Trust-Based Protection of Software Component Users and Designers

Peter Herrmann

University of Dortmund, Computer Science Department,
44221 Dortmund, Germany,
Peter.Herrmann@udo.edu

Abstract. Software component technology supports the cost-effective design of applications suited to the particular needs of the application owners. This design method, however, causes two new security risks. At first, a malicious component may attack the application incorporating it. At second, an application owner may incriminate a component designer falsely for any damage in his application which in reality was caused by somebody else. The first risk is addressed by security wrappers controlling the behavior at the component interface at runtime and enforcing certain security policies in order to protect the other components of the application against attacks from the monitored component. Moreover, we use trust management to reduce the significant performance overhead of the security wrappers. Here, the kind and intensity of monitoring a component is adjusted according to the experience of other users with this component. Therefore a so-called trust information service collects positive and negative experience reports of the component from various users. Based on the reports, special trust values are computed which represent the belief or disbelief of all users in a component resp. the uncertainty about it. The wrappers adjust the intensity of monitoring a component dependent on its current trust value.
In this paper, we focus on the second security risk. To prevent that a component user sends wrong reports resulting in a bad trust value of the component, which therefore would be wrongly incriminated, the trust information service stores also trust values of the component users. The trust values are based on valuations resulting from validity checks of the experience reports sent by the component users. Therefore an experience report is tested for consistency with a log of the component interface behavior which is supplied by the component user together with the report. Moreover, the log is checked for being correct as well. By application of Jøsang's subjective logic we make the degree, to which the experience reports of a component user are considered to compute the trust value of a component, conditional upon the user's own trust value. Thus, users with a bad reputation cannot influence the trust value of a component since their experience reports are discounted.

1 Introduction

Component-structured software gets more and more popular since applications can be cost-effectively composed from components which are developed independently from each other and are separately offered on an open market (cf. [1]). Suitable components are selected according to the particular needs of the desired user and are coupled to an application. The components are either executed locally on the application's host or run on a remote server and are integrated by means of a special telecommunication service.

The composition process, however, is aggravated by the heterogeneity of the component interfaces. Here, component contracts, which are ideally legally binding, prove helpful. They can be used to adapt the interfaces in order to fit to each other since the context dependencies of a component have to be explicitly stated in its contract. According to Beugnard et al. [2] a contract consists of four parts modeling the structure of a component interface (i.e., the methods with input and output parameters, events, exceptions), constraints about the interface behavior demanded from the component and its environment, synchronization aspects, and quantitative quality-of-service properties. Moreover, reflection and introspection [3] facilitate the coupling of components by providing special methods enabling the exploration of component properties, methods, and interfaces at runtime. These methods are utilized by visual application builder tools (e.g., [4]) making the component composition easier. Well-established platforms for component-structured software are Java Beans [3] and, in particular, Enterprise Java Beans [5], Microsoft's COM/DCOM [6], and the CORBA component model [7]. Each platform provides notations to describe component types, parameter types, and interfaces. Furthermore, means to introspect components and special composition support are provided as well.

The heterogeneity of the components also results in a new class of security risks. Compared with ordinary monolithic applications in component-structured software new principals and roles are introduced. Besides of application owners and users we have to consider also a number of different component designers as well as application builders and component service providers. On the one hand, these principals introduce their own security objectives which have to be fulfilled. On the other hand, each principal is also a potential threat to the application, its components, and the other principals. A taxonomy of security risks for components is introduced in [8] and an extended list is published in [9].

Here, we concentrate on two main security risks:

1. A malicious component must not be able to distort components of its environment and, in consequence, spoil the whole application incorporating it.
2. A component and its designer must not be incriminated falsely for damage in a component-structured application which in reality was caused by another principal.

In [10, 11] we introduce an approach addressing the first risk. It is based on the component contracts which are extended by models specifying security-relevant behavior to be fulfilled by a component and its behavior. We assume that malicious or compromised components behave in a way which diverges from the contract models. Therefore we use so-called security wrappers observing the interface behavior of the component at runtime. Security wrappers are specialized software wrappers (cf. [12]) which are pieces of code extending a component. A security wrapper is inserted at the interface between a component and its environment. It temporarily blocks an event passing the interface and checks it for compliance with the behavior models in the contract which are simulated. If the event complies with all models, it may pass. Otherwise, the wrapper seals the component temporarily in order to prevent harm for the component environment and notifies the application administrator.

Since the security wrappers cause a significant performance overhead of 5 to 10%, we combined the approach with active trust management [9, 13]. The intensity of the runtime enforcement by the security wrappers is adjusted according to the reputation of the observed component. To serve this purpose, a component user creates an experience report in intervals and sends it to the so-called trust information service. A positive report states that no contract model was violated by the component after sending the last report while in the case of detecting a contract violation a negative report is issued. The trust information service collects the reports from various component users and computes for each rated component a trust value (cf. [14]) stating the users' belief and disbelief in the particular component resp. their uncertainty about it. This trust value is used by the component owners to adjust the intensity of supervision by the wrappers which may reach from permanent observation via spot checks to the complete removal of a wrapper. A special trust manager component forms the link between a wrapper and the trust information service. It automates both the generation of experience reports and the control of the wrapper.

