Attack-resilient Compliance Monitoring for Large Distributed Infrastructure Systems

Mirko Montanari, Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
{mmontan2, rhc}@illinois.edu

***Abstract*—The security of monitoring systems is critical for maintaining an accurate view of the state of infrastructure systems such as enterprise networks and critical infrastructure systems. A malicious user that controls a monitoring system has the ability of delaying the detection of security attacks and sabotages, and can acquire information about the infrastructure that can enable additional attacks.**

In this paper we present a distributed architecture that increases the resilient of monitoring systems to attacks against their availability, integrity, and confidentiality. Our approach is based on distributing the knowledge of the state of the infrastructure to a large number of non-dedicated servers, so that the compromise of any limited number of hosts does not cause a compromise of the entire monitoring system. We present an algorithm able to integrate information across the distributed servers to evaluate complex security policies. We analyze the security properties of our approach, and we experimentally evaluate the performance and the resilience of our architecture. We show that, compared to current solutions, our solution increases the resilience of a monitoring system while reducing the load on each monitoring machine.

I. Introduction

Monitoring the compliance of infrastructure systems to security policies has been identified as a critical part of the risk management process of large organizations [1]. Infrastructure security policies define the proper operational conditions of the infrastructure. Their violation indicates that the infrastructure is operating in undesirable conditions that could create opportunities for attacks. Organizations specify a large number of infrastructure security policies for controlling the configurations of the network systems they manage. The National Institute for Standard and Technology (NIST) provides a large list of security policies for government systems [2], and the North America Electric Reliability Corporation (NERC) provides security policies for organizations controlling power grid assets [3]. For example, NERC policies specify that critical assets need to be placed in electronic security perimeters, and that all access points of the infrastructure need to be monitored and logged. Online compliance monitoring allows detecting quickly when changes in configurations, failures, or operator errors create policy violations.

As policy-compliance monitoring systems become a critical part of the infrastructure, they also become a new target for attacks. Malicious users controlling a monitoring system could hide policy violations to sabotage the infrastructure operations without being detected and to enable further attacks. In this paper we introduce an architecture for monitoring policy compliance in large infrastructure systems able to operate with limited security consequences even when part of the monitoring servers are under the control of malicious users. We distribute the compliance validation process to a large number of servers so that the compromise of any limited number of servers has little consequences on the integrity and the confidentiality of the monitoring system. Intuitively, we exploit the fact that the detection of each specific violation of a policy requires to integrate only a limited amount of information about the infrastructure state. Our architecture is able to tolerate compromises on monitoring elements with limited effects on the monitoring capabilities and on the knowledge that a malicious user obtains about the system. Validation of complex policies is performed by integrating information redundantly across a large number of hosts while maintaining low processing, memory, and bandwidth loads on each of them. Because of this property, we can substitute a small number of machines exclusively dedicated to monitoring with a large number of non-dedicated machines such as web servers and file servers.

The contribution of the paper is summarized as follows.

1) We present an architecture for compliance monitoring that tolerates attacks on its components.
2) We introduce a distributed algorithm for validating compliance which distributes the load across a large number of non-dedicated servers.
3) We perform a security evaluation and an experimental evaluation of our architecture and algorithm.

The rest of the paper is structured as follows. Section II describes the relation between this work and previous work in the area. Section III introduces policy compliance and our representation of state and policies. Section IV presents our architecture and algorithm. Section V provides our experimental results. Finally, Section VI concludes our work.

II. Related Work

The introduction of compliance requirements by government entities has been driving industry and research in the development of automated tools for assessing compliance. A recent report from NIST specified that continuous monitoring of the security and compliance of the infrastructure is a critical part in the security of US government systems [1]. The North America Reliability Council (NERC) published several policies that specify allowed configurations of devices that are part of the US power grid. The Payment Card Industry Data Security Council published standards to reduce exposure to compromises for organizations managing payment card data [4].

While the overall process of compliance-validation is based on periodic manual auditing, several tools have been introduced by the industry and the research community for partially automatic this process. NIST published standards for formally specifying security policies [2] and for creating tools that monitor compliance. However, such tools only analyze the configuration of single machines and are not able to monitor for policies which require aggregating information across several machines. Vulnerability assessment tools [5] are able to scan the network, identify vulnerable software, and can use attack graph [6] to evaluate the impact of such vulnerabilities on the security of the system. Our system is able to obtain such information and use similar algorithms for the computation of the impact of the vulnerability in the system. However, the network vulnerability scanner architecture is based on a few trusted servers that perform the monitoring. The compromise of one (or more) of such servers could compromise the robustness of the assessment process.

Previous work proposed another architecture for the monitoring of compliance based on the delegating of the monitoring tasks to end-nodes [7]. Such a system is still based on a limited number of redundant servers for aggregating information: the compromise of such server would reveal all information about the state of the system and could hide violations.

