Scalable Evaluation of Distributed On-line Network Monitoring for Behavioral Feedback in Trust Management

[1] Jorge López <jorge.eleazar.lopez_coronado@telecom-sudparis.eu>
[1] Stephane Maag <stephane.maag@telecom-sudparis.eu>
[2] Gerardo Morales <gmorales@galileo.edu>
[1] Institut Mines Telecom, Telecom SudParis CNRS UMR 5157,
9 rue Charles Fourier, Évry, Île-de-France, 91000, France
[2] RLICT Universidad Galileo,
7a. Av. Final, Calle Dr. Eduardo Suger Cofiño, Zona 10, Guatemala, Guatemala

Abstract. Collaborative systems are growing in use and popularity. The need to boost the methods concerning the interoperability is growing as well; therefore, trustworthy interactions of the different systems are a priority. The decision regarding with whom and how to interact with other users or applications depends on each system. We focus on providing trust verdicts by evaluating the behaviors of different agents, using distributed on-line network monitoring. This will provide trust management systems information regarding a trustee experience, for those trust management systems based on "soft trust". In this work, we propose a scalable evaluation method for any on-line network monitoring system, by using an auxiliary model, an extended finite state automaton (EFSA), and as well as other known methods to reduce the time complexity of the evaluation algorithm.

Keywords: trust management; on-line network monitoring; scalable evaluation;

1. Introduction

Internet applications have become one of the most popular ways to socially interact, make commerce, and create collaborative work; making them a daily part of our living. With time, the collaborative aspects supported by Internet have evolved bringing new tools, methodologies and concepts. These systems keep growing in use and in popularity. The need to boost the interoperability methods related to them is growing as well; making thus trustworthy interactions of the different systems a priority.

These concepts of trust have been brought to computer science. The systems need to interact with users and with other applications. The decision regarding with whom and how to interact with other users or applications depend on each application or system. There are many definitions of trust in the literature, but the one we adopt

here is the one commonly applied and defined in [1], "the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context". From this definition different types of trust management engines have been created.

Some trust management systems use security policies and authentication in order to provide the concept of trust. In these types of systems, to determine the entity called trustee, it implies there is a related authentication mechanism. The policy languages are used to express the actions allowed for each trustee. This is called ``hard trust'' because the actions can only be permitted or denied. For example, pioneering systems like PolicyMaker [2], KeyNote [3], REFEREE [4] and SD3 [5], have presented trust management systems based on security policies. More flexible and recent, hard trust management systems have been created, one among them is the work tool TrustBuilder2 [6].

"Soft trust" management systems, on the other hand are trust management systems that are based on concepts like experience, reputation and other dynamic evaluation parameters. For this purpose, the observations of the trustee behaviors are added to evaluate the trustee experience. Most of the works dedicated to trust estimations in different kinds of systems are based on local observations through monitored entities. One example of such systems can be found in [7].

One crucial point is that soft trust management engines assume the evaluation of the behavior is always granted and available for them. Additionally, to the best of our knowledge, no generic methods to check behaviors are found in the trust literature. As well, no formal approaches have been defined considering several points of observation. We propose to use distributed network monitoring techniques to analyze the packets exchanged between entities, in order to prove the interactions are trustworthy based on the observation of network messages in different points of observation. One important characteristic, which is always desirable in trust systems, is to have the trust information, as fast as possible. Using our proposed mechanism, we are able to provide the behavioral feedback of the systems on-line. Our aim is to provide trust information in a generic manner such that, any generic framework can use the information about these behaviors and incorporate it into the trust estimation algorithm. It is our point of view, that trust management systems will benefit from different inputs using different techniques; that is the reason why we do not aim to provide another approach how to assess trust, but rather providing existing trust management systems with behavioral evaluation of interactions.

On-line network monitoring cannot be directly applied by performing formal verification or model checking [8] techniques. The reason is that, a model of the system under test has to be derived in advance, and furthermore a set of properties can be verified for corresponding violations. Typically, the system description is omitted when performing on-line monitoring/passive testing, and therefore, this issue is left out of the scope of the paper. On the other hand, a formal specification of the system under test can be obtained by observing input/output traces and applying machine learning techniques [9]. However, when performing machine learning techniques, well known problems are encountered, such as statistics gathering, explosion problems (especially, for state transition models), etc. Therefore, in this paper, we discuss how a number of properties can be still verified for a system under test when the formal system specification is absent; and especially how to perform this process in a scalable way.

