Vertrauensbasierte Laufzeitüberwachung verteilter komponentenstrukturierter E-Commerce-Software

Peter Herrmann[1], Lars Wiebusch[2] und Heiko Krumm[1]

[1] Universität Dortmund, FB Informatik, 44221 Dortmund
[2] E-Plus Mobilfunk GmbH & Co. KG, 40468 Düsseldorf
Peter.Herrmann@udo.edu, lars-wiebusch@web.de, Heiko.Krumm@udo.edu

Zusammenfassung Die Entwicklung komponentenstrukturierter E-Commerce-Software ist kostengünstig und schnell, da man die Systeme recht einfach aus wiederverwendbaren Softwarekomponenten zusammensetzt. Allerdings führt diese Entwurfsmethode zu einer neuen Art an Problemen für die Datensicherheit dieser Systeme. Insbesondere besteht die Gefahr, dass eine bösartige Komponente die gesamte Anwendung, in die sie eingebunden ist, bedroht. Zur Abwehr dieser Gefahr verwenden wir Security Wrapper, die das Verhalten von Komponenten zur Laufzeit überwachen und die Sicherheitsanforderungen der Anwendung durchsetzen. Ein Security Wrapper beobachtet das Verhalten an der Schnittstelle einer Komponenten und vergleicht es mit den vom Komponentenentwickler garantierten Sicherheitspolicies, die in der Komponentenspezifikation formal beschrieben werden. Wir stellen vor, wie man die Sicherheitspolicies zustandsbasiert beschreibt und führen eine Sammlung an Spezifikationsmustern ein, aus denen man die Modelle der Sicherheitspolicies für eine Komponente ableitet. Schließlich zeigen wir den Einsatz der Security Wrapper anhand eines E-Procurement-Beispiels.
Darüberhinaus erläutern wir, wie man unter Berücksichtigung der Erfahrungen anderer Nutzer mit einer Komponente den Aufwand der Laufzeittests reduzieren kann. Dazu verwenden wir einen speziellen Vertrauensmanagement-Service, der gute und schlechte Erfahrungen unterschiedlicher Benutzer mit Komponenten verwaltet. Abhängig von diesen Erfahrungsberichten können die Security Wrapper das Ausmaß der Überwachung absenken, indem sie anstatt einer vollständigen Überwachung nur Stichproben durchführen oder die Überwachung sogar abbrechen.

1 Einleitung

Komponentenbasierte Systeme erfreuen sich einer immer größeren Beliebtheit, da sie mit geringerem Kosten- und Entwicklungsaufwand als monolithische Anwendungen entwickelt werden können. Die Softwarekomponenten, die jeweils nur eine Teilfunktion eines E-Commerce-Systems realisieren, werden separat vertrieben. Der Betreiber einer Anwendung wählt geeignete Komponenten aus und koppelt sie miteinander zu einem auf seine speziellen Bedürfnisse zugeschnittenen System (vgl. [40]). Eine Komponente wird dabei entweder lokal in die Applikation eingebunden oder sie läuft auf einem fremden Wirtsrechner, von dem aus sie mittels eines Telekommunikationsdienstes mit der Anwendung verbunden wird.

Allerdings ist die Architektur komponentenstrukturierter Systeme, die unter Umständen aus heterogenen Bausteinen bestehen, komplexer als diejenige klassischer objekt-orientierter Applikationen, wodurch neuartige Probleme auftreten. Insbesondere in Hinsicht auf die Datensicherheit entstehen neue Herausforderungen, da man neben Systembetreibern und -nutzern auch die Komponentenentwickler, Wirtsrechnerbetreiber und gegebenenfalls die Anwendungsersteller als beteiligte Prinzipale berücksichtigen muss. Zum einen hat jeder Prinzipal eigene Sicherheitsziele, die eingehalten werden müssen. Zum anderen ist jeder Prinzipal aber auch eine potentielle Bedrohung für das System und damit für die anderen Beteiligten. Die größte Gefahr

dabei ist, dass eine bösartige Komponente das gesamte System, in das sie eingebunden ist, unsicher macht. Um den Einsatz sicherer komponentenbasierter Systeme zu ermöglichen, benötigt man somit ein Verfahren, das die Sicherheit einer Anwendung auch in dem Fall garantiert, dass ihre Komponenten von nicht-vertrauenswürdigen Quellen erworben wurden.

Unser Ansatz zur Lösung der Sicherheitsprobleme setzt auf die Komponentenkontrakte auf, die die Schnittstelleneigenschaften einer Komponente explizit und im Idealfall rechtsverbindlich beschreiben. Nach Beugnard et al. [4] besteht ein Kontrakt aus vier Teilen, in denen die Schnittstellenstruktur (d.h., die Methoden bzw. Ereignisse mit ihren Eingabe- und Ausgabeparametern sowie Ausnahmebehandlung), das erwartete Verhalten der Komponenten und ihrer Umgebung, Synchronisationsaspekte sowie quantitative Dienstgüteeigenschaften festgelegt sind. Wir ergänzen die Verhaltensbeschreibung eines Kontrakt um Sicherheitspolicies, die die Sicherheitsziele des Anwenders während der Komponentennutzung gewährleisten sollen. Die Policies werden durch Verhaltensconstraints formal dargestellt. Dabei gehen wir davon aus, dass man bösartige Komponenten oder von Dritten verfälschter Code (z.B. durch einen Computer-Virus) erkennen kann, wenn das reale Schnittstellenverhalten von den Kontraktbeschreibungen abweicht. Unser Ansatz besteht aus zwei Phasen:

- Zur Entwurfszeit wird untersucht, ob die Kombination der in den Modellen beschriebenen Verhaltensweisen die von der gesamten Anwendung geforderten Sicherheitsziele erfüllt.
- Zur Laufzeit wird für alle relevanten Komponenten überprüft, ob ihr tatsächliches Verhalten mit den Verhaltensconstraints übereinstimmt.

Ein Verhaltensconstraint wird im Kontrakt durch ein Zustandstransitionssystem modelliert und in der temporalen Logik cTLA [16] spezifiziert, die eine modulare Beschreibung der einzelnen Eigenschaften in separaten Spezifikationen erlaubt. Wenn man auch die Sicherheitsziele der Anwendung in cTLA beschreibt, kann man durch logische Deduktion formal verifizieren, dass die Ziele von den Komponenten erfüllt werden.

Dieses Papier konzentriert sich auf die Laufzeitüberprüfungen von Komponenten, die durch so genannte Security Wrapper [18] erfolgt. Ein Security Wrapper wird an die Schnittstelle zwischen der Komponente und ihrer Umgebung eingefügt. Er blockiert die Ereignisse an der Schnittstelle und überprüft sie auf Übereinstimmung mit den cTLA-Verhaltensbeschreibungen. Wenn ein Ereignis übereinstimmt, wird es weitergeleitet. Falls der Wrapper jedoch ein Fehlverhalten entdeckt, blockiert er alle weiteren Schnittstellenereignisse und benachrichtigt den Anwendungsadministrator.

Die cTLA-Verhaltensmodelle sollen aus Anwendersicht so formuliert sein, dass man durch die Laufzeitüberwachung rechtzeitig alle Verhaltensabweichungen erkennt, die eine Gefahr für die Anwendung und ihre Daten darstellen. Diese Forderung wird am besten gewährleistet, wenn die Modelle das gewünschte Schnittstellenverhalten sehr detailliert beschreiben. Dann besteht allerdings die Gefahr, dass die Spezifikationen ebenso komplex wie eine Referenzimplementierung werden, wodurch der Spezifikationsaufwand und die Leistungseinbußen bei der Überprüfung nicht mehr gerechtfertigt sind. Deshalb beschreiben die Modelle die Sicherheitspolicies wesentlich abstrakter und drücken nur die für die Rolle der Komponente im Gesamtsystem wesentlichen Sicherheitsaspekte aus. Sie sind darüberhinaus so ausgelegt, dass typische Schwachstellen des Gesamtsystems und seiner Architektur sehr direkt angesprochen werden. Somit kann man eine Sammlung kanonischer Muster für Sicherheitspolicies erstellen, die jeweils typische Schwachstellen komponentenbasierter Software beheben.

Security Wrapper können zu signifikantem Mehraufwand an Rechnerleistung führen. Deshalb verfolgen wir zusätzlich zu der Lauftzeitüberwachung durch die Wrapper einen komplementären Ansatz [15], der explizites Vertrauensmanagement ermöglicht (vgl. [21]) und sich an das Prinzip der Reputation Systems anlehnt (vgl. [36]). Ein so genannter Vertrauensinformationsdienst wird eingesetzt, der in regelmäßigen Abständen von Systembetreibern Berichte

über gute oder schlechte Erfahrungen mit den eingesetzten Komponenten erhält. Aus den Ergebnissen der Befragungen berechnet der Dienst so genannte Vertrauenswerte (vgl. [21]), die interessierten Personen zur Verfügung gestellt werden. Somit kann ein Interessent die aktuellen Vertrauenswerte bei der Entscheidung berücksichtigen, welche Komponenten er beschafft. Außerdem kann man auch die Art und Intensität der Laufzeitüberwachung von den aktuellen Vertrauenswerten abhängig machen, so dass der Mehraufwand für eine vollständige Überprüfung nur für Komponenten aufgebracht werden muss, die entweder bereits zu Fehlverhalten geführt haben oder die noch recht unbekannt sind.

