Trust-Based Security Policy Enforcement of Software Components*

Peter Herrmann
University of Dortmund, Computer Science Department, 44221 Dortmund, Germany
Email: Peter.Herrmann@cs.uni-dortmund.de

Abstract

The software component technology facilitates the suitable and inexpensive creation of applications by composing independently developed components. This design method, however, causes new security risks. In particular, a malicious component is a threat to an incorporating application. To guard component-structured software against this threat, we use security wrappers monitoring the behavior of components during runtime and enforcing an application's security policy. A component monitor observes the interface behavior of a component and checks it for compliance with the security behavior description modelled in the employment contract of the component. Unfortunately, security wrappers cause a significant performance overhead which, however, can be reduced by trust management. Here, the decision about the kind and intensity of observing a component takes the experiences other application owners gained with it into consideration. In particular, we use a trust information service collecting good and bad experience reports about components. Moreover, a trust manager component forms a link between the security wrappers and the trust information service. On the one hand, the trust manager hands over experience reports to the trust information service describing the results of monitoring a component. On the other hand, it uses the collection of experience reports on a component, which is represented by a so-called trust value, to adjust the intensity of monitoring the component.

In this paper we introduce the elements of the trust-based runtime enforcement approach and clarify their employment by means of an example application.

1 Introduction

The approach of component-structured software facilitates the easy and cost-effective creation of applications well suited to the particular needs of a software owner by combining separate software components to a system. The components are created independently from each other and are separately offered on an open market (cf. [16]). The heterogeneity of the components, however, aggravates the composition process. On the one hand, the interfaces of the components have to be adapted in order to fit with each other. This is facilitated by means of — ideally legally binding — contracts describing all context dependencies of a component explicitly. On the other hand, the approach imposes new security aspects due to the high number of principals. In addition to application owners and users we have also to consider the component developers and the people combining the components to a system. Moreover, if components are not integrated locally to the application but executed on a remote host and incorporated by means of a telecommunication service, also the owners of the host computers have to be taken into consideration. These principals introduce their own security objectives. Moreover, each principal is also a potential threat to the application and to the other

*In Proceedings of the *1st Internal iTrust Workshop on Trust Management in Dynamic Open Systems*, Glasgow, 2002.

principals. Some of the resulting security aspects for component-structured systems are listed in the taxonomy of Lindqvist and Jonsson [11] and a more comprehensive list is introduced in [3].

Below, we concentrate on an important security aspect for application owners: the threat that a component behaves maliciously distorting components of its environment and, in consequence, spoiling the incorporating system. Our approach to guard a system against malicious components utilize the component contracts as well. According to Beugnard et al. [2] a contract consists of four parts specifying the structure of a component interface (i.e., methods and events with input and output parameters, exceptions), the desired behavior of the component and its environment, synchronization aspects, and quantitative quality-of-service properties. We extend the behavior description by models specifying security-relevant behavior assuming that malicious components or compromised code (e.g., by a virus or a Trojan horse infection) result in behaviors which diverge from the modelled behaviors. By means of the models we can prove at design time, that the components behave in accordance with the security properties of the system. Since both the behavior models in the contract and the specification of the security properties are specified formally in the specification language cTLA [4], one can carry out this proof formally by a logical deduction.

Here, we concentrate on the run-time enforcement of component contracts guaranteeing that the real behavior of a component conforms with the contract models. The behavior is observed by means of security wrappers [5, 6] which are inserted at the interface between a component of interest and its environment. A wrapper temporarily blocks all events passing the interface and checks them for compliance with the behavior models in the contracts. If an event complies with all models, it is allowed to pass. If it, however, contradicts with a model, the wrapper seals the component temporarily by blocking further events and notifies the application administrator.

Unfortunately, the use of security wrappers leads to a significant performance overhead of 5 to 10%. Therefore we combine the security wrappers with active trust management [3]. Following the idea of reputation systems (cf. [14]) we use a so-called trust information service. In intervals application owners send good or bad experience reports on a component to the trust information service which collects all reports. Moreover, based on a number of positive and negative valuations special trust values (cf. [7]) are calculated. The application owners use the trust values to adjust the kind and intensity of runtime enforcement comprising permanent monitoring of a component, spot-checks, and the complete removal of a wrapper. Thus, the performance overhead for checking a component depends on the experience various application owners got with it during runtime.

