Trust Management in Monitoring Financial Critical Information Infrastructures

Giorgia Lodi[1], Roberto Baldoni[1], Hisain Elshaafi[2], Barry P. Mulcahy[2], György Csertán[3], and László Gönczy[3]

[1] University of Rome La Sapienza, Italy
{lodi,baldoni}@dis.uniroma1.it
[2] Waterford Institute of Technology, Ireland
{helshaafi,bmulcahy}@tssg.org
[3] OptXware Research&Development Ltd, Hungary
{csertan,gonczy}@optxware.com

Abstract. The success of Internet-based attacks and frauds targeting financial institutions highlights their inadequacy when facing such threats in isolation. Financial players need to coordinate their efforts by sharing and correlating suspicious activities occurring at multiple, geographically distributed sites. CoMiFin, an European project, is developing a collaborative security framework, on top of the Internet, centered on the Semantic Room abstraction. This abstraction allows financial institutions to share and process high volumes of events concerning massive threats (e.g., Distributed Denial of Service) in a private and secure way. Due to the sensitive nature of the information flowing in Semantic Rooms, and the privacy and security requirements then required, mechanisms ensuring mutual trust among Semantic Room members (potentially competitive financial players) must be provided. This paper focuses on the design and preliminary implementation of a trust management architecture that can be configured with trust and reputation policies and deployed in Semantic Rooms.

Keywords: Financial critical infrastructures, collaborative environment, trust, reputation, monitoring, trust metrics.

1 Introduction

Context and Motivations. An increasing amount of financial services are provided over publicly accessible communication mediums such as the Internet. As a result, those services and their supporting IT infrastructures are exposed to a variety of coordinated and massive Internet-based attacks and frauds [1], [2], [3], [4] that cannot be effectively handled by any single financial organization. Therefore, financial institutions need to coordinate their efforts in order to offer the computational power and collective information assets that are capable of discovering and containing those threats [5]. Let us consider as an example large-scale stealthy scans. An attacker performs such scans against financial institutions with the aim of capturing sensitive information on edge sites of the

P. Chatzimisios et al. (Eds.): MobiLight 2010, LNICST 45, pp. 427–439, 2010.

target institutions, this typically includes reachability and status of specific ports and IP addresses [6]. The outcome of the scan is to create a list of potentially vulnerable hosts. This list can then be exploited by the attacker to compromise such hosts and, as an example, deny the access to hosts' services from customers and/or internal users. Stealthy scans are becoming increasingly sophisticated: they are typically initiated from many geographically dispersed locations simultaneously and target multiple hosts scanned at random [7]. Hence, it is likely that single existing detection systems (e.g., Intrusion Detection Systems), deployed in isolation at individual financial institutions, are incapable of discovering a sufficient number of scans within a predefined time window to identify these events as an attack. Collaborative security environments are, thus, required in order to detect such attacks, as they can be effectively deployed to aggregate and correlate a higher amount of data, coming from multiple financial sources, that collectively show evidence of an attack.

In the context of the EU funded project CoMiFin [8] we are developing a framework that enables the construction of such collaborative security environments centered on the *Semantic Room (SR)* abstraction. An SR allows financial institutions to customize private spaces on top of the Internet for secure data sharing and data processing. SRs have a specific strategic objective to fulfill (e.g., an SR for stealthy scans detection) and are regulated by contracts that specify the set of rights and obligations to be met for being part of them (e.g., security and privacy requirements). Such sharing of information raises trust issues with respect to the information flowing in the SR. Indeed, in this setting, there can exist different types of SRs with different levels of trust requirements. At one extreme there could be SRs formed by financial institutions that trust each other implicitly (e.g., branches of the same bank), and consequently trust the information being processed and shared in those SRs. At the other extreme, there could be SRs whose membership includes participants that are potential competitors in the financial market. In this case, the issue of trusting the information circulating in a semantic room becomes a point of great importance and if this issue is not adequately addressed, the semantic room abstraction will be infeasible as financial institutions will refrain from becoming members of it. Proper mechanisms that ensure a measurable level of trust among SR participants are therefore required.