Der Einsatz der Security Wrapper zur Laufzeitüberwachung wird anhand einer typischen E-Commerce-Anwendung veranschaulicht. Wir untersuchen ein komponentenorientiertes Warenbeschaffungssystem für Schnellrestaurants, das um Komponenten zur elektronischen Beschaffung (E-Procurement) ergänzt wurde.

Im nächsten Abschnitt wird eine Literaturübersicht über den Schutz von Systemen vor nicht vertrauenswürdigen Komponenten gegeben. Danach beschreiben wir kurz die Architekturen des Vertrauensinformationsdienstes und der Security Wrapper. Schließlich konzentrieren wir uns auf Anwendungsaspekte. Hier werden zunächst Konzepte und Muster von Komponentensicherheitspolicies in allgemeiner Form vorgestellt. Danach skizzieren wir die Beispielanwendung und zeigen ihre Schwachstellen auf. Schließlich erläutern wir die konkreten Sicherheitspolicies für das Beispiel und geben eine Strategie für eine vertrauensbasierte Laufzeitüberwachung an.

2 Literaturübersicht

Analog zu komponentenbasierter Software müssen auch bei mobilen Agentensystemen Systemteile gegen bösartiges Verhalten anderer Systemteile gesichert werden. Zum einen muss man die Wirtsrechner, auf denen fremde und deshalb nicht vertrauenswürdige Agenten ablaufen, vor bösartigen Verhalten dieser Agenten schützen. Zum anderen müssen auch die Agenten vor bösartigem Verhalten der Wirtsrechner und ihrer Betreiber bewahrt werden. Die Methoden zum gegenseitigen Schutz von Agenten und Wirtsrechnern haben im Wesentlichen das Ziel, sichere Kontrollflüsse, Speicherzugriffe und Methodenaufrufe zu garantieren (vgl. [26]). Neben Verfahren zur Isolierung sicherheitskritischer Agenten in einem speziellen geschützten Systemkern (vgl. z.B. [2]) und zur Nutzung kryptographischer Verfahren zum Schutz von Agenten während der Übertragung wurden insbesondere Ansätze zur Instrumentierung des Agentencodes untersucht. Hierbei wird der Code in einer Weise abgeändert, dass kritische Operationen entweder zur Entwicklungszeit oder zur Laufzeit auf Fehlverhalten analysiert werden, das zu möglichen Angriffen führen könnte. Zum Beispiel modelliert Schneider [38] die von kritischen Agenten einzuhaltenden Sicherheitspolicies mittels Automatenmodellen, die nach dem Prinzip der zustandsabhängigen Sicherheitsspezifikationen [5] aufgebaut sind. Der Agent läuft in einer speziellen Ablaufumgebung ab, die eine Operation nur ausführt, wenn sie dem zugehörigen Modell nicht widerspricht.

Auf Programmiersprachen basierende Sicherheit ist eine weitere Code-Instrumentierungsmethode, bei der aus dem Programmiercode des Agenten durch spezielle Übersetzer oder andere Analysemethoden sicherheitsrelevante Information gezogen wird. Mit Hilfe dieser Information kann man überprüfen, ob der Code die Sicherheitspolicies des Wirtsrechners erfüllt. Ein Beispiel hierzu ist der Java Byte Code Verifier, der Java-Programme auf Typkorrektheit und andere sicherheitsrelevante Eigenschaften hin untersucht. Formale Programmverifikation ist beim Einsatz des so genannten Proof Carrying Code [26] möglich. Hier fügt der Programmentwickler in den Code eine formale Spezifikation ein (z.B. Vor- und Nachbedingung von Funktionen, Schleifeninvarianten), mit deren Hilfe man die Korrektheit des Codes verifizieren kann. Beispiele für Proof Carrying Code sind der Touchstone-Übersetzer [34] und die effiziente Codezertifizierung [25]. Der Einsatz spezieller Programmergänzungen wurde weiterhin zur Typüberprüfung [31, 41] und zur Informationsflussanalyse [10, 32, 33] eingesetzt.

Ein Nachteil von Proof Carrying Code ist allerdings, dass die Programmergänzungen vom Codeentwickler vorgenommen werden, der sie zur Verbergung bösartigen Verhaltens fälschen kann. Man muss somit zusätzlich überprüfen, ob der Code die formale Spezifikation erfüllt. Hierzu eignen sich generische Software Wrapper, die den Code zur Laufzeit auf Einhaltung der Spezifikationen überwachen, ohne ihn zu verändern. Software Wrapper werden meist im Rahmen von Firewalls und Intrusion Detection Systemen eingesetzt (vgl. [1, 12, 30]). Ein weiterer Ansatz verwendet Software Wrapper, um zu verhindern, dass komponentenbasierte Software durch bösartige Systemaufrufe einzelner Komponenten gefährdet wird [11]. Auch unsere Security Wrapper sind eine spezielle Art Software Wrapper.

Komponentenkontrakte zur Überprüfung von Sicherheitsaspekten werden wie bei uns auch von Khan et al. eingesetzt (vgl. [24]). Allerdings konzentriert sich ihr Ansatz auf die Modellierung von Anforderungs-Garantiebedingungen zwischen einzelnen miteinander gekoppelten Komponenten. Ihr Modell hat eine recht einfache Struktur und beschreibt das Systemverhalten nicht. Es eignet sich deshalb nicht zur Modellierung detaillierter Maßnahmen zur Komponentenüberwachung.

Eine andere Methode zur Sicherung von Komponenten ist ihre Zertifizierung durch eine neutrale Instanz, die die Komponenten auf Sicherheitsaspekte, insbesondere auf Einhaltung der Kontrakte, überprüft und bei positivem Ausgang der Tests Zertifikate ausstellt. Voas [43] schlägt als Methoden für die Überprüfung der Komponenten eine Kombination aus Black Box-Tests (vgl. [29]), Fehlerinjektion (vgl. [44]) und operationale Laufzeittests (vgl. [45]) vor. Andere Methoden für Zertifizierungstests sind Codeinspektionen, wenn der Source-Code zur Verfügung steht, und Byte-Code-Verifikationen.

Unser Ansatz, einen Vertrauensinformationsdienst einzusetzen und damit die Erfahrungen anderer Komponentennutzer zu verwenden, ist eine weitere Methode, mit der man sowohl Laufzeitüberwachungsmaßnahmen als auch die Komponenten-Zertifizierung ergänzen kann. Er ist analog zu Label Bureaus, die zum Schutz von Kindern vor nicht jugendfreien Internet-Seiten verwendet werden (vgl. [39]). Dabei ordnet man einer fraglichen WWW-Seite ein Etikett (Label) zu, auf dem die Bewertungen des Seitenerstellers selbst, von interessierten Nutzern und von vertrauenswürdigen Dritten aufgelistet sind. Abhängig von diesen Bewertungen wird entschieden, ob die Seite von Kindern angesehen werden kann. Außerdem erweitert unser Ansatz das Konzept der Reputation Systems [36], in dem Teilnehmer an Internet-Transaktionen von ihren Geschäftspartnern bewertet werden. Die Bewertungsergebnisse stehen Personen, die an einer Transaktion mit der fraglichen Person interessiert sind, zur Entscheidungsfindung zur Verfügung. Ein bekanntes Beispiel ist das Feedback Forum des Internet-Auktionshauses eBay [9], in dem Käufer und Verkäufer ihre Partner mit Hilfe geschriebener Kommentare und einem Punktsystem bewerten können. Die Punkte eines Nutzers und die Kommentare können von allen eBay-Nutzern gelesen werden.

3 Vertrauensinformationsdienst

Der Vertrauensinformationsdienst sammelt gute und schlechte Bewertungen über Sicherheitsaspekte einer Komponente und stellt die Ergebnisse interessierten Nutzern zur Verfügung. Ein Nutzer kann die Bewertungen in seine Beschaffungsentscheidung einfließen lassen. Darüberhinaus kann er sie auch für die Laufzeitüberwachung heranziehen und die Intensität der Überwachung von ihnen abhängig machen.