III. Policy Compliance

Managing the security large-scale infrastructure systems such as enterprise network systems, airport systems, or power grid systems is a complex task because of the large number of devices and the large number of possible configurations of the system. The definition of infrastructure security policies can simplify the management by providing a base set of rules that needs to be respected at all times in the configurations and the state of the infrastructure. For example, security policies in enterprise network systems can define the type of communications allowed between different parts of the network or the programs that it is possible to run on each machine. Additionally, policies can specify conditions that represent correct or incorrect behavior of the system. For example, a policy in an airport system can specify that aircrafts are expected to connect to the airport wireless network upon landing for downloading software updates and for uploading aircraft heath data.

Violations of policies have consequences on the security, reliability, and efficiency of the infrastructure. Violations of enterprise security policies might open the system to attacks (e.g., by allowing unauthorized programs to be run on critical machines), and violations of other policies might reduce the safety of the infrastructure (e.g., undetected errors in the aircraft connections could delay maintenance operations). While compliance to a specified set of policies cannot guarantee security, it still provides a minimum level of security to the infrastructure by providing protection from attacks that would have been avoidable had proper security measure been taken.

Checking for compliance to policies requires integrating information across several devices. For example, the airport policy we presented requires integrated information about the position of the aircraft provided by onboard devices and information provided by airline applications. An enterprise policy that requires computing attack graphs to detect devices that can be compromised needs to integrate information about all hosts in the network. The rest of the paper describes how integrating this information does not necessarily require storing all information in a central location but can be performed redundantly across a large number of machines.

A. Infrastructure Policy

Logic, in particular Datalog, has been used extensively for representing the heterogeneous information that composes the security state of infrastructure systems. Zahid et. al. [8] represents state and policies using first-order logic. Ou et. al. use Datalog for representing attack-graphs based policies [6]. In this work we use the Resource Description Framework (RDF) for modeling the state of the system, and we use Datalog rules for representing policies [9].

The validation of each policy requires acquiring specific information about the state of the system. In our architecture, entities called *sensors* collect such information and send it a set of *monitoring servers* for analysis. Atomic information about the state and configurations are represented using RDF statements. Each RDF statement provides information about a specific resource, called *subject* of the statement. The type of information is identified by the second element of the statement, the *predicate*. The third element, called *object* identifies either a constant value (called *literal*) or another resource. For example, the fact that a device d is running a software s is represented by the statement $(d, runs, s)$.

Infrastructure policies define constraints over the admissible combinations of statements that are present in the state. We express policies as Datalog rules. A rule is composed of two parts, a body and a head. The body specifies a condition over the state, and the head specifies the conclusion that can be drawn when the body of the rule is true. This conclusion can either be the fact that a violation is present, or the fact that another statement about the state is true.

The body and the head of the rule are expressed as a conjunction of *statement patterns*. A statement pattern is a statement where subject or object are *variables*. We use the standard definition of Datalog unifications for defining when a statement pattern *matches* a statement. i.e., a set of statement patterns matches a set of statements if there exist a substitution of the variables in the statement patterns that makes them equals to the set of statements.

1) Example of Infrastructure Policies: In the aerospace domain, the increasing reliance of modern aircrafts on the airport infrastructure requires such systems to operate securely, safely, and efficiently. Infrastructure policies can be used to monitor that the infrastructure is operating in a good state. For example, the new e-Enabled fleets [10] require services that need to be provided by the airport infrastructure, such as Internet connectivity, aircraft software update services, and access to maintenance services. Several examples of policies can be defined in this context. We can mandate that approaching airplane must establish wireless link after they land (i.e., weight-on-wheels condition) so that software updates can be downloaded as following:

$$(A,\texttt{landed},R),(R,\texttt{part\_of},AIR),(N,\texttt{part\_of},AIR),\\ \neg(A,\texttt{connected\_to},N)\rightarrow(\mathit{policy}_1,\texttt{violation},A), \tag{1}$$

where the statement $(A, \texttt{landed}, R)$ indicates that the aircraft A landed on runway R, the statement $(K, \texttt{part\_of}, J)$ indicates that an entity K is part of the system J, and the statement $(A, \texttt{connected\_to}, N)$ indicates that the system A is connected to a network N. The consequence of the rule, $(\mathit{policy}_1, \texttt{violation}, A)$ indicates that a violation of policy_1 has been detected and it involves the aircraft A.

Additionally, we can monitor that airplane must be parked at gate when accessing certain airline applications, as following:

$$(A,\texttt{logged\_on},P),(P,\texttt{provided\_by},M),\\ (M,\texttt{part\_of},\mathit{airline}),(P,\texttt{use},\texttt{gate\_restricted}),\\ \neg(A,\texttt{located\_at},L),(L,\texttt{location\_type},\texttt{gate})\\ \rightarrow(\mathit{policy}_2,\texttt{violation},A), \tag{2}$$

where the statement $(A, \texttt{logged\_on}, P)$ indicates that the aircraft A is accessing application P, $(P, \texttt{provided\_by}, M)$ indicates that the host M is running the application P, $(A, \texttt{locate\_at}, L)$ states that the aircraft A is located at location L. The last two statements, $(L, \texttt{location\_type}, \texttt{gate})$ and $(P, \texttt{use\_restricted}, \texttt{gate})$ state that L is a airport gate, and that P can be used only by an aircraft located at the gate.