2. Distributed On-line Monitoring for Behavioral Evaluation

2.1. Approach

Our main objectives are: i) to be able to detect untrustworthy behaviors of entities where all other approaches fail to achieve it, providing feedback as fast as possible; ii) to provide a generic method to describe these untrustworthy behaviors and finally, iii) to test those described behaviors in a scalable manner.

To tackle the first point, the distributed network monitoring approach was proposed. With the use of distributed network monitoring, we can see behaviors that cannot be seen when using a single point of observation. Let us present a possible case scenario, a client computer is sending request to perform operations to a server. Both, the client and the server have a trust management engine, and they have allowed actions and replies from each other. The client computer sends a message of type "A" to the server and at the server and the server receives a type "B" message. The message type "B" is an allowed message type and the server performs the action. If the network traffic for both points of observation could be obtained and compared, an untrustworthy behavior can be detected. This example is illustrated in Fig. 1. Without correlating both points of observation, the untrustworthy behavior cannot be detected, even if having trust management engines incorporated, the systems will consider the interaction trustworthy.

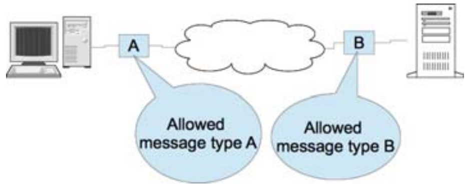

Fig. 1. Different allowed actions at communication ends.

We do not consider this simply a security issue. In fact, due to the trust definition we do not focus if the untrustworthy answer was due to an attack, a software failure (bug), system failure, misconfiguration, etc. The relevant fact is that the interaction was not proper and reporting the untrustworthy interaction as soon as possible is our goal.

2.2. Definitions and Assumptions

In order to accurately understand how our proposed approach solves the stated issues, first we need to introduce some preliminary concepts. A network packet (packet for short) is the abstraction of the transmitted bit-streams in a computer network; this abstraction allows us to interpret a packet as a formatted data unit. A packet is interpreted as a "Message" from a telecommunication protocol, for example a DNS query, a DNS response, etc. Analyzing a packet is to access the data inside that packet to search for particular values; these values have a defined meaning depending on the network protocol. Finally, network monitoring is the technique of analyzing the packets transmitted over a computer network. Several works like, [10], [11] and [12], proposed monitoring approaches considering local observations.

In our paper, we assume that the network packets are being forwarded from the different sources of interest to a monitoring server. Each of these sources contains network entities monitored through network interfaces called points of observation (P.O). We also assume that if the network entity has many interfaces, all the forwarded packets from the same network entity will be considered at the same point of observation.

The sequence of packets from a point of observation is called a network trace. A network trace (trace for short) is potentially infinite. When we have different traces from the points of observation, we can analyze the packets from one trace and create a relationship to another trace, defining the concept of distributed network monitoring.

In order to provide evaluation about behaviors, relationships between packets from different POs are created. The relationships are created with the packets' fields and conditions that hold over those fields in regards of other packets. Basically, the relationships are made performing comparisons. We can compare the values of these observations with constant values or variable values. The variable values are extracted from other packets (previously observed packets). These comparisons are defined formally in our work [13], by the definition of *atoms* and we also note that for the time relationships, we assume the network traces are synchronized using the NTP protocol [14]. Since there are multiple network traces from multiple POs, the comparisons can be done from: i) a specific network trace, that is using a specific point of observation, ii) any network trace, except a specific one or iii) any network trace, that is, at any point of observation, i.e., not specifying a point of observation.

The packet relationships and comparing the values will result in a composition. This composition is formally defined as a conjunction of atoms, which we call a *prototype*. A *prototype* is an abstract model of all the necessary and sufficient conditions a network packet should meet, including all its dependencies. For example, to describe a DNS query for an IP address, a packet prototype will be expressed in the formal language as: *p.flags.response* = *0* ^ *p.queries.type* = *'A'*.

