Sicherheit in verlässlichen adaptiven Informationssystemen

Sebastian Bächle

Technische Universität Kaiserslautern, Kaiserslautern, Germany
s_baechl@informatik.uni-kl.de

Zusammenfassung Informationssysteme sind aufgrund der Wichtigkeit der von ihnen verwalteten Daten ein sicherheitskritischer Faktor für Unternehmungen. Die Forderung nach Verlässlichkeit und Adaptivität dieser Systeme rückt dabei immer weiter in den Vordergrund. In immer komplexer interagierenden Netzen heterogener Systeme gilt es, Systeme zu entwickeln, die vor fremdem Zugriff und anderen Schäden sicher sind. Allerdings ergeben sich daraus Konsequenzen für die Sicherheitsmechanismen dieser Systeme. In dieser Arbeit wird die Bedeutung und Umsetzung von Sicherheitsanforderungen in verlässlichen adaptiven Systemen untersucht. Dabei sollen sowohl Techniken zum Entwurf sicherer Architekturen als auch konkrete Mechanismen wie die Etablierung von Vertrauen und die Erkennung und Behandlung von Angriffen eingehender betrachtet werden.

1 Einleitung

Informationssysteme bilden heute das Rückgrat nahezu aller Unternehmen und sie werden diesen Siegeszug auch weiter fortsetzen. Gerade im Bereich E-Business werden Flexibilität und Interoperabilität der Systeme immer stärker in den Mittelpunkt gestellt. Schlagworte wie On-Demand Computing, Adaptivität, Verlässlichkeit etc. bezeichnen daher die neuen Aufgaben der Gegenwart und Zukunft, die es anzugehen gilt. Die Sicherheit solcher Systeme spielt in diesem Zusammenhang eine besondere Rolle, da durch Sicherheitslücken in der IT-Infrastruktur die Existenzgrundlage eines Unternehmens akut bedroht ist. Angesichts der immer massiver werdenden Bedrohung durch gezielte Angriffe aus dem Internet und der gleichzeitigen Forderung nach Offenheit und garantierter Servicequalität, wird die Gewährleistung der Systemsicherheit allerdings eine zunehmend schwierigere Aufgabe. Die Umsetzung maximaler Konnektivität mit anderen Systemen und dynamischer Anpassung der Rechenkapazität auf weltweit verteilten Rechnern führt bei gleichzeitiger Forderung nach garantierter Leistungsqualität zu gegensätzlichen Architekturprinzipien [1]. Sollen auch noch Vertraulichkeit und Einbruchsicherheit in diesen Systemen garantiert werden, erscheint dies nur schwer umsetzbar. Um all diesen Ansprüchen gerecht zu werden, benötigt man neue Ansätze zur Entwicklung solcher verlässlicher adaptiver Informationssysteme (Dependable Adaptive Information Systems; DAIS) [2]. Zusammengefasst

gilt es also, Systeme zu entwickeln, die in immer komplexer interagierenden Netzen heterogener Systeme verschiedener Organisationen mit unterschiedlichen Sicherheitsbedürfnissen sicher vor fremdem Zugriff und anderen Schäden sind.

Um sich dieser Herausforderung stellen zu können, ist es notwendig, den Begriff Sicherheit im Kontext solcher Systeme genauer zu betrachen. Es stellt sich auch die Frage, wo in einem System Sicherheitsaspekte eine Rolle spielen und wie diese dort zu behandeln sind. Problematisch ist hierbei die in Informationssystemen fast immer realisierte Schichten- bzw. Multi-Tier-Architektur. Der Sinn dieses Konzeptes ist die schichtweise Abstraktion, die es erst ermöglicht, die Komplexität dieser Systeme handhabbar zu machen. Sogar ein einfaches webbasiertes Informationssystem existiert schon aus einer Persistenzschicht, einer Applikationsschicht und einer Präsentationsschicht. Betrachtet man nun noch weitere Entwicklungen wie Workflowmanagement-Systeme oder den Einsatz von Middleware als Abstraktion in der Anwendungsschicht, erreicht die Komplexität der bereitgestellten Funktionalität unüberschaubare Ausmaße. Selbst die einzelnen Systemschichten werden zum Teil mit einer Schichtenarchitektur realisiert (z. B. Datenbankmanagement-Systeme (DBMS) [3]). Im Folgenden bezieht sich der Begriff Schicht jedoch nur auf den Aufbau des Gesamtsystems und nicht auf die interne Realisierung z. B. der Persistenzschicht. In den meisten Fällen beruhen die Schichten auf mächtigen und hochoptimierten Standardplattformen, die für den jeweiligen Einsatz angepasst werden. Ein Beispiel hierfür sind wieder DBMS. Aus diesen Gründen scheint es weder wünschenswert noch praktikabel, von dieser Architektur abzuweichen.

Das Prinzip der schichtweisen Abstraktion beruht darauf, dass jede Schicht ihre Funktionalität jeweils nur der darüberliegenden Schicht zur Verfügung stellt, ohne etwas über diese zu wissen. Schichtenübergreifende und bidirektionale Informationsflüsse widersprechen also dem strikten Schichtenparadigma. Offensichtlich ist es für umfassende Sicherheitskonzepte aber unvorteilhaft, dass diese Einschränkung vorliegt. Es kann nicht davon ausgegangen werden, dass Sicherheitsrisiken nur auf einer Schicht auftreten und dass Angriffe nur auf eine Schicht begrenzt sind. Folglich muss jede Schicht ihr eigenes Sicherheitskonzept implementieren. Gängige Maßnahmen sind hier Mechanismen zur Identifikation und Zugriffskontrolle sowie die Verschlüsselung von Kommunikationskanälen. Gelingt es aber einem Angreifer, die Kontrolle über zumindest einen Teil einer Systemschicht zu erhalten, versetzt es ihn ebenfalls in die Lage, auf die darunter liegenden Schichten zuzugreifen. Gerade deshalb wäre es von Vorteil, ungewöhnliche Vorkommnisse auf den verschiedenen Schichten zu erfassen, um Angriffe erkennen und darauf reagieren zu können. Ein weiteres Sicherheitsproblem ist die mögliche Verteilung innerhalb einer Schicht. Gerade hier sind aufgrund der Kommunikationswege und der potentiell kompromittierten Hosts sicherheitsrelevante Fragestellungen besonders kritisch. Adaptive Systeme sind besonders hier auf dynamische Strukturen angewiesen.

In dieser Arbeit sollen deshalb die Sicherheitsanforderungen an DAIS und die daraus entstehenden Probleme eingehender untersucht werden. Es wird versucht, den Begriff Sicherheit mit konkreten Inhalten auszufüllen und anhand

derer die existierenden Gefahren zu identifizieren. Auf dieser Grundlage sollen Möglichkeiten neuer Konzepte und Methoden betrachtet werden, um den formulierten Sicherheitsanforderungen solcher Systeme sowohl im Entwurf als auch im späteren Betrieb Rechnung tragen zu können. Dabei soll vor allem das Problem der Überprüfbarkeit eines Sicherheitskonzeptes beachtet werden. Die Adaptivität der Systeme soll anschließend neben leistungsbezogenen auch auf sicherheitsbezogene Aspekte ausgeweitet werden. Konkret wird dies anhand der Selbstüberwachung und -anpassung von Systemen bei drohenden Sicherheitsrisiken vorgenommen.

In Abschnitt 2 dieser Arbeit werden verschiedene Begriffe wie Sicherheit, Vertraulichkeit, Verlässlichkeit, Fehlertoleranz etc., die im Rahmen von Sicherheitsbeschreibungen verwendet werden, erläutert und eingeordnet. Daneben werden auch die Gefährdungen dieser Qualitäten bestimmt, um die Begrifflichkeiten für die weiteren Betrachtungen zu klären. Abschnitt 3 geht auf die Möglichkeiten der Modellierbarkeit und Überprüfbarkeit von Sicherheit ein. Ein besonderes Augenmerk wird dabei auf die Realisierbarkeit von Vertrauen in verteilten Systemen gelegt. Die Idee eines *Web of Trust* spielt hierbei eine besondere Rolle. Intrusion Detection als aktive Sicherheitststrategie wird in Abschnitt 4 betrachtet. Abschließend erfolgt eine Zusammenfassung der vorgestellten Konzepte.

2 Begriffliche Grundlagen

Ein besonderes Problem bei der Diskussion von Sicherheit im Kontext von Computersystemen ist im deutschsprachigen Raum die Ungenauigkeit bei der Übersetzung aus dem Englischen. In der englischsprachigen Fachliteratur wird von unterschiedlichen Aspekten wie *Safety*, *Security* und *Confidentiality* etc. gesprochen, deren Differenzierung bei der Übersetzung in den allgemeineren deutschen Begriff Sicherheit verloren geht. Aus diesem Grund werden diese Begriffe hier nach der Schematik von Randell et al. eingehend erläutert [4]. Außerdem sollen eindeutige deutsche Bezeichungen festgelegt werden. Ausgegangen wird von den beiden Grundbegriffen *Verlässlichkeit* und *Sicherheit*. Sie bilden zusammen mit *Funktionalität*, *Leistungsfähigkeit* und *Kosten* die grundlegenden Eigenschaften eines Systems.

2.1 Verlässlichkeit und Sicherheit

Ein System wird als *dependable* betrachtet, wenn es in der Lage ist, bei der Ausführung der angebotenen Dienste auftretende Fehler so zu behandeln, dass diese nicht häufiger oder schwerwiegender sind, als dies tolerierbar ist. Das Auftreten von Fehlern wird hierbei explizit nicht verboten. Allerdings wird gefordert, dass diese eindeutig abgesteckte Grenzen nicht überschreiten, um die Nutzbarkeit des Systems nicht zu beeinträchtigen. Dependability ist also ohne Probleme mitVerlässlichkeit zu übersetzen. Alternativ existiert eine zweite Definition. *Dependability* beschreibt die Fähigkeit, eine angebotene Leistung so durchzuführen, dass dieser begründet vertraut werden kann. Diese Definition stützt sich auf den

Begriff Vertrauen und dessen Begründung. Da diese beiden Begriffe aber nur schwer exakt zu erfassen sind, ist die erste Variante zu bevorzugen.