Um die vertrauensbasierte Steuerung der Überwachung automatisieren zu können, beschreibt der Vertrauensinformationsdienst die Ergebnisse der Bewertungen in Form von Vertrauenswerten (vgl. [21]). Ein Vertrauenswert ist ein mathematischer Ausdruck, mit dem man zum einen das Vertrauen bzw. Misstrauen in eine Komponente beschreiben kann. Zum anderen kann man mit ihm auch ausdrücken, dass die Anzahl der Bewertungen noch zu gering ist, um Vertrauen

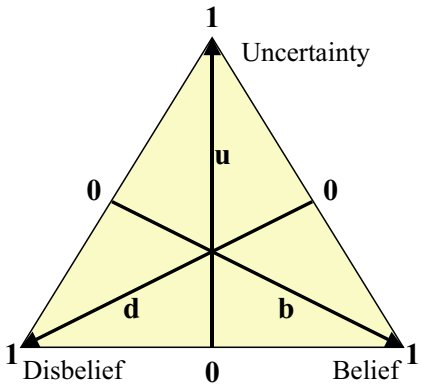

Abbildung 1. Opinion Triangle (entnommen aus [22])

zu erlangen. Jøsang hat in [22] ein so genanntes Opinion Triangle zur formalen Beschreibung von Vertrauenswerten definiert (siehe Abbildung 1). Reellwertige Variablen b (belief), d (disbelief) und u (uncertainty) aus dem Intervall zwischen 0 und 1 beschreiben das Vertrauen, das Misstrauen bzw. die Unsicherheit in eine Komponente. Da die Summe der drei Werte immer 1 betragen muss, kann man den aktuellen Vertrauenswert durch einen Punkt im Dreieck darstellen. Dabei zeigt ein hochliegender Punkt an, dass bisher nur wenige Bewertungen vorliegen. Bei einer steigenden Zahl an Ergebnissen liegt der Punkt immer niedriger. Wächst aufgrund vieler positiver Ereignisse das Vertrauen in die Komponente, liegt er eher auf der rechten Seite und bei wachsendem Misstrauen weiter links.

Vertrauenswerte werden aus der Anzahl positiver (Wert p) und negativer (Wert n) Bewertungen mit Hilfe von Metriken errechnet. Jøsang und Knapskog [23] empfehlen folgende Umrechnungsformel:

$$b = \frac{p}{p+n+1} \qquad d = \frac{n}{p+n+1} \qquad u = \frac{1}{p+n+1}$$

Hiernach kann eine negative Bewertung durch eine entsprechende Zahl positiver Wertungen ausgeglichen werden, so dass eine relativ tolerante Vertrauensmanagement-Philosophie vertreten wird. Dagegen schlagen Beth et al. [3] eine unversöhnliche Methode vor. Die Wahrscheinlichkeit, dass das Vertrauen b einen Wert α übersteigt, entspricht dabei der Formel

$$P(b > \alpha | p, n) = \begin{cases} 1 - \alpha^p & : \ n = 0 \\ 0 & : \ n > 0 \end{cases}$$

Daraus wird das Misstrauen und die Unkenntnis mit den Formeln

$$d = \begin{cases} 0 & : \ n = 0 \\ 1 & : \ n > 0 \end{cases} \qquad u = \begin{cases} 1 - b & : \ n = 0 \\ 0 & : \ n > 0 \end{cases}$$

berechnet. Nach dieser Metrik zerstört eine einzige negative Bewertung das Vertrauen in die Komponente für immer. Wenn keine negativen Wertungen aufgetreten sind, wächst das Vertrauen mit der Anzahl der gutartigen Bewertungen.

Um unterschiedliche Überwachungspolicies zu unterstützen, berechnet der Vertrauensinformationsdienst für jede Komponente zwei Vertrauenswerte nach den beiden oben vorgestellten Metriken. Die Struktur dieses Dienstes ist in Abbildung 2 aufgeführt. Er kommuniziert mit Komponentenentwicklern bzw. -vertreibern, mit interessierten Anwendungsbetreibern, die eine Komponente erwerben möchten oder sie zur Laufzeit überwachen, sowie mit Zertifizierungsinstanzen, die eine Komponente im Auftrag des Entwicklers oder eines Betreibers zertifizieren.

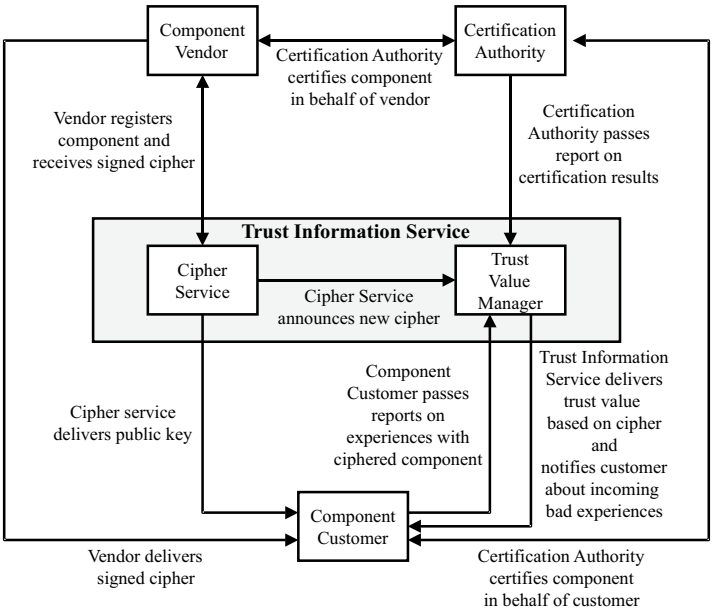

Abbildung 2. Vertrauensinformationsdienst

Komponentenentwickler können freie oder kommerzielle Komponenten bei dem Vertrauensinformationsdienst registrieren. Sie akzeptieren damit, dass Bewertungen von Nutzern in Form von Vertrauenswerten an Interessenten weitergegeben werden. Kaufinteressenten und Betreiber können jederzeit Vertrauenswerte einer Komponenten abrufen. Sie können darüberhinaus einen Alarm Service buchen, der sie sofort informiert, wenn eine ernste Warnung über die Komponente eingetroffen ist. Um eine hohe Zahl an Ergebnissen zu erhalten, werden alle Nutzer in zeitlichen Abständen um eine Bewertung gebeten. Darüberhinaus sollte ein Nutzer, wenn er eine schwerwiegende Verletzung der Sicherheit durch eine Komponente entdeckt, sofort eine Warnung an den Vertrauensinformationsdienst abgeben, der dann die anderen Nutzer benachrichtigt. Neben den Erfahrungen der Nutzer sind auch die Ergebnisse von Zertifizierungen von großer Bedeutung und fließen in die Vertrauenswertberechnung ein.

Um einen ausreichenden Datenschutz für die Komponentenentwickler zu garantieren, werden die Bewertungen getrennt von den Komponentenbeschreibungen gehalten. Der Verschlüsselungsdienst (Cipher Service) speichert die Daten einer registrierten Komponente und erzeugt einen eindeutigen Schüssel, den der Komponentenentwickler an Interessenten weitergeben kann. Der Vertrauenswertmanager (Trust Value Manager) speichert die Bewertungen und berechnet die Vertrauenswerte referenziert durch den Schlüssel, ohne die Komponenten im Klartext zu kennen. Somit hat weder der Verschlüsselungsdienst noch der Vertrauenswertmanager vollständige Kenntnis über die Bewertung der Komponenten im Klartext. Eine genauere Einführung in die Funktionen des Vertrauensinformationsdienstes wird in [15] gegeben.

4 Laufzeitüberwachung

Zur Überprüfung, ob das tatsächliche Systemverhalten einer Komponente ihrer Kontraktspezifikation entspricht, werden so genannte Security Wrapper eingesetzt [18]. Dazu koppelt man die Komponente nicht direkt mit ihren Partnerkomponenten, sondern leitet alle ein- und ausgehenden Schnittstellenereignisse über spezielle Komponenten. Dort werden die Ereignisse auf

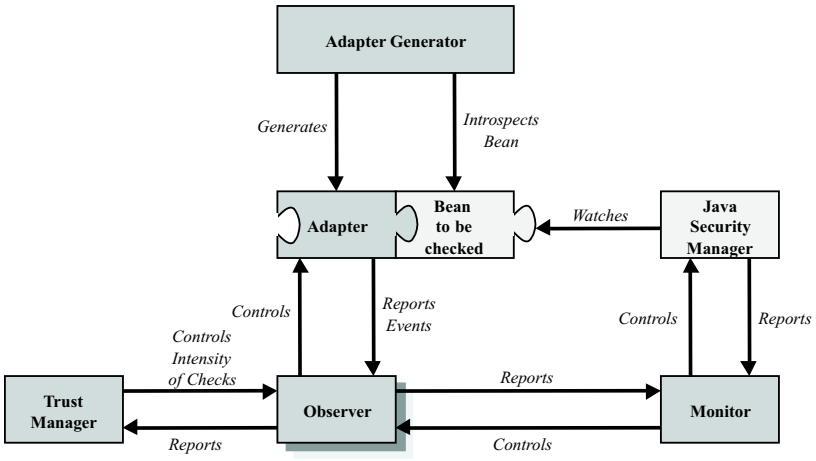

Abbildung 3. Security Wrapper-Architektur

Konformität mit den Sicherheitsspezifikationen im Schnittstellenkontrakt überprüft und nur bei Verhaltenskorrektheit weitergeleitet. Entspricht ein Schnittstellenereignis nicht der Spezifikation, wird der zuständige Applikationsadministrator benachrichtigt. Zudem kann ein Security Wrapper die fehlerhafte Komponente versiegeln, indem bis zu einer anderweitigen Entscheidung des Administrators keine Schnittstellenereignisse mehr weitergeleitet werden.