Unfortunately, the approach sketched above does not address the second security risk. On the contrary, by sending wrong negative experience reports about a component to the trust information service, a component user may easily incriminate the component and its designer. Since the trust values are not only used to control the security wrappers but also to support procurement decisions (cf. [9]), an incrimination may lead to a significant financial loss for the component designer. On the other side, a component may protect a designer of a malicious component by failing to send negative reports after detecting behavior violations.

In this paper we introduce an extension of the trust information service in order to prevent component users manipulating component trust values by issuing wrong experience reports. Now the trust information service also stores trust values of component users. The trust value of a component user is used to determine the degree, an experience report of this user is considered in computing a component's trust value. Thus, if a component user already sent false reports, a bad trust value is assigned to him and, in consequence, his experience reports are not considered for calculating the trust value of a component anymore.

In order to get decent valuations about component users, a user has to complement an experience report by a log of the events passing the interface of the evaluated component. The trust information service is supplemented by experience report checker components checking if the rating in an experience report is acknowledged by the behavior listed in the log. Moreover, by analyzing the log the experience report checker tests if the other components of the user's application fulfilled the component contract models constraining the component environment. Finally, an experience report checker tests the log for correctness. Since there are no possibilities to prevent a principal forging a log if the logged component is within the principal's control (cf. [15, 16]), the experience report checker tries to reconstruct a log by running a copy of the component in a sandbox. It performs a number of runs of the component using the inputs listed in the log. If in one of the runs the outputs from the component corresponds to the log as well, the experience report checker accepts the log as correct. Otherwise, it is rejected as possibly forged. If an experience report passes all three tests the experience report checker sends a positive rating of the component user and a negative rating otherwise. The component user's trust value is calculated from these ratings.

To avoid correctness proofs of logs by reconstructing the log events, a component user may also use a so-called witness host. Here, the component in question is not executed on the system of the component user but on a remote host and incorporated to the application by means of a telecommunication service. This witness host produces an own interface log and passes it to the trust information system instead of the component user himself. Since it is trusted by the trust information service, the log is considered to be correct and experience report checker has only to perform the two other tests.

In the sequel, we give at first a short introduction into trust management and, in particular, into the computation of trust values. Thereafter, we sketch the security wrappers and their trust-based control. Afterwards, we outline the extended trust information service. Finally, we introduce the experience report checker components and the witness hosts.

2 Trust Management and Trust Value Computation

According to Khare and Rifkin [17] the World Wide Web "will soon reflect the full complexity of trust relationships among people, computers, and organizations." The goal of trust management is to include trust relations between relevant human and computer entities in the decision which security mechanisms should be used in order to protect certain principals and their assets against malicious attacks. Jøsang [14] defines two different kinds of trust relationships reflecting interactions between humans resp. between humans and computers. He calls humans *passionate entities* and computers as well as other entities without a free will (e.g., organizations) *rational entities.* One trust relationship may exist between two passionate entities A and B. Here, A trusts B if A believes that B behaves without malicious intent. The other trust relationship considers trust of a passionate entity A in a rational entity B. Since B cannot be benevolent or malicious due to the lack of a free will, this relationship states the belief of A that B will resist any malicious manipulation caused by an external passionate entity C.

Beth et al. [18] define two different types of trust. An entity A has *direct trust* in another entity B if it believes in the benevolence of B itself. In contrast, A has *recommendation trust* in B, if it believes that B gives a reliable and honest assessment of the benevolence or nastiness of a third entity C.

Some interesting approaches of trust management exist in the field of access control. Since traditional discretionary and mandatory access control models are not adequate for large distributed systems as the Internet with a large number of fluctuating participants (cf. [19]), credential-based systems like PolicyMaker [20], REFEREE [21], KeyNote [22], and the Trust Establishment Toolkit [23] gain more and more popularity. In these systems third party entities issue credentials to principals if they have direct trust in them. If a principal wants to access a resource, it passes its credentials to the resource provider. Depending on his recommendation trust in the issuers of the credentials, the resource provider decides about granting access to the owner of the credentials.

A variant of credential-based systems are label bureaus [24]. Here, web pages are labelled in order to protect children from access to objectionable Internet sites. In contrast to the previous approaches, besides of trusted authorities (third-party labels) labels may be also issued by the web site designers (first-party) or by interested web site users (second-party).

Reputation systems (cf. [25]) are another application domain for trust management. Here, an entity rates another entity according to its experience in dealing with the counterpart. A reputation system collects the ratings and publishes them either completely or encoded in a certain scheme. The ratings provide other entities with support in deciding about the trustworthiness of the rated component. A well-known example is the feedback forum of the Internet auctioneer eBay [26] where sellers and buyers can rate each other. A more recent approach is the Feedback Collection and Reputation Rating Centre (FCRRC) [27] which is used to create reputation on parties to electronic contracts. According to Dellarocas [28], reputation systems, however, are vulnerable against attacks leading to wrong reputations of principals. In particular, sellers of a good may collude with buyers in order to get unfairly high ratings themselves or to provide other sellers with unfairly low ratings. In our system, which is also a special reputation system, we rule out this vulnerability by applying the experience report checkers (cf. Sec. 5).

