Vertrauensbasierte Sicherheit für mobile Objekte

Gregor Gärtner, Winfried E. Kühnhauser

Technische Universität Ilmenau
Mail@GGaertner.de, winfried.kuehnhauser@prakinf.tu-ilmenau.de

Zusammenfassung

Der Beitrag diskutiert die Tragfähigkeit vertrauensbasierter Sicherheitspolitiken für mobile Objekte und ihre Gastgebersysteme, zeigt die zur Bildung von Vertrauen geeigneten Objekt- und Systemeigenschaften auf und stellt eine einfache, über dem Vertrauensbegriff definierte Algebra zur Beschreibung vertrauensbasierter Sicherheitspolitiken vor. Am Beispiel der aktuellen Java Sicherheitsarchitektur (JDK 1.2) wird die Abbildung vertrauensbasierter Sicherheitspolitiken auf die elementaren Mechanismen einer konkreten Sicherheitsarchitektur demonstriert.

Schlüsselbegriffe: Vertrauen, mobile Objekte, mobiler Code, Sicherheitspolitik, Sicherheitsarchitektur, Java, E-Commerce, IT-Sicherheit, Internet.

1 Einleitung

Eine Vielzahl heutiger Strukturierungsparadigmen verteilter Systeme beruht auf einem dualen Client/Server - Rollenmodell, in welchem Systemkomponenten in der Rolle von Klienten Dienstleistungen anderer Systemkomponenten in Anspruch nehmen, die in einer Rolle als Server agieren. Dieses Modell stammt noch aus einer Zeit, in der behutsam die bekannten Kommunikationstechniken nicht verteilter Systeme an die Gegebenheiten verteilter Systeme adaptiert wurden und aus einfachen Prozedur- und Methodenaufrufen die Prozedur- und Methodenfernaufrufe wurden. Implizit wurde dabei auch gleich das Prinzip übernommen, das *Wissen* zur Erbringung einer Dienstleistung (der Algorithmus) in enge örtliche Beziehung zu den hierzu notwendigen *Ressourcen* (Prozessoren, Speicher, Kommunikationswege) zu setzen.

Inzwischen werden die Grenzen des Wachstums dieses Ansatzes deutlich. Der wachsende Funktionsumfang von Anwendungssystemen und die wachsende Zahl ihrer Dienste führt zu immer größeren Anzahlen an Servern und immer umfangreicherer Serverfunktionalität. Die klassischen Skalierungsprobleme der softwaretechnischen und administrativen Beherrschung großer verteilter Systeme treten hier deutlich zutage.

Als ein erfolgversprechender Weg zu besser skalierenden Ansätzen wird derzeit die Aufhebung der traditionellen Verschmelzung von Wissen und Ressourcen gesehen, indem beide Größen – Wissen und Ressourcen – als orthogonal zueinander betrachtet werden. Wird eine Dienstleistung benötigt, so ist es zunächst notwendig, das hierzu notwendige Wissen und die notwendigen Ressourcen zusammenzuführen, und dies erfordert nun – da Ressourcen im Allgemeinen weniger mobil sind als Wissen – die Kommunikation von Wissen. Wissenskommunikation treffen wir heute in Form mobiler Objekte oder mobilen Codes an.

Als Ausführungsplattformen für mobile Objekte sind in der heutigen IT-Landschaft virtuelle Java-Maschinen durch ihre Verbreitung im Kontext von Web-Browsern wohletabliert. Sie bilden die Grundlage einer Vielzahl jüngerer Internet-basierter E-Commerce- und E-Banking-Systeme, welche die Mobilität von Java-Klassen dazu nutzen, Wissen in Form von Algorithmen (bspw. Authentisierungsprotokolle oder Verschlüsselungsalgorithmen) auf Klientenmaschinen zu transportieren.

Bedingt durch diese Form industrieller Nutzung enthielt bereits die erste Generation virtueller Java-Maschinen Sicherheitskonzepte in Form von Separationsmechanismen (die „*sandbox*“), welche die Risiken der Ausführung fremder Java-Klassen für den Hostrechner begrenzen sollten. Die Starrheit dieser Mechanismen, ihre anschließende Flexibilisierung sowie ein steter Strom der Erkennung und Beseitigung von Sicherheitslücken bedingen derzeit einen evolutionären Prozess, der in der nunmehr dritten Generation der Java-Sicherheitsarchitektur [GMPS97] zu einer Fülle von Sicherheitsmechanismen geführt hat, die während des Ladens einer Java-Klasse (Abstammungsprüfung, Integritätsprüfung, Bytecode-Prüfung, Beweis der Konformität mit einer Sicherheitspolitik) oder während ihrer Ausführung (Sicherheitspolitiken, Sicherheitsdomänen, Zugriffsberechtigungen, Zugriffsobjekte, Zugriffs- und Ressourcenkonsumüberwachung) zur Anwendung kommen [Gon98].

Inzwischen werden jedoch auch die Grenzen dieser Entwicklung offensichtlich. Die Vielzahl der sichtbaren und nutzbaren Sicherheitskonzepte und -mechanismen konterkariert eines der elementarsten Prinzipien sicherer IT-Systeme: die Klarheit, Einfachheit und intellektuelle Beherrschbarkeit von Sicherheitspolitik, -mechanismen und -architektur. Die Vielzahl der Elemente der heutigen Java-Sicherheitsarchitektur und die Komplexität ihres Zusammenwirkens machen es sehr problematisch, zeitaufwendig und kostenintensiv, präzise Aussagen über die Einhaltung konkreter Sicherheitsanforderungen in einer auf mobilen Java-Klassen basierenden verteilten Applikation zu machen, geschweige denn ist es praktikabel, solche Aussagen einem formalen Beweisverfahren zu unterziehen. Den Verfassern ist kein einziger gelungener Versuch bekannt, beispielsweise durch eine erfolgreiche *Covert-Channel*-Analyse für ein konkretes verteiltes Anwendungssystem die Einhaltung eines Zugriffssteuerungsmodells in einer Java-Umgebung nach-

zuweisen. Für die Hersteller und Betreiber Internet-basierter E-Commerce- und E-Banking-Systeme hat dies zur Folge, dass trotz erheblicher Investitionen in Design, Analyse und Implementierung der IT-Sicherheitseigenschaften ein unkalkulierbarer Unsicherheitsfaktor verbleibt.