Die Grundlage eines verlässlichen Systems ist die Erfüllung bestimmter Qualitätsansprüche: *Availability* (Verfügbarkeit) steht für die Bereitschaft des Systems, seine angebotenen Dienste korrekt auszuführen und *Reliability* (Zuverlässigkeit) für die Forderung, diesen Dienst auch kontinuierlich korrekt durchführen zu können. Die Korrektheit eines Dienstes bezieht sich dabei auf die Einhaltung seiner Spezifikation. Diese beiden Aspekte bilden zusammen somit die Forderung nach Durchführungssicherheit in Bezug auf die konkrete Implementierung. Ebenso wird gefordert, dass ein angebotener Dienst weder für die Benutzer noch für die Umgebung des Systems irgendeine Art nachteiliger Konsequenzen verursacht. Der Begriff Umgebung umfasst dabei sowohl andere Programme als auch Hardware und sonstige physische Objekte. Diese Forderung wird mit Safety (Ausführungssicherheit) beschrieben. Um Verlässlichkeit zu erreichen, ist es darüber hinaus notwendig, dass weder das System noch seine Daten versehentlich oder im Zuge eines Angriffes verändert werden können. Man spricht in diesem Zusammenhang von *Integrity* (Integrität). Auch die Benutzerdaten müssen vor dem Zugriff nicht autorisierter, evtl. böswilliger Dritter sicher sein. Hierfür steht der Begriff *Confidentiality* (Vertraulichkeit). Die Einrichtung eines neuen Informationssystems ist ein großer finanzieller und auch organisatorischer Aufwand. Deshalb werden sie so angelegt, dass sie schnell an die sich immer häufiger ändernden Anforderungen der Unternehmen angepasst werden können. Zusätzlich ändern sich aber auch die potentiellen Gefährdungen der Systeme. Deshalb wird als letzter Punkt *Maintainability* (Wartungsfreundlichkeit) gefordert, um Verlässlichkeit zu erreichen.

Security (Sicherheit) ist wie Dependability eine Kombination verschiedener Qualitäten. Sicherheit erfordert jedoch nur eine Teilmenge der für Verlässlichkeit geforderten Eigenschaften, nämlich Verfügbarkeit, Integrität und Vertraulichkeit. Somit ist Sicherheit die Mindestvoraussetzung für ein verlässliches System. In Abb. 1 sind die Eigenschaften von Verlässlichkeit und Sicherheit noch einmal zusammengefasst.

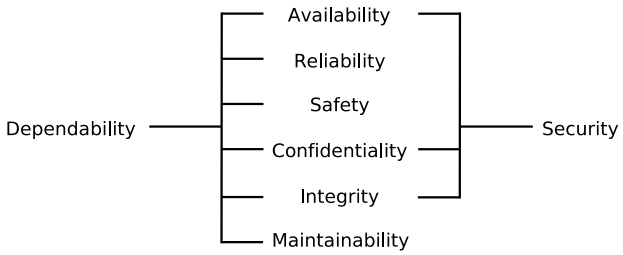

Abbildung 1. Eigenschaften von Verlässlichkeit und Sicherheit

Für den Betrieb verlässlicher adaptiver Systeme wird man in der Regel eine verteilte Systemarchitektur einsetzen. Durch die Verteilung, die eventuell sogar mehrere Systemschichten betrifft, wird der Begriff Vertrauen für diese Systemteile immer wichtiger. Während man mit Vorkehrungen zur Sicherstellung der Identität von Benutzern (Authentifikation) und einer Kontrolle der erlaubten Aktionen (Autorisierung) effektiv den Zugriff auf die eigenen Ressourcen sichern kann, ist beim Zugriff auf fremde Ressourcen die Sachlage genau umgekehrt. Es muss bestimmt werden, ob man einem Partner soweit vertrauen kann, um sich auf dessen Dienste verlassen zu können oder nicht. Vertrauen (*Trust*) ist demnach eine gerichtete binäre Relation. Der *Trustor* ist die Partei, die „denkende Einheit", die entscheiden muss, ob sie dem *Trustee* vertraut oder nicht [5]. Randell et al. bezeichnen Vertrauen daher ganz zweckmässig als akzeptierte Abhängigkeit. Diese kurze und prägnante Bezeichnung zeigt ganz deutlich die Bedeutung von Vertrauen für DAIS. Ein verlässliches System, welches fremde Dienste in Anspruch nehmen will, muss entscheiden, ob es dieses Risiko eingehen kann, ohne seine Aufgabe zu gefährden.

Ein Begriff, der momentan bei der Entwicklung zuverlässiger Systeme häufig verwendet wird, ist *Survivability* (Überlebensfähigkeit). Dieser martialisch anmutende Begriff wurde von Ellison et al. definiert [6]. Die Überlebensfähigkeit eines Systems beschreibt dessen Vermögen, in Anwesenheit von Angriffen, Fehlern oder unerwarteten Ereignissen seine Aufgabe fristgerecht zu erfüllen. Die Definition reicht über die Robustheit eines Systems hinaus und nimmt den zeitlichen Aspekt mit auf. Jedoch nicht zu verwechseln ist der Begriff Überlebensfähigkeit mit Fehlertoleranz. Während Überlebensfähigkeit die Qualität eines Systems beschreibt, ist Fehlertoleranz eine Technik, die verwendet wird, um diese Qualität zu erzielen [7]. Bei fehlertoleranten Techniken werden Fehler – egal, ob sie durch fehlerhafte interne oder externe Parameter entstanden sind – so behandelt, dass diese an der externen Schnittstelle nicht auftreten. Man bezeichnet solche Fehler als *maskiert*.

2.2 Fehler

Ähnlich wie bei dem Begriff Sicherheit ist es für Sicherheitsbetrachtungen sinnvoll, auch den Begriff Fehler genauer zu betrachten. Entscheidend für die Beschreibung der Eigenschaften eines Systems ist sein Verhalten an den Schnittstellen, über die es seine Dienste zur Verfügung stellt. Ein *Service Failure* oder einfach *Failure* (Fehlfunktion) beschreibt das Ereignis, das auftritt, wenn der von einem System angebotene Dienst von seiner korrekten Durchführung abweicht. Hierbei wird zwischen inhaltlichen und zeitlichen Abweichungen von der zu implementierenden Systemfunktion unterschieden. Von einer konsistenten Fehlfunktion spricht man, wenn die Fehlfunktion bei allen Benutzern auftritt. Bei einer inkonsistenten Fehlfunktion ist nur ein Teil der Benutzer von der nicht korrekten Dienstausführung betroffen.

Mit *Error* bezeichnet man den Teil des Gesamtsystemzustandes, der zum Auftreten einer Fehlfunktion führt. Eine adäquate deutsche Bezeichnung wäre

daher fehlerhafter oder ungültiger Zustand. Die Ursache eines solchen Fehlerzustandes ist ein *Fault* (Fehler). Fehler lassen sich über acht Gesichtspunkte klassifizieren: Zeitpunkt der Entstehung oder des Auftretens, Systemgrenze, phänomenologische Ursache, Dimension, Auswirkung, Absicht, Fähigkeit und Zeitraum (Abb. 2). Die einzelnen Dimensionen sollen hier aber nicht eingehender betrachtet werden.

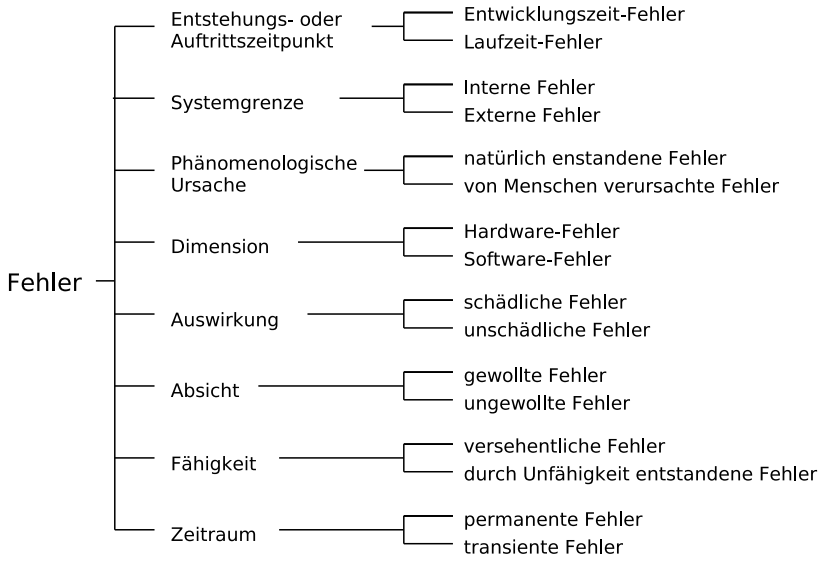

Abbildung 2. Klassifikation von Fehlern in Baumdarstellung

Wichtig für die Betrachtung von Sicherheit ist die Abhängigkeit der einzelnen Begriffe voneinander. Wird ein Fehler aktiv, d. h. erreicht die Programmausführung eine Stelle, an der ein Fehler zum Tragen kommt, erreicht das Programm einen fehlerhaften Zustand. Dieser fehlerhafte Zustand wird propagiert und führt zu einer Fehlfunktion. Diese Fehlfunktion kann ihrerseits wieder neue Fehler aktivieren. Randell et al. sprechen hier von der *Chain of Threats*. Ein System ist somit verwundbar (*vulnerable*), wenn intern ein Fehler existiert, der es einem – potentiell absichtlich verursachtem – externen Fehler ermöglicht, das System zu schädigen.

3 Entwurf sicherer Systeme

Um ein verlässliches System zu implementieren, ist es notwendig, dass die gesamte Architektur auf einen sicheren Betrieb ausgerichtet ist. Bereits kleinste Angriffsflächen können alle Sicherheitsmaßnahmen wirkungslos machen. Zu Beginn müssen daher die möglichen Bedrohungen identifiziert werden. Eine in der

Praxis weit verbreitete Möglichkeit ist die Erstellung von *Attack Trees*, die in Abschnitt 3.2 vorgestellt wird. Die Schwierigkeit der Modellierung und Überprüfung von Sicherheit ist in Abschnitt 3.3 thematisiert. Im Rahmen von DAIS sind vor allem die sicherheitsrelevanten Aspekte verteilter Architekturen interessant und da Vertrauen bei verteilten Systemen eine besondere Rolle einnimmt, werden Vertrauensmodelle in Abschnitt 3.4 eingehender betrachtet. In Abschnitt 3.5 wird Globe als Beispielarchitektur für sichere verteilte Objekte vorgestellt.