In order to use trust management in practice, one has to define a measure to state various degrees of trust. In [29], Jøsang introduces trust values which are triples of three probability values. Two values state the belief resp. disbelief in an entity while the third one describes uncertainty. This third value is necessary since the knowledge of an entity may be too small to give a decent assessment. A trust value can be modeled by a so-called opinion triangle (cf. Fig. 1). Here, the belief, disbelief, and uncertainty are specified by the values b, d, and u which are real numbers between 0 and 1. Since, moreover, a trust value fulfills the constraint $b + d + u = 1$, it can be denoted by a point in the triangle. A trust value stating a high degree of uncertainty is modeled by a point close to the top

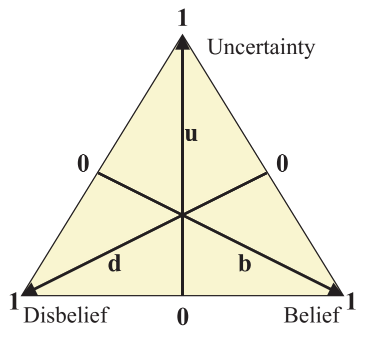

Fig. 1. Opinion Triangle (taken from [29])

of the triangle while points on the right or left bottom state great belief resp. disbelief based on large experience with the entity.

In his subjective logic [30], Jøsang extended the trust values to so-called opinions. Here, a forth probability value, the relative atomicity, was introduced. It denotes the degree of optimism or pessimism that the current uncertainty leads eventually to belief resp. disbelief. Since we do not use relative atomicity, we will apply the original trust value triples.

Trust values are used to describe both the direct trust in an entity and the trust in the recommendation of an entity about a third one. Jøsang and Knapskog introduce the following metric [31] to compute trust values from the number p of positive valuations and n of negative valuations of the entity in question:

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

The metric expresses a relatively liberal philosophy to gain trust since negative valuations can be compensated by an arbitrary number of positive assessments. Since this philosophy is probably too tolerant for some trust-management policies, we also apply the metric of Beth, Borcherding, and Klein [18]. This approach follows an unforgiving philosophy. For direct trust, the belief b is computed by the following formula v_d:

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

Basically, v_d describes the probability that the reliability of the trust in an entity (i.e., the belief b) is larger than the value α. Thus, the larger α will be selected, the lower will be the value of b. Beth's approach does not address the distinction between the disbelief d and the uncertainty u but one can calculate d and u by means of the formulas

$$d = \begin{cases} 0 & : \ n = 0 \\ 1 & : \ n > 0 \end{cases} \qquad u = \begin{cases} \alpha^p & : \ n = 0 \\ 0 & : \ n > 0 \end{cases}$$

In this metric, a single negative experience destroys the trust in an entity forever. In contrast, like in Jøsang's approach for recommendation trust negative experience can be compensated by positive valuations which is stated as follows:

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

Since, below, we do not need an explicit distinction between disbelief and uncertainty for recommendation trust, we assume that $d + u = 1 - v_r(p,n)$.

The combination of direct and recommendation trust values is addressed by Jøsang's subjective logic [30] which contains special trust combining operators. A trust value stating the direct trust in an entity x based on the recommendation of an entity r can be calculated by means of the discounting-operator $\otimes$. If the trust value $\omega_r = (b_r, d_r, u_r)$ describes the trust of oneself in the recommendations of r and $\omega_x^r = (b_x^r, d_x^r, u_x^r)$ the direct trust of r in x, the direct trust $\omega_{rx} = (b_{rx}, d_{rx}, u_{rx})$ of oneself in x based on the recommendation of r corresponds to the formula $\omega_{rx} \equiv \omega_r \otimes \omega_x^r$ where

$$\omega_r \otimes \omega_x^r \mathrel{\widehat{=}} (b_r b_x^r, b_r d_x^r, d_r + u_r + b_r u_x^r)$$

The consensus-operator $\oplus$ is used to calculate the trust value ω_x stating the direct trust in an entity x from two trust values $\omega_{r_1x} = (b_{r_1x}, d_{r_1x}, u_{r_1x})$ and $\omega_{r_2x} = (b_{r_2x}, d_{r_2x}, u_{r_2x})$ which describe the trust in x based on recommendations of two different entities r_1 and r_2. The trust value ω_x is computed by the formula $\omega_x \equiv \omega_{r_1x} \oplus \omega_{r_2x}$ and the operator $\oplus$ is defined by the formula

$$\omega_{r_1x} \oplus \omega_{r_2x} \mathrel{\widehat{=}} ((b_{r_1x}u_{r_2x} + b_{r_2x}u_{r_1x})/\kappa, (d_{r_1x}u_{r_2x} + d_{r_2x}u_{r_1x})/\kappa, (u_{r_1x}u_{r_2x})/\kappa)$$

where κ is equal to $u_{r_1x} + u_{r_2x} - u_{r_1x}u_{r_2x}$. Since the consensus-operator is commutative and associative, it can be used to calculate the trust of various recommendations.

