ATM: An Automatic Trust Monitoring Algorithm for Service Software*

Mohammad Gias Uddin
Dept. of Electrical and Computer Engineering
Queen's University, Kingston, Ontario, Canada
gias@cs.queensu.ca

Mohammad Zulkernine
School of Computing
Queen's University, Kingston, Ontario, Canada
mzulker@cs.queensu.ca

ABSTRACT

While providing services to stakeholders, service software can be exploited by potentially untrustworthy users. Given that, it is necessary to monitor the trust relationships between service providers and requestors for potential vulnerabilities they may invite to the total system. In this paper, we propose an Automatic Trust Monitoring algorithm called ATM based on the specification of trust relationships in trust scenarios and the quantification of the relationships through trust calculation schemes. Trust rules are generated from the trust scenarios ready to be deployed at run-time. A service requestor is penalized for the violation of a trust rule and rewarded for no such violation. This analysis facilitates the quantification of the trustworthiness of service requestors and the accuracy of the recommendations from other service providers that can be used to make dynamic decisions on the corresponding requestors. The monitor is implemented in a prototype file sharing grid and evaluated using file sharing applications.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Types of Systems—*Decision support*

Keywords

Trust, Interaction, Monitoring

1. INTRODUCTION

Mutual collaboration between service providers and requestors, facilitated through the deployment of services, necessitate trustworthy interactions to achieve goals, perform tasks, and utilize resources [2]. Service software are distributed, as stakeholders are scattered across different organizational domains and inherently dynamic, as entities can join and leave the systems at any time. Uncertainty is prevalent due to this open nature, so it may not be always sufficient to use *'hard security'* mechanisms to protect services from malicious and unwanted incidents. For example, a malicious requestor may try to illegally access others' resources[1]. Such a security breach can be detected using intrusion detection and can be avoided using access control mechanisms. However, the client cannot be trusted before granting services anymore. Trust is considered as *'soft security'*, and it is "a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action." [3]. Evidence-based trust monitoring allows the analysis of interactions from trust perspectives to enable better reasoning about future interactions. By doing so, trust monitoring provides security in the traditional sense, as well as mitigates harm from future such incidents [1].

In this paper, we propose an Automatic Trust Monitoring algorithm, ATM that uses trust rules and trust calculation schemes to passively monitor service-based interactions and quantify the trustworthiness of requestors at run-time. Trust rules are generated from trust scenarios [4]. A requestor is penalized for the violation of the trust rules and rewarded for no such misbehavior. This analysis quantifies the trustworthiness of the requestor and the accuracy of the corresponding recommendations. Providers exchange recommendations to convey their trust on requestors. The algorithm is implemented and evaluated in a prototype file sharing grid.

The underlying trust monitor offers a number of benefits. First, it can be used to analyze interactions based on different trust relationships. A provider builds trust relationships with requestors based on service-based interactions. Second, the data from the monitor facilitate automatic quantification of the trustworthiness of requestors at run-time. In the field of service software, reporting service usages, misuses, and quantifying requestor-trustworthiness can trigger crucial decision making about the systems and the corresponding stakeholders, thus making the software self-protective and flexible to the changes in the environment.

The rest of the paper is organized as follows. The related trust mechanisms are discussed in Section 2. Section 3 presents trust monitoring algorithm, ATM. Section 4 discusses the implementation and experimental evaluation of our prototype monitor. The paper is concluded with a few future work in Section 5.

*This research is partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

[1]For brevity, we use provider to denote service provider software and requestor to denote service requestor or user.

2. RELATED WORK

While there are a number of work related to the monitoring of trust, they mainly focus on designing access control systems and social network-based systems. English *et al.* [1] propose a trust monitoring architecture, without incorporating any automatic trust monitoring mechanism such as trust algorithm or trust calculation schemes and trust rules. The calculation of trustworthiness is performed based on risk assessment, considering the utility of the outcomes [12]. The monitoring of service providers from trust perspectives is addressed based on fuzzy logic to select appropriate provider [11]. Fact or rule-based monitoring is proposed for service-based software [5]. Trust-based access control systems have been developed by focusing on the policy definition languages and structures [8, 10, 13–15]. Unlike our software monitoring, hardware access control is proposed in [9]. However, it is not clear how the system acquires the corresponding trust values. Moreover, an automatic trust mechanism, such as an algorithm is necessary to use the policies. Another concern is the deployment of the policy languages in the target system that requires specific engines, such as language-based parsers and compilers. We address the above mentioned shortcomings by specifying trust rules in general state-based formalisms, so that the state-information of the target system can be used without any such access control mechanisms. The above mentioned work are compared and contrasted with our work in Table 1.