3.1 Allgemeine Sicherheitsmaßnahmen

Vor der Inbetriebnahme eines Systems gilt es, Fehler als Ursache von Sicherheitsproblemen mit allen Mitteln einzuschränken. Am wichtigsten ist es natürlich, schon während des Entwicklungsprozesses die Entstehung von Fehlern soweit wie möglich auszuschließen. Dies betrifft vor allem Themen des Software Engineering, wie das Entwickeln nach einem kontrollierten Vorgehensmodell und den Einsatz bewährter Designtechniken. Darüber hinaus empfiehlt sich z. B. die Verwendung geeigneter Werkzeuge und streng getypter Programmiersprachen. Diese Ansätze fasst man unter *Fault Prevention* (Fehlervermeidung) zusammen.

Fault Tolerance betrifft den Einsatz der bereits genannten fehlertoleranten Implementierungstechniken. Ein Beispiel hierfür ist die mehrfache Implementierung der gleichen Funktionalität durch unterschiedliche Ansätze. Das *Fault Removal* (Fehlerentfernung) beinhaltet in der Regel die formale Verifikation der Software sowie die Validation durch systematisches Testen. Enthält das System Mechanismen zur Fehlertoleranz, sollten diese natürlich auch durch die künstliche Einführung von Fehlern in den Testfällen überprüft werden. Dieses Vorgehen nennt man *Fault Injection*.

Am schwierigsten ist sicherlich das Vorhersagen von Fehlern in einer Architektur, das so genannte *Fault Forecasting*. Hier wird durch eine Evaluation des Systemverhaltens versucht, das mögliche Auftreten von Fehlern zu ermitteln. Bei der qualitativen Evaluation hat man das Ziel, Fehlerquellen zu identifizieren, zu klassifizieren und mögliche Fehlerkombinationen zu modellieren. Die quantitative oder auch probabilistische Evaluation versucht die Wahrscheinlichkeiten zu ermitteln, mit der bestimmte Systemeigenschaften erfüllt sind. Hat man schließlich eine mögliche Fehlerquelle gefunden, kann diese Lücke durch eine Modifikation der Architektur geschlossen werden. Zur Voraussage von Fehlern müssen Funktionen entwickelt werden, um die Reaktion eines Systems auf einen Stimulus vorauszusagen. Die Architektur des Systems und der konkrete Stimulus sind dabei als Parameter zu betrachten. Ellison et al. nennen eine solche Funktion *Quality Attribute Model* [8].

Es gibt eine große Menge an Architekturtaktiken, um Sicherheitslücken zu schließen. Redundanz ist etwa eine Möglichkeit, um die Verfügbarkeit bei Denial-of-Service-Attacken und bei der Abschaltung kompromittierter Systemteile aufrecht zu halten. Durch Designdiversität wird erreicht, dass ein Angreifer nicht mit der gleichen Methode in das komplette System eindringen kann. Die Entwickler können auch spezielle Systemteile einfügen, die alleine zur Abwehr von von Angriffen dienen und ansonsten keine Funktion erfüllen. *Honeypots* sind z. B.

Einrichtungen, die einem potentiellen Eindringling einen vermeintlich leicht zu knackenden Weg in das System vortäuschen, um ihn gezielt an falscher Stelle zu beschäftigen und zu identifizieren. *Tarpits* verlangsamen absichtlich den Datenverkehr, um Angreifer abzuschrecken oder um die Ausbreitung von Würmern einzudämmen.

Ein fast immer verwendetes Mittel zur Sicherung von Systemen sind Einrichtungen zur Authentifikation und Autorisierung der Benutzer. *Intrusion-Detection-Systeme* (IDS) werden eingesetzt, um automatisch Einbrüche bzw. Einbruchversuche zu erkennen und Gegenmaßnahmen einzuleiten. Eng verbunden mit dieser aktiven Suche nach Einbrüchen sind Recovery- und Adaptions-Einrichtungen. Beispiele hierfür sind etwa das automatische Ersetzen der aktuellen, eventuell beschädigten Daten mit denen aus einem sicheren Backup und die Änderung der Konfiguration, um diese Form des Angriffs beim nächsten Mal nicht mehr zu ermöglichen. Viele Sicherheitsmaßnahmen zielen darauf ab, es dem Angreifer so schwer wie möglich zu machen, an sicherheitsrelevante Informationen zu kommen. Hierzu zählen die Verschlüsselung von Kommunikationskanälen und Dokumenten und die physische, logische und evtl. auch temporale Separation von Daten. Eine ebenfalls nicht zu vernachlässigende Maßnahme zur Erhöhung der Systemsicherheit ist ein sicherheitsbewußtes Personalmanagement. Schließlich sind *Social Engineering* und vermeintlich vertrauenswürdige Mitarbeiter in der Praxis häufig die Begünstiger von Systemeinbrüchen.

3.2 Identifikation von Bedrohungen

In der Entwurfsphase ist es notwendig, die möglichen Angriffe auf das System im Voraus zu erkennen und diesen entgegenzuwirken. Schneiers Vorschlag der Attack Trees modelliert hierbei die Angriffspunkte als Knoten in einem Baummodell, um die Zusammenhänge von Gefährdungen und deren Bedeutung für das Gesamtsystem zu bestimmen [9]. Die Wurzel eines Attack Tree ist ein mögliches Primärziel des Angreifers, welches durch das System geschützt sein soll. Jeder Knoten eines Attack Tree identifiziert dabei ein mögliches (Teil-)Ziel des Angriffs. Unterschieden wird dabei eine UND-ODER-Zerlegung, d. h., es wird zwischen Zielen unterschieden, zu deren Erreichung sämtliche untergeordnete Teilziele erreicht werden müssen, und Zielen, bei denen es genügt, nur ein Teilziel zu erreichen. Die Blätter eines Attack Tree sind nicht mehr sinnvoll aufzuspaltende Elementarziele. Nahezu alle größeren Systeme haben mehrere Primärziele, die es zu schützen gilt. Deshalb wird für ein System in der Regel ein Wald von Attack Trees identifiziert.

Abb. 3 zeigt ein einfaches Beispiel eines Attack Tree aus einer hohen Sicht auf das System. Das Primärziel des Angreifers ist es, in den Besitz vertraulicher Daten zu kommen. Dazu kann er entweder einen Kommunikationskanal abhören, über einen gestohlenen Prozess auf die Dateien zugreifen oder sich physischen Zugang zu dem Rechner beschaffen. Das Abhören der Netzwerkkommunkation funktioniert z. B. mit einer Man-in-the-Middle-Attacke. Für diese muss der Angreifer sowohl dem Sender der Information vortäuschen, dass er der rechtmäßige

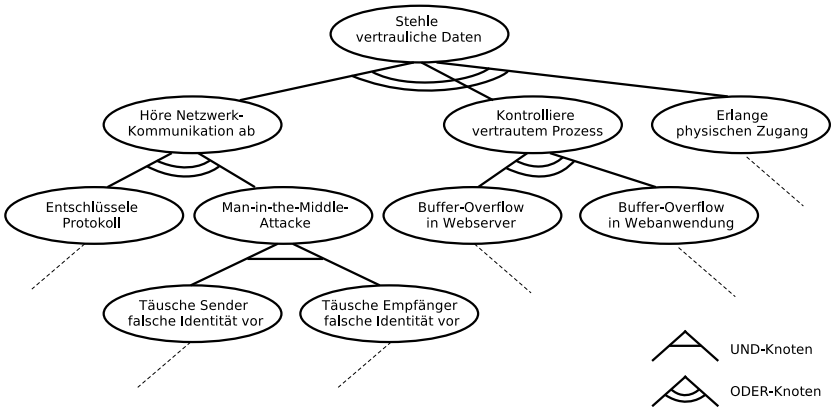

Abbildung 3. Beispiel eines Attack Tree aus einer hohen Sicht auf das System

Empfänger ist, als auch dem vorgesehenen Empfänger vortäuschen, dass er mit dem rechtmäßigen Sender kommuniziert.

Dieses kleine Szenario zeigt, wie intuitiv die Verwendung von Attack Trees ist. Die große Akzeptanz in der Praxis ist sicherlich zum Teil damit zu begründen, aber auch damit, dass die damit ermöglichte Strukturierung und Dokumentation der drohenden Gefahren für das Design sehr nützlich sind. Um die Arbeit mit Attack Trees auf eine theoretisch fundierte Basis zu stellen, haben Mauw und Oostdijk eine formale Spezifikation der Semantik eines Attack Tree definiert [10]. Damit wird es nun möglich, die einzelnen Knoten zusätzlich mit Attributen zu versehen und Transformationen der Bäume in der Designphase vorzunehmen. Häufig werden anstatt reiner Baumstrukturen auch beliebige Graphen verwendet, um die Komplexität großer Systeme besser modellieren zu können (z. B. [11]).

3.3 Modellierung und Überprüfbarkeit von Sicherheit

Die Systemsicherheit ist in der Regel eine qualitative bzw. nicht-funktionale Anforderung im Systementwurf. Das Problem hierbei ist aber, dass die Erfüllung solcher Anforderungen nur schwer kontrollierbar ist. An typischen nicht-funktionalen Anforderungen wie Erweiterbarkeit und Skalierbarkeit zeigt sich, wie schwer es ist, qualitative Eigenschaften zu messen und zu prüfen. Aus diesem Grund empfiehlt es sich, die Sicherheitsanforderungen bei der Modellierung eines Systems explizit als funktionale Anforderungen zu formulieren. Die eher qualitative Forderung „Die Daten sollen sicher sein vor dem Zugriff Unbefugter“ wird dann etwa zu den funktionalen Anforderungen „Kommunikation mit Clients verläuft nur über verschlüsselte Kanäle“, „Clients müssen immer authentifiziert und autorisiert werden“, „Methoden zur Verarbeitung von Client-Daten müssen Stack-Protection besitzen“, etc. Auf diese Weise wird die Erfüllung und die Kontrol-

lierbarkeit von Sicherheitsanforderungen viel weiter in den Vordergrund des gesamten Systementwicklungsprozesses gerückt. Im Folgenden soll ein Werkzeug vorgestellt werden, mit dem solche funktionalen Sicherheitsanforderungen modelliert werden können.

