Confidentiality of Event Data in Policy-based Monitoring

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

***Abstract*—Monitoring systems observe important information that could be a valuable resource to malicious users: attackers can use the knowledge of topology information, application logs, or configuration data to target attacks and make them hard to detect. The increasing need for correlating information across distributed systems to better detect potential attacks and to meet regulatory requirements can potentially exacerbate the problem if the monitoring is centralized. A single zero-day vulnerability would permit an attacker to access all information.**

This paper introduces a novel algorithm for performing policy-based security monitoring. We use policies to distribute information across several hosts, so that any host compromise has limited impact on the confidentiality of the data about the overall system. Experiments show that our solution spreads information uniformly across distributed monitoring hosts and forces attackers to perform multiple actions to acquire important data.

***Index Terms*—security; monitoring; policy compliance; confidentiality; distributed systems;**

I. Introduction

Directed attacks toward organizations are becoming commonplace. While these attacks often use targeted fishing emails for getting a foothold into the organization, once inside they require information about network topology, firewalls, and placement of critical systems to further perpetrate the attack [1]. Attackers need to identify critical resources, and malware needs to target systems with specific characteristics in order to exploit vulnerabilities. Searching for such information through the network using port scanning or using random infections of non-critical systems increases the chance of detecting the malware process [2]. The knowledge of network topology information, configurations, and critical system placement enables attackers to specifically target sensitive data and lowers the chance of detection.

Monitoring systems are a perfect target for acquiring such information. The increased need for situational awareness and policy compliance require integrating and correlating events coming from multiple sources. For example, software such as Splunk [3], Bro [4], and SEC [5] integrate events generated by logs, by network packet analysis, and by SNMP. However, while this monitoring helps detect attacks, the integration of information creates a large target for attacks that, if exploited, provides an attacker access to a large amount of information about the system's state.

This paper presents an algorithm aimed at protecting the confidentiality of the information produced by the monitoring system. We take advantage of the fact that the scale of modern infrastructure systems already requires the use of several hosts for distributing the load of monitoring (e.g., [6]). Instead of concentrating information in a single system, we distribute knowledge about the infrastructure into multiple monitoring servers that collaborate for detecting violations of security policies. The centralization of information used in other monitoring systems relies on the assumption that securing a single system is simpler than securing multiple systems. However, recent compromises of critical systems such as certification authorities [7], targeted attacks [1], and the presence of zero-days vulnerabilities challenge such an assumption. For example, the exploitation of a single zero-day vulnerability in the monitoring server would allow an attacker to acquire all information about the infrastructure. In our system, the exploitation of a single monitoring server would reveal only limited information. Additionally, our system is able to distribute the load across a large number of servers, thus enabling policy compliance to scale up to large-scale infrastructure systems.

We focus our analysis on policy-based monitoring systems. Many current monitoring systems use policies for analyzing and for correlating the events collected from the infrastructure. Such an approach is already used in several applications (e.g., [5], [4], [8]), and advanced applications of policy-based monitoring have been proposed for validating the compliance to regulatory policies such as PCI [9] or FISMA [10]. These policies are generally called "event correlation policies" as they express conditions over the logic co-occurrence of events. For example, the co-occurrence of an IDS event indicating the detection of an exploit packet and of a vulnerability-scanner event indicating the presence of a software vulnerable to such an exploit signifies the possible compromise of a device. The events used by these systems are generated by a large number of devices using a variety of sources such as SNMP data, intrusion detection systems (IDS), and log-analysis tools. In our architecture, these independent sources of information send events to a large number of monitoring servers distributed across the organizations.

We show that by expressing policies using Datalog rules [11] we can perform a decentralized event correlation that does not require concentrating events in any single system for processing. Using Datalog, we describe systems as a set of resources (e.g., computer systems, software systems, network connections) and their relations (e.g., a computer system is running a software program). We analyze the rules to identify

the events that need to be correlated for the identification of policy violations. We use a resource-based distribution of information across multiple servers, and we rewrite each Datalog policy in a set of "resource-centric" rules. Each resource-centric rule correlates information about a single resource in the system. The results of this process are forwarded to other servers to be correlated with events related to other resources, but only if they can potentially contribute to the detection of a policy violation.