Ein Security Wrapper für Java Beans-basierte komponentenstrukturierte Systeme wurde in [28] realisiert (siehe Abbildung 3). Für jedes zu überwachende Bean wird ein spezieller Adapter erzeugt, über den alle Schnittstellenereignisse des Beans mit seiner Umgebung laufen. Die Konformitätsüberprüfungen werden durch die so genannten Observer-Objekte durchgeführt, wobei für jede cTLA-Verhaltensspezifikation ein Observer vorgesehen ist. Ein über den Adapter laufendes Ereignis wird zunächst blockiert und die beteiligten Observer informiert. Jeder Observer überprüft durch Simulation seiner Verhaltensspezifikation, ob das Ereignis zulässig ist. Wenn alle Observer das Ereignis akzeptieren, wird es vom Adapter freigegeben und an seinen Empfänger weitergeleitet. Falls jedoch ein Observer eine Verletzung der Spezifikation erkennt, wird das fehlerhafte Ereignis ebenso wie die nachfolgenden Ereignisse vom Adapter blockiert. Ein weiterer Bestandteil des Wrapper ist der Monitor, der die Nutzerschnittstelle des Administrators bildet. Wenn ein Observer eine Kontraktverletzung erkennt, zeigt er sie dem Administrator auf dem Monitor an. Außerdem wird ein so genannter Adapter Generator verwendet, der ein zu überwachendes Bean per Introspektion analysiert und einen geeigneten Adapter erzeugt. Schließlich verwendet das Werkzeug den im Java-Standard enthaltenen Java Security Manager, mit dem der Administrator den Zugriff des Beans auf Systemressourcen einschränken kann, um verdeckte Kanäle auszuschließen, über die das Bean den Adapter umgehen könnte.

Die Verbindung zwischen einem Security Wrapper und dem Vertrauensinformationsdienst wird durch ein Trust Manager-Objekt gebildet (vgl. [15]). Der Trust Manager erhält periodisch vom Vertrauensinformationsdienst Anfragen über die Erfahrung mit den überwachten Softwarekomponenten. Falls die Observer seit der letzten Anfrage keine Kontraktverletzungen feststellen konnten, wird eine positive Beurteilung zurückgesendet, während Fehlverhalten der Komponente zu einem negativen Urteil führt. Falls der Trust Manager eine Kontraktverletzung als schwerwiegend ansieht, kann er darüberhinaus eine Alarmmitteilung an den Vertrauensinformationsdienst senden, der dann die anderen eingetragenen Nutzer des Beans warnt. Erhält umgekehrt der Trust Manager eine Warnung über ein schwerwiegendes Fehlverhalten einer vom

Security Wrapper überwachten Komponente, initiiert er sofort deren Versiegelung, um eine Gefährdung der Applikation zu verhindern.

Schließlich wird der Trust Manager auch zur Überwachungssteuerung eingesetzt. Obwohl Laufzeittests gezeigt haben, dass der Überwachungsaufwand bei der existierenden Implementierung nur zu einem Overhead von 5 % führt (vgl. [18]), erscheint eine vertrauensabhängige Steuerung der Überwachungsintensität sinnvoll. Der Trust Manager kann deshalb abhängig vom aktuellen Vertrauenswert, der in zeitlichen Abständen vom Vertrauensinformationsdienst geladen wird, unterschiedliche Überwachungspolicies durchführen. Bei geringer Anzahl an Bewertungen einer Komponenten oder bei einem schlechten Vertrauenswert wird die Komponente vollständig überwacht. Übersteigt der Vertrauenswert eine bestimmte Grenze, überprüft man die Komponente nur noch stichprobenhaft. Hierbei erfolgt die Konformitätsüberprüfung nur noch zeitweise[3] und die Länge der Überwachungspausen hängt ebenfalls vom Vertrauenswert ab. Schließlich kann man die Überwachung vollständig einstellen und den entsprechenden Observer löschen, wenn die Komponente als sicher angesehen wird.

5 Muster für Sicherheitspolicies

Eine bösartige Komponente kann die sie einbindende Anwendung auf vielfältige Art angreifen und ihre Vertraulichkeit, Integrität, Verfügbarkeit sowie Nichtabstreitbarkeit verletzen. Eine Gefahr für die Vertraulichkeit liegt in der Änderung von Datenflüssen im System, so dass fehlgeleitete Dateneinheiten von nicht dazu berechtigten Personen gelesen werden. Diese Angriffsart ist insbesondere für verteilte komponentenstrukturierte Systeme relevant, in denen die Komponenten auf verschiedene Rechner mit unterschiedlichen Zugriffspolicies verteilt sind. Eine weitere Angriffsart auf die Vertraulichkeit eines komponentenstrukturierten Systems sind verborgene Kanäle. Dabei wird Information entweder in zwischen Komponenten transportierten Daten versteckt (Steganographie) oder man nutzt die Reihenfolge, Anzahl oder den Ausführungszeitpunkt von Schnittstellenereignissen dazu, geheime Zusatzinformation darzustellen. Bei Angriffen auf die Systemintegrität werden die Funktionalität der Anwendung oder die gespeicherten Daten abgeändert. Hierbei kann man fehlerhafte Schnittstellenereignisse auslösen, Attribute mit falschen Werten belegen oder die Systemkonfigurationsparameter abändern. Die Verfügbarkeit von komponentenbasierter Software kann man durch die folgenden Angriffsarten gefährden:

- Ein von einer Komponente angebotener Dienst wird so häufig aufgerufen, dass dritte Komponenten nicht mehr bedient werden können (Denial-of-Service Angriffe).
- Eine Komponente blockiert ihren Partner dadurch, dass bestimmte geforderte Schnittstellenereignisse nicht ausgelöst werden. Der Partner kann während der Wartezeit nicht mehr dritte Komponenten bedienen.

Ein typischer Angriff auf die Nichtabsteitbarkeit von Komponenten ist dann erfolgt, wenn der Komponentenentwickler oder -betreiber in der Lage ist, später erfolgreich abzustreiten, dass die Komponente bestimmte Schnittstellenereignisse ausgelöst oder empfangen hat.

Um Anwendungen vor derartigen Angriffen zu schützen, muss der Betreiber geeignete Sicherheitsziele definieren und Sicherheitspolicies anwenden, die diese Ziele durchsetzen. Mit den Sicherheitspolicies wird das Verhalten einer Komponente in einer Weise eingeschränkt, dass Angriffe entweder unmöglich oder nur unter erheblichen Aufwand durchführbar sind. Um Angriffe auf die Vertraulichkeit zu unterbinden, muss der Datenfluss zwischen Komponenten so eingeschränkt werden, dass Daten nur zu Komponenten gelangen, bei denen ein Zugriff von nichtauthorisierten Lesern ausgeschlossen ist. Versteckte Kanäle kann man dadurch erschweren, dass nur deterministisches Schnittstellenverhalten zugelassen wird (vgl. [46]). Zur Verhinderung von

[3] Allerdings wird der Zustand der simulierten cTLA-Spezifikation immer aktualisiert, um später mit der Überwachung wieder aufsetzen zu können.

Steganographie müssen zum Beispiel Ausgabedaten in einer eindeutigen funktionalen Abhängigkeit von vorhergehenden Eingaben stehen. Um Angriffe auf die Integrität von Komponenten zu erschweren, werden Schnittstellenereignisse und deren Argumente eingeschränkt. Außerdem kann man zur Laufzeit überprüfen, ob die Ereignisse plausibel sind. Durch Erzwingung minimaler Wartezeiten zwischen bestimmten Schnittstellenereignissen kann man Denial-of-Service Angriffe verhindern, da durch die Wartezeiten genügend Zeit zur Abarbeitung von Aufträgen dritter besteht. Umgekehrt kann man Blockadeangriffe verhindern, indem man verlangt, dass ein erwartetes Schnittstellenereignis innerhalb einer maximalen Reaktionszeit ausgeführt wird. Relativ schwer zu realisieren sind Policies zur Verhinderung, dass Ereignisse später abgestritten werden. Ein Schritt hierzu ist die Verwendung einer neutralen Instanz, die Information über Schnittstellenereignisse sammelt und sie sich von den beteiligten Komponenten digital signieren lässt.

Die Security Wrapper (siehe Abschnitt 4) können nur das Verhalten an der Schnittstelle zwischen Komponenten überwachen jedoch weder interne Komponentenereignisse noch interne Attributbelegungen kontrollieren. Ein Mittel zur Einschränkung des Verhaltens von Schnittstellenereignissen ist die Festlegung, welche Ereignisse ausgeführt werden dürfen, wobei man auch die Historie vorangegangener Ereignisse berücksichtigen kann. Ein weiteres Mittel liegt in der Einschränkung des Ausführungszeitpunktes eines Ereignis. Diese Arten der Einschränkung kann man durch die vier folgenden Basis-Spezifikationsmuster formal beschreiben:

Enabling Condition: Ein Schnittstellenereignis darf mit einer bestimmten Argumentbelegung nur unter festgelegten Bedingungen geschaltet werden.

Enabling History: Ein Schnittstellenereignis darf mit einer bestimmten Argumentbelegung nur geschaltet werden, wenn zuvor eine festgelegte Folge an Ereignissen aufgetreten ist.

Minimum Waiting Time: Ein Schnittstellenereignis darf nur geschaltet werden, wenn seit einem vorhergehenden Ereignis eine minimale Wartezeit verstrichen ist.

Maximum Waiting Time: Ein Schnittstellenereignis muss geschaltet worden sein, bevor seit einem vorhergehenden Ereignis eine maximale Reaktionszeit verstrichen ist.