Die *Unified Modelling Language* (UML) ist der Industriestandard zur Beschreibung objektorientierter Systeme. Verschiedene Diagrammtypen und der Einsatz der *Object Constraint Language* (OCL) ermöglichen es, die Architektur und das Verhalten eines Systems nahezu vollständig zu spezifizieren. Die Spezifikation von Sicherheitsanforderungen wird aber nicht durch das Metamodell der UML unterstützt. Aus diesem Grund hat Jürjens *UMLsec* vorgeschlagen, das über den Extensionsmechanismus der UML zusätzlich die Spezifikation von Sicherheitsanforderungen erlaubt [12]. Ein Vorteil der UML ist es, dass mit ihren unterschiedlichen Diagrammtypen sowohl die statischen als auch die dynamischen Zusammenhänge der Objekte modelliert werden können. UMLsec konzentriert sich dabei hauptsächlich auf Klassen-, Verhaltens-, Aktivitäts- und Zustandsdiagramme. Da die physische Plattform ebenfalls ein entscheidender Faktor für die Sicherheit des Systems ist, werden auch Deployment-Diagramme einbezogen.

Das UMLsec selbst ist ein Profil, in dem zusätzliche Elemente definiert sind, um Diagrammelemente zu markieren. Zum einen sind neue *Stereotypen* verfügbar, die das Metamodell der UML um neue Typen mit einer festgelegten Semantik erweitern. Ein Beispiel für die neuen Stereotypen aus UMLsec ist z. B. ≪*encrypted*≫. Er erweitert die Basisklasse ≪*link*≫ und steht für eine verschlüsselte Verbindung zwischen zwei Objekten [13]. Daneben existieren auch *Tag-Value*-Paare zum Beschreiben bestimmter geforderter Eigenschaften. Eine aus Käufersicht faire Transaktion erfordert beispielsweise, dass die gekauften Güter in Ordnung sind und auch geliefert werden. Im Beispieldiagramm von Abb. 4 wird dies durch den *Tagged Value* {*fair exchange = Verkauf O.K.*} spezifiziert.

Der Tag *fair exchange* ist selbst wieder ein Stereotyp, der im UMLsec spezifiziert ist. Diese Sicherheitsanforderung ist in einem Modell nur dann erfüllt, wenn gilt, dass alle Tag-Value-Paare dieses Typs wahr sind, sobald eines wahr ist. Im linken Diagramm besteht z. B. die Möglichkeit, dass ein Kunde zwar bezahlt hat, aber die Lieferung nicht erhält, während der Kunde im rechten Diagramm die Möglichkeit hat, beim Ausbleiben der Lieferung sein Geld zurückzufordern. Auf diese Weise können bereits beim Modellieren eines Systems sicherheitsrelevante Aspekte konsistent und uniform miteinbezogen werden. Weitere Beispiele für den Einsatz von UMLsec in anderen Diagrammtypen sind in [14] zu finden.

Das Ziel, das mit UMLsec verfolgt wird, ist die automatische Verifikation der Sicherheitsanforderungen [15]. Damit lassen sich auch die Auswirkungen von Änderungen der Spezifikation auf ein System effizient analysieren. Zusätzlich ermöglicht die damit forcierte modellbasierte Software-Entwicklung den Einsatz der Codegeneratoren in CASE-Tools. Entsprechend angepasste Codegeneratoren könnten dann die im Modell spezifizierten Sicherheitsanforderungen, wie etwa verschlüsselte Kommunikationskanäle zwischen Komponenten zu großen Teilen automatisch erstellen.

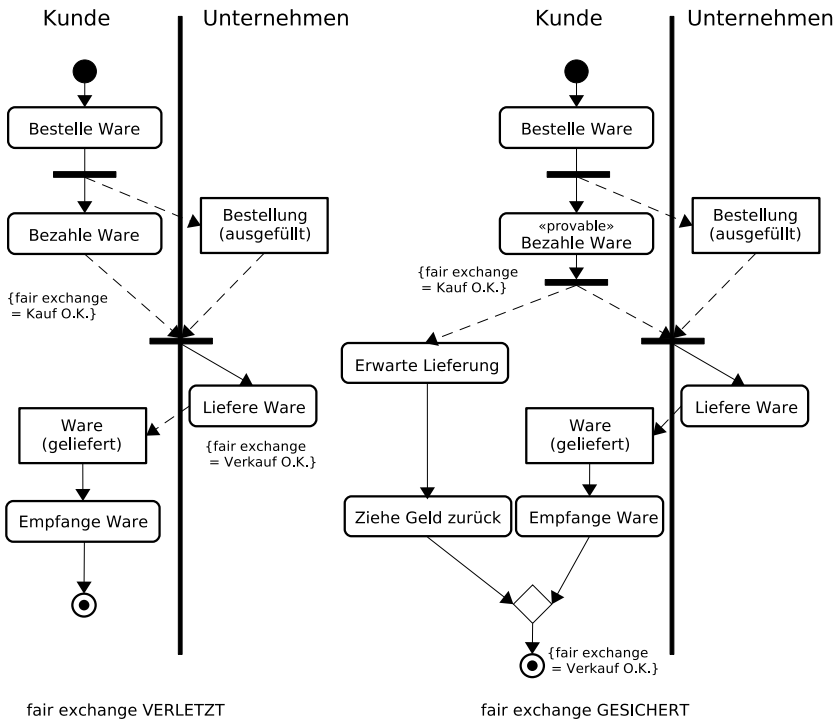

Abbildung 4. Beispiele für Aktivitätsdiagramme mit UMLsec

Die Verifikation der Sicherheit eines Systementwurfs bedeutet aber nicht, dass dieses System automatisch auch sicher ist. Es bedeutet lediglich, dass das modellierte System den spezifizierten Sicherheitsanforderungen genügt. Die Sicherheit eines Systems kann auch im besten Falle nur bis zu der von ihm kontrollierten Grenze beeinflusst werden. Schließlich ist selbst ein mit allen Sicherheitsmechanismen und Verschlüsselungsverfahren versehene System machtlos gegenüber einem böswilligen Administrator, der das System abschaltet.

An diesem Beispiel wird deutlich, wie sehr ein Systemdesign – meist unbewußt – von erfüllt geglaubten Sicherheitsanforderungen ausgeht. Natürlich erwartet man, dass der eingesetzte Administrator verantwortlich und im Sinne der Organisation handelt. Natürlich erwartet man auch, dass die Mitarbeiter nicht leichtfertig mit ihren Passwörtern umgehen und dass der verwendete Compiler auch wirklich nur den Maschinencode erzeugt, den man programmiert hat. Haley et al. nennen diese Annahmen *Trust Assumptions* [16]. Vor diesem Hintergrund kommen die Autoren zu der Überzeugung, dass sich Sicherheit nicht verifizieren lässt. Vielmehr kann im Rahmen des Requirement Engineerings nur überzeugend argumentiert werden, dass ein System die gestellten Sicherheitsanforderungen erfüllt [17].

Diese Betrachtungen zeigen, dass die Sicherheit eines Informationssystems weder zur Entwurfszeit einhundertprozentig sichergestellt werden kann noch, dass sie im Produktionsbetrieb nicht ständig hinterfragt und verbessert werden muss. Gerade die (Sicherheits-)Wartung spielt daher in DAIS eine besonders wichtige Rolle, da sie aufgrund ihrer dynamischen Umwelt und der Wichtigkeit der verwalteten Daten für Angriffe besonders interessant sind. Wie bereits erwähnt, kann ein System aber nur innerhalb seiner Systemgrenzen Sicherheitsmaßnahmen ergreifen. Vor diesem Hintergrund ist es für ein adaptives System wichtig, seine Kooperations- und Kommunikationspartner einschätzen zu können. Diese Fragestellung wird im folgenden Abschnitt näher betrachtet.

3.4 Vertrauensnetzwerke

Vielfach müssen Informationssysteme heute mit anderen Parteien kooperieren und verlagern damit einen Teil ihrer Dienstverantwortung auf Ressourcen, die nicht unter ihrer Kontrolle stehen. Dies können zum einen die Anwender selbst sein, die über eine Benutzerschnittstelle wie etwa einem Webbrowser mit dem System interagieren. Es werden aber vor allem immer mehr andere Systeme, wie z. B. die Informationssysteme der Vertriebspartner und Zulieferer, mit denen über Technologien wie Web Services kommuniziert wird. In solchen B2B-Netzwerken gibt es natürlich Vertrauensunterschiede. Langjährigen Partnern, bei denen man vielleicht auch weiß, wie gewissenhaft deren System administriert wird, bringt man ein größeres Vertrauen entgegen als anderen. Man wird ihnen z. B. auch den Zugriff auf eigene Ressourcen gewähren, der anderen verwehrt bleibt. Aber auch der eigene Host kann zu einer Gefahr werden, falls einer seiner Prozesse kompromittiert wird.

Für keines dieser Szenarien gibt es, ebenso wie für das eigene System, eine Möglichkeit, Sicherheit garantieren zu können. Allerdings müssen Wege gefunden werden, um dieses Risiko zu minimieren. Die wichtigste Maßnahme, bevor irgendeine Art von Kooperation eingegangen wird, ist die Authentifikation des Partners. Nachdem die Identität des Partners sichergestellt ist, muss abhängig davon entschieden werden, wie sehr man diesem vertraut und welche Rechte man ihm zuspricht. Da in einer dynamischen Umgebung nicht sämtliche Kommunikationspartner vorab bekannt sind, aber eine Zusammenarbeit möglichst schnell realisiert werden soll, bedient man sich sogenannter Vertrauensarchitekturen. Die Grundidee hinter Vertrauensarchitekturen ist es, dass man seine Vertrauenseinschätzung in einzelne Parteien innerhalb eines Netzwerkes auf die Einschätzungen anderer Parteien stützen kann. Diesen wiederum vertraut man selbst oder ihre Vertrauenswürdigkeit wird selbst wieder durch andere gestützt. Zum Einsatz kommen hier *Public-Key-Infrastrukturen* (PKI). Der Einsatz von symmetrischen Schlüsseln ist für vertrauensvolle Kommunikation aufgrund der riesigen Anzahl an Schlüsseln unpraktikabel. Grundsätzlich unterscheidet man bei Vertrauensarchitekturen hierarchische und dezentrale Strukturen.