The contribution of this paper is summarized as follows:

1) We introduce a resource-based approach to distribute policy compliance monitoring across multiple hosts.
2) We present an algorithm for rewriting Datalog rules and performing a distributed resource-based validation of complex policies.
3) We analyze monitoring events from real datasets and we evaluate experimentally the efficiency of our technique. Results show that our approach provides a 3-fold reduction of the number of events obtainable by an attacker when compared to other distributed approaches.

The rest of the paper is organized as follows. Section II analyzes related work in the area. Section III describes our model of event co-occurrence for policy-compliance systems. Section IV describes the adversary models we consider in this work. Section V introduces our algorithm. Section VI describes our experimental evaluation. Section VII describes the limitation of our approach and our future work. Finally, Section VIII concludes our work.

II. Related Work

Monitoring is a widespread service in modern systems. Most monitoring systems provide some limited protection of log data confidentiality. A basic protection of confidentiality is provided by protecting data-in-transit. For example, *syslog-ng* [13] uses TLS to transmit encrypted log data from the devices and the monitoring servers. However, the encryption of data-in-transit only leaves data vulnerable: if the monitoring server is compromised, the attacker has access to all past and future data.

More advanced solutions provide protection of the data-at-rest by creating encrypted and tamper-proof audit logs that can be accessed only by authorized users. One of the earliest mechanisms has been introduced by Schneier et al. [14]. Their approach uses one-way hash chains to protect the log files from modifications. Additionally, logs are encrypted to protect them from unauthorized access. Other solutions (e.g., Ma et al. [15]) extend such an approach to provide additional integrity protection. While these approaches are useful for protecting the integrity of the event logs and can be used in conjunction with our algorithm to provide trusted audit logs, their confidentiality protection is not suited for our scenario: as event correlation requires performing processing on the data, events need to be accessible to the monitoring server. Attackers compromising the server would have access to such data.

The Intrusion Detection System Bro [4] provides a distributed mechanism for performing event correlation. Communication between Bro nodes is performed on top of SSL to protect data-in-transit. Correlation between events is performed by programming policies using a pub/sub mechanism. For example, a distributed IDS cluster built on top of Bro has been presented by Vallentin et al. [6]. They use a flow-based hashing for distributing the packet-processing load across the instances. Inter-flow correlation is performed using the pub/sub mechanism. While the pub/sub mechanism provides flexibility in specifying policies and in defining their evaluation, it provides no guarantees that information about the system is distributed across nodes. It is up to the programmer to evaluate policies without creating such an aggregation of information. The algorithm we present in this work provides a mechanism for selecting automatically the events that each monitoring server should receive and send for validating policies. Our algorithm ensures that potential policy violations are detected and that information about the system is distributed across a large number of hosts.

More generally, the problem of protecting the privacy of log data has also been addressed in the context of sharing network traces across organizations. Several authors (e.g., [16] [17] [18] [19]) point out the security problems in providing access to this type of information to external entities and propose methods for anonymizing the data. However, such anonymization methods are not applicable to our case for several reasons. First, these techniques rely on aggregating data in a centralized server within each organization for processing. Second, even if anonymization could be performed directly on devices, these techniques are specific to network traces and they are not easily generalizable to the problem of policy compliance.

Other work focuses on protecting the monitoring system itself from compromises. For example, recently several secure monitoring solutions have been using Virtual Machine Introspection (VMI) for protecting the monitoring software from compromises. They run the monitoring software in a separated VM co-located with the host to monitor (e.g., Livewire [20]) and they access information by analyzing memory and disk data without the OS mediation. The security of these systems relies on the fact that compromising the monitoring VM is harder than compromising other VMs: the monitoring VM runs a small amount of software and, hence, exposes a small attack surface. However, while this assumption holds if the monitoring VM is used only for acquiring events from a particular system, it does not hold in the processing servers that correlate events across entire organizations. Such hosts need to be accessible through the network to allow devices to send events to them, and they need to run a substantial amount of software for validating policies and for providing network administrators access to data. Our algorithm reduces the consequences of compromises of such servers.