By application of the metrics we can compute the trust of a particular component user in a component based on his experience reports. Thereafter, by using the discount-operator we can weight this trust value based on the recommendation trust value of the user. Finally, we can compute the trust value of the component from the recommendation trust-weighted assessments of all users by associative application of the consensus-operator.

3 Security Wrappers

In order to check that a component fulfills the security objectives specified in the models of its component contract, we put a security wrapper in between the component and its environment which checks all incoming and outgoing events for compliance with the behavior models in the contracts [10, 11]. Figure 2 depicts the wrapper implementation [32] for Java Beans-based component-structured systems. An adapter bean is put in between the component to be scrutinized

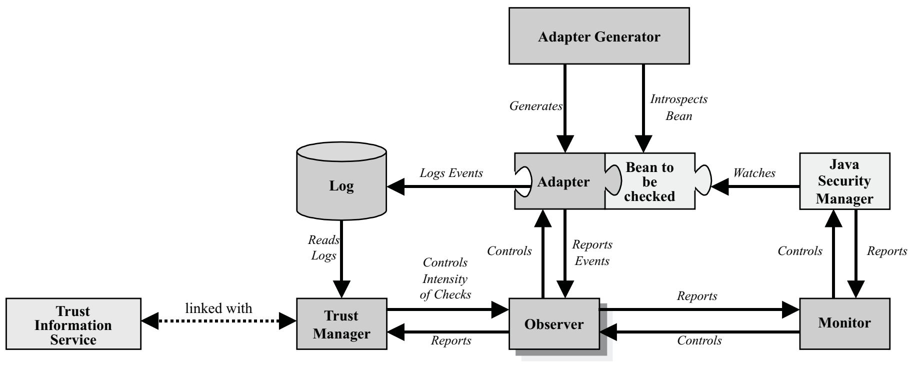

Fig. 2. Security Wrapper Architecture

and its environment. It blocks all events passing the interface temporarily and reports them to the observer beans. An observer simulates a contract model and checks if the event reported by the adapter complies with the simulated model. If the event violates the model, a report is forwarded to the monitor bean which acts as a user interface to the application administrator. Moreover, the observer notifies the adapter which seals the component until the opposite decision of the administrator. In contrast, if all observers report to the adapter that the event fulfills their models, the adapter forwards the event to its destination. Furthermore, the adapter lists all events passing the component interface in the log database in order to create a complete log of the interface behavior. The adapters are generated automatically by the adapter generator which uses the Java introspection mechanism [3] to analyze the interface of the component to be wrapped. Finally, we use also the built-in Java security manager to prevent the component using other channels than the wrapped component interface.

The intensity of the security policy enforcement is adjusted by the trust manager bean according to the current trust value of the wrapped component. The wrapped component may either be fully observed, spot checked only in times, or the supervision may be terminated[1]. As an example we introduced in [11, 13] a component-based application performing the commodity management of fast-food restaurants. In [13] we analyzed the component contracts of this application with respect to their relevance for the system security and subdivided them into three groups each containing models of a certain level of sensitivity. Afterwards, we defined for each group a wrapper management policy. According to this policy, the models of the first group describing the most sensitive security objectives have always to be fully enforced. Models of the second group may be spot checked if the belief value b of the component's trust value exceeds the value 0.999 according to the metric of Beth et al. [18] with the reliability value $\alpha = 0.99$. Moreover, if b is larger than 0.99999, the enforcement of the models

[1] A supervision should only be terminated if the belief in the component is very high since after the termination the wrapper cannot be reinstalled again.

may be terminated and corresponding observers may be removed. Finally, for the models of the third group we use the more tolerant metric of Jøsang and Knapskog [31] allowing spot checks for $b > 0.99$ and observation termination for $b > 0.999$. By application of these policies, the performance overhead of the wrapper for a trustworthy component could be reduced in steps from 5.4% to 3.2%.

Moreover, the trust manager forms the link between the security wrapper and the trust information service outlined below. In intervals, it reads the current trust value from the trust information service and adjusts the wrapper enforcement policy accordingly. Furthermore, on request of the trust information service it returns experience reports about the component interface behavior enclosed by the log of the behavior which is stored in the log database. If the component behavior complied with the contract models since transmitting the last report, a positive experience report is sent. If an observer detected a minor violation (e.g., in our commodity management application the violation of a contract model of the third group), a negative report is sent. In the case of a major violation (e.g., of contract models of the first or second group), the negative report is not sent after request from the trust information service but immediately. If the trust information service can verify the log or the component user is highly trusted (cf. Sec. 5), an alarm message is sent to all other users of the component in order to prevent harm on their systems. If the trust manager receives an alarm message caused by a negative experience report by another component user, it instructs the security wrapper to notify the application administrator and to seal the component.

4 Trust Information Service

Trust values of components and component users are computed by the trust information service. As delineated in Fig. 3, it is linked with component designers, with trust managers of user systems executing components, and with third-party certification authorities certifying components on behalf of component designers (cf. [33]). The trust information service consists of two parts in order to guarantee a high degree of privacy. A cipher service stores the registration data of components and component users and generates for each registered entity a unique cipher. The assessments of the entities are stored by the trust value manager which also computes and stores the trust values. Since, however, the assessments and the trust values are stored only by using the ciphers, neither the cipher service nor the trust value manager have full knowledge about the identities and trust values of the entities.

