A Distributed Monitoring Approach For Trust Assessment Based On Formal Testing

Jorge López, Xiaoping Che, Stephane Maag
Institut Mines Telecom, Telecom SudParis
CNRS UMR 5157
9, rue Charles Fourier, 91011, EVRY Cedex, France
Email(s): Jorge.Eleazar.Lopez_Coronado@telecom-sudparis.eu,
Xiaoping.Che@telecom-sudparis.eu,
Stephane.Maag@telecom-sudparis.eu

Gerardo Morales
Universidad Galileo, RLICT
7a. Av. Final, Calle Dr. Eduardo Suger Cofño
Zona 10, Guatemala, Guatemala
Email: gmorales@galileo.edu

Abstract—Communications systems are growing in use and in popularity. While their interactions are becoming more numerous, trust those interactions now becomes a priority. In this paper, we focus on trust management systems based on observations of trustee behaviors. Based on a formal testing methodology, we propose a formal distributed network monitoring approach to analyze the packets exchanged between the trustor, trustee and other points of observation in order to prove the trustee is acting in a trustworthy manner. Based on formal “trust properties”, the monitored systems behaviors provide a verdict of trust by analyzing and testing those properties. Finally, our methodology is applied to a real industrial DNS use case scenario.

Keywords—Trust; Monitoring; Formal method; Communication systems

I. INTRODUCTION

Nowadays, daily life activities are drastically pointed at Internet services, for instance, to socially interact, make commerce, and create collaborative work. Since these systems are growing in use and in popularity, the need to boost the methods concerned interoperability is also growing; making thus trustworthy interactions of systems a priority.

Each application or system has to manage the decision regarding with who and how to interact with other users or applications. The concept of trust and its integration in the networks management are then inevitable.

Trust is defined in the literature by several ways. But the one we adopt in this paper is the one commonly applied and defined in [6]: “the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context”. Several factors are then involved:

- The trustor trusts an entity called the trustee. This implies a mechanism for identification or authentication.
- Belief is a subjective concept for the trustor. It is regarding an expectation on the future, but, it could be influenced by the past events with the trustee.
- The trustor expects behaviors from the trustee. Behaviors considered dependably, secure and reliable.
- A trustee might have different levels of trust for different contexts, that is, situations.

We herein focus on “soft trust”, that is trust management systems that can also be based on experience and reputation. For this purpose, the observations of the trustee behaviors are added to evaluate the trustee experience. Furthermore, most of the works dedicated to trust estimations in different kinds of systems are based on local observations through monitored entities. No formal approaches have been defined for distributed monitored elements by considering several points of observations. We therefore propose in this paper to use distributed network monitoring techniques to analyze the packets exchanged between the trustor, trustee and other points of observation in order to prove the trustee is acting in a trustworthy manner.

Our main contributions are the following:

- A formal model to define trust in distributed systems.
- We propose a distributed monitoring approach to analyze trust properties in distributed networks.
- We create a tool in order to automatize the testing of the trust properties.
- We successfully apply our methodology to a real industrial DNS use case scenario to evaluate our approach.

The remainder of the paper is organized as it follows. We depict in Section II the works related to our proposed approach. In Section III, we describe the main issues and assumptions in distributed systems regarding trust and monitoring. Then, the formal model allowing to express the trust properties is defined in Section IV. The approach is therefore experimented in Section V and the results analyzed. Finally, we provide some perspectives and conclude in Section VI.

II. RELATED WORKS

Many trust management systems are based on security policies. For example, pioneering systems like Policy-Maker [1], REFEREE [4] and SD3 [9]; have presented trust management systems based on security policies. These type of systems work by exchanging credentials and applying the security policies to the authenticated entities. Despite their advantages, they are not generic and force users to adopt certain authentication mechanisms and policy languages.
Other frameworks based on security policies have been presented. The framework called XeNA was presented in [8]. In this work the authors propose to use eXtensible Access Control Markup Language (XACML) for access control management. Trust with security policies is considered “hard trust”; because of the rigid concept of only accepting certain entities and under certain situations.

Besides, there are other works that use less rigid parameters of trust. Systems that incorporate notions of trust as an experience/reputation concept have been proposed as well. In [13], the authors applied these concepts to mobile ad-hoc networks. They monitor the behaviors of the neighbor nodes and keep local information of those interactions.