Contribution. In this paper, we propose the design and preliminary implementation of a trust management architecture that is capable of specifying and communicating trust metrics among distributed SR financial institutions. An example of a trust metric we evaluate is the reputation value associated with each financial institution belonging to an SR. This measure reflects the perceived competence, integrity, and/or behavioural aspects of each financial institution that is providing information to an SR. The reputation value depends on the configuration of the trust algorithm for that metric and the result can be used to weight the quality of information provided thereby increasing or decreasing its significance. The architecture is able to dynamically update the level of trust attributed to participants according to monitored participants' behaviours,

undertaken in SRs; this includes compliance with the requirements specified in SR contracts. The design of the architecture is highly flexible and includes the ability to apply multiple trust update policies to SR information flows. In this paper we present a specific policy that dynamically computes at run time the reputation of SR participants based on their monitored behaviours and their reputation in other SRs to which they may belong.

Related work. The need for collaborative systems for coping with the current generation of threats and security attacks is highlighted in a number of works that can be found in the literature (e.g., [5][9][10]). In addition, the need to properly deal with trust in distributed environments in which many distrusting entities can collaborate is emphasized in some recent works (e.g., [11][12]). This highlights just how important it is to ensure a certain level of trust among collaborative entities, especially in contexts where sensitive information must be shared, as in our case.

Reputation management, that is, the combination of first hand experience with third party recommendations to create a feedback loop for determining trustworthiness, has proven to be an integral part of modern trust management systems [13], [14], [15] [16], [17], [18]. These works share a number of similarities with our approach. In all cases, a feedback-based mechanism is utilised in order to assess the reputation of participants in the system. In particular, a local reputation value is used that is analogous to our local history function as described in Section 3.2. In addition, all the works above are applied to P2P environments when computing the reputation of peers for file sharing purposes. Typically, in these P2P systems contracts are not used to regulate relationships among peers; the system is "flat" and the reputation is not controlled and differentiated on the basis of the type and requirements as contractually specified, as in our case.

The rest of this paper is organized as follows. Section 2 presents the Semantic Room abstraction where the trust management architecture can be used. Section 3 describes the design of the trust management architecture. In particular, this Section explains how monitoring SR contracts can impact the level of trust within SRs. Section 4 presents a premilinary implementation of our trust management architecture and finally Section 5 provides the conclusions of the paper.

2 The Semantic Room Abstraction

A Semantic Room is a federation of financial institutions that wish to be grouped together for the sake of information processing and sharing. The partners participating in a specific SR are referred to as the *members* of the SR.

Each SR is associated with a *contract* that defines the set of processing and data sharing services provided by that SR along with the data protection, isolation, trust, security, dependability, and performance requirements. The contract also contains the hardware and software requirements a member has to provision in order to be admitted into the SR. In Figure 1 SR members provide raw data to the SR(s) that they are part of. This data may include real-time data, inputs

from human beings, stored data (e.g., historical data), queries, and other types of dynamic and/or static content. Raw data is processed internally by each SR.

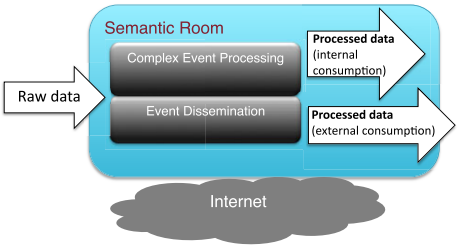

Fig. 1. Semantic Room abstraction

The resultant processed data can be used for internal consumption within the SR: in this case, derived events, models, profiles, blacklists, alerts and query results can be fed back into the SR so that the members can take advantage of the intelligence provided by the processing. SR members can, for instance, use these data to properly instruct their local security protection mechanisms in order to trigger informed and timely reactions independently of the SR management. In addition, a (possibly post-processed) subset of data can be offered for external consumption. SR members have full access to both the raw data that the members agreed to contribute by contract, and the data being processed and thus output by the SR. Data processing and results dissemination is carried out by SR members based on the contractual obligations and restricts specified in the SR contract.