A *prototype* is a part of the formal definition of formulae. One formula is a formal representation of what we will call a trust property. Many trust properties can be described and formalized in order to describe trust on an environment or context. Once the desired trust properties are checked on the network traces, we can give a verdict regarding the checked trust property. The possible verdicts are *pass* and *fail* if the statement is present. If the trust property does not reach a verdict, the result will be temporarily assigned as an inconclusive verdict. If many trust properties are described, then, different trust verdicts can be obtained.

The motivation and a method behind our approach were presented previously. Now, in order to test the proposed trust properties using distributed network monitoring we need to be able to express those properties. It is not sufficient to express the trust properties, in fact, we need to accurately express them, not leaving room for any ambiguity. Considering that, we need to employ a formalism. A formalism is not only useful to unambiguously express the properties, but, also for the software tools to be able to provide accurate verdicts. Without a doubt, our approach has a higher value, when verdicts can be automatized with a software program. Further, when providing a formalism, more researchers related to the field can generate trust properties to test. Because of those reasons we have created the necessary formal approach.

We decided to use our own approach rather than using other existing ones, the reason is that with the use of our formalization, we can describe the packets finely parameterized and at a granular level. Thus, we can make more complex and detailed relationships between packets. Another reason is that new application protocols rely heavily on the data parts and their semantics, for this reason they require a more data-oriented checking which the other approaches are not able to provide. Even old protocols have semantics that if the packets are treated as bit-streams some data values can be inaccurately obtained. For example, in the DNS protocol, the DNS notation and data compression method allow to specify a pointer to previously used data in the packet to avoid duplication of data (see [15]).

The formalism basic and most important principles are: the representation of a protocol message (packet) and the formal language lexical, syntactical and semantical properties; nevertheless, for the scope of this paper, only knowing the concepts regarding the language, namely, *atoms* and *prototypes* in particular are enough, and that is the reason why the interested reader might look for the formal language definition in our previous work [13]. Therefore, we present only the basic concepts regarding the packet hierarchical representation next.

A communication protocol message can be represented as a hierarchical set of label-value pairs. The representation of the packet will have the form defined by a message representation:

Definition 1: A message representation $\mathcal{M}$ is defined by the set of pairs $\mathcal{M} = \{(l, v) | l \in \mathcal{L} \wedge v \in \mathcal{S} \cup \Re \cup \mathcal{M}'\}$, where $\mathcal{L}$ is a predefined set of string labels, $\mathcal{S}$ represents the set of string values, $\Re$ represents the set of real numbers, and $\mathcal{M}'$ is a message representation sub-set.

For a given network protocol P, an associated message representation $\mathcal{M}$ can generally be defined by the set of labels and data values derived from the message format defined in the protocol specification. A message of a protocol P is any element $m \in \mathcal{M}$. For each $m \in \mathcal{M}$, we add two fields: a real number $t_m \in \Re^+$, which represents the time when the message m is received or sent by the monitored entity, and a PO string label which represents the point of observation from which the message m is collected.

Example of a message representation: a possible message for the DNS protocol [15]; specified using the previous definition could be:

$\mathcal{M}$={(time,154.576889000),(PO,“ADS”),(query_id,58921),(flags,{(response,0),(op code,std_query),(truncated,0),

(recursion_desired,1),(reserved,0),(non_auth_data_acceptable,0)}),(questions,1),(a nswers,0),(authorityRRs,0),

(additionalRRs,0),(queries,{(name,”telecom-sudparis.eu”),(type,”A”),(class,”IN”)})}

Representing a DNS query for the IP address of the associated domain name telecom-sudparis.eu.

For any given network protocol we have a mapping function between the bit-stream and the message representation.

Definition 2: The mapping function is the function $\mathcal{F}: \mathcal{B} \mapsto \mathcal{M}$, where $\mathcal{B}$ is the bit-stream of the network protocol and $\mathcal{M}$ is a message hierarchical representation as presented in Definition 1.

Once having the representation of the network messages, and the concepts of *prototypes* and *atoms*, some important constrains of on-line network monitoring systems need to be mentioned: i) a prototype has a set of conditions which can involve the packet itself or previously stored packets (dependencies); ii) for each packet, all prototypes must be tested, since, each packet could be observed at any given state during the execution time.