Diese Basismuster bilden die Grundlage für die im folgenden aufgeführten Muster, mit denen man spezielle Sicherheitspolicies für die Vertraulichkeit, Integrität, Verfügbarkeit und Nichtabstreitbarkeit von Komponenten modelliert. Die Spezifikationen wurden aus den Basismustern abgeleitet. Sicherheitspolicies für die Vertraulichkeit von komponentenbasierter Systeme kann man mit Hilfe der folgenden fünf Muster beschreiben:

Data Flow Access: Eine Dateneinheit darf nur an eine Komponente übergeben werden, wenn diese die Leserechte der Dateneinheit erfüllt.

Data Flow History: Eine Dateneinheit darf nur an eine Komponente übergeben werden, wenn zuvor eine bestimmte Folge an Schnittstellenereignissen aufgetreten ist.

Hidden Channel Functional Dependency: Eine weitergeleitete Dateneinheit steht in einer funktionalen Beziehung zu zuvor gesendeten oder empfangenen Daten.

Hidden Channel Enabling History: Die Schaltbedingung eines Schnittstellenereignisses mit einer bestimmten Argumentbelegung steht in einer funktionalen Abhängigkeit zu vorhergehenden Schnittstellenereignissen.

Hidden Channel Execution Time: Ein Schnittstellenereignis muss nach einem vorangegangenen Ereignis innerhalb eines festgelegten Zeitintervalls geschaltet werden.

Muster von Sicherheitspolicies zur Garantie von Integritäts-Sicherheitszielen werden im folgenden aufgeführt:

Integrity Enabling Condition: Ein Schnittstellenereignis mit einer bestimmten Argumentbelegung darf nur unter festgelegten Bedingungen geschaltet werden, so dass die Integrität der Komponente, die das Ereignis empfängt, nicht verletzt wird.

Integrity Enabling History: Ein Schnittstellenereignis mit einer bestimmten Argumentbelegung darf nur geschaltet werden, wenn zuvor eine festgelegte Folge an Schnittstellenereignissen aufgetreten ist. Dadurch garantiert man, dass die Integrität der Komponente, die das Ereignis empfängt, nicht verletzt wird.

Sicherheitspolicies, mit denen man die beiden Angriffsarten auf die Verfügbarkeit von Komponenten ausschließt, werden durch die folgenden Muster beschrieben:

Denial-of-Service Minimum Waiting Time: Ein Schnittstellenereignis darf nur ausgeführt werden, wenn nach einem vorhergehenden Ereignis eine minimale Wartezeit verstrichen ist.

Denial-of-Service Enabling History: Ein Schnittstellenereignis mit einer bestimmten Argumentbelegung darf nur geschaltet werden, wenn zuvor eine festgelegte Folge an Schnittstellenereignissen aufgetreten ist. Zusätzlich muss eine minimale Wartezeit seit dem letzten Ereignis der Folge eingehalten werden.

Blocking Maximum Waiting Time: Ein Schnittstellenereignis muss geschaltet werden, bevor nach einem vorangegangenen Ereignis eine maximale Reaktionszeit verstrichen ist.

Blocking Enabling History: Abhängig von der Folge vorangegangener Ereignisse muss ein Schnittstellenereignis innerhalb einer maximalen Reaktionszeit geschaltet werden.

Die Beschreibung der Nichtabstreitbarkeit von Schnittstellenereignissen unterstützen wir mit dem folgenden Muster:

Event Logging: Ein aus- oder eingehendes Schnittstellenereignis muss zusammen mit einer digitalen Signatur bei einer neutralen Instanz hinterlegt werden.

Die Sicherheitspolicies beschreiben Aspekte des Schnittstellenverhaltens und können somit als Zustandstransitionssysteme modelliert werden, die durch die Security Wrapper implementiert werden (vgl. [18]). Ein Schnittstellenereignis einer Komponente führt in den Simulationen der Muster zu jeweils einem Zustandsübergang. Dadurch berücksichtigen die Wrapper die Ereignishistorie und können überprüfen, ob Schnittstellenereignisse zu der Historie konform sind.

Die Muster der Sicherheitspolicies sind in der formalen Beschreibungstechnik cTLA modelliert (vgl. [16, 17]). cTLA ermöglicht separate Spezifikationen von Sicherheits-, Lebendigkeits- und Echtzeiteigenschaften in einem prozessorientierten Stil, der an höhere Programmiersprachen angelehnt ist. Darüberhinaus erlaubt cTLA auch die Erstellung constraintorienter Spezifikationen (vgl. [42]). Damit kann man die verschiedenen Sicherheitspolicies durch unterschiedliche Muster-Instanzen modellieren, die jeweils von einem eigenen Observer simuliert werden. Allerdings kann man auch verschiedene Spezifikationen miteinander zu einer umfassenderen Sicherheitsspezifikation komponieren. Damit kann man durch einen formalen Beweis zeigen, dass die in der kompositionalen Spezifikation zusammengefassten Sicherheitspolicies eine Sicherheitseigenschaft des Gesamtsystems erfüllen.

Der Entwickler einer Komponente kann in den Komponentenkontrakt entweder cTLA-Spezifikationen oder Beschreibungen in der bekannten Softwaremodellierungssprache UML [6], die man nach cTLA überführen kann (vgl. [13]), einfügen. Vor dem Einbinden der Komponente in die Anwendung überprüft der Komponentenbetreiber, ob die spezifizierten Policies für seine Sicherheitsziele ausreichen. Die Einhaltung der Spezifikationen durch die Komponente wird dann zur Laufzeit durch die Security Wrapper überprüft. Die Umsetzung der cTLA-Spezifikationen in ausführbaren Java-Code erfolgt im Moment von Hand, da ein cTLA-Java-Übersetzer zur Zeit noch nicht lauffähig ist. Die cTLA-Spezifikationen der Basismuster, der speziellen Muster sowie der Musterinstanzen, die im folgenden E-Commerce-Beispiel verwendet werden, sind im WWW verfügbar (siehe URL: ls4-www.informatik.uni-dortmund.de/RVS/P-SACS/eReq).

6 Komponentenbasierte E-Procurement-Anwendung

Das OBI-Konsortium hat zur Standardisierung der Prozeduren bei elektronischen Beschaffungsmaßnahmen (E-Procurement) die Norm OBI (Open Buying on the Internet) [35] herausgegeben. In diesem Standard wird eine Architektur für Beschaffungsaktivitäten definiert, die aus einem Käufer, einer Anzahl an Lieferanten, einem Bank- oder Kreditinstitut und einem Beschaffer (Requisitioner) besteht. Der Beschaffer führt im Auftrag des Käufers Beschaffungen von Gütern bei einem Lieferanten durch. Die Bank rechnet die Bestellungen ab. Das von OBI definierte Geschäftsprozessmodell sieht dabei die folgenden Schritte vor:

1. Der Beschaffer fordert beim Käufer Verbindungsadressen für Zugangsrechner von Lieferanten an.
2. Der Beschaffer holt für die zu erwerbenden Güter Angebote der Lieferanten ein.
3. Die Lieferanten erzeugen die Angebote und codieren sie nach dem EDI-Standard (vgl. [8]) in dem von OBI festgelegten Order Request-Format. Die Angebote werden dann entweder direkt an den Käufer gesendet zum Beschaffer übertragen, der sie dann an den Käufer weiter leitet.
4. Auf der Basis der Angebote wählt der Beschaffer — gegebenenfalls in Zusammenarbeit mit anderen Einheiten der Käuferorganisation — einen Lieferanten aus und erzeugt eine Bestellung.
5. Die Bestellung wird ebenfalls nach dem EDI-Standard im so genannten Order Format kodiert und dem ausgewählten Lieferanten übermittelt.
6. Der Lieferant führt die Bestellung aus.
7. Die Bank stellt im Auftrag des Lieferanten eine Rechnung an den Käufer aus und empfängt die Bezahlung.

Unser Anwendungsbeispiel simuliert den Einkauf und die Lagerhaltung von Schnellrestaurants. Es wurde als komponentenstrukturiertes System auf der Basis des SalesPoint-Framework entwickelt (vgl. [37]). SalesPoint ist Freeware und ermöglicht durch fertige Module für Geschäftsfunktionen wie Einkauf, Verkauf oder Leasing von Gütern sowie Module für Verwaltungsfunktionen wie Abrechnung, Lagerverwaltung oder Erzeugung des Produktkatalogs die recht einfache Erzeugung unterschiedlicher Warenwirtschaftssysteme. Das Framework ist in Java programmiert, sieht in seiner ursprünglichen Form jedoch keine Softwarekomponenten vor. Wir haben es deshalb angepasst und in drei Komponenten zerlegt, die die Verkaufsfunktionen eines Restaurants, die Lagerverwaltung sowie die Verwaltung des Produktkatalogs realisieren.

