A Formal Distributed Network Monitoring Approach for Enhancing Trust Management Systems

Jorge López and Stephane Maag
Institut Mines-Télécom/Télécom SudParis,
CNRS UMR 5157 SAMOVAR.
9, rue Charles Fourier
91011 EVRY Cedex, France.
{Jorge.Eleazar.Lopez_Coronado,
Stephane.Maag}@telecom-sudparis.eu

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

ABSTRACT

As the Digital Ecosystems are growing in use and in popularity, the need to boost the methods concerned by their interoperability is growing as well; making thus trustworthy interactions of the different agents (e.g., network systems) a priority. In our work, we focus on "soft trust", that is trust management systems that can be based on experience and reputation. Each trust system defines how they evaluate the trustee experience. The observations of the trustee behaviors are added to the trustee experience. Furthermore, most of the works dedicated to trust estimations in different kinds of ecosystems are based on local observations through monitored entities. No formal approaches have been defined for distributed monitored elements by considering several points of observations. This is what we intend in this work. We propose to use distributed network monitoring techniques to analyze the packets that the truster and trustee exchange in order to prove the trustee is acting in a trustworthy manner. A formal approach is defined to express trust properties and to evaluate them on real execution traces. Our approach is applied on DNS traces for assessing the trust among the entities.

Categories and Subject Descriptors

H.4 [**Security & Privacy**]: Trust & Risk; C2.3 [**Network Operations**]: Network Monitoring; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages

General Terms

Trust Management

Keywords

Trust systems, Network monitoring, Formal methods

1. INTRODUCTION

MEDES'13 October 29-31, 2013, Neumünster Abbey, Luxembourg

Internet has become one of the most popular ways to socially interact, make commerce, and create collaborative work. With time, the collaborative aspect, supported by Internet has evolved bringing new tools, methodologies and concepts.

The Digital Ecosystems is an emerging concept that involves human individuals, information services as well as network interaction and knowledge sharing tools along with resources as described by [4]. As the Digital Ecosystems are growing in use and in popularity, the need to boost the methods concerned by the interoperability is growing as well; making thus trustworthy interactions of the different agents (i.e., 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 who and how to interact with other users or applications depend on each application or system. One of the first works that introduced trust as a computer science concept is the one realized by [20]. In this work they assign a range of trust and distrust going from -1 to 1. While these fuzzy logic values are very intuitive for the characterization of trust as a computer science concept; this is a first interesting way of formalizing trust concepts. Instead of using a binary approach of complete trust these values are considered as fuzzy logic values. Fuzzy logic variables or values handle the concept of partial truth. For example to describe "how black is a gray color", you can use these type of values; a 0.1 value will indicate a pale gray. The same concept applies for a trust value of 0.95, this means a strong trust from the truster to the trustee. Many systems have adopted this methodology as well. For example, in the works realized by [24, 19, 26] the functions of trust associate the trust and distrust values on the same ranges, the closed interval from minus one to one.

There are many definitions of trust in the literature, but the one we adopt here is the one commonly applied and defined in [10] "the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context".

From the definition we note that it involves several factors:

- The truster trusts an entity called the trustee. This implies a mechanism for identification or authentication.
- Belief is a subjective concept for the truster. It is regarding an expectation on the future, but, it could be

influenced by the past events with the trustee, i.e., experience.

- The truster expects behaviors from the trustee. Behaviors considered dependably, secure and reliable.
- A trustee might have different levels of trust for different contexts.

The statements presented above are very important. From those points, many researchers have created many approaches of trust. For instance, several trust management systems use security policies and authentication in order to provide the concept of trust. In these types of systems the entity called trustee is related to an authentication mechanism. The policy languages express the actions allowed for each trustee. This is called "hard trust" because the actions can only be permitted or denied.

In our work, we focus our efforts on "soft trust", that is trust management systems can also be based on experience and reputation. Each trust system defines how they evaluate the trustee experience. The observations of the trustee behaviors are added to the trustee experience. Furthermore, most of the works dedicated to trust estimations in different kinds of ecosystems are based on local observations through monitored entities. No formal approaches have been defined for distributed monitored elements by considering several points of observations. This is what we intend in this work.