In this paper we outline the use of the trust-based security enforcement approach by means of an e-commerce example describing an electronic requisition process for fast-food restaurants (cf. [6]). In the next section we introduce the trust information service. Thereafter, we sketch the security wrapper-based runtime enforcement approach. Finally, we outline the example application and describe our trust-based enforcement policy.

2 Trust Information Service

A trust information service facilitates trust management of component-structured software by collecting experience reports and passing the reports to interested customers and application owners. In order, to enable automated trust-based runtime enforcement of components, the number of reports are described by the means of trust values (cf. [7]). Here, a mathematical formula describes the belief resp. disbelief in a component. Moreover, one can state that the number of experience reports are too low in order to give a decent evaluation. In [8], Jøsang introduces the so-called *Opinion Triangle* (cf. Fig 1) modelling trust values. Belief, disbelief, and uncertainty are specified by means of the values b, d, and u each being a real number in the interval $[0, 1]$. Since the constraint $b + d + u = 1$ holds, trust values can be depicted by a point in the triangle (see Fig 1). A point close to the top of the perpendicular states uncertainty about a component due to a low number of reports. If more

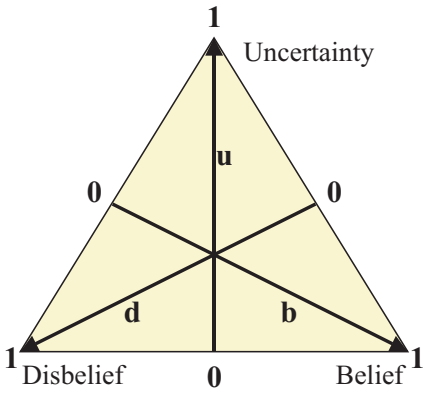

Figure 1: Opinion Triangle (taken from [8])

reports are received, the uncertainty value decreases. Then points on the right or left side describe great belief resp. disbelief in a component.

The trust values are calculated from the numbers p of positive and n of negative experience reports by applying metrics. For instance, Jøsang and Knapskog [10] calculate the three values b, d, and u of the opinion triangle from p and n by means of the following formulas:

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

This metric expresses a relatively tolerant trust management philosophy since a negative experience report can be compensated by a number of positive reports. In contrast, the metric of Beth, Borcherding, and Klein [1] expresses an unforgiving philosophy. The probability that b exceeds a certain value is stated by the formula

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

and d resp. u are calculated from b by the formulas:

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

According to this metric a single negative experience report destroys the belief in a component forever. If all reports are positive, b increases in dependence on the number of reports. The two metrics describe two different ways to gain trust and the decision which metric to select to adjust the observation of components should be guided by the damage, a successful component attack has on the institution running the malicious component. To support different enforcement policies, therefore the trust information service offers two trust values calculated with both metrics listed above.

The trust information service is delineated in Fig. 2. It is linked with component vendors resp. developers, with interested principals procuring components and monitoring them during runtime, and authorities certifying components in behalf of vendors or users. Component vendors may register any software component offered commercially or free of charge declaring themselves to agree that experience reports are collected from users and that corresponding trust values are offered to interested parties. Component customers and application owners may inquire trust values from the trust information service at any time. Moreover, they can subscribe to an alarm service which notifies them immediately if another user reported on malicious behavior of a component of interest. To receive as many reports as possible, all application owners are called in intervals to report on the components in the application. Moreover, if a component user detects a serious incident indicating

