebungen[3] durch Replikationen und Abstimmungen. SCHNEIDER setzt voraus, daß die Agenten sich auf jedem System deterministisch verhalten und von jedem System, mit Ausnahme des Quell- und Zielsystems, repliziert werden können.

Wie der Abbildung 6.1 zu entnehmen ist, versendet das Ausgangssystem Q eines Agenten eine Anzahl von replizierten Agenten, die jeweils eine andere Umgebung desgleichen Anbieters aufsuchen. Auf den Systemen des Anbieters wird in jeder Umgebung ein Agent ausgeführt und nach Beendigung des Auftrages an das System des nächsten Anbieters versandt. Dazu werden entsprechend der Anzahl der Umgebungen des nächsten Anbieters viele Replikationen des Agenten hergestellt und jeweils eine Replikation wird an eine Umgebung des Anbieters verschickt.

Das nachfolgende System erhält als Ergebnis der Berechnung auf dem vorhergehenden System eine Menge von Agenten zusammen mit deren Zustand. Aus der Menge der Agenten wird per Abstimmung bestimmt, welcher Agent für die weitere Berechnung verwendet wird. Jede Umgebung der Stufe i übernimmt das Ergebnis, das ihm von der Mehrheit der Umgebungen der Stufe $i-1$ gesendet wurde [Sch97].

Zu diesem Zeitpunkt verhilft das Verfahren lediglich zu einer erhöhten Fehlertoleranz, es bietet jedoch keinen Schutz vor maliziösen Umgebungen. Der Schutz vor maliziösen Umgebungen bedarf der Erkennung manipulierter Agenten. Hierzu führen Agenten ein

[3] Umgebungen werden von SCHNEIDER als Prozessoren bezeichnet.

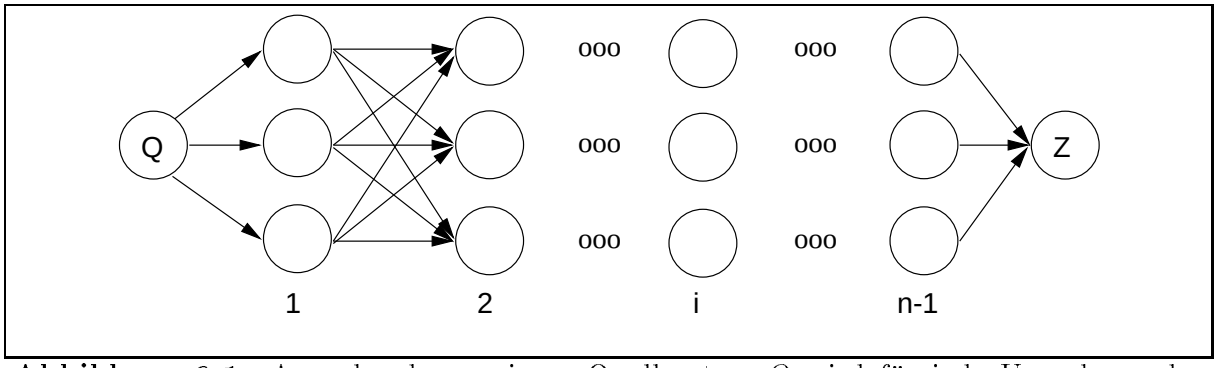

Abbildung 6.1.: Ausgehend von einem Quellsystem Q wird für jede Umgebung des Anbieters eine Replikation des Agenten erzeugt und versandt. Nachfolgend versendet der Anbieter i n Replikationen eines Agenten nach dessen Ausführung an das nachfolgende System, daß über n Umgebungen verfügt. Die Anbieter bestimmen jeweils aus der Mehrzahl der Ergebnisse, jenes Ergebnis, das für die weitere Ausführung des Agenten verwendet wird.

„Privileg", ein Geheimnis mit sich, anhand dessen Manipulationen erkannt werden können. Solange das Privileg geheim bleibt, können keine falschen Agenten mit dem Privileg generiert werden. Damit sich manipulierte Agenten durchsetzen, müßten maliziöse Umgebungen in der Lage sein, Abstimmungen zu beeinflussen. Es bedarf somit eines Mechanismus, mit dessen Hilfe Abstimmungen authentifiziert werden können. SCHNEIDER stellt in seiner Arbeit hierzu zwei Verfahren vor, die nachfolgend beschrieben werden.

Geteiltes Geheimnis

Das Protokoll für die Wahrung des geheimen Privilegs kann auf dem aus der Kryptografie bekannten Verfahren des *geteilten Geheimnis* (engl. *shared secret*) basieren. Das Geheimnis ist in seiner Gesamtheit nur dem Quell- und Zielsystem bekannt. Es gilt während der Migration des Agenten zu verhindern, daß andere Umgebungen als das Quell- und Zielsystem, daß Geheimnis in Erfahrung bringen oder es zerstören.

Agenten können keine Geheimnisse mit sich führen, ohne daß diese auf der Route kompromittiert werden. Ein Geheimnis wird aus diesem Grund in ein geteiltes Geheimnis überführt, daß aus n Fragmenten besteht. Ein Angreifer benötigt k Fragmente, um das Geheimnis aufzudecken. In einem Agentensystem, in dem jeder Anbieter über maximal $2k-1$ Umgebungen verfügt, wird ein $(2k-1, k)$ Schema verwendet. In diesem Schema besteht das geteilte Geheimnis aus $2k-1$ Fragmenten und zur Aufdeckung benötigt ein Angreifer k Fragmente.

Das Protokoll birgt allerdings das Problem in sich, daß maliziöse Umgebungen unterschiedlicher Stationen verschiedene Untermenge der Fragmente stehlen können. Gelangen die maliziösen Umgebungen zusammen in den Besitz einer Mehrheit der Fragmente, können sie das Geheimnis rekonstruieren. Sichere Erweiterungen des Protokolls haben zur Folge, daß das Verfahren ineffizient wird [Sch97].

Authentikationsketten

Das zweite Protokoll zur Wahrung des Privilegs basiert auf Authentikationsketten. Alle Umgebungen kennen die Identität des Quellsystems und der Agent führt ein unverfälschtes Zertifikat mit sich, daß seinen gesamten Migrationsweg beinhaltet. Das Protokoll nimmt an, daß die Mehrzahl der Umgebungen einer Stufe i nicht maliziös sind.

Ein Wähler q in Station $i+1$ kann falsche Agenten der Station i erkennen, wenn die Wählerschaft P_i der Station i bekannt ist und jeder Sender der Nachrichten authentifiziert werden kann, die er erhält.

Der Wähler nutzt sein Wissen um die Wählerschaft P_i, um jeden Agenten abzulehnen, der von einer Umgebung stammt, der nicht Mitglied von P_i ist. Von den restlichen Agenten wählt der Wähler einen derer aus, für die er $\lceil |P_i|/2 \rceil$ äquivalenter Replikationen erhalten hat. Die Wählerschaft P_i der Station i ermittelt eine Umgebung q der Station P_{i+1} aus sog. *Forwards*, die jedem Agent zugefügt werden, wenn er eine Station verläßt. Die Umgebungen der Station $i-1$ kennen die Wählerschaft P_i, da sie an all deren Mitglieder Replikationen der Agenten schicken. Der Wähler kann somit aus der Migrationsroute des Agenten ermitteln, welche Umgebungen Mitglied der Wählerschaft P_i sind und damit falsche Agenten ausschließen. Agenten, die zur nächsten Station versandt werden, wird ein Forward zugefügt und der Agent wird digital signiert. Das Protokoll macht sich den Umstand zunutze, daß die Umgebungen der Station $i-1$ die Mitglieder der Wählerschaft P_i kennen.

6.2.2. Beurteilung

Das von SCHNEIDER entwickelte Verfahren versucht das neue, durch das Agenten-Paradigma entstandene Sicherheitsproblem – dem Schutz des Agenten vor der Umgebung – zu lösen. Die anderen Sicherheitsdimensionen werden in seiner Sicherheitsarchitektur nicht in Betracht gezogen. Damit handelt es sich, wie in auch bei allen anderen nachfolgenden Sicherheitsarchitekturen, um eine Teillösung, die ausschließlich bestimmte Bereiche der Sicherheitsanforderungen abdeckt.

Der generelle Lösungsansatz von SCHNEIDER löst daß Problem der maliziösen Umgebungen nicht, da in der Lösung davon ausgegangen wird, daß die beteiligten Server voneinander unabhängig sind, d. h., es findet keine Kooperation zwischen maliziösen Servern eines Anbieters statt. Es ist unrealistisch davon auszugehen, daß die Umgebungen, die von einem Anbieter bereitgestellt werden, unabhängig voneinander sind. So steht zum einen dem Anbieter die Möglichkeit offen, alle Umgebungen maliziös auszulegen, so daß Agenten, die auf die Umgebungen migrieren, manipuliert werden. Zum anderen ist davon auszugehen, daß Sicherheitslücken, die in einer der Umgebungen entdeckt werden, auch in allen anderen existieren. Damit haben externe Angreifer die Möglichkeit, die Sicherheitslücken auszunutzen, um so Einfluß auf die Agenten zu nehmen. Die Annahme schränkt das Konzept auf eine Weise ein, die es nicht für Agentensysteme in offenen Umgebungen geeignet erscheinen läßt.

Fraglich ist für die von SCHNEIDER entwickelte Lösung weiterhin, ob der Einsatz von Replikation in Agentensystemen verwirklichbar ist, ohne daß die Vorteile, die mit der Agententechnologie erzielt werden sollen, verloren gehen. Im Falle, daß jedes System entlang der Route des Agenten aus n Umgebungen besteht, werden für das nächste System jeweils n^2 replizierte Agenten erstellt. Die damit verbundene Netzlast kehrt damit einen der wesentlichen Vorteile, die Verminderung der Last, ins Gegenteil.

Desweiteren sind replizierte Umgebungen nur auf Seiten großer Anbieter zu finden. Kleinere und mittlere Anbieter können ihre Dienste i. d. R. nur über eine Umgebung anbieten. Selbst wenn die Mehrzahl der Anbieter eine Reihe replizierter Server zur Verfügung hat, ist nicht davon auszugehen, daß der Sender eines Agenten die Möglichkeit hat, vorher zu bestimmen, auf welchem System der Agent zur Ausführung gelangt.

Neben den bereits erwähnten Mängeln der Architektur kann SCHNEIDER auch nicht die Privatheit des Codes und der Daten des Agenten sicherstellen. Ein umfassender Schutz des Agenten vor der Umgebung kann somit nicht gewährleistet werden.

6.3. Firewalls

Eine Firewall ist eine Komponente oder eine Menge von Komponenten, die der Zugang zu einem Netzwerk von einem anderen Netzwerk beschränkt [CZ95]. Sie besteht aus einer Anordnung von Hardware- und Software-Komponenten, die den alleinigen Übergang zwischen zwei Netzen darstellt. Firewalls erlauben die Konzentration der Sicherheitsmaßnahmen an einem Punkt, der Verbindung des internen Netzes mit dem externen Netz. Dabei setzt eine Firewall die Sicherheitspolitik eines Unternehmens gegenüber dem externen Netz durch. Nur genehmigte Dienste können die Firewall passieren und dieses auch nur entsprechend der definierten Regeln.

Firewalls sind ein bereits länger eingeführter Sicherheitsmechanismus. Sie wurden entsprechend nicht in Hinblick auf die spezifischen Sicherheitsprobleme entwickelt, die aus dem Einsatz von Agentensystemen resultieren. Organisationen und Unternehmen, die mit einem offenem Netz, wie beispielsweise dem Internet, verbunden sind, haben i. d. R. eine Firewall installiert und nutzen diese als wesentliche Komponente zum Schutz vor Angriffen. Es soll deshalb an dieser Stelle untersucht werden, welchen Mehrwert Firewalls im Agentenumfeld beinhalten können.

Firewalls lassen sich zunächst in zwei Kategorien unterscheiden, die nachfolgend beschrieben werden: Paketfilter und Applikation-Gateway.

6.3.1. Paketfilter

Paketfiltersysteme delegieren Pakete zwischen internen und externen Rechnern. Sie erlauben die Blockade von bestimmten Paketen in Reflektion auf die Sicherheitspolitik des Systems [CZ95]. Kriterien für die Beurteilung von Nachrichtenpaketen sind die Quell- und Zieladresse, das Protokoll und der Port auf der Quell- und Zielseite. Ein Paketfil-

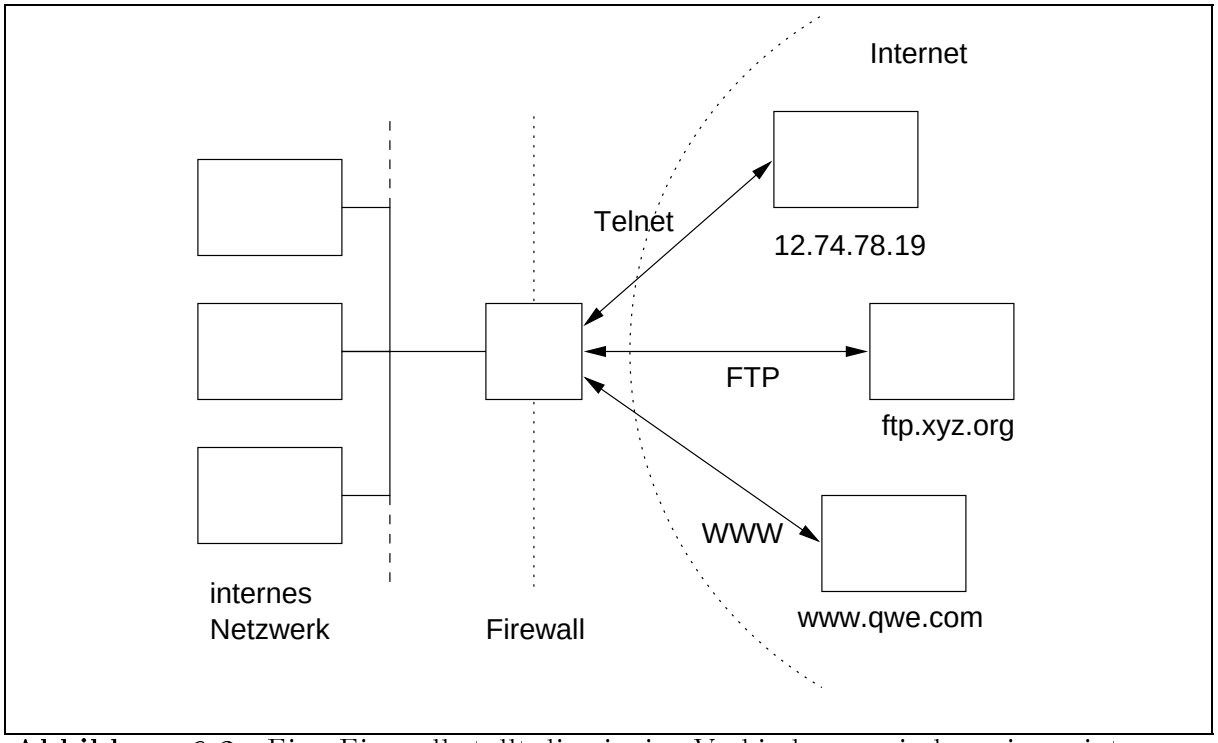

Abbildung 6.2.: Eine Firewall stellt die einzige Verbindung zwischen einem internem und einem externem Netzwerk dar. Sie ermöglicht eine Filterung auf Basis von Paketen der Verbindungsschicht und/oder auf Anwendungen der Anwendungsschicht.

ter überprüft nicht nur, ob er ein Paket weiterleiten *kann*, sondern stellt darüberhinaus anhand der definierten Regeln fest, ob das Paket weitergeleitet werden *soll* [CZ95]. Die Entscheidung über die Weiterleitung von Nachrichtenpaketen wird unabhängig für jedes einzelne Paket getroffen.

Für Pakete, die durch den Paketfilter abgelehnt werden, werden Informationen aufgezeichnet, denen zu entnehmen ist, aufgrund welcher Regeln das Paket abgelehnt wurde. Im Falle abgelehnter Pakete kann das Quellsystem über Fehlermeldungen informiert werden, daß die Verbindung mit dem Zielsystem nicht funktioniert hat. Zu beachten ist, daß der Versand von Fehlermeldungen semantisch nicht korrekt ist, einen kleinen Teil der Bandbreite benötigt und desweiteren dem Quellsystem möglicherweise Informationen über den Aufbau der Filtertechniken des Zielsystems liefert.

Paketfilter können nicht auf Identitäts- oder Objektbasis filtern. Das Verwehren eines Dienstes für bestimmte Nutzer ist nicht möglich, da ein Nutzer nicht durch einen Paketfilter identifiziert werden kann. Gleiches gilt für Objekte wie beispielsweise Dateien.

6.3.2. Applikation-Gateway

Eine Applikations-Gateway[4] arbeitet vollständig auf der Anwendungsebene. „Auf dem Gateway laufen Proxies als spezialisierte Programme ab, die in der Lage sind, Verbindungen für ein spezielles Protokoll entgegenzunehmen, die übertragenen Daten auf Applikationsebene zu verarbeiten und dann weiterzuleiten [Str97]." Für die Filterung werden Informationen genutzt, die in der Anwendungsschicht vorhanden sind. Proxies kennen die Semantik der Protokolle. Proxies sind jedoch nicht in der Lage, Systeme vor Daten zu schützen, da sie diese verstehen und entscheiden müßten, ob die Daten gefährlich sind.

Proxies gewährleisten die vollständige Trennung von Verbindungen zwischen dem internen und dem externen Netz. Direkte Verbindungen zwischen den beiden Netzen werden durch den Einsatz eines Applikation-Gateways unterbunden. Applikation-Gateways kennen die Semantik der Verbindung und können deshalb auf Applikationsebene eingreifen und Veränderungen vornehmen.

6.3.3. Beurteilung

Die Sicherheitsanforderungen, die zu Beginn des Kapitels definiert werden, können von Firewalls nicht adressiert werden. Firewalls können ausschließlich zum Schutz einer Umgebung vor Agenten eingesetzt werden. Trotzdem werden Firewalltechniken in einer Reihe von kommerziellen Produkten eingesetzt, die Netzwerke vor maliziösen Agenten schützen sollen.

Chapman und Zwicky führen in [CZ95] aus, daß eine Firewall nicht zum Schutz vor Viren dienen kann. Die Entdeckung eines Virus in einem zufälligen Nachrichtenpaket verlangt, daß die Firewall erkennt, daß ein Nachrichtenpaket Teil eines Programmes ist. Applikations-Gateways sind ebenso nicht in der Lage, Viren aus einem Netzwerk fernzuhalten, da diese nicht an eine spezifische Applikation gebunden sind. Agenten sind keine Viren, sie können jedoch aus den gleichen Gründen nicht durch eine Firewall vollständig blockiert werden.

Eine Firewall, die in die Lage versetzt werden soll, jegliche Agenten zu filtern, müßte in der Lage sein, Agenten zu erkennen. Dazu gehört, daß die Firewall beispielsweise HTML-Dokumente entsprechend den eingesetzten Browsern interpretiert werden. Bereits an dieser Stelle gilt es für Firewalls ein wesentliches Problem zu bewältigen, denn jeder Browser interpretiert ein Dokument auf unterschiedliche Weise. Die Mehrzahl der Browser ist in der Lage, ein HTML-Dokument auch dann zu interpretieren, wenn dieses syntaktisch nicht korrekt ist. Es wäre Aufgabe einer Firewall, daß Verhalten eines jeden Browsers nachzubilden [MRR97]. Agenten können über eine Vielzahl von Protokollen in ein Intranet bzw. auf einen Arbeitsplatzrechner gelangen. Daraus erwächst für Firewalls die Anforderung, daß Agenten aus all jenen Protokollen (HTTP, FTP, NNTP, Email, etc.) herausgefiltert werden.

[4] Wird häufig auch als Application-Proxy bezeichnet.

Ein entsprechender Filtermechanismus ist äußerst komplex, wenn nicht gar unmöglich [Hop97a]. Angreifer können innerhalb eines durch eine Firewall geschützten Netzes Opfer identifizieren und diesen elektronische Nachrichten, die in einem MIME-Objekt mit der Kennzeichnung `Content-type: text/html` verpackt sind, schicken. Ist der Client des Anwenders für elektronische Nachrichten in der Lage HTML-Dokumente zu interpretieren, kann ein im Dokument referenzierter Agent so zur Ausführung gebracht werden [MRR97]. Desweiteren ist eine genaue Betrachtung der Entwicklung neuer Technologien bzw. Anpassungen von existierenden Technologien erforderlich, um sicherzustellen, daß ausgehend von einem sicheren Ist-Zustand auch für zukünftige Zeitpunkte zu gewährleisten ist, daß Agenten aus dem Datenstrom herausgefiltert werden.

Die Blockade aller Agenten anhand spezifischer Signaturen, ist ebenfalls nur schwerlich möglich. Es gibt eine Vielzahl von Kodierungsformen, die es erlauben, beispielsweise Java-Agenten (Aglets, Applets, Servlets, etc.) zu übertragen, ohne daß die Signatur von Java-Klassen an der Firewall in Erscheinung tritt. Hierfür können Agenten sich in Archiven (JAR, Zip) befinden, uuencoded oder in einem MIME-Objekt kodiert sein. Signierte Java-Agenten sind nicht mehr anhand der Magic-Number zu erkennen. Signierte Java-Agenten werden mittels JAR-Dateien übertragen. JAR-Dateien genügen dem ZIP-Format und weisen entsprechend die gleiche Magic-Number wie „normale“ Zip-Dateien auf. Eine Blockade von Java-Agenten ist damit anhand der Magic-Number nicht mehr möglich. Denkbar sind darüberhinaus Agenten, die in beliebigen anderen Formaten eingebettet sind.

Weiterhin kann die Identifizierung der Signatur auf IP-Ebene mit Problemen versehen sein. Die komplette Signatur kann auf verschiedene Pakete aufgeteilt sein, so daß mehrere Pakete untersucht werden müßten.

Mit der Ablösung des Internet-Protokolls IPv4 durch IPv6 tritt für Paketfilter das Problem auf, daß IP-Pakete nicht mehr im Klartext, sondern in verschlüsselter Form des ESP vorliegen. Der Paketfilter ist dann nicht mehr in der Lage, entsprechende Pakete auszuwerten, wenn er nicht über die korrespondierenden Schlüssel verfügt. Eine Lösung in diesem Zusammenhang stellt der Einsatz von Sicherheitsgateways dar, wie sie von IPv6 unterstützt werden. In dem Fall hat die Firewall zusätzlich die Rolle des Sicherheitsgateways und übernimmt damit die Aufgabe, alle Pakete zu ver- und entschlüsseln.

Die Anlage eines Applikation-Gateways ist mit großen Aufwendungen verbunden, wenn es möglich sein soll, daß unterschiedliche Agentensysteme angeboten bzw. weitere Dienstleistungen durch die Firewall geschützt werden. Für jede Dienstleistung muß ein entsprechend spezialisierter Proxy existieren, dem die Semantik der Verbindung bekannt ist. Desweiteren kommen hohe Belastungen auf die Rechenleistung zu, da für jede einzelne Verbindung ein Proxy-Prozeß gestartet werden muß.

Eine vollständige Blockade von Agenten ist häufig nicht wünschenswert. Eine Reihe von Anwendungen benötigen den Einsatz von Agenten, so daß ein Raum für diese Agenten geschaffen werden muß. Denkbar wäre eine Positiv-Liste, in der jene Agenten enthalten sind, die in das interne Netzwerk migrieren dürfen. Zu klären ist die Frage, wie die Identifikation der Agenten erfolgt. Angriffe in Form von Masquerading oder Spoofing

sind weiterhin zu verhindern.

Ein positiver Aspekt einer Firewall-Lösung ist der Schutz auf systemweiter Ebene. Die Vielzahl der Sicherheitsarchitekturen agieren auf Arbeitsplatzebene, so daß eine Vielzahl von Systemen korrekt konfiguriert und geschützt werden müssen.

In einer vom Bundesamt für Sicherheit in der Informationstechnologie (BSI) in Auftrag gegebenen Studie wird darauf hingewiesen, daß mittlerweile „zwei Drittel aller Bedrohungen eines Intranets aus dem Netzwerk selber kommen [Neu98]. Bedrohungen innerhalb eines Intranets können nicht mit einer Firewall begegnet werden.

6.4. Kerberos

Kerberos ist eine weitere Sicherheitsarchitektur, die bereits lange vor dem Aufkommen der ersten Agentensysteme entwickelt worden ist. Kerberos könnte in geschlossenen Agentensystemen sicherstellen, daß lediglich legitimierte Nutzer Dienste des Systems in Anspruch nehmen können.

6.4.1. Konzept

Kerberos wurde 1983 am MIT im Rahmen eines Projektes namens Athena entwickelt. Kerberos ist ein System zur gemeinsamen Nutzung geheimer Schlüssel von Entitäten über ein Netzwerk. Es erlaubt den Diensten ihre Nutzer eindeutig zu identifizieren.

Die Authentikation des Anwenders erfolgt durch dessen Anmeldung in das System.[5] Hierzu gibt er eine Identität und ein Paßwort an. Daraus wird eine Nachricht generiert, die an den Kerberos-Server gesandt wird. Die Nachricht besteht aus der Identität des Anwenders und der aktuellen Uhrzeit, die mit dem Paßwort des Anwenders verschlüsselt wird. Über die Identität des Anwenders ermittelt der Server das Paßwort und entschlüsselt die Uhrzeit. Im Erfolgsfall erhält der Client vom Kerberos-Server ein *Ticket Granting Ticket*, daß erneut mit dem Paßwort des Client verschlüsselt wird. Alle Tickets innerhalb des Systems werden stets in verschlüsselter Form übertragen.

Lediglich die initiale Authentifikation erfordert die Angabe des Client-Paßwortes. In den nachfolgenden Kommunikationsphasen werden ausschließlich Sitzungsschlüssel genutzt.

Das Ticket Granting Ticket enthält einen Sitzungschlüssel K_{ses} und ein Ticket für den *Ticket Granting Service*, das mit dem Sitzungs- und dem Schlüssel des Dienstes verschlüsselt ist. Für die Inanspruchnahme eines Dienstes im Netzwerk wird nachfolgend stets der Ticket Granting Service kontaktiert, der dem legitimierten Nutzer ein Ticket für die Nutzung des angegebenen Dienstes übersendet.

Der Dienst des Ticket Granting Servers verringert die Gefahr, daß der Schlüssel eines Clients einem Angreifer bekannt wird. Gleichzeitig wird die Nutzung von Kerberos transparenter.

[5] Beschrieben wird das Verfahren, das mit Kerberos 5 angewendet wird.

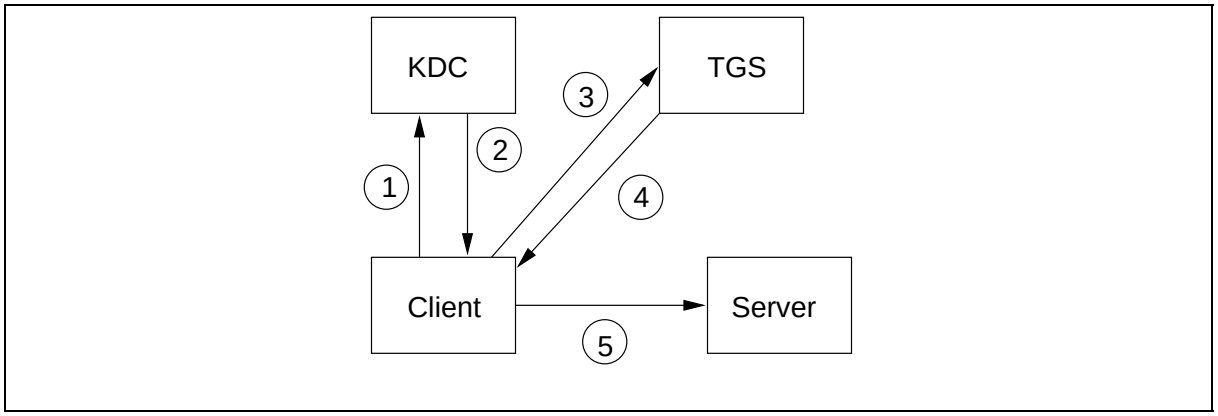

Abbildung 6.3.: 1. Der Client authentisiert sich gegenüber dem Kerberos-Server. 2. Der Client erhält vom Server einen Session-Key zur Kommunikation mit dem Ticket Granting Service (TGS). 3. Der Client schickt eine verschlüsselte Nachricht an den TGS. 4. Die Antwortnachricht enthält ein Ticket zur Kommunikation mit einem Dienst. 5. Dem Dienst wird das Ticket übersandt (nach [KNT91]).

Der Ticket Granting Service ist logisch vom Kerberos-Server separiert, befindet sich jedoch meist auf dem gleichen Rechner und hat Zugriff auf die Datenbank der Clients sowie auf die Schlüssel des Kerberos-Servers [KNT91].

Die gemeinsamen Schlüssel des Kerberos-Servers und seiner Clients sind die Basis dafür, daß die Dienste, die durch die Clients in Anspruch genommen werden, in die Authentizität der Tickets vertrauen. Der Kerberos Server speichert die Paßwörter innerhalb des Netzwerks genutzter Paßwörter in verschlüsselter Form. Der Kerberos-Server wurde zustandslos entwickelt; er beantwortet lediglich Anfragen der Nutzer und gibt Tickets heraus. Damit ist eine Replikation des Servers sehr einfach möglich.

Kerberos ist ein bidirektionales Authentikationsverfahren, daß gewährleistet, daß stets sowohl der Client als auch der Server sich authentifizieren.

Kerberos 5 bietet über den Data Encryption Standard (DES) hinaus die Möglichkeit, auch andere Verschlüsselungsverfahren einzusetzen. Bislang gibt es allerdings keine Implementationen, die dieses ausnutzen [GS97].

6.4.2. Beurteilung

Wie bereits einleitend angeführt, kann Kerberos der Authentikation in geschlossenen Systemen dienen. Die Bereitstellung von Paßwörtern für Verbindungen wird weiterhin genutzt, um die Vertraulichkeit der zu übertragenden Daten zu gewährleisten. Kerberos kann damit lediglich eine Komponente in der Sicherheitsarchitektur eines Agentensystems sein. Es bildet für sich keine Sicherheitsarchitektur, die die aus dem Agenten-Paradigma resultierenden Sicherheitsprobleme löst.

So Kerberos als eine Komponente in der Sicherheitsarchitektur eines Agentensystemes eingesetzt wird, ist zu beachten, daß der Dienst eine Reihe von Mängeln aufweist.

Zentrale Komponenten in verteilten Systemen sind stets ein Problem, da ein Ausfall einer derartigen Komponente, die Verfügbarkeit aller Dienste betrifft, die von dem Dienst der Komponente abhängig sind. Wie bereits dargelegt, sind Kerberos-Server zustandslos ausgelegt, so daß sie einfach zu replizieren sind. Das damit umgangene Problem der zentralen Komponente ist allerdings lediglich eine Problemverlagerung. Jetzt ist der Betreiber der Server mit dem Problem konfrontiert, daß alle replizierten Server physikalisch geschützt werden müssen.

Der Schutz der Teilnehmerpaßwörter erfolgt mit einem Schlüssel. Gelangt ein Angreifer in den Besitz des Schlüssels, hat er Zugriff auf alle Schlüssel der Teilnehmer, die auf dem Server abgelegt sind. Kerberos unterstützt ausschließlich symmetrische Verschlüsselungsverfahren. In Hinblick auf die vielfältigen Einsatzmöglichkeiten von asymmetrischen Verschlüsselungsverfahren, beispielsweise den oben vorgestellten digitalen Signaturen,[6] stellt dieses einen Mangel für den Einsatz von Kerberos in Agentensystemen dar.

Die Systemsoftware, die zum Betrieb von Kerberos erforderlich ist, wird durch den Dienst selbst nicht vor Trojanern oder Viren geschützt. Hier wird bereits deutlich, daß Kerberos lediglich einen spezifischen Dienst erbringt, der die Unterstützung weiterer Dienste der Sicherheitsarchitektur erfordert.

Weitere Mängel, die Bellovin 1991 in Kerberos 4 identifiziert hat, sind mittlerweile in der hier vorgestellten Version 5 von Kerberos behoben worden [BM91].

6.5. Onion Routing

6.5.1. Konzept

Verschlüsselung gewährleistet ausschließlich die Integrität und Vertraulichkeit der Kommunikation über ein Netzwerk. Ein Schutz vor Verkehrsflußanalysen bieten kryptografisch geschützte Verbindungen, wie beispielsweise eine mittels SSL geschützte Verbindung über das Internet, nicht. Die Informationen im unverschlüsselten Kopf der Pakete enthalten die erforderlichen Daten, mit denen Angreifer bestimmen können, welche Entitäten miteinander kommunizieren. Eine Vertraulichkeit des Verkehrsflusses ist nicht gegeben.