Previous work provides mechanisms for reducing the number of events sent to the event correlation nodes by performing part of the compliance monitoring on each host [21]. Our work focuses on protecting the remaining part of the events that cannot be processed locally. Other work introduces an

architecture for compliance that distributes information across several hosts [22]. However, their approach is limited to RDF policies and its ability of distributing events in a real scenario is unclear. We use more general Datalog policies and we provide a more detailed evaluation of the advantages provided by distributing data.

Our work is based on the principles of intrusion tolerance [12]. We exploit the distributed nature of the problem of event-correlation to provide a specific intrusion-tolerant solution.

III. Policy Compliance and Event Processing

The management of the security of large infrastructure systems often relies on defining "policies." Policies are rules that specify high-level security requirements and are used for selecting the proper states and configurations of systems. In the past few years there has been an increasing interest for the development of policies that apply to entire classes of industries. Regulatory entities and industries introduced classes of policies such as PCI-DSS [9] for credit card industries, NERC CIP [23] for power grid systems. Several of the policies specified in NERC CIP or PCI-DSS relate to network, OS, and application configurations. Their goal is to specify a minimal level of security to which all systems need to comply. Network administrators can monitor the compliance of their systems to these policies by specifying *monitoring rules*. These rules validate the configuration of security devices, of server applications, and of end-host computer systems. For example, a completely automated monitoring for compliance to one of NERC CIP policy requires defining rules that detect when machines are used by the critical devices that control the power grid, and that ensure that such machines are placed within a protected electronic perimeter (e.g., firewalls).

A. Monitoring Rules

Monitoring systems generate events when there is an "interesting" change in the state of the system. Examples of events are the running of a new program, the log-in of a user, or the detection of a potential attack by an IDS. Monitoring rules check for the co-occurrence of events to identify when the system is operating in an undesirable state. Monitoring rules could define security requirements in a way which is orthogonal to the methods used for enforcing security and they can be used for providing an audit trace. For example, the NERC CIP policy requirement above might be implemented using firewall systems. A monitoring rule might monitor for events that identify critical systems and for events that indicate the presence of connections from outside the electronic perimeter. In the rest of the paper we use the terms monitoring rule and policy interchangeably to indicate an event-correlation rule.

Events in the policy compliance and network management scenarios generally report information about some "resource" in the system. A resource can be a computer system, an IP, a network, a software program, or a user. For example, an event indicating that a computer system is now connected to a new network represents a new relation between a resource representing a computer system and a resource representing the network. In this framework, monitoring rules express conditions over resources in the system and their relations. For example, a security policy might pose a condition that a resource of type critical system cannot be connected to a resource of type public Internet.

Datalog provides a logic-based language for representing resources and their relations. We use Datalog with the addition of time operators [24] for specifying events and rules. In Datalog, the type of an event is described by its *predicate*, while the resources that the event describes are the parameters of the predicate. For example, a computer $host$ connected to a network net with an IP ip is represented as `connected(`$host$`,` net`,` ip`)`. We assume that all resources (e.g., $host$, net, ip) are identified by unique names. Logic and Datalog have been used by other work to express policies and to evaluate the security of systems (e.g., [25]).

Events are associated with timestamps in a way similar to the concept of *situation* of Amit [26]. Simple events, such as the detection of a malicious packet from an IDS, are instantaneous and are associated with a single timestamp. Complex events represent states in the system and they are represented by two timestamps: a start timestamp and an end timestamp (that might be unknown if the system is still in the state). Rules in this context can express window-based event correlation and a state-based correlation (i.e., event correlation that is based on reconstructing the current state of the system). These two models can express infrastructure policies currently defined in regulatory documents. The process of validating the *compliance* of a network system to the policies is performed by integrating events in a knowledge base (KB), and by checking if any of the monitoring rules trigger the presence of a violation.

For example, we can consider a new event `outperimeter` generated when a network is not protected by a firewall. We express a policy stating that a computer should not be connected to a network not protected by a firewall using the following formula:

$$\begin{aligned}&\texttt{connected}(H, N, IP), \texttt{outperimeter}(N)\\ &\rightarrow \texttt{violation}(H, N).\end{aligned} \tag{1}$$

The parameters of the events in capital letters are variables. The policy requires checking for the co-occurrence of the events `connected` and `outperimeter`, with the proper parameters. We support the different types of time relations that can be defined over time windows. Timestamps are added as implicit parameters of the events and time specifications are translated into conditions over events' start and end times.