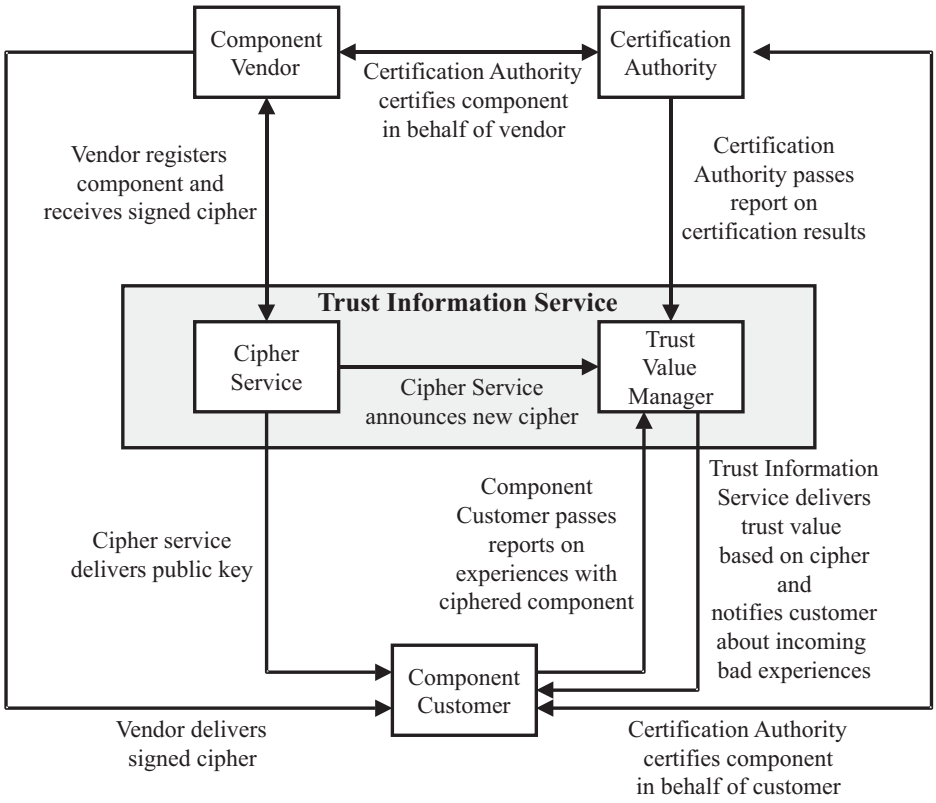

Figure 2: Trust Information Service

malicious code, he should inform the trust information service immediately which causes the alarm service to notify all subscribers. Moreover, the results of component certifications are considered in the calculation of trust values.

To guarantee a high degree of privacy for the component vendors, the experience reports are separated from the component descriptors. Therefore the trust information service consists of a cipher service and a trust value manager. The cipher service stores the registration data of a component and its vendor and generates a cipher which the vendor hands over to interested component users. The trust value manager stores the experience reports and trust values based on the ciphers without knowing the true identity of a component. Thus, neither the cipher service nor the trust value manager have complete knowledge about a component and its trust values.

3 Security Wrapper Architecture

The enforcement of the security objectives specified in the component contract models is performed by security wrappers (cf. [5, 6]). Here, a component to be observed is not connected directly with its environment. Instead, special wrapper components are put in between which check all in- and outgoing interface events for compliance with the behavior described in the contract models. If a contract model is violated by an event, the security wrapper notifies the application administrator. Moreover, it seals the component by blocking further events until release from the administrator.

Figure 3 depicts a wrapper implementation for component-structured software based on Java Beans [12]. Each observed bean is wrapped by an adapter component discerning all events passing the bean interface. The adapter blocks an incoming or outgoing event temporarily and notifies the observer components. Each observer component checks the compliance of the event with a component contract model by simulating the model. If the event is accepted by all observers, the