3. Scalable Evaluation of On-line Network Monitoring Systems

The evaluation process in an on-line monitoring system consists in evaluating if each packet satisfies the desired trust properties we need to check. Therefore, a scalable way for the evaluation algorithm is perhaps the biggest requirement. The trust properties have a set of conditions (*atoms*) that packet's data need to match against constant values or against the values of previously stored packets, as explained before. After matching the packet's conditions, checking if the matched packet completed a trust property is necessary. In order to provide verdicts regarding the trust properties we developed a first approach using the algorithm presented in our work in [16]. The worst-case analysis of the time work performed by the previously mentioned algorithm is expressed by the following equation:

$$T(eval\_prots) = 3N_p + \sum_{i=1}^{N_p} NPA_i + \sum_{i=1}^{N_p} 2NPD_i + \sum_{i=1}^{N_p}(N_p - i)QL_i$$
$$+\sum_{i=1}^{N_p}(N_p - i)(QL_i * NDA_i)$$

Where N_p is the number of prototypes in the formulae, NPA_i is number of atoms that require no dependencies of the *ith* prototype, NPD_i is the number of dependencies of the *ith* prototype, QL_i is the length of the queue of stored packets of the *ith* prototype, and NDA_i is the number of atoms that require dependencies of the *ith* prototype.

The experimental results achieved with the first algorithm are good. However, due to the on-line monitoring constraints, we are required to create the most scalable algorithm for the evaluation of trust properties. Based on the time complexity analysis of our algorithm, we note that the term that dominates the equation is the term, $\sum_{i=1}^{N_p}(N_p - i)(QL_i * NDA_i)$; from this term we can observe that *atoms* (conditions) need to be checked against the stored packet queues and this is being repeated up to $(N_p - i)$ times. In order to create a scalable algorithm, we need to avoid repeating any checks for all packets.

In order to improve the algorithm, known techniques are applied. First, we propose to make use of a data structure that will aid avoiding repeated checks. In addition to that, we propose to keep a track of previously visited packets in the stored queue to avoid re-visiting packets, which did not match previous tests, and therefore, not to check stored packets that do not meet all the necessary conditions.

We have chosen to use a tree-structured (single rooted) extended finite state automaton (EFSA) as the structure for the scalable evaluation. The reason is that, this structure fits the desired purpose of the algorithm. We propose evaluating the packets by doing the atomic test once and to keep track of the already tested atoms (a transition model based on predicates) and then, when a packet is found to match a *prototype* (at some accept state), execute some actions (updating functions), for instance, storing the packet on a queue or reporting a property verdict. These types of models have become popular to achieve scalable algorithms, for example, several works like, [17,18] use different types of automata, finite, non-deterministic, hybrid and extended to evaluate a regular expression language to achieve a scalable deep packet inspection.

Our target is to generate the EFSA from the necessary *prototypes*. The strategy in order to avoid repeating atomic tests is to generate transitions from the root state, adding predicates of the atomic tests which are more common at the beginning and

creating related atomic tests (atomic tests which are part of the same *prototype*) along the same path; for the next prototypes, uncommon atoms will be branches added at the current state after following the common transitions. Therefore, our algorithm to generate the EFSA relies on three principal actions: i) comparing each atom and add a count of how many times it appears in the formulae; ii) sort the prototypes putting first the ones containing the most common atoms, then, do a nested-sorting according the second most common atom, and so on; iii) finally, going along the path of the EFSA creating new nodes branching with its respective transitions based on the atoms or just following the already existing ones (starting from the root) and adding the proper updating functions. The algorithm to generate our EFSA based on the formulae prototypes can be found in the Algorithm 1.

```
Input: Prototypes $\mathcal{P}$ , State root
Output: EFSA
foreach $atom \in \mathcal{P}$ do
    repeated[atom] ← count(atom, $\mathcal{P}$);
end
maxRepeated ← max(repeated);
sortedPrototypes ← $\mathcal{P}$;
repeat
    atom ← atomRepeated(maxRepeated, $\mathcal{P}$);
    sortedPrototypes ← sortWRT(sortedPrototypes, atom)              /* nested-sorting */
    repeated[atom] ← -repeated[atom];
    maxRepeated ← max(repeated);
until maxRepeated>1;
current ← root;
foreach $P \in sortedPrototypes$ do
    while (atom ← higerCountAtom(P)) > 0 do
        if $atom \notin transitions(current)$ then
            addTransition(current, atom);
        end
        delete(atom,P);
        doTransition(current, atom);
    end
    addUpdatingFuncs(current, selectFuncs(P))        /* appropriate for the prototype */
end
```

Algorithm 1. EFSA generation algorithm.