A component designer may register a component with the cipher service (cf. [9]). In order to avoid man-in-the-middle attacks, the components are sent accompanied by a digital signature to the cipher service. The cipher service creates a unique cipher of the component and forwards it to the trust value manager. Moreover, it also generates a digital signature based on a hash value of the component code and the cipher and sends it to the component designer.

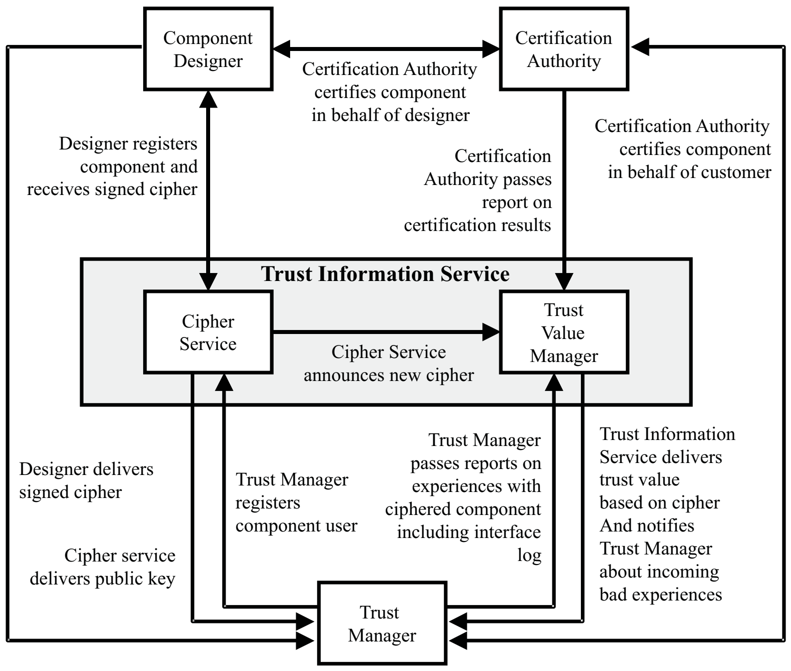
Component Designer
Certification Authority
Certification Authority certifies component in behalf of designer
Designer registers component and receives signed cipher
Certification Authority passes report on certification results
Certification Authority certifies component in behalf of customer
Trust Information Service
Cipher Service
Trust Value Manager
Cipher Service announces new cipher
Designer delivers signed cipher
Cipher service delivers public key
Trust Manager registers component user
Trust Manager passes reports on experiences with ciphered component including interface log
Trust Information Service delivers trust value based on cipher And notifies Trust Manager about incoming bad experiences
Trust Manager

Fig. 3. Trust Information Service

Thus, neither the component designer nor anybody else may afterwards alter a component without the change being detected by the component users. The component designer may hand the signed cipher over to interested component users.

Trust managers who want to use the trust information service have to register their component users, too. Here, the cipher service also generates a unique cipher which is signed and sent to the trust manager and the trust value manager. If a trust manager is interested in a trust value of a component, it sends its cipher to the trust value manager. The trust value manager returns the two trust values calculated according the two metrics introduced in Sec. 2. Moreover, a trust manager may ask for the recommendation trust value of its own component user but — to improve privacy — not of other component users. Finally, the trust manager sends experience reports about components to the trust value manager. An experience report is accompanied by the cipher and the interface log of the evaluated component as well as the cipher of the component user.

For all registered component users $u_1, \ldots, u_n$ the trust value manager stores the number of positive and negative ratings issued by the experience report checkers which will be outlined in Sec. 5. From these ratings it computes twice the recommendation trust value ω_{u_i} of each component user u_i according to both metrics introduced in Sec. 2. Moreover, it keeps for each registered component c the experience reports of all component users. Based on this information, the

Component Users	Ratings of u_i		Rec. Trust Values ω_{u_i}	Ratings of c by u_i		Dir. Trust Values $\omega_c^{u_i}$	Trust Values $\omega_{u_i c}$	Trust Value ω_c
	pos.	neg.	Jøsang	pos.	neg.	Jøsang	Jøsang	Jøsang
u_1	12	4	(.71,.23,.06)	7	0	(.88,.00,.12)	(.62,.00,.38)	
u_2	15	0	(.94,.00,.06)	8	0	(.89,.00,.11)	(.84,.00,.16)	(.84,.04,.12)
u_3	6	9	(.37,.57,.06)	2	5	(.25,.63,.12)	(.09,.23,.68)	
	pos.	neg.	Beth	pos.	neg.	Beth	Beth	Beth
u_1	12	4	(.08,.92)	7	0	(.07,.00,.93)	(.01,.00,.99)	
u_2	15	0	(.14,.86)	8	0	(.08,.00,.92)	(.01,.00,.99)	(.02,.00,.98)
u_3	6	9	(.00,1.0)	2	5	(.00,1.0,.00)	(.00,.00,1.0)	

Table 1. Example for the computation of trust values

trust value ω_c of c can be calculated for both metrics using the discounting and consensus-operators of the subjective logic [30]. At first for each metric the value $\omega_c^{u_i}$ of the trust of u_i in c is calculated. Thereafter, the trust value manager computes the trust value $\omega_{u_i c}$ stating the trust in c based on the recommendation of u_i by application of the formula $\omega_{u_i c} = \omega_{u_i} \otimes \omega_c^{u_i}$. Finally, ω_c is computed from these trust values by means of the formula $\omega_c = \omega_{u_1 c} \oplus \omega_{u_2 c} \oplus \ldots \oplus \omega_{u_n c}$.