Table 1: Trust-based monitoring approaches

Work	Domain	Trust Mechanism
ICTM [8]	Information system	Policy-based access control
P2PTC [9]	Trusted computing	Policy-based hardware access control
ATN [10]	Open grid system	Trust negotiation, access control
TBRM [12]	Information system	Policy-based access control
TRBAC [14]	Information system	Policy-based access control
TBAC [13]	Information system	Policy-based access control
TrustBack [15]	Information system	Role-based access control
FuzzyWeb [11]	Web services	Rule-based service access
SPM [1]	Ubiquitous system	Interaction-based system monitoring
TAP [5]	Service-based software	Policy-based software access control
Our work	Service-based software	Interaction-based service monitoring

3. ATM: THE ALGORITHM

ATM (Automatic Trust Monitoring) is used to analyze interactions, calculate trust, and provide service decisions on service request events (sRQ). We describe ATM using an example trust rule, `IllegalAccessAttempt` [4]. Suppose a file sharing server provides file open, search, upload, download services to users. Since the servers are publicly accessible, there is always the risk of clients maliciously trying to access the resource of other clients. A popular such attempt is the SQL injection attack [18] against the file search service. The attack is performed by maliciously crafting a search query, such as ";x==x" in the search field. If the server is vulnerable to this attack, it will make two queries from the input: ';' and 'x==x'. The second query always returns true, so the server might provide all the file information from its database some of which may be confidential. A client may accidentally input such query once or twice. However, if the client inputs such query beyond an acceptable limit, it surely is untrustworthy. The trust scenario is specified in Figure 1. The initial state is triggered to the *req* state, after a request for file searching is received from a user. After granting the service to the user (*i.e.,* from *req* state to *search* state), the user searches for files in the file database (`submitQuery(...)`). If the user inputs normal query (`isMalformedQuery(...)==false`), the interaction with the user is considered as satisfactory (*i.e.,* successful final state $I(S)$) . However, if the user inputs malformed query, the *alert* state is triggered, counting each such attempts (`countAttempt(...)`). For attempts beyond an acceptable limit (e.g., N), the user is considered untrustworthy and the corresponding interaction is treated as unsuccessful ($I(U)$). For attempts less than N times but more than once, the interaction remains in the alarming state ($I(A)$). The value of N is set based on the importance of the service and the impact of its misuse.

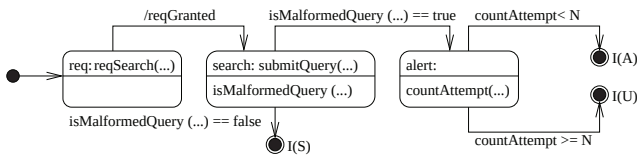

Figure 1: An example trust scenario specification: Illegal file access attempt

Calculation of trust does not take place, if the interaction remains in the alarming state. The inputs are trust rules and sRQ. The algorithm analyzes service parameters in sRQ for any misuse of services by the corresponding requestor. Based on the analysis of each sRQ, the trust values are updated. An alert is generated for a violation of a trust rule. A trust rule in ATM can have one of the two categories, belief and disbelief. A trust rule describing malicious outcome belongs to a disbelief category. In contrast, a trust rule considering favorable outcome is provided a belief category. For example, the `IllegalAccessAttempt` trust rule belongs to disbelief category. Moreover, we consider another trust rule called `SuccessfulSearchAttempt` with belief category to analyze interactions with requestors for successful search result, which will be violated for any untrustworthy search attempt from users. In ATM, it is assumed that the monitor uses five repositories for storing trust rules, alert reports, calculated trust values, recommendations, and recommendation accuracies in `Trust Rules`, `Alerts`, `Direct Trust`, `Recommendations`, and `Recom-Accuracy` respectively. Algorithm 1 presents the pseudo-code of ATM, operating in four phases: initialization of variables, analysis of sRQ, update in trust values, and notifications based on the analysis. The four phases are described in the next four subsections.