Die Sicherstellung der Vertraulichkeit des Verkehrsflusses ist ein lang ignoriertes Problem außerhalb des militärischen Bereiches. Die zunehmende Kommerzialisierung des Internets und die damit einhergehende Erstellung von Benutzerprofilen zeigt deutlich, daß ein Schutz vor Verkehrsflußanalysen immer drängender wird. Anonyme Zahlungsprotokolle können nur dann wirklich anonym sein, wenn die Kanäle, über die die Transaktionen erfolgen, ebenfalls die Anonymität gewährleisten. Unternehmen geben ihre Interessen preis, wenn beobachtbar ist, welche Dokumente sie regelmäßig aufrufen [SRG97].

Diesem Mißstand soll mit Onion Routern begegnet werden, die den Aufbau anonymer Verbindungen erlauben. Onion Routing ist ein in Hinblick auf das WWW entwickelter Dienst, der ein anonymes Browsen im Netz erlaubt. Der Dienst des Onion Routings

[6] Digitale Signaturen können auch mit symmetrischen Verschlüsselungsverfahren erstellt werden.

wurde applikationsunabhängig entwickelt, so daß ein Einsatz auch in Agentensystemen möglich sein sollte.

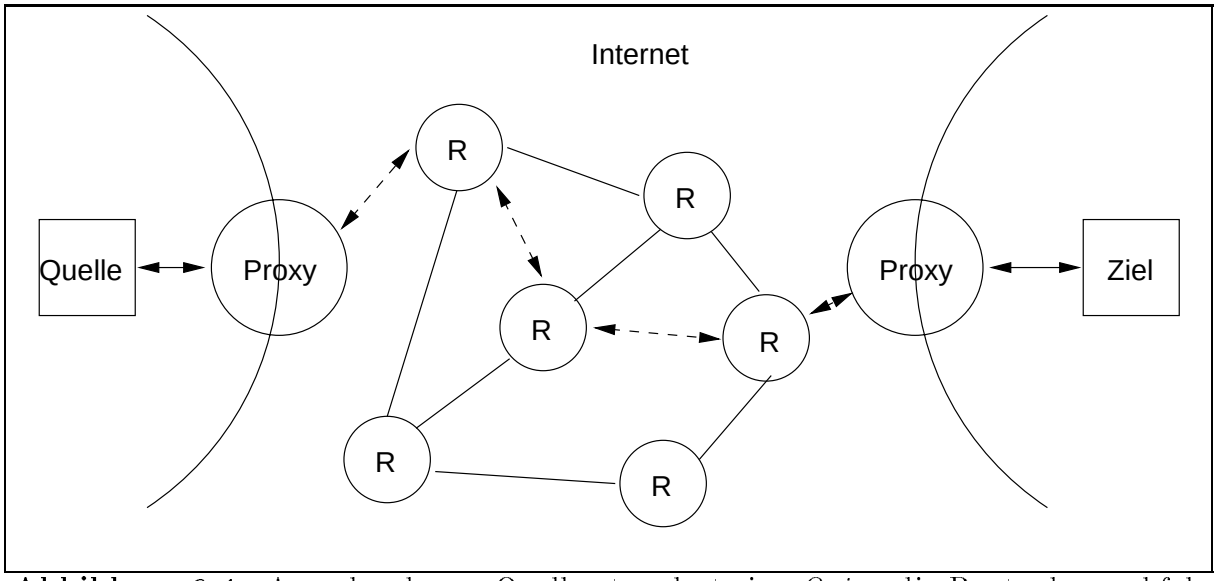

Abbildung 6.4.: Ausgehend vom Quellsystem legt eine *Onion* die Route der nachfolgenden Pakete durch das Netz der Router fest. Jeder Router entfernt von einem Nachrichtenpaket „seine" Verschlüsselung, so daß die Nachricht das Ziel schließlich in Klartext erreicht (nach [SRG97])

Eine Verbindung, die zu einem Zielsystem[7] aufgebaut wird, verläuft nicht direkt zu diesem, sondern durchläuft eine Reihe von Onion Routern. Onion Router sind bidirektionale Mixe, die in nahezu Echtzeit arbeiten. Zwischen den Routern bestehen langbestän-dige Verbindungen, über die die anonymen Verbindungen gemultiplext werden. Für jede anonyme Verbindung ist die Reihenfolge der zu durchlaufenden Onion Router strikt definiert. Jeder Onion Router kennt jeweils nur seinen Vorgänger und seinen Nachfolger in der Kette der Stationen, die eine Nachricht auf einer anonymen Verbindung durchläuft.

Der Zugang zu einem Onion Router-Netzwerk erfolgt über einen applikationsspezifischen Proxy, der die Reihenfolge der Onion Router für die Nachricht festlegt. Der Proxy konstruiert eine geschichtete Datenstruktur, die als *Onion* bezeichnet wird. Der Aufbau der anonymen Verbindung erfolgt durch den Versand der Onion in das Router-Netzwerk.

Jede Schicht der Onion ist mit dem öffentlichen Schlüssel eines Onion Routers verschlüsselt. Desweiteren sind alle Schichten einer Onion mit einem Zeitstempel versehen, der die Gültigkeitsdauer festlegt. Nach Überschreiten des Zeitstempels wird die Onion abgelehnt.

Der Onion Proxy übergibt die von ihm erzeugte Onion, den „Eingangstrichter" (engl. *entry funnel*), dem ersten Onion Router in der Kette. Der Eingangstrichter multiplext mehrere Verbindungen, die er vom Onion Proxy erhält, über eine Verbindung. Aufgabe

[7] In der Terminologie von SYVERSON et al. als *Responder* bezeichnet.

des Ausgangstrichters ist es, die gemultiplexten Verbindungen wieder in einzelne aufzulösen. Die Trichter agieren applikationsunabhängig und verstehen das Onion-Routing-Protokoll, das definiert, wie mit gemultiplexten Verbindungen verfahren wird [RSG98].

Neben den Informationen über das nächste Ziel enthält jede Schicht der Onion einen „Keim“ (engl. *seed*), mit dem Schlüssel zur Kodierung der zu übertragenden Daten erzeugt werden können. Jede Nachricht wird mit den Schlüsseln aller Router kodiert. Beim Durchlauf der Kette entfernt jeder Route „seine“ Verschlüsselung, so daß die Nachricht abschließend im Klartext beim Empfänger ankommt.

Empfängt ein Router eine Onion, entfernt er seine Schicht und erfährt anhand der nachfolgenden Schicht, welcher Router das nächste Ziel ist. Die Datenstruktur Onion besitzt eine feste Größe, die nach dem Entfernen einer Schicht mittels Padding vom aktuellen Router aufgefüllt wird. Der letzte Router in der Kette versendet die Datenstruktur an den Proxy des Empfängers. Der Proxy leitet die Onion schließlich an den Empfänger weiter. Erreicht die Onion das Ziel der Verbindung, ist eine anonyme Verbindung etabliert.

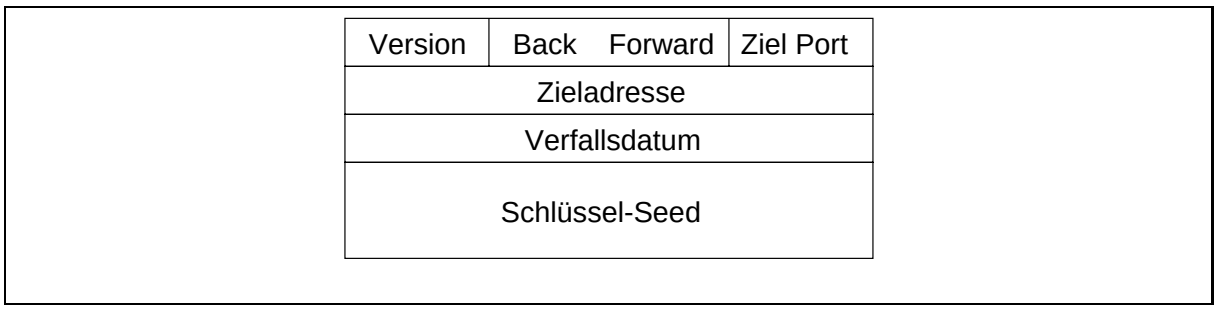

Abbildung 6.5.: Eine Onion setzt sich aus den obigen Feldern zusammen. Die Felder Forward und Backward verweisen jeweils auf die kryptografischen Funktionen die in der entsprechenden Richtung auf die Schicht anzuwenden sind. Das Feld Schlüssel-Seed enthält die notwendigen Informationen, mit denen jeder Router „seinen“ Schlüssel generieren kann (nach [RSG98]).

Die Verschlüsselung jeder Nachricht für jeden einzelnen Router hat zur Folge, daß sich eine Nachricht in jedem einzelnen Onion Router anders darstellt. Damit ist gewährleistet, daß eine Nachricht nicht zurückverfolgt werden kann.

Neben der Verhinderung der Verkehrsflußanalyse[8] können mit Onion Routing Verbindungen etabliert werden, in denen auch dem Empfänger der Nachricht verborgen bleibt, vom wem die Nachricht stammt. Dazu entfernt der Sender alle ihn identifizierenden Informationen aus der Nachricht und kann sich so sicher sein, daß der Kommunikationspartner ihn nicht identifizieren kann [RSG98].

Onion Routing erfolgt auf Applikationsebene. Der Onion Proxy kann ausschließlich

[8] Es sei angemerkt, daß Angreifer, die den Verkehr in einem Netzwerk komplett beobachten können, immer noch durch zeitliche Parallelitäten erkennen können, daß zwei Parteien sehr wahrscheinlich miteinander kommunizieren.

festlegen, welche Router zu durchlaufen sind und in welcher Reihenfolge dieses erfolgen soll. Das physikalische Routing, d. h., den Weg, den die Pakete zwischen den Onion Routern einschlagen, erfolgt weiterhin durch ein Verbindungsprotokoll, wie z. B. IP. Die Onion Router treten jeweils als Empfänger und Sender der Nachrichtenpakete auf. Damit wird gewährleistet, daß der ursprüngliche Sender einer Nachricht auch auf Ebene des Verbindungsprotokolls nicht identifiziert werden kann.

Für das Aufsetzen anonymer Verbindungen sind verschiedene Konfigurationen möglich. So der Initiator einer Verbindung einen Onion Router kontrolliert, wird dieser auf dem Firewall-System installiert. Vertraut der Initiator einem externen Onion Router, kann zu diesem eine verschlüsselte Verbindung aufgebaut werden. Ebenso wie in der Firewall-Konfiguration können so anonyme Verbindungen aufgebaut werden.

6.5.2. Beurteilung

Onion Routing ist für Agentensysteme insbesondere deshalb von Interesse, da es den Sicherheitsaspekt der Anonymität adressiert. Keines der hier vorgestellten Sicherheitssysteme beinhaltet einen Mechanismus, mit dem Agenten vor Verkehrsflußanalysen geschützt werden oder ihre Identität vor der empfangenden Umgebung verbergen können.

Die Anonymität von Agenten ist ein deutlicher Hinweis darauf, daß die in Kapitel 2 angeführten Sicherheitskriterien nicht vollständig miteinander vereinbar sind. Der Betreiber einer Umgebung für Agenten, hat ein großes Interesse, die Aktionen, die durch Agenten veranlaßt werden, bestimmten Entitäten zuzuordnen. Neben den wirtschaftlichen Interessen ist dieses insbesondere aus Sicht der Sicherheit des ausführenden Systems erforderlich.

Aufgrund der US-Exportbeschränkungen für kryptografische Produkte existieren Implementationen für Online Routing derzeit ausschließlich in den **USA!** . An der Universität Hamburg erfolgt im Rahmen eine Diplomarbeit von MÖLLER die Implementierung eines Anonymisierungsverfahrens, daß auf das Onion Routing-Verfahren aufbaut [Möl99].

6.6. Mobile Kryptografie

Die von SANDER und TSCHUDIN konzipierte Sicherheitsarchitektur der mobilen Kryptografie stellt, so es sich verwirklichen läßt, einen effektiven Schutz der Agenten vor Bedrohungen durch die Umgebung sicher.

6.6.1. Konzept

Mit der mobilen Kryptografie glauben SANDER und TSCHUDIN ein Verfahren entwickelt zu haben, daß es erlaubt, die sichere Ausführung von Agenten von der Erforderlichkeit einer vertrauenswürdigen Umgebung zu entkoppeln [ST97a]. Damit wäre die Existenz

einer Trusted Computing Base (TCB) nicht länger keine zwingende Voraussetzung für sichere Agentensysteme.

Kernkomponente des Verfahrens ist die Ausführung von Agenten in verschlüsselter Form. Nach dem Empfang des Agenten durch das Wirtssystem ist keine Entschlüsselung des Agenten erforderlich. Damit bleibt dem Wirtssystem die Funktion des Agenten vollständig verborgen. Lediglich die vom Agenten erfragten Eingabewerte lassen Rückschlüsse auf die Funktionalität des Agenten zu. Die vom Agenten produzierten Zwischen- und Endergebnisse liegen ebenfalls in verschlüsselter Form vor. Daraus folgt, daß mobile Kryptografie neben der Einhaltung der Ausführungsprivatheit auch die Integrität der Ausführung gewährleisten kann. Eine Manipulation eines Agenten in verschlüsselter Form kann nicht zielgerichtet erfolgen, so eine Entschlüsselung des Agenten durch das Wirtssystem unmöglich ist.

SANDER und TSCHUDIN definieren mobile Kryptografie als „die Studie mathematischer Techniken in Bezug auf Aspekte der Informationssicherheit mobilen Codes in Netzwerken [ST97a]."

Das Problem der Ausführung einer verschlüsselten Funktion in nicht-interaktiver Form kann wie folgt beschrieben werden:

> „Alice verfügt über einen Algorithmus zur Berechnung der Funktion f. Bob hat eine Eingabe x und ist bereit $f(x)$ zu berechnen, wobei Alice nicht möchte, daß Bob etwas *Substantielles* über f lernt. Weiterhin soll Bob während der Berechnung von $f(x)$ nicht mit Alice interagieren müssen".

SANDER und TSCHUDIN haben hierfür ein Protokoll entwickelt, daß eine nicht-interaktive Evaluierung von verschlüsselten Polynomen über Ringe $\boldsymbol{Z/NZ}$ erlaubt. Das Protokoll erfordert als Interaktion lediglich den Austausch des Agenten und des Endergebnisses. Eine Ausdehnung des Verfahrens von algebraischen Schaltkreisen zu booleschen ist derzeit Gegenstand der Forschung. So eine Lösung für boolesche Schaltkreise existiert, ist damit eine Lösung für allgemeine Programme gegeben, da jede Turingmaschine durch boolesche Schaltkreise simuliert werden kann [ST97a].

Die Anforderung, alle Informationen über einen Booleschen Schaltkreis zu verbergen, ist allerdings zu stark. Erforderlich wäre die Verwendung eines universellen booleschen Schaltkreises. Die Komplexität eines universellen booleschen Schaltkreises wächst exponentiell mit der Größe der Eingabe (ein spezieller Boolescher Schaltkreis). Damit entzieht sich der universelle boolesche Schaltkreis der Berechenbarkeit. Für eine Mehrzahl der Anwendungen reicht es hingegen aus, wenn lediglich Teile des Booleschen Schaltkreises verborgen bleiben [ST97b].

Das von SANDER und TSCHUDIN entwickelte Protokoll arbeitet wie folgt:

1. Alice verschlüsselt f.
2. Alice erzeugt ein Programm $P(E(f))$ welches $E(f)$ implementiert.

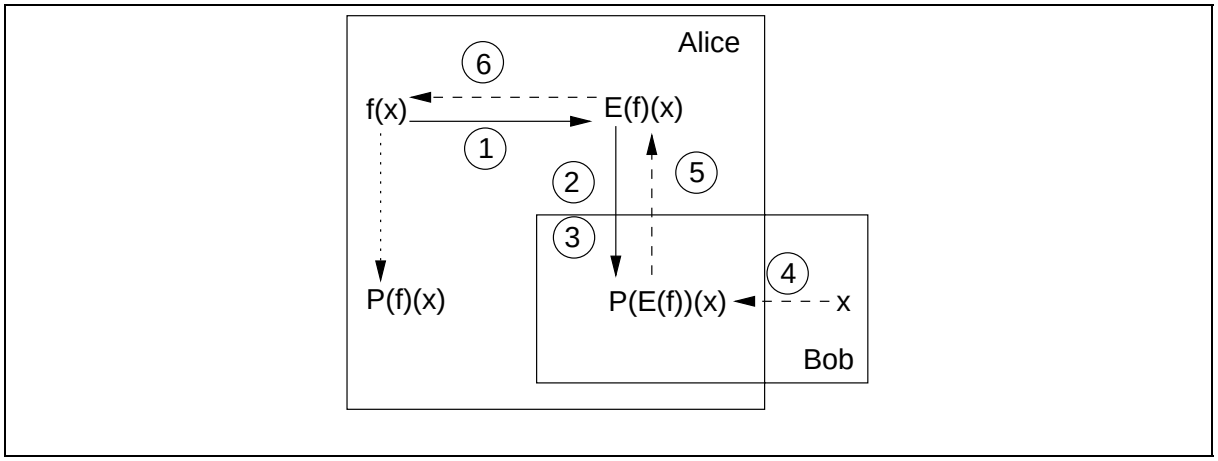

Abbildung 6.6.: 1. Alice verschlüsselt eine Funktion $f()$, 2. Alice implementiert $E(f)$, 3. Alice versendet $P(E(f))$ an Bob, 4. Bob wendet $P(E(f))$ auf x an, 5. Bob sendet das Resultat an Alice, 6. Alice entschlüsselt das Ergebnis (nach [ST97a]).

3. Alice sendet $P(E(f))$ an Bob.

4. Bob führt $P(E(f))$ auf x aus.

5. Bob sendet $P(E(f))(x)$ an Alice.

6. Alice entschlüsselt $P(E(f))(x)$ und erhält $f(x)$.

FEIGENBAUM und MERRIT haben 1991 die Frage aufgeworfen, ob es eine Public-Key Verschlüsselungsfunktion E gibt, so daß sowohl $E(x+y)$ und $E(xy)$ einfach aus $E(x)$ und $E(y)$ berechnet werden können. Falls eine derartige Funktion existiert, könnten polynomiale Ausdrücke berechnet werden, ohne daß die Eingabe- und Ausgabedaten offengelegt werden müßten. Bislang gibt es keine Funktion, die die oben aufgestellte Bedingung erfüllt [ST97a].

SANDER und TSCHUDIN haben entdeckt, daß die obige Forderung vereinfacht werden kann. Ein Klartextprogramm P soll auf ein verschlüsseltes Programm P_E abgebildet werden, so daß Alice in der Lage ist, $P(x)$ aus $P_E(x)$ für einen unbekannten Eingabewert x abzuleiten [ST97b]. Anforderung für die Programme P und P_E ist, daß beide zueinander kompatibel sind. Die mathematische Analogie hierzu ist ein algebraische Struktur. Die Analogie zu kompatiblen Transformationen sind Homomorphismen. Die für die Programme erforderliche Kompabilität muß nicht notwendigerweise die Stärke des Homomorphismus aufweisen. Anstatt eine Abbildung

$$\delta(x+y) = \delta(x) + \delta(y)$$

zu verlangen, ist es ausreichend, wenn sich $\delta(x+y)$ effizient aus $\delta(x)$ und $\delta(y)$ berechnen läßt.

Das von SANDER und TSCHUDIN entwickelte Verfahren der mobilen Kryptografie basiert auf homomorphen Verschlüsselungsverfahren und Funktionskompositionstechniken [ST97a].

Eine (Verschlüsselungs-) Funktion $E : R \to S$ ist

- ▷ additiv homomorph, wenn es einen effizienten Algorithmus PLUS gibt, um $E(x+y)$ auf $E(x)$ und $E(y)$ zu berechnen, ohne x und y offenzulegen;
- ▷ gemischt multiplikativ homomorph, wenn es einen effizienten Algorithmus MIXED-MULT gibt, um $E(xy)$ aus $E(x)$ und y zu berechnen, ohne x offenzulegen [ST97a].

Für $\boldsymbol{Z}/n\boldsymbol{Z}$-Kryptosysteme impliziert die erste Eigenschaft (additiv homomorph) die zweite (gemischt multiplikativ homomorph). Die Abbildung

$$E : \boldsymbol{Z}/(p-1)\boldsymbol{Z} \to \boldsymbol{Z}/p\boldsymbol{Z}, x \mapsto g^x$$

ist für eine Primzahl p und einen Generator von $(\boldsymbol{Z}/p\boldsymbol{Z})^\times$ additiv homomorph. Um x aus $y := E(x)$ abzuleiten, muß daß logarithmische Problem $y = g^x$ gelöst werden, welches als hart angenommen wird. Alice wählt eine Primzahl p, für die das logarithmische Problem einfach zu lösen ist, und einen Generator g aus $(\boldsymbol{Z}/p\boldsymbol{Z})^x$, den sie geheimhält. Die Sicherheit des Verfahrens hängt damit von der Geheimhaltung von g ab [ST97b].

SANDER und TSCHUDIN haben nachgewiesen, daß es reicht, eine additiv homomorphe Verschlüsselungsfunktion

$$E : \boldsymbol{Z}/\boldsymbol{N}\boldsymbol{Z} \to R$$

zu haben, um eine ausführbare Encrypted Functions (EEF) auf Polynome $p \in \boldsymbol{Z}/\boldsymbol{N}\boldsymbol{Z}$ zu implementieren. Es bleibt jedoch Untersuchungsgegenstand, inwieweit homomorphe Verschlüsselungsschemata den Sicherheitsanforderungen genügen.

Neben EEF bieten Kompositionstechniken einen weiteren Weg, wie eine Berechnung vorborgen werden kann. Unter der Annahme, daß Alice eine rationale Funktion s besitzt, die sie effizient invertieren kann, kann sie eine Funktion f zu

$$h = s \circ f$$

komponieren. Dann ist

$$E(f) = h$$

die verschlüsselte Funktion f und das oben vorgestellte Protokoll kann zur Berechnung von f durchgeführt werden. Die Sicherheit dieses Verfahrens beruht auf der Schwierigkeit der Dekomposition, der Ableitung von f aus $E(f)$. ZIPPEL hat gezeigt, daß kein Algorithmus existiert, der daß Problem der Dekomposition mit polynomiellem Zeitaufwand

löst [ST97a]. SHAMIR hat einen Konstruktionsmechanismus entwickelt, der es erlaubt, effizient invertierbare, rationale Funktionen s zu bestimmen.

Bislang existiert kein Verfahren mit dem sich beliebige Funktionen verschlüsseln lassen, die weiterhin ausführbar sind. Auch für die Menge aller Polynome existiert kein entsprechendes Verfahren. SANDER und TSCHUDIN haben aber gezeigt, daß für eine beschränkte Menge von Polynomen und rationalen Funktionen Verfahren existieren.

6.6.2. Beurteilung

Mobile Kryptografie würde die dritte Dimension des Sicherheitsbegriffes – dem Schutz der Agenten vor den Wirtssystemen – erfolgreich abdecken. Derzeit ist mobile Kryptografie allerdings lediglich ein theoretisches Modell. Die Verschlüsselung von Programmen, die wiederrum ausführbar sind, ist derzeit nicht möglich. Es existieren keine Verfahren, mit denen derartige Kodierungen vorgenommen werden können. Zur Zeit können lediglich rationale Funktionen entsprechend verschlüsselt werden. Es ist nicht absehbar, ob sich zukünftig die Möglichkeit bieten wird, daß von SANDER und TSCHUDIN entwickelte Verfahren einzusetzen.

Ausgegangen von der Annahme, daß sich die erforderlichen Voraussetzungen für das Verfahren erfüllen lassen, würde die mobile Kryptografie ein effektives Verfahren stellen, mit dem sich Agenten vor den Umgebungssystemen schützen ließen. Sowohl die Privatheit der Daten und des Codes als auch die Integrität derer ließe sich gewährleisten. Manipulationen am Agenten könnten weiterhin erfolgen, die Verschlüsselung würde allerdings sicherstellen, daß Manipulationen nicht zielgerichtet vorgenommen werden können. Dieses gilt für den Fall, daß Agenten lediglich interne Methoden benutzen.

Agenten sind jedoch keine in sich abgeschlossene Kapseln. Die Transitive Hülle des Agenten beinhaltet – neben den Objekten des Agenten – Referenzen auf Standardmethoden, die Teil der Ausführungsumgebung sind, in der der Agent ausgeführt wird. Standardmethoden können keine verschlüsselten Werte übergeben werden. So kann die Anfrage nach einer IP-Adresse, für die eine Bibliotheksmethode verwendet wird, nur unverschlüsselt erfolgen. Darüberhinaus kann die Kommunikation mit allen Diensten, die der Agent in Anspruch nimmt, nur in Klartextform erfolgen. Gleiches gilt für die Kommunikation mit anderen Agenten. Es stellt sich somit die Frage, ob mit mobiler Kryptografie Agentensysteme entwickelt werden können, die den Anwendungsfeldern, wie sie in Kapitel 7 vorgestellt werden, gerecht werden können. Es gilt zu untersuchen, in welchem Maße Informationen, die die Agenten durch die Nutzung von Standardmethoden preisgeben, von maliziösen Umgebungen genutzt werden können, um Informationen über einen Agenten zu erhalten. Desweiteren hat ein Agent auch in der Architektur der mobilen Kryptografie keinen Einfluß auf die Ausführung von Standardmethoden. Der Umgebung bietet sich damit die Möglichkeit, dem Agenten falsche Werte als Ergebnis zurückzuliefern. In Rahmen eines Blackbox-Tests könnte so möglicherweise die Semantik eines Agenten ermittelt werden.

Trotz der angeführten kritischen Punkte ist die Architektur der mobilen Kryptografie

ein interessantes Verfahren, daß die Forschung im Bereich der Sicherheit von Agentensystemen wesentlich bereichert und neue Lösungsmöglichkeiten aufzeigen könnte.

6.7. Playground

MALKHI et al. haben eine Sicherheitsarchitektur für Java-Agenten konzipiert, die eine Trennung des Agenten von der Ein- und Ausgabe vorsieht.

6.7.1. Konzept

Agenten werden in dieser Architektur nicht länger auf dem Rechner des Anwenders ausgeführt, der sie angefordert hat; sie werden stattdessen auf einem speziellem System, dem „Playground“ zur Ausführung gebracht. Das System des Anwenders dient lediglich als grafische Schnittstelle zur Kommunikation mit Agenten [MR97].

Die wesentlichen Komponenten dieser Architektur sind der Grafikserver, der Playground und ein Proxy. Der Grafikserver ist auf dem System des Anfragers angesiedelt und stellt die grafische Schnittstelle zwischen dem Agenten und dem Anwender zur Verfügung. Jegliche Interaktion mit dem Agenten erfolgt ausschließlich über diese Schnittstelle. Der Grafikserver stellt gleichzeitig die einzige Komponente dar, der der Anwender vertrauen muß. Der Grafikserver ist als eine Menge von Klassen implementiert, die mit den AWT-Klassen der JVM korrespondieren. Die Klassen des Grafikservers stellen das entfernte Gegenstück zu den AWT-Klassen aus dem JDK dar. Alle Ein- und Ausgaben des Agenten werden an den Grafikserver weitergeleitet. Der Grafikserver selbst kann wiederum als Agent implementiert werden. Der Aufruf von Methoden am Grafikserver und am Agenten erfolgt über RMI.

Der Proxy dient sowohl dem Browser als auch dem Playground als Schnittstelle zum externen Netz. Er bezieht Dokumente für den Browser, Agenten für den Playground und transferiert die Daten in Formate, die vom Browser und Playground genutzt werden können. Alle Aufrufe von AWT-Methoden durch den Agenten werden durch Aufrufe der Playgroundseitigen Stubs für die korrespondierenden entfernten AWT-Methoden am Grafikserver ersetzt. Das schließt das Parsen des Bytecodes nach AWT-Methoden und der textuellen Ersetzung der Methodennamen ein.

Der Playground stellt einen geschützten Bereich ähnlich der Sandbox dar, innerhalb dessen die Agenten ausgeführt werden. Innerhalb des Playgrounds werden alle Operationen des Agenten durchgeführt. Lediglich die grafische Interaktion mit dem Anwender erfolgt über den Grafikserver und stellt damit die einzige Verbindung zum Systems des Anwenders dar.

Der initiale Bezug eines Agenten in dieser Architektur erfolgt durch die Referenzierung eines solchen innerhalb eines HTML-Dokumentes. Eine HTML-Anfrage wird an den Proxy weitergeleitet. Dieser stellt die Verbindung mit dem Server her, auf dem das HTML-Dokument vorliegt. In dem bezogenen HTML-Dokument werden alle Agenten-

Auszeichnungen durch Verweise ersetzt, die den Grafikserver auf Seiten des Anwenders referenzieren. Die derart modifizierte Seite wird an den Browser weitergeleitet, der daraufhin eine Instanz des Grafikservers generiert. Im nächsten Schritt bezieht der Proxy den Agenten aus dem Netz. Nach dem Bezug wird der Code des Agenten dahingehend modifiziert, daß dessen Ein- und Ausgaben Methoden am Grafikserver referenzieren. Anschließend wird der Agent an den Playground weitergeleitet, wo eine Instanz des Agenten erzeugt wird. Von dort aus nutzt der Agent den Grafikserver im Browser des Anwenders als I/O-Terminal [MR97].

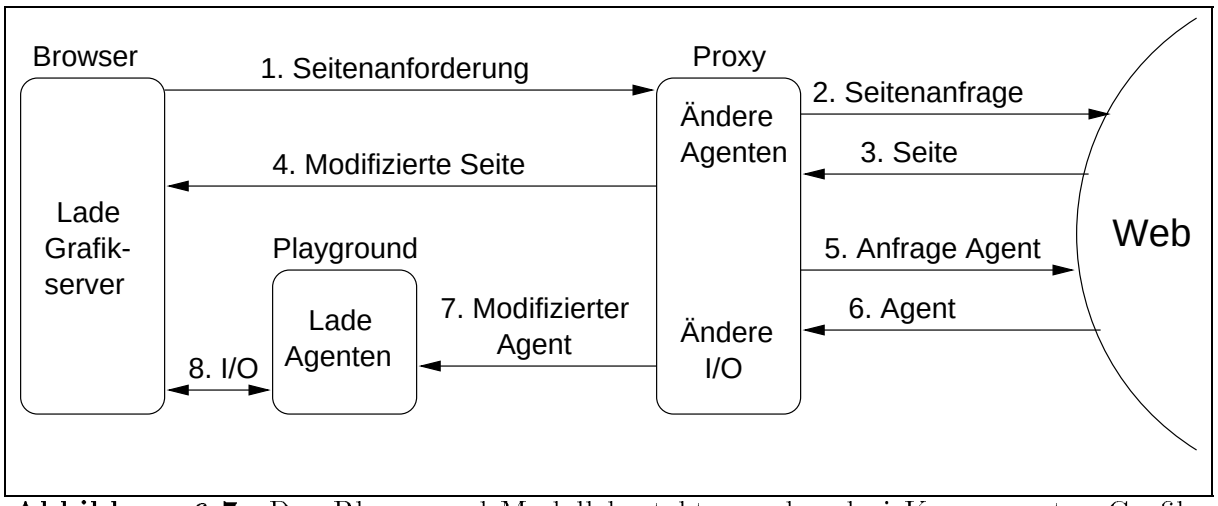

Abbildung 6.7.: Das Playground-Modell besteht aus den drei Komponenten Grafikserver, Playground und Proxy. Agenten werden über einen Proxy bezogen und im Playground ausgeführt. Die Interaktion mit den Agenten erfolgt über den Grafikserver.

Für die Kontaktaufnahme mit dem Agenten wird in dem modifiziertem HTML-Dokument zusätzlich eine Adresse als Parameter eingefügt, über die der Kontakt zum Agenten aufgebaut wird. Im Zuge der Initialisierung bindet das Grafikserver-Objekt eine entfernte Instanz, die durch die Adresse spezifiziert wird, an sich selbst. Der Proxy hält eine Tabelle der Adressen, die jeder Agenten-Auszeichnung zugewiesen werden und teilt dem Playground eine Adresse mit dem gleichen Verfahren zu. Der Parameter des Agenten, der im Playground ausgeführt wird, wird mit der Adresse des Grafikservers ausgestattet, der die Ein- und Ausgaben des Agenten verwaltet und darstellt.

Der Playground-Ansatz stellt sicher, daß Agenten keine Zugriff auf wertvolle Systemressourcen erhalten. Gleichzeitig werden Anwender daran gehindert, beliebige Agenten aus einem Netzwerk zu beziehen. Alle Dokumente referenzieren lediglich den vertrauenswürdigen Grafikserver. Der Applet-Class-Loader des Browsers lädt lediglich lokale Klassen; alle durch den Grafikserver referenzierten Klassen befinden sich im lokalem System.