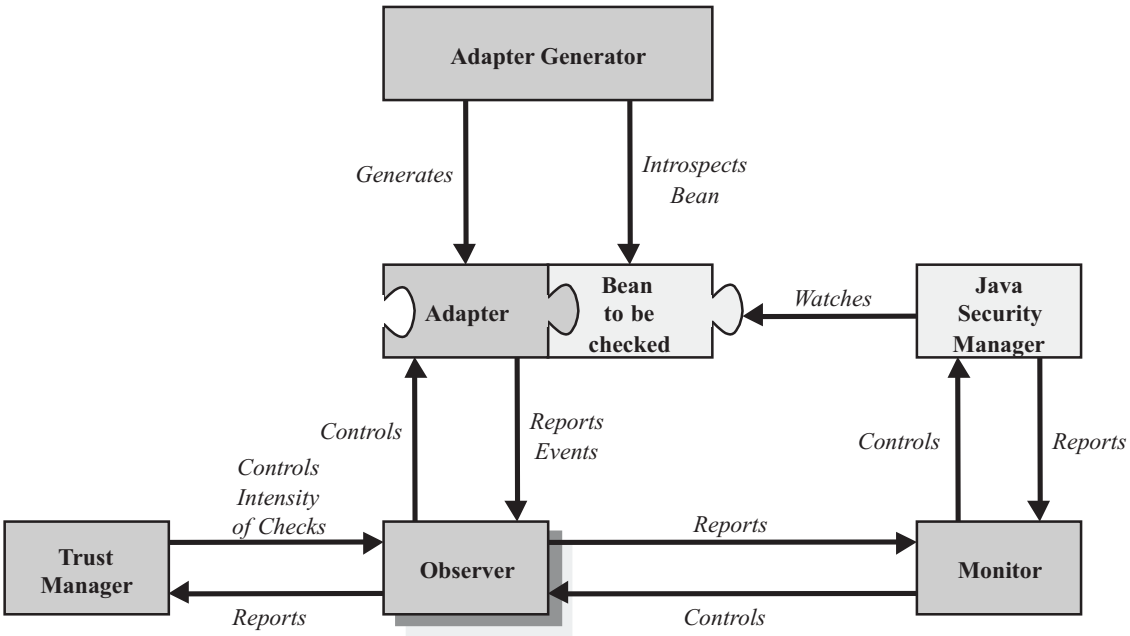

Figure 3: Security Wrapper Architecture and Trust Manager

adapter forwards it to its destination. If an observer, however, detects a contract violation, the adapter blocks all further events until receiving a release order from the application administrator. Moreover, the adapter notifies the administrator by means of the monitor component which forms the administrator's interface. Another component is the adapter generator which creates adapters automatically by applying the Java introspection mechanism in order to detect the event structure of a bean. Finally, the wrapper utilizes the built-in Java security manager to prevent hidden data channels of a wrapped bean. The security manager permits only events linking the observed bean with its adapter.

Trust managers are special components linking a security wrapper with the trust information service. Mainly, a trust manager controls the contract enforcement process. In intervals it inquires the current trust value of an observed component from the trust information service and adjusts the enforcement policies accordingly. If the trust value of a component states a high degree of uncertainty or disbelief, the component is permanently fully observed. If the value indicating the belief in a component exceeds a certain limit, the monitoring is reduced to spot checks where the compliance checks are only performed in times. In this mode, however, the current states of the simulated contract models are adjusted permanently in order to enable the reactivation of the compliance checks. Finally, if a trust value describes a very high belief in a component, the component can be assumed to be correct and good-natured and the particular security wrapper is removed.

Other tasks of a trust manager comprise the transmission of experience reports to the trust information service. If the trust manager receives a call for reports from the trust information service, it returns positive reports about all monitored components which did not violate the contract models since the last report. In contrast, a minor contract model violation of a component leads to a negative experience report on it. In the case of a major violation the trust manager notifies the trust information service immediately in order to execute the alarm service warning other component users. If the trust manager receives an alarm message about a component, it causes the wrapper to notify the application administrator and the adapter to seal the component.