3.1 Initializations

In this phase, service parameters are initialized and trust rules are loaded into monitor knowledge base. Moreover, configuration files are examined for necessary information that are needed to analyze the corresponding interaction (*i.e.,* service request event).

The trust rules are loaded in `TRStore` ready to be deployed

input : Trust rules (R), Service request event (sRQ).
output: Update in trust values after each interaction. An alert (a) is generated for any misuse of the provided service, and a service decision is logged after the analysis of each interaction.

1 **foreach** *trust-rules* $r \in R$ *in the* `Trust Rules` *repository* **do**
2 | `TRStore`:= `TRStore` $\cup$ r;
3 **end**
4 Get serviceRequestor rID, serviceName s_i, serviceParams p, session id sID, sessionTime t from sRQ;
5 Load serviceConstraints c based on s_i from the service configuration file;
6 Load possible trust-rule names TN based on s_i from the monitor configuration file;
7 $b := 0$, $d := 0$, $n_b := 0$, $n_d := 0$, `TrustStatus`:= $Satisfactory$, `DetermineTrustVals`:= $False$;
8 **for** $i = 0$; $i < TN.length$; $i++$ **do**
9 | **if** $TN[i].category$ = *"disbelief"* **then**
10 | | $n_d = n_d + 1$;
11 | | **if** $TN[i]$.`finalState()` = $Unsuccessful$ *OR* $TN[i]$.`finalState()` = $Alarming$ **then**
12 | | | `TrustStatus`:= $Unsatisfactory$;
13 | | | **if** $TN[i]$.`isViolated()` = $Unsuccessful$ **then**
14 | | | | $d := d + (disbelief)TN[i].importance$;
15 | | | | `DetermineTrustVals`:= $True$;
16 | | | **end**
17 | | **end**
18 | **end**
19 | **else if** $TN[i].category$ = *"belief"* **then**
20 | | $n_b = n_b + 1$;
21 | | **if** $TN[i]$.`finalState()` = $Successful$ **then**
22 | | | $b := b + (belief)TN[i].importance$;
23 | | | `DetermineTrustVals`:= $True$;
24 | | **end**
25 | **end**
26 | **if** $i = TN.length - 1$ **then**
27 | | `TrustDecisionNotifier`:= $true$;
28 | | **if** `DetermineTrustVals` = $True$ **then**
29 | | | Calculate confidence μ using b, d, n_b, and n_d;
30 | | | `NotifyTrustRecomValueAnalyzer`:= $true$;
31 | | **end**
32 | **end**
33 **end**
34 **if** `NotifyTrustRecomValueAnalyzer` = $true$ **then**
35 | Get previous direct trust from the `Direct Trust` repository;
36 | Update direct trust for rID on s_i at time t;
37 | Store direct trust in the `Direct Trust` repository;
38 | Get all previous recommendations related to rID and s_i from the `Recommendations` repository;
39 | Calculate recommendation accuracy for rID on s_i at time t;
40 | Get all previous recommendation-accuracy related to rID and s_i from `Recom-Accuracy` repository;
41 | Update recommendation-accuracy for rID on s_i at time t;
42 | Store recommendation-accuracy in the `Recom-Accuracy` repository;
43 **end**
44 **if** `TrustDecisionNotifier` = $true$ **then**
45 | **if** `TrustStatus` = $Satisfactory$ **then**
46 | | Construct `sTD` using `TrustStatus` and rID, s_i, t;
47 | **else**
48 | | Construct `sTD` using `TrustStatus` and a;
49 | **end**
50 | Log `sTD`;
51 **end**
52 **function** state `finalState()`
53 state:= $Successful$;
54 **if** p *in* sRQ *violate the* c *related to* $TN[i]$ **then**
55 | Get previous alerts in `countAttempt` from `Alerts` repository;
56 | Generate alert a using rID, $TN[i]$, s_i, sID, and t;
57 | Log alert a in the `Alerts` repository;
58 | `countAttempt`:= `countAttempt` + 1;
59 | **if** `countAttempt` $\geqslant$ $AllowedAttempt$ **then** state:= $Unsuccessful$ **else** state:= $Alarming$
60 **end**
61 `Return` state;