In [15], the authors created TRUST-OrBAC. In this work the authors propose to add an experience module on top of the security policy engine. We also share the idea that trust systems should incorporate different approaches of trust. Another interesting work that combine different approaches to provide trust information is the work presented in [7]. The authors build the SULTAN tool that incorporates also notions of risk into its design, quantifying risk and actually applying these notions into their trust evaluation algorithms.

To the best of our knowledge, none of the systems or frameworks use distributed network monitoring in order to provide trust information. All the exposed works provide a view only with the systems interactions between trustor and trustee. If beyond the observation of a satisfactory behavior when the system is not acting in a trustworthy manner, these type of systems will not be able to detect this behavior since the trustor having a single point of observation will consider this interaction as trustworthy when actually this is not the case. In the next section we analyze how untrustworthy interactions can occur with a single point of observation. Furthermore, the trustor will consider this interaction as trustworthy. We also analyze how with the use of distributed monitoring we can prove these interactions to be untrustworthy.

III. DISTRIBUTED NETWORK MONITORING APPROACH

Network monitoring is the technique of analyzing the packets transmitted over the network. Analyzing a packet (a DNS query, a DNS response, etc.) is to access the data inside that packet and search for particular values. These values are fields defined depending on the protocol. Several works proposed monitoring approaches considering one local observation [11, 2, 3].

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 are network entities monitored though interfaces called points of observation (PO). We also assume that if the network entity has many interfaces, all the forwarded packets from the same network entity will be considered 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.

With the use of distributed network monitoring, we can see behaviors, that when using a single point of observation will not be possible to see. One possible example of how distributed network monitoring can help discover these type of behaviors is when one host “A” is sending a message “M” to a host “B”. This message is only intended from host “A” to host “B”. Then, we observe at a different point of observation that the message “M” has been copied and is destined to another host. This scenario is described by Figure 1:

![Figure 1. Data mirrored to another host](image)

Many other behaviors can be constructed from distributed network monitoring. The complete scenario is revealed when relationships are created from different network traces. Several works like [16, 14] using different approaches have exposed intrusion detection systems (IDS) that are collaborative or distributed as a solution to detect possible attacks (such as, to detect a distributed denial of service (DDoS)). Collaborative or distributed IDSs can observe that different hosts located at different networks are sending network packets to the final common target. Thus, concluding that a distributed denial of service attack occurs.

The relationships are created with the packets fields and conditions that hold over those fields in regards of other packets. The relationships are made by doing comparisons. We can compare the values of these observations with constant values or variable values. The variable values are extracted from other packets. These comparisons are defined formally later in this work by the definition of atoms. The atoms formalization is found on Definition 2. Please note that for the time relationships, we assume the network traces are synchronized using the NTP protocol [12]. Since there are multiple network traces from multiple P.Os, the comparisons can be done from:

1) A specific network trace, that is through a specific point of observation.
2) Any network trace, that is, any point of observation, i.e., not specifying a point of observation.

By using the packet relationships and comparing the
values will result in a composition. This composition is formally defined also at Section IV by the definition of formulas in Definition 3. 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 a 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, fail if the statement is present. If the trust property does not reach a verdict, the result will be temporarily assigned to an inconclusive verdict. If many trust properties are described, then, different trust verdicts can be obtained. As stated before, these concepts will be formally defined in the following section.

IV. Formal Approach

A. Basics

Our formal approach is based on definitions, syntax and semantics that we have already published. They have been applied successfully in the areas of functional and performance testing of network protocols. That is the reason why, we do not give all details on the used model. However, the interested reader can refer to [3] for all relevant information. Though we do not deeply detail the semantic in this paper, interested reader can refer to [3] for all relevant information.

A communication protocol message is a collection of data fields of multiple domains. Data domains are defined either as atomic or compound. An atomic domain is defined as a set of numeric or string values. A compound domain is composed of atomic domains.

For a given network protocol \( P \), a compound domain \( M_p \) can generally be defined by the set of fields and data domains derived from the message format defined in the protocol specification/requirements. A message of a protocol \( P \) is any element \( m \in M_p \).

For each \( m \in M_p \), we add two fields: a real number \( t_m \in \mathbb{R}^+ \) which represents the time when the message \( m \) is received or sent by the monitored entity and \( P,O \) a string label which represents the point of observation from which the message \( m \) is collected.