Eine geringe Anzahl von Java-Klassen lassen sich im Playground nicht ausführen. Die Konstruktion des Playgrounds erfordert keine Änderungen des Browsers.

6.7.2. Beurteilung

Die Playground-Architektur wurde für Java-Agenten entwickelt und stellt eine Möglichkeit dar, die von ihnen für das Anwendersystem ausgehenden Gefahren zu minimieren. Ziel der Architektur ist ausschließlich der Schutz des Wirtssystemes vor den Agenten. Weiterhin beschränkt sich die Architektur von MALKHI et al. ausschließlich auf Java Applets, die von einem Server auf ein Anwendungssystem migrieren.

Die vorgestellte Architektur stellt im wesentlichen eine Problemverlagerung dar. Bei hinreichend korrekter Implementierung gehen vom Playground keine Gefahren für Anwendungssysteme aus. Wie zu gewährleisten ist, daß vom Playground keine Gefahren ausgehen, wird in dem Dokument von MALKHI et al. nicht weiter ausgeführt [MR97]. Ein wesentliches Problem ist die Basis der Architektur, die JVM. Das vorangegangene Kapitel hat gezeigt, daß Java eine Reihe von Sicherheitsmängeln aufweist, die ausgenutzt werden können, um die auf Java aufbauenden Sicherheitsarchitekturen zu umgehen. Es hat sich gezeigt, daß die in Java implementierten Sicherheitsmechanismen nicht ausreichend sind. Implizit geht dieses auch aus der Arbeit von MALKHI et al. hervor, da sie die JVM vom Anwendungssystem auf ein anderes System verlagert haben. Damit verbleibt jedoch die Frage, wie der Playground zu schützen ist. In ihrer Arbeit gehen MALKHI et al. von der Annahme aus, daß auf dem Playground keine sensitiven Daten vorhanden sind, Sicherheitsmängel also zu keinen Problemen führen können. Weiterhin führen sie aus, daß eine Plazierung des Playgrounds außerhalb des Firewall, also außerhalb des lokalen Netzes, erfolgen kann.

Der Zugriff der Agenten auf lokale Ressourcen innerhalb des Netzes beschränkt sich ausschließlich auf die grafische Schnittstelle. Damit erfolgt eine wesentliche Einschränkung der möglichen Anwendungen der Agenten. Ein Einsatz dieser Architektur auf Serversystemen kann damit ausgeschlossen werden.

6.8. Time limited Blackbox

6.8.1. Konzept

Zum Verständnis eines Programmes wird ein mentales Modell des Programmes benötigt. Nur dieses erlaubt es dem Programmierer innerhalb kurzer Zeit, ein Programm zu verstehen. Entsprechend benötigt ein Angreifer, der einen Agenten angreifen will, ein mentales Modell des Agenten, um diesen zielgerichtet manipulieren zu können. So es ein Verfahren gibt, mit dem der Aufbau eines mentalen Modelles eines Agenten erschwert werden kann, ermöglicht dies die Ausführung eines Agenten über einen längeren Zeitraum, ohne daß dieser zielgerichtet manipuliert werden kann. HOHL hat ein entsprechendes Verfahren entwickelt, daß er als „Code-Mess-Up“ bezeichnet. Mit diesem Verfahren werden aus einer Agentenspezifikation eine Vielzahl von Agenten generiert, deren Semantik dem Code nur mit großen Aufwand zu entnehmen ist.

Ohne Begrenzung der Zeit ist es jedoch weiterhin möglich, auch „verschleierten“ Code zu entschleiern und so einen Zusammenhang zwischen Kontrollfluß und Ausführungssemantik herzustellen. Aus diesem Grund ist das Verfahren der Code-Verschleierung alleine nicht ausreichend. Hinzukommen muß eine Beschränkung der Lebenszeit des Codes und der Daten. Hierzu sind in dem von HOHL entwickeltem Verfahren alle wichtigen Datenelemente mit einem „Verfallsdatum“ versehen, nach dessen Ablauf sie ihre Gültigkeit verlieren. Die Verfallsdaten werden digital signiert. Neben der Ausstattung der einzelnen Datenelemente wird auch der komplette Agent mit einem Verfallsdatum versehen und digital signiert.

Ein Agent, der den obigen Ausführungen entspricht, wird von HOHL als zeitlimitierte Blackbox bezeichnet. Nach HOHL ist ein Agent eine zeitlimitierte Blackbox, wenn

- ▷ Code und Daten einer Agentenspezifikation für eine bestimmte Zeit nicht gelesen werden können;
- ▷ Code und Daten einer Agentenspezifikation für eine bestimmte Zeit nicht modifiziert werden können;
- ▷ Angriffe nach Ablauf des Zeitintervalls keinen Einfluß haben [Hoh98].

Ein wesentliches Verfahren zur Verschleierung der Semantik eines Agenten ist die Rekomposition. Rekomposition unterteilt die Variablen der Agentenspezifikation in eine Menge von Segmente. Jede Variable des generierten Agenten enthält lediglich ein Segment einer ursprünglichen Variable. Der Zugriff auf eine ehemalige Variable erfolgt durch Konvertierungsmethoden, die eine Menge von Subelementen als Argumente nehmen, und als Ergebnis den Variableninhalt liefern.

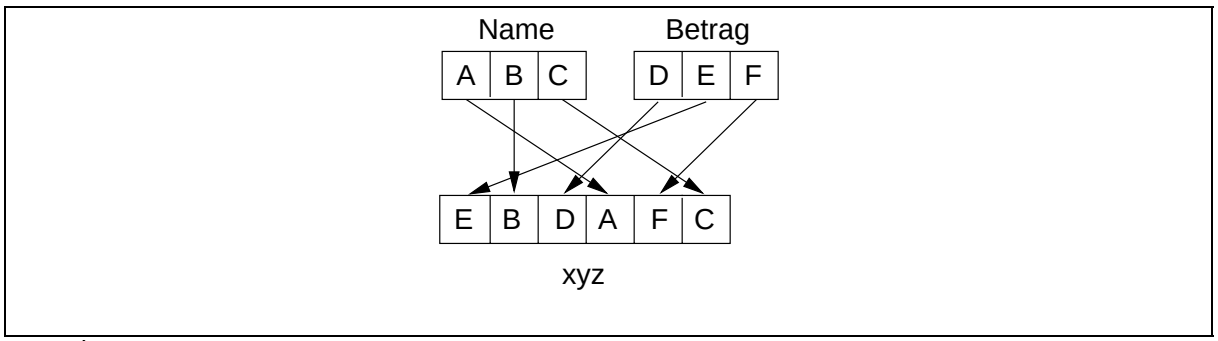

Abbildung 6.8.: Verwürfelungsalgorithmen sorgen dafür, daß Variablen in Segmente unterteilt und die Segmente zu neuen Variablen zusammengesetzt werden. Die Semantik der ursprünglichen Variablen geht damit verloren. Konvertierungsroutinen sind erforderlich, um auf die ursprünglichen Inhalte zugreifen zu können.

In einem nächsten Schritt wird die Programmstruktur, wie sie in prozeduralen Sprachen durch Blöcke und Prozeduren gegeben ist, aufgelöst. Es wird ein Programmcode erzeugt, der abschließend nahezu keine innere Struktur aufweist. Folgende Schritte werden vollführt:

- ▷ Lokale Variablen werden durch Variablen mit globaler Sichtbarkeit ausgetauscht;
- ▷ Prozeduraufrufe werden durch den Code der Prozedur ersetzt;
- ▷ Blöcke werden durch Anweisungen ersetzt, die eine der `goto`-Anweisung entsprechende Semantik aufweisen;
- ▷ Codefragmente werden eingefügt, die von Agenten zu keinem Zeitpunkt aufgerufen werden („toter Code“).

Ein weiteres Verfahren sorgt dafür, daß der Kontrollfluß des Agenten nicht statisch dem Programmtext entnommen werden kann. Statische Kontrollflußelemente werden durch Sprünge ersetzt, die von Daten der Laufzeit abhängen. Die Kontrollflußanweisung, die hier im wesentlichen zum Einsatz kommt, ist die `case`-Anweisung. Es erfolgt eine Konvertierung des zur Kompilierungszeit festgelegten Kontrollflusses in zur Laufzeit zu bestimmende Sprünge. Das Verfahren kann verbessert werden, wenn statt einfacher Variablen komplexe Ausdrücke verwendet werden.

Schließlich werden alle Aufrufe von sensitiven Bibliotheksfunktionen durch den korrespondierenden Code der Funktion ersetzt, bevor der Konvertierungsalgorithmus auf den Agenten angewendet wird.

HOHL stellt die folgenden Anforderungen an den Konvertierungsalgorithmus:

- ▷ Der Algorithmus muß mit einem sehr großen Raum von Parametern parametrisierbar sein, so daß Wörterbuch-Angriffe (engl. Dictionary-Attacks) verhindert werden können.
- ▷ Der Schutz des Agenten darf nicht durchbrochen werden, ohne daß der Code ausgeführt worden ist.
- ▷ Teile des Agenten sollen auf vertrauenswürdige Systeme transportierbar sein [Hoh98].

Hochwertige Algorithmen zur Erstellung polymorpher Agenten sollten sich nicht vollständig statisch analysieren lassen. Das funktionale Verhalten eines Agenten wird erst durch seine teilweise Ausführung festgelegt.

Die zeitliche Limitierung erfolgt durch die Einführung von Verfallsdaten für Datenelemente. Daten mit Verfallsdatum können nur von vertrauenswürdigen Instanzen stammen. Insgesamt ist es lediglich für eine beschränkte Anzahl von Datenelementen erforderlich, daß sie ein Verfallsdatum beinhalten. Daten dieser Klasse sind selbstenthalten und können gegen andere ausgetauscht und für Dienste eingesetzt oder als Beweis für eine Identität eingesetzt werden. HOHL bezeichnet diese Daten als „Token“. Ein Token besteht aus den ursprünglichen Daten, dem Herausgeber und einem Verfallsdatum. Ein derartiges Token wird mit einer digitalen Signatur versehen. Ausschließlich Token werden mit einem Verfallsdatum versehen.

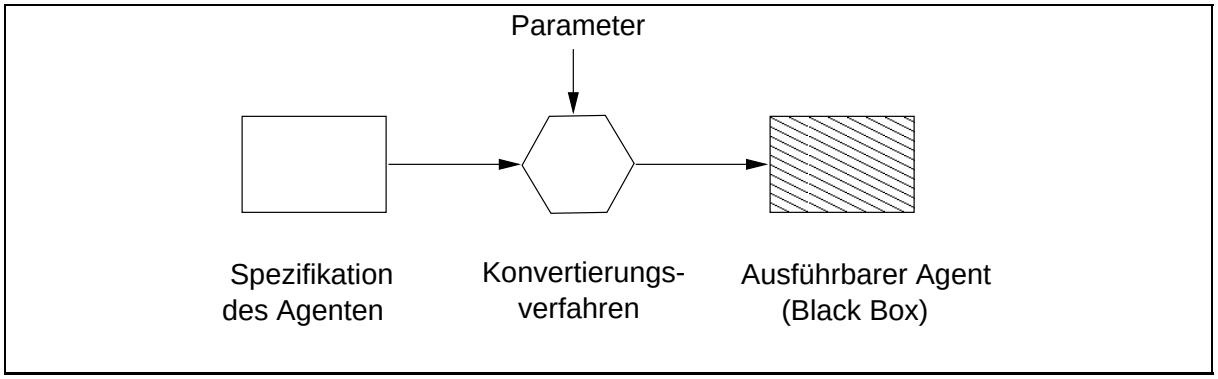

Abbildung 6.9.: Aus einer Spezifikation eines Agenten wird mittels einer parametrisierbaren Konvertierungsfunktion ein Blackbox-Agent erzeugt.

Schlüssel können nicht durch Token geschützt werden, da ihre Lebensdauer zu lang ist, als daß das Blackbox-Verfahren den Schlüssel über diesen Zeitraum schützen könnte. Andere Daten als Token können nicht zur Kommunikation mit Dritten verwendet werden, da diese nicht geschützt sind.

Die Gültigkeit der Datenelemente, die nicht mit einem Verfallsdatum versehen sind, entspricht der Gültigkeit des gesamten Agenten, der ebenfalls mit einem Verfallsdatum versehen wird. Daten, deren Verfallsdatum abgelaufen ist, werden von anderen Entitäten nicht akzeptiert. Eine maliziöse Umgebung kann also in den Besitz der Daten des Agenten gelangen, kann aber nach Ablauf des Verfallsdatums diese nicht weiterverwenden.

Zeitlimitierte Blackbox-Agenten ähneln in gewisser Weise dem von SANDER und TSCHUDIN entwickelten Modell der mobilen Kryptografie. Im Gegensatz zu dem oben angeführten Lösungsansatz wird im Falle der zeitlimitierten Blackbox-Agenten aber lediglich eine einfache Umbenennung von Bezeichnern und eine Umstellung von Codefragmenten durchgeführt, die keinen Einfluß auf das Ergebnis nehmen. Dementsprechend liegt das berechnete Ergebnis des Agenten auch in Klartextform vor.

6.8.2. Beurteilung

Das von HOHL entwickelte Verfahren der zeitlimitierten Blackbox hat eine Sicherheitsarchitektur zum Schutz des Agenten vor der Umgebung zum Ziel. Die anderen Sicherheitsdimensionen werden mit dieser Architektur nicht adressiert. Ebenso wie das Verfahren der mobilen Kryptografie, soll die Privatheit sowie die Integrität von Code und Daten gewährleistet werden.

HOHL hat ein Verfahren entwickelt, mit dem es ihm möglich ist, den Code eines beliebigen Agenten zu manipulieren, so daß es scheinbar nicht möglich ist, den Code in kurzer Zeit wieder herzustellen. Das diese Invertierung nur mit erheblichem Zeitaufwand vorgenommen werden kann, ist jedoch lediglich eine Annahme. Solange nicht gezeigt ist, daß es sich um eine Einwegfunktion handelt, kann ein Verfahren entwickelt werden, daß es innerhalb kurzer Zeit ermöglicht, den ursprünglichen Code eines Agenten wieder

herzustellen. „Code-Mess-Up“ kann nicht nachweislich sicherstellen, daß die definierten Sicherheitsziele auch erreicht werden.

In auf zeitlimitierte Blackbox-Agenten basierende Agentensystemen wird in regelmäßigen Zeitabständen ein neuer Agent – eine Mutation – generiert, der die gleiche Funktionalität, jedoch eine abweichende Kodierung aufweist. Polymorphe Techniken sind bereits aus der Virenforschung bekannt. Hier wird sie von den Virenautoren eingesetzt, um die Erkennung des Viruses durch ein Antivirenprodukt zu verhindern. Es gibt seit längerem jedoch zuverlässige Mechanismen, mit denen polymorphe Viren erkannt werden können. Es ist davon auszugehen, daß entsprechende Mechanismen auch für polymorphe Agenten entwickelt werden können. Demnach ist ein erhöhter Aufwand notwendig, um die erste Generation eines polymorphen Agenten zu dekodieren, nachfolgende Generationen können dann mittels entsprechend entwickelter Werkzeuge automatisch durch das Wirtssystem dekodiert und gezielt manipuliert werden.

Es ist nicht ausreichend, die Manipulation von Agenten zu erschweren. Transaktionen, die im Umfeld der elektronischen Dienstemärkte erfolgen, beinhalten eine rechtliche Bindung für den erfolgten Vorgang. Es ist somit erforderlich, daß die von den Teilnehmern gewünschten Sicherheitsgarantien nachweislich eingehalten werden. Die Darlegung, daß Manipulation erschwert worden sind, ist in diesem Zusammenhang weit weniger als ausreichend. Ohne weitreichende Garantien wird niemand bereit sein, entsprechend unsichere Technologien langfristig einzusetzen.

Maliziöse Rechner können alle Datenelemente beliebig lange manipulieren, nachfolgende Rechner werden die Ergebnisse in den Datenelementen nur dann akzeptieren, wenn die Lebenszeit der Datenelemente nicht bereits abgelaufen ist. In [Hoh97] wird ausgeführt, daß die Lebenszeit der Datenelemente wenige Sekunden umfassen sollte. In Bezug auf Multihop-Agenten ist dieser Zeitraum viel zu kurz gewählt. Auch wenn die auf den jeweiligen Zwischenstationen durchgeführten Berechnungen nur wenige Sekunden Ausführungszeit benötigen, erfordert das Marshalling/Unmarshalling der Agenten einschließlich des Transports wesentlich mehr Zeit. Beim Transport des Agenten über das Internet kommt zudem i. d. R. ein unzuverlässiger Übertragungsdienst zum Einsatz, so daß der Verlust von Datenpaketen zu weiteren Verzögerungen führen wird.

Das Verfallsdatum hilft nicht, die Vertraulichkeit der Daten des Agenten sicherzustellen. Mit Hilfe des Verfallsdatum wird lediglich sichergestellt, daß die maliziöse Umgebung die Daten nicht weiterverwenden kann. Offen ist dabei, ob wirklich jeder Dienst, der die von der maliziösen Umgebung entdeckten Daten, erhält lediglich jene mit gültigem Datum akzeptiert. Die Heterogenität der offenen, verteilten Netze und der dort angebotenen Dienste läßt es wahrscheinlicher sein, daß eine maliziöse Umgebung einen Dienst finden wird, der die „erbeuteten“ Daten auch ohne bzw. mit abgelaufenem Verfallsdatum akzeptieren wird. Sensitive Informationen wie Firmengeheimnisse und private Daten können nicht durch Token geschützt werden, da es für einen Angreifer ausreichend ist, in ihren Besitz zu gelangen. Die Daten werden von Angreifern nicht in Token-Form weitergegeben werden.

Agenten bleiben angreifbar, wenn sie mit anderen Agenten kommunizieren. Kommu-

nikation zwischen Agenten kann nicht in verschlüsselter Form erfolgen, so daß maliziöse Wirtssysteme die Möglichkeit haben, Einfluß auf den Kommunikationsfluß zu nehmen. Eine digitale Signierung der Nachrichten ist nicht möglich, da Agenten keine privaten bzw. symmetrischen Schlüssel mit sich führen können.

HOHL führt selber aus, daß derzeit nicht absehbar ist, welche Angriffsmöglichkeiten für Blackbox-Modelle existieren [Hoh98]. Die Entwicklung eines formalen Modells ist erforderlich, um systematisch nachweisbar sichere Lösungen für das Verstecken von Variablen zu erarbeiten.

Negativen Einfluß hat das Verfahren des „Code-Mess-Up" auf die Zuverlässigkeit der Agenten. In diesen Systemen hängt die Korrektheit eines Agenten zusätzlich von der Korrektheit des Mutationsgenerators ab. Die gestiegene Komplexität der Agentengenerierung erhöht gleichzeitig die Gefahr von Implementationsfehlern. Das Problem verschärft sich, da durch die Verschleierung der mutierte Agent nicht mehr lesbar im Sinne der Softwaretechnik ist und Implementationsfehler in diesem Code deshalb nahezu nicht mehr nachzuvollziehen sind.

Black-box Tests können durchgeführt werden, um zu ermitteln, welche Funktion der Agent erfüllt. Von Interesse sind insbesondere jene Speicherbereiche in denen Veränderungen stattgefunden haben.

Konvertierungsroutinen können als Angriffspunkt genutzt werden, um in kurzer Zeit die ursprünglichen Variableninhalte wieder herzustellen. Dazu ist es für den Angreifer erforderlich, innerhalb kurzer Zeit die entsprechenden Routinen zu identifizieren.

6.9. Soziale Kontrolle

6.9.1. Konzept

RASMUSSON und JANSSON schlagen „sanfte" Sicherheit als ein Sicherheitskonzept für Agentensysteme vor. Sanfte Sicherheit entsteht durch soziale Kontrolle in offenen Netzen. Sanfte Sicherheit erwartet und akzeptiert, daß es möglicherweise unerwünschte Eindringlinge gibt [RJ96]. Ziel ist die Entdeckung der Eindringlinge und die Verhinderung der Schädigung weiterer Akteure. Es soll keine Möglichkeit bestehen, auf einem Wege alle Sicherheitsmechanismen des Systems zu umgehen. Teilnehmer sind solange akzeptiert, wie sie niemanden schädigen. So eine maliziöse Aktion durchgeführt wird, soll dem Teilnehmer der Zugang zum System verweigert.

Die soziale Kontrolle leitet aus dem Gruppenverhalten der Teilnehmer ab, ob ein Agent maliziöses Verhalten aufweist. Soziale Kontrolle soll erwirken, daß die Teilnehmer des Systems selbst für die Sicherheit verantwortlich sind. Soziale Kontrolle soll in offenen Netzen ohne zentrale Instanz eingesetzt werden. RASMUSSON et al. sehen es nicht als erforderlich an, daß die Agenten sich bei Eintritt in das Agentensystem zunächst registrieren [RRJ97].

Soziale Kontrolle orientiert sich an dem Umgang, wie er in Transaktionen der Real-

welt erfolgt. Aufbauend auf einer Reputation, die eine Organisation, ein Unternehmen oder eine Person besitzt, ist man bereit, ein Geschäft mit der Gegenseite zu tätigen. Weist die Gegenseite eine schlechte Reputation auf, hat dies zur Folge, daß keine Entität bereit sein wird, ein Geschäft mit dieser zu tätigen. Dieses Vorgehen übertragen RASMUSSON et al. auf Agentensysteme.

Ein wichtiger Aspekt sind dabei Ergebnisse der Spieltheorie. RASMUSSON zieht das Gefangenen-Dilemma für Agentensysteme heran. Treffen Agenten lediglich einmalig aufeinander, suchen sie den höchst möglichen Gewinn, der darin besteht, daß sie den anderen Agenten hintergehen. Treffen Agenten jedoch mehrmals aufeinander, ist eine Strategie überlegen, gemäß der Agenten sich solange gutartig verhalten, wie sich auch die Kooperationpartner gutartig verhalten.

Agenten nutzen „Klatsch“ (engl. gossip) um Informationen über die Vertrauenswürdigkeit anderer Agenten zu erhalten. Gossip-Agenten teilen einem Agenten A proaktiv mit, ob eine Kooperation mit Agent B eingegangen werden kann. Ein Agent, der als erster mit einem neuen Agenten in Kontakt tritt, geht dabei das Risiko ein, daß er an einen maliziösen Agenten gerät, der bislang keine Reputation aufweisen kann.

Falschmeldungen bezüglich der Reputation von Agenten können eingeschränkt werden, wenn der Agent für die Verbreitung von Gerüchten bezahlen muß. Der Vorgang der Verbreitung von Gerüchten kann als „Werbung“ betrachtet werden, da Agenten die Möglichkeit haben, Gerüchte über ihre eigene positive Reputation zu verbreiten.

Agenten sollen in die Lage versetzt werden, miteinander zu interagieren, Informationen bereitzustellen, ohne daß maliziöse Agenten die Tätigkeit gutartiger Agenten negativ beeinflussen.

Es ist Aufgabe der Agenten zu entscheiden, welchen Agenten zu vertrauen ist. RASMUSSON et al. haben hierzu eine Reihe von Möglichkeiten vorgestellt, wie die soziale Kontrolle in Agentensystemen umgesetzt werden kann. Ein Verfahren der sozialen Kontrolle besteht im Handeln mit Informationen über die Vertrauenswürdigkeit von Agenten.

6.9.2. Beurteilung

Das von RASMUSSON et al. auf die Agentensysteme übertragene System der sozialen Kontrolle ist keine Sicherheitsarchitektur bzw. kein technischer Sicherheitsmechanismus, wie er in der Computersicherheit entwickelt wird. Das Verfahren basiert ausschließlich auf der Annahme, daß sich das Verhalten, wie es in der Realwelt existiert, auf Agentensysteme übertragen läßt. Agenten erwerben hierfür eine Reputation, die im folgenden als Entscheidungsbasis für andere Agenten genommen wird, um zu entscheiden, ob eine Kooperation mit einem Agenten eingegangen werden kann.

Eine Reihe von Aspekten werden in den Arbeiten von RASMUSSON et al. nicht adressiert. Zunächst gehen sie von der Annahme aus, daß Agenten „leben“. Agenten besitzen eine Reputation, die sie während ihrer Lebenszeit ausbauen. Dem liegt die implizite Annahme zugrunde, daß Agenten langlebige Objekte sind, die mehr als einmal mit bestimmten anderen Agenten in Verhandlungen treten. Nur dann ließe sich mit Hilfe des

Gefangenen-Dilemmas der Schluß ziehen, daß Agenten den höchsten Gewinn dann erzielen läßt, wenn sie mit gutartigen Agenten kooperieren. Dies steht im Widerspruch zu den bislang existierenden Agentensystemen, in denen Agenten einen lediglich temporären Charakter haben. Weiterhin läßt sich auch in Systemen mit langlebigen Agenten nicht ausschließen, daß Agenten zum Einsatz kommen, deren einziges Ziel es ist, Sicherheitslücken ausnutzen, um so Kontrolle über Komponenten des Agentensystemes zu erhalten. Es handelt sich dabei um Angriffe, die sich nicht mit betriebswirtschaftlichen Maßstäben bewerten lassen.

Agenten haben die Möglichkeit, die Reputation anderer Agenten negativ zu beeinflussen, in dem sie fälschlicherweise behaupten, sie wären von diesen angegriffen worden. Maliziöse Agenten haben desweiteren die Möglichkeit, eine Vielzahl von Agenten zu produzieren, die alle im Sinne des Produzenten aussagen, d. h., diesem zu einer positiven Reputation verhelfen, auch wenn dieser maliziöse Aktionen durchführt.

Bereits im letzten Kapitel wurde darauf verwiesen, daß sich Verfahren der Realwelt nicht unbedingt auf Agentensysteme übertragen lassen. In Hinblick auf die Übertragung des Systems der sozialen Kontrolle ist ein wesentlicher weiterer Schwachpunkt die nicht vorhandene rechtliche Kontrolle der durchgeführten Aktionen. Schlagen Transaktionen in der Realwelt fehl, steht der betrogenen Partei die Möglichkeit offen, juristische Schritte einzuleiten. In der Welt der Agentensysteme ist ein entsprechendes Äquivalent derzeit nicht existent. Teilnehmer in Agentensystemen können aus unterschiedlichen Rechtsräumen stammen, die untereinander unvereinbar sind und damit eine juristische Verfolgung von maliziösen Aktionen unmöglich machen.

Zusammenfassend stellt sich heraus, daß das Konzept der sozialen Kontrolle nicht in der Lage ist, maliziöse Aktionen zu verhindern. Weiterhin fehlen in Agentensystemen die Mittel, mit denen nachgewiesen werden kann, wer für maliziöse Aktionen verantwortlich ist. Die derzeitigen Agentensysteme lassen sich nicht alleine mit betriebswirtschaftlichen Maßstäben bewerten, so daß die von RASMUSSON et al. erzielten Ergebnisse hier nicht anwendbar sind.

6.10. Vertrauenswürdige Hardware

Mittlerweile existieren eine Reihe von Sicherheitsarchitekturen für Agentensysteme, die auf die Existenz vertrauenswürdiger Hardware aufbauen. Im folgenden werden zwei der Architekturen vorgestellt.

6.10.1. Kryptografisch geschützte Objekte

Das von WILHELM entwickelte Konzept basiert auf einer vertrauenswürdigem Tamper Proof Environment (TPE). Das TPE stellt einen vollständigen Rechner dar, der eine virtuelle Maschine zur Ausführung von Agenten beinhaltet. Die vertrauenswürdige Umgebung unterstützt

- ▷ das Laden, Migrieren und Entfernen von Agenten;
- ▷ die Interaktion zwischen Host und Agent oder zwischen verschiedenen Agenten im dem TPE;
- ▷ die Verifikation mehrerer Eigenschaften des TPE [Wil97a].

Eine TPE ist eine Blackbox. Die Interaktion zwischen Nutzer und dem TPE erfolgt ausschließlich über eine eingeschränkte Schnittstelle, die sich unter vollständiger Kontrolle des TPEs befindet. Ein direkter Zugriff auf Objekte innerhalb der vertrauenswürdigen Umgebung ist nicht möglich. Die externe Schnittstelle zur Ausführungsumgebung wird durch das Betriebssystem bereitgestellt.

Die Architektur von WILHELM erfordert lediglich, daß der Besitzer eines Agenten dem Hersteller des TPEs vertraut. Mit dem Vertrauen, daß dem Hersteller entgegengebracht wird, vertraut der Besitzer des Agenten darauf, daß die veröffentlichte Sicherheitspolitik des TPEs durchgesetzt wird. Dem Besitzer des TPEs hat keine Möglichkeit, die Ausführung des Agenten zu beeinflussen.

Der Transport von Agenten erfolgt mittels des CryPO-Protokolls ausschließlich in verschlüsselter Form. Das Protokoll unterteilt sich in die Phasen der Initialisierung und der der Nutzung der vertrauenswürdigen Hardware.

Die Umgebung besitzt einen privaten Schlüssel, der initial bei Inbetriebnahme des Gerätes generiert wird und somit niemanden bekannt ist, auch nicht dem Besitzer des TPE. Das TPE ist mit einem Rechner verbunden, der unter Kontrolle des Besitzers steht. Über eine wohldefinierte Schnittstelle sind die nachfolgenden Operationen möglich:

- ▷ die Veröffentlichung des öffentlichen Schlüssels (K_{TM}) des Herstellers;
- ▷ den Versand des Zertifikates $Cert_{TPE} = (K_{TPE})_{S_{TM}}$ durch den Hersteller an den Besitzer des TPEs und
- ▷ die Registrierung der Referenz[9] des Besitzers des TPEs in einem oder mehreren Verzeichnisdiensten [WBS98].

Neben der Gewährleistung durch den Hersteller, daß von außen kein Einfluß auf die Maschine genommen werden kann, der nicht die wohldefinierte Schnittstelle nutzt, wird ein Zertifikat für die Ausführungsumgebung innerhalb der Maschine mitgeliefert. Das vom Hersteller signierte Zertifikat enthält Informationen über den Hersteller, den Typ, den Garantien und den öffentlichen Schlüssel des TPEs. Das Zertifikat erhält der Besitzer des TPEs. Der Besitzer eines Agenten muß dem Hersteller des TPEs vertrauen, daß die Maschine sich entsprechend den postulierten Garantien verhält.

In der Nutzungsphase stellt der Besitzer eines Agenten zunächst eine Anfrage an einen Verzeichnisdienst, um die Referenz der Umgebung zu ermitteln, in der der Agent ausgeführt werden soll. Der Besitzer des Agenten verifiziert das Zertifikat des TPEs und

[9] Die Referenz besteht aus der physikalischen Netzwerkadresse und dem Zertifikat $Cert_{TPE}$ des TPEs.

überprüft, ob die Sicherheitsgarantien der Ausführungsumgebung seinen Sicherheitsanforderungen entsprechen. Im positiven Fall wird der Agent mit dem öffentlichen Schlüssel K_{TPE} des TPE verschlüsselt und zur Umgebung versandt.[10] Das Wirtssystem, das den Agenten nicht entschlüsseln kann, leitet diesen an das TPEs weiter. Diese entschlüsselt den Agenten und führt diesen aus. Am Ende der Ausführung wird der Agent in verschlüsselter Form zurück an sein Ausgangssystem versandt. Der Schutz des TPEs vor dem Agenten wird durch das CryPO-Protokoll nicht adressiert.