In addition to the SR members, there can exist clients of the SR. These clients cannot contribute raw data directly to the SR; however they can be consumers of the processed data that the SR is willing to make available for external consumption (Figure 1).

SRs can communicate with each other. In particular, processed data produced by an SR can be injected into another SR (e.g., through the SR client role described above) as regulated by its contract. The ability for SRs to communicate with one another may enable a composition of multiple services, provided by each individual SR, into higher-level functionalities. For instance, processed data in the SR related to "DDoS attacks on FIs" can be used by a more specialized SR such as "DDoS attacks on banks in a country" whose data can be, in turn, used by the SR related to "DDoS attacks on a specific bank in a specific country" in order to provide partners with richer services (Figure 2).

Figure 2 depicts a hierarchy of SRs in which a member (in the example bank1_Agency1) belongs to more than one SR. The hierarchy structure enabled by the SR abstraction can be effectively used in order to control the level of privacy and security provided by the SR for the information being accessed, shared, and processed inside it. Specifically, in "private" SRs where only members of the same organization participate (in Figure 2 the leaf nodes in the hierarchy) sensitive information, possibly concerning financial transactions, might be fully visible and accessible to the members of that SR. In this case, members trust each other as they belong to the same organization and mutual trust is inherently ensured. In contrast, in large and heterogeneous SRs (e.g., in Figure 2 the "DDoS attacks to FIs" SR), the level of privacy and security specified in the SR contract (in Figure 2 the SLA associated with each SR), and required by FIs for the exchanged and processed information can be significantly different from that

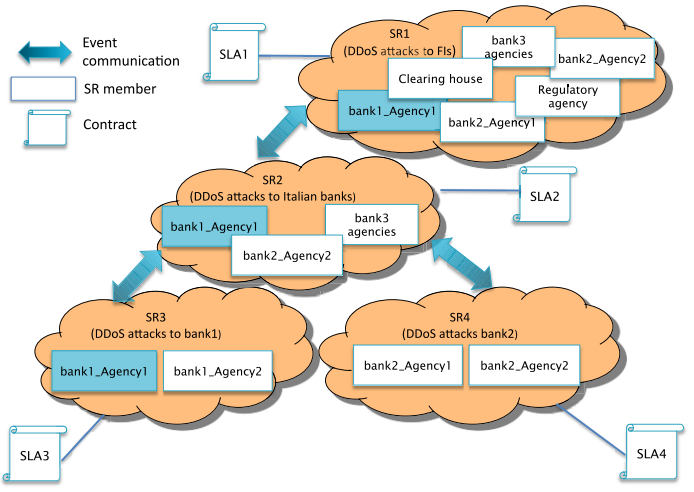

Fig. 2. SR hierarchy and communication

required in the leaf node SRs. This is due to the fact that in large SRs FIs can be potential competitors in the financial market. Competitor FIs may decide to federate despite this in order to take advantage of the intelligence produced by the SR; however, they do require more stringent requirements on the SR members' behaviours before making available their own data to the SR. In these large scale SRs with distrusting members, it is necessary to support the information processing and sharing while guaranteeing mutual trust among members (i.e., while ensuring information trustworthiness). This requirement can be crucial: without any trust guarantees, these SRs might run the risk of not starting up as distrusting participants can be reluctant to provide their own data to them.

The architecture we introduce in the next section is designed so as to meet such trust requirements including an incentive scheme that promotes constructive participation of members.

3 The Trust Management Architecture

We have designed a trust management architecture which stores, monitors, and updates trust levels of SR members based on the behaviours of those members.

Each SR deploys its own Trust Manager (TM) that, in order to monitor those behaviours, interacts with other two main subsystems we have included in our design; namely, the Event Processing and Metrics Monitoring subsystems, and possibly with other TMs deployed in other SRs.