Bei einem hierarchischen Aufbau des Vertrauensmodells spricht man von einer *Trust Center Hierarchy*. Der Identitätsnachweis einer Partei geschieht mit

Hilfe von Zertifikaten. Diese werden von einer allgemein anerkannten *Certification Authority* (CA) ausgestellt. Zum Nachweis, dass es sich wirklich um ein echtes und gültiges Zertifikat handelt, wird eine *Public-Key-Verschlüsselung* eingesetzt, mit der eine CA die Zertifikate signiert. Der Industriestandard für solche Zertifikate ist *X.509* (RFC 3280). Sobald das Zertifikat einer Partei nicht mehr sicher ist, weil es z. B. „geknackt“ wurde, wird es in eine *Certificate Revocation List* eingetragen, um dies Interessenten mitteilen zu können. Die Authentifizierung einer Partei wird entweder über die gemeinsame CA beider Parteien oder über eine der höheren Hierarchiestufen (vgl. Abb. 5) vorgenommen. Eine *Policy Certification Authority* (PCA) stellt selbst keine Zertifikate für Benutzer aus. Sie zertifiziert nur den öffentlichen Schlüssel von Certification Authorities. An oberster Stelle steht die *Internet Registration Policy Authority* (IRPA). Sie ist eine behördliche Einrichtung, die als oberste Vertrauensinstanz einsteht. Dieses Vertrauensmodell ist heute Standard im Internet und wird von Firmen und auch von vielen privaten Webseiten genutzt.

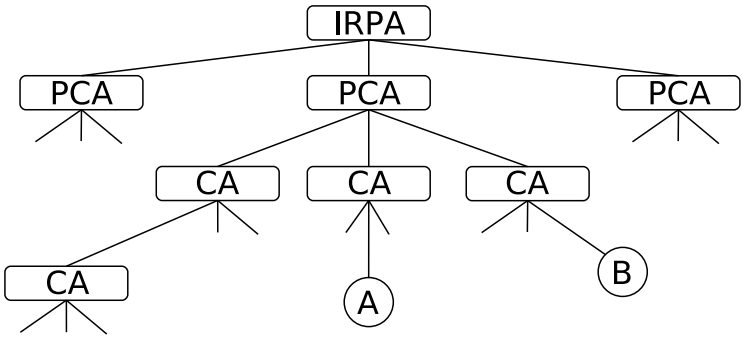

Abbildung 5. Aufbau eines hierarchischen Vertrauensnetzwerkes

Camp [18] kritisiert die hierarchische Struktur aus mehreren Gründen. Zum einen sieht er die Gefahr, dass sämtliche Sicherheitsgarantien auf ein einzelnes Zertifikat einer einzelnen Partei beschränkt sind. Kann ein Benutzer oder ein System dieses Zertifikat vorlegen, wird ihm der volle Zugriff gewährt. Auf eine feingranulare Zugriffskontrolle wird zur Reduktion von Rechenzeit und Verwaltungsarbeit verzichtet. Ein weiteres Problem ist die Abhängigkeit von der jeweils übergeordneten Instanz. Beim Ausfall einer hohen oder gar der höchsten Instanz wird die komplette Vertrauensbasis außer Betrieb gesetzt. Dieser Fall ist für ein DAIS besonders kritisch, da ein Problem der eigenen CA genügt, um das System interaktionsunfähig zu machen. Als letztes sieht Camp noch das Problem, dass bei einem starken Wachstum und starker Nutzung der Hierarchie Geschwindigkeitsengpässe entstehen, die den reibungslosen Betrieb gefährden.

Dezentrale Modelle versuchen die Nachteile, die aus der Abhängigkeit von einer übergeordneten Instanz entstehen, zu vermeiden. Dieser stark propagierte

und vielversprechende Ansatz fordert den Aufbau eines *Web of Trust* (Vertrauensnetzwerk). Hierbei agiert jede teilnehmende Partei als Certification Authority, wobei jede sich selbst ein digitales Zertifikat bzw. Schlüsselpaar erzeugt. Den öffentlichen Schlüssel schickt sie dann an einen Schlüsselserver, der für alle zugänglich ist. Das Problem besteht nun darin, festzustellen, ob der öffentliche Schlüssel eines Kommunikationspartners auch wirklich zu diesem gehört. In der hierarchischen Struktur wird diese Funktion durch die vertrauenswürdigen CAs übernommen. Die Idee des Web of Trust liegt nun darin, dass man sich entweder über direkte, gesicherte Wege sich den richtigen öffentlichen Schlüssel besorgt oder dass man anderen Teilnehmern vertraut, die bezeugen, dass es sich um den korrekten Schlüssel handelt. In Abb. 6 ist das Schema exemplarisch dargestellt.

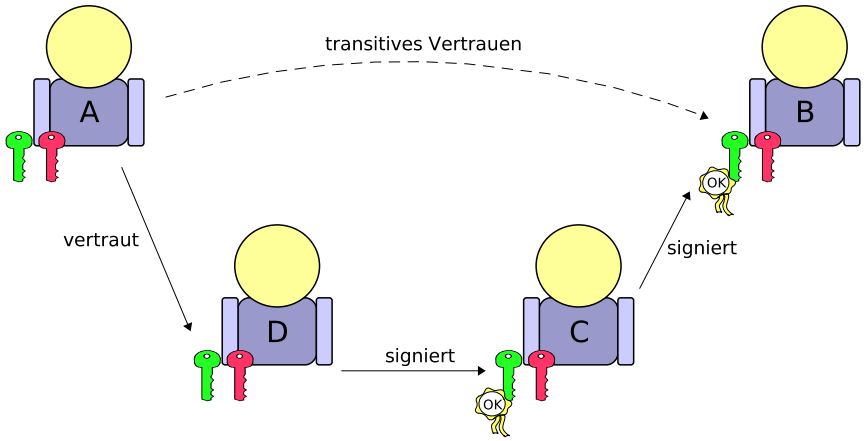

Abbildung 6. Transitives Vertrauen in einem dezentralen Web of Trust

Mitglied C, welches den öffentlichen Schlüssel eines anderen Teilnehmers B garantiert identifizieren kann, signiert diesen mit seinem eigenen Schlüssel und hinterlegt dies bei einem Schlüsselserver. Ein Interessent A kann nun von dem Schlüsselserver diese Bestätigung von B erfragen und entscheiden, ob er C und evtl. anderen Parteien, die den öffentlichen Schlüssel von B signiert haben, vertraut. Wenn er dies tut, kann er sich die umständlichere direkte Schlüsselanfrage bei B sparen. Diese Lösung nennt man transitives Vertrauen. Ab einer gewissen Größe des Netzes ist es jedoch unwahrscheinlich, dass man immer jemanden kennt, der einem den öffentlichen Schlüssel eines Kommunikationspartners bestätigen kann. Wird Cs Schlüssel nun von einer Partei D signiert, deren öffentlicher Schlüssel A bereits bekannt ist und der A vertraut, kann A davon ausgehen, dass der Schlüssel von B korrekt ist. Auf diese Weise wird ein Web of Trust, als ein untereinander eng verknüpftes Netzwerk aus Vertrauensaussprachen, aufgebaut. Viele kurze und disjunkte Wege in diesem Netzwerk aus Schlüsselbestätigungen sind dabei eine Art Maß für eine hohe Vertrauenswürdig-

keit. Die bekannteste Form eines solchen Vertrauensnetzwerkes ist das *Pretty-Good-Privacy*-Netzwerk (PGP), das für vertrauenswürdigen Email-Verkehr eingesetzt wird. Hier haben die Teilnehmer zusätzlich die Möglichkeit, in verschiedenen Abstufungen anzugeben, wie sehr sie dem signierten öffentlichen Schlüssel eines anderen vertrauen. Anhand dieser Gewichtung kann dann die spezifische *Key Legitimacy* errechnet werden.

Der dezentrale Ansatz hat aber auch einige Nachteile. Zum einen besteht die Gefahr manipulierter Schlüsselserver. Diese sollten daher ebenfalls möglichst sicher sein und ihre Daten regelmäßig untereinander austauschen, um die Auswirkungen eines kompromittierten Schlüsselservers einzudämmen. Die signierten Schlüssel können aber auch direkt zwischen den Hosts über sichere Kanäle ausgetauscht werden. Dadurch wird jedoch die Suche und die Invalidierung von signierten Schlüsseln wesentlich komplexer. Gerade im unternehmerischen Einsatz ist auch die Rechtssicherheit ein wichtiger Punkt. Geltende Gesetze bauen aufgrund der Verfolgbarkeit der Identifikationswege und der grundlegend höheren Vertrauenswürdigkeit nur auf den hierarchischen Ansatz. Die dezentrale Struktur birgt nämlich die Gefahr geringerer Sicherheit, da es in ihr modellbedingt keine höhere Instanz gibt, der man recht sicher vertrauen kann. Diesem Nachteil kann aber durch eine enge Vermaschung des Netzes und durch regelmäßige Kontrolle des bereits ausgesprochenen Vertrauens entgegengewirkt werden. Dies entspricht auch dem Umgang mit Vertrauen in der realen Welt. Hier ist Vertrauen keine statische Eigenschaft über einen bestimmten Zeitraum, sondern es wird mitunter nach jeder Interaktion neu evaluiert.

In diesem Abschnitt wurde der Fokus auf das bisher größte Einsatzgebiet von Vertrauensmodellen, nämlich der sicheren Identifikation von Kommunkationspartnern gelegt. Der Vertrauensbegriff kann aber auch in anderen Gebieten eine Rolle in Informationssystemen spielen. Je nach Vertrauenseinschätzung des Systems kann z. B. nur eine Untermenge der angebotenen Dienste zur Verfügung gestellt werden. Oder ein service-orientiertes System kann entscheiden, dass es die Anfrage eines Clients mit unklarem Vertrauensstatus auf einem Rechner mit einer besonders sicheren, aber aufwendigen Implementierung ausführen will, während vertrauenswürdige Anfragen auf Servern mit schnelleren, aber weniger abgesicherten Implementierungen bearbeitet werden. Die Globe-Architektur, die im Folgenden vorgestellt wird, nutzt ein hierarchisches Vertrauensmodell, um neben der Authentifizierung von Clients auch die Autorisierung dieser vorzunehmen sowie die Funktionalität von Komponenten auf bestimmten Hosts einzuschränken.