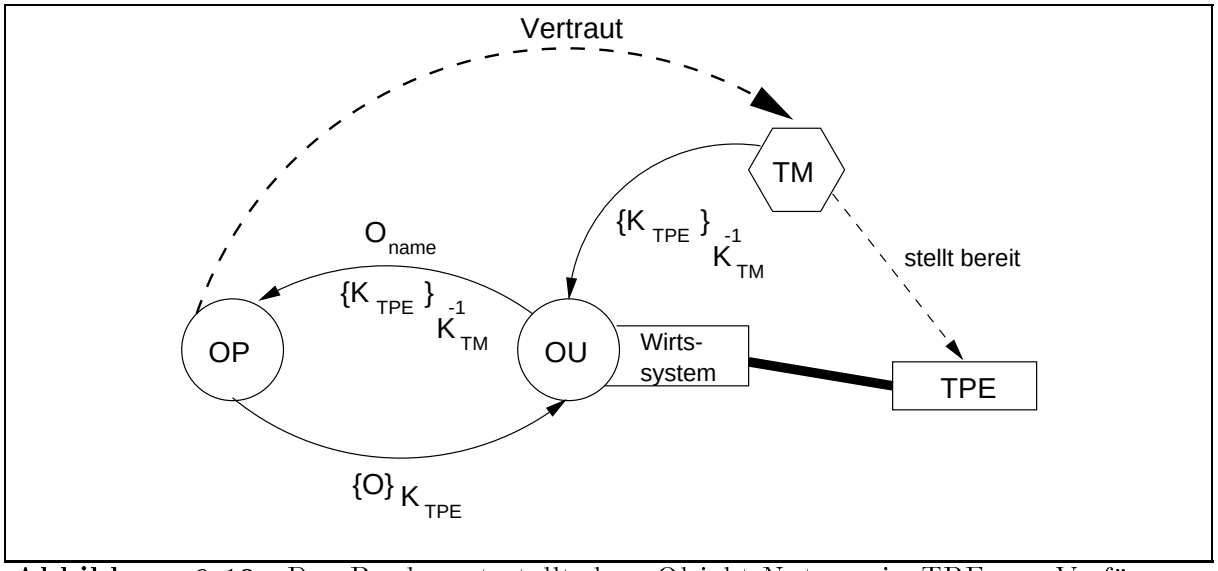

Abbildung 6.10.: Der Produzent stellt dem Objekt-Nutzer ein TPE zur Verfügung. Der Nutzer fordert vom Objekt-Produzenten ein bestimmtes Objekt an und übergibt dazu das Zertifikat des TPEs. Mit dem dort enthaltenen öffentlichen Schlüssel wird das Objekt verschlüsselt und kann somit vom Objekt-Nutzer nur an das TPE weitergeleitet werden, ohne Einblick in das Objekt zu erhalten (nach [Wil97a]).

Im Falle, daß ein Agent auf ein anderes TPE migrieren will, bezieht das TPE das Zertifikat des Zielsystems. Das TPE überprüft, ob das Zielsystem die Sicherheitsanforderungen des Agenten erfüllt. Auf welcher Basis die Entscheidung getroffen wird, wurde bislang nicht definiert [Wil97b]. Anschließend wird der Agent mit dem öffentlichen Schlüssel des Zielsystems verschlüsselt und übertragen.

WILHELM et al. sehen in ihrer Architektur eine Reihe von Vorteilen gegenüber anderen Vertrauensmodellen. TPEs würden durch spezialisierte Hersteller konstruiert, die ein hohes Verständnis für Sicherheits- und Datenschutzprobleme aufbrächten. Eine zentralisierte Kontrolle ergibt sich für die Nutzer, da nur wenige Unternehmen als Hersteller von TPEs agieren werden. Die Herstellung von TPEs ist schwierig und könne somit nur von Unternehmen unternommen werden, die über die erforderlichen Ressourcen verfügen, um eine gute Reputation aufzubauen. Schließlich ist der Hersteller des TPEs

[10] Der Besitzer fügt dem Agenten zusätzlich seinen öffentlichen Schlüssel bei.

ausschließlich für die Durchsetzung der Sicherheitspolitik zuständig. Er hat nichts mit den Dienstleistungen des Besitzers des TPEs zu tun [WBS98].

6.10.2. Zentrale Konfigurationsverwaltung

Die von DEVANBU et al. vorgestellte Sicherheitsarchitektur löst Sicherheitsprobleme durch vertrauenswürdige Hardware im Verbund mit verifizierender Software und einer zentralen Konfigurationsverwaltung. Sicherheitslücken auf Seiten der Wirtssysteme werden nicht durch Aktualisierungen der Software sondern durch eine Schlüsselverwaltung geschlossen. Die Kombination aus Schlüsselverwaltung und Verifikation soll gewährleisten, daß Sicherheitslücken in den Wirtssystemen nicht ausgenutzt werden, obwohl diese weiterhin existieren.

Im Gegensatz zu WILHELM, wird die vertrauenswürdige Hardware im Konzept von DEVANBU et al. auf Seiten des Produzenten des Agenten angesiedelt. Die vertrauenswürdige Hardware wird dort für die Analyse des Agenten eingesetzt. Die eingesetzte vertrauenswürdige Software analysiert Agenten dahingehend, ob von diesem eine Reihe von Sicherheitsmerkmalen eingehalten werden. Ein mögliche Vorgehensweise ist beispielsweise, die Ansiedlung des Bytecode-Verifiers – wie er in Java eingesetzt wird – auf die Seite des Code-Produzenten zu verlagern. Die vom Produzenten des Agenten nicht beeinflußbare Maschine unterzeichnet einen Agenten mit einer digitalen Signatur, so er der durchgeführten Analyse standgehalten hat.

Die vertrauenswürdige Software auf Seiten des Produzenten attestiert mit der Signierung des Agenten, daß dieser erfolgreich verifiziert worden ist. Der Agent A wird zusammen mit D, daß die durchgeführte Verifikation beschreibt und Informationen über die Version der Verifikationssoftware und der Kennung des Herstellers enthält, versandt. Das Wirtssystem Z erhält vom Quellsystem Q den Agenten A zusammen mit D und der Signatur $(A, D)_{Q_S}$. Das Wirtssystem verifiziert die Gültigkeit der Signatur und überprüft im Falle eines positiven Ergebnisses, ob die Sicherheitspolitik des Agenten den lokalen Sicherheitsanforderungen entspricht.

Aufgabe der zentralen Verwaltung ist die Aktualisierung der vertrauenswürdigen Software und der korrespondierenden Schlüssel, die innerhalb der vertrauenswürdigen Hardware auf Seiten des Produzenten installiert sind. Damit verbunden ist die Aktualisierung jener Zertifikate, für die es in der vertrauenswürdigen Hardware auf Seiten der Produzenten korrespondierende Schlüssel gibt. Die Schlüsselverwaltung kann dazu genutzt werden, defekte bzw. veraltete Geräte aus dem Dienst zu nehmen. Eine weitere Aufgabe ist die Behebung von Sicherheitsproblemen auf Seiten der Wirtssysteme. Zusätzlich sollen die Risiken durch eine Kompromittierung der vertrauenswürdigen Software vermindert werden [DFS98].

Im Falle, daß ein Fehler in der Verifikationssoftware entdeckt wird, zieht das Konfigurationsmanagement das Zertifikat der vertrauenswürdigen Software zurück. Erhält ein Wirtssystem Z im Anschluß einen Agenten, der mit einem Schlüssel signiert wurde, dessen Zertifikat mittlerweile zurückgezogen wurde, kann die Signatur nicht verifiziert

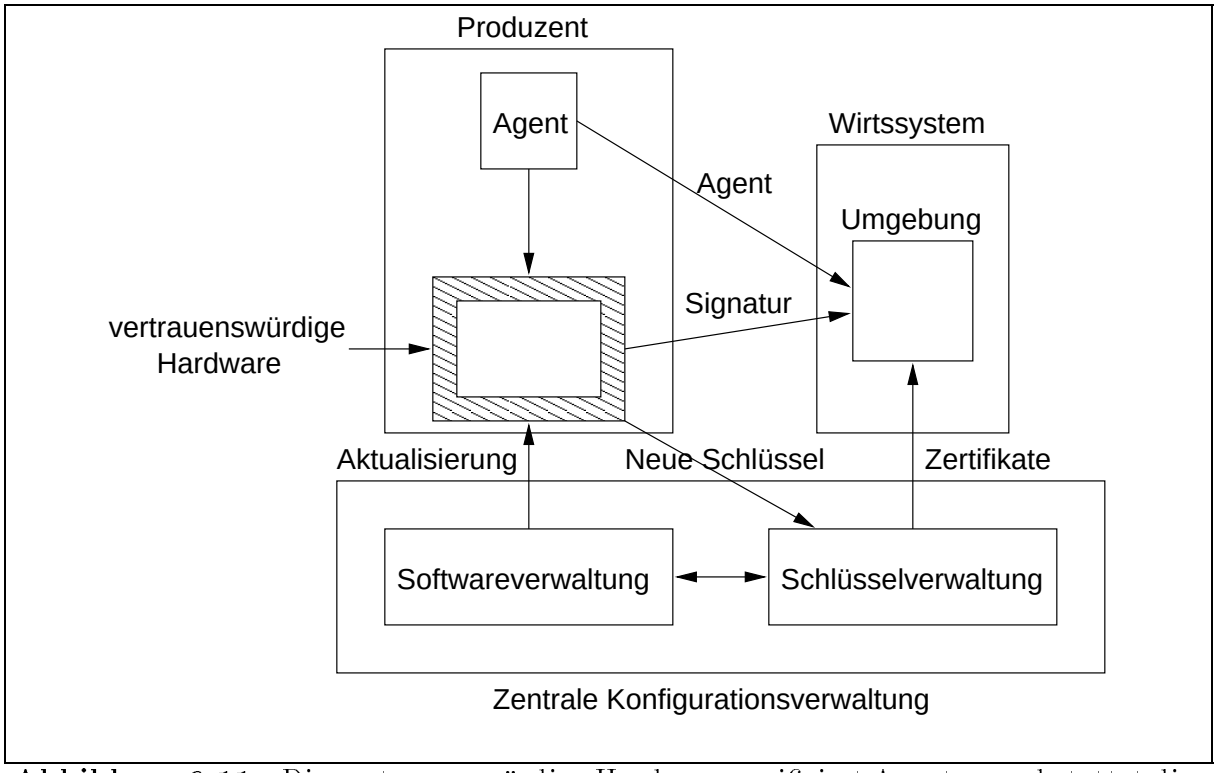

Abbildung 6.11.: Die vertrauenswürdige Hardware verifiziert Agenten und stattet diese mit einer Signatur aus. Im Falle von Sicherheitslücken werden Zertifikate zurückgezogen, so daß die korrespondierenden Schlüssel ihre Gültigkeit verlieren. Mit der Aktualisierung der Software ist gleichzeitig ein neuer Schlüssel verbunden (nach [DFS98]).

werden. Der Agent gelangt nicht zur Ausführung. Der Produzent muß sich nachfolgend um eine neue Version der Verifikationssoftware bemühen, die den entdeckten Sicherheitsfehler nicht länger aufweist.

Werden Fehler in der sicherheitsrelevanten Software des Wirtssystems ermittelt, wird ebenfalls eine neue Version der vertrauenswürdigen Software für den Produzenten bereitgestellt. Alle Wirtssysteme, deren sicherheitsrelevante Software den entdeckten Fehler aufweisen werden aufgefordert, zukünftig lediglich Agenten zu akzeptieren, deren Verifikation die Sicherheitslücke in den Verifikationsvorgang einbezogen hat.

DEVANBU et al. sehen ihre Entwicklung als Möglichkeit, um die Sicherheit von auf Java basierenden Agentensystemen zu erhöhen. Der Verifier Javas übernimmt dabei die Rolle der vertrauenswürdigen Software und ist damit lediglich noch auf den Systemen der Produzenten vorhanden. Der Java Verifier übernimmt die bekannten Überprüfungen (siehe Abschnitt 5.3.1). Hält der Agent den Überprüfungen stand, wird er digital signiert und kann anschließend auf Wirtssysteme migrieren. Ein Wirtssystem kann anhand der Signatur erkennen, ob der Agent den Verifier erfolgreich passiert hat. Somit kann der Agent auf dem lokalen System zur Ausführung gelangen. Entsprechend dem oben vorge-

stellten Verfahren, werden Sicherheitslücken durch die Rücknahme von Schlüsseln und durch die Veröffentlichung neuer Schlüssel geschlossen.

Ein Einsatz von Proof-Carrying Code (PCC)[11] ist ebenfalls denkbar. Das Beweisverifikationsverfahren befindet sich dabei innerhalb der vertrauenswürdigen Hardware. So der Beweis korrekt ist, wird der Agent zusammen mit den Verifikationsbedingungen signiert. Das Wirtssystem, daß einen Agenten empfängt, überprüft die Signatur und stellt anschließend fest, ob die Verifikationsbedingungen den lokalen Sicherheitsanforderungen entsprechen. Vorteil des Verfahrens ist die Vermeidung zusätzlicher Last auf den Wirtssystemen, da die Beweise bereits auf den Quellsystemen verifiziert werden. Für den Produzenten ergibt sich der Vorteil, daß das Wirtssystem den Beweis nicht erhält und somit auch kein detaillierter Einblick in den Agenten möglich ist.

6.10.3. Beurteilung

Die vorgestellten Architekturen lösen die Sicherheitsprobleme der Agentensysteme durch den Einsatz vertrauenswürdiger Hardware. Dabei wird in der Architektur von Wilhelm die Hardware auf Seiten des Umgebungssystems angesiedelt, während Devanbu et al. die vertrauenswürdige Hardware auf Seiten des Agenten-Produzenten ansiedeln.

Wilhelm kann mit seiner Lösung sicherstellen, daß der Betreiber einer Umgebung für Agenten diese nicht manipulieren kann und auch keinen Einblick in den Code und die Daten erhält. Eine vertrauliche Kommunikation mit anderen Agenten innerhalb und außerhalb der TPE ist ebenfalls möglich. Agenten können öffentliche Schlüssel mit sich führen und diese zur Verschlüsselung oder zur digitalen Signierung von Dokumenten nutzen.

Devanbu et al. adressieren mit ihrer Architektur den Schutz der Umgebungssysteme vor Agenten. Sicherheitslücken innerhalb eines Agentensystems müssen in dieser Architektur nicht zwingend durch die Einspielung von aktualisierten Programmversionen auf Seiten der Wirtssysteme geschlossen werden. Stattdessen wird sichergestellt, daß Agenten Sicherheitslücken nicht ausnutzen, die entdeckt worden sind.

Devanbu et al. führen aus, daß der Beweis nicht enthüllt wird. Ein Mehrwert ergibt sich jedoch nur dann, wenn auch der Code des Agenten nicht enthüllt wird. Die Mehrheit der heutigen Agentensysteme basieren jedoch auf Java. Agenten können somit leicht wieder in Quelltext überführt werden. Agenten in nativer Form sind allerdings ebensowenig vor Reengineering geschützt, der Aufwand ist jedoch höher als im Falle von Java.

Beide Lösungen stellen im wesentlichen geschlossene Systeme dar, da die Existenz vertrauenswürdiger Hardware erforderlich ist. Derzeit existieren vertrauenswürdige Hardwarekomponenten lediglich in sensitiven Anwendungsbereichen. Der Einsatz von vertrauenswürdiger Hardware in Agentensystemen stellt derzeit einen Faktor dar, der die Offenheit der Systeme beschränkt, da nicht ausreichend Komponenten verfügbar sind.

[11] Näheres zu PCC erfolgt im nächsten Abschnitt.

Desweiteren existieren keine anerkannten, vertrauenswürdigen Hersteller solcher Komponenten, denen die Teilnehmer von Agentensystemen vertrauen könnten. Ebenso wie in allen anderen Systemen, in denen Vertrauen erforderlich ist, stellt sich die Frage, auf welcher Basis das Vertrauen gegenüber den Herstellern derartiger Hardware basieren sollte.

6.11. Proof-Carrying Code

6.11.1. Konzept

Proof-Carrying Code (PCC) ist ein Mechanismus, der die Konstruktion überprüfbarer mathematischer Beweise von Programmeigenschaften erlauben soll, als auch die formale Spezifikation des Verhaltens sicherheitsrelevanter Eigenschaften ermöglichen soll [FL97]. Damit lassen sich Garantien über das wesentliche Verhalten mobiler Agenten etablieren lassen [LN97].

Der formale Beweis soll sicherstellen, daß der Agent der Sicherheitspolitik entspricht. Damit soll einhergehen, daß keine Sicherheitsprobleme aufgrund mangelhafter Implementationen der Sicherheitspolitik auftreten können. Probleme, die aufgrund von Implementationsfehlern des Bytecode Verifiers, Class-Loaders oder des Security-Managers auftreten, würden damit nicht mehr in Erscheinung treten. Die angeführten Komponenten würden durch eine zentrale Komponente, den Typprüfer für Logic Framework (LF), ersetzt werden [FL97]. Die Kodierung des Beweises erfolgt in einer Sprache, die auf dem Lambda-Kalkül basiert. Die Validierung des Beweises wird auf die Typüberprüfung in LF abgebildet. Dazu werden Beweise als Ausdrücke und Prädikate aus Typen repräsentiert. Ein Vorteil des Verfahrens ist die Unabhängigkeit des Typüberprüfers von der Sicherheitspolitik. Es muß allerdings eine Übereinkunft zwischen dem Code-Produzenten und dem Wirtssystem über die Logik geben, in der gewünschte Eigenschaften bewiesen werden. Die Logik sollte eine standardisierte Form für die Beweise aufweisen, so daß diese durch das Wirtssystem überprüft werden kann [GHN97]. Es ist Aufgabe des Code-Produzenten, formal zu beweisen, daß ein Programm den vom Wirt geforderten Sicherheitsanforderungen entspricht.

Die Verifikation des Beweises erfolgt statisch vor Ausführung des Codes, so daß Sicherheitsprobleme frühzeitig erkannt werden können. Eine dynamische Verifikation stellt nicht sicher, daß ein Programm ein korrektes Ergebnis produzieren wird. Dynamische Verifikation besagt lediglich, daß bislang kein falsches Ergebnis produziert worden ist. Desweiteren muß der Verifikationsprozeß sehr effizient sein, ansonsten erfolgt eine Degradierung der Performanz. Die Überprüfung eines formalen Beweises ist hingegen relativ einfach, seine Erzeugung ist hingegen sehr aufwendig.

Die Ausführung nicht vertrauenswürdigen Codes birgt einen großen Overhead in sich, wenn die Verifikation dynamisch erfolgen muß. Im Falle von PCC wird der Overhead auf die Seite des Produzenten verlagert [GHN97]. Der Empfänger eines Agenten überprüft

lediglich, ob die vom Agenten eingehaltenen Sicherheitsanforderungen mit den lokalen Sicherheitsanforderungen korrespondieren. Der Beweis, den der Agent mit sich führt, ist die einzige Entscheidungsgrundlage für die Ausführung des Agenten. Keinen Einfluß hat das Quellsystem und der Entwickler des Agenten und auch die bislang vom Agenten besuchten Umgebungen spielen im Entscheidungsprozeß keine Rolle. Agenten sind damit „selbst-zertifizierend" [LN97].

Im Falle des PCC ist nur eine vertrauenswürdige Komponente notwendig – der Algorithmus zur Überprüfung der Beweise. Alle anderen Komponenten müssen nicht vertrauenswürdig sein. Etwaige Manipulationen werden durch den Verifikationsalgorithmus erkannt. Erfolgt eine Manipulation, so daß der Beweis nicht länger gültig ist, wird der Agent abgelehnt. Gleiches gilt, wenn der Beweis durch die Manipulation weiterhin gültig bleibt, aber nicht mehr der lokalen Sicherheitspolitik genügt. Eine Manipulation kann darüberhinaus zu einem ungültigem Beweis führen, der nicht interpretiert werden kann. Auch dann wird der Agent abgelehnt. Genügt der Beweis weiterhin der lokalen Sicherheitspolitik, kann auch ein manipulierter Agent ausgeführt werden [LN97].

PCC gestattet dem Wirtssystem, das einen Agenten ausführen will, daß der Agent eines nicht vertrauenswürdigen Produzenten den lokalen Sicherheitsanforderungen genügt. Die Sicherheitsanforderungen sind durch das Wirtsystem derart gewählt, daß Agenten die Sicherheit des Systems nicht kompromittieren können.

Jede Implementation eines PCC muß zumindest vier Elemente beinhalten:

- ▷ Eine formale Sprache zur Beschreibung der Sicherheitspolitik;
- ▷ eine formale Semantik der Sprache, die zur Implementierung des nicht vertrauenswürdigen Codes verwendet wird;
- ▷ eine Sprache, in der die Beweise ausgedrückt werden und
- ▷ einen Algorithmus zur Validierung der Beweise [LN97].

Die Art und Weise, in die ein Beweis erzeugt wird – manuell, durch einen zertifizierenden Compiler[12] oder einen Theorembeweiser – ist völlig unerheblich für das Wirtssystem. Das Wirtssystem muß der Beweisgenerierung nicht einmal Vertrauen entgegenbringen.

6.11.2. Beurteilung

PCC dient dem Schutz des Wirtssystems vor maliziösen Agenten. Hierzu wird anhand der Verifikation eines Beweises festgestellt, ob ein Agent den lokalen Sicherheitsanforderungen entspricht. So der Verifikationsalgorithmus korrekt implementiert worden ist und die lokalen Sicherheitsanforderungen der Anwendung gerecht werden, schützt PCC die Umgebung vor Angriffen durch Agenten. PCC ist proaktiv, die Überprüfung auf

[12] Lee und Necula sehen zukünftig die Möglichkeit, daß zertifizierende Compiler als Teil des Übersetzungsprozesses den Beweis generieren [LN97].

Einhaltung der Sicherheitsanforderungen erfolgt vor Ausführung des Agenten, so daß anschließend sichergestellt sein sollte, daß vom Agenten keine maliziösen Aktionen ausgehen können.

Die automatische Generierung der Beweise ist derzeit eines der größten Probleme von PCC-Systemen. Es ist unrealistisch anzunehmen, eine manuelle Erstellung der Beweise würde der Situation in Agentensystemen gerecht werden. LEE und NECULA erwarten die Entwicklung eines zertifizierenden Compilers, der gleichzeitig mit der Codegenerierung den zugehörigen Beweis erstellt. Bislang existieren derartige Compiler jedoch nicht.

Ein weiteres fundamentales Problem ist die Axiomatisierung der Arithmetik. Es gilt festzustellen, welche Axiome für die Durchführung des Beweises und dessen Validierung erforderlich sind. Produzent und Konsument müssen über die Gültigkeit der gewählten Axiome übereinstimmen. Somit wird es erforderlich sein, daß das Wirtssystem die Gültigkeit der Aussagen über die Arithmetik im Beweis verifiziert. Forschungsgegenstand sind die Natur der Aussagen, ihr Einfluß auf die Ausdruckskraft der Beweise, die Größe der Beweise und die Dauer der Beweisverifikation. Weitergehende Entwicklungen, Analysen und Experimente sind hier erforderlich [LN97].

Schließlich lassen sich laut RUBIN nicht alle Eigenschaften bezüglich des Informationsflusses und der Vertraulichkeit durch PCC auf ihre Einhaltung verifizieren [RDEG98]. Damit sind über PCC weitere Mechanismen erforderlich, um den Schutz des Wirtssystemes vor den Agenten zu gewährleisten.

6.12. Tracing

6.12.1. Konzept

Ziel ist der Schutz der Agenten vor der Umgebung. Jegliches Fehlverhalten des Umgebungssystems soll durch den Einsatz kryptografischer Spuren entdeckt werden. Die Spur enthält die Informationen über die Operationen, die ein Agent während seiner Lebenszeit durchgeführt hat [Vig97]. Nach Beendigung der Ausführung eines Agenten soll der Besitzer anhand der Spur feststellen können, ob diese mit einer korrekten Ausführung korrespondiert.

Jegliche unautorisierten Modifikationen des Agentencodes und -zustands sollen identifiziert werden. Damit einher geht bereits, daß Modifikationen durch die Umgebung nicht verhindert werden können. Es könnte sich lediglich sicherstellen lassen, daß die Modifikationen im nachhinein erkannt werden könnte. Tracing ist damit ein weiteres reaktives Verfahren.

VIGNA geht für seine Sicherheitsarchitektur von folgenden Annahmen aus:

- Es existiert eine PKI;
- Der Code des Agenten ist statisch;

▷ Jede Entität besitzt ein Schlüsselpaar bestehend aus öffentlichem und privatem Schlüssel;

▷ Die Implementationen der Interpreter sind dahingehend zertifiziert, daß sie korrekt sind und der Semantik der Sprache entsprechen [Vig97].

Unterschieden werden zwei Kategorien von Instruktionen. *Weiße* Instruktionen verändern den Zustand eines Programmes lediglich auf Basis der programminternen Variablen. *Schwarze* Instruktionen führen dagegen zu Änderungen des Programmzustandes durch Informationen, die der ausführenden Umgebung entnommen sind [Vig97].

Die Ausführungsspur T_C eines Programmes C besteht aus eine Menge von Paaren $< n, s >$, wobei n Instruktionen eindeutig identifiziert und s den Wert interner Variablen für schwarze Instruktionen enthält. Im Falle weißer Instruktionen ist s leer. Im Falle der

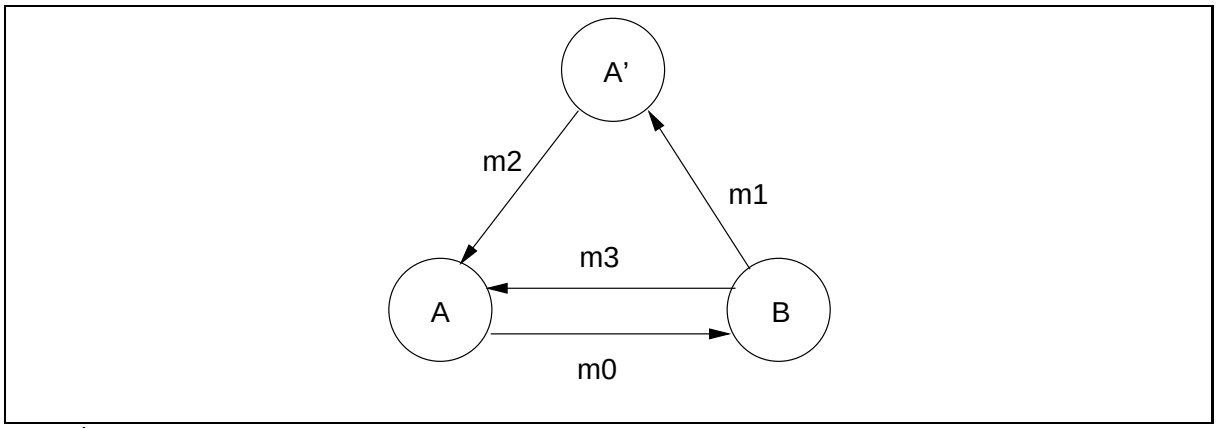

Abbildung 6.12.: Der Besitzer A eines Agenten übertragt diesen zu einem System B mit der Nachricht m_0. Hier wird der Agent ausgeführt und die Aussführungsspur m_1 wird an ein weiteres System übertragen. Das Ergebnis m_3 sendet das System B an das Quellsystem des Agenten. Vermutet A einen Betrug, fordert er von dem System A' die Ausführungsspur an, die im in Form der Nachricht m_2 übersandt wird.

entfernten Ausführung eines Agenten C des Benutzers A auf dem Wirtssystem B wird die folgende Nachricht an B versandt:

$$A \xrightarrow{m_0} B : A, B_p(C), A_s(H(C), B, A', t_A, i_A)$$

Enthalten in der Nachricht ist die Identität des Senders A, der Code des Agenten $B_p(C)$, der mit dem öffentlichen Schlüssel des empfangenden Wirtssystemes verschlüsselt ist. Der dritte Teil der Nachricht, der mit dem privaten Schlüssel des Quellsystems A_s verschlüsselt ist, enthält einen Hashwert des Agenten $H(C)$, der durch das Wirtssystem überprüft wird. Daneben ist die Identität des Empfängers, die Identität des Empfangssystems für den Trace, ein Zeitstempel und ein eindeutiger Bezeichner zum Schutz vor

Wiedereinspielungsangriffen enthalten. Nachdem B anhand dessen die Nachricht authentifiziert hat, sendet er an den ausgewiesenen Empfänger des Traces die folgende Nachricht:

$$B \xrightarrow{m_1} A' : B, B_s(A, A_s(H(C), B, A', t_A, i_A))$$

Aufgrund dieser Nachricht kann A durch Übersendung des Hashwertes $H(C)$ nachweisen, daß B die Ausgangsnachricht m_0 erhalten hat. Nachfolgend führt B den Agenten C aus und produziert dabei den Trace T_C. Nach Beendigung der Ausführung sendet B einen Hashwert $H(S^1)$ des finalen Zustandes des Agenten und einen Hashwert $H(T_C)$ des Traces an A':

$$B \xrightarrow{m_2} A' : B, B_s(H(S^1), H(T_C), i_A)$$

Das Ergebnis der Ausführung des Agenten wird schließlich in verschlüsselter Form an A versandt:

$$B \xrightarrow{m_3} A : B, B_s(A_p(S^1), i_A)$$

Wenn A einen Betrug durch B vermutet, überprüft A das Ergebnis S^1, indem A von B den Trace T_C und von A' die Nachricht m_2 anfordert. Mittels des Trace wird die Ausführung des Agenten nachvollzogen. Wenn das Ergebnis der Ausführung von dem von B gelieferten Ergebnis S^1 abweicht, hat B betrogen. Die Komplexität des Validationsprozesses wächst linear mit dem Trace. Eine dritte vertrauenswürdige Instanz A' ist nur dann notwendig, wenn A nicht ständig mit dem Netzwerk verbunden ist.

VIGNA hat für das von ihm entwickelte Verfahren eine Sprache namens Secure Agent Language with Tracing of Actions (SALTA) entwickelt. SALTA stellt ein Untermenge der von BORENSTEIN entwickelten Sprache Safe-Tcl dar. Neben den durch Safe-Tcl eingeführten Restriktionen wurde sie um die Kommandos `request`, `service`, `reply` und `go` erweitert.

Ein wesentliches Problem des Tracing-Verfahrens ist die Größe der Ausführungsspur, die während der Ausführung eines Agenten erzeugt wird. YEE hat zwei Verfahren entwickelt, die es ermöglichen, den Anteil der Ausführungsspur, der übertragen werden muß, zu begrenzen [Yee97].

Das erste – als „Holographic Proof“ bezeichnete – Verfahren beruht auf dem *Zero-Knowledge Proof*. Eine Funktion $p(x, y)$ ergibt 1, wenn der Trace y zu einem Programm x korrekt ist, d. h., x wurde ausgeführt. Statt den gesamten Trace y an das Quellsystem zu versenden erlaubt ein holographischer Beweis y', daß mittels nur weniger Bits $p(x, y) = 1$ überprüft werden müssen [Yee97]. Jedoch wird trotzdem ein vollständiger Zugriff auf y' benötigt, so daß auch hier keine Bandbreite gespart wird.

Statt des holographischen Beweises kann ein „Computational Sound Proof“ Anwendung finden. In diesem Verfahren wird ein Hashwert über einem holographischen Beweis gebildet. Die Wurzel des baumförmigen Hash-Schemas wird an das Quellsystem versandt. Dieses wendet die Hashfunktion auf die Wurzel an. Nachfolgend werden Anfragen an die ausführende Umgebung des Agenten gestellt, anhand derer sichergestellt werden kann, daß der Agent ausgeführt worden ist. Derzeit werden für holographische Beweise NP-vollständige Verfahren benötigt.

6.12.2. Beurteilung

Das von VIGNA entwickelte Verfahren versucht, die korrekte Ausführung der Agenten in fremden Umgebungen nachweisbar werden zu lassen. Dazu wird ein Trace erzeugt, der die durchgeführten Operationen des Agenten wiederspiegeln soll. Der Besitzer des Agenten soll anhand der Ausführungsspur ermitteln können, ob der Agent korrekt ausgeführt worden ist.

Voraussetzung ist, daß mehrere Agenten keinen gemeinsamen Speicher nutzen und nur einen Ausführungspunkt besitzen. Desweiteren wird angenommen, daß der Code statisch ist. Das dynamische Hinzuladen von Bibliotheken kann zu Problemen führen, wenn dadurch unterschiedliche Programmzustände erreicht werden. Unterschiedliche Versionen auf dem Quell- und dem Wirtssystem veranlassen das Quellsystem zu der Annahme, daß der Agent nicht korrekt ausgeführt worden ist.

Weiterhin geht VIGNA von der Existenz zertifizierter Interpreter aus. Das Beispiel der Sprache Java zeigt nur zu deutlich, daß es ohne Einsatz von Werkzeugen aus der theoretischer Informatik nicht möglich ist, einen sicheren Interpreter zu entwickeln, der sich zertifizieren ließe. Aus eben diesem Grunde existiert bislang kein weitverbreiteter Interpreter, der sich durch eine vertrauenswürdige Instanz zertifizieren ließe.

Die von VIGNA angeführte Lösung erfordert das Vorhandensein einer funktionierende Infrastruktur für öffentliche Schlüssel. Es wurde bereits in den vorhergehenden Abschnitten darauf hingewiesen, daß eine entsprechende Infrastruktur nicht vorhanden ist.

Abschließend ist die Frage aufzuwerfen, ob das Quellsystem in der Lage ist, jeden Schritt, den der Agent auf einem entfernten System vollführt hat, auf dem lokalen System zu reproduzieren. Neben den Parametern, die der Agent auf dem entfernten System erhält, existieren weitere Aspekte, die die Ausführung eines Agenten beeinflussen. Hierzu gehören beispielsweise instabile Zustände auf dem entfernten System.