Auf der anderen Seite sind die Elemente der heutigen Java-Sicherheitsarchitektur wohlbegründet: Jedes Element ist rückführbar auf eine konkrete reale Bedrohung. Während also einerseits die Vielzahl der Elemente und ihr komplexes Zusammenwirken die Grenzen des Wachstums sichtbar werden lässt, ist andererseits ihre jeweilige Aufgabe unverzichtbar.

Dieser Beitrag setzt sich mit diesem Konflikt auseinander und geht dabei den Weg, Sicherheitspolitiken auf sehr hoher und anwendungsnaher Abstraktionsebene zu formulieren. Der Schlüsselbegriff hierbei ist *Vertrauen*; hierunter wird die Annahme verstanden, dass eine bestimmte Einheit (beispielsweise ein mobiles Objekt oder ein Gastgebersystem) sich in einer erwarteten, wohl definierten Form verhält. Auf Vertrauen basierende Sicherheitskonzepte stellen ein orthogonales Konzept zu jenen dar, die über aktive Maßnahmen Sicherheit für potentiell nicht vertrauenswürdige Einheiten bieten. Wir untersuchen, welche Möglichkeiten und Perspektiven auf Vertrauen basierende Sicherheitskonzepte für mobilen Code oder mobile Objekte besitzen; Techniken also, die bereits heute im E-Commerce-Bereich eingesetzt werden, um kunden- und applikationsspezifisches Wissen in Form von Algorithmen zu transportieren. Dabei erachtet unser Ansatz neben der Sicherheit des Gastgebersystems die Sicherheit des mobilen Objektes als gleichwertig; hierdurch wird der Transport sensitiver Informationen innerhalb mobiler Objekte (Bilanzen, Transaktionsdaten, Kreditkarteninformationen) vertrauenswürdig und erschließt damit ein weites Feld neuer Anwendungen. Wir schließen unseren Beitrag mit einem Beispiel, welches die Realisierung einer vertrauensbasierten Sicherheitspolitik in der Java-Sicherheitsarchitektur demonstriert.

2 Vertrauensbegründende Eigenschaften

Dieser Abschnitt setzt sich mit den Eigenschaften von mobilen Objekten und Gastsystemen auseinander, die zum Aufbau von Vertrauensbeziehungen nötig sind. Dabei stellen wir zwei Fragen in den Vordergrund: *Welche Eigenschaften* eines mobilen Objekts/Gastsystems sind signifikant um Vertrauen aufzubauen? *Wer* stellt diese Eigenschaften sicher?

Die Signifikanz der Eigenschaften lassen sich in zwei Klassen einteilen: Die Eigenschaften der ersten Ordnung sind im Regelfall für jedes mobile Objekt oder Gastsystem von Bedeutung, während die Eigenschaften der zweiten Ordnung nur in bestimmten Anwendungsfällen zu berücksichtigen sind. Die Aussage, welche Eigenschaften bei einem Objekt oder einem Gastsystem vorhanden sind, wird mit Hilfe von Etiketten getroffen, die einem mobilen Objekt oder Gastsystem mittels kryptographischer Verfahren in verbindlicher Weise angeheftet werden.

2.1 Beispiel

Zunächst sollen die Eigenschaften an einem konkreten Beispiel veranschaulicht werden. Das Szenario besteht aus einem Großunternehmen, dessen Mitarbeiter eine Java Entwicklungsumgebung (IDE) gemeinsam mit einer Schulung für die neue Programmiersprache erwerben möchte. Während der Mitarbeiter kleinere Beträge unmittelbar aus der Abteilungskasse zur Verfügung hat, muss für größere Summen ein Beschaffungsantrag beim Einkauf eingereicht werden. Möchte ein Mitarbeiter an einer Schulung teilnehmen, muss er dies bei der zuständigen Personalabteilung beantragen. Um das Beispiel zu vereinfachen, soll davon ausgegangen werden, dass der Mitarbeiter die IDE bereits ausgewählt hat und diese nun möglichst günstig erwerben möchte. Die Vorgänge sollen durch ein mobiles Objekt mit der typischen Funktionalität eines mobilen Agenten erledigt werden. Folgende Schritte sind in Abbildung 1 zu sehen:

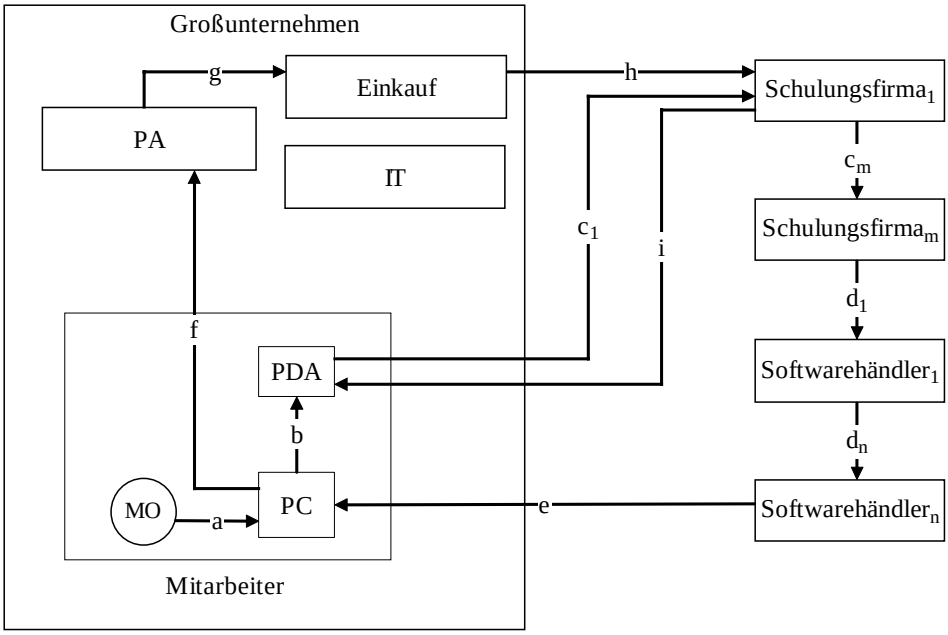

Abbildung 1: Erwerb einer IDE und Schulung

a) Das mobile Objekt (MO) wird am PC erstellt. Es enthält Informationen, welches Produkt erworben werden soll sowie die Kriterien für die Auswahl einer geeigneten Schulung. Da die Kosten für die IDE unterhalb eines gewissen Betrags liegen, wird das mobile Objekt mit elektronischem Geld aus der Abteilungskasse ausgestattet.