In our paper we provide feedback regarding the behavior of a trustee. Furthermore, we aim at providing trust information in a generic way so any generic framework can use the information about these behaviors and incorporate it into the trust estimation process. It is our point of view that trust systems will benefit from different techniques inputs.

In this work, we propose to use distributed network monitoring techniques to analyze the packets that the truster and trustee exchange in order to prove the trustee is acting in a trustworthy manner.

Our main contributions are the following:

- We define a formal model to express trust properties in network ecosystems.
- We propose a distributed monitoring approach to analyze trust properties in distributed networks.
- We successfully apply our methodology to a DNS use case scenario.

As stated before we applied our method to verify the DNS responses. We can probe that these responses are manipulated or altered and thus conclude about the trustworthiness of the server responding the queries from the DNS resolver.

The remainder of the paper is organized as it follows. We depict in Section 2 the works related to our proposed approach. In Section 3, we describe the main issues in distributed systems regarding trust and monitoring. Then, the formal model allowing to express the trust properties is defined in Section 4.2. The approach is therefore experimented in Section 5.3 and the results analyzed. Finally, we provide some perspectives and conclude in Section 6.

2. RELATED WORKS

Many trust management systems are based on security policies. For example, pioneering systems like PolicyMaker[3], KeyNote[2], REFEREE[6] and SD3[13]; 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. These systems have several advantages. However, they are not generic and force users to adopt certain authentication mechanisms and policy languages.

A more generic work is presented by TrustBuilder2[15]. In this work the authors presented a generic and extensible framework for authentication and security policy exchanges. The extensible features of the tool should allow interaction between their tool and information provided by an external module. For example, the trust information provided by the use of distributed network monitoring. We agree philosophically on a generic and reconfigurable framework is the best path to make existing software incorporate trust notions.

Other frameworks based on security policies have been presented. The framework called XeNA was presented in [12]. 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.

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. For example, the work realized by [23], the authors applied these concepts to mobile ad-hoc networks (MANet). They monitor the behaviors of the neighbor nodes and keep local information of those interactions. Also they share this information to other nodes on the network. Using both local and remote information they can calculate how trustworthy a node in the network is.

In [19, 26], 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 [11]. The authors build a tool called SULTAN. This tool 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 truster and trustee. If beyond the observation of a satisfactory behavior the system is not acting in a trustworthy manner, then, these type of systems will not be able to detect this behavior.

3. 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.

In this 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 enti-

ties 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 [18].

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. Several works like [28, 25, 16] 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.

Many other behaviors can be constructed from distributed network monitoring. The complete scenario is revealed when relationships are created from different network traces. These relationships are created with the packets fields and conditions that hold over those fields in regards of other packets.

Please note that for the time relationships, we assume the network traces are synchronized using the NTP protocol [21]. Aside from this relationship between the packets, we can compare the value of these observations with constant values or variable values. The variable values are extracted from other packets. Since there are multiple network traces from multiple PO, 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.

By using the packet relationships and comparing the values will result in a composition of what we will call a **"trust property"** on the monitored system. Once the desired trust properties are checked on the network traces, we can give a verdict regarding the checked trust property. The possible values are **pass**, **fail** if the statement is present. If the trust property does not reach a verdict, then, the result will be **inconclusive**. These concepts will be formally defined in the following section.

4. FORMAL APPROACH

4.1 Basics

A communication protocol message is a collection of data fields of multiple domains. Data domains are defined either as *atomic* or *compound* [5]. An *atomic* domain is defined as a set of numeric or string values. A *compound* domain is defined as follows.

Definition 1. A *compound* value v of length $n > 0$, is defined by the set of pairs $\{(l_i, v_i) \mid l_i \in L \wedge v_i \in D_i \cup \{\epsilon\}, i = 1...n\}$, where $L = \{l_1, ..., l_n\}$ is a predefined set of labels and D_i are data domains. A *compound domain* is then the set of all values with the same set of labels and domains defined as $\langle L, D_1, ..., D_k \rangle$.

Once given a network protocol P, a *compound* domain M_p can generally be defined by the set of labels 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 PO a string label which represents the point of observation from which the message m is collected.