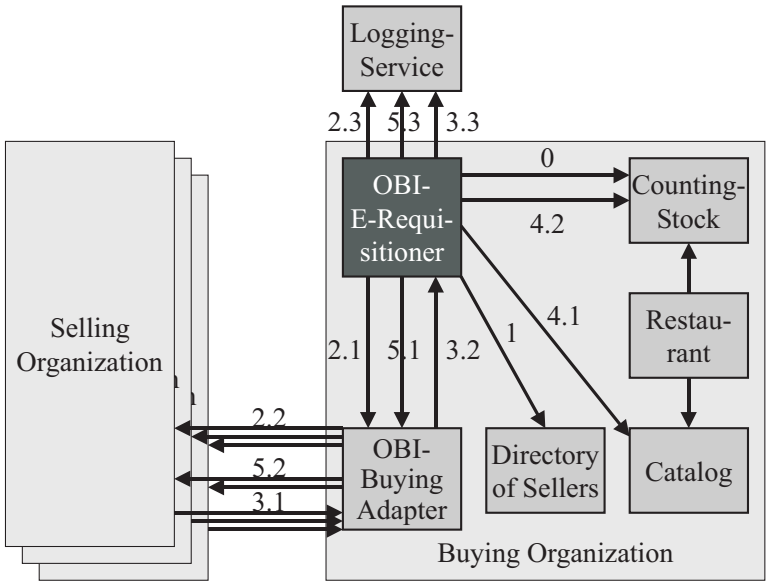

Figure 4: E-Procurement-System for fast-food restaurants

4 Component-Structured E-Procurement Example

The example application (cf. [6]) performs the commodity management of fast-food franchise restaurants. It is based on the standard Open Buying on the Internet (OBI, [13]) which standardizes the tender request, tender, and order procedures of electronic procurements. Fig. 4 delineates the components of the example. The component *Restaurant* realizes the sale functions of the restaurant while *Counting Stock* and *Catalog* provides management functions for the restaurant's counting stock resp. catalog of offered goods. These three components are refined from the SalesPoint-Framework [15], a non-profit collection of Java modules facilitating the construction of various shop systems. Moreover, we created the component *OBI-E-Requisitioner* which realizes the automated requisition[1] of goods. Moreover, we added a *Directory of Sellers* containing the addresses and range of goods for sale of suppliers. Finally, *OBI-Buying Adapter* manages the formatting of tender requests, tenders, and orders according to the OBI specification and acts as an interface to the suppliers. The buying organization (i.e., the fast-food restaurant) is implemented by the combination of these six components. The corresponding selling organizations (i.e., the suppliers) were developed from the SalesPoint-Framework, too. Furthermore, we created a trusted third party logging service in order to support non-repudiation of transactions. The components can be downloaded from the WWW (URL: ls4-www.cs.uni-dortmund.de/RVS/P-SACS/eReq).

The steps of a procurement are depicted by the edge labels in Fig. 4. In step 0, the e-requisitioner component inspects the counting stock in intervals in order to detect the shortage of a good in the stock. If new goods are needed, the e-requisitioner requests the addresses of selling organizations from the directory of sellers (step 1). Thereafter tender requests are generated and forwarded to the selling organizations via the buying adapter (step 2). The sellers react with tenders which are sent to the buying adapter and delivered to the e-requisitioner (step 3). In step 4 the e-requisitioner consults the catalog and the counting stock beans, makes a procurement decision based on the tenders, the stock volume, and the sale prices, and creates an order. Finally, the order is sent to the buying adapter which forwards them to the corresponding selling organization (step 5). In order to log the tender requests, incoming tenders, and orders, in the steps 2, 3, and 5 the requisitioner sends the corresponding log data to the logging service.

[1] In order to use automated procurement, we extended the OBI standard which assumes that the requisition is performed by humans only.

The procurement process is controlled by the electronic requisitioner. Therefore the correct and secure execution of the component *OBI-E-Requisitioner* is crucial for the buying organization. If the e-requisitioner behaves maliciously, various security violations of the application are possible (e.g., forwarding of competitor's tenders to preferred selling organizations, ordering not from the least expensive seller, hurting the buyer by too large, too small, resp. too late orders, repudiation of orders). Thus, we have to protect the application by means of a security wrapper observing *OBI-E-Requisitioner*. In particular, we created 13 component contract models stating four confidentiality policies, four integrity policies, four availability policies, and a non-repudiation policy. The models are specified in cTLA (cf. [4]) and were transferred to Java code in order to be simulated. The cTLA specifications and the Jva code are available on the WWW (URL: ls4-www.cs.uni-dortmund.de/RVS/P-SACS/eReq).

To guarantee confidentiality, we demand that a tender must be requested only from sellers which are in the directory of sellers. Moreover, tender requests are sent to sellers only which, according to the directory of sellers, are in the range of articles offered. Thus, information about the existence of the procurement procedure and the portfolio of the buying organization are only forwarded to appropriate sellers. Two other security policies make the use of hidden channels (e.g., steganography: concealing information in transferred data) more difficult by avoiding non-deterministic interface behavior (cf. [17]). At first, the order amount for an article depends unambiguously from the amount of the particular article in the stock. At second, a tender request and an order may be executed only if the last order was carried out in the meantime.