Die Größe des Traces stellt ebenfalls ein Problem dar. Die Reduktion auf Änderungen des Kontrollflusses und den schwarzen Instruktionen bildet eine Lösung zur Reduzierung der Tracegröße. Darüberhinaus kann der Trace auf einige wenige Instruktionen beschränkt werden. Es stellt sich jedoch die Frage, wie diese spezifischen Instruktionen identifiziert werden und wer diese Identifikation vollzieht. Erfolgt diese unabhängig vom Anwendungskontext können wichtige Instruktionen entfallen.

Auch wenn sich mit den angeführten Ausführungsspuren sicherstellen läßt, daß der Agent einmal korrekt ausgeführt worden ist, ist nicht gleichzeitig gewährleistet, daß der Agent einmalig ausgeführt wird. Es läßt sich nicht verhindern, daß der Agent ein weiteres Mal ausgeführt wird und anschließend mit den dabei erzielten Ergebnissen migriert.

6.13. Type Hiding

6.13.1. Konzept

Entsprechend dem Ansatz, wie ihn BORENSTEIN bereits mit Safe-Tcl verfolgt hat,[13] haben FELTEN et al. ein Sicherheitskonzept für Java entwickelt, das auf der Tatsache beruht, daß gefährliche Klassen entfernt bzw. durch sichere ersetzt werden [WBDF97].[14]

Die Sicherheitspolitik ergibt sich in diesem Verfahren aus der Art und Weise, in der Klassen-Namen zur Laufzeit in Entitäten der Laufzeit aufgelöst werden. Die Auflösung der Klassen-Namen kann lokal beeinflußt werden, indem „gefährliche" Klassen der Standard-Bibliothek durch ungefährliche ersetzt bzw. komplett entfernt werden.

Sensitive Klassen werden durch kompatible Klassen ersetzt. Diese überprüfen die ihnen übergebenen Argumente und rufen gegebenenfalls die originale Klasse auf. Ein Beispiel hierfür ist der Zugriff auf das lokale Dateisystem. Durch die Ersetzung der sensitiven Klasse, die Zugriffe auf das Dateisystem implementiert, kann der Zugriff eines Agenten im lokalen Dateisystem auf ein bestimmtes Verzeichnis oder eine einzige Datei beschränkt werden.

Die Menge der Abbildungen $Name \rightarrow Klasse$ bildet eine Konfiguration. Eine Policy-Engine entscheidet für jeden Agenten, welche Konfiguration zu wählen ist. Konfigurationen müssen alle Klassen beinhalten, die als sensitiv angesehen werden. Wird eine sensitive Klassen ausgelassen, können Sicherheitsprobleme die Folge sein. Die Entscheidung der Policy-Engine basiert auf dem „Ruf des Besitzers" des Agenten. So der Besitzer der Engine nicht bekannt ist, wird der Anwender für eine Entscheidung konsultiert. Der Mechanismus erlaubt es, alle sicherheitsrelevanten Entscheidungen vor Ausführung eines Agenten zu treffen [WBDF97].

Type Hiding läßt sich in der Sprache Java durch eine Modifikation des Class-Loaders erreichen. Der Class-Loader sorgt für die Abbildung des Klassennamens auf die Implementation der Klasse. Die Referenzierung einer Klasse kann den Class-Loader zu drei unterschiedlichen Aktionen veranlassen:

- ▷ er generiert eine Exception, da die Entität keinerlei Zugriff auf die Klasse erhält,
- ▷ liefert eine Instanz der geforderten Klasse, da die Entität vollen Zugriff auf diese hat, oder
- ▷ liefert eine Instanz einer Subklasse, die in der Konfiguration der Entität spezifiziert ist [WBDF97].

WALLACH et al. haben eine Referenzimplementation auf Basis des Microsoft Internet Explorers erstellt. Die Implementation umfaßt 4500 Zeilen Javacode, wobei ein wesentlicher Teil die Benutzungsschnittstelle ausmacht.

[13] Siehe hierzu [Mar97d], und [Bor94].

[14] Kurz vor Ende dieser Arbeit hat WALLACH seine Doktorarbeit abgeschlossen, die das hier vorgestellte Verfahren vertieft und weitergehend bearbeitet [Wal99].

6.13.2. Beurteilung

Das von WALLACH et al. entwickelte Konzept dient als Ersatz für die derzeitige Sicherheitsarchitektur Javas. Das von Sun entwickelte Konzept basiert auf dem Typsystem der Sprache. WALLACH et al. verfolgen eine Lösung, die dem von BORENSTEIN entwickelten Konzept des „Versteckens“ gefährlicher Klassen entspricht.

Type Hiding kann ausschließlich dazu genutzt werden, die Sicherheitspolitik eines Wirtssystems gegenüber Agenten durchzusetzen. WALLACH et al. weisen in ihrer Arbeit darauf hin, daß die Sicherheit eines Wirtssystems nur dann sichergestellt werden kann, wenn alle „gefährlichen“ Klassen identifiziert und durch kompatible Klasse ersetzt worden sind. Offen ist, welche Klassen als „gefährlich“ bewertet werden müssen. Das Beispiel der Denial-of-Service-Angriffe durch Applets zeigt, daß eine eindeutige Entscheidung nicht realisierbar ist. Damit hängt die Sicherheit des Wirtssystems von der Urteilskraft des Anwenders ab, der zu bewerten hat, welche Konfiguration für ankommende Agenten anzuwenden ist. Es wurde bereits mehrfach ausgeführt, daß die Verlagerung der Sicherheitsentscheidungen auf den Anwender nicht akzeptabel ist, so dieser nicht über das entsprechende Fachwissen verfügt. Auch im Falle des Type Hiding ist eine Entscheidung des Nutzers für jeden Agenten erforderlich, der dem System bislang nicht bekannt ist. Auf welcher Grundlage diese Entscheidung zu treffen ist, führen WALLACH et al. in ihrer Arbeit nicht aus.

Ein weiteres Problem kann in der Weitergabe von Rechten unter dem Type Hiding-Ansatz bestehen. Hat ein Agent Zugriff auf das Netzwerk aber nicht auf das lokale Dateisystem, kann ihm eine Referenz auf eine Datei durch einen anderen Agenten übergeben werden. Die Klasse `FileInputStream` wird ebenso von `InputStream` abgeleitet wie die Klassen, die den Netzwerk-Code implementieren. Wird einem Agenten der Zugriff auf `InputStream` verwehrt, kann dieser auch keine Netzwerkverbindungen aufbauen. Das beschriebene Problem existiert für alle Klassen innerhalb der Standard-Bibliothek Javas, die gemeinsame Klassen nutzen [WBDF97].

Einem Agenten erteilte Rechte können diesem zur Laufzeit nicht entzogen werden, ohne daß alle Klasse entladen werden, die im gleichen Namensraum instantiiert wurden.

6.14. Zusammenfassung

Keine der vorgestellten Sicherheitsarchitekturen können allen Sicherheitsanforderungen entsprechen. In Anbetracht der sich widersprechenden Ziele, wie dem Schutz des Agentensystems vor maliziösen Agenten und dem Schutz der Agenten vor maliziösen Umgebungen, kann dieses auch von keiner Sicherheitsarchitektur geleistet werden. Es hat sich weiterhin herausgestellt, daß die von FARMER et al. erzielten Ergebnisse, die bereits im Kapitel 2 vorgestellt worden sind, weiterhin Bestand haben. Den vorgestellten Sicherheitsmodellen ist es nicht gelungen, jene Sicherheitsziele umzusetzen, die von FARMER et al. als nicht erreichbar definiert worden sind.

Die vorherigen Abschnitte haben weiterhin deutlich werden lassen, daß eine integrierte Sicherheitsarchitektur erforderlich ist, die den gewünschten Sicherheitsanforderungen gerecht wird. Eine integrierte Lösung setzt sich aus unterschiedlichen Komponenten zusammen, die jeweils einen bestimmten Bereich der Sicherheitsanforderungen abdecken. Hier wird klar, daß auch Komponenten eingesetzt werden können, die nicht spezifisch in Hinblick auf Agentensysteme entwickelt worden sind, wie beispielsweise digitale Signaturen, Firewalls oder Kerberos.

Das Paradigma des Agenten erfordert darüberhinaus aber spezifische Sicherheitskomponenten, die den Gefahren begegnen, die aus der neuen Technik resultieren. Es ist deutlich geworden, daß insbesondere der Schutz des Agenten vor dem Wirtssystem eine neue Herausforderung für die Computersicherheit ist, deren abschließende Lösung noch nicht absehbar ist. Die vorgestellten Architekturen, die dem Schutz des Agenten dienen sollen, existieren bislang ausschließlich als theoretische Modelle. Keines der im Kapitel 3 vorgestellten Agentensysteme setzt bislang eine Sicherheitsarchitektur ein, die den Schutz des Agenten vor dem Wirtssystem vorsieht.

Es ist weiterhin deutlich geworden, daß alle vorgestellten Sicherheitsarchitekturen Mängel aufweisen, die beachtet werden müssen, so eine integrierte Sicherheitsarchitektur für Agentensysteme zu konzipieren ist. Wesentlich ist bei der Konzeption des Agentensystemes die Anforderungen der Anwendung. Das nachfolgende Kapitel soll mögliche Lösungen andeuten, ohne dabei abschließende Lösungen zu bieten.

7. Sicherheitsarchitekturen für unterschiedliche Anwendungen

„*Java is secure to survive in the network-based environment.*"
– Gosling & McGilton

Die rasante Entwicklung der Agentensysteme ist nicht zuletzt dadurch zu erklären, daß sich eine Vielzahl unterschiedlicher Anwendungsgebiete Vorteile von deren Einsatz versprechen. In den elektronischen Dienstemärkten werden Agenten als Vermittler zwischen Anbieter und Nachfrager gesehen. Agenten können die Arbeit mit verteilten Gruppen unterstützen und damit die Groupware-Produkte bereichern. Die mit dem „Informationszeitalter" einhergehenden Informationsflut kann durch den Einsatz von intelligenten Agenten beherrscht werden. Intelligente Agenten sind in der Lage, die wesentlichen Informationen zu erkennen und weiterzuleiten. Der Einsatz von mobilen Geräten nimmt ständig zu. Der Nutzwert der Geräte läßt sich wesentlich steigern, wenn Aufgaben in ein Netzwerk verlagert werden können, mit dem man nur temporär verbunden ist. Schließlich können Agentensysteme in internen Netzwerken zum Einsatz gelangen, um dort Aufgaben aus dem Bereich des System-Managements wahrzunehmen.

Die angeführten Anwendungsfelder haben alle spezifische Sicherheitsanforderungen, denen es gerecht zu werden gilt. Bislang kommen Agentensysteme vielfach zum Einsatz ohne daß sie in der Lage sind, die spezifischen Sicherheitsanforderungen zu erfüllen. Zukünftig wird dieser Zustand nicht länger akzeptabel sein. Auch wenn die Zahl der Angriffe derzeit gering ist, ist davon ausgegangen, daß die Existenz realer Anwendung Angriffe nach sich zieht, die bislang nicht aufgetreten sind.

Eine durch die Wheelgroup erhobene Studie gibt Auskunft über die Gefahren, denen sich Organisationen, Unternehmen und Privatanwender aussetzen, wenn sie im Internet arbeiten. Zusammengefaßt kam man zu folgendem Ergebnis:

Frequenz der Angriffe Die Spanne der Angriffshäufigkeit reicht von 0.5 bis 5.0 Angriffe pro Monat und pro Unternehmen, wobei Firmen, die sich im Bereich elektronischen Dienstemärkte betätigen, wesentlich stärker betroffen sind.

Angreifertypus Angriffe im Internet werden im zunehmenden Maße von Personen unternommen, die nur einen Bruchteil des Wissens haben, das für entsprechende

Angriffe benötigt wird. Dieser Schluß läßt sich aus dem Umstand ziehen, daß Angriffe, die neu entdeckte Sicherheitslücken ausnutzen, gehäuft zwei bis drei Monate nach der ersten Publikation der Sicherheitslücke auftreten. Zu diesem Zeitpunkt ist die Ausführung entsprechender Angriffe ausgiebig dargestellt und häufig durch die Bereitstellung geeigneter Tools nahezu für jeden Anwender durchführbar.

Quelle der Angriffe Die Studie offenbarte, daß 48 % der Angriffe von ISP´s ausgingen, woraus der Schluß zu ziehen ist, daß der Großteil der Angriffe von Privatpersonen oder kleinen Unternehmen ausgehen.

Web Ziele Alle Angriffe, die sich gegen Web-Angebote richteten, richteten sich gegen Anbieter kommerzieller Angebote [Whe97].

Es gibt keine allgemeinen Sicherheitsanforderungen, die für alle Anwendungen gültig sind. Der Einsatz eines Agentensystems in einer Anwendung erfordert es somit, die spezifischen Sicherheitsanforderungen der Anwendung zu identifizieren und diese umzusetzen. Aus den Erkenntnissen des vorangegangenen Kapitels ergibt sich, daß die Entwicklung einer integrierten Sicherheitsarchitektur hierfür erforderlich ist. Die nachfolgenden Abschnitte werden zeigen, daß die bereits vorgestellten Sicherheitsarchitekturen jeweils nur Teilsaspekte der Sicherheitsanforderungen erfüllen können.

Nachfolgend werden die Anwendungsgebiete „elektronische Dienstemärkte“ und „System-Management“ dahingehend untersucht, welche spezifischen Sicherheitsanforderungen in den Anwendungen existent sind. Im Hinblick auf die Ergebnisse wird eine Sicherheitsarchitektur skizziert, die versucht, den spezifischen Anforderungen gerecht zu werden.

7.1. Elektronische Dienstemärkte

Der Einsatz von Agenten wird in unterschiedlichen Bereichen vorangetrieben. Das Gebiet der elektronischen Dienstemärkte ist derzeit jener Bereich, der sich am meisten von dem Durchbruch der Agententechnologie verspricht. Bislang ist die Entwicklung der elektronischen Dienstemärkte hinter den gestellten Erwartungen zurückgeblieben. Der Einsatz von Agenten soll helfen, daß gesteckte Ziel zu erreichen.

7.1.1. Inhalt

Das Gebiet der elektronischen Dienstemärkte besteht im wesentlichen aus zwei Gruppen – Anbietern und Nachfragern – zwischen denen Leistungen ausgetauscht werden. Der Anbieter offeriert Leistungen, die durch den Nachfrager in Anspruch genommen werden können. Für die Inanspruchnahme einer Leistung ist durch den Nachfrager eine Gegenleistung zu erbringen. Der Ablauf eines Prozesses kann durch die Phasen „Angebot“, „Verhandlung“ und „Diensterbringung“ beschrieben werden.

Angebot In der Phase „Angebot“ versucht der Nachfrager aus der Menge der Angebote jenes zu bestimmen, das seinen Vorstellungen am besten entspricht. Hierzu migriert ein Agent vom System eines Nachfragers in die Umgebungen unterschiedlicher Anbieter. In den Umgebungen der Anbieter ermittelt der Agent, basierend auf die durch den Nachfrager gemachten Vorgaben, daß beste Angebot des jeweiligen Anbieters. Im Rahmen seiner Reise besucht der Agent eine Reihe von Umgebungen und kehrt danach auf das Ausgangssystem zurück. Als Ergebnis präsentiert er das beste Angebot aus der Menge der Angebote, das er während seiner Migration ermittelt hat.

Verhandlung Die Verhandlungsphase baut auf den Ergebnissen der Angebotsphase auf. Nachdem aus der Menge der Anbieter jener mit dem bestem Angebot bestimmt wurde, bedarf es der Aushandlung des Vertrages. Zu diesem Zeitpunkt migriert der Agent erneut auf das System des Anbieters mit besten Angebot, um mit diesem in Verhandlungen zu treten. Das Ergebnis der Verhandlungen ist ein Vertrag zwischen dem Nachfrager und dem Anbieter. Der Vertrag beschreibt die Leistungen beider Entitäten, die zur Erfüllung der Transaktionen zu erbringen sind.

Diensterbringung In der Phase der Diensterbringung erfolgt zwischen den beiden Vertragspartnern der Austausch der vertraglich zugesicherten Leistungen. Die Leistung kann auf Seiten des Nachfragers die Übermittlung eines Geldbetrages durch den Agenten und auf Seiten des Anbieters die Übertragung einer Information umfassen. Migriert der Agent von dem System eines Anbieters auf das eines Nachfragers, kann der Agent als digitales Gegenstück zu einem Handlungsreisenden betrachtet werden.

Das Anwendungsfeld der elektronischen Dienstemärkte stellt einen der dynamischsten Bereiche des Internets dar. Die Wirtschaft arbeitet intensiv daran, das Internet für kommerzielle Anwendungen zu öffnen. Schon jetzt kommen eine Reihe von Agenten zum Einsatz. Mit „BargainFinder“ steht ein Agent zur Verfügung, der aus der Vielzahl der Anbieter eines Produkts (in diesem Fall Musik-CD's) den für den Nachfrager günstigsten ermittelt. Seit Ende des vorletzten Jahres bietet die Deutsche Bank ihr Homebanking-Angebot auch im Internet an. Kunden der Bank können über ein Java-Applet Einsicht in ihr Konto nehmen, Überweisungen tätigen und weitere Dienstleistungen der Bank in Anspruch nehmen. Mittlerweile sind eine Vielzahl der Banken diesem Beispiel gefolgt.

7.1.2. Sicherheitsanforderungen

Das Anwendungsgebiet elektronischer Dienstemarkt wird im Abschnitt 7.1.1 durch drei Phasen umrissen. Nachfolgend werden nun die Sicherheitsanforderungen der einzelnen Phasen bestimmt.

Angebot

Die in elektronischen Dienstemärkten erbrachten Leistungen sind nicht neu. Es existieren korrespondierende Anwendungen, die in der „Realwelt" die gleichen Aufgaben erbringen. Entsprechend sind die Sicherheitsanforderungen der Anwendungen, wie sie in der Realwelt vollzogen werden, auch in den virtuellen Abbildungen präsent. Hinzu kommen die spezifischen Anforderungen, die aus dem Einsatz von Informationstechniken resultieren.

Die Ermittlung des besten Angebots erfolgt in der „Realwelt" häufig anonym. In der virtuellen Welt der offenen Netze ist es für den Nachfrager wesentlich, daß dessen *Anonymität* auch hier gewahrt bleibt. Die Mittel der Informationstechnologien erlauben es wesentlich weitergehende Profile über das Verhalten von Nachfragern in elektronischen Dienstemärkten zu erstellen, als dieses in der „Realwelt" möglich ist. Der Einsatz von Agenten in elektronischen Dienstemärkten, die die Funktion des Angebot-Einholens verwirklichen, muß die Anonymität des Nachfragers respektieren und sichern.

Das Einholen des besten Angebots kann nur dann den Vorstellungen des Nachfragers entsprechen, wenn der Agent durch die Anbieter nicht manipuliert werden kann. Es bedarf somit der Wahrung der *Integrität* des Codes und der Daten des Agenten. Wären Manipulation am Agenten durch einen Anbieter möglich, könnte dieser das Ergebnis der Auswertung dahingehend beeinflussen, daß es ihn als günstigsten Anbieter ausweist.

Das Einholen eines Angebots durch einen Agenten beinhalten für einen Nachfolger nur dann einen Mehrwert, wenn der Anbieter das gemachte Angebot für einen ausreichend langen Zeitraum aufrecht erhält, so daß der Nachfrager die Möglichkeit hat, mit dem Anbieter in Verhandlungen zu treten. Als Sicherheitskriterium ist damit die *Verbindlichkeit* zu wahren. Im anderen Fall könnte der Anbieter dem Agenten Lockangebote unterbreiten, die einzig dem Zweck dienen, den Agenten für eine anschließende Verhandlungsphase auf das System des Anbieters zu locken.

Für die Entität, auf deren System ein Agent zur Ausführung gelangt, ist es stets von entscheidender Bedeutung, daß der Agent lediglich die spezifizierte Funktionalität aufweist und diese auch korrekt implementiert ist. Es sind somit die Sicherheitskriterien *Funktionalität* und *Korrektheit* durch den Agenten zu erfüllen, damit das den Agenten ausführende System keinen Schaden nehmen kann.

Die Phase „Angebot" zeigt, daß Sicherheitsanforderungen sich widersprechen können. Das vom Nachfrager gewünschte Sicherheitskriterium Anonymität ist für den Anbieter nicht akzeptabel. Agenten, die auf dessen System zur Ausführung gelangen, bedürfen einer Identifikation, so daß beim Eintritt eines Schadensfall beweisbar dargelegt werden kann, daß der Schaden durch diesen Agenten verursacht wurde. Die sich widersprechenden Ziele lassen sich vereinbaren, wenn der Nachfrager nicht anonym auftritt, sondern ein Pseudonym benutzt. Dem Anbieter ist nicht bekannt, welches Pseudonym von einem bestimmten Nachfrager verwendet wird, er hat jedoch die Möglichkeit, bei Eintritt eines Schadensfall die Aufhebung der Pseudonymität des Nachfragers zu verlangen. Es zeigt sich, daß eine weitere Instanz benötigt wird, der alle Beteiligten vertrauen. Die weiter Instanz tritt als Schiedsrichter im Schadensfall auf und entscheidet dann, ob die vom

Geschädigten erbrachten Beweise ausreichend sind, um die Verbindung einer Identität zu einem Pseudonym offenzulegen.

Verhandlung

In Agentensystemen, in denen Agenten eingesetzt werden, um Verhandlungen durchzuführen, sind die Sicherheitskriterien *Integrität*, *Korrektheit* und *Verbindlichkeit* zu erfüllen. Sowohl der Anbieter als auch der Nachfrager ist daran interessiert, daß die ausgehandelten Vertragsbedingungen nicht durch die Gegenseite nach Abschluß der Verhandlungen manipuliert werden können. Damit einher geht, daß beide Vertragspartner an die im Vertrag festgelegten Paragraphen gebunden sind. Beide Parteien sind eine Verbindlichkeit eingegangen, die nicht durch eine Seite aufgelöst werden darf. Desweiteren ist es für den Nachfrager essentiell, daß der von ihm entsandte Agent nicht manipuliert wird. Eine Manipulation des Agenten hätte zur Folge, daß der Vertrag nicht den Anforderungen des Nachfragers entspricht. Der Betreiber einer Umgebung für Agenten ist, wie bereits ausgeführt wurde, daran interessiert, daß der Agent korrekt implementiert ist.

Diensterbringung

Nach Abschluß eines Vertrages stehen beide Vertragspartner in der Pflicht, die von ihnen vertraglich zugesicherten Leistungen zu erbringen. Erfolgt der Austausch der Leistungen auf digitalem Wege, ist es erforderlich, daß das Sicherheitskriterium *Vertraulichkeit* gewahrt wird. Ein Dritter, der den Austausch der Leistungen beobachten und einsehen kann, könnte in der Lage sein, die ausgetauschten Leistungen für sich in Anspruch zu nehmen. Wesentlich für den erfolgreichen Abschluß einer Transaktion ist die *Verbindlichkeit*. Keine der beteiligten Parteien darf in der Lage sein, die Leistungen des anderen zu erhalten, ohne daß eine Eigenleistung erfolgt. Für den negativen Fall ist es von Bedeutung, beweisbar darlegen zu können, daß eine der Parteien seine Leistungen nicht erbracht hat. Das Sicherheitskriterium der *Zurechenbarkeit* muß erfüllt sein.

Erneut muß sichergestellt sein, daß der Code und die Daten des Agenten nicht manipuliert werden können. Im Falle, daß der Agent von einem Anbieter stammt, ist weiterhin das Sicherheitskriterium Vertraulichkeit zu erfüllen. Der entsandte Agent stellt für den Anbieter ein Wirtschaftsgut dar, das es zu schützen gilt. Bei den Agenten des Nachfragers handelt es sich i. d. R. um generierte Agenten. Die Privatheit des Agentencodes ist für den Nachfrager deshalb nicht von Bedeutung. Der Anbieter hat hingegen einen Agenten entwickelt, der eine für ihn wesentliche Anwendung erbringt.

7.1.3. Empfohlene Sicherheitsarchitekturen

Es zeigt sich, daß die notwendigen Sicherheitskriterien des Nachfragers und des Anbieters verschieden sind. Entsprechend stellen sich unterschiedliche Anforderungen an die

Sicherheitsarchitektur. Für beide Parteien sind die Sicherheitskriterien Funktionalität, Integrität, Korrektheit, Verbindlichkeit und Zurechenbarkeit von entscheidender Bedeutung. Der Nachfrager ist darüberhinaus in bestimmten Phasen daran interessiert, daß seine Identität verborgen bleibt. Wie bereits ausgeführt, läßt sich die Anonymität nicht mit dem Sicherheitskriterium der Zurechenbarkeit vereinbaren, das vom Anbieter eingefordert wird. Die Sicherheitsarchitektur muß aus diesem Grunde die Pseudonymität des Nachfragers unterstützen. Der Anbieter ist – wie sich gezeigt hat – daran interessiert, daß die Vertraulichkeit des Agentencodes gewahrt bleibt.

Der vorhergehende Abschnitt hat gezeigt, daß eine Sicherheitsarchitektur für ein Agentensystem im Anwendungsfeld der elektronischen Dienstemärkte eines vertrauenswürdigen Dritten bedarf, der in den ablaufenden Phasen als als Schiedsrichter agiert. Daraus folgt bereits, daß die Sicherheitsarchitektur nicht ausschließlich aus technischen Komponenten stehen kann. Die Einführung eines vertrauenswürdigen Dritten benötigt eine sozialen Komponente – Vertrauen.

Funktionalität/Korrektheit

Die Umgebung, in der fremde Agenten zur Ausführung gelangen, ist darauf angewiesen, zu erfahren, welche Funktionalität ein Agent beinhaltet. Die Funktionalität eines Agenten ist die Basis für die Entscheidung, ob dieser zur Ausführung gelangen darf. Die Funktionalität eines Programms erschließt sich nur dann, wenn die Semantik des Programms bekannt ist. Die Semantik eines Agenten kann aber nicht durch IT-Systeme bestimmt werden. Analysewerkzeugen ist es lediglich möglich, aus dem Aufrufen von Standardmethoden ein vages Bild der Funktionalität zu liefern.

Die Korrektheit von Programmen ist ein weiteres Sicherheitskriterium, das sich nicht erreichen läßt. Das Halteproblem lehrt, das kein Programm existiert, daß für ein beliebiges Eingabeprogramm entscheiden kann, ob dieses anhält. Entsprechend läßt sich nicht automatisch verifizieren, welche Funktionalität ein Programm beinhaltet und ob die Funktionalität korrekt implementiert worden ist.

Integrität

Die Integrität gilt es sowohl für die eingesetzten Agenten als auch für die ausführenden Umgebungen zu wahren. Die Gewährleistung der Integrität verhindert Manipulationen am Code und an den Daten. Die Integrität der ausführenden Umgebung stellt sicher, daß eine Kompromittierung des Wirtssystems durch Manipulationen ausgeschlossen wird.

Für die Sicherstellung der Integrität bieten sich mehrere Mechanismen an, die zu einem Bastionskonzept zusammengefaßt werden können. Die erste Komponente ist eine Firewall, die den Übergang vom externen zum internen Netzwerk sichert. Kommerzielle Anbieter haben bereits jetzt i. d. R. eine Firewall installiert. Nachfrager werden ebenso durch eine Firewall geschützt, wenn sie in einem Intranet agieren. Lediglich der private Nachfrager wird sein System nicht durch eine Firewall schützen können.

Die nächste Hürde, die ein maliziöser Agent zu überwinden hätte, ist eine vertrauenswürdige Hardware, in der die Agenten zur Ausführung gelangen. Die vertrauenswürdige Hardware gewährleistet, daß die Integrität eines Agenten und seiner Daten gewahrt bleiben, wenn er in die Umgebung des Anbieters migriert. Der migrierende Agent wird digital signiert, so daß er während der Übertragung vor Manipulationen geschützt ist. Im Anschluß kommt der Agent auf der vertrauenswürdigen Hardware zur Ausführung. Das Vertrauen des Nachfragers in den Hersteller bildet die Basis dafür, daß der Nachfrager die begründete Annahme hat, daß Manipulationen des Agenten ausgeschlossen sind. Dem Anbieter bietet die vertrauenswürdige Hardware ebenfalls Schutz. Zugriffe auf externe Ressourcen ist nur über eine definierte Schnittstelle möglich, die die Systeme des Anbieters und den Agenten des Nachfragers schützt (siehe Abbildung 7.1. Das von WILHELM entwickelte TPE-Modell ist eine Lösungsmöglichkeit für eine vertrauenswürdige Hardware. Diese Komponente ist ebenfalls nur auf Seiten der Anbieter zu finden. Private Nachfrager sind mit den Kosten der Komponente überfordert.

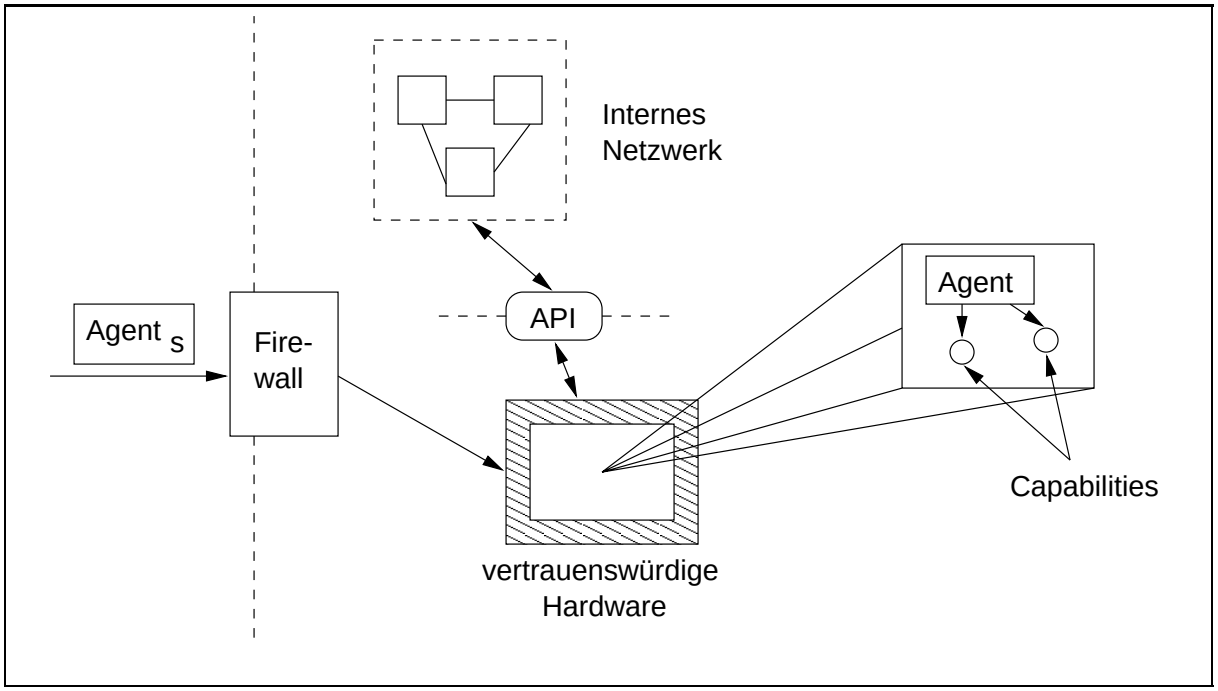

Abbildung 7.1.: Das Bastionskonzept für ein sicheres Agentensystem besteht aus den Komponenten Firewall, vertrauenswürdige Hardware und Capabilities. Ein maliziöser Agent muß alle Sicherheitsmechanismen durchbrechen, um Schaden im System zu erzeugen.

Innerhalb der vertrauenswürdigen Hardware befindet sich mit einem Capability-System die letzte Komponente des Bastionskonzepts. Alle Agenten werden in unterschiedlichen Sicherheitsdomänen zur Ausführung gebracht und werden nur mit den erforderlichen Rechten durch Capabilities ausgestattet. Die skizzierte Lösung erlaubt die

Gewährleistung der Integrität, wie sie vom Anbieter und vom Nachfrager eingefordert wird. Erforderlich hierfür sind jedoch eine Reihe von Vertrauensentscheidungen. Eine rein technische Lösung ist nicht gegeben. Vertrauen spielt in dieser Architektur eine große Rolle. Die Probleme, die durch die Vermischung von sozialen und technischen Aspekten auftreten, sind in den Abschnitten 5.2.1 und 6.1 deutlich geworden.

Gelangt ein Agent eines Anbieters auf einem Nachfrager-System zur Ausführung, können Manipulationen nicht vermieden werden. Die Aufwendungen für das oben angeführte Bastionskonzept sind durch den Nachfrager nicht tragbar. Dem Anbieter ist es lediglich möglich, den Code des Agenten digital zu signieren. Damit kann jedoch nicht sichergestellt werden, daß der Agent durch den Nachfrager korrekt ausgeführt oder der Agent zur Laufzeit nicht durch Dritte, die sich Zugang zum System des Nachfragers verschafft haben, manipuliert wird.