Example 1. A possible message for the DNS protocol [22], specified using the previous definition could be

$m = \{(time, \text{'863.596183000'}), (PO, \text{Auth\_DNS\_Srv})$
$(query\_id, 6912), (flags, \{(response, 0),$
$(opcode, \text{std\_query}), (truncated, 0), (recursion\_desired, 1),$
$(reserved, 0), (non\_auth\_data\_acceptable, 0)\}),$
$(questions, 1), (answers, 0), (authority\_RRs, 0),$
$(additional\_RRs, 0), (queries,$
$\{(name, \text{telecom-sudparis.eu}), (type, \text{A}), (class, \text{IN})\})\}$

representing a DNS query for the domain telecom-sudparis.eu. The value of *time* '863.596183000' ($t_0 + 863.596183000$) is a relative value since the PO started its timer (initial value t_0) when capturing traces.

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

4.2 Syntax and Semantics of our formalism

As described in the 3 section, our approach focuses on applying distributed network monitoring to the trust management domain. In order to achieve that, we used our previous work[14]. In this work the syntax and semantics have been extended to include several POs. 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 2. A *term* is defined in the Backus Normal Form (BNF) as $term ::= c \mid x \mid x.l.l...l$ 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:

$m = \{(time, \text{'154.576889000'}), (PO, \text{Auth\_DNS\_Srv})$
$(query\_id, 58921), (flags, \{(response, 0),$
$(opcode, \text{std\_query}), (truncated, 0), (recursion\_desired, 1),$
$(reserved, 0), (non\_auth\_data\_acceptable, 0)\}),$
$(questions, 1), (answers, 0), (authority\_RRs, 0),$
$(additional\_RRs, 0), (queries,$
$\{(name, \text{telecom-sudparis.eu}), (type, \text{A}), (class, \text{IN})\})\}$

In this message, the value of $recursion\_desired$ inside $flags$ can be represented by **m.flags.recursion_desired** by using the *selector variable*.

Definition 3. A *substitution* is a finite set of bindings $\theta = \{x_1/term_1, ..., x_k/term_k\}$ where each $term_i$ is a *term* and x_i is a variable such that $x_i \neq term_i$ and $x_i \neq x_j$ if $i \neq j$.

Definition 4. An *atom* is defined as

$$A ::= \overbrace{p(term, ..., term)}^{k} \mid term = term \mid term \neq term \mid term < term \mid term > term \mid term + term = term$$

where $p(term, ..., term)$ is a predicate of label p and arity k. The *timed atom* is a particular atom defined as $\overbrace{p(term_t, ..., term_t)}^{k}$, where $term_t \in T$.

Example 3. Let us consider the message m of the previous example. A point of observation constraint on m can be defined as 'm.PO = Auth_DNS_Srv'. These atoms help at defining timing aspects as mentioned in Section 4.1.

The relations between *terms* and *atoms* are stated by the definition of clauses. A *clause* is an expression of the form

$$A_0 \leftarrow A_1 \wedge ... \wedge A_n$$

where A_0 is the head of the clause and $A_1 \wedge ... \wedge A_n$ its body, A_i being *atoms*.

Definition 5. A formula is defined by the following BNF:

$$\phi ::= A_1 \wedge ... \wedge A_n \mid \phi \rightarrow \phi \mid \forall_x \phi \mid \forall_{y>x} \phi \mid \forall_{y<x} \phi \mid \exists_x \phi \mid \exists_{y>x} \phi \mid \exists_{y<x} \phi$$

where $A_1, ..., A_n$ are *atoms*, $n \geq 1$ and x, y are variables.

In our approach, while the variables x and y are used to formally specify the messages of a trace, 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, ϕ holds".

The semantics used in our work is related to the traditional Apt–Van Emdem–Kowalsky semantics for logic programs [27], from which an extended version has been provided in order to deal with messages and trace temporal quantifiers. Based on the above described operators and quantifiers, we provide an interpretation of the formulas to evaluate them to $\top$ ('*Pass*'), $\bot$ ('*Fail*') or '?' ('*Inconclusive*'). By lack of space, we do not detail here the evaluation and interpretation of the formal properties on traces. However, the interesting reader can refer to [14] in which all the algorithms are defined.

We formalize trust by using the syntax above described 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**.