Algorithm 1: ATM: Automatic Trust Monitoring

at run-time (Lines 1-3). The service requestor ID (rID), service name (s_i), service parameters (p), session ID (sID), and session time (t) are retrieved from the sRQ (Line 4). The service constraints (c) for s_i are loaded (Line 5), and the possible trust rule names (TN) are obtained from service configuration files (Line 6). The constraints for using the file search service could be the usage of alphanumeric characters in the search query, while the trust rules for this service are `IllegalAccessAttempt` and `SuccessfulSearchAttempt`. b and d contain the total belief and disbelief in quantified forms obtained from an sRQ respectively. n_b and n_d count the total number of trust rules with belief and disbelief categories respectively. The `TrustStatus` denotes whether any trust is violated in the sRQ. Before analyzing the event, b, d, n_b, and n_d are initialized to 0, the `TrustStatus` is set as $Satisfactory$, and the `DetermineTrustVals` is initialized as $False$ (Line 7).

3.2 Trust-Based Event Analysis

The monitor uses this phase to analyze service request events against the corresponding trust rules. Alerts are generated for the violations of the trust rules. Moreover, the corresponding requestor is penalized for any trust rule violation and rewarded for no such violation by assigning numeric values that are used to quantify the confidence of the provider on the corresponding interaction.

The service request event (sRQ) is checked against the trust rules TN to analyze the trustworthiness of rID on s_i (Lines 8-33). A trust rule in TN is denoted as $TN[i]$, where i is the index of the trust rule. Based on the analysis of sRQ, the provider computes confidence on the corresponding requestor. 'Confidence' is a quantified satisfaction of a trustor on a trustee. If a trust rule $TN[i]$ has a disbelief category, n_d is increased by 1, and the sRQ is checked against $TN[i]$ to see whether the sRQ violates it (Line 9-11). If the final state of the checking is $Alarming$ or $Unsuccessful$, `Trust Status` is updated to $Unsatisfactory$ to indicate a service misuse (Line 12). Moreover, if the final state of the checking is $Unsuccessful$, then d is updated by using the importance of $TN[i]$, and the trust values are calculated by updating `DetermineTrustVals` as $True$ (Lines 13-16). A trust rule can be assigned high, medium, or low trust value with highest belief as 1, medium as 0.8, low as 0.6, and lowest disbelief as 0, medium as 0.2, and low as 0.4. For example, if $TN[i]$ has the importance `MEDIUM` and the category is disbelief, then the corresponding trust value added to d is 0.2. If $TN[i]$ has a belief category, the sRQ is checked against it with an increase in n_b, and b is updated for no violation of $TN[i]$, with the update in `DetermineTrustVals` as $True$ (Lines 19-25). If all the trust rules in TN have been checked, the service decision is logged by setting `TrustDecisionNotifier` as $True$ (Line 27). Moreover, if the `DetermineTrustVals` is $True$, the confidence (μ) on this particular interaction is calculated (Lines 28-29). Then the trust and recommendation values are calculated and updated (Line 30).

The total belief (I_b) (range $[0, 1]$) of provider $E1$ on requestor $E2$ for service s_i at time t from interaction I (*i.e., sRQ*) is calculated using Eq. 1, where $B(rn)$ contains the belief value of the trust rule indexed as rn, and n_b is the total number of trust rules related to belief outcome. Similarly, total disbelief I_d (range $[0, 1]$) is calculated using Eq. 2, where $D(rn)$ contains the disbelief value of the trust rule indexed as rn, and n_d is the total number of trust rules with disbelief outcome. The confidence (μ) (range $[0, 1]$) of $E1$ on $E2$

about service s_i is calculated using Eq. 3, having w_b (range $[0,1]$) as the weight assigned to I_b.