To clarify the approach, Tab. 1 delineates an example computation of the trust value ω_c of a component c based on the experience reports of three component users u_1, u_2, and u_3 based on both metrics. The example points out that the negative rating of the component c by user u_3 has no significant influence on the trust value of c since u_3 has a very bad recommendation trust value resulting from various questionable experience reports. Moreover, the example clarifies that the metric of Beth et al. with the selected reliability $\alpha = 0.99$ is much more conservative than the one of Jøsang and Knapskog. While the belief value of ω_c according to Jøsang's metric is already relatively close to 1 indicating great belief in c, it remains very close to 0 in Beth's metric showing nearly complete uncertainty.

Besides of trust managers, the trust value manager is also interested in experience reports from trusted third party certification authorities (cf. [33]). Therefore, component designers may send certificates of their components during registration or later. Then, the cipher service asks the certification authority to send an assessment report describing the results of the certification process to the trust value manager. Since the certification process tends to be more profound (but also more expensive) than policy enforcement by security wrappers, we weight these valuations like 50 ordinary reports from component users.

Finally, the trust value manager uses the recommendation trust values of component users to decide about forwarding alarm messages. If a component user u reports a severe security violation by a component, alarm messages to other component users are only generated immediately if the belief b of u's trust value exceeds 0.9999 according to the metric of Beth et al. [18] which is reached after 917 positive valuations. Otherwise, the log passed with u's report has to be

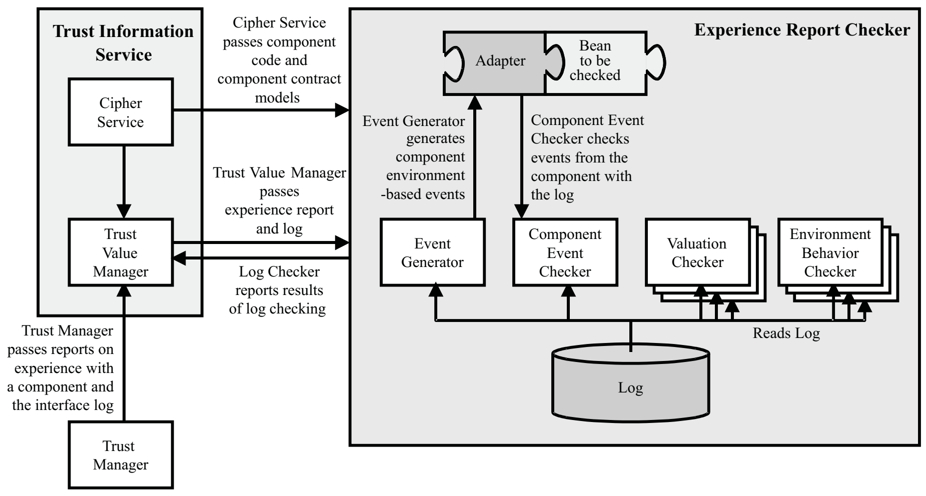
Trust Information Service
Cipher Service
Trust Value Manager
Cipher Service passes component code and component contract models
Trust Value Manager passes experience report and log
Log Checker reports results of log checking
Trust Manager passes reports on experience with a component and the interface log
Trust Manager
Experience Report Checker
Adapter
Bean to be checked
Event Generator generates component environment -based events
Component Event Checker checks events from the component with the log
Event Generator
Component Event Checker
Valuation Checker
Environment Behavior Checker
Reads Log
Log

Fig. 4. Experience Report Checker

checked by an experience report checker before warning the component users. By this policy we try to make a compromise between informing component users early about security risks and preventing wrong incriminations of component designers which would be fostered by unfounded alarm messages.

5 Component User Evaluation

The recommendation trust of component users is computed based on the correctness of their experience reports which are validated by using the logs of the events passing the component interface. An experience report checker validates an experience report by carrying out three different tests. If the tests are passed, it sends a positive rating of the component user to the trust value manager and otherwise a negative rating.

The structure of the experience report checker is delineated in Fig. 4. The valuation checkers are used to perform the first test. They check if the experience report complies with the log. Similarly to the observers in the security wrapper (cf. Sec. 3), a checker simulates a model of the component contract but, in contrast to the observers, uses the log as an input. If the component user sent a positive experience report, all valuation checkers have to state that their simulated contract models were not violated by the log. If the experience report is negative, at least one valuation checker has to detect a violation of the simulated model according to the log.