3.5 Sicherheit in der Middleware

Ein wichtiger Aspekt für die Entwicklung und den Betrieb von gut skalierenden Systemen mit Mehrschichtarchitektur ist die Bereitstellung einer leistungsfähigen Middleware. Neben der Realisierung verteilter Kommunikation, der Integration heterogener Daten und dem Transaktionsmanagement wird hier häufig auch eine Lastbalancierung vorgenommen. Gängige Infrastrukturen sind hier etwa J2EE und CORBA. Mit Hilfe von Verzeichnisdiensten können Clients die

verteilten Dienste lokalisieren und nutzen. Zu besseren Skalierbarkeit können die Dienste, die in Form verteilter Objekte angeboten werden, auch geclustert werden. Durch diese Redundanz erhöht sich natürlich auch die Ausfallsicherheit solcher Systeme. Da Middleware nur die Infrastruktur, nicht aber die Funktionalität des Systems bereitstellt, ist hier die Umsetzung zweier unterschiedlicher Sicherheitskonzepte möglich. Von programmierter Sicherheit spricht man, wenn in der realisierten Funktionalität eigene Sicherheitsmechanismen implementiert werden. Um die Anwendungsentwicklung aber flexibler und schneller zu gestalten, wird in der Middleware jedoch häufig deklarative Sicherheit genutzt. Hier werden die Sicherheitsmechanismen von der Middleware als Dienst bereitgestellt und können beim sog. *Deployment* der Anwendung konfiguriert werden.

Als Beispiel für deklarative Sicherheit in der Middleware ist das Sicherheitskonzept von J2EE schematisch in Abb. 7 dargestellt. Für den Zugriff auf die Methoden von EJBs und Web-Ressourcen werden bestimmte Rollen definiert, die Zugriff darauf haben. Diese werden vom *Application Provider* beim Deployment der Applikation definiert. Die *Container Security Services* sorgen dafür, dass der Zugriff nur gewährt wird, wenn das entsprechende *Subject* (Benutzer, Rechner) vorher mit Hilfe der *Java Authentication and Authorization Services* (JAAS) authentifiziert und autorisiert wird. Für ein Subject werden dabei die entsprechenden *Credentials* (Identitätsbelege) wie Passwörter, Zertifikate etc. und *Principals* (Identitäten) wie z. B. der Benutzername festgestellt. Die *Transport Level Security* kümmert sich um die Verschlüsselung der Kommunikationskanäle, um vertrauenswürdige Kommunkation zu garantieren.

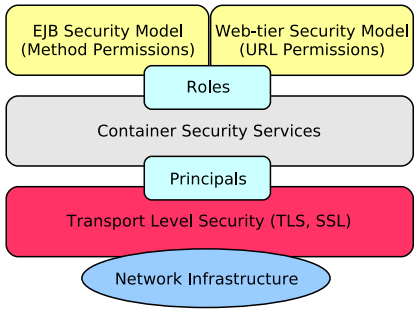

Abbildung 7. Sicherheitskonzept der J2EE

Mit diesen Maßnahmen können die Kommunikation und die Zugriffe auf die System-Ressourcen gesichert werden. Ein Problem bleibt jedoch unbeachtet. Es existiert kein Mechanismus, um die Sicherheit des Hosts, auf dem die Komponenten installiert sind, zu kontrollieren. Sowohl andere Prozesse im System als auch die Laufzeitumgebung der Komponente können eine Bedrohung sein. Werden die Komponenten auch noch aus Performanzgründen auf andere Rechnern repliziert, kann nicht sichergestellt werden, dass diese nicht bereits kompromit-

tiert sind oder es sich überhaupt um den richtigen Host handelt. Gerade also bei der Replikation von Objekten gibt es Schwierigkeiten, die Sicherheit des Systems zu garantieren.

Die Globe-Architektur ist eine Middleware für verteilte Objekte, bei der eine Komponente selbst entscheidet, für wie vertrauenswürdig sie einen Host hält und welche Funktionen sie über diesen Host anbietet [19]. Das zentrale Konzept in Globe sind *Distributed Shared Objects* (DSO). Ein DSO ist ein eindeutig identifizierbares logisches Objekt, welches aus mehreren *Local Objects* besteht. Replikas nennt man diejenigen Local Objects, die einen Teil des Zustandes des DSO speichern. Der Zustand eines DSO ist demnach definiert durch die Gesamtheit aller Replikas (Abb. 8).

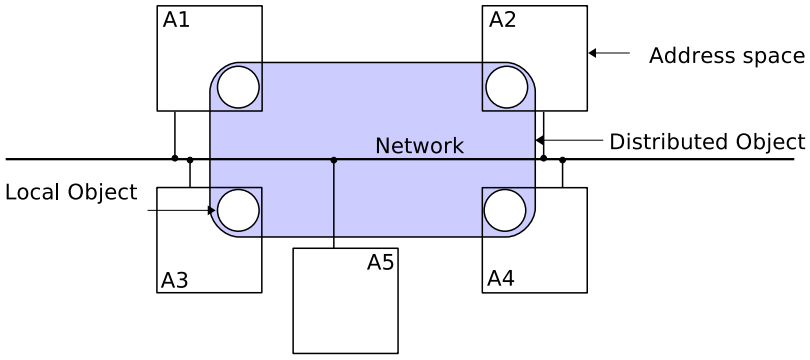

Abbildung 8. Verteilung eines DSO über vier Adressräume

Damit ein Client ein DSO nutzen kann, muss er ein lokales Objekt in seinem Adressraum erstellen, welches dann in der Regel als Proxy für den Zugriff auf eine Replika dient. Das Ziel dieser Architektur ist es, die Replikas dynamisch und so nahe wie möglich beim Benutzer zu platzieren, um die Performanz zu steigern. Die Replikas werden mit Hilfe der ID des DSO über einen *Location Service* gefunden. Die Replikas selbst laufen auf einem *Globe Object Server*, den man mit dem Object Request Broker der CORBA-Welt vergleichen kann.

Der Aufbau einer Replika ist in Abb. 9 skizziert. Das *Communication Subobjekt* ist zuständig für die Kommunikation mit Local Objects in anderen Adressräumen. Das *Replication Subobjekt* sorgt dafür, dass der Zustand der Replika mit den anderen Replikas konsistent bleibt. Das *Control Subobject* dient der Vermittlung zwischen dem Replication Subobject und dem Semantics Subobject. Darüber hinaus ist es analog zu einem Skeleton in CORBA zuständig für die Verarbeitung von Client-Anfragen. Im *Semantics Subobject* ist schließlich die eigentliche Funktionalität gekapselt. Für jedes DSO kann eine eigene *Security Policy* definiert werden, dessen Einhaltung durch das Security Subobject gewährleistet wird.

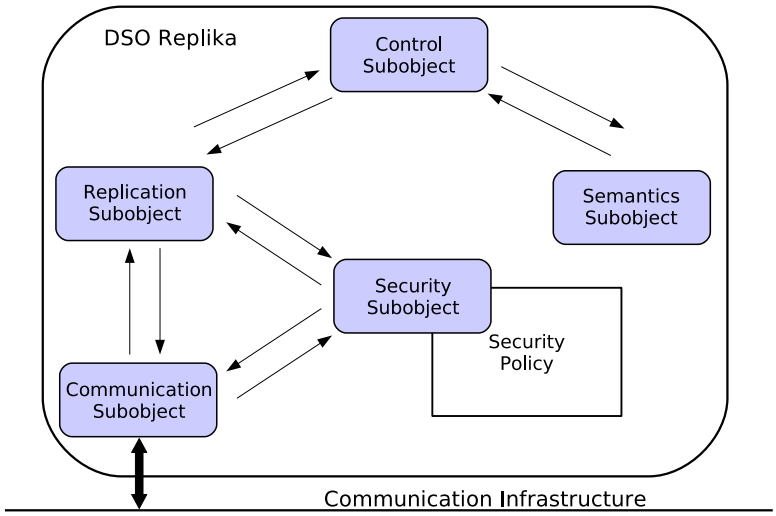

Abbildung 9. Struktur einer Replika in Globe

Das Globe-Sicherheitskonzept hat drei Ziele, nämlich sichere Bindungen, Plattformsicherheit und den sicheren Aufruf von Methoden. Die Grundlage davon ist das *Globe Trust Model* [20]. Jedes DSO und jede seiner Replikas besitzen ein Paar aus öffentlichem und privatem Schlüssel, die für die Authentifizierung und Zugangskontrolle genutzt werden. Digitale Zertifikate für Benutzer, Replikas oder administrative Tätigkeiten werden genutzt, um Rechte an einen öffentlichen Schlüssel zu binden. Wer die Kenntnis des passenden privaten Schlüssels nachweisen kann, erhält diese Rechte. In den Benutzer-Zertifikaten wird spezifiziert, welchen Benutzern die Ausführung bestimmter Methoden des DSO gestattet wird. Die Zertifikate einer Replika beschreiben, welche Methoden sie anbieten darf. Auf diese Weise können z. B. sehr sicherheitskritische Methoden auf diejenigen Replikas eingeschränkt werden, die zu einem kleinen Kreis an Kern-Replikas auf besonders sicheren Rechnern gehören. Die Administrations-Zertifikate spezifizieren, welche Arten von Zertifikaten erstellt werden dürfen. Jedes Zertifikat ist entweder mit dem privaten Schlüssel des DSO oder eines Adminsitrations-Zertifikates signiert. Dadurch entstehen Zertifikationsketten, die den Vertrauensraum in einem Globe-System aufspannen.

Dieses Verfahren würde aber in einem realen System mit sehr vielen DSOs schlecht skalieren. Aus diesem Grund wurde die Möglichkeit geschaffen, die Vertrauenshierachie in eine bestehende externe Vertrauenshierarchie wie z. B. dem Netz des ISP einzubinden. Es besteht auch die Möglichkeit, dass eine Organisation die öffentlichen Schlüssel ihrer DSOs mit einem von einer CA zertifizierten Schlüssel signiert. Jedes DSO bietet deshalb eine Methode an, mit der solche externe Zertifikate abgefragt werden können. Auf diese Weise kann vermieden

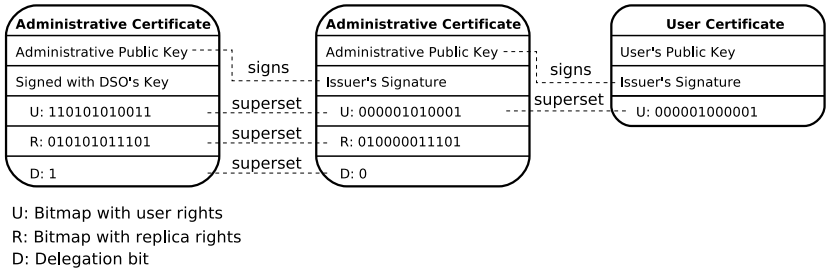

Abbildung 10. Zertifikationskette in Globe

werden, dass die einzelnen Vertrauensursprünge unnötig oft zurückverfolgt werden müssen.