A trace is a sequence of messages of the same domain containing the interactions of a monitored entity in a network, through an interface (or P.O) with one or more peers during an arbitrary period of time. The P.O also provides the relative time set \( T \subset \mathbb{R}^+ \) for all messages \( m \) in each trace.

B. Syntax and Semantics of our formalism

As described in the section III, our approach focuses on applying distributed network monitoring to the trust management domain. In order to achieve that, we used our previous work [10]. In here, the syntax and semantics have been extended to include several P.Os. The syntax is based on Horn clauses is defined to express properties that are checked on extracted traces. We briefly describe it in the following. Formulas in this logic can be defined with the introduction of terms and atoms, as it follows.

Definition 1. A term is defined in the Backus Normal Form (BNF) as \( \text{term} ::= c \mid x \mid x.l.l...l \) where \( c \) is a constant in some domain, \( x \) is a variable, \( l \) represents a label, and \( x.l.l...l \) is called a selector variable.

Example 2. Let us consider the following message:

\[ m = \{(\text{time}, `154.576889000`), (PO, Auth_DNS_Srv) (query_id, 58921), (flags, \{(\text{response}, 0), (opcode, \text{std_query}), (truncated, 0), (recursion_desired, 1), (reserved, 0), (\text{non_auth_data_acceptable}, 0)\})\} \]

In this message, the value of recursion_desired inside flags can be represented by \( m.\text{flags}.\text{recursion_desired} \) by using the selector variable.

Definition 2. An atom is defined as \( \text{A} ::= p(\text{term}_1, \ldots, \text{term}_k) \mid \text{term} = \text{term} \mid \text{term} \neq \text{term} \mid \text{term} < \text{term} \mid \text{term} > \text{term} \mid \text{term} + \text{term} = \text{term} \) where \( p(\text{term}_1, \ldots, \text{term}_k) \) is a predicate of label \( p \) and arity \( k \). The timed atom is a particular atom defined as \( p(\text{term}_{t1}, \ldots, \text{term}_{tk}) \), where \( \text{term}_{ti} \in T \).

Example 3. Let us consider the message \( m \) of the previous example. A point of observation constraint on \( m \) can be defined as ‘\( m.\text{PO} = \text{Auth_DNS_Srv} \)’. These atoms help at defining timing aspects as mentioned in Section IV-A.

The relations between terms and atoms are stated by the definition of clauses. A clause is an expression of the form: \( A_0 \leftarrow A_1 \land \ldots \land A_n \), where \( A_i \) are atoms.

Definition 3. A formula is defined by the following BNF:

\[ \phi ::= A_1 \land \ldots \land A_n \mid \phi \rightarrow \phi \mid \forall x(\phi) \mid \forall y > x(\phi) \mid \forall y < x(\phi) \mid \exists x(\phi) \mid \exists y > x(\phi) \mid \exists y < x(\phi) \]

where \( A_i \) are atoms, \( n \geq 1 \) and \( x, y \) are trace messages.

The quantifiers commonly define “it exists” (\( \exists \)) and “for all” (\( \forall \)). Therefore, the formula \( \forall x \phi \) means “for all messages \( x \) in the trace, \( \phi \) holds”. Based on the above described operators and quantifiers, as well as our semantics [3], we provide an interpretation of the formulas to evaluate them to \( \top \) (‘Pass’), \( \bot \) (‘Fail’) or ‘?’ (‘Inconclusive’).

We formalize trust by using our syntax and the truth values \( \{\top, \bot, ?\} \) are provided to the interpretation of the obtained formulas on real protocol execution traces. These formulas represent and allow to model trust properties.
V. Experiments

A. Scenario Description

We choose to tackle the same scenario as chosen by [9] and [5]; since this is still an open issue on trust and security. Also, the implications of trust and security on this scenario can affect all types of end users and systems. The problem can be described as trusting DNS responses to queries.

The DNS original design does not take into consideration any concept of trust or security. If the DNS query responses are modified, it can have severe implications. End users and systems can be deceived and they can send data to the wrong destination. The information can be later delivered to the real destination; the end user or system might not notice any untrustworthy interaction. End users can be directed to phishing pages, advertising pages or any other. Moreover, the system can have a trust management engine in place and consider all the interactions as trustworthy. The objective is to get the spoofed responses to arrive before the original one.