The TM design is highly flexible to include different types of configurations. Authorized administrators can configure (through a web-based interface) a number of parameters related to the way in which the level of trust is computed. In particular, as illustrated in the use case diagram in Figure 3, administrators can set the preferred update policy of the trust management architecture. This

policy can be obtained either from an available set of policies (e.g., a policy repository) or from the SR contract where requirements on how the trust is to be computed can be specified (see below). The administrator can also specify the values of constant parameters such as default trust for new members, and trust threshold to be used in order to identify when interested parties are to be notified of changes to a member's trust (note that, also for these parameters, administrators can properly configure the trust management architecture using the value that might be included in SR contracts).

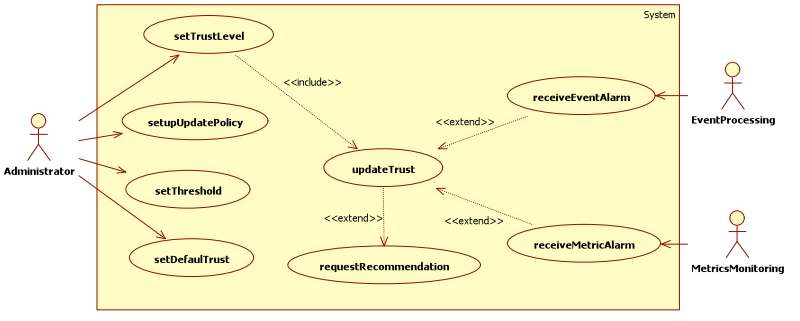

Fig. 3. Trust Management: Main Use Case

Administrators can use those configurations to specify how the initial trust value of each member is calculated and to control how these levels of trust are dynamically computed and updated at run time.

The Event Processing subsystem of an SR interacts with its related TM by sending alarms related to an event of a member that may trigger trust changes. Analogously, the Metrics & Monitoring (MeMo) subsystem of an SR sends alarms to its related TM in case a change in the SR metrics evaluation (e.g. SLA violation/adherence) for a member occurs, as this change can affect other SR members' trust in that member. Additionally, the Event Processing subsystem uses notifications of the trust levels from the TM in order to filter the events based on the sources of those events. For example, events originating from a member with a low trust level might be ignored or assigned to a low priority. Since trust itself is one of the metrics that the system monitors, the MeMo subsystem receives trust updates so that it can monitor the adherence to the SR contract's trust requirements in the same way as it does for the other SR contract metrics such as security, performance, dependability, etc.

Finally, note that it may be possible for the trust management architecture to calculate the trust for a new SR member by asking recommendations related to that member to other SR's TMs. As depicted in Figure 2 a member can belong to more than one SR and the interactions among TMs of different SRs can be required in order to obtain information on the trust levels of that member in different SRs. In general, this approach helps to build reliable trust information on the members by considering their behaviours in many SR collaborative environments.

3.1 SR Contract Monitoring for Trust Management

As previously highlighted trust is a crucial requirement in both the design and operation phases. Trust management in the operation phase utilizes a control algorithm that takes decisions based on trust metric values, which are provided by a metrics monitoring system of the SRs.

Figure 4 shows the general metrics monitoring architecture used in Semantic Rooms. The purpose of the Metrics & Monitoring (MeMo) is to continuously measure and present the status information of services and resources and to generate notification for other components in case of undesired status changes (e.g. SR contract violations).

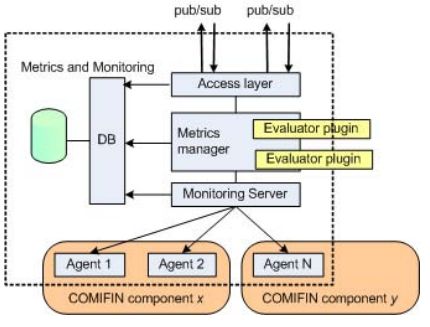

Fig. 4. MeMo architecture

Trustworthiness monitoring can refer to various levels of a system architecture. Typically, high level trust metrics are composed of low-level indicators (number of false alert messages, number of viruses found, number of lost messages, etc.). These can be measured by typical system monitoring tools (Monitoring Server), such as Nagios [19] or Tivoli Monitoring [20], which have good tool support for measurement of IT component parameters by using low-level sensors or agents (Agent 1..N); however, no sophisticated evaluation methods are offered. Therefore, a plugin-based monitoring architecture is designed which allows for plugging-in mathematical tools, rule engines, Complex Event Processing facilities or even other system components such as the Trust management architecture to be used as sophisticated metrics evaluator in the system. Our purpose is to separate SR specific evaluation (e.g. detection of domain-specific DDoS attacks) from infrastructure level monitoring.