All these example policies require integrating information across multiple devices for assessing compliance. The first policy requires integrating information from the aircrafts or the control tower with information generated by network monitoring devices, while the second policy requires integrating again information from the aircraft and information generated by airline applications. However, none of these policies requires access to the entire state of the system. For example, both policies can be validated independently for each aircraft: all devices that have information about a specific aircraft send information to the same node. Each single group of information is independently sufficient for assessing all possible compliance validations in the system. Our architecture takes advantage of this intuition for distributing the validation redundantly across multiple hosts.

IV. Robust Monitoring Architecture

The monitoring process is composed of two parts: acquiring information about the state of the system, and integrating it to validate compliance to the policies. In our architecture, two different entities perform these tasks. *Sensors* monitor the state of the system, and *monitoring servers* collect information and validate compliance.

Sensors are software agents or hardware devices that acquire the information used for assessing policy compliance. For example, a software agent can act as a sensor for acquiring information about the running processes in every machine.

In an airport infrastructure, software agents and hardware devices located on aircrafts provide information about the aircraft location, while applications running on the airline servers provide information about access. To assure security and reliability, each software agent sends information to several monitoring servers. Additionally, redundant sensors should monitor the different parts of the state of the system (e.g., host-based monitoring and traffic flow monitoring can be used to detect communications between hosts).

Monitoring servers keep a local knowledge base that stores their partial view about the state of the system. Such knowledge base contains the information received from the sensors and partially analyzed information provide by other monitoring servers during the compliance validation process. At a high level, the validation is performed by creating aggregation trees that connect servers which contain information useful for the detection of each violation. At each step of the aggregation, a portion of the policy is verified and result of the partial validation is forwarded to other monitoring servers.

The structure of the communication between monitoring servers is define by an analysis of the policies of the infrastructure. To distribute the load of the validation across multiple servers, we distribute the task of validating each resource (e.g., a process, a device, a user) to a redundant set of monitoring servers. All policy validations that involve one of these resource are handled by such a set of monitoring servers.

A compilation process transforms policies into multiple redundant aggregation trees. The nodes in the trees are called *rendezvous points* and represent resources. The edges represent a need of exchanging information between the monitoring servers associated to each resource. State information is sent from the leaves to the root. At each step, rendezvous points perform partial checks of the policy. Messages are forwarded toward the root only if they are relevant for detecting policy violations. When the messages reach the root, we have all information to validate compliance for the devices that are part of the tree. Once a violation is detected, the root sends a message to a system under the control of the administrators so that remedial actions can be taken.

The redundancy in the aggregation makes the compliance validation process resilient to a limited number of compromises of the monitoring servers, and reduces the impact of larger compromises. The distribution of the resources across multiple monitoring servers reduces the amount of information

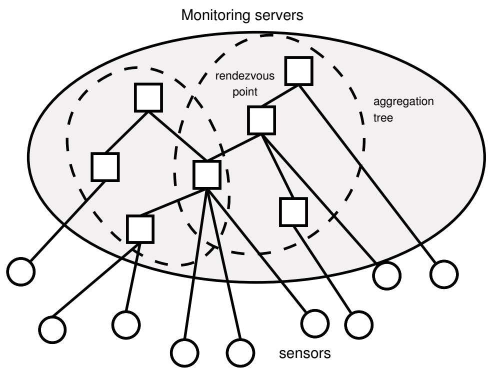

Fig. 1. Example of an instance of the Dora architecture with no replication. Devices are represented by circles and send information to rendezvous nodes. Rendezvous points are shared by multiple trees, when possible.

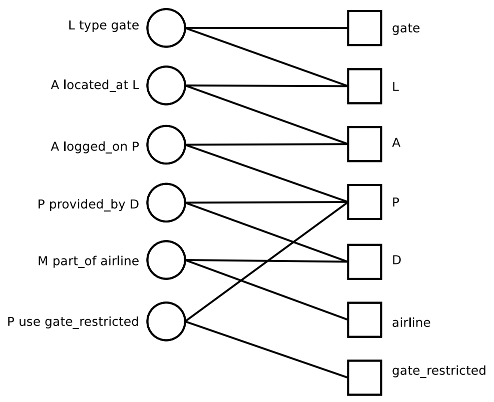

Fig. 2. Conversion of Rule 2 into the bipartite graph.

about the infrastructure state that is revealed when monitoring servers are compromised.