In many occasions DNS caches change the records, willingly or not. Possibly, ISPs can do this at their caching DNS servers; motivated to save bandwidth or to hijack the non-existent domains to forward users to publicity sites. We can also consider the case the caching DNS server is returning the wrong records due to a system failure or software bug. In this case the caching DNS server lacks of competence.

If the caching DNS server is not competent to act dependably, securely, and reliably to return the correct DNS records, it is not acting in a trustworthy manner by definition. That is the reason why we do not consider entirely a security issue. This is a trust issue, trustworthy interactions between the DNS caching server and the resolvers are a must.

By the use of distributed network monitoring we can improve the trustworthiness in the DNS responses. Combining information from different points of observation allow to identify situations that with a single observation will not be possible. This is what we experiment and demonstrate in the following detailed DNS scenario.

Let us assume the following case, when an employee is working at a remote location, which can be a hotel with public Internet connection as illustrated in Figure 2. The employee is using a company application that sends data to a company application server “domain.tld”. The hotel’s caching DNS server sends the wrong information to the client computer. The client computer will send the application data to a third party machine. This third party machine eavesdrops the communication and then re-routes traffic to the original destination, that is the company’s application server. Both the server at “domain.tld” and the client computer will not detect any malicious behavior. The client computer sends the DNS traffic to the monitoring server. Also, the authoritative DNS server for “domain.tld” sends the DNS traffic to the monitoring server as well.

In this case, two points of observation are in place, P.O.1 and P.O.2. Through the P.O.1, we monitor the DNS responses of the authoritative DNS server for the domain “domain.tld”. In the P.O.2, we monitor the responses obtained by the resolver, that is the client computer. At the monitoring server, when the two responses for the same query are compared, we notice they differ. Then, we can provide useful information about the trustworthiness of the hotel’s caching DNS server.

B. Formal Specification

We have formally defined one formula in order to express a trust property. As described in the scenario, our target is to guarantee that the responses from the DNS resolvers match the responses from the authoritative DNS server. With the use of distributed network monitoring and our formal specification we can declare the necessary trust property. After evaluation, it will provide trust verdicts applied to the previously described scenario.

The trust property is: $\Psi = “\text{For all responses from an authoritative DNS server, all future responses from other points of observation are the same replies of the authoritative DNS server if the queries are the same}”$. This can be expressed in the following formula:

$$\Psi = \{\forall x (req_f(x, ADS)) \rightarrow \exists y > x (res_f(y, x, ADS))\} \rightarrow \{\forall a (req_enf(a, y, ADS)) \rightarrow \exists b > a (res_enf(b, a, y, ADS))\}$$

Where the intermediate clauses are defined such as:

- $req(x) \leftarrow x.flags.response = 0$
- $res(y) \leftarrow x.flags.response = 1$
- $eq_q(x, y) \leftarrow x.queries = y.queries$
- $eq_a(x, y) \leftarrow x.answers = y.answers$
- $from(x, P) \leftarrow x.PO = P$
- $nfrom(x, P) \leftarrow x.PO \neq P$
- $after(x, y) \leftarrow x.time > y.time$
property testing. The tool captures and filters the network traffic so only interesting traffic is passed to the evaluation process.

Two threads are created, one for periodically updating the current status to the user and the other for the global timeout of packets. The global timeout of packets refers to the maximum time a packet can be kept if this packet is not a dependency of any other packet within a property. This global timeout is used by the on-line monitoring to ensure that potentially incomplete properties do not occupy resources permanently. When a packet is decided to be removed by the global timeout function if the packet has not been used for completing one property, then a TIMEOUT FAIL verdict will be reported for that property. The value of the timeout is determined by an expert in the protocol.

A packet prototype describes all the necessary conditions for a packet to be considered an element of a property. This includes dependencies to other prototypes and relationships between them, i.e., comparisons. When a packet matches all the properties, it will become a type of the specified prototype. For example, consider a simple property:

\[ \Phi = \forall x (x.\text{flags.request} = 0 \rightarrow \exists y > x(y.\text{flags.request} = 1 \land y.id = x.id)) \]

The prototype \( y \) that belongs to the property \( \Phi \) describes that, in order for a packet to be considered as a "\( y \)" it needs to have the request flag "on". Also, it has a dependency of type \( x \). Both of their "ids" should match.