Abbildung 4 beschreibt die Komponenten unseres Beispiels. Um die elektronische Beschaffung der Speisen und Getränke zu automatisieren, haben wir neben den von SalesPoint abgeleiteten Komponenten *Restaurant*, *Counting Stock* und *Catalog* drei weitere Komponenten implementiert. Die Beschaffungen werden durch die Komponente *OBI-E-Requisitioner* vorgenommen, die den im OBI-Standard vorgesehenen Beschaffer ersetzt. Außerdem wurde eine Komponente *Directory of Sellers* eingefügt, die die Adressen der Lieferanten und ihre Produktpalette speichert. Schließlich verwenden wir eine Komponente *OBI-Buying Adapter*, die die Kodierung und Dekodierung von Angebotsanfragen, Angeboten und Bestellungen übernimmt und als Schnittstelle zu den Systemen der Lieferanten dient. Die sechs Komponenten wurden miteinander zum Warenwirtschaftsystem des Schnellrestaurants gekoppelt. Die Lieferantensysteme wurden ebenfalls mit Hilfe der Module des SalesPoint-Framework entwickelt. Schließlich haben wir die Komponente *Logging Service* als neutrale Instanz implementiert, um das spätere Abstreiten des Austausch von Angebotsanfragen, Angeboten oder Bestellungen zu erschweren. Da wir uns im Weiteren nur für die Phasen bis zur Abgabe der Bestellung interessieren, haben wir, um das System einfach zu halten, auf die Komponenten zur Realisierung der Bank verzichtet. Das Beispielsystem kann von der WWW-Projektseite geladen werden (siehe URL: ls4-www.informatik.uni-dortmund.de/RVS/P-SACS/eReq).

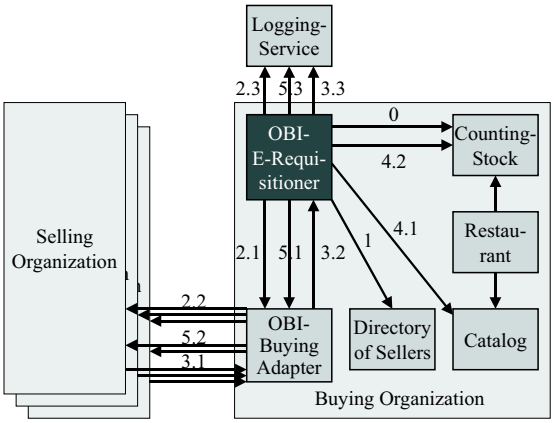

Abbildung 4. Beispiel eines E-Procurement-Systems für Schnellrestaurants

Für unser Ziel, die Warenbeschaffung zu automatisieren, mussten wir allerdings den OBI-Standard, der keine formale Definition für Anfragen nach Angeboten vorsieht, ergänzen. Dazu haben wir EDI durch cXML (Commercial eXtensible Markup Language) [7] als Transfersyntax ersetzt, die im Gegensatz zu EDI ein Format für Angebotsanfragen vorsieht. cXML basiert auf der populären Datencodiersprache XML.

Die einzelnen Schritte einer Beschaffung werden durch die Kanten in Abbildung 4 dargestellt. Da der E-Requisitioner nicht nur für den Beschaffungsablauf zuständig ist, sondern auch entscheidet, wann die Beschaffung gestartet werden soll, haben wir einen neuen ersten Schritt 0 eingeführt, in dem der E-Requisitioner das Lager in zeitlichen Abständen inspiziert und abhängig von der Menge an vorhandenen Waren festlegt, ob und für welche Güter eine Beschaffung vorgenommen wird. Wenn neue Güter bestellt werden müssen, liest der E-Requisitioner vom Verkäuferverzeichnis die Adressen geeigneter Lieferanten (Schritt 1). Danach erzeugt er eine Angebotsanfrage und übermittelt sie über den OBI-Buying Adapter an die Lieferanten (Schritt 2). Die Lieferanten übermitteln die Angebote an den Adapter, der sie an den E-Requisitioner weiter leitet (Schritt 3). Anschließend konsultiert der E-Requisitioner den Produktkatalog und die Lagerhaltung und wählt abhängig von den Angeboten, den Verkaufspreisen und dem Warenbestand einen Lieferanten aus (Schritt 4). Schließlich wird die Bestellung an den Adapter gesendet, der sie an den entsprechenden Lieferanten weiter leitet (Schritt 5). Die Schritte 6 und 7 wurden nicht realisiert. Um die Angebotsanfragen, Angebote und Bestellungen später nicht abstreiten zu können, sendet der E-Requisitioner in den Schritten 2, 3 und 5 die gesendeten bzw. empfangenen Daten zusammen mit seiner digitalen Signatur an die neutrale Instanz.

7 Verhaltensüberwachung im Beispielsystem

Um einen korrekten und sicheren Ablauf der Beschaffungen zu garantieren, muss sich der E-Requisitioner immer gutmütig verhalten. Er darf zum Beispiel nicht Angebote an konkurrierende Lieferanten weitergeben, um sie bei der Auftragsvergabe zu bevorzugen. Auch muss er immer ein günstigstes Angebot für seine Bestellung auswählen und darf die Lagerhaltung des Käufers nicht durch zu große, zu kleine oder zu späte Bestellungen schädigen. Im folgenden nehmen wir an, dass der E-Requisitioner von einer nicht vollständig vertrauenswürdigen Quelle erworben wurde. Zur Sicherung des gesamten Systems müssen wir ihn deshalb durch einen Security Wrapper überwachen lassen. Wir haben dazu 13 Sicherheitspolicies formuliert, die für den korrekten Systemablauf wichtig sind. Anschließend haben wir mit Hilfe der cTLA-Verfeinerungsoperation

(vgl. [14]) aus den in Abschnitt 5 vorgestellten Mustern cTLA-Spezifikationen dieser Policies erzeugt. Im nächsten Schritt wurden die Spezifikationen in Java-Code überführt, der dann in die Observer des Security Wrappers eingebettet wurde.

Im folgenden listen wir die Sicherheitspolicies kurz auf[4]. Zum Schutz der Vertraulichkeit der Anwendung werden die folgenden Policies gefordert:

1. Eine Angebotsanforderung an einen Lieferanten darf nur Güter enthalten, die dieser Lieferant in seinem Warenbestand hat (Data Flow Access). Damit erhalten Lieferanten keine Information über Güter, die nicht für sie relevant sind.
2. Eine Angebotsanforderung darf nur an Lieferanten gesendet werden, die im Lieferantenverzeichnis aufgeführt sind (Data Flow Access). Damit werden Informationen über die vom Restaurant vertriebenen Speisen und Getränke nur an Lieferanten weitergegeben, die von der Käuferorganisation akzeptiert sind.
3. Die Menge eines angefragten bzw. bestellten Artikels hängt eindeutig mit dem Lagerbestand des Artikels zusammen (Hidden Channel Functional Dependency). Damit soll ausgeschlossen werden, dass der E-Requisitioner durch Angabe bestimmter Bestellmengen geheime Zusatzinformation an einen Lieferanten weiter gibt.
4. Eine neue Beschaffungsrunde darf nur gestartet werden, wenn die vorhergehende mit der Warenlieferung abgeschlossen ist (Hidden Channel Enabling History). Damit soll ausgeschlossen werden, dass eine Bestellung in mehrere Unteraufträge aufgeteilt wird. Die Art der Aufteilung könnte eine geheime Zusatzinformation enthalten.

Zur Integritätssicherung verwenden wir die folgenden Sicherheitspolicies:

1. Eine Bestellung darf nur vorgenommen werden, wenn zuvor eine bestimmte Anzahl an Angeboten eingegangen ist (Integrity Enabling History). Dadurch wird verhindert, dass der E-Requisitioner durch eine schnelle Bestellung bei einem bevorzugten Lieferanten später eintreffende Angebote nicht berücksichtigt.
2. Der E-Requisitioner wählt immer eines der günstigsten Angebote aus (Integrity Enabling History).
3. Die Daten der Lagerverwaltung, des Produktkatalogs, des OBI-Buying Adapter und der neutralen Instanz werden nicht geändert (Integrity Enabling Condition „false“ für alle Methoden, die die Daten ändern). Damit wird die Datenintegrität der mit dem E-Requisitioner gekoppelten Komponenten geschützt.
4. Bei einer Bestellung liegt die Bestellmenge immer in einem festen Bestellintervall (Integrity Enabling Condition). Dadurch werden zu große oder zu kleine Bestellmengen vermieden.

Zur Verhinderung von Angriffen auf die Verfügbarkeit des Systems soll die E-Requisitioner-Komponente die folgenden Sicherheitspolicies erbringen:

1. Zwischen zwei an die Lagerverwaltung, den Produktkatalog, das Lieferantenverzeichnis oder den Buying Adapter gesendete Schnittstellenereignisse liegt immer eine minimale Wartezeit (Denial-of-Service Minimum Waiting Time). Dadurch wird verhindert, dass die Komponenten ständig belastet werden und damit nicht mehr andere Komponenten (z.B. die Verkaufskomponente) bedienen können.
2. Der Warenbestand in der Lagerverwaltung wird innerhalb eines maximalen Zeitintervalls kontrolliert (Blocking Maximum Waiting Time). Dadurch ist sichergestellt, dass der E-Requisitioner einen niedrigen Lagerbestand einer Ware rechtzeitig erkennt.
3. Wenn der E-Requisitioner feststellt, dass eine Ware knapp ist, startet er den Prozess zur Ersatzbeschaffung innerhalb einer maximalen Zeit (Blocking Enabling History). Dadurch verhindert man, dass der E-Requisitioner eine Bestellung so lange aufschiebt, bis die Ware nicht mehr verfügbar ist.