b) Um in dem vom Mitarbeiter vorgesehenen Zeitraum geeignete Termine für die Schulung zu belegen, bewegt sich das mobile Objekt in den Personal Digital

Assistent (PDA) des Mitarbeiters. Dort nimmt es die Termininformation des in Frage kommenden Zeitraums auf.

c) Das mobile Objekt wandert nun über das Intra- in das Internet und sucht eine geeignete Schulung für die aufgestellten Kriterien. Dabei wird ein möglicher Termin mit Hilfe der bereits enthaltenen Kalenderinformationen abgestimmt. Die Preise und Leistungen jeder einzelnen Firma inklusive mobiler, multimedialer Objekte zu Werbe- und Informationszwecken werden übernommen.

d) Das mobile Objekt sucht nach dem günstigsten Anbieter der Entwicklungsumgebung. Der Kauf wird mit elektronischem Geld durchgeführt und die Entwicklungsumgebung inklusive eines Quittungsobjekts aufgenommen.

e) Das mobile Objekt bringt die IDE zum Mitarbeiter und schlägt mögliche Schulungsangebote vor. Der Mitarbeiter entscheidet nun, welche Schulung er besuchen möchte.

f) Die Schulungsteilnahme wird der Personalabteilung (PA) gemeldet.

g) Weil die Summe für die Schulung über einem gewissen Betrag liegt, wird die Kostenübernahme der Schulung beim Einkauf beantragt. Nach dessen Zustimmung nimmt das mobile Objekt Daten für die Erstellung der Rechnung auf, die das Schulungsunternehmen später dem Einkauf zustellt.

h) Das mobile Objekt bestätigt bei der Schulungsfirma den vorläufig reservierten Termin und übermittelt die Daten für die Rechnungserstellung.

i) Mit einer Bestätigung und den Schulungsunterlagen ausgestattet kehrt das mobile Objekt zum Mitarbeiter zurück.

2.2 Eigenschaften zur Etablierung von Vertrauen in Gastsysteme

Folgenden Eigenschaften sind geeignet, eine Vertrauensbeziehung von einem mobilen Objekt zu einem Gastsystem aufzubauen:

- **Integrität und Vertraulichkeit in der Ablaufumgebung**
  - *Keine Manipulation von Daten, Kode und Ausführung* (1. Ordnung). Diese Garantien sind essentiell, damit das Objekt in der Intention des Besitzers agieren und die gestellten Aufgaben korrekt lösen kann. So könnte in einem Gastrechner einer Schulungsfirma ein Virus in das Objekt eingeschleust werden, welches später in sensible Bereiche des Unternehmens vordringt(e, f, g, i)[1]. Daher wurde in der Sicherheitspolitik des Großunternehmens festgelegt, dass mobile Objekte, die in das Firmennetzwerk zurückkehren, nur solche Netzknoten besuchen

[1]Die Buchstaben in der Klammer verweisen auf die Orte des Beispielszenarios in Abbildung 1, wo nach Ausführung der korrespondierenden Transition ein Bedrohungspotential vorliegt.

dürfen, die über die oben genannte Zusicherung verfügen. Diese kann beispielsweise von dem Hersteller einer Ablaufumgebung oder einer spezialisierten Firma gegeben werden ($\bar{e}$, $\bar{f}$)[2].

– *Systemaufrufe verhalten sich exakt nach ihrer Spezifikation* (1. Ordnung). Wenn ein mobiles Objekt erstellt wird, muss der Programmierer darauf Vertrauen können, dass die Spezifikation der Ablaufumgebung genau eingehalten wird. In unserem Szenario wäre es denkbar, dass Vergleichsoperationen der Ablaufumgebung auf einem Gastsystem so geändert sind, dass das eigene Angebot des Gastsystembetreibers immer als das günstigste zurückgegeben wird (c, d). Der Hersteller der Ablaufumgebung oder eine Firma, die Ablaufumgebung und Spezifikation unabhängig vergleicht, können eine solche Eigenschaft bescheinigen ($\bar{k}$).

– *Besitzer und Internetadresse des Gastsystems* (1. Ordnung). Das Gastsystem und damit auch sein Besitzer haben physischen Zugang zu hospitierenden mobilen Objekten. Daher hängt die Sicherheit in hohem Maße davon ab, ob einem solchen Gastsystem getraut werden kann (c, d). Dazu ist eine einwandfreie Identifizierung im Zusammenhang mit der Netzadresse erforderlich, die von einer staatlichen Zertifikationsautorität durchgeführt werden sollte ($\bar{h}$). Die Reputation eines Anbieters baut sich durch die Erfahrungen der Netzgemeinde auf.

– *Kein Ausspähen von Code, Daten und Kommunikation mobiler Objekte* (2. Ordnung). Im elektronischen Handel ist es häufig erforderlich, mobilen Objekten vertrauliche Informationen mitzugeben, die zum Erfüllen einer Aufgabe benötigt werden. So enthält das mobile Objekt in unserem Beispiel elektronisches Geld zum Einkaufen der Software und Teile des Terminkalenders des Mitarbeiters. Ein anderes Problem besteht im zweitbesten Angebot: Wenn ein Softwarehändler die Preise seiner Konkurrenten auslesen kann, kann er seinen Preis soweit erhöhen, dass er gerade noch den Zuschlag erhält, obwohl er vorher günstigster war (c, d). Da es in vielen anderen Fällen vorstellbar ist, dass die Vertraulichkeit eines Objektes nicht bedeutend ist (z.B. bei wissenschaftlicher Literaturrecherche), wird diese Eigenschaft nur die Ordnung 2 eingruppiert, auch wenn sie in einigen Bereichen von großer Bedeutung ist. Wenn Vertraulichkeit gegeben sein soll, kann sich dazu der Betreiber eines Gastsystems rechtlich verpflichten oder eine Zertifikationsautorität mit Kontrollmöglichkeiten dieses bescheinigen ($\bar{d}$).

[2]Die überstrichenen Buchstaben in der Klammer verweisen auf die korrespondierenden Zertifikationsbeziehungen in Abbildung 2.