$$I_b(E1,E2,s_i,t) = \frac{b(E1,E2,s_i,t)}{n_b} = \frac{\sum_{rn=0}^{n_b} B(rn)}{n_b}. \quad (1)$$

$$I_d(E1,E2,s_i,t) = \frac{d(E1,E2,s_i,t)}{n_d} = \frac{\sum_{rn=0}^{n_d} D(rn)}{n_d}. \quad (2)$$

$$\mu(E1,E2,s_i,t) = w_b I_b(E1,E2,s_i,t) + (1-w_b) I_d(E1,E2,s_i,t). \quad (3)$$

3.3 Trust Value Update

Based on the quantification of the confidence on an interaction in the previous phase (Section 3.2), the monitor further uses the confidence value to calculate the direct trust on the corresponding requestor. The direct trust value thus obtained is used to determine the accuracies of the recommendations previously used to grant the corresponding service to the requestor.

The trust values are updated upon the receipt of the confidence value (Lines 34-43). The previous direct trust is retrieved from the `Direct Trust` repository (Line 35), and then updated based on the obtained confidence value (Lines 36-37). Moreover, all the recommendations related to s_i and rID are retrieved from the `Recommendations` repository (Line 38), and their accuracies are calculated, updated, and stored in the `Recom-Accuracy` repository (Lines 39-42).

The direct trust T_D (range $[0,1]$) of $E1$ on $E2$ for service s_i at time t is calculated using Eq. 4, where δ (range $[0,1]$) is a weighting factor. The value of T_D changes after each interaction.

$$T_D(E1,E2,s_i,t) = \delta T_D(E1,E2,s_i,t-1) + (1-\delta)\mu(E1,E2,s_i,t). \quad (4)$$

The accuracy (A) (range $[0,1]$) of a recommender $E3$ in providing a recommendation to a provider $E1$ about requestor $E2$ regarding service s_i is calculated using Eq. 5, where $\Delta R(E3,E1,E2,s_i,t)$ calculates the difference between the provided recommendation and the calculated direct trust (following [7]). $R(E3,E1,E2,s_i,t)$ denotes the recommendation value provided by $E3$ to $E1$ about $E2$ regarding service s_i at time t. Each provider keeps an accuracy table (AT) in the `Recom-Accuracy` repository, where it updates the accuracy of every recommendation after the corresponding interaction. The accuracy of $E3$ to $E1$ about $E2$ regarding service s_i at time t in the AT is denoted by $AT(E3,E1,E2,s_i,t)$. The update in the AT is performed using Eq. 6 by considering previous recommendation accuracy ($AT(E3,E1,E2,s_i,t-1)$) and new recommendation accuracy ($A(E3,E1,E2,s_i,t)$). ζ (range $[1,0]$) weights the importance of previous accuracy and current accuracy of a recommender. Using Eq. 5, and 6, unreliable recommendations can be detected. A recommender is considered most reliable if it has accuracy 1 and most unreliable if it has accuracy 0.

$$A(E3,E1,E2,s_i,t) = 1 - \Delta R(E3,E1,E2,s_i,t),$$
$$here\ \Delta R(E3,E1,E2,s_i,t) = |R(E3,E1,E2,s_i,t) - T_D(E1,E2,s_i,t)|. \quad (5)$$

$$AT(E3,E1,E2,s_i,t) = \zeta AT(E3,E1,E2,s_i,t-1) + (1-\zeta)A(E3,E1,E2,s_i,t). \quad (6)$$

3.4 Trust Decision Notification

This phase notifies the target system of the status of the corresponding interaction (*i.e.,* service request event) by making suggestions whether the target system will continue or terminate the interaction with the requestor.

ATM provides trust-based service decision to the target system that the target system may use to provide service reply to the user. A service trust decision (`sTD`) is constructed based on the obtained `Trust Status` (Lines 44-51), with using either rID, s and the session time t (Lines 45-46), or the corresponding alert from the `Alerts` repository (Lines 47-48), depending on detection of misusing the provided service. The `sTD` is logged for the target system (Line 50). A service trust decision is constructed as $\{sr, s_i, Unsatisfactory, IllegalAccessAttempt, t\}$ if the requestor (sr) makes the interaction unsatisfactory by misusing a search service at time t, whereas it is $\{sr, s_i, Satisfactory, t\}$ for no such violation.