Example of an EFSA generation: Let us consider the trust property: "For all responses from an authoritative DNS server, all future responses from other points of observation are the same replies from the authoritative DNS server if the queries are the same".

Using our approach we express this trust property by having the following prototypes:

$p1 \leftarrow p.dns.flags.R = 0 \wedge p.PO = 'ADS'$

$p2 \leftarrow p.dns.flags.R = 1 \wedge p.PO = 'ADS' \wedge p.dns.ID = p1.dns.ID$

$p3 \leftarrow p.dns.flags.R = 0 \wedge p.PO \neq 'ADS' \wedge p.dns.queries = p1.dns.queries$

$$p4 \leftarrow p.dns.flags.R = 1 \land p.PO \neq' ADS' \land p.dns.ID = p3.dns.ID \land p.dns.answers = p2.dns.answers$$

The atom count for these prototypes is:

$p.dns.flags.R = 0 \leftarrow 2$
$p.PO = 'ADS' \leftarrow 2$
$p.dns.flags.R = 1 \leftarrow 2$
$p.dns.ID = p1.dns.ID \leftarrow 1$
$p.PO \neq' ADS' \leftarrow 2$
$p.dns.queries = p1.dns.queries \leftarrow 1$
$p.dns.ID = p3.dns.ID \leftarrow 1$
$p.dns.answers = p2.dns.answers \leftarrow 1$

For this particular example, the order of the prototypes is not altered when sorting them. Finally, the generated EFSA by our algorithm (Algorithm 1) is represented in the Fig. 2.

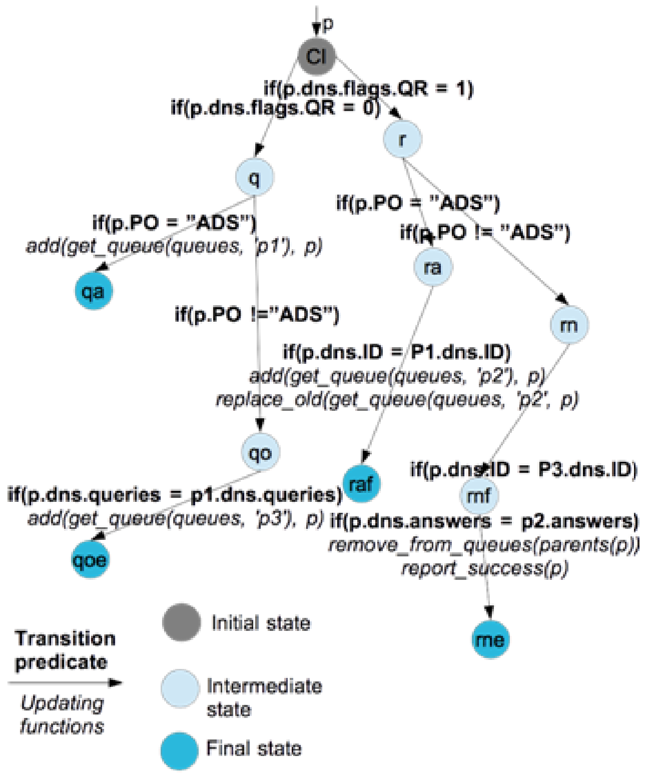

Fig. 2. Generated EFSA example

Once having the generated EFSA, we can introduce the proposed algorithm that is used to evaluate the packets using the auxiliary data structure we generated. The algorithm is shown as Algorithm 2:

```
Input: EFSA A, StoredPackets sps, Packet $p_i$
Output: Property verdicts
repeat
    transitioned ← false;
    foreach predicate in getTransitions(getCurrentState(A)) do
        if hasDependencies(predicate) then
            partialResult ← false;
            tempStoredPackets ← createStoredPackets();
            foreach storedPacket in getStoredPackets(sps, predicate) do
                if evalDependency(predicate, $p_i$, storedPacket) then
                    partialResult ← true;
                    add(tempStoredPackets, storedPacket)     /* avoid unnecessary checks */
                end
            end
            setStoredPackets(sps, predicate, tempStoredPackets);
            transitioned ← partialResult;
        else
            transitioned ← evalIndependent(predicate, $p_i$);
        end
        if transitioned then
            foreach updatingFunc in getUpdatingFuncs(A, predicate) do
                execute(updatingFunc, $p_i$);
            end
            setCurrentState(A, getStateTransition(A, predicate));
            goto end_loop;
        end
    end
    end_loop :
until transitioned;
```

Algorithm 2. Evaluation algorithm using EFSA.