- **Ressourcenanforderung mobiler Objekte**
  - *Garantien* (2. Ordnung). Mobile Objekte können Anforderungen besitzen, welche Ressourcen sie in welchem Umfang für die gestellte Aufgabe benötigen. Dies hilft dem Schutz vor Verfügbarkeitsangriffen ebenso wie der Garantie von Dienstequalitäten. So könnte in unserem Beispiel ein Softwarehändler, der um einen zeitlich limitierten Sonderpreis der Konkurrenz weiß, das mobile Objekt so lange aufhalten, bis das Angebot ausgelaufen ist, um dann selbst den Zuschlag zu bekommen (c, d). Folgende Ressourcen sollten berücksichtigt werden: Prozessorleistung, Speicherbedarf, Netzwerkbandbreite, Eingabe- und Ausgabeeinheiten, Massenspeicherbedarf, Systemaufrufe von Bibliotheken und Prozessmanagement. Die Garantie, welche Ressourcen für welche Zeiträume den Objekten zur Verfügung stehen, kann nur der Betreiber des Gastsystems geben ($\bar{j}$).
- **Juristische Zusicherungen**
  - *Verbindlichkeit von Informationen und Transaktionen* (2. Ordnung). Es ist selbstverständlich, dass sowohl Transaktionen wie auch Informationen eines Gastsystems verbindlich sein müssen. Wenn ein mobiles Objekt den günstigsten Anbieter für die zu beschaffende IDE auswerten möchte, muss gewährleistet sein, dass die herausgegebenen Informationen verbindlich sind und damit das richtige Ergebnis ermittelt werden kann (c, d). Rechtliche Grundlagen dieser Art sind daher von einer staatlichen Zertifikationsbehörde zu bescheinigen ($\bar{l}$).
  - *Kopierrechtsnachweis* (2. Ordnung). Ein mobiles Objekt kann kopiergeschützte Informationen erwerben. Damit die Kopierrechte gewahrt bleiben, muss das Gastsystem nachweisen, dass es ein Recht hat, diese Informationen weiterzugeben. In unserem Falle muß der Softwarehersteller nachweisen, daß er die Software des IDE Herstellers vertreiben darf, bevor das Objekt die Software aufnimmt.

2.3 Eigenschaften zur Etablierung von Vertrauen in mobile Objekte

Folgende Eigenschaften sind geeignet, eine Vertrauensbeziehung von einem Gastsystem zu einem mobilen Objekt aufzubauen:

- **Integrität und Vertraulichkeit für Daten und Code des Gastsystems**
  - *Keine Funktionalität zum Ausspionieren von Daten und Code* (1. Ordnung). Unabhängig von der Nutzung eines Gastsystems ist dessen Ver-

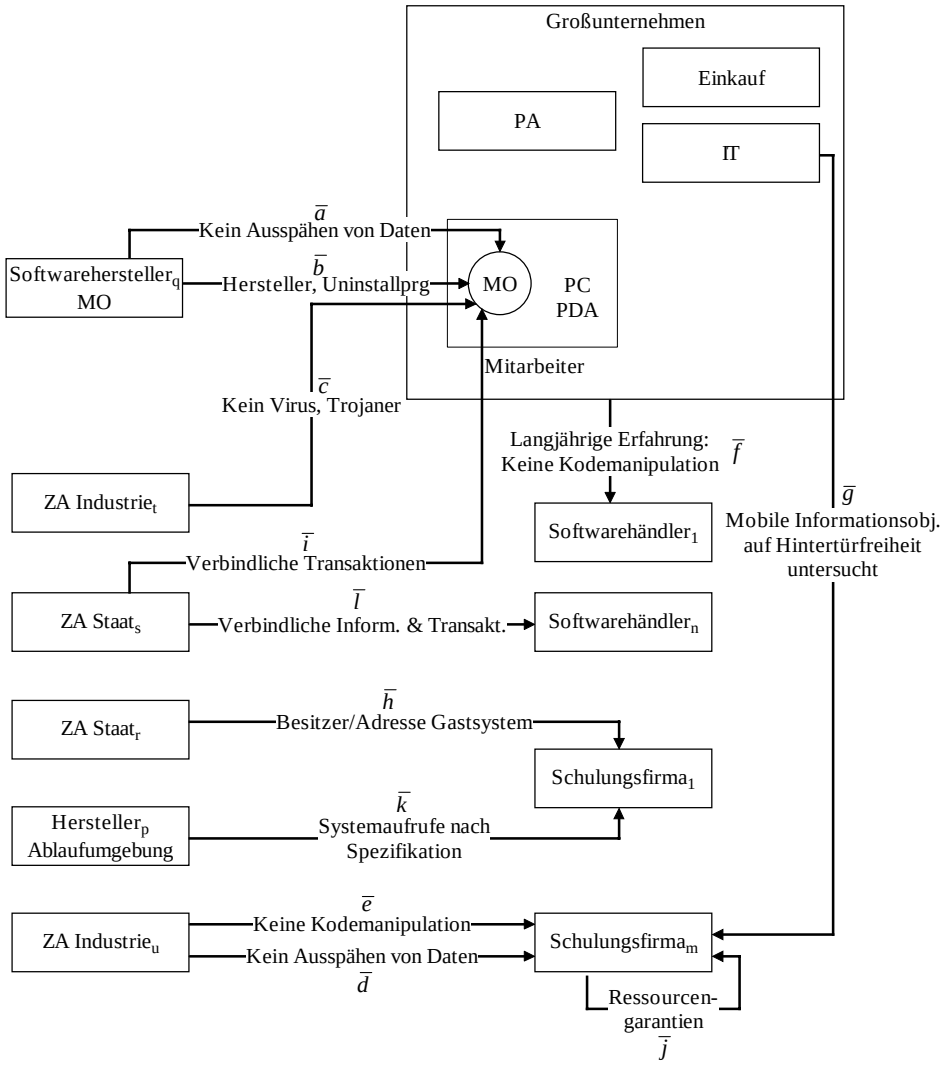

Abbildung 2: Zertifikationsbeziehungen im Szenario

traulichkeit und Integrität eine zentrale Forderung. Im angeführten Beispiel könnten beispielsweise vertrauliche Daten des Großunternehmens (e, f, g, i) oder die Kundendaten eines Handelsunternehmens (c, d) ausgespäht werden. Diese Eigenschaften können vom Hersteller eines mobilen Objektes bestätigt werden, der auch eine Liste von Systemaufrufen mitliefern sollte, damit das Gastsystem eine potentielle Bedrohung besser einschätzen kann ($\bar{a}$). Ferner kann eine spezialisierte Zertifikationsautorität nach Durchsicht des Quellkodes diese Eigenschaft bescheinigen.