The function `finalState()` checks whether the sRQ violates the trust-rule $TN[i]$ (Lines 52-61). The final state of the checking is initialized as $Successful$ by default (Line 53). The service parameters (p) are checked against the service constraints (c) related to $TN[i]$ (Line 54). If the p does not match the c, a service misuse is detected. Then all the previous alerts corresponding to this particular service and from the same requestor are analyzed and calculated in a `countAttempt` variable (Line 55). Now `countAttempt` is incremented by 1, and an alert a is generated and logged in the `Alerts` repository (Lines 56-58). If the `countAttempt` exceeds server maximum allowed attempts, the state is updated as $Unsuccessful$, otherwise as $Alarming$ (Line 59) (recall Figure 1), returning the state as the final state (Line 61).

Based on the direct trusts and recommendation accuracies, total trusts on requestors can be calculated. A requestor can be granted a service, if its total trust for the service is at least equal to a server-allowed minimum trust value. A requestor thus always needs to maintain proper service usage records, helping the corresponding server to become self-protective against untrustworthy users.

4. IMPLEMENTATION AND EXPERIMENTAL EVALUATION

We have developed a prototype file sharing grid, having a number of file sharing servers and clients, with each file sharing server incorporating the proposed trust monitor. The service requestors can choose any service providers in the system, as well as the service providers can also interact with other providers. We evaluate our approach by analyzing the algorithm under different scenarios.

The prototype is implemented in Java. Services and trust rules are configured in XML. The repositories are developed as database tables in MySQL 5.0. The development platform is Jade 3.5, a well known Java-based agent development environment, where agents are software entities capable of interacting in decentralized systems [17]. The trust rules are generated from the most common file sharing concerns [4], focusing on three types of services: file upload, open, and search (see Table 2).

We discuss here two important cases of the algorithm. In the first case, the sRQ event has the form $\{sr_1, sp_1,$

Table 2: Trust scenarios for a file sharing server [4]

Trust scenarios	Description
File Excess	Users may upload files beyond the limit of the server and, making the upload service unavailable for others.
File Spamming	Users may upload illegal and insignificant files to waste storage space on the server.
File Harmful	Users may upload files containing malicious scripts which can harm other users.
Illegal Access Attempt	Users may try to illegally access others' personal files in the resource database by manipulating the file search service.
Remote File Inclusion	Users may manipulate file open service to execute malicious files by including them remotely outside the server sandbox.

$SearchFile$, $;x == x$, $sr1300089544370$, $t_{session}\}$, sr_1 is requestor, sp_1 is provider, $SearchFile$ is the service name, $;x == x$ is search query, $sr1300089544370$ is the session ID, and $t_{session}$ is the interaction time. We assume that such an event request is allowed only once. The `finalState()` function in the `IllegalAccessAttempt` rule returns $Unsuccessful$, so an alert is generated in the form $\{sr_1, SearchFile, IllegalAccess\ Attempt, sr1300089544370, t_{session}\}$. Then the trust values are calculated, notifying the target system of an unsatisfactory interaction. In the second case, the sRQ has the form $\{sr_1, sp_1, SearchFile, afileName, sr1300089544371, t_{session}\}$. The `finalState()` function returns $Successful$, so the `SuccessfulSearchAttempt` trust rule is satisfied, and no alert is generated.

5. SUMMARY AND FUTURE WORK

In this paper, we propose an automatic trust monitoring algorithm called ATM (Automatic Trust Monitoring) that takes as input the service-based events between a provider and a requestor and compares those events against the corresponding trust rules. Software trust relationships are described in trust rules. A requestor is penalized for a trust rule violation and rewarded for no such violation. This analysis facilitates the measurement of the trustworthiness of requestors and the accuracies of recommendations that help in making dynamic decisions. The overall approach is elaborated using examples from file sharing domains. A prototype file sharing grid is implemented that shows the effectiveness of the proposed work.

The monitoring approach, however, does not consider the session hijacking attack, where the identity of a trustworthy user is spoofed by the attacker. Moreover, our approach may be prone to the sybil and the colluding malicious node attacks, where a number of false recommenders impersonate legitimate users. Future work in this direction will be carried out to safeguard our monitoring mechanism against the above mentioned attacks.