Finally, we can proceed to calculate the complexity of the evaluation algorithm using the auxiliary EFSA (as shown in Algorithm 2). The work of the algorithm can be expressed by:

$$T(eval\_states) = \sum_{i=1}^{|S|} (\Theta(1) + T(eval\_trans_i)),$$

where $|S|$ is the cardinality of the set of states in the EFSA. Respectively, the work of $T(eval\_trans_i)$:

$$T(eval\_trans_i) = \sum_{j=1}^{|TA_i|} (\Theta(1) + \Theta(1) + \Theta(1) + T(eval\_sp_j) + \Theta(1) + \Theta(1))$$

$$+ \Theta(1) + \sum_{j=1}^{|U_i|} (\Theta(1)) + \Theta(1),$$

where $|TA_i|$ is the cardinality of the set of transitions of the *ith* element of the state set, $|U_i|$ is the cardinality of the set of updating functions for the *ith* state executed transition. Subsequently, The work of $T(eval\_sp_j)$:

$$T(eval\_sp_j) = \sum_{k=1}^{|Q_j|} (\Theta(1) + \Theta(1) + \Theta(1)),$$

where $|Q_j|$ is the length of the queue of the *ith* prototype stored packets queue.

Substituting and simplifying the equations we get that (we omit the algebraic operations):

$$T(eval\_states) = 3|S| + 5\sum_{i=1}^{|S|} |TA_i| + \sum_{i=1}^{|S|} |U_i| + 3\sum_{i=1}^{|S|} \sum_{j=1}^{|TA_i|} |Q_i|$$

We note that counting from all states each transition is the equivalent to count all transitions, i.e., $|T|$, the cardinality of all transitions. Similarly, counting from all states each updating function is the equivalent to count all updating functions, i.e., $|U|$ is the cardinality of all updating functions. After this substitution in the previous equation we get:

$$T(eval\_states) = 3|S| + 5|T| + |U| + 3\sum_{i=1}^{|T|} |Q_i|$$

The complexity of our algorithm results in an improved linear complexity, $O(|T|) = \sum_{i=1}^{|T|} |Q_i|$. We also note that any algorithm that runs in linear time can only modify a linear amount of memory cells and therefore, the space complexity of the algorithm yields a linear space complexity. It is also important to remark that the complexity of the algorithm (both in time and space) highly depends on the length of the stored queues of packets. In our previous works we have proposed having a continuous parallel process that given a timeout threshold, will remove from the packet queues unused packets. We do this in order to avoid resource starvation in the monitoring system.

4. Conclusions

In this paper, we have presented a scalable approach to evaluate on-line network monitoring systems. Furthermore, we have introduced an algorithm that regardless of the language used to express the monitoring properties is capable of generating an auxiliary model to evaluate them; the only requirements are the basic concepts and constraints that any on-line network monitoring system has. The proposed method after creating the data-structure uses a second algorithm that we presented in order to evaluate the packets and provide verdicts regarding them in a linear time.

Our contribution focuses on providing verdicts in a scalable manner and as stated in Section 3, the algorithm highly depends on the length of the queues of previously stored packets. Therefore, our future work includes proposing a complete model that takes timeouts into account and also to extend our current language to be able to express variable timeouts for each *prototype* individually. Naturally, developing a tool that incorporates the proposed approach is included into our perspectives.

References

[1]. T. Grandison, M. Sloman. A survey of trust in internet applications. IEEE Communications Surveys and Tutorials, 2000, vol. 3, no. 4, pp. 2-16.

[2]. M. Blaze, J. Feigenbaum, J. Lacy. Decentralized trust management. Proc. the IEEE Symposium on Security and Privacy, 1996. pp. 164–173. Oakland, CA, USA.

[3]. M. Blaze, J. Feigenbaum, A.D. Keromytis. Keynote: Trust management for public-key infrastructures. Proc. the Springer 6th International Workshop of Security Protocols, 1999. pp. 59–63. Cambridge, UK.