– *Ausschließlich spezifizierte Änderungen an Daten und Code* (1. Ordnung). In manchen Fällen sind Änderungen an Dateien (z.B. Registry

unter Windows) für die Funktionsweise eines mobilen Objektes notwendig. Damit ein Gastsystem abschätzen kann, ob dies in Konflikt mit der vorhandenen Konfiguration steht, sollte eine Liste von zu verändernden Daten mitgeliefert werden. Diese Liste liefert der Hersteller des mobilen Objektes.

- *Keine Viren, Trojanische Pferde oder Hintertüren vorhanden* (1. Ordnung). Diese Eigenschaften sind grundlegend für die Sicherheit eines Gastsystems. So wäre eine Vireninfektion in einem Handelsunternehmen (c, d) fatal, weil durch einen regen Austausch mobiler Objekte eine hohe Verbreitung mit potentiellen Geschäftspartnern (e, i) erfolgen würde. Diese Eigenschaft sollte von einer Autorität der Wirtschaft ($\bar{c}$) oder dem Hersteller eines mobilen Objektes erfolgen. Die IT Abteilung eines Unternehmens könnte aber auch eine Prüfung der mobilen Objekte bei solchen Partnern, mit denen intensive Geschäftsbeziehungen bestehen, vornehmen. Dies wäre im Szenario bei der $\text{Schulungsfirma}_{\text{m}}$ gegeben ($\bar{g}$), deren Informationsobjekte[3] über Schulungskurse in Schritt c aufgenommen wurden.
- *Hersteller und Besitzer eines mobilen Objekts* (1. Ordnung). Der Softwarehersteller eines mobilen Objekts sowie dessen Besitzer, der das Objekt für den Einsatz angepasst hat, sind ein wichtiger Faktor in der Entscheidung, wie vertrauenswürdig ein solches Objekt ist. Ein Hersteller sollte diese Daten zu einem mobilen Objekt anbieten ($\bar{b}$).
- *Deinstallierungsoption* (2. Ordnung). Falls ein mobiles Objekt mit längerer Lebensdauer (wie z.B. die erworbene IDE) eine große Anzahl von Systemänderungen macht, sollte dem Gastsystem garantiert sein, dass alle Änderungen rückgängig gemacht werden können (e). Der Hersteller eines mobilen Objektes kann eine solche Garantie geben ($\bar{b}$).

- **Ressourcennutzung mobiler Objekte**
  - *Garantien* (2. Ordnung). Zur Unterbindung von Verfügbarkeitsangriffen und Abgabe von Qualitätsgarantien über Dienste kann vom mobilen Objekt eine Liste eingefordert werden, welche Ressourcen (Prozessorleistung, Speicher, Netzwerkbandbreite, Massenspeicher, Prozessmanagement) in welchem Umfang benötigt werden. Dies kann durch den Hersteller oder Besitzer des mobilen Objekts bescheinigt werden.

- **Juristische Zusicherungen**
  - *Verbindlichkeit von Informationen und Transaktionen* (2. Ordnung). Für ein Gastsystem ist es oftmals von Bedeutung, dass Personen im

[3] In Abbildung 1 wurden diese mobilen Informations- und Werbeobjekte zwecks Übersichtlichkeit nicht dargestellt.

juristischen Sinne die Verantwortung für Informationsaussagen und Transaktionen ihrer mobilen Objekte übernehmen. So muss sich ein Schulungsunternehmen darauf verlassen können, dass sein Kunde auch deren Kosten übernimmt (h). Eine solche Eigenschaft sollte von einer staatlichen Zertifikationsautorität bescheinigt werden (ī) oder beispielsweise auch von einem Kreditkartenunternehmen, über welches eine Transaktion abgerechnet wird.

3 Spezifikation vertrauensbasierter Sicherheitspolitiken

Dieses Kapitel beschreibt einen Ansatz zur Spezifikation von Vertrauensbeziehungen zwischen mobilen Objekten und ihren Gastsystemen. Auf der Grundlage der Szenariendiskussion im vorangehenden Kapitel werden die zwei Grundelemente einer vertrauensbasierten Sicherheitspolitik identifiziert und ihre Semantik wird mittels einfacher Prädikatenlogik beschrieben. Anschließend stellen wir auf der Grundlage der prädikatenlogischen Beschreibung einen Spezifikationskalkül vor, der eine Ableitung ausführbarer Sicherheitspolitiken erlaubt. Zur Illustration greifen wir in einem Beispiel einen Ausschnitt des obigen Szenarios auf und zeigen dessen Formulierung im Spezifikationskalkül. Schließlich demonstrieren wir den Praxisbezug durch Integration der Beispielspezifikation als domänenspezifische Sicherheitspolitik in die JDK1.2 Sicherheitsarchitektur.

3.1 Grundelemente einer Spezifikation

Die Grundelemente unserer Vertrauenslogik sind Principals, Etiketten (oder *Labels*) und Aussagen über ihre Vertrauenswürdigkeit. In der Bedeutung dieser Begriffe lehnen wir uns eng an Burrows, Abadis und Needhams *Logic of Belief* [BAN89] und die Terminologie in [BL76] an; tatsächlich kann der im folgenden verwendete Labelbegriff als Verallgemeinerung der Bell/LaPadula-Labels zur Beschreibung der Vertraulichkeitsklassen von Dokumenten und der Vertrauenswürdigkeit von Principals angesehen werden. Hier wie dort dienen Labels zum Ausdruck zugesicherter (zertifizierter) und zur Schaffung von Vertrauen geeigneter Eigenschaften.

Abbildung 3 zeigt den grundsätzlichen Aufbau eines Labels. Die hier gezeigten Komponenten sind als exemplarisch zu verstehen und ergeben sich in konkreten Sicherheitsinfrastrukturen aus dem konkreten Lebenszyklusmodell von Zertifikaten. Eine ausführliche Auflistung findet sich in [Gär00].