IV. Adversary Model

We define an attacker, Eve, interested in acquiring more information about the state and the structure of the infrastructure system. We assume that the attacker has a limited initial knowledge and she is interested in compromising the monitoring system to acquire more knowledge. While compromising the monitoring system is not the only method for acquiring

more information about the system's structure (e.g., she could compromise a set of servers in the network and monitor traffic and communications), we assume that attacking it maximizes the efficiency of the attack. We assume that the attacker has knowledge about the monitoring system.

First, we make a few reasonable assumptions about the capability of the attacker for compromising hosts. We assume that attackers can compromise monitoring servers, but these compromises are rare events. We assume that some effort is required for compromising an additional monitoring server (i.e., an attacker cannot compromise all servers at the same time with no effort). Such an effort can represent the difficulty of acquiring an additional set of credentials, or the difficulty of compromising another machine located in the network of the monitoring server so that firewall protections can be circumvented.

Then, we define more clearly the goals of the attackers. We assume that the attacker is interested in both the presence of policy violations and in the raw events collected by the monitoring system. Policy violations indicate directly possible venues for attacks, but they are generally fixed quickly by network administrators. This provides a limited window of opportunity to attackers. Other data can be used to identify critical systems or plan the next step of the attack.

The type of raw events interesting to an attacker can change depending on the type of monitoring performed by organizations. We consider multiple types of attackers interested in acquiring different types of information from the monitoring system. We classify attackers as follows:

1) MAX_ALL: An attacker wants to maximize her overall knowledge about the system.
2) MAX_RESOURCE_CRITICAL: An attacker wants to maximize her knowledge about a limited set of critical resources.
3) MAX_TYPE_CRITICAL: An attacker wants to obtain information about specific critical types of events (e.g., the presence of a vulnerability on machines)

We evaluate the protection provided by our system to these different types of attackers, and we show that our resource-based distribution of events provides a better protection than other solutions.

V. Distributed Event-based Compliance

An architecture for event correlation is generally composed of two types of devices: *event sources* and *monitoring hosts*. Event sources are the end-devices subject to the monitoring. Information about these devices is provided directly to the monitoring system in form of events, or is translated into events from information collected by SNMP queries, IDS alerts, Syslog, application-log parsing, or network scanning. When new information about the state of a device is detected, event sources generate events and deliver them to the monitoring hosts. The monitoring hosts receive these events and correlate them to identify policy violations. For the type of policies we consider, a simple way to perform the correlation process is to integrate information into a single knowledge base and apply reasoning. If the conditions of one of the rules are satisfied, a statement indicating the presence of a violation is generated in the knowledge base.

Distributing the knowledge about the system to multiple monitoring servers improves the security of the system toward our attack model for several reasons. First, the fact that a single compromise is not sufficient anymore for acquiring the information searched by attackers forces them to perform multiple actions. The increased activity gives multiple opportunities to IDS systems to detect such malicious behavior. To get a qualitative idea of this effect, we use a simple probability model. If we consider the probability of detecting an attack at each action independent p_a, we have that the probability of detection p_{dt} grows with the number of servers k as $p_{dt} = 1 - (1 - p_a)^k$. With no distribution, this number is constant and equal to p_a.

Second, a centralized monitoring system provides a simple target for attack. While rare, zero-day vulnerabilities or stolen credentials can be used by an attacker for compromising the system and, hence, accessing the entire monitoring information. In our distributed approach, we rely on monitoring servers managed by different organization units. The servers are placed in different networks and they are managed using different credentials. In such a configuration, accessing the entire state of the system requires compromising multiple credentials or exploiting multiple zero-days vulnerabilities to communicate with the different servers. Even when a single monitoring server is compromised, such an exploit has limited effects on the amount of information obtained by the attacker.