5. EXPERIMENTS

5.1 The Domain Name System

The Domain Name System (DNS) is standardized on RFC 1035 [22], is a hierarchical naming system for network entities. Its most common known use is to associate a domain names to numerical IP address; although it can be used to store several other types of information. It is one of the most essential services in the Internet. The main purpose is to make easy for users to remember specific hostnames or service providers. For instance, it is easy to remember the name "google.com", yet, is hard to remember "173.194.40.167" which is actually one of the IP addresses associated for that domain name. Moreover, for all the services in the Internet, DNS provides information as well, for example, information about where certain service as the mail exchanger is located for certain domain.

The functional aspect of DNS as described before is hierarchical. A domain name consists of labels that are concatenated to each other and separated by dots. For example the domain name "www.example.com" consists of three labels, "www", "example" and "com". The labels hierarchy goes from left to right, being the rightmost label the one with the highest hierarchy. The rightmost label is called top level domains (TLD). The information of the TLDs are stored on the root DNS servers. The TLD DNS servers have the information of the authoritative DNS servers for each domain. The authoritative DNS servers are the servers that have the official DNS records for a domain.

The domain name resolution mechanism starts with a DNS resolver. The DNS resolvers asks its local caching DNS server the domain in question. The caching DNS server is usually a recursive resolver. It queries the root DNS servers for the information about the TLD of the domain in question. After that it queries the TLD server to obtain the information about the authoritative DNS server. Then it continues this recursive process until it reaches an answer. The final step is to return the DNS resolver an answer.

We shortly described the domain name system in order to have clear concepts of the problematic we decided to tackle. In the following subsection we describe the scenario.

5.2 Scenario Description

We choose to tackle the same scenario as chosen by [13] and [9]; 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 send data to the wrong destination. The target is to divert the data from the original source. 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.

To understand how a DNS response can be changed, we first analyze the structure of a DNS query and response. A DNS query has the following relevant parameters: (i) Source address and source port, (ii) Destination address and destination port (53), (iii) Query ID and (iv) Query name, type and class.

The response back includes the same fields. The use of the query ID is to synchronize/correlate the response back. Knowledge about three parameters is required to successfully spoof a response: the query ID, source port and Destination address.

Many attacks against the DNS systems have been studied. The attacks basically work by winning a "race condition".

The objective is to get the spoofed responses arrive before the original one. If a specific domain is the target of the spoofing, the queries can be forced.

On 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 consider this case as the caching DNS server is not acting dependably. 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. In [7] the authors analyze many cases including the hijacking of non existent domains for commercial abuse and the software implementation errors.

If the *caching DNS server is not competent to act dependably, securely, and reliably to return the correct DNS records*, then, 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 is a must.

A DNS security extension, DNSSEC, is proposed at RFC 4033 [1]. In a nutshell, DNSSEC uses electronic signed data. The signatures are encrypted using asymmetric keys. The signatures are added in a hierarchical chain. All DNS resolvers have the public keys from the root DNS servers. The DNS resolvers do the chain resolution. This makes a DNS resolver also able to verify the returned records actually belong to an authoritative server. DNSSEC has backward compatibility in mind. This is good, since DNS resolvers can decide not to implement the DNSSEC verification and they will be able to continue working despite the fact the authoritative records implement DNSSEC. However, a true implementation on the Internet can take many years; because, in order for it to truly work, a complete change on all internet DNS authoritative servers and DNS resolvers is needed. A similar case happens with the IPv6 protocol [8].

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 the following detailed DNS scenario.

Let us assume the following case, when an employee is working at a remote location, let us suppose a public internet connection on a hotel in another continent. 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. We illustrate this scenario on Figure 1.

Two points of observation are in place, PO1 and PO2. In the point of observation 1, we monitor the DNS responses of the authoritative DNS server for the domain "domain.tld". In the point of observation 2, we monitor the responses obtained by the resolver, that is the client computer. At