Mit solchen Labeln lassen sich nun Grundelemente zur Formulierung von Vertrauen wie folgt formulieren.

- **TRUST** *ca* [**WITH RESPECT TO** *label*]
  TRUST *label* [**WHEN** *condition*]

Name des Labels	Eineindeutige Identifizierung des Labels (e.g. zu Rückrufzwecken)
Name des Signierers	Eineindeutige Identifizierung des signierenden Principals (Person oder Organisation)
Name der assoziierten Entität	Eineindeutige Identifizierung der Entität, über die das Label eine Aussage trifft (e.g. Gastrechner, mobiles Objekt)
Datum der Signierung, Gültigkeitsdauer, etc.	Informationen zur Implementierung eines Lebenszyklusmodells
Prüfungsverfügungen	durch den Aussteller vorgeschriebene Validitätsprüfungen (e.g. bei jeder Benutzung)
Zugesicherte Eigenschaft	

Abbildung 3: Aufbau eines Labels

Vertrauen kann einerseits gegenüber einem Principal in der Rolle einer Zertifikationsautorität deklariert werden, andererseits gegenüber einem bestimmten Label; beide Erklärungen können durch optionale Einschränkungen verfeinert werden. So wird auf einem Gastsystem der Bescheinigung der Zertifikationsautorität$_t$ (Abb. 2) des mobilen Objekts vertraut, dass sich kein Virus im mobilen Objekt befindet (`TRUST za`$_t$ `WITH RESPECT TO KeinVirus-Label`). Dies wird durch die Bedingung eingeschränkt, dass das Zertifikat bei jeder Ausführung des mobilen Objekts auf Widerruf überprüft wird (`TRUST KeinVirus-Label WHEN LabelPrüfung=JedeAusführung`). Eine solche Festlegung der Überprüfungshäufigkeit kann auch in den Metadaten eines Labels festgelegt werden, wo eine Zertifikationsautorität festlegen kann, wie häufig das ausgestellte Label auf Widerspruchsfreiheit geprüft werden soll, damit die Aussage dieser Autorität als frisch angesehen werden kann.

- *label* **ALLOWS** *action* [**LIMITED BY** *resource-constraint*] [**WHEN** *condition*]

  Diese Deklaration bestimmt, welche Aktionen bei welchem Label erfolgen dürfen, optional eingeschränkt von der Anzahl der zu vergebenden Ressourcen und einer Bedingung. So darf ein mobiles Objekt, welches auf einem Gastsystem hospitiert, ein Schreibzugriff in das „*tmp*“-Verzeichnis bis zu einer Größe von 10MB vornehmen, wenn Virenfreiheit bescheinigt wurde und noch mindestens 100MB auf der lokalen Festplatte an Speicherplatz verfügbar ist (`KeinVirus-Label ALLOWS write LIMITED BY (OS_API.fname="tmp", OS_API.fsize=10MB) WHEN OS_API.diskfree >= 100MB`).

Zur Beschreibung der Semantik dieser Grundelemente verwenden wir im Folgenden eine einfache Prädikatenlogik. Dabei reichen die folgenden vier Prädikate gemeinsam mit den genannten Schlußregeln aus.

3.1.1 Prädikate

trusted(ca) Prädikat einer Zertifikationsautorität, welches ihre Vertrauenswürdigkeit ausdrückt; dieses umfassende Prädikat kann feingranularer durch das zweistellige Prädikat

trusted(ca,l) ausgedrückt werden, welches die Vertrauenswürdigkeit der Zertifikationsautorität *ca* nur in Bezug auf das Label l ausdrückt.

fresh(l) Prädikat eines Labels, welches eine spezifizierbare Frischebedingung erfüllt, z.B. dessen Gültigkeit innerhalb eines bestimmten Zeitintervalls überprüft wurde.

trusted(l) Prädikat eines Labels, dem vertraut wird.

enables(l,a) Zweistelliges Prädikat, welches ausdrückt dass Label l die Aktion a erlaubt.

allowed(a) Prädikat einer Aktion, die auf Grund hergestellten Vertrauens zulässig ist; stellt das Ziel der Anwendung der folgenden Schlussregeln dar.

3.1.2 Schlussregeln

Die Schlussregeln legen fest, auf welche Weise – ausgehend von Vertrauensspezifikationen – die Erlaubnis zum Ausführen einer Operation durch eine mobiles Objekt abgeleitet wird.

(1) Wenn wir einer Zertifikationsautorität trauen, dann trauen wir ihr auch bezüglich bestimmter von ihr ausgegebener Label:

$$\forall ca, l : trusted(ca) \Rightarrow trusted(ca, l)$$

(2) Wenn wir einer Zertifikationsautorität hinsichtlich eines von ihr ausgegebenen Labels trauen und dieses Label frisch ist, dann trauen wir auch dem Label selbst:

$$\forall ca, l : trusted(ca, l) \wedge fresh(l) \Rightarrow trusted(l)$$

(3) Wenn wir einem Label trauen und dieses Label eine Aktion ermöglicht, dann erlauben wir diese Aktion.

$$\forall l, a : trusted(l) \wedge enables(l, a) \Rightarrow allowed(a)$$

3.1.3 Semantik der Spezifikationselemente

Die Semantik der Spezifikationselemente wird durch eine gebräuchliche semigraphische Notation angegeben, die die Anwendung von Schlussregeln recht übersichtlich macht; sie ist zu lesen in der Form „Falls die Formeln oberhalb der Linie gelten, so gilt anschließend das Prädikat unterhalb der Linie“.

$$\frac{\textbf{TRUST}\ \text{ca}}{trusted(ca)}$$

$$\frac{\textbf{TRUST}\ \text{ca}\ \textbf{WITH RESPECT TO}\ \text{l}}{trusted(ca, l)}$$

$$\frac{\textbf{TRUST}\ \text{l}\ \textbf{WHEN}\ \text{condition}\ \wedge\ true(condition)}{fresh(l)}$$

$$\frac{\text{l}\ \textbf{ALLOWS}\ \text{a}}{enables(l, a)}$$

$$\frac{\text{l}\ \textbf{ALLOWS}\ \text{a}\ \textbf{LIMITED BY}\ \text{resource-constraint} \wedge \textit{true(resource-constraint)}}{enables(l, a)}$$

$$\frac{\text{l}\ \textbf{ALLOWS}\ \text{a}\ \textbf{WHEN}\ \text{condition}\ \wedge\ true(condition)}{enables(l, a)}$$

$$\frac{\begin{array}{l}\text{l}\ \textbf{ALLOWS}\ \text{a}\ \textbf{WHEN}\ \text{condition}\ \textbf{LIMITED}\ \text{BY resource-constraint} \\ \wedge\ true(condition)\ \wedge\ true(resource - constraint)\end{array}}{enables(l, a)}$$