We can qualitatively estimate the effects of this advantage. For ease of calculation, we consider a simplified attack model where we assume that an attacker has a probability p of successfully compromising a host. Such a probability is related to a simplified notion of the "effort" required for comprising an additional server: p represents the probability of success given a constant effort. In the centralized case, the probability p_c of compromising the central server is equal to p and represents the probability that the entire knowledge about the system is compromised. In our case, an attacker that wants to have access to the information needs to compromise multiples servers. We call p_d the probability of compromising at least k servers over the n monitoring servers, and we define it as follows:

$$p_d = \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{(n-i)} \tag{2}$$

Distributing the information across n servers is better than centralizing it if $p_d < p_c$, i.e., if the probability p_d of compromising at least k servers containing the information searched by the attacker is lower than the probability p_c of compromising the centralized server. To maximize the difference between the p_d and p_c we need to have an high value of k: we need to force attackers to compromise multiple machines to find the information they need. Such a number k does not need to be very large: if we have 20 monitoring

servers and we consider a probability $p = 0.01$ for an attacker to compromise one of the monitoring servers, a value of $k = 3$ leads to a probability $p_d = 0.001$. This value is robust to changes in probability p. For example, we can consider the common belief that "securing one server is easier than securing multiple machines." To analyze its effects, we can assume that compromising the centralized monitoring server is twice as hard as compromising one of the monitoring servers. Even in this case, if we take a probability $p_c = 0.01$ and a probability of compromising one of the monitoring servers as $p = 0.02$, the probability p_d with $k = 3$ is 0.007. With $k = 4$, this probability goes down to 0.0006.

Hence, while distributing the information increases the attack surface and might make finding one single vulnerable monitoring server more likely, attackers need to compromise multiple machines to acquire the information relevant for their attacks. The multiple steps of the attack make it easier to detect and less likely to succeed completely. Based on this observation, we build our policy-based monitoring system so that information is distributed uniformly across several monitoring nodes. Our experiments show that we can maintain the value k large for the attack models we consider.

The process of matching events within each monitoring host is performed by a service called "broker." A broker is a software service that can run in a dedicated machine or can be co-located with other services. Each broker receives events related to a limited subset of the resources of the organization. They use our algorithm for exchanging information and, hence, for finding all violations presents in the system. In this way, event load and information about the system are distributed across a large number of hosts. Resources are assigned to brokers at random, so that no single host concentrates the knowledge about the resources of a critical organization unit. Each monitoring server can subscribe to violations of specific rules or to violations involving specific resources. These subscriptions are distributed across brokers. Once a broker finds a violation, it delivers a notification to the subscribed monitoring servers. We limit the amount of subscriptions that each server can submit. While communication between brokers is necessary, such communication is limited only to the event-correlation service. Other services can be isolated through firewalls to reduce the attack surface. An architecture supporting our algorithm is shown in Fig. 1.

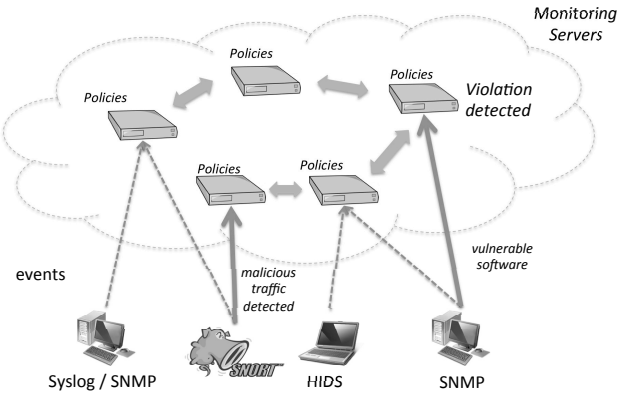

Fig. 1. Architecture of a policy-based event correlation system supporting our algorithm. Devices send events to different brokers depending on the resources involved in the event. The monitoring servers interact and identify violations by correlating events in a distributed manner. Each server maintains limited information about the overall state.

A. Distributed Event Correlation Algorithm

Our distributed algorithm defines 1) how events are distributed across brokers and 2) what information needs to be shared across brokers to verify compliance. As policy compliance policies define conditions over resources and their relations, we base our algorithm on *resources*.

We use the intuition that events taking part in the violation of a monitoring rule are related to each other as they provide information about a limited set of resources that, together, create the violation of the policy. For example, we can consider a policy that specifies that a user in a computer lab should not be logged on two machines at the same time. A monitoring system could generate an event when a user logs into a machine and when the user logs out. A violation would be represented by two logins of the same user on different computers. We can validate compliance by integrating events about the same user on the same broker. Different users can be managed by different brokers without limiting the ability of the system to detect violations. We generalize this intuition to complex policies by identifying events that share resources.