Resource level monitoring settings (i.e. placement and configuration of sensors/agents) are generated on the basis of the SR contracts. As the resource pool (and therefore the monitoring requirements) of an SR may change frequently, the monitoring configuration is maintained in a model-based way to adapt sensor/agent setups quickly.

In a *plugin-based monitoring architecture* the TM is used as a plug-in with enhanced trust calculation features. The trust maintenance algorithm will be implemented in the TM while MeMo keeps track of system status information both for low-level security metrics and trust values computed by the TM.

MeMo offers both a *query functionality* and a *publish/subscribe middleware interface* to forward measurement data. This supports a scalable, distributed management architecture for SRs, e.g., security alerts are available for both the components managing the SR and the TM which updates trust metrics.

3.2 Pluggable Trust Policies: A Reputation Framework

We have designed the trust management architecture in order to be configured with different trust policies. In this paper, we focus on a policy that allows us to update the reputation of an SR member at run time. Specifically, we consider here the distributed system depicted in Figure 2 in which an SR member can be involved in more than one SR. As stated above, SRs are regulated by contracts, which we call SLAs from now on; each SLA defines different QoS requirements for which SR members are willing to pay. These requirements can include in which way reputation of members can be computed. In particular, SLAs might specify an initial reputation value (see below) and the functions that are used to compute and update it according to the behaviours of SR members within SRs.

The reputation defined into an SLA can be used as input for other SLAs, thus enabling links among SLAs and, consequently among SRs. The binding among SR SLAs permits different SR members to take advantage of the knowledge on a specific member reputation in other SRs in order to enforce their trust in it and build, as previously mentioned, reliable levels of trust of the members. This knowledge can be crucial in some cases as it might be used for also determining how fast or slow can the reputation judgement towards that SR member be changed. For example, if an SR member does not always behave correctly in the eyes of the other members in the same SR, its reputation in other SRs can count more and influence the final reputation evaluation.

Owing to this scenario, our reputation policy works as follows. It uses the trust parameters included in the SR SLA (e.g., the rules for determining the reputation). Thus, let *sr* be an SR member involved in more than one SR; we claim that in general its reputation is a function that can be defined as in 1:

$$R_{sr_j} = f(history_j, \{R_{sr_i}, ..., R_{sr_k}\} \setminus \{R_{sr_j}\}) \quad (1)$$
$$\text{where} \quad i, j, k \in \{SR1, SR2, SR3, ...\}$$

To compute the reputation, two principal elements are taken into account: (i) the behavior of *sr* in an SR; we call it *local history* i.e., the evaluation of the *sr*'s actions carried out in the SR, (ii) and the reputation that *sr* has in other SRs in case bindings among SLAs are enabled. Note that, if more than one bind exists with other SRs, the reputation function above is able to take into consideration all the reputation functions derived by the linked SRs (what we have called above recommendations).

The local history is a function that is computed, as in case of 1 above, by our policy as follows:

$$history_j = g(init\_rep\_value, [facts_1(sr), ..., facts_k(sr)]) \quad (2)$$
$$\text{where} \quad j \in \{SR1, SR2, SR3, ...\}$$

In other words, the local history is defined by considering an initial reputation value associated with an SR member, and an evaluation of the actions performed by *sr* over the entire duration of the SLA (we call these evaluations *facts* in

formula 2). These facts are the input of a feedback-based mechanism through which other SR members belonging to the SR can judge the *sr*'s behavior based on the facts they have observed in that SR. Note that the facts are determined on the basis of the SR contract monitoring carried out by the MeMo subsystem. The result of the monitoring triggers interactions between MeMo and TM so that TM can determine accordingly the behaviours of an SR member.