[4]. Y.-H. Chu, J. Feigenbaum, B. Lamacchia, P. Resnick, M. Strauss. Referee: Trust management for web applications. O'Reilly World Wide Web Journal, 1997, vol. 2, no. 3, pp. 127-139.

[5]. T. Jim. Sd3: A trust management system with certified evaluation. Proc the IEEE Symposium on Security and Privacy, 2001. pp. 106–115. Oakland, California, USA.

[6]. A. J. Lee, M. Winslett, K. J. Perano. Trustbuilder2: A reconfigurable framework for trust negotiation. Proc. the Third IFIP WG 11.11 International Conference, 2009. pp. 176–195. West Lafayette, IN, USA.

[7]. Z. Movahedi, M. Nogueira, G. Pujolle. An autonomic knowledge monitoring scheme for trust management on mobile ad hoc networks. Proc. the IEEE Wireless Communications and Networking Conference, 2012. pp. 1898–1903. Paris, France.

[8]. G. Holzmann. The spin model checker : primer and reference manual. Addison-Wesley Professional. 2003, pp. 1-596.

[9]. M.-N. Irfan, C. Oriat, R. Groz. Model inference and testing. Elsevier Advances in Computers, 2013, vol. 89, pp. 89-139.

[10]. D. Lee, R. Miller. Network protocol system monitoring-a formal approach with passive testing. IEEE/ACM Transactions on Networking, 2006, vol. 14, no. 2, pp. 424-437.

[11]. A. R. Cavalli, S. Maag, E. M. de Oca. A passive conformance testing approach for a manet routing protocol. Proc. the ACM Symposium on Applied Computing (SAC), 2009. pp. 207–2011. Honolulu, Hawaii, USA.

[12]. X. Che, F. Lalanne, S. Maag. A logic-based passive testing approach for the validation of communicating protocols. Proc. the 7th International Conference on Evaluation of

Novel Approaches to Software Engineering (ENASE), 2012. pp. 53–64. Wroclaw, Poland.

[13]. X. Che, J. Lopez, S. Maag, G. Morales. Testing trust properties using a formal distributed network monitoring approach. Springer Annals of telecommunications - Annales des télécommunications, 2014. pp. 1-11. doi: 10.1007/s12243-014-0454-3.

[14]. D. L. Mills. Internet time synchronization: the network time protocol. IEEE Transactions on Communications, 1991, vol. 39, no. 10, pp. 1482-1493.

[15]. P. V. Mockapetris, RFC 1035 Domain names — implementation and specification. Internet Engineering Task Force, 1987.

[16]. J. López, X. Che, S. Maag. An online passive testing approach for communication protocols. Proc. the 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2014. pp. 136–143. Lisbon, Portugal.

[17]. R. Smith, C. Estan, S. Jha, S. Kong .Deflating the big bang: Fast and scalable deep packet inspection with extended finite automata. Proc. Conference on Data Communication, SIGCOMM '08, 2008. pp. 207–218. New York, NY, USA.

[18]. M. Becchi, C. Wiseman, P. Crowley. Evaluating regular expression matching engines on network and general purpose processors. Proc. The 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2009. pp. 30–39. New York, NY, USA.

Масштабируемый метод оценки управления доверием на основе распределенных систем онлайн мониторинга

[1] Х. Лопез <jorge.eleazar.lopez_coronado@telecom-sudparis.eu>
[1] С. Мааг <stephane.maag@telecom-sudparis.eu>
[2] Ж. Моралес <gmorales@galileo.edu>
[1] Institut Mines Telecom, Telecom SudParis CNRS UMR 5157,
9 rue Charles Fourier, Évry, Île-de-France, 91000, France
[2] RLICT Universidad Galileo,
7a. Av. Final, Calle Dr. Eduardo Suger Cofiño, Zona 10, Guatemala, Guatemala

Аннотация. Корпоративные системы для организации и поддержания совместной работы становятся все более популярными. В условиях роста использования таких систем разработка методов, обеспечивающих надежное доверительное взаимодействие вовлеченных агентов, становится одной из приоритетных задач. Решение о том, с какими агентами (другими пользователями или приложениями) и каким образом осуществлять взаимодействие, может быть различным для различных систем. В данной работе мы акцентируем внимание на предоставлении вердикта о степени доверия на основе оценки поведения различных агентов с использованием распределенного сетевого он-лайн мониторинга. Предложенная оценка предоставляет системам управления, основанным на «мягком доверии» информацию об опыте доверителя. В данной работе мы предлагаем масштабируемый метод оценки для любого он-лайн мониторинга с использованием вспомогательной модели расширенного конечного полуавтомата и известных методов для уменьшения временной сложности алгоритма оценки.