A. Distributed Online Rule Analysis Algorithm

The architecture of our system is shown in Fig. 1. The Distributed Online Rule Analysis (DORA) algorithm coordinates the exchange of information across monitoring servers for obtaining a complete assessment of the policy violations present in the infrastructure. Given the Datalog rules representing policies, it computes a set of aggregation trees across monitoring servers that integrate all the information required for detecting violations.

The algorithm is composed of two phases executed on every monitoring server: compilation and execution. The compilation phase transforms the set of rules into *rule elements* and *state triggers*. Rule elements represent the computation performed in each monitoring server, and communication elements represent the information exchanged across servers to ensure the completeness of the assessment.

During the execution phase, rule elements are added to each monitoring server's knowledge base to perform local inference, and communication elements are used by each monitoring server for selecting the statements to share and for identifying the rendezvous point to use as a destination for each of these statements. The communication and naming layer is provided by a DHT system. We use a secure DHT system to protect the communication layer from compromises.

Every time that a sensor sends new statements to one of the monitoring server, such statements can trigger new local inference and new communications. These updates are propagates across the aggregation trees and update the current compliance state of the infrastructure.

1) Compilation Phase: The compilation process converts a policy into a set of rule elements and a set of *state triggers*. A state trigger is a tuple (q, D) composed of a persistent query q over the local state of the device, and a variable D as *destination pattern* which must appear in the result of the query q. During execution, a state trigger performs the persistent query q to the local state and sends all its updates to the rendezvous points indicated by the value assigned to the variable D of each tuple in the result. The local rules and the state triggers are added to every device and provide the only policy-dependent elements required during execution.

The conversion goes through three steps. First, we explicitly represent the relation between rendezvous points and statement patterns by representing the rule as a bipartite graph. We label each edge with a number that represents the estimated number of messages sent or received by the rendezvous points. Second, we convert the bipartite graph into an aggregation-tree *pattern* by choosing a root and computing a spanning tree. Each choice of the root creates aggregation trees that differ on the resources that are used as rendezvous points. We select the tree that minimizes our cost function. Third, we convert the aggregation-tree pattern into local rules and state triggers.

The first step of the algorithm represents the relation between statement patterns and possible rendezvous points. We create a bipartite graph composed of two types of nodes: *variable nodes* for variables or resources, and *statement nodes*, which represent statement patterns contained in the policy. For each variable or resource in the policy, we create a variable node. For each statement pattern, we create a statement node. If a statement node uses a variable or resource as subject or object, it is connected to the corresponding variable node with an arc. Fig. 2 shows an example of such a bipartite graph.

The second step is the conversion of the bipartite graph in an aggregation tree. We select a variable node as root and we create a spanning tree. The selection of the root node can be performed using several heuristics like expected communication costs or expected amount of information that needs to be maintained in each host. However, for the scope of this paper, we choose as root the smaller node in lexicographic order. An example of aggregation tree is shown in Fig. 3.

The last step is the conversion of the aggregation tree into the rule set and the state triggers. We recursively perform a deep-first visit in the aggregation tree starting from the root. Statement nodes are converted into state triggers (q, D) by using their statement pattern as query q and the name of the parent as destination pattern D. Variable nodes are converted into local rules which have a body composed of the statement patterns of the children, and a state trigger that delivers the

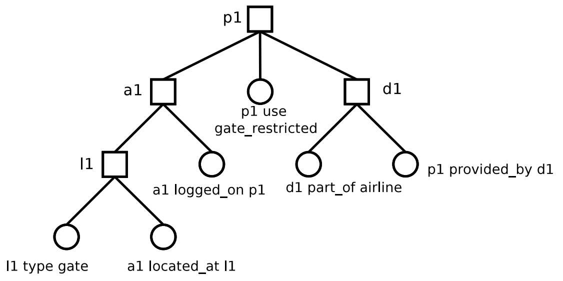

Fig. 3. Aggregation tree with root P generated from our example rule. Circles are devices sending state updates, square are rendezvous points. A different tree is generated for each $p_i \in P$

consequence of the rule to the rendezvous point up in the hierarchy. At the root, the state triggers send information to the network administrators. The pseudo-code of the algorithm is shown in Algorithm 1.