Finally, when a packet is considered a type of a prototype, it gets stored on a list of that prototype. Also, if the packet has dependencies it gets associated to those dependencies. When all the prototypes of a property are completed or a packet of those associations gets a timeout, verdicts on that property are provided. The associated packets get removed from the respective lists.

Typically each packet gets tested for each prototype, unless a prototype has dependencies and any of the dependency lists is empty. In this case, the packet does not even get tested for that prototype. This is a performance enhancement. All the design motivation for our testing tool is to have a fast matching algorithm so a packet can be kept, tagged or discarded as fast as possible. Due to the lack of space in this paper we do not include pseudo-code for the algorithm.

In the next subsection we will discuss the testing architecture and the results obtained when we simulated the case scenario and used our tool to obtain verdicts about \( \Psi \).

D. Testing Architecture and Results

We conducted our experiments in a live server provided by an industrial partner, Tilidom\(^2\). These experiments were realized at "ns2.tilidom.com", one of the authoritative DNS servers for the domain "tilidom.com". We illustrate our testing architecture in Figure 3.

![Architecture for the testing framework](image)

Figure 3. Architecture for the testing framework

Although our testing architecture is not fully distributed, we consider different POs. For this testing architecture we consider the combination between source IP address 62.73.5.21, source port 53 and destination IP address 62.73.5.21, destination port 53 as the PO named "ADS", for the authoritative DNS queries and replies. This entitles us to differentiate the PO between client DNS requests and authoritative DNS responses.

At the "ns2.tilidom.com" server, our prototype tool captures live packets on one network interface where it receives queries from the caching DNS servers and responds to those queries. At the same server and network interface, we did DNS queries for the domain tilidom.com using a DNS lookup utility, the "host" command. We queried several known public DNS servers like, Google DNS, OpenDNS, Level3 DNS, etc. All these queries and responses completed the \( \Psi \) property giving PASS verdicts as expected.

<table>
<thead>
<tr>
<th>Query</th>
<th>Google DNS</th>
<th>Level3 DNS</th>
<th>RogueDNS</th>
<th>AuthDNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>62.73.5.28</td>
<td>62.73.5.28</td>
<td>62.73.5.28</td>
<td>62.73.5.28</td>
</tr>
<tr>
<td>Verdict</td>
<td>PASS</td>
<td>PASS</td>
<td>PASS</td>
<td>FAIL 1</td>
</tr>
</tbody>
</table>

Table 1. EXPERIMENT RESULTS

\(^1\)http://www.tcpdump.org/manpages/pcap-filter.7.txt  
\(^2\)http://tilidom.com/
We also created a DNS service in other server using the bind\textsuperscript{3} software. At that service we created a master zone for the domain “tilidom.com” and created different values for the DNS records. These values differed from the answers of the authoritative DNS server values. After, from the authoritative DNS server we queried this simulated rogue DNS server and the tool reported a FAIL verdict.

VI. CONCLUSION AND PERSPECTIVES

In this paper, we define a testing framework using a formal distributed approach to monitor and test trust properties by evaluating real network traces from different points of observation. While most of the approaches are based on local probes, we define in this work a correlation of the testing verdicts to evaluate the trust in distributed systems. A formal syntax and semantics are defined to express trust properties. A tool was developed in order to test the formalized trust properties into real time captures. The tool was applied on DNS live traces through a distributed use case. Interesting and promising results are provided.

While our obtained results are valuable, our testing architecture is not fully distributed. We plan to extend our testing architecture to send different packets captured from the P.Os and process them with our tool at a centralized monitoring server. Furthermore, we plan to create a generic plug-in for sending interesting traffic to monitoring servers; the perspectives includes sending the traffic using some encrypted transport. We also plan to extend the trust properties to include probably DNS updates on the trust verdicts.

Additionally, the traces may be collected by our server at different periods and then eventually with an important delay (due to the diverse queries/responses along the testing time). Thus, it is expected to analyze multiple P.Os and compare the trust properties against stored statistical results. It would decrease the delay between the observations and the verdicts. Finally, we plan to apply our approach to other systems to test trust properties on diverse agents.

REFERENCES


\textsuperscript{3}\url{https://www.isc.org/downloads/bind/}