In the second test the log events triggered by the component environment have to be checked for compliance with the component contract models constraining the interface behavior. This test is necessary since a component user may easily provoke wrong behavior by inducing an environment component to send wrong events to this component. An environment behavior checker also simulates a contract model by using the events listed in the log. The test is passed if all environment checkers accept the log entries.

Finally, the third test reflects that nobody can prevent a component user to forge a log as long as the component is executed under the control of this user (cf. [15, 16]). Therefore it would be easy to change a log in a way that it complies with a wrong experience report. To detect this kind of fraud, the experience report checker tries to reconstruct the log by running the component in a sandbox environment. From the cipher service it loads the component code which due to the digital signature of the cipher service is identical with the code running on the component user's system (cf. Sec. 4). The event generator creates the events of the component environment according to the log entries and sends them to the component. The resulting events triggered by the component are forwarded to the component event checker which checks them for compliance with the events listed in the log. If an event does not correspond with the corresponding log entry, the run is discarded. We laid down that a test run may be repeated nine times in order to treat nondeterministic component behavior. The test is passed if at least one of these runs is consistent with the log.

The third test causes a severe problem. A component developer may build a nondeterministically behaving component in order to harm the reputation of component users since experience report checks of this component will often fail the third test and, in consequence, the recommendation trust values of the users will get worse. To prevent this problem, we introduced an alternative solution for proving the correctness of logs, too. It utilizes the possibility to execute a component-structured application on a distributed system. In particular, the component in question may be executed on a remote host. Here, on the application site a proxy is running instead of the component itself. The application site is linked via a network with the site executing the component and the proxy organizes the transmission of incoming and outgoing events through this link. In our Java-based solution we use the communication protocol RMI (Remote Method Invocation, [34]) to perform the transmission of the event objects. Furthermore, the log of the component interface behavior may be created not only on the application site but also on the remote host. Here, we use special witness hosts which are trusted by the trust information service. Since these trusted hosts send the logs to the trust value manager instead of the component users, the trust value manager can accept the correctness of the log without an experience report checker performing the third test. Thus, if a component user feels that he got an unjustified bad recommendation trust value since his experience reports often failed the third test, he can forward components to witness hosts. Thereafter, his experience reports will pass the tests and his trust value will recover.

The tests performed by the experience report checker rule out the collusion of component designers and users in order to manipulate trust values of components by sending wrong experience reports to the trust information service (cf. [28]). The manipulations will be detected and, in consequence, the experience reports will not be considered in the computation of component trust values anymore. Thus, we avoid the addressed security risk that components and their designers may be falsely incriminated.

Since the trust information system extension is not yet finished now, we cannot identify the exact performance requirements of the experience report checkers. We estimate the duration of the first and second tests as limited since the contract models can be simulated based on the log entries without the expenditure of running a real component. In contrast, the third test is considered more expensive, in particular, if the log reconstruction runs have to be repeated. This test, however, is only needed if no witness hosts are used. If not all experience reports can be checked in the time between two rounds of inquiring component user valuations, we skip the checks of the remaining reports. To guarantee that the most relevant checks are performed, we carry them out in the following order: At first, we check reports about severe security violations in order to send alarm messages as early as possible. Thereafter, we check the other negative experience reports since they are more relevant for wrong incriminations of component designers than positive reports. We give priority to reports from users with a low belief-value b in their recommendation trust values. Finally, we check positive experience reports where again users with a low value b are preferred.

6 Concluding Remarks

We proposed our approach for the fair trust-based enforcement of component-structured software. The amount of enforcement depends on the trust values which are computed based on reports stating the experience of running a component in question. Moreover, we check the validity of the experience reports and calculate recommendation trust values of component users based on these checks. By discounting experience reports of users with a bad recommendation trust value, we tackle the security objective of incriminating components and their designers wrongly by issuing false reports. Of course, our approach is still vulnerable against time bomb attacks where a principal behaves correctly for a while to get good ratings and thereafter carries out attacks (cf. [25]). Therefore, we recommend to define conservative security wrapper management policies in order to allow a reduction of the runtime enforcement only if time bomb behavior is not sensible anymore since correct behavior of a component leading to a good reputation renders a higher profit to the component designer than the gain based on a successful attack. This is reflected in our commodity management example [13], where we enforce the most crucial component contracts permanently.