Algorithm 1 Pseudo-code for the Dora compilation algorithm

```
$n \in V \cup S$ {visited node}
$p \in V \cup S$ {parent node}
$r \in V$ {parent resource node}
$G := (V \cup S, A)$ {G is bipartite graph}
$addquery(q, n)$: query $q$ sending data to rendezvous $n$
$addrule(q, p, h)$: body $q$, head predicate $p$, head object $h$

function $build(n, p, r, G)$
if $n \in V \wedge p \neq \emptyset$ then
  $addquery(p, n)$
  Q = {};
  for all $(n, v) \in A$ do
    $G' = (V \setminus n \cup S, A \setminus (n, v))$
    $q = build(v, n, n, G')$
    $Q = Q \cup q$
    $addquery(q, n)$
  end for
  $p_{rule} = random()$
  $addrule(Q, p_{rule}, r)$
  $pattern = (\_, p_{rule}, p)$
  return $pattern$
end if
if $n \in S$ then
  $E = \{(v, n) \| v \in V\}$
  $G' = (V \cup S, A \setminus E)$
  return $build(v, n, r, G')$
else
  $E = \{(v, n) \| v \in V\}$
  $G' = (V \cup S, A \setminus E)$
  $Q = \{\}$
  for all $(n, v) \in A$ do
    $q = build(v, n, n, G')$
    $Q = Q \cup q$
  end for
  $p_{rule} = random()$
  $addrule(Q, p_{rule}$, violation$)$
  return $\emptyset$
end if
```

The body of the rule created at the variable node is formed by the patterns of the children statement nodes, and the patterns generated by the consequences of the rules at the variable nodes two steps down in the tree. The consequence of the rule represents the result of the partial evaluation of the policy. The head of the local rule contains statements that keep track of all information about the matching that need to be forwarded to higher levels of the tree. If only true/false information needs to be preserved, then the blank node contains a single predicate with a true/false object. If more information about variables in the matching needs to be preserved, then the blank node has predicates to preserve this information. For example, a rule (A, p_1, B), (B, p_2, C), $(C, p_3, D) \rightarrow (C, \texttt{violation}, \texttt{true})$ can be processed by two rendezvous nodes associated with the values of the variables B, and C. When analyzing the rendezvous node B, we generate a rule $(A, p_1, B), (B, p_2, C) \rightarrow (rule_1, value_1, C)$. At the rendezvous node C, the rule we generate is $(rule_1, value_1, C), (C, p_3, D) \rightarrow (C, \texttt{violation}, \texttt{true})$.

2) Execution Phase: All rules and state triggers generated by the compilation phase are added to the KB of all devices. We do not need to determine a-priori which devices are going to be part of an aggregation tree: each device need to only reacts to the messages that are sent to it.

During the execution phase, sensors send their state and state update information to monitoring server. A deployment parameter, called *replication factor* defines the number of monitoring servers at which information is delivered. For a replication factor f, sensors send their information to f monitoring server. Each receiving server adds this information to its local knowledge base and considers each state trigger (q, D). For each trigger, we perform the query q on the state KB and we send each returned tuple to the rendezvous points identified by the value of the variable D of the tuple. For a rendezvous point $D = u$, the update message is sent to f redundant monitoring servers identified by a family of functions $H_i(u)$ with $i = 0, \ldots, f-1$. When the KB is modified, the result of the persistent queries changes and triggers the sending of update messages to rendezvous points.

When a device receives a state update from at least $\lceil f/2 \rceil$ monitoring servers, it adds the received statement to its local KB. With the new data available, local rules can trigger new inference, which can create changes in the persistent queries results. These changes are sent to the respective rendezvous points according to the state triggers. When the update message reaches the root of the tree and a violation is detected, the notification is broadcasted to all monitoring servers.

At every step of the tree, the DHT routing delivers the message to the monitoring servers identified by each $H_i(u)$. Different monitoring servers can act as forwarding nodes for such messages. These servers maintain a set of KBs, called *forwarding KBs*, that keeps track of the messages forwarded to each of the destinations $H_i(i)$. Failures or the introduction of new monitoring servers can change the next hop for each destination. When this happens, the forwarding nodes sends the forwarding KB to the next hop so that the complete state can be reconstructed.

Additionally, we need to ensure that the compromise of a forwarding node cannot compromise multiple aggregation trees: if a single compromised node is the next-hop in the routes to the majority of rendezvous points, the aggregation process for that policy violation is compromised. To avoid this situation, we uniformly partition the space of the monitoring servers in f disjoint groups based on the initial digits of the id. We ensure that each aggregation tree is completely contained within each of these groups by using a family of functions $H_i(u)$ that consistently maps H_i and H_j, with $i \neq j$, in different parts of the space.

B. Security of the monitoring system

Protecting the security of a monitoring system requires protecting its availability, integrity, and confidentiality properties.

Compromises of these properties lead to a reduced level of security of the infrastructure being monitored. In particular, a compromise of availability allows an attacker to act undetected in the system. For example, an attacker could disable the monitoring system and make the infrastructure operate in unsafe conditions. A compromise of the integrity increases the capabilities of the attacker: specific violations can be hidden and the state of the system visible to administrator altered. Additionally, spurious violations could be generated to confuse response to emergencies. Compromise of the confidentiality of the monitoring system provides attackers a view of the infrastructure configuration and state. This information can be used for planning complex attacks. We compare the security of our solution with a centralized replicated solution. We estimate the effects of attacks and we show that our solution offers increased resilience to attacks toward the availability, integrity, and confidentiality of the monitoring system.