3.2 Spezifikationssprache

Zur Spezifikation einer vertrauensbasierten Sicherheitspolitik benötigen wir nun noch eine konkrete Spezifikationssprache. Wir geben diese auszugsweise in einer Backus-Naur Form an; ≺Ident≻ ist dabei ein wohlgeformter Identifier in der Politik, welcher Principals und Label identifiziert und dessen konkrete Syntax hier nicht weiter spezifiziert werden muss; das leere Produktionssymbol ist „ε“.

Politik – Deklaration

≺POL≻ ::= „Policy“ ≺Ident≻ „=“ ≺pol_decl≻ „End“ ≺Ident≻ „ . “ ;

≺pol_decl≻ ::= „Certification Authority Set:“ ≺ca-declaration≻
„Label Set:“ ≺label-declaration≻
„Action Set:“ ≺action-declaration≻
„Trust Specification:“ ≺trust-specification≻
„Right Specification:“ ≺right-specification≻ ;

Deklaration der Zertifikationsautoritäten

≺ca-declaration≻ ::= ≺ca-list≻ | ε „ ; “ ;
≺ca-list≻ ::= ≺Ident≻ „ ; “ | ≺Ident≻ „ ; “ ≺ca-list≻ ;

Deklaration der Label

≺label-declaration≻ ::= ≺label-list≻ | ε „ ; “ ;
≺label-list≻ ::= ≺Ident≻ „ ; “ | ≺Ident≻ „ ; “ ≺label-list≻ ;

Deklaration der Aktionen

≺action-declaration≻ ::= ≺action-list≻ | ε „ ; “ ;
≺action-list≻ ::= ≺Ident≻ „ ; “ | ≺Ident≻ „ ; “ ≺action-list≻ ;

Spezifikation von Vertrauen

≺trust-specification≻ ::= ≺trust-list≻ | ε „ ; “ ;
≺trust-list≻ ::= ≺trust-statement≻ „ ; “ | ≺trust-statement≻ „ ; “ ≺trust-list≻ ;
≺trust-statement≻ ::= „TRUST“ ≺ca≻ [„WITH RESPECT TO“ ≺label≻] |
„TRUST“ ≺label≻ [„WHEN“ ≺condition≻] ;
≺ca≻ ::= ≺Ident≻ ;
≺label≻ ::= ≺Ident≻ ;
≺condition≻ ::= ≺B-Exp≻ ;

Deklaration der durch Label erschlossenen Operationsrechte

≺right-specification≻ ::= ≺right-list≻ | ε „ ; “ ;
≺right-list≻ ::= ≺right≻ „ ; “ | ≺right≻ „ ; “ ≺right-list≻ ;
≺right≻ ::= ≺label≻ „ALLOWS“ ≺action≻ [„LIMITED BY“ ≺resource-constraint≻]
[„WHEN“ ≺condition≻] ;
≺action≻ ::= ≺Ident≻ ;
≺resource-constraint≻ ::= ≺B-expre≻ ;

Bemerkungen

Das Nichtterminalsymbol ≺B_Exp≻ steht für einen wohlgeformten boole'schen Ausdruck im Prädikatenkalkül erster Ordnung, in dem ausschließlich Variablen der Sicherheitspolitik mittels $\forall, \exists, \wedge, \vee, \Rightarrow$ etc. verknüpft sind. Die Schreibweise wohlgeformter Ausdrucke erfolgt in Anlehnung an [Man74], wobei ebenfalls die dortige verkürzte Form („$\forall x$“ statt „$(\forall x)$“, „$x \neq y$“ statt „$\neg(x = y)$“ etc. immer dann verwendet wird, wenn dies zu keinen Missverständnissen führen kann.

Die Deklaration der Aktionen dient der Verständigung zwischen mobilem Objekt und Gastrechner; sie erfolgt in Form einer *Interface Definition Language* (IDL), wie sie unter anderem in Middlewaresystemen zu diesem Zwecke eingesetzt wird (e.g. [Obj99]).

3.3 Eine Beispielspezifikation

Die folgende Spezifikation skizziert das Beispiel aus Abschnitt 3.1. Die Formulierung in der Spezifikationssprache ist unabhängig von einer spezifischen Plattform eines Gastrechners; dies äußert sich in der Formulierung der Aktionsnamen und Ressourcenbeschränkungen. Die Transformation in eine plattformspezifische Notation kann einerseits manuell erfolgen; andererseits greifen für eine automatische Transformation die bereits oben erwähnten IDL-Techniken.

Policy Example =
Certification Authority Set:
za_t;
Label Set:
KeinVirus-Label;
Action Set:
write;
Trust Specification:
TRUST za_t **WITH RESPECT TO** *KeinVirus-Label*;
TRUST *KeinVirus-Label* **WHEN** (LabelCheck=EachUse);
Right Specification:
KeinVirus-Label **ALLOWS** *write*
LIMITED BY (OS_API.fname=„tmp“, OS_API.fsize=10MB)
WHEN (OS_API.diskfree >= 100MB);
End Example.

Aus dieser Spezifikation läßt sich nun durch sukzessive Anwendung der Schlussregeln das Prädikat $allowed(write)$ für ein mobiles Objekt ableiten, welches mit einem *KeinVirus-Label* versehen ist, ausgestellt von der Zertifikationsautorität za_t und validiert vor der Benutzung, und dies unter der Randbedingung der Einhaltung der angegeben Limitationen geschieht.