Unter sicherer Bindung versteht man in Globe die gesicherte Verbindung von DSO und öffentlichem Schlüssel, Replika und DSO sowie DSO und repräsentiertem Geschäftsobjekt. Um eine gesicherte Bindung zwischen dem DSO und seinem öffentlichen Schlüssel zu garantieren, wird der öffentliche Schlüssel zu einem Teil der ID des DSO. Damit ist die eindeutige Zusammengehörigkeit gesichert. Wie bereits erwähnt, besitzt jede Replika ein digitales Zertifikat, das spezifiziert, welche Methoden durch diese Replika ausgeführt werden dürfen. Dieses Zertifikat ist entweder mit dem Schlüssel des DSO signiert oder wird durch die Zertifikatkette transitiv mit dem DSO verbunden. Somit ist garantiert, dass die Replika auch wirklich zu dem DSO gehört. Um als Benutzer sicher gehen zu können, dass das benutzte DSO auch wirklich der Dienst ist, denn man in Anspruch nehmen will, gibt es mehrere Möglichkeiten. Zum Beispiel kann der Objekt-Handle über eine sichere HTTP-Verbindung von der Webseite des Anbieters bezogen werden. Im Falle einer Bank wäre auch denkbar, dass der Kunde von der Bank einen Datenträger mit der ID des DSO bekommt, der zusätzlich noch Informationen enhält, wie das DSO gefunden werden kann.

Die Plattformsicherheit bei Globe beinhaltet zwei Aspekte. Der Host, der eine Replika ausführen soll, wird dadurch geschützt, dass die Replikas nur innerhalb einer Sandbox ausgeführt werden. Darüber hinaus, muss eine Replika, welche aus Performanzgründen eine neue Instanz auf einem anderen Host installieren möchte, diesem beweisen, dass sie die nötigen Administrationsrechte besitzt und dass sie wirklich ein Teil des entsprechenden DSO ist. Nach dieser Vermittlungsphase erstellt die Administrativ-Replika auch ein Zertifikat für die neue Replika. Schwieriger, wenn nicht gar unmöglich, ist es, die Replikas selbst vor kompromittierten Hosts zu schützen. Es wird aber zumindest versucht, das Risiko zu minimieren. Der vorgestellte Mechanismus, mit dem eine Replika nur die Funktionalität anbieten darf, die ihr Zertifikat erlaubt, ist nur ein Punkt. Ein weiterer ist die Zusammenfassung von mehreren Replikas auf sicheren, selbst kontrollierten Servern zu Kern-Replikas. Nur von diesen dürfen dann z. B. Änderungsoperationen durchgeführt werden. Eine Replika auf einem unsicheren Host

darf dann zwar Leseoperationen durchführen und kann dem Benutzer somit auch falsche Daten liefern, allerdings verursacht sie damit nur bei den mit ihr kommunizierenden Clients Fehler und der Rest des Netzwerkes bleibt unbeeinflusst. Ein dritter Mechanismus, der angewendet werden kann, wird *State Signing* genannt. Dabei werden die Daten, bevor sie zum Client ausgeliefert werden, mit dem privaten Schlüssel der Replika signiert und das Semantics Subobject des Clients kann dann die Echtheit der Antwort überprüfen. Auf diese Weise können auch kompromittierte Laufzeitumgebungen die Daten einer Replika nicht beeinflussen. Lediglich gegen ein Denial-of-Service kann sich eine Replika auf einem Host nicht schützen.

Um den sicheren Aufruf von Methoden zu ermöglichen, ist ebenfalls wieder der Zertifikatmechanismus zuständig. Ein Benutzer muss zum einen die Rechte besitzen, um eine Methode auf einer Replika aufzurufen und diese muss wiederum die entsprechenden Rechte zum Ausführen der Methode besitzen. Um auch die Kommunikation beim Aufruf der Methode zu sichern, werden sichere Kanäle verwendet.

4 Sicherheitsstrategie: Intrusion Detection

Eine sicherheitsbewußte Systemarchitektur alleine ist aufgrund der ständig neuen Bedrohungen für ein DAIS unter Umständen nicht mehr ausreichend. Wie aus den bisherigen Ausführungen deutlich wird, machen es die Komplexität der Systeme und die geforderte Interoperabilität nahezu unmöglich, ein lückenloses Sicherheitskonzept zu planen und zu implementieren. Um die Sicherheit dennoch weiter zu erhöhen, werden heute verstärkt Maßnahmen getroffen, um neue, unbekannte Angriffe zur Laufzeit zu erkennen und deren Einfluss auf das Gesamtsystem zu minimieren. In diesem Zusammenhang spricht man häufig von einem *intrusion-aware design*. Die Funktionalität des Systems wird dabei so realisiert, dass sie tolerant gegenüber Angriffen wird und es ermöglicht, den Betrieb aufrecht zu erhalten.

4.1 Grundlagen

Die Grundannahme, auf der Intrusion Detection basiert, ist folgende: „Das Ausnutzen von Systemschwachstellen bedingt eine anormale Nutzung, weshalb Verletzungen der Systemsicherheit anhand dieser anormalen Nutzungsmuster erkannt werden können.“ [21]. Ein *Intrusion Detection System* (IDS) arbeitet unabhängig von der eigentlichen Funktionalität der Systemteile als eine Art Kontrollschleife, die kontinuierlich das System überwacht. Es besteht im wesentlichen aus einer Menge von Sensoren, einer Auswertungs- und Beurteilungskomponente (Detektor) und einer Handlungskomponente. Mit Hilfe der Sensoren gelangen Informationen über den Zustand von Systemteilen zur Auswertungs- und Beurteilungskomponente. Dort werden diese Informationen darauf hin ausgewertet, ob ein Angriff stattfindet oder gar schon stattgefunden hat. Die Ergebnisse dieser Auswertungen werden von der Handlungskomponente genutzt, um aus den Reaktionsmöglichkeiten die adäquaten auszuwählen und diese durchzuführen.

Die Sensordaten liefern die zu analysierenden *Auditdaten* und können von den unterschiedlichsten Komponenten auf verschiedenen Systemebenen generiert werden. Auf der Netzwerkebene können z. B. Portscans, ein steiler Anstieg der Verbindungsanfragen oder ein stark verlangsamter Netzdurchsatz Symptome für einen Angriff sein. Auf Betriebssystemebene können ungewöhnliche Muster in Systemaufrufen, neue oder veränderte Dateien und unbekannte, neue Prozesse Symptome sein. Auf der Anwendungsebene hingegen können viele ungültige Anmeldungsversuche oder abgestürzte Systemteile als mögliche Anzeichen eines Angriffes erfasst werden [22].

Bei der Detektion von Angriffen unterscheidet man zwei grundlegende Herangehensweisen [23]. Die statistische Anomalienerkennung basiert auf Schwellwerten oder definierten Nutzungsprofilen. Alle Vorgänge im System werden dabei als potentielle Angriffe angesehen, wenn sie ein vorher als normal definiertes Ausmaß überschreiten oder wenn sich ein Benutzer in einer ihm untypischen Weise verhält. Die regelbasierte Erkennung hingegen definiert bestimmte Vorkommnisse, deren Auftreten einen Angriff auf das System bedeuten kann. In der Praxis kommen zumeist aber nur anomaliebasierte Ansätze zum Einsatz, da der regelbasierte Ansatz schwer zu realisieren ist und davon ausgeht, dass sämtliche Angriffssymptome bekannt sind.

Ist ein potentieller Angriff erkannt worden, muss auf diesen reagiert werden. Viele existierende Systeme belassen es dabei, eine entsprechende Nachricht an den Systemadministrator zu schicken, der dann manuell Maßnahmen einleitet. Für hochdynamische Systeme ist die damit einhergehende Verzögerung und der immense Aufwand jedoch nicht zufriedenstellend. Es müssen weitere Vorkehrung getroffen werden, um der Handlungskomponente einen Katalog an Steuerungsmöglichkeiten bereitzustellen, damit diese den Betrieb aufrecht erhalten kann. In einem idealen Szenario wäre das etwa die Trennung der Kommunikationsverbindungen, ein automatisches Recovery von potentiell veränderten Daten und der Neustart der betroffenen Komponente. Aber auch kleinere Eingriffe, wie die temporale Einschränkung auf bestimmte Dienste und die Umleitung auf Dienstimplementierungen mit strengeren Sicherheitsmechanismen können angemessen sein, wenn der Angriffsverdacht nicht sicher ist.

4.2 Probleme in adaptiven Systemen

Für DAIS ergeben sich vor allem bei der Informationsversorgung und der Auswertung vielerlei Probleme. Die benötigten Auditdaten können aus allen Systembereichen herangezogen werden, um einen Angriffsverdacht zu untermauern. Wie aber bereits in der Einleitung angesprochen, ist die Schichtenarchitektur von Informationssystemen genau die Hürde, die einer systemweiten Zustandsbestimmung widerspricht. Das Abstraktionsparadigma bietet keine Möglichkeit, Daten aus den einzelnen Schichten zu kombinieren und globale Angriffsmuster zu erkennen. Allerdings ist das Erkennen eines Angriffes die notwendige Voraussetzung für die Einleitung von Abwehr- bzw. Recovery-Maßnahmen. Daraus ergeben sich zwei Lösungsmöglichkeiten. Entweder wird auf jeder Schicht ein

eigenes, unabhängiges IDS eingerichtet oder die strikte Kapselung wird aufgegeben. Jede Schicht stellte dann eine Schnittstelle zur Verfügung, über die sie Auditdaten an ein systemweites IDS liefert und Steuerungsoptionen zur Verfügung stellt.

Der erste Ansatz hat den Vorteil, dass die Trennung und somit die Austauschbarkeit der Schichten gewährleistet bleibt und dass das IDS eventuell auf die Gegebenheiten der jeweiligen Schicht besser angepasst werden kann. Allerdings hat man mit dem geringeren Fokus der Informationsversorgung auch weniger Informationen zur Angriffsdetektion. Zudem blieben die Reaktionsmöglichkeiten auf die jeweilige Schicht begrenzt. Die schichtenübergreifende Detektion hat hier den Vorteil, dass sehr viel mehr Informationen und auch feinere Steuerungsmöglichkeiten über alle Ebenen zur Verfügung stehen. Diese Freiheit würde aber mit einer sehr viel aufwendigeren Auswertung und hohen Kommunikationskosten teuer erkauft. Auch die Konzeption eines IDS, welches die Daten heterogener Systemteile unterschiedlichster Hersteller sinnvoll verarbeiten soll, ist eine mehr als schwierige Aufgabe. Als möglicher Kompromiss zwischen diesen beiden Ansätzen wäre eine standardisierte Kooperation zwischen den IDS der einzelnen Schichten denkbar.