With respect to integrity, two security policies guarantee that all selling organizations have a fair chance to win an order. Therefore we demand that an order may be generated only after a certain minimum number of tenders were received. Moreover, the electronic requisitioner must order one of the least expensive tenders. A third security policy demands that *OBI-E-Requisitioner* does not change the values in the coupled components in order to avoid attacks on them. Finally, unreasonable order amounts are prevented by demanding certain minimum and maximum values.

With respect to availability, we distinguish between denial-of-service attacks, where a component is called too often in order to prevent it to serve third components, and blocking attacks, where a desired interface action is refused blocking the partner component. Denial-of-service attacks on the partner components are prevented by demanding that calls may be performed only after minimum waiting time intervals. Three other policies are used to avoid blocking attacks of partner components slowing down the procurement process which might lead to empty stocks. Therefore, the polling of the counting stock, the transmission of the tender requests, and the transmission of orders have to be triggered within a maximum waiting time.

Finally, we use a non-repudiation security policy assuring that the buying organization can audit requested and incoming tenders as well as orders. Therefore the tender requests, the tender deliveries, and the orders have to be logged at the logging service.

The security wrapper checks the events passing the interface of the component *OBI-E-Requisitioner* for compliance with the 13 security policies. According to performance tests, the wrapper causes a performance penalty of 5.4% in our reference implementation. To reduce this penalty, we use a trust manager that manages the security policies based on three different trust management levels.

The highest level comprises the integrity policies since too expensive procurement of goods may severely damage the buying organization. Likewise, too large orders of the mostly perishable goods leads to significant expense as well. Moreover, also illegal value changes of partner components are dangerous since by them the application may easily be spoiled. Furthermore, we use the highest trust management level to the two confidentiality policies preventing the use of hidden channels. Here, the main danger is that tenders of competitors may be forwarded to a preferred supplier. The highest level is also assigned to the non-repudiation policy since a violation of this policy may cause severe

legal consequences for the buying organization. Due to the relevance of these security policies, we demand that the policies are constantly monitored and that a violation leads not only to a negative experience report but also to an immediate alarm notification to the trust information service.

The second trust management policy is used for the three availability policies preventing a retardation of the procurement process. While an empty stock leads also to financial losses, this kind of attack can be easier detected since we assume that, as a precaution, the restaurant managers supervises the stock from time to time as well. Thus, a shortage of a product can probably be recognized and cleared before any damage is caused. For this policy, the trust manager applies the metric of Beth, Borcherding, and Klein [1] permitting spot checks if the belief value b exceeds 0.999. According to the probability formula of this metric (cf. Sec. 2), about 7000 positive experience reports[2] have to be received to reach a sufficient value of b. The security wrapper may be removed if b exceeds 0.99999 which corresponds to about 11600 positive reports. Like in the highest level, a detected policy violation leads to an immediate alarm message to the trust information service.

The remaining security policies are less critical. The confidentiality policies preventing the leaking of the assortment of goods is not important since, in general, the range of goods sold in a fast-food restaurant are well-known. Likewise, the danger of a denial-of-service attack is less critical since this attack is usually immediately noticed by the sales people who have problems to use the tills and the error sources can be easily detected and cleared. For these security policies we use the more tolerant metric of Jøsang and Knapskog [10]. Spot checks are allowed if b exceeds 0.99 while a value of 0.999 enables the trust manager to remove the security wrapper at all. Thus, at least 100 resp. 1000 positive experience reports have to be collected and the number of positive reports must exceed the number of negative reports by the factor 100 resp. 1000. In contrast to the other trust management levels, a violation does not lead to an immediate alarm message but only to a negative experience report.

In our test executions, the trust manager causes a significant reduction of the performance penalty. After receiving about 100 positive reports, the enforcement of the policies of the third trust management level was reduced to spot checks which caused a reduction from 5.4% to around 5.0%. When 1000 good reports were received, the corresponding observers were removed and the penalty decreased to 4.2%. After collecting 7000 positive reports, the three policies of the second level were only spot checked and a penalty of 3.7% was gauged. Finally, after 11600 positive reports the observers for the policies of the second trust management level were removed and the performance penalty was reduced to 3.2%. Thus, by applying trust management we could reduce the penalty almost by half.