3.4 Das Beispiel als Java-Domänenpolitik

Die Java-Sicherheitsarchitektur in der derzeitigen Version 1.2 besitzt grundsätzlich zwei Möglichkeiten zur Formulierung individueller Sicherheitspolitiken. Zum einen kann die zu jeder Java-Installation gehörende Standard-Sicherheitspolitik (implementiert durch die Java-Klasse `Policy`) durch eine Konfigurationsdatei individuell angepasst werden; üblicherweise geschieht dies mit Hilfe des `policytool`-Werkzeugs. Diese Variante ist sehr bequem, jedoch hinsichtlich der erreichbaren Ausdruckskraft recht beschränkt: möglich sind hiermit lediglich Regeln der allgemeinen Form „*falls Objekt von Ort X stammt und Zertifikationsautorität Y den Code unterschrieben hat, dann erlaube Operation Z*“. Ausgedrückt in der Notation der Konfigurationsdatei liest sich eine solche Regel in der Form:

```
keystore "my.keystore";
grant
  signedBy "za_t",
  codeBase "http://tu-ilmenau.de/~kuehnhau/java/mobileObjects/*"
  { permission java.io.FilePermission "tmp", "write"; };
```

`codeBase` gibt hierbei die Herkunft des mobilen Objekts an, `signedBy` fordert, dass der Code des mobilen Objekts aus einem JAR-File stammt, welches mit demjenigen privaten Schlüssel signiert wurde, der zu dem in der Schlüsseldatenbank `my.keystore` unter dem Namen `za_t` hinterlegten öffentlichen Schlüssel passt.

Dies reicht offensichtlich nicht aus, um auf natürliche Weise vertrauensbasierte Sicherheitspolitiken ausdrücken zu können. Wir betrachten daher eine zweite Alternative zur Formulierung individueller Sicherheitspolitiken in der Java-Sicherheitsarchitektur. Diese Spezifikationsform beruht prinzipiell auf dem Ersetzen der Standard-Klasse `Policy` durch eine eigene Implementierung. Im folgenden Beispiel wollen wir uns auf das zentrale Element dieser Klasse konzentrieren, die Methode `getPermissions`. Diese Methode erhält als Parameter Informationen über ein mobiles Objekt und berechnet hieraus eine Sammlung von dem Objekt zugebilligten Rechten. Diese Informationen sind in einem `CodeSource`-Objekt zusammengefasst, welches unter anderem die Herkunft des mobilen Objektes (in Form einer URL) sowie die an das Objekt angehefteten Label kapselt. Aus Platzgründen verwenden wir im Beispiel eine Java-ähnliche verkürzende Schreibweise; anzumerken ist schließlich, dass diese Implementierung automatisch aus der Spezifikation generierbar ist.

```
abstract PermissionCollection getPermissions(CodeSource codesource)
{ CASetType CASet = getCANames("my.keystore");
  LabelSetType LabelSet = lookup(KeinVirus-Label,codesource.Labels);

  // compute trust
  if (includes(CASet, za_t)) then
    if (signed(za_t, KeinVirus-Label)) then
      marktrusted(za_t, KeinVirus-Label);
  if (trusted(za_t, KeinVirus-Label)) then
    if (checklabelvalidity(za_t, KeinVirus-Label)) then
      markfresh(KeinVirus-Label);
  if (trusted(za_t, KeinVirus-Label) && fresh(KeinVirus-Label)) then
    marktrusted(KeinVirus-Label);

  // compute permissions
  if (trusted(KeinVirus-Label)) then
  { p = composePermissions("java.io.FilePermission","tmp","write");
    // following resource limitation and test strongly depend on
    // resource management facilities of implementation platform
    r = composeRestrictions
          ("java.io.FileSizeRestriction","tmp", 10485760);
    if (OS_API.diskfree >= 104857600) then return(p,r);
  }
  return(nil,nil);
}
```

Das erste Statement ist generiert aus der ersten ≺trust-specification≻ der Spezifikation und leitet das Prädikat *trusted(za_t, KeinVirus-Label)* her. Das zweite Statement ist generiert aus der zweiten ≺trust-specification≻ und leitet das Prädikat *fresh(KeinVirus-Label)* her. Das dritte Statement ist generiert aus der

Schlussregel (2) und leitet das Prädikat *trusted(KeinVirus-Label)* her. Das vierte Statement schließlich ist generiert aus der ≺right-specification≻ der Spezifikation und leitet das Prädikat *allowed(write)* her.

4 Zusammenfassung

Die Verwendung von Vertrauen als Grundlage zur Formulierung von Sicherheitspolitiken wurde an einem Beispiel aus dem elektronischen Handel motiviert. Vertrauensbasierte Sicherheitspolitiken ermöglichen es, Vertrauen gegenüber zugesicherten Objekt- und Systemeigenschaften zu formulieren und hieraus Rechte abzuleiten, welche die Interaktionen zwischen mobilen Objekten und ihren Gastgebersystemen bestimmen. Eine Vertrauensalgebra mit einer zugehörigen Spezifikationssprache unterstützt durch ihren hohen Abstraktionsgrad und geringen semantischen Abstand zwischen Realität und Formulierungsprache das elementare Prinzip der Einfachheit sicherer Systeme, ohne dass die Ausdrucksmächtigkeit leidet. Eine exemplarische Umsetzung einer Sicherheitspolitikspezifikation auf die Java-Sicherheitsarchitektur zeigte schließlich die praktische Relevanz des vorgestellten Konzepts.

Literatur

[BAN89] Michael Burrows, Martin Abadi, and Roger Needham. A Logic of Authentication. In *Operating Systems Principles*, pages 1–13. ACM, December 1989.

[BL76] D.E. Bell and L.J. LaPadula. Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report AD-A023 588, MITRE, March 1976.

[Gär00] Gregor Gärtner. Foundations of Trustworthy Mobile Objects. Technical report, Distributed Systems Department, University of Ilmenau, Faculty for Computer Science and Automation, April 2000.

[GMPS97] Li Gong, Marianne Mueller, Hemma Prafullchandra, and Roland Schemers. Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2. In *USENIX Symposium on Internet Technologies and Systems*, pages 103–112, Monterey, CA, December 1997.

[Gon98] Li Gong. Secure Java Class Loading. *IEEE Internet Computing*, 2(6):56–61, November 1998.

[Man74] Zohar Manna. *Mathematical Theory of Computation.* McGraw-Hill Book Company, 1974. ISBN 0-07-039910-7.

[Obj99] Object Management Group. The Common Object Request Broker: Architecture and Specification, October 1999. Version 2.3.1.