The initial reputation is a value in the range of [0,1] and represents the initial credibility of the member in the eyes of the other members. This credibility can be influenced by many factors: a previous business experience with the member, the type and the quality of service the member supply to the SR. For instance, we might think that some SR members have a previous long running business relationship with a specific member with which are part of an SR; in this case their initial reputation judgement towards that member can be high; that is, close to 1; in contrast, there can be other SR members that do not have any previous business experience with that member so that they can be reluctant to consider it credible, and wish to evaluate its behavior in the SR in order to define its reputation.

The initial reputation value is then dynamically adjusted at run time by the local history function according to the judgements the SR members express on the specific *sr*'s actions.

4 Trust Manager Design and Implementation

Figure 5 depicts the class diagram of the TM and the subsystems with which it interacts. This preliminary design is based on the architecture described above.

Trust is an object that has a number of attributes including the trust level or the score of the member, the recency, and the confidence of the trust. The recency indicates how recent the trust level is. This allows trust services consumers to determine how reliable the trust level is, and it is useful for update operations as older trust values can be considered less important. Trust confidence ranges between 0 and 1. It is calculated based on the source of the trust value i.e. based on the experience or local history (more confidence) vs. recommendations (less confidence) and the number of trust updates. The more trust is updated, the more the confidence value.

MeMo and Event Processing subsystems subscribe to updates on the trust values and the TM receives metrics and events trust alarms from MeMo and Event Processing, respectively, that it uses in order to update members' trust levels.

The sequence diagram in Figure 6 depicts the use case resulting from an interaction with the MeMo subsystem described above. Specifically, the MeMo sends an alarm to the TM. The alarm triggers a trust update: existing trust is loaded and the configured update policy can be applied in order to update the trust.

If we refer to the pluggable policy described in Section 3.2 the trust can be updated by two f and g functions we have used in our current implementation.