We first analyze resilience toward attacks to the confidentiality of the system. We assume an attacker able to compromise monitoring servers with some effort. To represent this concept, we assume that an attacker needs to spend an effort E proportional to the number of servers compromised. We consider the state of the system composed of s statements. Such describe r resources, each of them described by a set of statements s_r. We assume that each monitoring server is either a resource node or a forwarding node in t aggregation trees. We consider both solutions with a replication factor f.

In a replicated centralized solution, even with one compromised server, confidentiality of the state of the system is completely compromised. i.e., all information about the system state becomes accessible by malicious users. In the Dora architecture, compromise of a monitoring server leaks to the attacker only a limited amount of information: statements about the resource managed by the server and statements about the aggregation trees that a have the monitoring server as a node. We consider two cases: the case in which attackers desire to obtain as much information as possible about the state (i.e., attackers searching for possible exploits) and the case in which attackers target a specific statement about the system (i.e., targeted attack). In a centralized solution is sufficient to compromise one of the servers with an effort E for acquiring the entire state of the system. In our solution the information about the state is distributed and replicated across monitoring servers. To reconstruct the entire state, attackers need to be compromise a number of servers proportional to n/f with a significantly higher effort of $E\lceil n/f \rceil$. If attackers are targeting a specific statement or a specific resource, the effort required in both solution is the same: attackers need to compromise one of the replicated servers with an effort E.

Resilience toward integrity and availability compromises can be analyzed in a similar fashion. Again, we compare the two solutions for the case in which attackers targeting to control as much violations as possible and for the case in which attackers are targeting to hide or to introduce a violation.

The number of possible violations in an infrastructure depends on the number of policies and their length. For a system with n resources and f policies of average length l, the maximum amount of different violations that it is possible to generate is fn^l (i.e., each possible combination of resources for the length of the policy).

For the centralized solution, the compromise of a number of monitoring servers greater than the majority leads to a complete compromise of the system. Hence, the number of violations that it is possible to generate or hide goes from 0 for an attacker that does not have control of the majority of monitoring nodes, to n^l for an attacker that has control of at least $c > n/2$ monitoring nodes. Hence, an attacker is able to control n^l violations with an effort proportional to $\lceil f/2 \rceil E$.

In the Dora architecture, obtaining control of a violation requires controlling the majority of the rendezvous nodes managing a specific resource, or the related forwarding nodes. The effort required to perform this operation is proportional to $\lceil f/2 \rceil E$. Introducing an arbitrary violation is more complex, as attackers need to control all the resources involved with an effort I of $\lceil f/2 \rceil E$ in the corner case where all resources are mapped to the same node, and $l\lceil f/2 \rceil E$ when the l resources are mapped to different nodes. Controlling the entire set of violations requires controlling the majority of rendezvous nodes for all resources. As rendezvous nodes assigned to different trees are mapped to different servers, the effort for compromising all violations is proportional to $\frac{n}{\lceil f/2 \rceil}E$.

In summary, our architecture significantly increases the effort required for successfully performing attacks that compromise large portions of the monitoring systems, while still providing the same level of protection to targeted attacks.

V. Experimental Results

In this section we measure the ability of our monitoring system to distribute the load of monitoring across several machines, and its ability of resisting to attacks directed toward it. Our implementation of Dora is built using Java 1.5. The communication layer is based on the Freepastry 2.1 library. The rule-base reasoning engine is provided by the Jena Semantic Web Framework[1].

We tested our rule analysis algorithm on policies taken from previous work. In particular, we analyzed policies that require creating attack graphs [6], policies encoding NIST and NERC rules that require distributed information integration [9], and policies defined for the aerospace domain [10]. We found that most policies involve a number of resources between 1 and 5. Even if this number depends on the representation of the configuration we used, we believe that the choice of different representations will change this number of only slightly.

We run our architecture in a small network for testing its functionalities, and we analyze the fundamental characteristics of our architecture by running our system on top of the freepastry event-based simulator. We analyze the system's behavior when parameters change. To better parameterize the characteristics of the system, we create a synthetic data set of configurations and policies. We associate a parameter *length*

[1]http:///www.freepastry.org, http://jena.sourceforge.net.

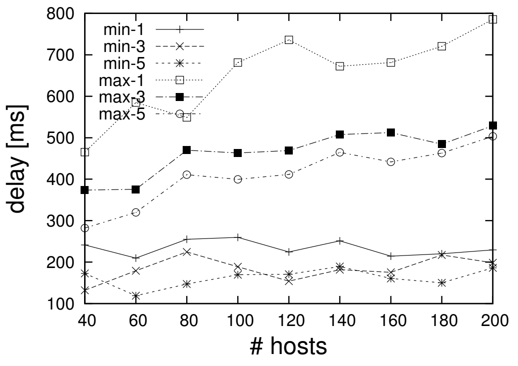

Fig. 4. Delay in the detection of violations over the number of servers. We consider a rule length of 3 resources.