6. REFERENCES

[1] English, C., Terzis, S., Nixon, P. Towards self-protecting ubiquitous systems: monitoring trust-based interactions, in *Personal and Ubiquitous Computing*, **10**(1); 2005. Springer: 50–54.

[2] Bichler, M., Lin, K.J. Service-oriented computing, in *IEEE Computer*, **39**(3); 2006. IEEE CS Press: 99–101.

[3] Gambetta, D. Can we trust trust?, in *Trust: Making and Breaking Cooperative Relations*, Chapter 13, 1988. University of Oxford: 213–237.

[4] Uddin, M.G., Zulkernine, M. UMLtrust: Towards developing trust-aware software, in *23rd Annual ACM Symposium on Applied Computing*, Brazil 2008. ACM Press: 831–836.

[5] Rajbhandari, S., Contes, A., Rana, O. F., Deora, V., Wootten, I. Trust assessment using provenance in service oriented applications, in *Proc of the 10th IEEE on Int Enterprise Distributed Object Computing Conf Workshops*, Hongkong, 2006. IEEE CS Press: 65–72.

[6] Uddin, M.G., Zulkernine, M., Ahamed, S.I. CAT: A context-aware trust model for open and dynamic systems, in *23rd Annual ACM Symposium on Applied Computing*, Brazil 2008. ACM Press: 2024–2029.

[7] Azzedin, F., Maheswaran, M. Trust modeling for peer-to-peer based computing systems, in *Proc of the Int Syposium on Parallel and Distributed Processing*, San Francisco, USA, 2003. IEEE CS Press: 10pp.

[8] Etalle, S., Winsborough, W. Integrity constraints in trust management,in *Proc of the 10th Symp on Access Control Models and Technologies*, Sweden 2005. ACM Press:1–10.

[9] Sandhu, R., Zhang, X. Peer-to-peer access control architecture using trusted computing technology,in *Proc of the 10th ACM Symposium on Access Control Models and Tech*, Sweden, 2005. ACM Press: 147–158.

[10] Ryutov, T., Zhou, L., Neuman, C., Foukia, N., Leithead, T., Seamons, K. Adaptive trust negotiation and access control for grids, in *Proc of the 6th IEEE/ACM Int Workshop on Grid Computing*, Washington 2005. IEEE CS Press: 55–62.

[11] Sherchan, W., Loke, S., Krishnaswamy, S. A fuzzy model for reasoning about reputation in web services, in *Proc of 21st Annual ACM Symposium on Applied Computing*, Dijon 2006. ACM Press: 1886–1892.

[12] Lin, C., Varadharajan, V. Trust based risk management for distributed system security - a new approach, in *Proc of the 1st Int Conf on Availability, Reliability and Security*, Vienna, Austria, 2006. IEEE CS Press: 6–13.

[13] Dimmock, N., Belokosztolszki, A., Eyers, D., Bacon, J., Ingram, D., Moody, K. Using trust and risk in role-based access control policies, in *Proc of the 9th ACM Symp on Access control models and technologies*, New York, USA, 2004. ACM Press: 156–162.

[14] Dimmock, N., Bacon, J., Ingram, D., Moody, K. Risk models for trust-based access control (TBAC), in *Proc of the 3rd Annual Conf on Trust Management (LNCS v3477)*, 2005. Springer: 364–371.

[15] Chakraborty, S., Ray, I. TrustBAC: Integrating trust relationships into the RBAC model for access control in open systems, in *Proc of the 11th ACM Symposium on Access Control Models and Technologies*, California, USA, 2006. ACM Press: 49–58.

[16] Rochford, K., Coghlan, B., Walsh, J. An agent-based approach to grid service monitoring, in *Proc of the 5th Int Symposium on Parallel and Distributed Computing*, Timisoara 2006. IEEE CS Press: 345–351.

[17] Bellifemine, F., Caire, G., Poggi, A., Rimassa, G. Jade: A white paper, in *EXP in Search of Innovation*, **3**(3); 2003.

[18] Snyder, C., Southwell, M. Preventing SQL injection, in *Pro PHP Security*, SpringerLink: 249-261.