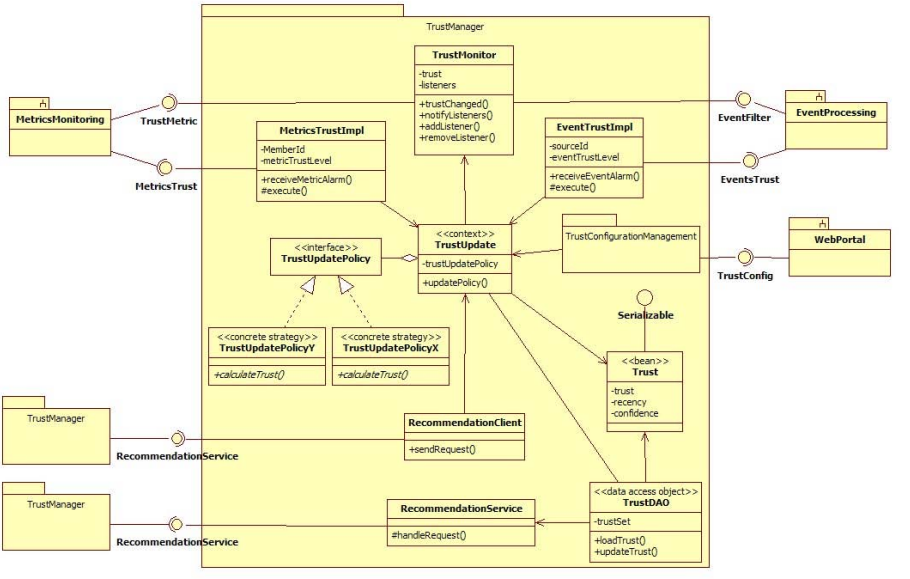

Fig. 5. Trust Management Architecture: Class Diagram

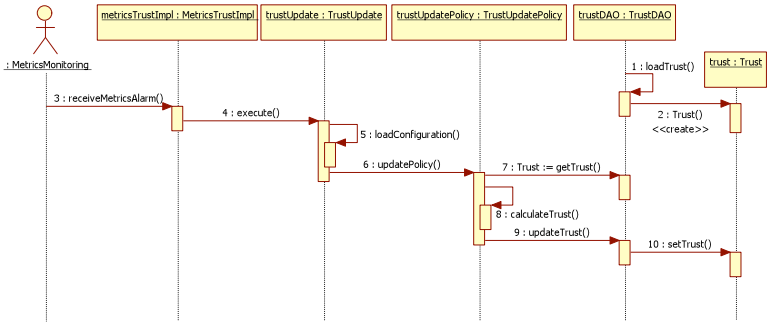

Fig. 6. Trust Management Architecture: Sequence Diagram

Specifically, when a member in SR x interacts with a another specific SR member in the same SR its reputation is computed using the following function:

$$f_x = \frac{\sum_{i=1}^{n} R_i \cdot T_{i,x} + R_x}{n+1} \tag{3}$$

where R_i represents the reputation value provided by SR i, $T_{i,x}$ is the reputation of SR i with respect to SR members in SR x, R_x is the current reputation of the specific member with respect to the other members and, finally, n is the number of reputation values read from other SRs.

The value returned by the function is the new reputation computed after each interaction with the member. Note that, since we assume interactions to be deterministic (e.g. we assume that all interactions with the member happens with the same order for all the other members in an SR) and the method used to calculate f is deterministic, all members in SR x agree on the reputation value toward the specific SR member.

After some interactions we suppose that administrators at each member can check through a set of facts the exact behaviour of a specific SR member. This can be modelled by defining the function g. In our case, g is defined in such a way that the current reputation value for a member is modified after each verification and increased if the verification confirms the expectations ($R_x \geq 0.5$ and correct behaviour or $R_x < 0.5$ and bad behaviour), and reduced otherwise. The same type of update is executed on the reputation values for other SRs too: if the reputation value read from an SR confirms the verified behaviour of the specific SR member, the corresponding reputation value is increased, otherwise it is decreased. More specifically we used the following function:

$$g = (-1)^s \cdot K \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-0.5)^2}{2\sigma^2}} \tag{4}$$

where K is a constant real multiplicative factor, σ is the standard deviation and s is parameter that has value 0 if the verification confirms the reputation value or value 1 otherwise (we assumed $K = 0.005$ and $\sigma = 0.15$). The rationale behind this formula is that reputation values in the middle of the range are not useful to take decisions; therefore, the formula is designed to quickly polarize the reputation value either towards a large or small value and then maintain the reputation in this state despite possible sources of fluctuations.

Using these functions we have also conducted an experimental evaluation in order to assess the effectiveness of our approach. For the sake of brevity, we do not include these results in this paper; interested readers can refer to [21] for detailed results.

5 Conclusions

In this paper, we have introduced the SR abstraction that allows financial institutions to share and process high volumes of events concerning massive threats (e.g., stealthy scans, Distributed Denial of Service) in a private and secure way. This collaborative event processing facilitates the correlation of suspicious events that would otherwise be overlooked by isolated institutions. The identification and prediction of those threats can thus trigger timely local reactions. Such sharing of sensitive information raises trust issues with respect to the information flowing in the SR. Robust mechanisms that ensure a measurable level of trust among distrusting SR participants are therefore required. To this end, we have proposed a Trust Management architecture that monitors, stores, and updates trust levels of SR members based on the behaviours of those members. In doing so, it interacts with both the Event Processing subsystem, in order to evaluate the events generated by the members, and the MeMo subsystem, in order

to dynamically update the level of trust. Monitoring of SR contract rights and obligations allows members adherence to SR contract requirements (including trust thresholds) to be evaluated and incorporated into the trust feedback loop. The TM design is highly flexible and allows for different trust update policies. In this paper, we have described the design and a preliminary implementation of a policy that computes and manages at run time the reputation of SR members based on their behaviours in the SRs and their reputation in other SRs that they may belong to.