References

1. Szyperski, C.: Component Software — Beyond Object Oriented Programming. Addison-Wesley Longman (1997)

2. Beugnard, A., Jézéquel, J.M., Plouzeau, N., Watkins, D.: Making Components Contract Aware. IEEE Computer **32** (1999) 38–45
3. Sun Microsystems: Java Beans Specification. Available via WWW: java.sun.com /beans/docs/spec.html (1998)
4. Lüer, C., Rosenblum, D.S.: WREN — An Environment for Component-Based Development. Technical Report #00-28, University of California, Irvine, Department of Information and Computer Science (2000)
5. Sun Microsystems: Enterprise Java Beans Technology — Server Component Model for the Java Platform (White Paper). Available via WWW: java.sun.com/products /ejb/white_paper.html (1998)
6. Microsoft: The Microsoft COM Technologies. Available via WWW: http://www. microsoft.com/com/comPapers.asp (1998)
7. Object Management Group: CORBA Component Model Request for Proposals (1997)
8. Lindqvist, U., Jonsson, E.: A Map of Security Risks Associated with Using COTS. IEEE Computer **31** (1998) 60–66
9. Herrmann, P.: Trust-Based Procurement Support for Software Components. In: Proceedings of the 4th International Conference on Electronic Commerce Research (ICECR-4), Dallas, ATSMA, IFIP (2001) 505–514
10. Herrmann, P., Krumm, H.: Trust-adapted enforcement of security policies in distributed component-structured applications. In: Proceedings of the 6th IEEE Symposium on Computers and Communications, Hammamet, IEEE Computer Society Press (2001) 2–8
11. Herrmann, P., Wiebusch, L., Krumm, H.: 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, Kluwer Academic Publisher (2002) 195–209
12. Fraser, T., Badger, L., Feldman, M.: Hardening COTS Software with Generic Software Wrappers. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, IEEE Computer Society Press (1999) 2–16
13. Herrmann, P.: Trust-Based Security Policy Enforcement of Software Components. In: Proceedings of the 1st Internal iTrust Workshop On Trust Management In Dynamic Open Systems, Glasgow (2002)
14. Jøsang, A.: The right type of trust for distributed systems. In: Proceedings of the UCLA Conference on New Security Paradigms Workshops, Lake Arrowhead, ACM (1996) 119–131
15. Schneier, B., Kelsey, J.: Cryptographic Support for Secure Logs on Untrusted Machines. In: Proceedings of the 7th USENIX Security Symposium, San Antonio, USENIX Press (1998) 53–62
16. Bellare, M., Yee, B.: Forward Integrity for Secure Audit Logs. Technical report, Computer Science and Engineering Department, University of California at San Diego (1997)
17. Khare, R., Rifkin, A.: Weaving a Web of Trust. World Wide Web Journal **2** (1997) 77–112
18. Beth, T., Borcherding, M., Klein, B.: Valuation of Trust in Open Networks. In: Proceedings of the European Symposium on Research in Security (ESORICS). Lecture Notes in Computer Science 875, Brighton, Springer-Verlag (1994) 3–18
19. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D.: The Role of Trust Management in Distributed Systems Security. In Vitek, J., Jensen, C., eds.: Internet Programming: Security Issues for Mobile and Distributed Objects. Springer-Verlag (1999) 185–210

20. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized Trust Management. In: Proceedings of the 17th Symposium on Security and Privacy, Oakland, IEEE (1996) 164–173
21. Chu, Y.H., Feigenbaum, J., LaMacchia, B., Resnick, P., Strauss, M.: REFEREE: Trust Management for Web Applications. World Wide Web Journal **2** (1997) 127–139
22. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.D.: The KeyNote Trust Management System, Version 2. Report RFC-2704, IETF (1999)
23. Herzberg, A., Mass, Y.: Relying Party Credentials Framework. Electronic Commerce Research Journal (2003) To appear.
24. Shepherd, M., Dhonde, A., Watters, C.: Building Trust for E-Commerce: Collaborating Label Bureaus. In Kou, W., Yesha, Y., Tan, C.J., eds.: Proceedings of the 2nd International Symposium on Electronic Commerce Technologies (ISEC'2001). LNCS 2040, Hong Kong, Springer-Verlag (2001) 42–56
25. Resnick, P., Zeckhauser, R., Friedman, E., Kuwabara, K.: Reputation Systems: Facilitating Trust in Internet Interactions. Communications of the ACM **43** (2000) 45–48
26. eBay Inc.: Feedback Forum. Available via WWW: pages.ebay.com/services/forum /feedback.html (2002)
27. Milosevic, Z., Jøsang, A., Dimitrakos, T., Patton, M.A.: Discretionary Enforcement of Electronic Contracts. In: Proceedings of the 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2002), Lausanne (2002) 39–50
28. Dellarocas, C.: Immunizing Online Reputation Reporting Systems Against Unfair Ratings and Discriminatory Behavior. In: Proceedings of the 2nd ACM Conference on Electronic Commerce (EC'00), ACM Press (2000) 150–157
29. Jøsang, A.: An Algebra for Assessing Trust in Certification Chains. In Kochmar, J., ed.: Proceedings of the Network and Distributed Systems Security Symposium (NDSS'99), The Internet Society (1999)
30. Jøsang, A.: A Logic for Uncertain Probabilities. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **9** (2001) 279–311
31. Jøsang, A., Knapskog, S.J.: A metric for trusted systems. In: Proceedings of the 21st National Security Conference, NSA (1998)
32. Mallek, A.: Sicherheit komponentenstrukturierter verteilter Systeme: Vertrauensabhngige Komponentenberwachung. Diplomarbeit, Universität Dortmund, Informatik IV, D-44221 Dortmund (2000)
33. Voas, J.: A Recipe for Certifying High Assurance Software. In: Proceedings of the 22nd International Computer Software and Application Conference (COMPSAC'98), Vienna, IEEE Computer Society Press (1998)
34. Sun Microsystems Palo Alto: Java Remote Method Invocation — Distributed Computing for Java. Available via WWW: java.sun.com/marketing/collateral /javarmi.html (1999)