5 Concluding Remarks

In this submission we outlined our approach for the trust-based enforcement of formal contract models consisting of a trust management service collecting experience reports and calculating trust values, of security wrappers guaranteeing the enforcement of security policies, and of trust managers managing the enforcement process and generating experience reports. We intend to extend the trust management service in order to reduce the danger of wrong incriminations of component vendors by application users who may send unjustified bad reports leading to negative trust values. Here, we think of special arbiter components which decide about accepting negative experience reports. The mediation may be based on the logs of interface event chains leading to the bad report as well as on certain witness components which are trusted third parties checking the interface events of a component independently from the security wrappers.

Moreover, one can improve the quality of trust values by evaluating the application owners as well. Besides of experience reports about components the trust information service collects also

[2]The trust manager demands a probability of at least 0.999 to relax the enforcement policy.

reports on application owners provided by the arbiter components. These trust values can be used to gain so-called recommendation trust, i.e., the trust that a recommendation in a third party is correct. Both Beth, Borcherding, and Klein [1] and the Subjective Logic of Jøsang [9] facilitate the consideration of recommendation in trust values about entities. Thus, the experience reports about components are rated higher if they come from highly trusted application owners who gave many decent reports before.

References

[1] T. Beth, M. Borcherding, and 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, pages 3–18, Brighton, 1994. Springer-Verlag.

[2] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making Components Contract Aware. *IEEE Computer*, 32(7):38–45, 1999.

[3] P. Herrmann. Trust-Based Procurement Support for Software Components. In *Proceedings of the 4th International Conference on Electronic Commerce Research (ICECR-4)*, pages 505–514, Dallas, Nov. 2001. ATSMA, IFIP.

[4] P. Herrmann and H. Krumm. A Framework for Modeling Transfer Protocols. *Computer Networks*, 34(2):317–337, 2000.

[5] P. Herrmann and H. Krumm. Trust-adapted enforcement of security policies in distributed component-structured applications. In *Proceedings of the 6th IEEE Symposium on Computers and Communications*, pages 2–8, Hammamet, July 2001. IEEE Computer Society Press.

[6] P. Herrmann, L. Wiebusch, and H. Krumm. State-Based Security Policy Enforcement in Component-Based E-Commerce Applications. In *Proceedings of the 2nd IFIP Conference on E-Commerce, E-Business & E-Government (I3E)*, Lisbon, 2002. Kluwer Academic Publisher.

[7] A. Jøsang. The right type of trust for distributed systems. In *Proceedings of the UCLA conference on New security paradigms workshops*, pages 119–131, Lake Arrowhead, Sept. 1996. ACM.

[8] A. Jøsang. An Algebra for Assessing Trust in Certification Chains. In J. Kochmar, editor, *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'99)*. The Internet Society, 1999.

[9] A. Jøsang. Subjective Evidential Reasoning. In *Proceedings of the 9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2002)*, Annecy, July 2002.

[10] A. Jøsang and S. J. Knapskog. A metric for trusted systems. In *Proceedings of the 21st National Security Conference*. NSA, 1998.

[11] U. Lindqvist and E. Jonsson. A Map of Security Risks Associated with Using COTS. *IEEE Computer*, 31(6):60–66, 1998.

[12] A. Mallek. Sicherheit komponentenstrukturierter verteilter Systeme: Vertrauensabhängige Komponentenüberwachung (in German). Diploma Thesis, Universität Dortmund, Informatik IV, D-44221 Dortmund, 2000.

[13] OBI Consortium. *OBI Technical Specifications — Open Buying on the Internet*, draft release v2.1 edition, 1999.

[14] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara. Reputation Systems: Facilitating Trust in Internet Interactions. *Communications of the ACM*, 43(12):45–48, Dec. 2000.

[15] L. Schmitz. The SalesPoint Framework — Technical Overview. Available via WWW: ist.unibw-muenchen.de/Lectures/SalesPoint/overview/english/TechDoc.htm, Nov. 1999.

[16] C. Szyperski. *Component Software — Beyond Object Oriented Programming*. Addison-Wesley Longman, 1997.

[17] J. Zöllner, H. Federrath, H. Klimant, A. Pfitzmann, R. Piotraschke, A. Westfeld, G. Wicke, and G. Wolf. Modeling the security of steganographic systems. In *Proceedings of the 2nd Workshop of Information Hiding*, LNCS 1525, pages 345–355, Portland, 1998. Springer-Verlag.