Both the design and implementation of our TM and the other subsystems are still preliminary: we are currently working in the context of the EU funded project CoMiFin in order to enhance and extend them and evaluate them in more complex scenarios (e.g., asynchronous settings and failures in the collaborative SR environment).

Acknowledgements

This research is partially funded by the EU project CoMiFin (Communication Middleware for Monitoring Financial Critical Infrastructure). The authors wish to thank their project colleagues for their comments on the described design and implementation.

References

1. Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., Weaver, N.: Inside the Slammer Worm. IEEE Security and Privacy 1, 33–39 (2003)
2. DDoS: National Australia Bank it by DDoS attack. http://www.zdnet.com.au/news/security/soa/National-Australia-Bank-hit-by-DDoS-attack/0,130061744,339271790,00.htm (2010)
3. DDoS: Update: Credit card firm hit by DDoS attack, http://www.computerworld.com/securitytopics/security/story/0,10801,96099,00.html (2010)
4. Fraud: FBI investigates 9 Million ATM scam (2009), http://www.myfoxny.com/dpp/news/090202_FBI_Investigates_9_Million_ATM_Scam
5. Locasto, M.E., Parekh, J.J., Keromytis, A.D., Stolfo, S.J.: Towards collaborative security and p2p intrusion detection. In: IEEE Workshop on Information Assurance and Security. United States Military Academy, West Point (2005)
6. Staniford, S., Hoagland, J.A., McAlerney, J.M.: Practical automated detection of stealthy portscans. Journal of Computer Security 10, 105–136 (2002)
7. Zhou, C.V., Leckie, C., Karunasekera, S.: A survey of coordinated attacks and collaborative intrusion detection. Computer and Security 29, 124–140 (2010)
8. CoMiFin: CoMiFin - Communication Middleware for Monitoring Financial Critical Infrastructures (2010), http://www.comifin.eu
9. Krügel, C., Toth, T., Kerer, C.: Decentralized event correlation for intrusion detection. In: Kim, K.-c. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 114–131. Springer, Heidelberg (2002)
10. Xie, Y., Sekar, V., Reiter, M.K., Zhang, H.: Forensic analysis for epidemic attacks in federated networks. In: ICNP, pp. 43–53 (2006)

11. Cachin, C., Keidar, I., Shraer, A.: Trusting the cloud. SIGACT News 40, 81–86 (2009)
12. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Technical report, University of California, Berkeley (2009)
13. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: WWW 2003: Proceedings of the 12th international conference on World Wide Web, pp. 640–651. ACM, New York (2003)
14. Sun, L., Jiao, L., Wang, Y., Cheng, S., Wang, W.: An adaptive group-based reputation system in peer-to-peer networks. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 651–659. Springer, Heidelberg (2005)
15. Huynh, T.D., Jennings, N.R., Shadbolt, N.R.: An integrated trust and reputation model for open multi-agent systems. Autonomous Agents and Multi-Agent Systems 13, 119–154 (2006)
16. Gupta, M., Judge, P., Ammar, M.: A reputation system for peer-to-peer networks. In: NOSSDAV 2003: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video, pp. 144–152. ACM, New York (2003)
17. Zhu, Y., Shen, H.: Trustcode: P2p reputation-based trust management using network coding. In: Sadayappan, P., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2008. LNCS, vol. 5374, pp. 378–389. Springer, Heidelberg (2008)
18. Bachrach, Y., Parnes, A., Procaccia, A.D., Rosenschein, J.S.: Gossip-based aggregation of trust in decentralized reputation systems. Autonomous Agents and Multi-Agent Systems 19, 153–172 (2009)
19. Nagios: Nagios (2010), `http://www.nagios.org`
20. Tivoli: IBM Tivoli Monitoring (2010), `http://www-01.ibm.com/software/tivoli/products/monitor/`
21. Baldoni, R., Doria, L., Lodi, G., Querzoni, L.: Managing reputation in contract-based distributed systems. In: OTM Conferences (1), pp. 760–772 (2009)