[4] Dabei gibt der in Klammern gesetzte Bezeichner den Namen des Musters an, aus dem die Spezifikation abgeleitet wurde.

4. Nach Empfang der in der ersten Policy zur Integritätssicherung angegeben Mindestanzahl an Angeboten, die für eine Bestellung notwendig ist, führt der E-Requisitioner die Bestellung innerhalb einer maximalen Reaktionszeit aus (Blocking Enabling History). Auch hiermit wird sichergestellt, dass Bestellungen nicht zu lange verzögert werden.

Schließlich muss man garantieren, dass der Käufer später gegenüber den Lieferanten aber auch dem Entwickler der E-Requisitioner-Komponente nachweisen kann, dass Angebotsanfragen und Bestellungen tatsächlich getätigt und dass Angebote tatsächlich empfangen wurden. Dazu verwenden wir die folgende Sicherheitspolicy:

1. Angebotsanfragen, Angebote und Bestellungen müssen der neutralen Instanz innerhalb einer maximalen Zeit unter Hinzufügung einer eindeutigen digitalen Signatur gemeldet werden (Event Logging).

Der Security Wrapper überprüft, ob die angegebenen 13 Sicherheitspolicies von der Komponente eingehalten werden. Messungen haben ergeben, dass in unserer Beispielimplementierung die Mehrbelastung für diese Laufzeittests etwa bei 5 % liegt. Um bei wachsendem Vertrauen in die Komponente den Überprüfungsaufwand durch Stichproben oder die Aufgabe der Überprüfung einzelner Policies senken zu können, setzen wir neben dem eigentlichen Security Wrapper auch den Vertrauensinformationsdienst sowie einen Trust Manager ein. Zur Konfigurierung des Trust Managers müssen wir dazu für jede einzelne Policy festlegen, welche Überprüfungsmaßnahmen bei welchen Vertrauenswerten durchgeführt werden. Dazu betrachten wir zunächst, wie sicherheitskritisch die Einhaltung der jeweiligen Policy für das Gesamtsystem ist.

Die Sicherheitspolicies garantieren zum einen, dass Information über die vom Restaurant vertriebenen Waren nur an bestimmte Lieferanten weitergegeben wird. Im Bereich der Schnellrestaurants scheint der Geheimhaltungsbedarf aufgrund der allgemeinen Bekanntheit der Produktpalette jedoch nicht groß zu sein. Allerdings muss man berücksichtigen, dass ein Restaurant geschädigt werden kann, wenn die Einführung eines neuen Artikels der Konkurrenz frühzeitig bekannt wird. Aufgrund dieser gegenteiligen Effekte verwenden wir einen mittelstarken Schutz. Die ersten beiden Vetraulichkeits-Sicherheitspolicies werden nur vollständig überprüft, wenn nach der Metrik von Jøsang und Knapskog [23] die Variable b des aktuellen Vertrauenswertes unter dem Wert $0,99$ liegt. Stichpunkthafte Überprüfungen führen wir durch, wenn b einen Wert zwischen $0,99$ und $0,999$ hat, während die Überwachung bei noch höheren Werten von b eingestellt wird. Dass heißt, das wir vollständige Überprüfungen durchführen, wenn die Anzahl positiver Ereignisse die Anzahl negativer Ereignisse um weniger als den Faktor Hundert übersteigt. Bei einem Faktor von mehr als Tausend wird die Überwachung unwiderruflich abgebrochen.

Die Forderung, dass bei den Bestellvorgängen Angebote nicht an konkurrierende Lieferanten gelangen können, ist dagegen von sehr großer Bedeutung. Da die OBI-Buying Adapter-Komponente an Lieferanten nur die Übertragung von cXML-Angebotsanforderungen und Bestellungen erlaubt, stehen für die illegale Übermittlung der Angebote nur versteckte Kanäle zur Verfügung, die man durch die zwei zuletzt genannten Vertraulichkeitspolicies verhindern will. Die Einhaltung dieser Policies ist deshalb so wichtig, dass wir sie mit dem Wrapper permanent vollständig überprüfen.

Von großer Bedeutung sind auch alle vier Sicherheitspolicies zur Garantie der Integrität. Bei Verletzung der ersten beiden Policies kann der Käufer durch zu teure Bestellungen geschädigt werden. Sehr wesentlich ist auch der durch die dritte Policy erzwungene Schutz der Datenintegrität, da deren Verletzung andere Angriffe nach sich ziehen kann[5]. Schließlich ist auch die vierte Sicherheitspolicy für die Systemintegrität von Bedeutung, da zu hohe Bestellmengen der meist leicht verderblichen Güter zu hohen finanziellen Schäden führen können. Wir überprüfen aus diesem Grund alle vier Policies ständig.

[5] Zum Beispiel könnte ein preislich günstiger Lieferant aus dem Lieferantenverzeichnis gelöscht werden.

Die Anforderungen an die Systemverfügbarkeit sind gegenüber denjenigen an die Integrität von etwas geringerer Bedeutung. Zwar sind Denial-of-Service Angriffe unangenehm, da sie den Verkauf schädigen können. Allerdings wird das Verkaufspersonal das Problem schnell erkennen, so dass die Fehlerquelle frühzeitig gefunden und behoben werden kann. Da uns der Gefährdungsgrad mit dem des Bekanntwerdens der Produktpalette vergleichbar erscheint, verwenden wir hier ebenfalls die Werte $b \geq 0,99$ für Stichproben und $b \geq 0,999$ für den Abbruch der Überprüfung.

Leere Lager aufgrund zu später Bestellungen sind im Vergleich zu Denial-of-Service-Angriffen kritischer, da sie zu finanziellen Einbußen führen. Allerdings sind wir hier der Meinung, dass die Restaurantleitung auch unabhängig vom E-Requisitioner auf den Lagerbestand achten sollte, so dass ein Warenengpass meist noch rechtzeitig erkannt und behoben werden kann. Wir verwenden hier etwas rigidere Überwachungsmethoden als für die Sicherheitspolicy gegen Denial-of-Service-Angriffe. Zum einen setzen wir die Metrik von Beth et al. ein [3], um die Policies schon nach einer beim Vertrauensinformationsdienst eingegangenen negativen Bewertung immer vollständig zu überwachen. Außerdem fangen wir erst bei einem Wert von $b \geq 0,999$ mit Stichproben an. Die Überprüfung wird erst bei einem Wert von $b \geq 0,99999$ vollständig aufgegeben.

Die Meldung von Interaktionen mit den Lieferanten an die neutrale Instanz ist von hoher Bedeutung, da sie rechtliche Relevanz erlangen kann. Wir überprüfen die Einhaltung der entsprechenden Sicherheitspolicy deshalb ständig.

8 Schlußbemerkungen

In diesem Papier zeigten wir die Nutzung formaler Methoden zur Beschreibung von Sicherheitspolicies in Komponentenkontrakten. Darüberhinaus stellten wir die Security Wrapper vor, mit denen man durchsetzen kann, dass die Policies von den Komponenten zur Laufzeit eingehalten werden. Schließlich wurde eine kurze Einführung in das Vertrauensmanagement von Komponenten gegeben. In unserem E-Procurement-Beispiel führte der Einsatz der Wrapper zu einem Leistungsmehraufwand von etwa 5%. Der Einsatz des Trust Managers, mit dem man bei guten Vertrauenswerten einige Observer abschalten kann, reduziert den Mehraufwand auf etwa 3%.

Neben dieser Arbeit konzentriert sich unser Ansatz auch auf die Entwicklung eines cTLA-Frameworks zur Modellierung der Sicherheitseigenschaften von komponentenbasierten Systemen. Das Framework enthält eine Sammlung an Mustern zur Beschreibung von Sicherheitspolicies einzelner Komponenten, wie wir sie in Abschnitt 5 vorgestellt haben. Daneben wird ihm eine weitere Sammlung an Mustern hinzugefügt, mit der man an das Gesamtsystem gestellte Sicherheitsanforderungen formal beschreiben kann. Darüberhinaus werden von uns bereits verifizierte Theoreme hinzugefügt, die jeweils aussagen, dass eine Kombination von Komponenten-Sicherheitspolicies eine Systemsicherheitsanforderung erbringt. Mit den Mustersammlungen kann man die Sicherheitsanforderungen recht einfach spezifizieren, indem man entsprechende Muster parametrisiert und zu einer umfassenderen Sicherheitsspezifikation kombiniert. Das Framework erleichtert zudem die logischen Deduktionsbeweise. Ein Beweis kann nämlich in Lemmata zerlegt werden, die den Frameworktheoremen entsprechen. Formale Verifikationen sind insbesondere für extrem sicherheitskritische Anwendungen relevant, da sie vom ISO/IEC Common Criteria-Standard [20] für Sicherheitsanalysen derartiger Systeme zwingend vorgeschrieben sind. Wir haben den Ansatz von Spezifikationsframeworks bereits erfolgreich auf den Gebieten der Telekommunikationsprotokolle [14, 16] und der hybriden technischen Systeme [17] angewendet. Das zu entwickelnde Spezifikationsframework für die Komponentensicherheit kann die grundlegenden Techniken und Werkzeuge dieser Frameworks übernehmen (vgl. dazu [16, 19, 27]).