Unabhängig von der Realisierung der Datenversorgung und Reaktionsmöglichkeiten ist das dynamische Umfeld von DAIS ein Problem bei der Erkennung von Angriffen. Es stellt sich die Frage, welche Auditdaten überhaupt verwendbar sind. Statistische Anomalienerkennung beruht hauptsächlich auf Merkmalen wie Durchsatz und Häufigkeit von Aktionen und betrachtet deren Abweichungen vom durchschnittlichem Systemverhalten. Solche Schwankungen sind aber gerade einer der Hauptaspekte eines DAIS und stellen in diesem Rahmen keine Anomalie dar. Für ein IDS wäre es nicht unterscheidbar, ob es sich nun um einen Angriff handelt oder um einen „normalen“ Wechsel der Systemlast. Der regelbasierte Ansatz steht vor dem gleichen Problem. Zudem stellt sich die Frage, wie das Auswertungsmodell erzeugt werden kann. Die manuelle Erzeugung von Modellen durch Experten ist viel zu aufwändig und zu teuer. In der Regel werden für IDS daher Data-Mining-Verfahren angewendet, um auf einer großen Trainingsmenge an Sensordaten die Modelle zu erlernen. Allerdings ist auch die Erzeugung dieser Trainingsdaten sehr aufwändig und somit teuer. Eskin et al. schlagen daher die adaptive Erzeugung von Modellen vor [24]. Ihr Vorschlag beinhaltet eine zusätzliche Generatorkomponente, die im laufenden Betrieb das Auswertungsmodell für den Detektor liefert und dieses bei Bedarf auch erneuern kann. Dieser Ansatz ist so gewählt, dass er unanhängig vom Algorithmus zur Generierung des Modells ist. Der von den Autoren vorgeschlagene Algorithmus beispielsweise, speichert die Auditdaten in einer Datenbank und versucht dann mit probabilistischen Analysen Anomalien in den gesammelten Daten zu erkennen. Von einer solchen Architektur, die bei der Generierung neuer Modelle auch die sich ändernde Systemlast berücksichtigt, könnte somit auch ein DAIS profitieren. Da das Problem der wenig aussagekräftigen Auditdaten aber generell in einem dynamischen Umfeld besteht, scheint ein Ausweichen auf andere Merkmale die einzige Alternative.

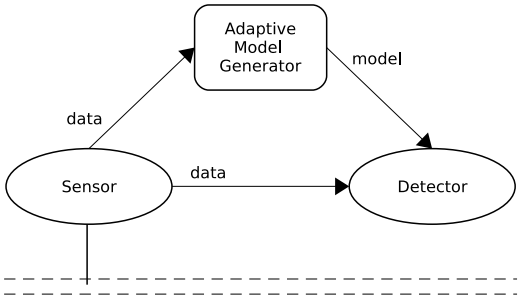

Abbildung 11. Aufbau zur adaptiven Modellerzeugung

5 Zusammenfassung

Informationssysteme sind aufgrund der von ihnen verwalteten Daten ein lohnendes Ziel für Angriffe. Umso wichtiger ist es, dass bei ihrer Entwicklung ein besonderes Augenmerk auf die Sicherheit der Systeme gelegt wird. Vor allem in Systemen, in denen eine hohe Verlässlichkeit und eine hohe Anpassungsfähigkeit gefordert wird, nehmen Sicherheitsmaßnahmen eine besondere Stellung ein. Zum einen sind sie ein Erfordernis für einen verlässlichen Betrieb, aber gleichzeitig sind sie auch nur schwer in adaptiven Architekturen umzusetzen. Zwei wichtige Stützpfeiler für die zukünftige Realisierung von Sicherheit in DAIS sind die vertrauenswürdige Nutzung von Ressourcen und die automatische Reaktion auf Angriffe. Für die Etablierung von Vertrauen innerhalb eines Informationssystems bieten sich Zertifikate oder Public-Key-Verfahren an. Anhand des Beispiels der Globe-Architektur wurde gezeigt, wie dieser Ansatz ausgeweitet werden kann, um auch die Vertrauenswürdigkeit eines Hosts zu sichern. Die Erkennung und Abwehr von Angriffen auf ein DAIS hat mit dem Problem zu kämpfen, dass die meisten der konventionell genutzten Systeminformationen in einer hochdynamischen Umwelt nicht aussagekräftig sind. Mit der Entwicklung neuer, lastunabhängiger Sicherheits-Indikatioren würde aber einer intensiven Nutzung der effektiven Steuerungsmöglichkeiten von Intrusion-Detection-Systemen nichts mehr im Wege stehen. Abschließend bleibt ebenso festzuhalten, dass der Entwicklungsprozess die zentrale Rolle bei der Entwicklung neuer und sicherer Informationssysteme spielt. Sicherheit muss hier nicht als eine Eigenschaft, sondern als eine Funktion des Systems angesehen werden. Unterstützen kann man diesen Prozess durch den Einsatz von Analyse-Instrumenten wie Attack Trees und dem UMLsec-Profil als Erweiterung der Modellierungssprache UML.

Literatur

1. Härder, T.: DBMS Architecture - still an open problem. In: Tagungsband der GI-Fachtagung „Datenbanksysteme in Business, Technologie und Web (BTW)“. (2005) 2–28

2. Härder, T.: DBMS Architecture - new challenges ahead. Datenbank-Spektrum (14) (2005) 38–48 dpunkt-Verlag.
3. Härder, T., Rahm, E.: Datenbanksysteme: Konzepte und Techniken. 2nd edn. Springer-Verlag (2001)
4. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.E.: Basic concepts and taxonomy of dependable and secure computing. In: IEEE Transactions on Dependable and Secure Computing. Volume 1. (2004) 11–33
5. Jøsang, A., Keser, C., Dimitrakos, T.: Can we manage trust? In: Proceedings of the 3rd International Conference on Trust Management, iTrust. Volume 3477 of Lecture Notes in Computer Science. (2005) 93–107
6. Ellison, R.J., Fisher, D.A., Linger, R.C., Lipson, H.F., Longstaff, T., Mead, N.R.: Survivable network systems: An emerging displine. Technical Report CMU/SEI-97-TR-013, ESC-TR-97-013, Carnegie Mellon University, Software Engineering Institute (1997)
7. Knight, J.C., Sullivan, K.J.: On the definition of survivability. Technical Report CS-TR-33-00, University of Virginia, Department of Computer Science (2000)
8. Ellison, R.J., Moore, A.P., Bass, L., Klein, M., Bachmann, F.: Security and survivability reasoning frameworks and architectural design tactics. Technical Report CMU/SEI-2004-TN-022, Carnegie Mellon University, Software Engineering Institute (2004)
9. Schneier, B.: Attack trees: Modeling security threats. Dr. Dobbs Journal (1999)
10. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Proceeding of the 8th International Conference on Information Security and Cryptology. Lecture Notes in Computer Science (2005)
11. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Proceedings of the 1998 Workshop on New Security Paradigms (NSPW), ACM Press (1998) 71–79
12. Jürjens, J.: Towards development of secure systems using UMLsec. In: Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering (FASE). Volume 2029 of Lecture Notes in Computer Science. (2001) 187–200
13. Jürjens, J.: UMLsec: Extending UML for secure systems development. In: Proceedings of the 5th International Conference on The Unified Modelling Language (UML). Volume 2460 of Lecture Notes in Computer Science. (2002) 412–425
14. Jürjens, J.: Developing secure systems with UMLsec: From business processes to implementation. In: Tagungsband der GI-Fachtagung „Verlässliche IT-Systeme (VIS) - Sicherheit in komplexen IT-Infrastrukturen“, Vieweg (2001)
15. Jürjens, J., Shabalin, P.: Automated verification of UMLsec models for security requirements. In: Proceedings of the 7th International Conference on The Unified Modelling Language: Modelling Languages and Applications. Volume 3273 of Lecture Notes in Computer Science. (2004) 365–379
16. Haley, C.B., Laney, R.C., Moffett, J.D., Nuseibeh, B.: Picking battles: The impact of trust assumptions on the elaboration of security requirements. In: Proceedings of the 2nd International Conference on Trust Management, iTrust. Lecture Notes in Computer Science (2004) 347–354
17. Haley, C.B., Moffett, J.D., Laney, R., Nuseibeh, B.: Arguing security: Validating security requirements using structured argumentation. In: Proceedings of the 3rd Symposium on Requirements Engineering for Information Security (SREIS'05) held in conjunction with the 13th International Requirements Engineering Conference. (2005)

18. Camp, L.J.: Designing for trust. In: AAMAS 2002 International WorkshopTrust, Reputation, and Security: Theories and Practice,. Volume 2631 of Lecture Notes in Computer Science. (2002) 15–29
19. Bakker, A., Amade, E., Ballintijn, G., Kuz, I., Verkaik, P., van der Wijk, I., van Steen, M., Tanenbaum, A.S.: The globe distribution network. In: Proceedings of the Freenix Track: 2000 USENIX Annual Technical Conference. (2000) 141–152
20. Popescu, B.C., van Steen, M., Tanenbaum, A.S.: A security architecture for object-based distributed systems. In: Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC 2002). Volume 2995., IEEE Computer Society 2002 (2002) 161–171
21. Denning, D.E.: An intrusion-detection model. In: IEEE Transactions on Software Engineering (TSE). Volume 13. (1987) 222–232
22. Pal, P.P., Webber, F.: Intrusion tolerant systems. In: Proceedings of the IEEE Information Survivability Workshop (ISW-2000). (2000)
23. Strand, L.: Adaptive distributed firewall using intrusion detection. Master's thesis, University of Oslo, Department of Informatics (2004)
24. Eskin, E., Miller, M., Zhong, Z.D., Yi, G., Lee, W.A., Stolfo, S.: Adaptive model generation for intrusion detection systems. In: Proceedings of the 2000 ACM CCS Workshop on Intrusion Detection and Prevention (WIDS-2000), Athens, Greece (2000)