Damit geht einher, daß auch die Umgebung, die den Agenten des Anbieters zur Ausführung bringt, nicht in dem Maße geschützt werden kann, wie dieses für Agenten der Fall ist, die auf Seiten des Anbieters zur Ausführung gelangen. Das System eines privaten Nachfragers ist nahezu ausschließlich auf die Sicherung der Umgebung durch die Basistechnologien angewiesen. Die Ausführungen des Kapitels 5 haben gezeigt, daß die Technologien neben Implementationsmängeln eine Vielzahl von Designmängeln aufweisen.

Pseudonymität

Das Sicherheitskriterium der Pseudonymität läßt sich mit dem in Abschnitt 6.5 vorgestellten Verfahren des Onion Routings erfüllen. Die Abbildung 7.2 stellt das Verfahren dar. Zu Beginn generiert der Nachfrager ein Token, das er nachfolgend als Pseudonym verwendet. Das Pseudonym wird zusammen mit der Identität des Nachfragers und einem öffentlichen Schlüssel an den vertrauenswürdigen Dritten versandt. Die gesamte Nachricht wird zusätzlich digital signiert. Der vertrauenswürdige Dritte erkennt anhand der Signierung die Identität des Nachfragers und bestätigt das Pseudonym, indem er es digital signiert. In dieser Form wird es an den Nachfrager zurückgesandt.

Will der Nachfrager einen Agenten auf ein System eines Anbieters migrieren lassen, generiert er den Agenten und fügt diesem einen öffentlichen Schlüssel und das Pseudonym bei. Der öffentliche Schlüssel, den der Agent mit sich führt, stimmt nicht mit dem Schlüssel überein, der zur Signierung der Nachricht an den vertrauenswürdigen Dritten verwendet wurde. Der hier versandte Schlüssel ist ausschließlich dem Nachfrager bekannt und kann somit nicht zur Identifikation des Nachfragers herangezogen werden. Der Agent wird durch den Nachfrager nicht direkt an den Anbieter versandt, da der Anbieter den Anfrager ansonsten anhand der Verkehrsinformation identifizieren könnte. Stattdessen wird der Agent in ein Netz von Onion Routern versandt, die gewährleisten, daß der Migrationsweg nicht zum Nachfrager zurückverfolgt werden kann.

Die Ausführung des Agenten eines Nachfragers, dessen Identität nicht bekannt ist, birgt das Problem im Falle eines Sicherheitsproblems die Aktionen des Agenten keiner

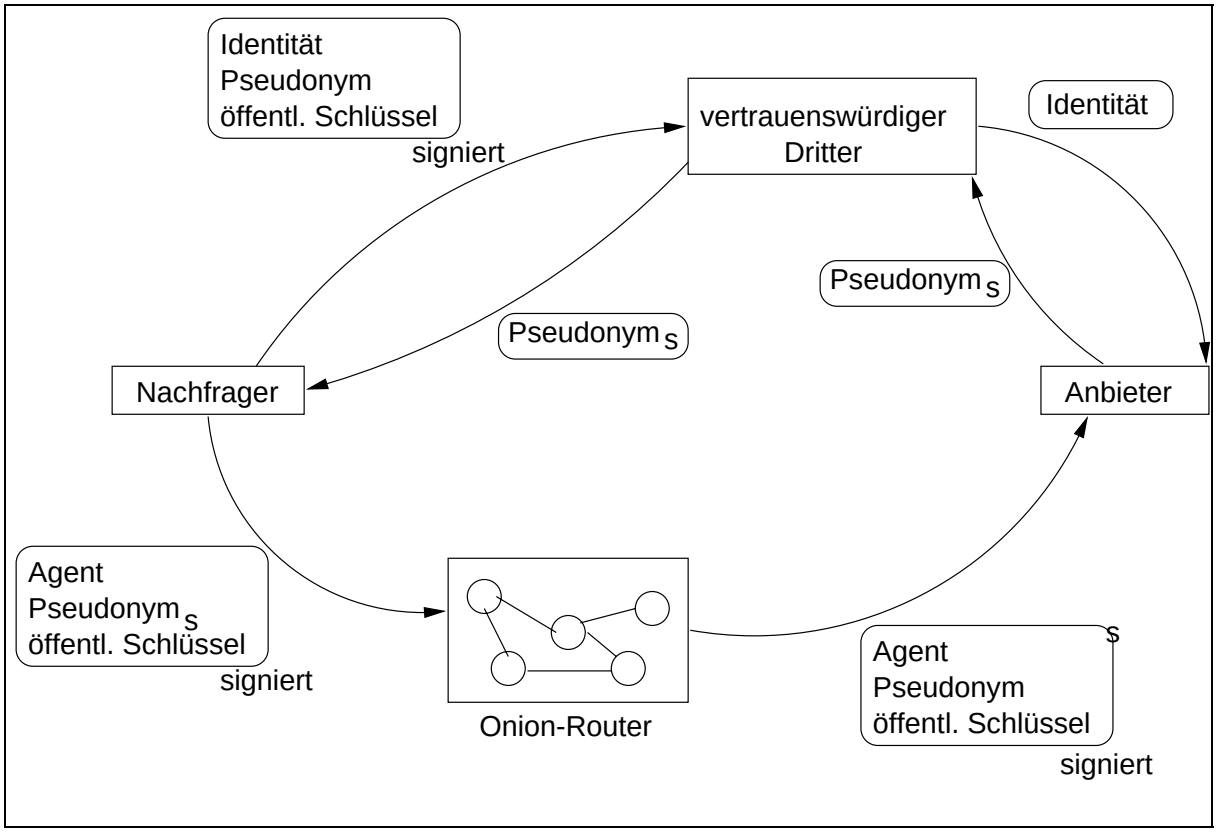

Abbildung 7.2.: Die Existenz eines vertrauenswürdigen Dritten und die Nutzung eines Netzwerkes von Onion Routern erlaubt dem Nachfrager, pseudonym zu agieren. Im Schadensfall kann der Anbieter sich an den vertrauenswürdigen Dritten mit der Bitte wenden, die zu einem Pseudonym gehörige Identität aufzudecken.

Entität zuordnen zu können. Der Anbieter hat jedoch die Möglichkeit, sich an den vertrauenswürdigen Dritten mit dem Pseudonym zu wenden. Anhand der übergeben Information kann der vertrauenswürdige Dritte entscheiden, ob er der die mit dem Pseudonym verbundene Identität offenlegt.

Mittels des öffentlichen Schlüssels, der dem Agent beiliegt, wird der Agent, im Anschluß an die Ausführung, durch die vertrauenswürdige Hardware verschlüsselt. In dieser Form wird der Agent dann an das Netz der Onion Router übergeben. Der Agent gelangt über die anfangs etablierte Verbindung schließlich auf das Ausgangssystem zurück.

Verbindlichkeit

Das Sicherheitskriterium der Verbindlichkeit ist essentiell für elektronische Dienstemärkte. Die Durchsetzung dieses Sicherheitskriteriums erfolgt erneut durch die Konsultation eines vertrauenswürdigen Dritten. Keine der in Kapitel 6 vorgestellten Sicherheitsarchitekturen beinhaltet einen Mechanismus, der sicherstellt, daß Leistungen eingehalten

werden, die zuvor dem Partner zugesichert wurden.

Im Falle eines Angebots kann der Nachfrager den Anbieter dazu auffordern, das Angebot beim vertrauenswürdigen Dritter signieren zu lassen. Der Anbieter versendet hierzu eine Nachricht, die eine Beschreibung des Angebots und einen Termin enthält, bis zu welchem der Anbieter sich an das Angebot gebunden fühlt. Ähnlich wird beim Abschluß des Vertrages verfahren. Hier weist der vertrauenswürdige Dritte den Vertragspartnern aus, daß sie beide den gleichen Vertrag unterzeichnet haben.

Vertraulichkeit

Der Agent eines Anbieters stellt für diesen ein Wirtschaftsgut dar. Der Schutz der Privatheit des Agentcodes ist damit eine Anforderung, die es zu erfüllen gilt. Das Kapitel 6 beinhaltet mit den Verfahren mobile Kryptographie und Time-limitied-Blackboxes zwei neue Sicherheitsarchitekturen, die sich bemühen, dieser Anforderung gerecht zu werden. Es zeigte sich jedoch, daß das letztere Verfahren zu schwerwiegende Mängel aufweist, als daß es für Agentensysteme geeignet wäre. Mobile Kryptographie ist hingegen bislang lediglich ein theoretisches Verfahren, für das bislang nicht absehbar ist, wann eine praktische Implementierung vorliegen könnte.

Im Abschnitt 6.10 wurde die Sicherheitsarchitektur der vertrauenswürdigen Hardware vorgestellt. DEVANBU und WILHELM haben zwei unterschiedliche Verfahren vorgestellt. Es zeigt sich jedoch, daß in beiden Verfahren die vertrauenswürdige Hardware auf Seiten des Anbieters angeordnet wird. Vertrauenswürdige Hardware kann damit keinen Beitrag für die Sicherheitsarchitektur eines Agentensystems im Bereich der elektronischen Dienstemärkte leisten, wenn es darum geht, die Vertraulichkeit des Agentencodes gegenüber dem Nachfrager zu wahren.

Der Schutz des Agentencodes der Anbieter kann in Anwendungen der elektronischen Dienstemärkte damit derzeit nicht gesichert werden. Alle Anbieter, die bislang Agentensysteme einsetzen, nehmen in Kauf einen Teil ihres geistigen Eigentums offenzulegen.

Zurechenbarkeit

Eine Sicherheitsarchitektur, deren Korrektheit nicht formal bewiesen wurde, kann Sicherheitslücken beinhalten, die von Angreifern ausgenutzt werden können. Für das ausführende System ist es von großem Interesse, daß alle durchgeführten Aktionen eines Agenten einer Identität zugeordnet werden können. Ein entsprechender Auditing-Mechanismus ist in keiner der in Kapitel 6 angeführten Sicherheitsarchitekturen enthalten.

Wesentlich für das Sicherheitskriterium der Zurechenbarkeit ist eine eindeutige Zuordnung einer Operation zu einer Identität, die die Operation veranlaßt hat. In den Agentensystemen der elektronischen Dienstemärkte ist dieses i. d. R. nicht die Benutzer-ID unter der der Agent ausgeführt wird. In mobilen Agentensystemen ist die Identität der Quelle mit den durch den Agenten durchgeführten Operationen zu verknüpfen. Es

zeigt sich, daß auch die Basistechnologien, wie sie in Kapitel 5 vorgestellt wurden, bislang keine derartige Aufzeichnung erlauben.

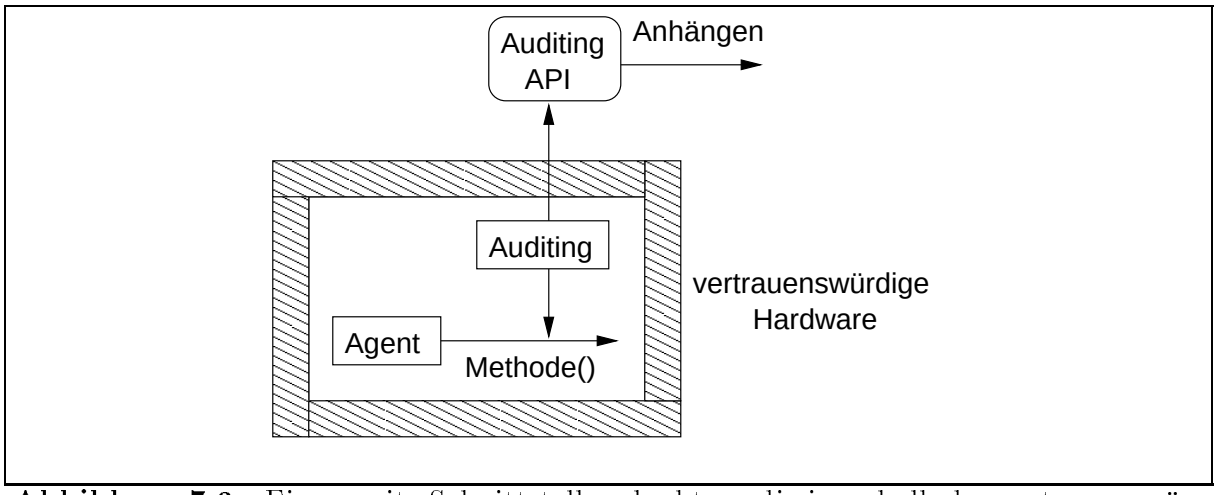

Abbildung 7.3.: Eine zweite Schnittstelle erlaubt es, die innerhalb der vertrauenswürdigen Hardware aufgezeichneten Operation außerhalb der Umgebung zu transportieren. Die Informationen werden auf einem sicheren Medium abgelegt.

Im Rahmen der bereits angeführten vertrauenswürdigen Hardware wäre eine weitere Schnittstelle denkbar, die ausschließlich dazu dient, die von Agenten ausgelösten Aktionen außerhalb der vertrauenswürdigen Hardware abzulegen. Alle Methodenaufrufe des Agenten werden dabei durch einen Auditing-Mechanismus innerhalb der Hardware ermittelt (siehe Abbildung 7.3) und über die Auditing-API auf ein Medium geschrieben, das den Entitäten innerhalb der vertrauenswürdigen Umgebung nicht zugänglich ist. Der Auditing-API beinhaltet lediglich Methoden, mit denen Informationen angehängt werden können.

Es sei darauf hingewiesen, daß das Sicherheitskriterium der Zurechenbarkeit ohne Wert ist, so die erlangten Informationen nicht ausreichend sind bzw. nicht eingesetzt werden können, um den Verursacher mit juristischen Mitteln für die verursachten Schäden haftbar zu machen. Die Sicherheit von Agentensystemen basiert damit erneut auf einer nicht-technischen Komponente.

Ergebnis

Die Realisierung von Anwendungen aus dem Gebiet der elektronischen Dienstemärkte erfordert die Erfüllung von Sicherheitskriterien, die durch den Anbieter und den Nachfrager definiert werden. Die Analyse hat ergeben, daß die Kriterien Funktionalität, Integrität, Korrektheit, Pseudonymität (Anonymität), Verbindlichkeit, Vertraulichkeit und Zurechenbarkeit gewährleistet sein müssen, damit beide Parteien ein sicheres Agentensystem nutzen können.

Die Sicherheitskriterien lassen sich bis zu einem gewissen Grad erfüllen, so die Agenten in Umgebungen des Anbieters zur Ausführung gelangen. Dort lassen sich Sicherheitskomponenten zu einer Sicherheitsarchitektur integrieren, die von beiden Parteien akzeptiert werden kann. Unerläßlich ist jedoch die Existenz eines vertrauenswürdigen Dritten, dem alle Beteiligten vertrauen. Damit einher geht die Erfordernis einer funktionierenden PKI, die eine vertrauliche Übertragung von Agenten und deren Daten ermöglicht. Die Erstellung von digitalen Signaturen ist eine weitere Funktionalität, die die Existenz einer PKI notwendig macht.

Anwendungen der elektronischen Dienstemärkte, die es erforderlich machen, daß Agenten auf den Systemen der Nachfrager ausgeführt werden, bergen über die bereits beschriebenen Probleme weitere. Die Sicherung der Integrität von Agenten, Daten und Umgebung kann aufgrund der beschränkten Mittel als nicht erreichbar beschrieben werden. Gleiches gilt für das Kriterium der Zurechenbarkeit.

Unabhängig von dem Ort der Ausführung von Agenten sind die Kriterien der Funktionalität und der Korrektheit nicht durchsetzbar. Alle beteiligten Entitäten können lediglich Erkenntnisse erlangen, die für sie zu der begründeten Annahme führen, daß eine Ausführung von Agenten keine Auswirkungen auf andere Agenten und die Umgebungen haben.

In Hinblick auf die in Kapitel 4 identifizierten Werte in Agentensystemen muß davon ausgegangen werden, daß die Nichterfüllung der angeführten Sicherheitskriterien zu wesentlichen Verlusten führen kann. In die Betrachtung ist Einzubeziehen, daß die elektronischen Dienstemärkte erst vor dem Durchbruch stehen. Werden zu einem solchen Zeitpunkt Technologien eingesetzt, die die Sicherheit der Anwendung beeinträchtigen, kann diese dazu führen, daß das gesamte Anwendungsgebiet an Akzeptanz verliert.

7.2. Systemmanagement

7.2.1. Inhalt

Netzwerke stellen Dienste zur Verfügung, die von einer immer größer werdenden Menge von Anwendern genutzt werden. Ein exponentielles Wachstum der Netze ist häufig die Folge. Verbunden mit der großen Zahl an Teilnehmern ist der große finanzielle Verlust bei Ausfall eines Netzes. Nur ein effektives Netzwerk-Management ermöglicht ein funktionierendes Netz, in dem auftretende Probleme innerhalb kürzester Zeit lokalisiert und gelöst werden können. Agenten, die eine klar spezifizierte Aufgaben übernehmen, z. B. die Beobachtung von Log-Dateien, die bei Auftreten eines bestimmten Ereignisses die nötigen Aktionen veranlassen, stellen eine flexible und skalierbare Lösung dar. Mit ihrer Hilfe lassen sich die wichtigen Ereignisse, die eine Reaktion erfordern, aus der Gesamtmenge herausfiltern. Die Verwaltung unterschiedlicher System-Konfigurationen läßt sich mit Agenten einfacher handhaben. So kann die Veränderung einer System-Konfiguration einen Agenten dazu veranlassen, alle Rechner, die der Konfiguration ent-

sprechen, anzupassen. Agenten sind dazu geeignet, ungewöhnliche Ereignisse zu verfolgen und bei überschreiten einer Reizschwelle den Administrator zu benachrichtigen. Agenten im Netzwerk-Management eingesetzt, sind flexibel, fehlertolerant und erweiterbar.

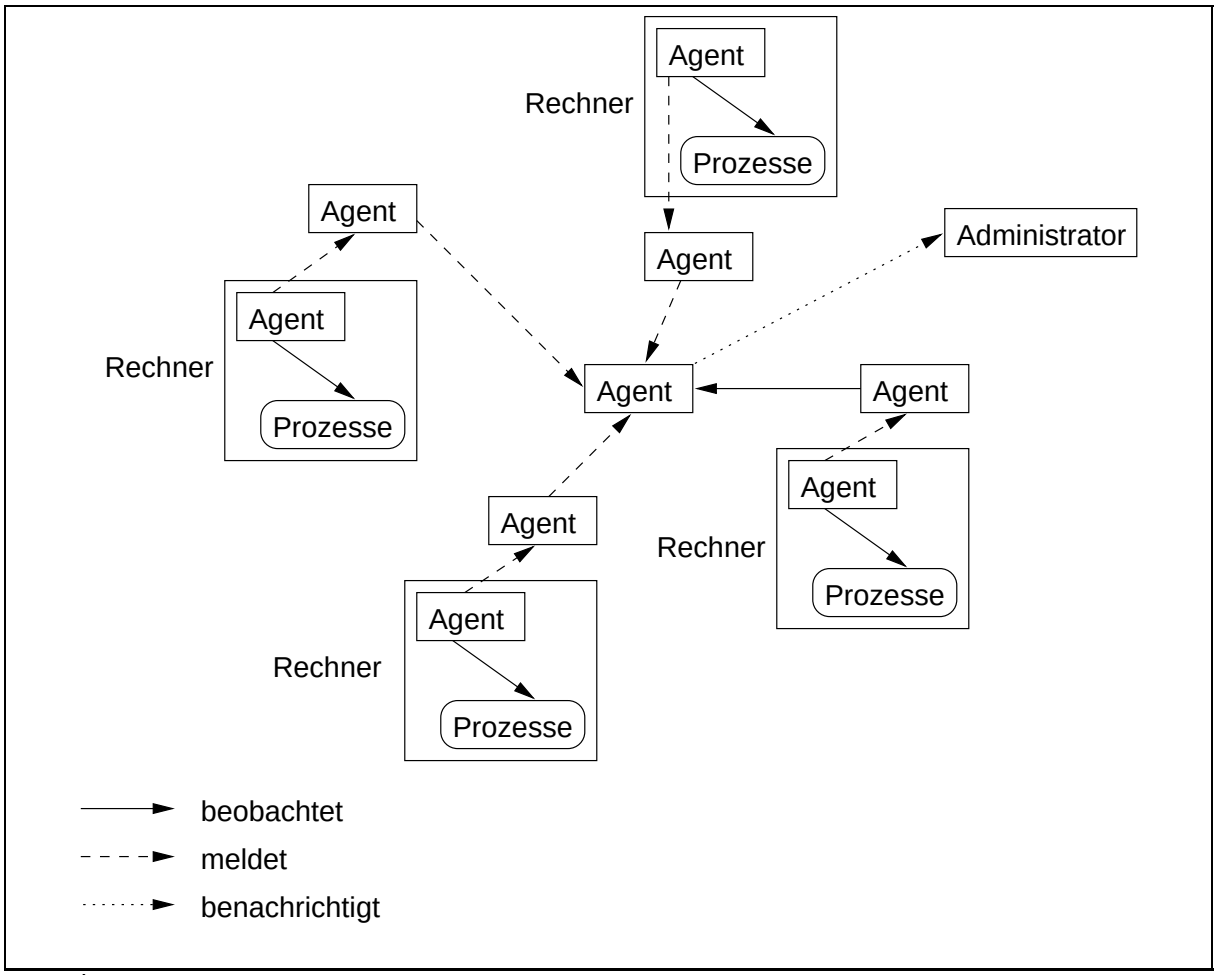

Abbildung 7.4.: Die Agenten innerhalb des Netzwerks „betreuen“ jeweils einen Rechner und leiten wichtige Ereignisse an die übergeordneten Agenten weiter. Ein zentraler Agent alarmiert die System-Administration, wenn ein Eingriff erforderlich ist.

Die Abbildung 7.4 zeigt eine mögliche Anwendung für Agenten im System-Management. Jeder Maschine im Netzwerk ist ein Agent zugeordnet, der die lokalen Ereignisse auswertet. Einem weiteren Agenten, der außerhalb einer jeden Maschine angeordnet ist, werden durch den erstgenannten Agenten besondere Ereignisse zugeleitet. Auch dieser Agent wertet die ihn erreichenden Ereignisse aus und meldet sie schließlich an einen zentralen Agenten weiter. Der zentrale Agent ist der letzte der Agentenkette und alarmiert den Administrator beim Eintritt ungewöhnlicher Ereignisse.

SPAFFORD et al. haben eine weitere Anwendungsmöglichkeit von Agenten im Bereich des System-Managements identifiziert. In zwei Arbeiten legen sie dar, daß Agenten

die Funktion eines Intrusion Detection System (IDS) in Netzwerken übernehmen können [CS95, BKS+97].

7.2.2. Sicherheitsanforderungen

Wie im vorangegangenen Anwendungsgebiet der elektronischen Dienstemärkte wird für das Anwendungsgebiet System-Management zunächst ebenfalls analysiert, welche Sicherheitskriterien für auf Agentensystemen basierenden Anwendungen von wesentlicher Bedeutung sind.

Integrität

Agenten, die Aufgaben im Bereich des System-Managements übernehmen, sind mit weitgehenden Rechten ausgestattet. Die im ersten Szenario vorgestellten Agenten, die ausschließlich Informationen sammeln, besitzen die geringsten Rechte. Die beiden folgenden Szenarien beinhalten hingegen Agenten, die nahezu in beliebige Prozesse eingreifen können und dabei eine Vielzahl von Tätigkeiten autonom erledigen. Manipulationen an Agenten sind aus diesem Grund vollständig zu vermeiden, ansonsten sind die Agenten des System-Managements der Einstiegspunkt, um die Sicherheitsmechanismen des Systems zu umgehen.

Manipulationen durch die eingesetzten Agenten werden nicht angenommen. Agentensysteme, die im System-Management eingesetzt werden, sind entweder Eigenentwicklungen oder vollständige Produkte, die von einem Hersteller bezogen werden. Agenten, die System-Management-Aufgaben übernehmen, stammen nicht aus einem anderen Netzwerk. Sie migrieren nicht in ein Netzwerk, um dort Aufgaben zu erledigen.

Korrektheit

Agenten des System-Managements sind im gesamten Netzwerk vorhanden. Implementationsmängel innerhalb des Agentensystems können sich auf sämtliche angeschlossene Systeme auswirken. Mängel im Agentensystem können dazu führen, daß das gesamte Netzwerk kollabiert. Alle Anwendungen innerhalb des Netzwerks wären davon betroffen. Die gesamte Produktivität würde somit zum Erliegen kommen.

Zur Vermeidung der von Implementationsmängeln ausgehenden Gefahren ist deshalb zu fordern, daß die Korrektheit für Agentensysteme sichergestellt ist, die Anwendungen im Bereich des System-Managements implementieren.

Verfügbarkeit

Es ist Aufgabe des System-Managements, die in einem Netzwerk zusammengefaßten Systeme zu pflegen und den „normalen“ Ablauf der Anwendungen zu gewährleisten. Der Ausfall einer Anwendung des System-Managements hat zur Folge, daß den Aufgaben nicht mehr nachgegangen werden kann. Die Folge dessen wäre, daß auf Ereignisse im

Netz, wie beispielsweise dem Ausfall einer Komponente, nicht mehr angemessen reagiert werden könnte. Die Verfügbarkeit der Anwendungen des System-Managements ist somit unbedingt sicherzustellen.

Zurechenbarkeit

Das System-Management vollführt vielfältige Tätigkeiten in Netzwerken. Kommen dabei Agenten zum Einsatz, ist darauf zu achten, daß die durchgeführten Operationen nachvollzogen werden können. Die Zurechenbarkeit der Operationen stellt sicher, daß Implementationsmängel der Agenten, die im Netzwerk zu Problemen führen, aufgedeckt werden können. Lediglich in Agentensystemen, für die formal bewiesen wurde, daß sie korrekt sind, kann auf den Aspekt der Zurechenbarkeit verzichtet werden.

7.2.3. Empfohlene Sicherheitsarchitektur

Die Analyse des Anwendungsgebiets System-Management hat ergeben, daß die Sicherheitskriterien Integrität, Korrektheit, Verfügbarkeit und Zurechenbarkeit zu gewährleisten sind, damit ein entsprechendes Agentensystem sicher operieren kann. Es wurde bereits angeführt, daß die eingesetzten Agenten nicht von außen in das Agentensystem migrieren. Entsprechend muß die Sicherheitsarchitektur keine Risiken adressieren, die von Außentätern ausgehen.

Die Anwendungen des System-Managements werden unter der Kontrolle der System-Administration ausgeführt. Die Nutzer des Netzwerks können nicht dem Agentensystemen interagieren. Eine Einflußnahme durch die Entsendung von Agenten oder den Aufbau einer Umgebung für die System-Agenten wird durch das Betriebssystem verhindert. Agentensysteme des System-Managements sind abgeschlossene Systeme, die einzig von der System-Administration bedient werden.

Unter den genannten Annahmen ist eine Sicherheitsarchitektur für die Durchsetzung der Sicherheitskriterien erforderlich, sie ist aber nicht Bestandteil des Agentensystems sondern des Betriebssystems, das in dem Netzwerk eingesetzt wird. Die Sicherheitskriterien Integrität, Verfügbarkeit und Zurechenbarkeit müssen durch das Betriebssystem durchgesetzt werden. Wie auch für alle anderen Prozesse ist es Aufgabe des Betriebssystems, die Prozesse des Agentensystems vor jenen der Anwender zu schützen. Gleichzeitig ist auch der Schutz der Prozesse der Anwender vor denen des Agentensystem zu sichern.

Der Charakter eines Agentensystems, das ausschließlich aus stationären Agenten besteht bzw. dessen Agenten nur innerhalb des Intranets migrieren, unterscheidet sich nicht von den anderen lokalen Anwendungen. Es zeigt sich, daß die besonderen Sicherheitsanforderungen der Agentensysteme ausschließlich aus dem Aspekt der Mobilität resultieren. Die durch die Dimensionen des Sicherheitsbegriffs zum Ausdruck gebrachten Erfordernisse sind dort von Belang, wo externe Agenten in ein Netzwerk migrieren.

8. Ausblick

„*Thus, the question is not – what can cryptography do for mobile code security – but – what can mobile code do for security.*"
– Christian Tschudin

Ist der Einsatz von Agentensystemen in Anwendungen aus der Sicht der Sicherheit zu akzeptieren? Die vorliegende Arbeit hat Erkenntnisse erzielt, die für die Beantwortung der Frage herangezogen werden können. Es zeigt sich, daß sich mit dem Einsatz von Agentensystemen spezifische Sicherheitsprobleme verbinden, denen es mit geeigneten Sicherheitsmechanismen zu begegnen gilt. Chess hat darauf hingewiesen, daß eine Reihe von Annahmen für die Wahrung der Systemsicherheit in Agentensystemen nicht länger Bestand haben.

Agenten, die autonom in Netzen zwischen verschiedenen Umgebungen migrieren können, machen es erforderlich, neben dem Schutz der Umgebung vor den migrierenden Agenten, weitere Schutzmaßnahmen zu ergreifen. Neue Schutzaspekte werden durch die anderen beiden Dimensionen des Sicherheitsbegriffs, wie er in der Arbeit verwendet wurde, dargestellt. Mit dem notwendigen Schutz der Agenten vor den Wirtssystemen ergibt sich eine neue Anforderungen, deren tiefergehende Untersuchung bislang aussteht.

Das Kapitel 5 hat gezeigt, daß die in Agentensystemen eingesetzten Basistechnologien Sicherheitsmechanismen beinhalten, die jedoch nicht ausreichend sind, um den besonderen Anforderungen gerecht zu werden. Neben den bislang nicht adressierten Dimensionen des Sicherheitsbegriffs sind die bestehenden Sicherheitsmechanismen mangelhaft umgesetzt. Ein Konzept für die entwickelten Sicherheitsmechanismen ist der Mehrzahl Produkte nicht zu entnehmen. Die Forschung im Bereich der Agentensysteme hat die aufgezeigten Probleme erkannt und versucht, die Sicherheitsprobleme der Agentensysteme durch geeignete Sicherheitsarchitekturen zu beheben. Es zeigte sich im Kapitel 6 jedoch, daß die entwickelten Sicherheitsarchitekturen bislang in keinem der existierenden Agentensysteme eingesetzt werden. Desweiteren werden lediglich einzelne Aspekte der Systemsicherheit adressiert. Eine integrierte Sicherheitsarchitektur, die den Schutzbedürfnissen von realen Anwendungen gerecht werden kann, ist bislang nicht existent.

Der Schutz des Agenten vor der Umgebung wird von mehreren Architekturen versucht zu adressieren. Es zeigt sich, daß das Verfahren der mobilen Kryptografie bislang lediglich ein theoretisches Konstrukt ist. Nicht absehbar ist, ob sich das von Sander und Tschudin entwickelte Verfahren jemals in die Praxis wird umsetzen lassen. Weiterhin

bleibt zu klären, ob die aufgezeigten Probleme des Verfahrens gelöst werden können. Das von HOHL entwickelte Konzept erscheint durch die angesprochenen Mängel ungeeignet für den Einsatz in Agentensystemen. Die anderen Lösungen bedürfen der Existenz einer PKI und dem Vertrauen gegenüber einer Reihe von unterschiedlichen Entitäten. Damit wird die Offenheit der Agentensysteme in einer Weise beschränkt, so daß die durch die Agenten-Technologien erzielten Vorteile gemindert werden.

Der Versuch der Darstellung einer integrierten einer Sicherheitsarchitektur für den Bereich der elektronischen Dienstemärkte hat offenbart, daß der Schutz der Agenten vor der Umgebung von großer Bedeutung für die Berechtigung von Agentensystemen ist. Es zeigte sich weiterhin, daß die Sicherheitsanforderungen der Anwendungen der elektronischen Dienstemärkte nicht hinreichend erfüllt werden können. Die vorgestellte Lösung und ähnliche Ansätze erfordern von den Beteiligten Vertrauensentscheidungen. Die vorangegangenen Kapitel haben gezeigt, daß die Vermischung von sozialen und technischen Aspekten zu Problemen führen. Bislang gibt es keine Vorstellung darüber, auf welcher Basis eine begründete Entscheidung hinsichtlich der Gewährung der Ausführung von Agenten getroffen werden kann. Die diskutierten Sicherheitsarchitekturen ziehen sich letztlich darauf zurück, daß in kritischen Fällen der Nutzer des Agentensystems für die Entscheidung, ob ein Agent zu Ausführung gelangen darf, konsultiert wird. Hierbei handelt es sich nur dann um einen zulässigen Schritt, wenn der Nutzer über ein hinreichendes Wissen verfügt und die für die Entscheidung erforderlichen Informationen erhält. In den vorhandenen Agentensystemen werden beide Punkte nicht erfüllt.