Literatur

1. F. M. Avolio und M. J. Ranum. A Network Perimeter with Secure External Access. In *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, Glenwood, 1994.
2. B. Bershad, S. Savage, P. Pardyak, E. G. Sirer, D. Becker, M. Fiuczynski, C. Chambers und S. Eggers. Extensibility, safety, and performance in the SPIN operating system. In *Proceedings of the 15th Symposium on Operating System Principles*, Seiten 267–284. ACM, 1995.
3. T. Beth, M. Borcherding und B. Klein. Valuation of Trust in Open Networks. In *Proceedings of the European Symposium on Research in Security (ESORICS)*, Lecture Notes in Computer Science 875, Seiten 3–18, Brighton, 1994. Springer-Verlag.
4. A. Beugnard, J.-M. Jézéquel, N. Plouzeau und D. Watkins. Making Components Contract Aware. *IEEE Computer*, 32(7):38–45, 1999.
5. J. Biskup und C. Eckert. About the enforcement of state dependent security specifications. In T. Keefe und C. Landwehr (Hrsg.), *Database Security*, Seiten 3–17. Elsevier Science (NorthHolland), 1994.
6. G. Booch, J. Rumbaugh und I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Longman, 1999.
7. cXML.org. *cXML User's Guide*, Ausgabe 1.2.006, Aug. 2001.
8. Data Interchange Standards Association. *X12 Standard*, Ausgabe 4050, Dez. 2001.
9. eBay Inc. Feedback Forum. Verfügbar über WWW: pages.ebay.com/services/forum/feedback.html.
10. E. Ferrari, P. Samarati, E. Bertino und S. Jajodia. Providing flexibility in information flow control for object-oriented systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, Seiten 130–140, Oakland, 1997.
11. T. Fraser, L. Badger und M. Feldman. Hardening COTS Software with Generic Software Wrappers. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 1999.
12. I. Goldberg, D. Wagner, R. Thomas und E. Brewer. A Secure Environment for Untrusted Helper Applications. In *Proceedings of the 6th USENIX Security Symposium*, 1996.
13. G. Graw, P. Herrmann und H. Krumm. Constraint-Oriented Formal Modelling of OO-Systems. In *Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS 99)*, Seiten 345–358, Helsinki, Juni 1999. Kluwer Academic Publisher.
14. P. Herrmann. *Problemnaher korrektheitssichernder Entwurf von Hochleistungsprotokollen*. Deutscher Universitätsverlag, 1998.
15. P. Herrmann. Trust-Based Procurement Support for Software Components. In *Proceedings of the 4th International Conference on Electronic Commerce Research (ICECR-4)*, Seiten 505–514, Dallas, Nov. 2001. ATSMA, IFIP.
16. P. Herrmann und H. Krumm. A Framework for Modeling Transfer Protocols. *Computer Networks*, 34(2):317–337, 2000.
17. P. Herrmann und H. Krumm. A Framework for the Hazard Analysis of Chemical Plants. In *Proceedings of the 11th IEEE International Symposium on Computer-Aided Control System Design (CACSD2000)*, Seiten 35–41, Anchorage, 2000. IEEE CSS, Omnipress.
18. P. Herrmann und H. Krumm. Trust-adapted enforcement of security policies in distributed component-structured applications. In *Proceedings of the 6th IEEE Symposium on Computers and Communications*, Seiten 2–8, Hammamet, Juli 2001. IEEE Computer Society Press.
19. P. Herrmann, H. Krumm, O. Drögehorn und W. Geisselhardt. Framework and Tool Support for Formal Verification of High Speed Transfer Protocol Designs. Erscheint in *Telecommunication Systems*, 2002.
20. ISO/IEC. *Common Criteria for Information Technology Security Evaluation*, 1998. International Standard ISO/IEC 15408.
21. A. Jøsang. The right type of trust for distributed systems. In *Proceedings of the UCLA conference on New security paradigms workshops*, Seiten 119–131, Lake Arrowhead, Sept. 1996. ACM.
22. A. Jøsang. An Algebra for Assessing Trust in Certification Chains. In J. Kochmar (Hrsg.), *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'99)*. The Internet Society, 1999.
23. A. Jøsang und S. J. Knapskog. A metric for trusted systems. In *Proceedings of the 21st National Security Conference*. NSA, 1998.

24. K. Khan, J. Han und Y. Zheng. A Framework for an Active Interface to Characterise Compositional Security Contracts of Software Components. In *Proceedings of the Australian Software Engineering Conference (ASWEC'01)*, Seiten 117–126, Canberra, 2001. IEEE Computer Society Press.
25. D. Kozen. Efficient code certification. Technischer Bericht 98–1661, Computer Science Department, Cornell University, 1998.
26. D. Kozen. Language-Based Security. In M. Kutylowski, L. Pacholski und T. Wierzbicki (Hrsg.), *Proceedings of the Conference on Mathematical Foundations of Computer Science (MFCS'99)*, Lecture Notes in Computer Science 1672, Seiten 284–298. Springer-Verlag, 1999.
27. L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, Mai 1994.
28. A. Mallek. Sicherheit komponentenstrukturierter verteilter Systeme: Vertrauensabhängige Komponentenüberwachung. Diplomarbeit, Universität Dortmund, Informatik IV, D-44221 Dortmund, 2000.
29. B. P. Miller, L. Fredrikson und B. So. An Empirical Study of the Reliability of Unix Utilities. *Communications of the ACM*, 32(12):32–44, Decz. 1990.
30. M. A. Monroe. Security Tool Review: TCP Wrappers. *;login:*, 18(6):15–16, 1993.
31. G. Morrisett, D. Walker, K. Crary und N. Glew. From System F to typed assembly language. In *Proceedings of the 25th ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, Seiten 85–97, San Diego, 1998.
32. A. C. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages (POPL'99)*, San Antonio, 1999.
33. A. C. Myers und B. Liskov. Complete, Safe Information with Decentralized Labels. In *Proceedings of the IEEE Symposium on Security and Privacy*, Seiten 186–197, Oakland, 1998.
34. G. C. Necula. *Compiling with proofs*. Doktorarbeit, Carnegie Mellon University, 1998.
35. OBI Consortium. *OBI Technical Specifications — Open Buying on the Internet*, Draft Release Ausgabe V2.1, 1999.
36. P. Resnick, R. Zeckhauser, E. Friedman und K. Kuwabara. Reputation Systems: Facilitating Trust in Internet Interactions. *Communications of the ACM*, 43(12):45–48, Dez. 2000.
37. L. Schmitz. The SalesPoint Framework — Technical Overview. Verfügbar über WWW: ist.unibw-muenchen.de/Lectures/SalesPoint/overview/english/TechDoc.htm, Nov. 1999.
38. F. B. Schneider. Towards fault-tolerant and secure agentry. In *Proceedings of the 11th International Workshop on Distributed Algorithms (WDAG'97)*, Lecture Notes in Computer Science 1320, Seiten 1–14. ACM SIGPLAN, Springer-Verlag, 1997.
39. M. Shepherd, A. Dhonde und C. Watters. Building Trust for E-Commerce: Collaborating Label Bureaus. In W. Kou, Y. Yesha und C. J. Tan (Hrsg.), *Proceedings of the 2nd International Symposium on Electronic Commerce Technologies (ISEC'2001)*, LNCS 2040, Seiten 42–56, Hong Kong, Apr. 2001. Springer-Verlag.
40. C. Szyperski. *Component Software — Beyond Object Oriented Programming*. Addison-Wesley Longman, 1997.
41. D. Tarditi, G. Morrisett, P. Cheng, C. Stone, R. Harper und P. Lee. TIL: A type-directed optimizing compiler for ML. In *Proceedings of the Conference on Programming Language Design and Implementation*. ACM SIGPLAN, 1996.
42. C. A. Vissers, G. Scollo und M. van Sinderen. Architecture and specification style in formal descriptions of distributed systems. In S. Agarwal und K. Sabnani (Hrsg.), *Protocol Specification, Testing and Verification VIII*, Seiten 189–204, Elsevier, 1988. IFIP.
43. J. Voas. A Recipe for Certifying High Assurance Software. In *Proceedings of the 22nd International Computer Software and Application Conference (COMPSAC'98)*, Wien, Aug. 1998. IEEE Computer Society Press.
44. J. Voas, G. McGraw, A. Ghosh und K. Miller. Glueing together Software Components: How good is your Glue? In *Proceedings of the Pacific Northwest Software Quality Conference*, Portland, Okt. 1996.
45. R. Wabbe, S. Lucco, T. E. Anderson und S. L. Graham. Efficient software-based fault isolation. In *Proceedings of the 14th Symposium on Operating System Principles*, Seiten 203–216. ACM, 1993.
46. J. Zöllner, H. Federrath, H. Klimant, A. Pfitzmann, R. Piotraschke, A. Westfeld, G. Wicke und G. Wolf. Modeling the security of steganographic systems. In *Proceedings of the 2nd Workshop of Information Hiding*, LNCS 1525, Seiten 345–355, Portland, 1998. Springer-Verlag.