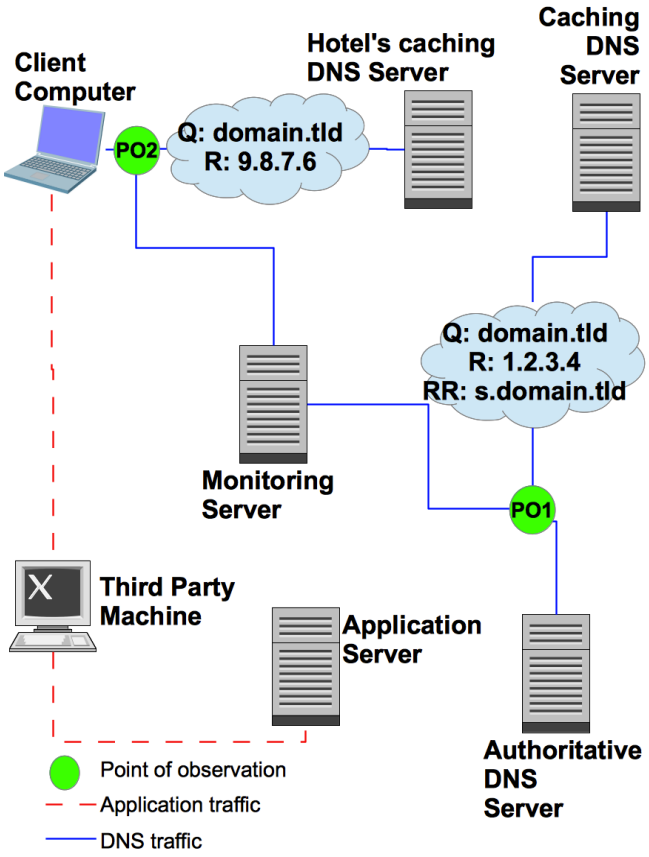

Figure 1: Distributed monitoring of DNS responses

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.

The monitoring server will receive the network traces of DNS traffic from different sources in order to provide trust information about different situations. To better illustrate this behavior we provide a message sequence chart (MSC). In this MSC, the monitoring server receives information from two different sources: the DNS resolver, which is at the client computer and the authoritative DNS server. The messages can be received in different order at the monitoring server. The synchronization of the queries is done by the query id fields and the source of the observations. We illustrate this in Figure 2. Here is the description of the messages: (1) Authoritative DNS server incoming query for the domain.tld record. (2) Authoritative DNS server outgoing response and outgoing referral record for the authoritative server. (3) DNS resolver outgoing query for the domain.tld record. (4) DNS resolver incoming response for the domain.tld record.

We use the formal approach and create trust properties to provide trust information in the following subsection.

5.3 Experiments Studies

In order to simulate the scenario, this can be setup with one DNS resolver and two DNS servers. One authoritative DNS server for a domain name and one caching DNS server that intentionally changes the values of the answers for that

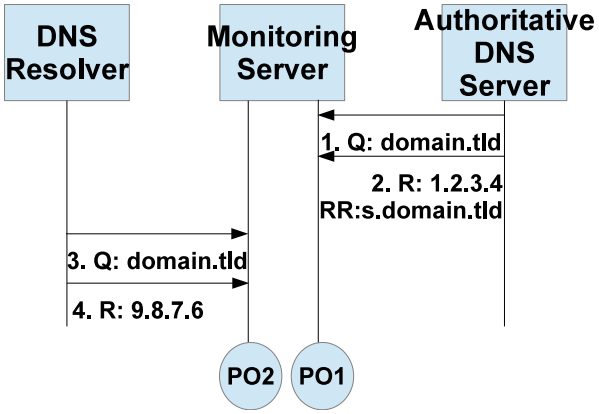

Figure 2: Monitoring server MSC

particular domain. The DNS resolver will use this modified caching DNS server as its DNS server. Traces can be collected in the DNS resolver and the authoritative DNS server. Those traces will contain the DNS packets exchanged with the peers. The DNS resolver will have only one peer, which is the caching DNS server. The authoritative DNS server will have many peers, all the caching servers querying for the domain. Note that any caching DNS server querying the authoritative DNS server can help provide information to that network trace. Furthermore, the caching DNS server that changes the records most probably will not be a peer in the authoritative DNS server trace. Also, for this case only DNS packets can be captured, all other traffic can be discarded. These traces can be provided by many tools like Wireshark [1]. The traces can then be exported at the monitoring server. The evaluation of the trust properties at the monitoring server will then provide the trust information.

We have formally defined two formulas in order to express trust properties. 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 specifications we can declare the necessary trust properties. While the properties have herein bee provided manually by an expert, they also can be generated automatically if a formal specification is available [29, 17]. After evaluating them we will provide trust verdicts applied to the previously described scenario. We describe the trust properties and then formalize them.