Das Kapitel 7 zeigt, daß die besonderen Sicherheitsprobleme der Agentensysteme aus dem Aspekt der Mobilität resultieren. Der Einsatz von Agentensysteme, die ausschließlich auf stationären Agenten beruhen, bringt keine wesentlichen neuen Sicherheitsprobleme mit sich. Die Mobilität ist jedoch der Aspekt der heutigen Agentensysteme, von dem sich die Nutzer den höchsten Gewinn versprechen. Die Entwicklung der Softwaresystem sieht mittlerweile vor, daß Produkte aus unabhängigen Komponenten zusammengesetzt werden können, die insgesamt eine vollständige Anwendung ergeben. Komponenten können bei Bedarf dynamisch zur Laufzeit in das Produkt für die Erfüllung einer bestimmten Aufgabe integriert werden. Nach Abschluß der Aufgabe wird die Komponente wieder entfernt. Ein ähnliches Konzept wird mit der Entwicklung der Network Computer (NC) vorangetrieben.

Es bedarf der Fortführung der Forschung im Bereich der mobilen Agenten. Sollen mobile Agentensysteme künftig sicher in Anwendungen einsetzbar sein, sollen Anwender künftig in der Lage sein, ihre Anwendungen dynamisch zur Laufzeit aus Komponenten zusammenzusetzen, die aus einem Netzwerk bezogen werden, sollen NC nicht länger ein Produkt der Forschungslabore bleiben, bedarf es der Lösung der aufgezeigten Sicherheitprobleme.

Es bleibt abschließend darauf hinzuweisen, daß eine funktionierende Sicherheitsarchitektur einem Produkt nicht zwingend zum Vorteil gereifen muß. Die Podiumsdiskussion zum Abschluß der TrEC´98 zeigte, daß heute Anbieter erfolgreich sein können, wenn sie wenig oder keine Sicherheitsmechanismen in ihre Produkte einfließen lassen, jene Pro-

dukte mit integrierten Sicherheitsmechanismen dagegen vom Markt verschwinden. Hier zeigt sich, daß bislang nicht ausreichend deutlich gemacht wird, welche Auswirkungen unsichere Anwendungen im Zeitalter der „Informationsgesellschaft“ haben können. Es ist die Auffassung des Autors, daß sich mittel- bis langfristig die Erkenntnis durchsetzen wird, daß die Sicherheit eines Produkts ein Wettbewerbsvorteil ist. Zukünftig werden IT-Systeme den komplexer werdenden Aufgaben nur dann gerecht werden können, wenn die spezifischen Sicherheitsanforderungen erfüllt werden können.

Anhang

A. Fallbeispiele

Die Ausführungen der Arbeit haben deutlich werden lassen, daß in Agentensystemen Gefahren existieren, die zu beachten sind, so Agenten-Technologien in Anwendungen zum Einsatz gelangen sollen. Die folgenden Fallbeispiele sollen zeigen, daß die aufgezeigten Sicherheitsprobleme nicht alleine theoretischer Natur sind, sondern vielmehr reale Gefahren wiederspiegeln.

A.1. Big-Brother

Big-Brother ist ein Agent, der auf Basis von Voyager in der Version 2.0beta 1 vom Autor entwickelt worden ist. Ziel der Entwicklung war die Darlegung, daß die derzeit existierenden Agentensysteme nicht ausreichend sicher sind. Big-Brother zeigt, daß bereits die Sicherheitsmechanismen, die den Schutz der Agenten untereinander gewährleisten sollen, in der Implementation von Voyager nicht existent sind. Die von Big-Brother angewandten Angriffstechniken sind allesamt gegen andere Agenten gerichtet, die sich in der gleichen Umgebung wie der maliziöse Agent aufhalten.

Die Abbildung A.1 zeigt die grafische Schnittstelle zum Agenten BigBrother. Angriffe gegen andere Agenten erfolgen in mehreren Schritten. Zunächst bedarf es der Etablierung eines Quellsystems für den maliziösen Agenten. Hierzu wird ein Voyager-Server mit dem Kommando `voyager 3456` gestart. Die Oberfläche des Agenten Big-Brother wird mit dem Kommando `appletviewer MonitorApplet.html` instantiiert. Abschließend wird eine weitere Umgebung benötigt, die das System widerspiegelt, auf dem andere Agenten migrieren und von dem Agenten Big-Brother angegriffen werden.

Damit sind die Voraussetzungen geschaffen, um eine Instanz des maliziösen Agenten zu generieren. Hierzu wird in dem Textfeld neben dem Button „Launch BigBrother“ eine IP-Adresse einschließlich einer Port-Nummer eingetragen. Mit der Betätigung des Buttons wird eine Instanz eines Big-Brother Agenten generiert, der sogleich auf das durch die IP-Adresse spezifizierte System migriert. In diesem Beispiel befindet sich das andere System auf dem gleichen Rechner, es wird lediglich eine andere Port-Nummer verwendet (9000).

Die Ankunft des Big-Brother-Agenten produziert auf der Konsole eine Meldung „`Registered...`“, wie sie auch in der Abbildung A.2 zu sehen ist. Der Agent Big-Brother hat sich damit in der Voyager-Umgebung mit dem Port 9000 als `SystemListener` regi-

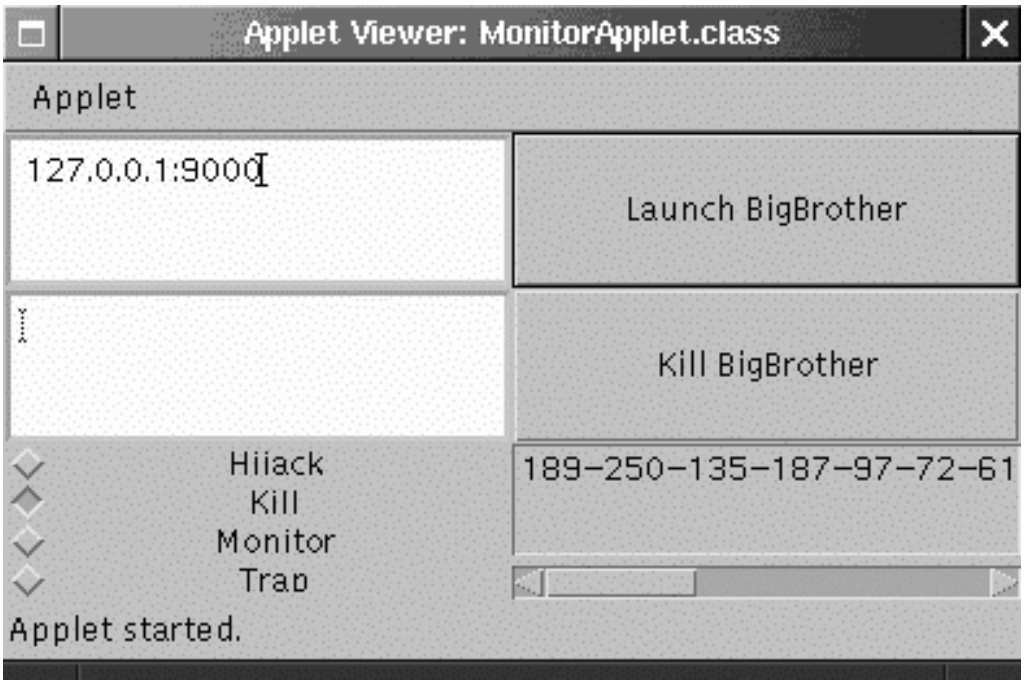

Abbildung A.1.: Die grafische Benutzungsschnittstelle zum Agenten Big-Brother.

striert. Objekte, die sich in einer Voyager-Umgebung als `SystemListener` registrieren, werden über alle System-Ereignisse informiert, die in der Voyager-Umgebung generiert werden. Eine Reihe solcher Ereignisse werden durch Agenten ausgelöst, die in der Umgebung agieren. Besucht in der Folge ein Agent das Voyager-System, werden bei dessen Ankunft Ereignisse generiert, die durch den Agenten Big-Brother ausgewertet werden. Die generierten Ereignisse sind der Abbildung A.2 zu entnehmen.

Die Registrierung des maliziösen Agenten erfolgt durch die vordefinierte Methode `registerListener`, die vom Agenten implementiert wurde und diesem als Callback-Methode im Falle einer Migration übergeben wird. Die Implementation der Methode ist der Abbildung A.3 zu entnehmen.

Bei der Ankunft eines Agenten wird an jedem registrierten `SystemListener` die Methode `systemEvent` aufgerufen. Die Methode wurde für den BigBrother implementiert und ist in der Abbildung A.4 zu sehen. Die Ankunft eines Agenten löst ein Ereignis vom Typ `SystemEvent.SENDING` aus. Das der Methode übergebene Ereignis beinhaltet eine textuelle Beschreibung, wie sie in Abbildung A.2 zu sehen ist. Eine Beschreibung beginnt jeweils mit `SystemEvent ( ... )`. Der Agent extrahiert aus der textuellen Beschreibung den GUID des ankommenden Objekts.

Mit der Extraktion des GUIDs besitzt der Agent Big-Brother eine virtuelle Referenz auf den gerade angekommenden Agenten, der es ihm ermöglicht, die öffentlichen Methoden des anderen Agenten aufzurufen. Die Schnittstelle zum Agenten Big-Brother

```
voyager 9000
voyager(tm) 2.0 beta 1, copyright objectspace 1997
address = 127.0.0.1:9000
Registered...

SystemEvent( load closure from remote application Visitor,
127.0.0.1:1173 at Sat Jan 02 16:17:38 GMT+01:00 1999
[127.0.0.1:9000] )
SystemEvent( load classes from remote application [Visitor],
127.0.0.1:1173 at Sat Jan 02 16:17:39 GMT+01:00 1999
[127.0.0.1:9000] )
SystemEvent( load closure from remote application VVisitor,
127.0.0.1:1173 at Sat Jan 02 16:17:39 GMT+01:00 1999
[127.0.0.1:9000] )
SystemEvent( load classes from remote application [VVisitor],
127.0.0.1:1173 at Sat Jan 02 16:17:39 GMT+01:00 1999
[127.0.0.1:9000] )
SystemEvent( sending Sync( __ack(ILcom.objectspace.voyager.
Address;)V, 127.0.0.1:9000/599972973 -> 127.0.0.1:1173/98-236-
91-189-137-95-247-90-120-171-129-128-128-129-132-21 ) to
127.0.0.1:1173/98-236-91-189-137-95-247-90-120-171-129-128-
128-129-132-21 (166 bytes ) at Sat Jan 02 16:17:39 GMT+01:00 1999
[127.0.0.1:9000] )

Hello World

Ahhhh...
```

Abbildung A.2.: Die Abbildung zeigt die Konsole einer Voyager-Umgebung, in der sich der Agent Big-Brother registriert hat. Ein weiterer Agent ist in die Umgebung migriert und hat eine Meldung produziert. Der Agent Big-Brother hat abschließend vom angreifenden System die Aufforderung erhalten, den anderen Agenten zu terminieren.

```
public void registerListener() {
  try {
    System.out.println("Registered...");
    address = Voyager.getAddress();
    Voyager.addSystemListener(this);
  }
  catch(Exception e) {
    System.out.println(e.toString());
  }
}
```

Abbildung A.3.: Die obige Methode `registerListener` sorgt dafür, daß der Agent Big-Brother über alle System-Ereignisse der Voyager-Umgebung informiert wird.

```
public void systemEvent(SystemEvent e) {
  String message = e.toString();
  String arg2 = (String) e.getArg2();
  int code = e.getCode();

  switch( code ) {
  case SystemEvent.SENDING:
    if ( message.indexOf("Address") != -1 ) {
      System.out.println(e.toString());
      start = arg2.indexOf("/")+1;
      guid = arg2.substring(start);
      host = address;
      victims.addElement(guid);
      try {
        monitor.register(guid,host);
      }
      catch(VoyagerException ve) {
        System.out.println(ve.toString());
      }
    }
    break;
  }
}
```

Abbildung A.4.: Die Methode `systemEvent()` wird durch die Voyager-Umgebung am Agenten BigBrother aufgerufen. Die Methode wertet die System-Ereignisse aus und extrahiert aus ihnen den GUID neuer Agenten.

zeigt den GUID des ermittelten Agenten im rechten unteren Textfeld an (siehe Abbildung A.1). Neben dem Textfeld sind die Aktionen angeordnet, die auf identifizierte Agenten ausgeführt werden können. Durch einen Doppelklick auf einen GUID wird der Agent Big-Brother dazu aufgefordert, den korrespondierenden Agenten zu entführen, zu terminieren, zu beobachten oder zu fangen.

```
public void hijackObject(String guid, String host) {
  VObject victim;

  try {
    victim = (VObject) VObject.forObjectAt(host+"/"+guid);
    victim.moveTo(home, "sayHello");
  }
  catch(VoyagerException ve) {
    System.out.println(ve.toString());
  }
}
```

Abbildung A.5.: Die Methode `hijackObject` entführt einen Agenten, der durch den übergebenen GUID und der zugehörigen IP-Adresse spezifiziert wird.

Die Abbildung A.5 stellt die Methode `hijackObject` dar, die den Agenten Big-Brother dazu veranlasst, einen anderen Agenten zu entführen. Übergeben wird der Methode der GUID des anderen Agenten und die IP-Adresse des Systems, auf dem sich der Agent aufhält. Der Agent Big-Brother generiert aus den übergebenen Strings eine virtuelle Referenz auf den anderen Agenten. Mittels der virtuellen Referenz kann dann anschließend die Methode `moveTo()` am Agenten aufgerufen werden, die den ihn in diesem Fall dazu veranlaßt, auf das System des Angreifers zu migrieren.

In dem Beispiel aus Abbildung A.2 wurde der Agent Big-Brother hingegen dazu veranlaßt, den identifizierten Agenten zu terminieren. Hierzu wird die Methode `killObject` am Agenten BigBrother aufgerufen. Wie in allen anderen Methoden, wird auch hier eine virtuelle Referenz auf den Opfer-Agenten generiert, über die schließlich die Methode `dieNow()` am Opfer-Agenten aufgerufen wird. Anschließend wird der Agent aus der Liste der erkannten Agenten entfernt.

Die Mächtigkeit des Voyager-Systems erlaubt dem Agenten Big-Brother auch die Ausführung von Methoden auf Agenten, die daß System, auf dem sie von Big-Brother erkannt wurden, bereits wieder verlassen haben. Verläßt ein Agent eine Voyager-Umgebung, wird ein sog. *Forwader* generiert, der alle den Agenten betreffenden Anfragen an diesen weiterleitet.

Das Fallbeispiel Big-Brother zeigt sehr deutlich, daß es mit einfachen Mitteln möglich ist, Agenten zu implementieren, die sich die Möglichkeiten eines Agentensystems zu nutze machen, um andere Agenten anzugreifen. Im Falle von Voyager wurde die Aufgabe we-

```
public void killObject(String guid, String host) {
  VObject victim;

  try {
    victim = (VObject) VObject.forObjectAt(address+"/"+guid);
    victim.dieNow();
    monitor.deregister(guid, address);
  }
  catch(VoyagerException ve) {
    System.out.println(ve.toString());
  }
}
```

Abbildung A.6.: Die Methode `hijackObject` terminiert einen Agenten, der durch die übergebene GUID und der zugehörigen IP-Adresse spezifiziert wird.

sentlich erleichtert. Es sind eine Reihe von Methoden öffentlich aufrufbar, die dieses nicht ohne weitere Prüfung durch den Agenten, an dem sie aufgerufen werden, sein sollten. Zu den Methoden zählen u a. `dieNow()`, `moveTo()` und `addObjectListener()`. Desweiteren darf es Agenten nicht möglich sein, sich in Voyager-Umgebungen als `SystemListener` registrieren zu lassen. Damit stehen dem angreifenden Agenten jene Informationen zur Verfügung, die es ihm ermöglichen, virtuelle Referenzen zu anderen Agenten zu generieren. Es sei darauf hingewiesen, daß alle dargestellten Angriffe auch dann durchgeführt werden können, wenn ein Security-Manager instantiiert ist.

Es zeigt sich hier deutlich, welche Auswirkungen sich aus dem Umstand ergeben, daß Java – und damit auch Voyager – keine Besitzer-Metapher unterstützen. Jeglichen Objekten ist der Aufruf jener Methoden gestattet, die als `public` deklariert sind. Alle oben angeführten Methoden – sie sind an jedem Voyager-Objekt aufrufbar – sind als öffentlich deklariert. Das Objekt, an dem eine öffentliche Methode aufgerufen wird, hat keine Möglichkeit, die Identität des Aufrufers in Erfahrung zu bringen. Auf der Ebene des Agenten kann somit keine Entscheidung bezüglich des Aufrufes öffentlicher Methoden durch andere Objekte getroffen werden. Auf Ebene des Security-Managers ist eine derartige Entscheidung möglich, sie wird dort aber nicht getroffen. Ein Security-Manager, der in einer Voyager-Umgebung installiert ist, kann die Identität eines aufrufenden Agenten ermitteln, er kann jedoch nicht entscheiden, ob der Aufruf einer Methode zu gestatten ist. Hierzu wäre es erforderlich, daß der Agent, an dem eine öffentliche Methode aufgerufen wird, seine Sicherheitsanforderungen mit sich führt, die vom Security-Manager interpretiert werden können. Die Sicherheitsanforderungen des Agenten würden dann in den Entscheidungsprozeß des Security-Managers einfließen. Dieses wird derzeit weder durch Java noch durch Voyager noch von einem anderen Agentensystem unterstützt.

A.2. Hostile Applets

LADUE hat bereits frühzeitig darauf hingewiesen, daß Java-Agenten implementiert werden können, die sich auch dann maliziös verhalten, wenn alle Sicherheitsmechanismen Javas aktiviert sind. Die von ihm entwickelten *Hostile Applets* machen sich den Umstand zu nutze, daß der Aspekt der Diensteverweigerungsangriffe in der Entwicklung der Sicherheitsarchitektur Javas keine Rolle spielte. Die Browser-Hersteller haben zwar im Nachhinein eine Reihe von Methoden entwickelt, mit denen einige Diensteverweigerungsangriffe erkannt und unterbunden werden können, das grundsätzliche Problem der Hostile Applets bleibt aber weiterhin bestehen.

LADUE hat die von ihm entwickelten Hostile Applets im Rahmen einer Web-Seite veröffentlicht.[1] Hier sind Agenten zu finden, die einen nicht endenden Ton produzieren, den Bildschirm des Rechners schwärzen oder die Ressourcen des Rechners belegen.

Die Abbildung A.7 enthält ein Code-Fragment des Hostile Applets *Noisy bear*. Das Fragment zeigt, daß der Agent die Methode `stop()` überschreibt und damit die vollständige Kontrolle über seine Lebenszeit erhält. Der Versuch den Agenten zu beenden führt lediglich dazu, daß der Agent den aktuellen Thread stoppt und anschließend sofort wieder startet. Noisy Bear beinhaltet damit eine Endlosschleife, die nur durch das Beenden des Browsers unterbrochen werden kann.

```
public void stop() {
     if (announce != null) {
     announce.stop();
     announce = null;
     }
 }
```

Abbildung A.7.: Noisy Bear nutzt die Möglichkeit, die `stop()` eines Agenten überschreiben zu können. Damit unterliegt es der Kontrolle des Agenten, wann er terminiert wird. Eine abschließende Terminierung durch den Anwender kann nur durch das Beenden des Browsers erreicht werden.

ScapeGoat nutzt die Funktionalität des Communicators aus, einen „weiteren" Browser zu starten. Der weitere gestartete Browser ist jedoch lediglich ein weiterer Thread innerhalb des Prozesses „Communicator". Im Falle des Internet Explorers beinhaltet das Öffnen eines weiteren Browser-Fensters hingegen den Start eines neuen Prozesses. Java-Agenten können aufgrund der Restriktionen der JVM keine neuen Prozesse kreieren, so daß der Agent ausschließlich in der Umgebung des Communicators funktioniert. Das Code-Fragement zeigt, daß die `run()`-Methode in regelmäßigen Abständen eine Ausnahme generiert. In der Ausnahmenahmebehandlung wird anschließend ein neues

[1] http://www.rstcorp.com/hostile-applets/index.html

Fensters erzeugt. Für den Nutzer bedeutet dies, daß eine nicht endende Anzahl von Communicator-Fenstern produziert werden.

```
public void run() {
    try {Thread.sleep(delay); }
    catch (InterruptedException ie) {}
    getAppletContext().showDocument(site, "_blank");
}
```

Abbildung A.8.: Der Agent ScapeGoat sorgt mit seiner Implementation der Methode `run()` dafür, daß in periodischen Abständen ein neues Fenster des Communicators erzeugt wird.

Das letzte Beispiel beinhaltet eine von LADUE entwickelte Hostile Application. Mit dieser Anwendung weist LADUE eindringlich darauf hin, daß die von THOMPSON vorgestellte Vision eines trojanisierten Compilers nur allzu real ist. *Hijacker* nutzt die Tatsache aus, daß der Java-Compiler im JDK durch ein Skript (`javac`) aufgerufen wird. Die Applikation manipuliert die Klasse des Compilers dahingehend, daß jeder kompilierten Klasse der String „Hijacked!“ beigefügt wird. Wie LADUE richtig hinweist, offenbart diese harmlose Trojanisierung des Compilers das Potential der Gefahren, die aus einem manipulierten Compiler resultieren. Hier wird erneut deutlich, daß es erforderlich ist, daß die Entwicklungsumgebung einschließlich des Entwicklungsprozesses den Sicherheitsansprüchen der zu entwickelnden Anwendung gerecht wird.

```
byte[] byter = new byte[hijacked.length];
  for (int i=0; i<hijacked.length; i++) {
      byter[i] = (byte)hijacked[i];
  }
  try {
      PrintStream jack=new PrintStream(new FileOutputStream(mainfile));
      jack.write(byter, 0, byter.length);
      jack.close();
  } catch (IOException ioe) {}
```

Abbildung A.9.: Das Code-Fragment liest das in der Application enthaltene Array aus, daß den Bytecode für die Klasse `Main` enthält. Es wird versucht, die Klasse `Main` in den Pfad zu schreiben, der durch das Skript `javac` ausgewertet wird. Gelingt dieses, wird nachfolgend jeder kompilierten Java-Klasse der String „Hijacked“ beigefügt.

A.3. Remote Explorer

Remote Explorer ist ein Virus, der sich auf den Windows NT-Plattformen Server und Workstation verbreitet[2]. Entdeckt wurde der Virus am 17. Dezember 1998 im Netzwerk des amerikanischen Telekommunikationsunternehmens MCI WorldCom. Nach Angaben von Network Associates (NAI) ist Remote Explorer der erste Virus, der sich selbstständig, ohne Intervention eines Nutzers, in einem Netzwerk fortpflanzt. Der Virus macht sich damit als erster Netzwerktechniken zu nutze, um seine Operationen durchzuführen. Remote Explorer ist kein Agent, er weist jedoch mit seiner eigenständigen Replikation in Netzwerken Merkmale auf, die ihn als Wurm ausweisen. Wie die Ausführungen im Kapitel 3 haben deutlich werden lassen, befindet er sich damit in enger Verwandschaft zu Agenten.

Der Virus installiert sich im Treiber-Verzeichnis des Betriebssystems. Hierzu erzeugt er eine Replikation von sich, kopiert diese in das Treiber-Verzeichnis und benennt diese als `IE403R.SYS`. Gleichzeitig installiert der Virus sich als Dienst unter dem Namen „Remote Explorer". Der Virus infiziert `EXE`-Dateien. Die Infektion von Programmen wird durch eine Komprimierung verschleiert. Der Virus führt hierfür eine Dynamic Linking Library (DLL) mit sich, die die Infektion und Komprimierung unterstützt. Dateien, die nicht infiziert werden können (`TXT`- und `HTML`-Dokumente) werden durch den Virus verschlüsselt [Net98d]. Neben der Verschlüsselung von Dateien beinhaltet der Virus keine weitere Schadensfunktion.

Bislang ist nicht eindeutig identifiziert, welche Mechanismen der Virus verwendet, um sich innerhalb eines Netzwerkes zu verbreiten. Erkannt wurde bislang, daß der Virus die Privilegien des Domain-Administrators hierfür verwendet, d.h., der Virus kann sich nur dann in einem Intranet verbreiten, wenn er unter der Identität des Administrators ausgeführt wird. Nur dann ist er auch in der Lage, sich als Systemdienst im Betriebssystem zu installieren. Anwender haben keine Berechtigung Systemdienste einzurichten.

Das Beispiel des Remote Explorers wurde in die Reihe der Fallbeispiele aufgenommen, um deutlich zu machen, daß neben der Gefahr, daß Viren in Agenten eingepflanzt werden, eine weitere, bislang unbeachtete existiert. Zukünftig können Viren sich Techniken zu Nutze machen, die in verteilten Systemen und insbesondere in Agentensystemen entwickelt worden sind, um sich in offenen bzw. geschlossenen Netzwerken zu replizieren.

A.4. Strange Brew

Strange Brew ist der erste Java-Virus, der Java Applets und Java Applications infizieren kann. Entdeckt wurde der Virus von der automatischen Virensuchmaschine „Seeker" des Antiviren-Herstellers Symantec. Die Infektion von Java Applets ist nach Aussage der Antiviren-Hersteller nahezu ausgeschlossen, da dem Virus die hierfür erforderlichen

[2] Auf den anderen Windows-Plattformen kann der Virus zwar existieren, eine Verbreitung ist von diesen jedoch nicht möglich [Net98d].

Rechte durch den Browser verweigert werden. Der infizierte Prozeß wird durch die JVM terminiert. Alle auf Java basierenden Anwendungen sind hingegen von der Infizierung durch den Virus bedroht.

Die Verbreitung von Strange Brew erfolgt somit im wesentlichen über Java Applications. Sobald der Virus die Kontrolle durch den Aufruf eines infizierten Programmes erhält, versucht er weitere Java-Klassen zu infizieren. Dateien, die vom Virus infiziert wurden, weisen eine Länge auf, die durch 101 teilbar ist. Somit sucht der Virus eine Klasse, die 101 nicht als Teiler hat. Basierend auf internen Kriterien entscheidet der Virus, ob eine gefundene Datei für die Infektion geeignet ist. Infizierte Java-Klassen sind weiterhin lauffähig [NCT98].

Die Infektion hat zur Folge, daß der Code des Virus in die zu infizierende Klasse eingefügt wird. Der Klassencode wird dahingehend manipuliert, daß die Programmkontrolle zunächst stets dem Viruscode übergeben wird. Abschließend wird eine Padding-Sequenz angefügt, die die Programmlänge derart erweitert, so daß diese durch 101 teilbar ist.

Nach dem Abschluß der Infizierung aller Java-Dateien im aktuellen Verzeichnis terminiert der Virus und die Kontrolle geht an die infizierte Java-Klasse über. Im Anschluß befindet der Virus sich nicht länger im Primärspeicher.

Der Prozeß der Infizierung weist eine Reihe von Mängeln auf, so daß das Ergebnis einer Infektion inkorrekt und die weitere Ausführung eines Java-Programmes unterbindet. Darüberhinaus beinhaltet der Virus keine intendierte Payload.

Strange Brew zählt bislang nicht zu den Viren, die in der Liste „In the wild“ zusammengefaßt werden und die jene Viren umfaßt, die aktuell häufig in Erscheinung treten.

Die Entdeckung von Java-Viren kann nicht mit den bisherigen Enigines erfolgen, die für herkömmliche Viren eingesetzt werden. Die Bekämpfung von Java-Viren erfordert speziell abgestimmtes Programm [Pat98].

Bereits LADUE hat im Rahmen seiner Hostile Applications einen Java-Virus entwickelt. Es zeigt sich, daß die „sichere“ Sprache Java über die Mittel verfügt, mit denen Viren entwickelt werden können. Für Agentensysteme, die auf Java basieren, ist damit die Gefahr gegeben, daß Komponenten der Systeme von Viren infiziert werden können. Die Gefahr der Vireninfektion kann von Agenten ausgehen oder gegen diese gerichtet sein.

Es ist nachdrücklich darauf hinzuweisen, daß Java-Viren durch die Plattformunabhängigkeit der Sprache eine neue Qualität der Gefahren aufweist. Die Plattformunabhängigkeit läßt die Infektion jedes Systems zu, für das eine JVM existiert.