Ключевые слова: управление доверием; онлайн сетевой мониторинг; масштабируемая оценка.

Список литературы

[1]. T. Grandison, M. Sloman. A survey of trust in internet applications. IEEE Communications Surveys and Tutorials, 2000, vol. 3, no. 4, pp. 2-16.

[2]. M. Blaze, J. Feigenbaum, J. Lacy. Decentralized trust management. Proc. the IEEE Symposium on Security and Privacy, 1996. pp. 164–173. Oakland, CA, USA.

[3]. M. Blaze, J. Feigenbaum, A.D. Keromytis. Keynote: Trust management for public-key infrastructures. Proc. the Springer 6th International Workshop of Security Protocols, 1999. pp. 59–63. Cambridge, UK.

[4]. Y.-H. Chu, J. Feigenbaum, B. Lamacchia, P. Resnick, M. Strauss. Referee: Trust management for web applications. O'Reilly World Wide Web Journal, 1997, vol. 2, no. 3, pp. 127-139.

[5]. T. Jim. Sd3: A trust management system with certified evaluation. Proc the IEEE Symposium on Security and Privacy, 2001. pp. 106–115. Oakland, California, USA.
[6]. A. J. Lee, M. Winslett, K. J. Perano. Trustbuilder2: A reconfigurable framework for trust negotiation. Proc. the Third IFIP WG 11.11 International Conference, 2009. pp. 176–195. West Lafayette, IN, USA.
[7]. Z. Movahedi, M. Nogueira, G. Pujolle. An autonomic knowledge monitoring scheme for trust management on mobile ad hoc networks. Proc. the IEEE Wireless Communications and Networking Conference, 2012. pp. 1898–1903. Paris, France.
[8]. G. Holzmann. The spin model checker : primer and reference manual. Addison-Wesley Professional. 2003, pp. 1-596.
[9]. M.-N. Irfan, C. Oriat, R. Groz. Model inference and testing. Elsevier Advances in Computers, 2013, vol. 89, pp. 89-139.
[10]. D. Lee, R. Miller. Network protocol system monitoring-a formal approach with passive testing. IEEE/ACM Transactions on Networking, 2006, vol. 14, no. 2, pp. 424-437.
[11]. A. R. Cavalli, S. Maag, E. M. de Oca. A passive conformance testing approach for a manet routing protocol. Proc. the ACM Symposium on Applied Computing (SAC), 2009. pp. 207–2011. Honolulu, Hawaii, USA.
[12]. X. Che, F. Lalanne, S. Maag. A logic-based passive testing approach for the validation of communicating protocols. Proc. the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2012. pp. 53–64. Wroclaw, Poland.
[13]. X. Che, J. Lopez, S. Maag, G. Morales. Testing trust properties using a formal distributed network monitoring approach. Springer Annals of telecommunications - Annales des télécommunications, 2014. pp. 1-11. doi: 10.1007/s12243-014-0454-3.
[14]. D. L. Mills. Internet time synchronization: the network time protocol. IEEE Transactions on Communications, 1991, vol. 39, no. 10, pp. 1482-1493.
[15]. P. V. Mockapetris, RFC 1035 Domain names — implementation and specification. Internet Engineering Task Force, 1987.
[16]. J. López, X. Che, S. Maag. An online passive testing approach for communication protocols. Proc. the 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2014. pp. 136–143. Lisbon, Portugal.
[17]. R. Smith, C. Estan, S. Jha, S. Kong .Deflating the big bang: Fast and scalable deep packet inspection with extended finite automata. Proc. Conference on Data Communication, SIGCOMM '08, 2008. pp. 207–218. New York, NY, USA.
[18]. M. Becchi, C. Wiseman, P. Crowley. Evaluating regular expression matching engines on network and general purpose processors. Proc. The 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2009. pp. 30–39. New York, NY, USA.