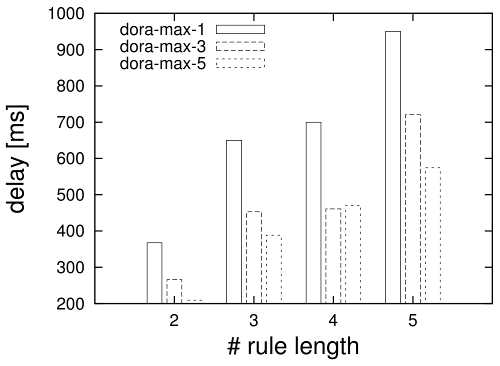

Fig. 5. Delay [ms] in the detection of violations over the length of the policy. We consider a network of 100 hosts.

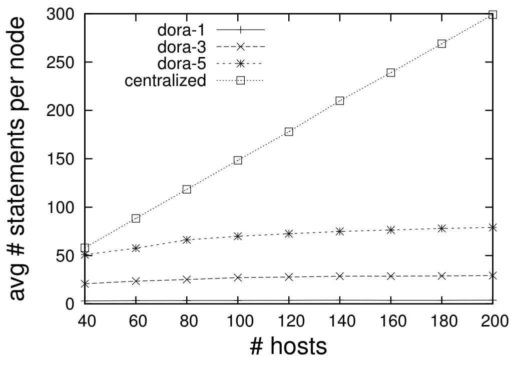

Fig. 6. Statements stored in each server over the size of the infrastructure. We consider a rule length of 3 resources.

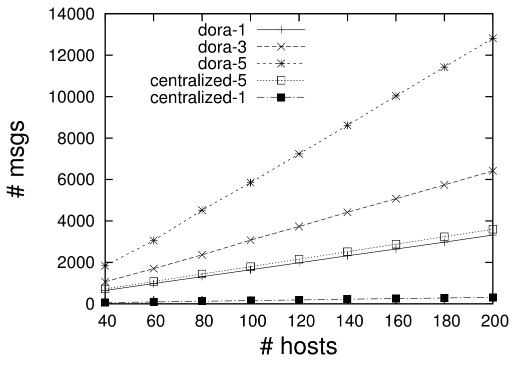

Fig. 7. Total number of messages in the system as a function of the size of the network. We consider a rule length of 3 resources.

to policies which represents the number of resources involved in its evaluation. Policy are expressed as chains of statement $p_1(a, b), p_2(b, c), p_3(c, d) \rightarrow violation(a, d)$. Long policies are representative of complex policies that need to integrate information across several resources.

A. Performance experiments

We compare the different performance dimensions of our system with the performance of a centralized monitoring system. The parameters of the two systems are set so that each device in the infrastructure communicates with the same number of monitoring servers. For example, in a triple redundant system for monitoring, each device sends its monitored events to three monitoring servers. In our architecture, we use three redundant paths so that each device sends data to three different monitoring servers.

The first set of experiments measures the overhead introduced by our monitoring architecture. We measure the overhead in term of memory, communication, and delay. To quantify the memory overhead, we measure the state information stored in each monitoring server. For simplicity, we considered a network with the same number of devices and monitoring servers. We find that the Dora architecture is successful in distributing statements across monitoring servers, and that each monitoring server only needs to store locally a very limited amount of state information. We show these results in Fig. 6. Additionally, we measure the communication overhead introduced by the Dora algorithm. Even if Dora introduces more message exchanges, we find that the amount of communication grows linearly with the size of the infrastructure as in the centralized solution. Additionally, the overhead introduced by Dora remains acceptable as the load is distributed across several monitoring servers. These data are shown in Fig. 7. The delay in detecting messages also remains in acceptable limits. We measure the minimum and maximum delay in detecting policy violations for different network sizes, different policy lengths, and different amount of replication. The results are shown in Fig. 4 and Fig. 5. Introducing replication reduces the min and max delay as the larger number of rendezvous points reduces the possibility that a few slow communication links slow down the entire process.

B. Security evaluation

The second set of experiments measures the robustness of our solution to confidentiality and integrity attacks. We compare our solution to a replicated centralized architecture.

We estimate the robustness of the architecture to confidentiality attacks by measuring the amount of state information provided to attackers when a set of random nodes is compromised. We assume that the compromise of a server provides attackers all information contained in it (i.e., local KB and forwarding KB). We measure the amount of information

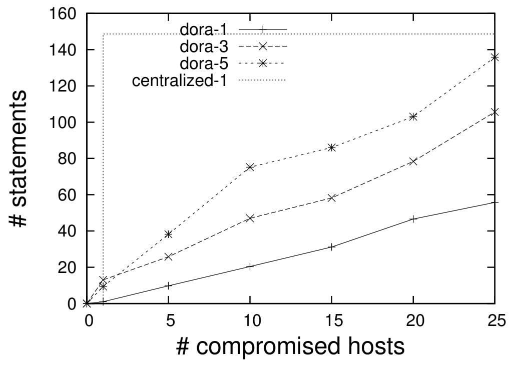

Fig. 8. Number of statements about the state of the system obtained by the attacker when x nodes are compromised. We consider a network of 100 hosts and a rule length of 3 resources.