The first trust property is: Φ = *"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:

$\Phi = \{\forall_x(req\_f(x, ADS)) \rightarrow \exists_{y>x}(res\_f(y, x, ADS))\} \rightarrow \{\forall_a(req\_anf(a, y, ADS)) \rightarrow \exists_{b>a}(res\_enf(b, a, y, ADS))\}$

Where the intermediate clauses are defined such as:

$req(x) \leftarrow x.flags.response = 0$ $//req$ predicate compares the flag response and if the flag is unset this means packet x is a request.
$res(y) \leftarrow x.flags.response = 1$ $//res$ predicate compares the flag response and if the flag is set this means packet x is a response.
$eq\_q(x, y) \leftarrow x.queries = y.queries$ $//eq\_q$ predicate compares if the queries of the two packets are equal.
$eq\_a(x, y) \leftarrow x.answers = y.answers$ $//eq\_a$ predicate compares if the answers of the two packets are equal.
$from(x, P) \leftarrow x.PO = P$ $//from$ predicate compares if the packet comes from a specified point of observation P.
$nfrom(x, P) \leftarrow x.PO \neq P$ $//nfrom$ predicate compares if the packet comes from a PO different from P.
$after(x, y) \leftarrow x.time > y.time$ $//after$ predicate compares if the field value time of x is greater than the one of y.
$resp(y, x) \leftarrow res(y) \wedge y.query_id = x.query\_id \wedge eq\_q(x, y)$ $//resp$ predicate checks if the packet y is a response of packet x, comparing both $query\_ids$.
$req\_f(x, P) \leftarrow req(x) \wedge from(x, P)$ $//req\_f$ predicate compares if packet x is a request and if it is coming from a specified point of observation P.
$req\_nf(x, P) \leftarrow req(x) \wedge nfrom(x, P)$ $//req\_nf$ predicate compares if the packet x is a request and if it comes from point of observation different from P.
$res\_f(y, x, P) \leftarrow resp(y, x) \wedge from(y, P)$ $//res\_f$ predicate compares if packet y is a response of packet x and if packet y comes from the specified point of observation P.
$res\_nf(y, x, P) \leftarrow resp(y, x) \wedge nfrom(y, P)$ $//res\_nf$ predicate compares if the packet y is a response of packet x and if packet y comes from a PO different from P.
$req\_a(x, y) \leftarrow after(x, y) \wedge req(x) \wedge eq\_q(x, y)$ $//req\_a$ predicate compares if packet x occurred after packet y, packet x is a request and both packets queries are the same.
$req\_anf(x, y, P) \leftarrow req\_a(x, y) \wedge nfrom(x, P)$ $//req\_anf$ predicate checks if the request x is after the packet y and packet x comes from a point of observation different from P.
$res\_enf(x, y, z, P) \leftarrow res\_nf(x, y, P) \wedge eq\_a(x, z)$ $//res\_enf$ predicate checks if packet x is a response from packet y, x comes from a point of observation different from P and it compares if the answers of x and z are the same.

"ADS" is the assigned label to the Authoritative DNS server PO (see Section 4.1).

The second trust property is related to the DNS updates. An update in the DNS values will imply that caching DNS servers might have the wrong records. However, this is the expected behavior; the caching DNS servers can have the wrong values for the validity of the time to live (TTL) of each record. Even if this is the expected behavior, this means the caching DNS server has untrustworthy records. In this work we do not focus on how to measure or interpret the trust information we provide. Nevertheless, this information could be used by caching DNS servers to re-query the authoritative DNS server when a change in the records is performed. We will consider that a caching DNS server with the wrong DNS records can still be trustworthy if the following trust property holds: Ψ = *"For all responses from an authoritative DNS server, if it exists a future response from other points of observation that is not same response of the authoritative DNS server and the queries are the same. Then, a previous authoritative DNS response with the same value as the conflicting response must exist; also, its TTL should be bigger than the difference between the conflicting response and the previous authoritative response"*. This can be expressed by

[1]Wireshark homepage http://www.wireshark.org/

the following formula: $\Psi = \{(\alpha \rightarrow \beta) \rightarrow \gamma\}$

Where the intermediate clauses are:

$\alpha = \{\forall_x(req\_f(x, ADS)) \rightarrow \exists_{y>x}(res\_f(y, x, ADS)\}$
$\beta = \{\exists_a(req\_anf(a, y, ADS)) \rightarrow \exists_{b>a}(res\_dnf(b, a, y, ADS)\}$
$\gamma = \{\exists_m(req\_bf(m, a, ADS)) \rightarrow \exists_{n>m}(res\_eft(m, n, b, ADS))\}$

For which we define:

$ne\_a(x, y) \leftarrow x.answers \neq y.answers$ //ne_a predicate compares if the answers of packet x are different from the answers of packet y.
$bef(x, y) \leftarrow x.time < y.time$ //bef predicate compares if packet x occurred before packet y.
$ttl\_a(x, y) \leftarrow x.anwers.TTL+x.time > y.time$ //ttl_a predicate compares if the field value time to live of x plus the time at which the packet occurred is greater than the y occurrence.
$req\_b(x, y) \leftarrow bef(x, y) \wedge req(x) \wedge eq\_q(x, y)$ //req_b predicate checks if packet x occurred before packet y, x is a request and both packets have the same queries.
$req\_bf(x, y, P) \leftarrow req\_b(x, y) \wedge from(x, P)$ //req_bf predicate checks if packet x occurred before packet y, x is a request, both packets have the same queries and x comes from the specified PO P.
$res\_dnf(y, x, z, P) \leftarrow res\_nf(y, x, P) \wedge ne_a(y, z)$ //res_dnf predicate checks if packet y is a response from packet x, y comes from a PO different from P and the answers of y and z differ.
$res\_ef(y, x, z, P) \leftarrow res\_f((y, x, P) \wedge eq\_a(y, z)$ //res_ef predicate compares if packet y is a response of packet x, y comes from P, and the answers of y and z are the same.
$res\_eft(y, x, z, P) \leftarrow res\_ef(y, x, z, P) \wedge ttl_a(y, z)$ //res_eft predicate checks if packet y is a response from x, y comes from P, compares if the answers of x and z are the same and if y time to live plus the time at which the packet occurred is greater than the time packet z occurred.

We manually applied Φ and Ψ trust properties on a small network trace containing DNS packets. For this particular network trace all results from the Φ trust property were $\top$ ('*Pass*'). This means that on that network trace no DNS updates or untrustworthy responses were provided.

6. CONCLUSION AND PERSPECTIVES

In this paper, we propose a formal distributed approach to monitor and test trust properties by evaluating trust objectives on real distributed traces. While most of the approach are based on local probes (points of observations or interfaces), we define in this work a correlation of the testing verdicts to evaluate the trust in distributed ecosystems. A formal syntax and semantics are defined to express trust properties which are applied on DNS traces through a distributed use case. Interesting and promising results are provided.

While our results have been obtained manually, we are currently extending one of our tool, datamon[2] to consider the distributed points of observations. This tool allows to take into account a set of functional properties (not some trust ones) to check them on a single trace. This should be able, soon, to take as inputs a set of trust properties and a set of execution traces contained in our monitoring server. Furthermore, we note that 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 points of observation and compare the trust properties against stored statistical results from these multiple points of observation. It would decrease the delay between the observations and then the provided verdicts.
Finally, we plan to apply our approach to other digital ecosystems to test trust properties labelled to diverse agents.

[2] `http://www-public.int-evry.fr/~lalanne/datamon.html`

7. REFERENCES

[1] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Dns security introduction and requirements. RFC 4033 (Proposed Standard), 2005.

[2] M. Blaze, J. Feigenbaum, and A. D. Keromytis. Keynote: Trust management for public-key infrastructures. In *Proceedings of the Security Protocols, 6th International Workshop, Cambridge, UK*, pages 59–63. Springer, 1999.

[3] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, USA*, pages 164–173, 1996.

[4] H. Boley and E. Chang. Digital ecosystems: Principles and semantics. In *Proceedings of the Digital EcoSystems and Technologies Conference*, pages 398–403, 2007.

[5] X. Che, F. Lalanne, and S. Maag. A logic-based passive testing approach for the validation of communicating protocols. In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE, Wroclaw, Poland*, pages 53–64. SciTePress, 2012.