B. Abkürzungsverzeichnis

ACL Agent Communication Language

AH Authentication Header

AMP Agent Meeting Points

AOP agentenorientierte Programmierung

API Application Programmable Interface

Ara Agents for Remote Actions

ARPA Advanced Research Projects Agency

ASCII American Standard Code for Information Interchange

ATP Agent Transfer Protocol

AWT Abstract Window Toolkit

BTX Bildschirmtext

CA Certification Authority

CAB Cabinet

CC Common Criteria

CERN Conseil Europeen pour la Recherche Nucleaire

CGI Common Gateway Interface

COM Component Object Model

CORBA Common Object Request Broker Architecture

CPU Central Processing Unit

CRL Certificate Revocation List

CryPO Cryptographically Protected Objects

DES Data Encryption Standard

DLL Dynamic Linking Library

DNS Domain Name System

DOS Disk Operating System

DTP Desktop Publishing

EEF Encrypted Functions

ESP Encapsulating Security Payload

FQN Full Qualified Name

FTP File Transfer Protocol

GUID Globally Unique Identifier

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IA Intelligente Agenten

ICMP Internet Control Message Protocol

IDS Intrusion Detection System

IP Internet Protocol

ISP Internet Service Provider

IRC Internet Relay Chat

IT Informationstechnologie

ITSEC Information Technology Security Evaluation Criteria

JAR Java Archive

JDK Java Development Kit

JRI Java Runtime Interface

JVM Java Virtual Machine

JECF Java Electronic Commerce Framework

JIT Just-In-Time

LF Logic Framework

KI Künstliche Intelligenz

KIF Knowledge Interchange Format

KQML Knowledge Query and Manipulation Language

MAC Message Authentication Code

MPL Mobile Programming Language

MTA Message Transfer Agent

MIME Multipurpose Internet Mail Extensions

NC Network Computer

NNTP Network News Transfer Protocol

NPL Network Programming Language

OCX Object Component Extension

OLE Object Linking and Embedding

ORB Object Request Broker

OSI Open System Interconnection

PC Personal Computer

PCC Proof-Carrying Code

PDA Personal Digital Assistant

PEM Personal Enhanced Mail

PGP Pretty Good Privacy

PIN Persönliche Identifikationsnummer

PKCS Public Key Cryptography Standards Group

PKI Public Key Infrastructure

PSC Physikalisch sicherer Co-Prozessor

RMI Remote Method Invocation

RP Remote Programming

RPC Remote Procedure Call

RSA Rivest Shamir Adleman

S-HTTP Secure Hypertext Transfer Protocol

SALTA Secure Agent Language with Tracing of Actions

SMTP Simple Mail Transfer Protocol

SRT Secure Request Technology

SSI Server Side Include

SSL Secure Sockets Layer

TACOMA Tromsø And COrnell Moving Agents

TAN Transaktionsnummer

TCB Trusted Computing Base

TCL Tool Command Language

TCP Transmission Control Protocol

TIE Trusted Internet Environment

Tk Toolkit

TPE Tamper Proof Environment

UDP User Datagram Protocol

URL Uniform Resource Locator

VCC Virtual Class Creator

WAIS Wide Area Information Server

WWW World-Wide Web

Literaturverzeichnis

[Ale97] Alexander, D. S.: A Generalized Computing Model of Active Networks / Distributed Systems Lab. University of Pennsylvania, Februar 1997. – Forschungsbericht. http://www.cis.upenn.edu/ salex/ps/proposal.ps

[Atk95] Atkinson, Randall: IP Authentication Header. August 1995. – Naval Research Laboratory

[Bat94] Bates, Joseph: The Role of Emotion in Believable Agents. In: *Communications of the ACM* 37 (1994), Juli, Nr. 7, S. 122–125. – http://www.cs.cmu.edu:8001/afs/cs.cmu.edu/project/oz/web/papers/ba-and-emotion.ps

[BDR+96] Blaze, Matt ; Diffie, Whitfield ; Rivest, Ronald L. ; Schneier, Bruce ; Shimomura, Tsutomu ; Thompson, Eric ; Wiener, Michael: Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security. 1996. – Forschungsbericht. ftp://ftp.research.att.com/dist/mab/keylength.ps

[Bel89] Bellovin, Steven M.: Security Problems in the TCP/IP Protocol Suite / AT&T Bell Laboratories. Murray Hill, April 1989. – Forschungsbericht. http://www.research.att.com/ smb/papers/ipext.ps

[Bel93] Bellovin, Steven M.: Packets found on an Internet. In: *Computer Communications Review* 23 (1993), Juli, Nr. 3, S. 26–31. – http://www.research.att.com/~smb/papers/packets.ps

[Bel95] Bellovin, Steven M.: Using the domain name system for system break-ins. In: *Proceedings of the Fifth Usenix UNIX Security Symposium*. Salt Lake City, USA, Juni 1995. – http://www.research.att.com/~smb/papers/dnshack.ps, S. 199–208

[Bel96] Bellovin, Steven M.: Problem Areas for the IP Security Protocols. In: *Proceedings of the Sixth Usenix UNIX Security Symposium*. San Jose, CA, Juli 1996. – ftp://ftp.research.att.com/dist/smb/badesp.ps

[BF97] Balfanz, Dirk ; Felten, Edward W.: A Java Filter / Department of Computer Science. Princeton University, Oktober 1997 (567-97). – Forschungsbericht. http://www.cs.princeton.edu/sip/pub/javafilter.html

[Bil96] Billon, Jean-Paul: Java Security: Weaknesses and Solutions. Dezember 1996. – http://www.dyade.fr/actions/VIP/JS_pap2.html

[BKS+97] Balasubramaniyan, Jai Sundar ; Krishnan, Ganesh ; Spafford, Eugene ; Stein, Karyl ; Sundaram, Aurobindo: Software Agents for Intrusion Detection / Department of Computer Science. Purdue University, Mai 1997. – Forschungsbericht

[BM91] Bellovin, Steven M. ; Merritt, Michael: Limitations of the Kerberos Authentication System. In: *Proceedings of the Winter 1991 Usenix Conference*. Dallas, Januar 1991. – ftp://research.att.com/dist/internet-security/kerblimit.usenix.ps

[Bor94] Borenstein, Nathaniel S.: EMail With A Mind of Its Own: The Safe-Tcl Language for Enabled Mail. In: *IFIP WG 6.5 Conference on Upper Layer Protocols, Architectures, and Applications (ULPAA)*. Barcelona, 1994. – http://minsky.med.Virginia.edu:80/sdm7g/Projects/Python/safe-tcl/

[BZW98] Brenner, Walter ; Zarnekow, Rüdiger ; Wittig, Hartmut: *Intelligente Softwareagenten*. 1. Springer-Verlag, 1998

[CGH+95] Chess, David ; Grosof, Benjamin ; Harrison, Colin ; Levine, David ; Parris, Colin ; Tsudik, Gene: Itinerant Agents for Mobile Computing. In: *IEEE Personal Communication Services Magazine* (1995), Oktober, Nr. 5, S. 34–49. – http://www.research.ibm.com/massive/rc20010.ps

[CGPV97] Cugola, Gianpaolo ; Ghezzi, Carlo ; Picco, Gian P. ; Vigna, Giovanni: Analyzing Mobile Code Languages. In: *Mobile Object Systems: Towards the Programmable Internet*. 1. Heidelberg, Deutschland : Springer-Verlag, April 1997 (Lecture Notes in Computer Science). – http://www.elet.polimi.it/~vigna/pub/lncs.ps.gz

[Cha97] Chang, Phil I.: Inside the Java Web Server - An Overview of Java Web Server 1.0, Java Servlets and the JavaServer Architecture. August 1997. – http://java.sun.com/features/1997/aug/jws1.html

[Che98] Chess, David M.: Security Issues in Mobile Code Systems. In: Vigna, Giovanni (Hrsg.): *Mobile Agents and Security*. 1. Heidelberg, Deutschland : Springer-Verlag, 1998 (Lecture Notes in Computer Science), S. 1–14

[CMRB96] Condict, Michael ; Milojičić, Dejan ; Reynolds, Franklin ; Bolinger, Don: Towards a World-Wide Civilization of Objects. In: *Proceedings of the 7th ACM SIGOPS European Workshop*. Connemara, Ireland, September 1996. – http://www.osf.org/RI/DMO/WebOs.ps

[Coh97] Cohen, Fred: Re: MS on the CCC ActiveX virus (RISKS-18.83). In: Neumann, Peter G. (Hrsg.): *Risks-Forum Digest* Bd. 18. 1997. – http://catless.ncl.ac.uk/Risks/18.84.html

[CS95] Crosbie, Mark ; Spafford, Eugene: Defending a Computer System using Autonomous Agents. In: *18th National Information Systems Security Conference*, 1995. – http://www.cs.purdue.edu/homes/mcrosbie/research/NISSC95.ps

[CW97] Campione, Mary ; Walrath, Kathy: *The Java Tutorial*. Addison-Wesley, Dezember 1997 (The Java Series). – http://java.sun.com/docs/books/tutorial/

[CZ95] Chapman, D. B. ; Zwicky, Elizabeth D.: *Building Internet Firewalls*. 2. O´Reilly & Associates, Inc., November 1995

[DEK97] Drossopoulou, Sophia ; Eisenbach, Susan ; Khurshid, Sarfraz: Is the Java Type System Sound? In: *Proceedings of the Fourth International Workshop on Foundations of Object-Oriented Languages*. Paris, France, Januar 1997. – http://www-ala.doc.ic.ac.uk/papers/S.Drossopoulou/JavaSoundJour.ps.Z

[Del96] Dellinger, Joe A.: Risks of web robots. In: Neumann, Peter G. (Hrsg.): *Risks-Forum Digest* Bd. 17. 1996. – http://catless.ncl.ac.uk/Risks/17.70.html

[DF97] Dean, Drew ; Felten, Edward W.: Secure Mobile Code: Where do we go from here? In: *DARPA Workshop on Foundations for Secure Mobile Code*. Monterey, CA, März 1997. – http://www.cs.nps.navy.mil/research/languages/statements/ddean.ps

[DFS98] Devanbu, P. ; Fong, P. ; Stubblebine, S.: Techniques for trusted software engineering. In: *Proceedings of the 20th International Conference on Software Engineering*. Kyoto, Japan, 1998. – http://www.research.att.com/~stubblebine/98icse.ps

[DFW96] Dean, Drew ; Felten, Edward W. ; Wallach, Dan S.: Java Security: From HotJava to Netscape and Beyond. In: *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, CA, Mai 1996. – http://www.cs.princeton.edu/sip/pub/secure96.html

[DFWB97] Kap. Java Security: Web Browsers and Beyond In: DEAN, Drew ; FELTEN, Edward W. ; WALLACH, Dan S. ; BALFANZ, Dirk: *Internet Beseiged: Countering Cyberspace Scofflaws*. New York : ACM Press, Oktober 1997. – http://www.cs.princeton.edu/sip/pub/internet-beseiged.html

[DS97] DEVANBU, Prem ; STUBBLEBINE, Stuart: Research directions for automated software verification: Using trusted hardware. In: *12th IEEE International Conference on Automated Software Engineering – ASE '97*. Incline Village, Nevada, USA, November 1997. – http://www.research.att.com/~stubblebine/97ase.ps

[DW95] DEAN, Drew ; WALLACH, Dan S.: Security Flaws in the HotJava Web Browser / Department of Computer Science. Princeton University, November 1995 (TR501). – Forschungsbericht. ftp://ftp.cs.princeton.edu/reports/1995/501.ps.Z

[Eic94] EICHMANN, D.: Ethical Web Agents. In: *Proceedings of the Second International World-Wide Web Conference*. Chicago, USA, Oktober 1994. – http://rbse.jsc.nasa.gov/eichmann/www-f94/ethics/ethics.html

[FBDW97] FELTEN, Edward W. ; BALFANZ, Dirk ; DEAN, Drew ; WALLACH, Dan S.: Web Spoofing: An Internet Con Game. In: *20th National Information Systems Security Conference*. Baltimore, Maryland, Oktober 1997. – http://www.cs.princeton.edu/sip/pub/spoofing.html

[Fel97] FELTEN, Edward: Myths about digital signatures. In: NEUMANN, Peter G. (Hrsg.): *Risks-Forum Digest* Bd. 18. 1997. – http://catless.ncl.ac.uk/Risks/18.83.html

[Fel98] FELTEN, Edward: More Java woes. In: NEUMANN, Peter G. (Hrsg.): *Risks-Forum Digest* Bd. 19. 1998. – http://catless.ncl.ac.uk/Risks/19.86.html

[FGS96] FARMER, W.M. ; GUTTAG, J.D. ; SWARUP, V.: Security for Mobile Agents: Authentification and State Appraisal. In: BERTINO, E. (Hrsg.) ; KURTH, H. (Hrsg.) ; MARTELLA, G. (Hrsg.) ; MONTOLIVO, E. (Hrsg.): *Proceedings of the Fourth ESORICS*. Rome, Italy, September 1996 (LNCS), S. 118–330

[FL97] FEIGENBAUM, Joan ; LEE, Peter: Trust Management and Proof-Carrying Code in Secure Mobile-Code Applications. In: *DARPA Workshop on Foundations for Secure Mobile Code*. Monterey, CA, März 1997. – http://www.cs.nps.navy.mil/research/languages/statements/leefei.ps

[Fle97] FLEGEL, Ulrich: *Sicherheitsaspekte in TCP/IP-Rechnernetzen*. Technische Universität Braunschweig, Institut für Betriebssysteme und

Rechnerverbund, Diplomarbeit, Juni 1997. – ftp://ftp.ibr.cs.tu-bs.de/pub/local/papers/sicherheit-flegel.ps.gz

[FM96] FRITZINGER, J. S. ; MUELLER, Marianne: Java Security / Sun Microsystems, Inc. 1996. – Forschungsbericht. http://java.sun.com/security/whitepaper.ps

[GAA+95] GILBERT, Don ; APARICIO, Manny ; ATKINSON, Betty ; BRADY, Steve ; CICCARINO, Joe ; GROSOF, Benjamin ; O'CONNOR, Pat ; OSISEK, Damian ; PRITKO, Steve ; SPAGNA, Rick ; WILSON, Les: The Role of Intelligent Agents in the Information Infrastrucure / IBM. USA, 1995. – Forschungsbericht. http://www.dcc.unicamp.br/ euzebio/Agents/eight-applications.html

[Gas88] GASSER, Morrie: *Building a secure computer system*. Van Nostrand Reinhold, 1988

[GB97] GRIMM, Robert ; BERSHAD, Brian N.: Security for Extensible Systems. In: *Proceedings of the 6th Workshop on Hot Topics in Operating Sytems (HotOS-VI)*. Cape Cod, Massachusetts, Mai 1997. – http://www.cs.washington.edu/homes/rgrimm/research/hotos97.ps, S. 62–66

[GHN97] GUNTER, Carl A. ; HOMEIER, Peter ; NETTLES, Scott: Infrastructure for Proof-Referencing Code. In: *DARPA Workshop on Foundations for Secure Mobile Code*. Monterey, California, März 1997. – http://www.cis.upenn.edu/~gunter/wip/fsmc.ps.Z

[GM95] GOSLING, James ; MCGILTON, Henry: The Java Language Environment: A White Paper. Oktober 1995. – ftp://ftp.javasoft.com/docs/papers/langenviron-a4-ps.tar.Z

[GMPS97] GONG, Li ; MUELLER, Marianne ; PRAFULLCHANDRA, Hemma ; SCHEMERS, Roland: Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2. In: *Proceedings of the USENIX Symposium on Internet Technologies and Systems*. Monterey, California, Dezember 1997. – http://java.sun.com/people/gong/papers/jdk12arch.ps.gz

[Gon98] GONG, Li: *Java Security Architecture (JDK 1.2)*. Version 1.0. Palo Alto: Sun Microsystems, Inc., Oktober 1998. – http://java.sun.com/people/gong/papers/jdk12spec.pdf

[Gre98] GREANIER, Todd M.: Building a bigger sandbox. In: *JavaWorld* 3 (1998), August, Nr. 8. – http://www.javaworld.com/jw-08-1998/jw-08-sandbox.html

[GS96] GARFINKEL, Simson ; SPAFFORD, Gene ; RUSSELL, Deborah (Hrsg.): *Practical Unix and Internet Security*. O´Reilly & Associates, Inc., 1996

[GS97] GARFINKEL, Simson ; SPAFFORD, Gene: *Web Security & Commerce*. 1. O´Reilly & Associates, Inc., Juni 1997

[Her96] HERMANS, Björn: Intelligent Software Agents on the Internet: an inventory of currently offered functionality in the information society & a prediction of (near-)future developments / Tilburg University. The Netherlands, Juli 1996. – Forschungsbericht. http://www.hermans.org/agents

[Hoh97] HOHL, Fritz: An Approach to Solve the Problem of Malicious Hosts / Institut für Parallele und Verteilte Höchstleistungsrechner. Universität Stuttgart, März 1997 (1997/03). – Forschungsbericht. ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-1997-03/TR-1997-03.ps.gz

[Hoh98] HOHL, Fritz: Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In: VIGNA, Giovanni (Hrsg.): *Mobile Agents and Security*. 1. Heidelberg, Deutschland : Springer-Verlag, 1998 (Lecture Notes in Computer Science), S. 92–113

[Hop97a] HOPWOOD, David: A Comparison between Java and ActiveX Security. In: *Compsec ´97*, 1997. – http://www.users.zetnet.co.uk/hopwood/papers/compsec97.html

[Hop97b] HOPWOOD, David: Re: Comments and corrections on Authenticode (Atkinson, RISKS-18.85). In: NEUMANN, Peter G. (Hrsg.): *Risks-Forum Digest* Bd. 18. 1997. – http://catless.ncl.ac.uk/Risks/18.86.html

[IBM98] IBM: *Internet Security in the Network Computing Framework*. 1998 (Redbooks). – http://www.redbooks.ibm.com/SG245220/sg245220.html

[Joh96] JOHNS, Paul: *Signing and Marking ActiveX Controls*. Microsoft, Oktober 1996

[JRS95] JOHANSEN, Dag ; RENESSE, Robbert van ; SCHNEIDER, Fred B.: Operating System Support for Mobile Agents. In: *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems (HTOS)*, 1995. –

http://cs-tr.cs.cornell.edu/Dienst/Repository/2.0/Body/ncstrl.cornell/TR94-1468/postscript, S. 42–45

[Ken91] KENT, Stephen: RFC1108: Security Options for the Internet Protocol. November 1991. – BBN Communications

[KLO97] KARJOTH, Günther ; LANGE, Danny B. ; OSHIMA, Mitsuru: A Security Model for Aglets. In: *IEEE Internet Computing* (1997), Juli, Nr. July/August, S. 68–77. – http://computer.org/internet/ic1997/w4068abs.htm

[KNT91] KOHL, John T. ; NEUMAN, B. C. ; TS´O, Theodore Y.: The Evolution of ther Kerberos Authentication Service. In: *Spring 1991 EurOpen Conference*. Tromsø, Norway, 1991. – ftp://athena-dist.mit.edu/pub/kerberos/doc/krb_evol.PS

[Kok97] KOKE, Andreas: Java und ActiveX – Gefahr aus dem Internet. In: *5. Deutschen IT-Sicherheitskongreß des BSI 1997* Bundesamt für Sicherheit in der Informationstechnik, 1997. – http://www.bsi.de/literat/tagungsb/koke.htm

[Kos95a] KOSTER, Martijn: Robots in the Web: threat or treat? April 1995. – http://info.webcrawler.com/mak/projects/robots/threat-or-treat.html

[Kos95b] KOSTER, Martijn: A Standard for Robot Exclusion. 1995. – http://info.webcrawler.com/mak/projects/robots/norobots.html

[LaD97] LADUE, Mark D.: When Java was One: Threats from Hostile Byte Code and Java Platform Viruses. 1997. – Forschungsbericht. http://www.rstcorp.com/hostile-applets/final_wjw1.ps

[LaD98] LADUE, Mark: Problems with Netscape's Communicator 4.04 and 4.05. 1998. – http://www.rstcorp.com/hostile-applets/Com4Applets.html

[Lam97] LAMINACK, Brent: ActiveX security? TISK, TISK. In: NEUMANN, Peter G. (Hrsg.): *Risks-Forum Digest* Bd. 18. 1997. – http://catless.ncl.ac.uk/Risks/18.86.html

[LC96] LANGE, Danny B. ; CHANG, Daniel T.: IBM Aglets Workbench - Programming Mobile Agents in Java. September 1996. – http://www.trl.ibm.co.jp/aglets/whitepaper.htm

[LD96] LINGNAU, Anselm ; DROBNIK, Oswald: Making Mobile Agents Communicate: A Flexible Approach. In: *etaCOM '96*. Johann Wolfgang Goethe-Universität, Frankfurt am Main, Germany, 1996. – ftp://ftp.tm.informatik.uni-frankfurt.de/pub/papers/agents/etacom96-paper.ps.gz

[LN97] LEE, Peter ; NECULA, George: Research on Proof-Carrying Code for Mobile-Code Security. In: *DARPA Workshop on Foundations for Secure Mobile Code*. Monterey, CA, März 1997. – http://www.cs.nps.navy.mil/research/languages/statements/necula.ps

[LO98] LANGE, Danny B. ; OSHIMA, Mitsuru: Mobile Agents with Java: The Aglet API / General Magic Inc. Sunnyvale, USA, 1998. – Forschungsbericht. http://www.genmagic.com/technology/danny.html/Wwwj.pdf

[LY97] LINDHOLM, Tim ; YELLIN, Frank: *The Java Virtual Specification*. Reading, MA : Addison-Wesley, 1997

[Mar97a] MARIMBA: Crossing the Firewall: Securing Internet Software Distribution. 1997. – http://www.marimba.com/datasheets/security-wp.html

[Mar97b] MARIMBA. Introducing the Castanet 2.0 Software Distribution System. Oktober 1997

[Mar97c] MARIMBA: Using Security Certificates. Juli 1997. – Version 8.0.4308

[Mar97d] MARTIENSSEN, Marco: Sicherheitsaspekte von Internet-Agenten / Arbeitsbereich für Anwendungen der Informatik in Geistes- und Naturwissenschaften. Universität Hamburg, September 1997. – Forschungsbericht

[Mey98] MEYER, Betrand: Here is what I am not telling you. In: NEUMANN, Peter G. (Hrsg.): *Risks-Forum Digest* Bd. 19. 1998. – http://catless.ncl.ac.uk/Risks/19.54.html

[MF97] MCGRAW, Gary ; FELTEN, Edward: Understanding the keys to Java security – the sandbox and authentication. In: *JavaWorld* 2 (1997), Mai, Nr. 5. – http://www.javaworld.com

[MGW97] MILOJIČIĆ, Dejan S. ; GUDAY, Shai ; WHEELER, Richard: Old Wine in New Bottles – Applying OS Process Migration Technology to Mobile Agents / The Open Group Research Institute. Cambridge, April 1997. – Forschungsbericht. http://www.camb.opengroup.org/RI/java/moa/Finland.fr.ps

[Mic96] MICROSOFT. Microsoft Authenticode Technology. Oktober 1996

[Mic97a] MICROSOFT: Certificate Management. Juli 1997. – http://www.microsoft.com/ie/ie40/features/sec-certman.htm

[Mic97b] MICROSOFT: Developer FAQ for Java Code Signing in Microsoft Internet Explorer 4.0. Oktober 1997. – http://www.microsoft.com/java/sdk/20/tools/signcode.htm

[Mic98a] MICROSOFT: Signing a Cabinet File with Java Permissions Using Signcode. 1998. – http://www.microsoft.com/java/sdk/20/tools/signcode.htm

[Mic98b] MICROSOFT: Trust-Based Security for Java. 1998. – http://www.microsoft.com/java/sdk/20/articles/trust_based_security.htm

[Möl99] MÖLLER, Ulf: *Anonymisierungsverfahren für WWW-Zugriffe*. Universität Hamburg, Arbeitsbereich für Anwendungen der Informatik in Geistes- und Naturwissenschaften, Diplomarbeit, 1999

[MM97] MUELLER-MAGUHN, Andy. Mail. März 1997

[Mor85] MORRIS, Robert T.: A Weakness in the 4.2BSD Unix TCP/IP Software / AT&T Bell Laboratories. Murray Hill, New Jersey, Februar 1985. – Forschungsbericht

[MR97] MALKHI, Dahlia ; REITER, Michael: Unreliable Intrusion Detection in Distributed Computations. In: *Proceedings of the 10th Computer Security Foundations Workshop*. AT&T Bell Laboratories, Juni 1997. – http://www.research.att.com/~dalia/pubs/fc97-ftp.ps.gz

[MRR97] MARTIN, David M. ; RAJAGOPALAN, Sivaramakrishnan ; RUBIN, Aviel D.: Blocking Java Applets at the Firewall. In: *Proceedings of the 1997 Symposium on Network and Distributed Systems Security*, 1997. – http://www.cs.nyu.edu/cgi-bin/cgiwrap/~rubin/block.ps, S. 16–26

[Mur98] MURPHY, Kieron: First big bites Java 1.2 – Mozilla stung. In: *JavaWorld* 3 (1998), August, Nr. 8. – http://www.javaworld.com/javaworld/jw-00-1998/jw-00-12bug.html

[NCT98] NACHENBERG, Carey ; CHIEN, Eric ; TRILLING, Stephen: JavaApp.Strange Brew. August 1998. – http://www.symantec.com/avcenter/data/javaapp.strangebrew.html

[Net97] NETSCAPE: Netscape Object Signing: Establishing Trust for Downloaded Software. Juli 1997. – http://developer.netscape.com/library/documentation/signedobj/trust/index.htm

[Net98a] NETSCAPE: Introduction to the Capabilities Classes. Juni 1998. – http://developer.netscape.com/docs/manuals/signedobj/capabilities/index.html

[Net98b] NETSCAPE: Netscape Security Features Evaluation Guide – Competitive Comparison. 1998. – http://home.netscape.com/assist/security/resources/eval_guide/compare.html

[Net98c] NETSCAPE: Netscape Security Features Evaluation Guide – Key Advantages. 1998. – http://home.netscape.com/assist/security/resources/eval_guide/advantages.html

[Net98d] NETWORK ASSOCIATES: Remote Explorer. Dezember 1998. – http://www.nai.com/products/antivirus/remote_explorer.asp

[Neu98] NEUROTEC: Analyse der Risiken ausführbarer Web-Contents / im Auftrage des BSI. 1998. – Forschungsbericht. http://www.bsi.de/aktuell/ococat.zip

[Obj97] OBJECTSPACE: *The Agent ORB for Java*. ObjectSpace, Inc., September 1997

[Opp92] OPPLIGER, Rolf: *Computersicherheit: eine Einführung*. 1. Braunschweig/Wiesbaden : Vieweg, 1992

[Ord96] ORDILLE, Joann J.: When agents roam, who can you trust? In: *Proceedings of the First Conference on Emerging Technologies and Applications in Communications*. Portland, Mai 1996. – http://cm.bell-labs.com/cm/cs/doc/96/5-09.ps.gz

[Ous95] OUSTERHOUT, J.: Scripts and Agents: The New Software High Ground. 1995. – http://www.scriptics.com/people/john.ousterhout/agent.ps

[Pat98] PATRIZIO, Andy: First Virus Written In Java Found. August 1998. – http://www.techweb.com/wire/story/TWB19980820S0010

[Pfl89] PFLEEGER, Charles: *Security in Computing*. 1. New Jersey, USA : Prentice-Hall, Inc., 1989

[PSK97] PAOLI, Flavio D. ; SANTOS, Andre L. D. ; KEMMERER, Richard A.: Vulnerability of "secure" Web Browsers. In: *Proceedings of the National Information Systems Security Conference*. University of California, Oktober 1997. – http://www.cs.ucsb.edu/~andre/nissc97.html

[PSK98] PAOLI, Flavio D. ; SANTOS, Andre L. D. ; KEMMERER, Richard A.: Web Browsers and Security. In: VIGNA, Giovanni (Hrsg.): *Mobile Agents and Security*. 1. Heidelberg, Deutschland : Springer-Verlag, 1998 (Lecture Notes in Computer Science), S. 235–256

[RDEG98] Rubin, Aviel D. ; Daniel E. Geer, Jr.: Mobile Code Security. In: *IEEE Internet Computing* (1998), Nr. November/December. – http://www.cs.nyu.edu/ rubin/ieee.ic.ps.gz

[RJ96] Rasmusson, Lars ; Jansson, Sverker: Simulated Social Control for Secure Internet Commerce / Swedish Institute of Computer Science. 1996. – Forschungsbericht. http://www.sics.se/~sverker/public/papers/nsp96lra.pdf

[RRJ97] Rasmusson, Lars ; Rasmusson, Andreas ; Janson, Sverker: Using Agents to Secure the Internet Marketplace – Reactive Security and Social Control. In: *Proceedings of the PAAM´97*. London, Großbritannien, April 1997. – http://www.sics.se/~lra/papers/paam97/security.ps.gz

[RSG98] Reed, Michael G. ; Syverson, Paul F. ; Goldschlag, David M.: Anonymous Connections and Onion Routing. In: *IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection* (1998). – http://www.onion-router.net/Publications/JSAC-1998.pdf

[Sar97] Saraswat, Vijay: Java is not type-safe. 1997. – http://www.research.att.com/~vj/bug.html

[Sch93] Schuba, Christoph L.: *Addressing Weaknesses in the Domain Name System Protocol*. Purdue University, COAST Laboratory, Diplomarbeit, August 1993. – http://www.cs.purdue.edu/homes/spaf/tech-reps/9428.ps

[Sch97] Schneider, Fred B.: Towards Fault-tolerant and Secure Agentry. In: *11th International Workshop on Distributed Algorithms*. Saarbrücken, Deutschland, September 1997. – http://cs-tr.cs.cornell.edu:80/Dienst/Repository/2.0/Body/ncstrl.cornell%2fTR97-1636/postscript

[SH82] Shoch, John F. ; Hupp, John A.: The Worm Programs – Early Experience with a Distributed Computation. In: *Communications of the ACM* 25 (1982), März, Nr. 3, S. 172–180

[Sib96] Sibert, W. O.: Malicious Data and Computer Security / InterTrust Technologies Corporation. 1996. – Forschungsbericht. http://www.trouble.org/security/maldata.html

[SKK+96] Schuba, Christoph L. ; Krsul, Ivan V. ; Kuhn, Markus G. ; Spafford, Eugene H. ; Sundaram, Aurobindo ; Zamboni, Diego: Analysis of a Denial of Service Attack on TCP / Department of Computer Science. Purdue University, 1996. – Forschungsbericht.

ftp://coast.cs.purdue.edu/pub/COAST/papers/christoph-schuba/schuba-krsul-kuhn-spaf-sundaram-zamboni-synkill.ps.Z

[Smi96] SMITH, Richard M.: Making good ActiveX controls do bad things. In: NEUMANN, Peter G. (Hrsg.): *Risks-Forum Digest* Bd. 18. 1996. – http://catless.ncl.ac.uk/Risks/18.61.html

[Smi98] SMITH, Richard M.: ActiveX Controls – You just can´t say no! In: NEUMANN, Peter G. (Hrsg.): *Risks-Forum Digest* Bd. 19. 1998. – http://catless.ncl.ac.uk/Risks/19.55.html

[Som92] SOMMERVILLE: *Software Engineering*. 4. Addison-Wesley, 1992

[Spa88] SPAFFORD, Eugene H.: The Internet Worm Program: An Analysis / Department of Computer Science. Purdue University, Dezember 1988 (CSD-TR-823). – Forschungsbericht. http://www.cs.purdue.edu/homes/spaf/tech-reps/823.ps

[Spa91] SPAFFORD, Eugene H.: The Internet Worm Incident / Department of Computer Science. Purdue University, September 1991 (CSD-TR-933). – Forschungsbericht. http://www.cs.purdue.edu/homes/spaf/tech-reps/933.ps

[SRG97] SYVERSON, Paul F. ; REED, Michael G. ; GOLDSCHLAG, David M.: Private Web Browsing. In: *Journal of Computer Security Special Issue on Web Security* 5 (1997), Nr. 3, S. 237–248. – http://www.onion-router.net/Publications/JCS-1997.pdf

[SS75] SALTZER, Jerome H. ; SCHROEDER, Michael D.: The Protection of Information in Computer Systems. In: *Proceedings of the IEEE* Bd. 63. September, 1975. – http://www.mediacity.com/ norm/CapTheory/ProtInf/, S. 1278–1308

[ST97a] SANDER, Tomas ; TSCHUDIN, Christian: Towards Mobile Cryptograpghy / International Computer Science Institute. Berkeley, California, November 1997 (TR-97-049). – Forschungsbericht. http://www.icsi.berkeley.edu/ sander/publications/tr-97-049.ps

[ST97b] SANDER, Tomas ; TSCHUDIN, Christian F.: Protecting Mobile Agents Against Malicious Hosts. In: *LNCS on "Mobile Agent Security"*, 1997. – http://www.icsi.berkeley.edu/ sander/publications/MA-protect.ps

[Str97] STROBEL, Stefan: *Firewalls für das Netz der Netze: Sicherheit im Internet: Einführung und Praxis*. 1. Heidelberg : dpunkt-Verlag, 1997

[SUN96] SUN. The Java Servlet API. 1996

[Tha93] THALLER, Georg E.: *Computersicherheit*. 1. Braunschweig/Wiesbaden : Vieweg, 1993 (DUD-Fachbeiträge 18)

[Tho84] THOMPSON, Ken: Reflections on Trusting Trust. In: *Communications of the ACM* 27 (1984), August, Nr. 8, S. 761–763. – http://www.acm.org/classics/sep95/

[Tol97] TOLKSDORF, Robert: Programming Languages for the Java Virtual Machine. Dezember 1997. – http://grunge.cs.tu-berlin.de/~tolk/vmlanguages.html

[TW96] TENNENHOUSE, David L. ; WETHERALL, David J.: Towards an Active Network Architecture. In: *Computer Communication Review* 26 (1996), April, Nr. 2. – http://www.tns.lcs.mit.edu/publications/ccr96.html

[Ven97a] VENNERS, Bill: Java security: How to install the security manager and customize your security policy. In: *JavaWorld* 2 (1997), November, Nr. 11. – http://www.javaworld.com/javaworld/jw-11-1997/jw-11-hood.html

[Ven97b] VENNERS, Bill: Java´s security architecture. In: *JavaWorld* 2 (1997), August, Nr. 8. – http://www.javaworld.com

[Ven97c] VENNERS, Bill: Security and the class loader architecture. In: *JavaWorld* 2 (1997), September, Nr. 9. – http://www.javaworld.com

[Ven97d] VENNERS, Bill: Security and the class verifier. In: *JavaWorld* 2 (1997), Oktober, Nr. 10. – http://www.javaworld.com/javaworld/jw-10-1997/jw-10-hood.html

[Vig97] VIGNA, Giovanni: Protecting Mobile Agents through Tracing. In: *Mobile Object Systems ECOOP Workshop'97*, 1997. – http://www.elet.polimi.it/people/vigna/mos97.ps.gz

[VS97] VOLPANO, Dennis ; SMITH, Geoffrey: Language Issues in Mobile Program Security / Department of Computer Science. 1997. – Forschungsbericht. http://www.cs.nps.navy.mil/research/languages/papers/atsc/lncs98.ps.Z

[Wal99] WALLACH, Dan S.: *A New Approach to Mobile Code Security*. University of Princeton, Department of Computer Science, Diplomarbeit, Januar 1999. – http://www.cs.princeton.edu/sip/pub/dwallach-thesis.pdf

[WBDF97] WALLACH, Dan S. ; BALFANZ, Dirk ; DEAN, Drew ; FELTEN, Edward W.: Extensible Security Architectures for Java. In: *Proceedings of the 16th ACM Symposium on Operating System Principles*. Princeton University, Oktober 1997. – http://www.cs.princeton.edu/sip/pub/sosp97.html

[WBS98] WILHELM, U.G. ; BUTTYÁN, L. ; S.STAAMANN: On the problem of trust in mobile agent systems. In: *Symposium on Network and Distributed System Security* Internet Society, 1998. – http://lsewww.epfl.ch/Documents/postscript/WBS98.ps

[WF98] WALLACH, Dan S. ; FELTEN, Edward W.: Understanding Java Stack Inspection. In: *Proceedings of 1998 IEEE Symposium on Security and Privacy*. Oakland, California, Mai 1998. – http://www.cs.princeton.edu/sip/pub/oakland98.html

[Whe97] WHEELGROUP: ProWatch Secure Network Security Survey. 1997. – http://www.wheelgroup.com/netrangr/PWS_survey.html

[Wil97a] WILHELM, Uwe G.: Cryptographically Protected Objects / Ecole Polytechnique Fédérale de Lausanne. Lausanne, Switzerland, 1997. – Forschungsbericht. http://lsewww.epfl.ch/~wilhelm/Papers/CryPO.ps.gz

[Wil97b] WILHELM, Uwe G.: Increasing privacy in mobile communication systems using cryptographically protected objects. In: *Verläßliche IT-Systeme*. Freiburg, Deutschland, November 1997. – http://lsewww.epfl.ch/Documents/postscript/Wil97.ps, S. 319–334

[Yee97] YEE, Bennet S.: A Sanctuary for Mobile Agents. In: *DARPA Workshop on Foundations for Secure Mobile Code*. Monterey, CA, März 1997. – http://www.cs.nps.navy.mil/research/languages/statements/bsy.ps