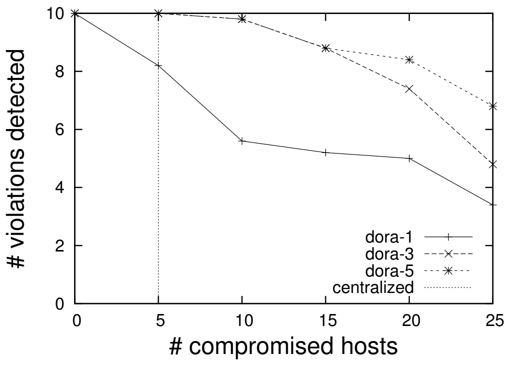

Fig. 9. Number of violations detected when x hosts are compromised. We consider a network of 100 hosts and a rule length of 3 resources.

leaked as the number of distinct statements acquired by the attacker. We find that, in the Dora architecture, the amount of statements acquired grows linearly with the number of compromised machines. In the centralized system, the entire information about the state is compromised as soon as one server is compromised. These results are shown in Fig. 8.

We estimate the robustness of the architecture to attacks toward the monitoring integrity by measuring its ability to operate when part of the monitoring machines is compromised. We focus on hiding violations and we implement compromised machines as machine dropping application packets so that statements matching policies are not detected. We perform a set of experiments where we randomly select monitoring servers to compromise. The performance of our system degrades gratefully as the number of compromised machines increases. In a centralized architecture, the transition from a safe monitoring system to a compromise monitoring system is abrupt. Once the few machines that maintain the replicated view of the system are compromised, the entire state of the system cannot be trusted anymore. The results of these experiments are shown in Fig. 9.

In summary, we show that our architecture provides a little overhead in each of the machine that are part of the monitoring system, and increases robustness to compromises.

VI. Conclusions

We presented a robust architecture for validating compliance to security policies in large-scale systems. The task of aggregating the system state and validating its compliance is distributed across several devices so that no single server stores the entire state of the system, and so that a limited number of compromised devices cannot affect the validation process by hiding violations or introducing fictitious violations. Our evaluation shows how the load introduced in the infrastructure by the monitoring system is low, and how the robustness to confidentiality and integrity compromises is increased.

In our future work we will focus on several issues. First, we will focus on improving the Dora algorithm by introducing different optimization functions (e.g., reduce information stored on each node) in the choice of the root node during the compilation process. Second, our algorithm does not include a concept of time: causal relations between events are currently ignored, and this could create a false positives or false negatives for a short period of time. While compliance validation focuses on long-lived violations and temporary conditions does not present a problem, a general monitoring system would benefit from the ability of tracking these relations. Third, we plan to deploy our architecture in a distributed infrastructure to validate the results obtained in our simulations.

Acknowledgements

This material is based on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

This work was partially supported by the Boeing Trusted Software Center.

References

[1] K. Dempsey, A. Johnson, A. C. Jones, A. Orebaugh, M. Scholl, and K. Stine, "Information Security Continuous Monitoring for Federal Information Systems and Organizations," NIST, Tech. Rep., 2010.

[2] Joint Task Force Transformation Initiative, "Recommended Security Controls for Federal Information Systems and Organizations - SP 800-53," NIST, Tech. Rep. August 2009, 2009.

[3] North American Electric Reliability Corporation, "NERC CIP 002-009," NERC - North American Electric Reliability Corporation, Tech. Rep., 2007.

[4] Payment Card Industry Security Standards Council, "Payment Card Industry (PCI) Data Security Standard," Tech. Rep. October, 2010.

[5] Tenable Network Security, "Nessus: the Network Vulnerability Scanner," 2009. [Online]. Available: http://nessus.org/nessus/

[6] X. Ou, W. Boyer, and M. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, p. 345.

[7] M. Montanari, E. Chan, K. Larson, W. Yoo, and R. H. Campbell, "Distributed Security Policy Conformance," in *IFIP International Information Security Conference*, 2011.

[8] Z. Anwar and R. Campbell, "Automated Assessment of Compliance with Security Best Practices," *Critical Infrastructure Protection II*, vol. 290, pp. 173–187, 2009.

[9] R. H. Campbell and M. Montanari, "Multi-Aspect Security Configuration Assessment," in *ACM Workshop on Assurable & Usable Security Configuration (SafeConfig)*, 2009.

[10] M. Montanari, R. H. Campbell, K. Sampigethaya, and M. Li, "A Security Policy Framework for eEnabled Fleets and Airports," in *IEEE Aerospace Conference*, 2011.