[6] Y.-H. Chu, J. Feigenbaum, B. Lamacchia, P. Resnick, and M. Strauss. Referee: Trust management for web applications. *O'Reilly World Wide Web Journal*, 2(3):127–139, 1997.

[7] D. Dagon, N. Provos, C. P. Lee, and W. Lee. Corrupted dns resolution paths: The rise of a malicious resolution authority. In *Proceedings of the Network and Distributed System Security Symposium, NDSS, San Diego, California, USA*. The Internet Society, 2008.

[8] S. Deering and R. Hinden. *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification*. Internet Engineering Task Force, 1998.

[9] L. Fan, Y. Wang, X. Cheng, and J. Li. Prevent dns cache poisoning using security proxy. In *Proceeding of IEEE 12th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2011, Gwangju, Korea*, pages 387–393, 2011.

[10] T. Grandison and M. Sloman. A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials*, 3(4):2–16, 2000.

[11] T. Grandison and M. Sloman. Trust management tools for internet applications. In *Proceedings of Trust Management, Springer First International Conference, iTrust, Heraklion, Crete, Greece*, pages 91–107, 2003.

[12] D. A. Haidar, N. Cuppens-Boulahia, F. Cuppens, and H. Debar. Xena: an access negotiation framework using xacml. *Annales des Télécommunications*, 64(1–2):155–169, 2009.

[13] T. Jim. Sd3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy, Oakland, California, USA*, pages 106–115, 2001.

[14] F. Lalanne and S. Maag. A formal data-centric approach for passive testing of communication protocols. *IEEE/ACM Trans. Netw.*, 21(3):788–801, 2013.

[15] A. J. Lee., M. Winslett, and K. J. Perano. Trustbuilder2: A reconfigurable framework for trust negotiation. In *Proceedings of Trust Management III, Third IFIP WG 11.11 International Conference, IFIPTM, West Lafayette, IN, USA*, pages 176–195, 2009.

[16] C.-C. Lo, C.-C. Huang, and J. Ku. A cooperative intrusion detection system framework for cloud computing networks. In 280-284, editor, *Proceedings of the IEEE 39th International Conference on Parallel Processing Workshops*, 2010.

[17] S. Maag, C. Grepet, and A. R. Cavalli. A formal validation methodology for manet routing protocols based on nodes' self similarity. *Computer Communications*, 31(4):827–841, 2008.

[18] S. Maag and F. Zaïdi. Testing methodology for an ad hoc routing protocol. In *PM2HW2N*, pages 48–55, 2006.

[19] M. E. Maarabani, A. Cavalli, K. Toumi, and C. Andres. A vector based model approach for defining trust in multi-organization environments. *2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS)*, 0:1–8, 2012.

[20] S. P. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, Stirling, Scotland, UK., 1994.

[21] D. L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991.

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

[23] Z. Movahedi, M. Nogueira, and G. Pujolle. An autonomic knowledge monitoring scheme for trust management on mobile ad hoc networks. In *IEEE Wireless Communications and Networking Conference, WCNC 2012, Paris, France*, pages 1898–1903, 2012.

[24] I. Ray and S. Chakraborty. A vector model of trust for developing trustworthy systems. In *Computer Security - ESORICS, 9th European Symposium on Research Computer Security, Sophia Antipolis, France*, pages 260–275. Springer, 2004.

[25] S. Roschke, F. Cheng, and C. Meinel. A flexible and efficient alert correlation platform for distributed ids. In *Proceedings of the IEEE Fourth International Conference on Network and System Security, NSS, Melbourne, Victoria, Australia*, pages 24–31, 2010.

[26] K. Toumi, C. Andrés, and A. R. Cavalli. Trust-orbac: A trust access control model in multi-organization environments. In *Proceedings of Information Systems Security, 8th International Conference, ICISS, Guwahati, India*, pages 89–103, 2012.

[27] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[28] S. T. Zargar, H. Takabi, and J. B. D. Joshi. Dcdidp: A distributed, collaborative, and data-driven intrusion detection and prevention framework for cloud computing environments. In 332-341, editor, *Proceedings of IEEE 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom, Orlando, FL, USA*, 2011.

[29] M. Zhigulin, N. Yevtushenko, S. Maag, and A. R. Cavalli. Fsm-based test derivation strategies for systems with time-outs. In *QSIC*, pages 141–149, 2011.