We partition the process of validating policies by resources. Our algorithm validates compliance by aggregating events in a series of steps. Each step correlates events related to a single resource. If the single-step correlation identifies that the resource can potentially contribute to a violation, we send a summary to the next step so that it can be correlated with additional resources.

We distribute resources uniformly across brokers using a hash function. If the entire policy is defined on a single resource r, then the policy is validated completely in the broker managing r. Events about r are directly delivered to the broker and inserted into a local knowledge base. For example, we can take the resource $host_a$ (associated to a broker x) and the policy $\texttt{critical}(A), \texttt{poweroff}(A) \rightarrow \texttt{violation}$. The two events $\texttt{critical}(host_a)$ and $\texttt{poweroff}(host_a)$ are delivered to the broker x and inserted in its local knowledge base. The knowledge base matches both events and finds a violation. It is easy to prove that if two single-resource events are correlated by a policy, both events are always sent to the same broker.

When events are related to multiple resources, the problem becomes more complex. For example, a policy that relates IDS events with the presence of vulnerable programs running on a computer system requires integrating events about several resources such as computer systems, software, and specific network flows. In general, events involved in a violation cannot be related to a single resource. For example, an event could state that a computer system $host_a$ is running a program $\texttt{runs}(host_a, p)$ that is untrusted $\texttt{untrusted}(p)$. Another event could specify that a system $host_b$ is criti-

cal for the system $\texttt{critical}(host_b)$. A third event could state that there is a connection between $host_a$ and $host_b$, $\texttt{connection}(host_a, host_b)$. Even if these events are related to each other, no single resource is shared across all of them.

B. Rule Rewriting

Our algorithm performs the distributed resource-based correlation by rewriting monitoring rules. We analyze a policy and create an equivalent set of rules that we call *resource ruleset*. Each rule requires information about a single resource and represents a partial violation of the policy. If a partial violation of the policy is found, a new event is generated. We forward the event to the broker managing one of the resources connected to the potential violation. For example, a policy could specify that there is a violation if a machine connected to an internal network receives a connection directed to a vulnerable program. We represent this policy as $\texttt{connected}(H, N)$, $\texttt{internal}(N)$, $\texttt{runs}(H, S)$, $\texttt{vulnerable}(S)$, $\texttt{conn}(H, S, IP) \rightarrow \texttt{violation}(H, S, IP)$. We can rewrite the policy as follows:

$$\begin{array}{l} \texttt{runs}(H,S),\texttt{vulnerable}(S) \rightarrow \texttt{partial}_S(H,S) \\ \texttt{internal}(N),\texttt{connected}(H,N) \rightarrow \texttt{partial}_N(H) \\ \texttt{partial}_S(H,S),\texttt{partial}_N(H),\texttt{conn}(H,S,IP) \rightarrow \\ \qquad\qquad \texttt{violation}(H,S,IP) \end{array} \quad (3)$$

The first rule relates events about the resource identified by the value of the variable S. The second rule relates events about the resource identified by the value of the variable N. The third rule relates the partial information from the previous rules with events about the resource H. The conclusions of the first two rules are events of type `partial` representing a partial processing of the original rule. This new set of rules generates the same statements $\texttt{violation}(H, S, IP)$ as the original statements, but it is formulated as a sequence of rules that filter events in different steps.

As the body of each new rule relates events about a specific resource, we can find all of its conclusions by aggregating in the same broker all events about such a resource. For example, if the two events $\texttt{internal}(net_1)$ and $\texttt{connected}(H, net_1)$ (for all H) are sent to the broker associated with the resource net_1, such a broker is able to compute all the couples of resources (net_1, H) that match the body of the rule. We summarize the partial evaluation with the statements $\texttt{partial}_N(H)$. Changing the value net_1 (i.e., value of the variable N) changes the broker in charge of the correlation. For example, if we consider $N = net_2$, the two events $\texttt{internal}(net_2)$ and $\texttt{connected}(H, net_2)$ are sent to a different broker associated with the resource net_2. The partial conclusions computed by each broker are forwarded to the brokers managing the resource identified by the value of the variable H of $\texttt{partial}_N$. On the brokers associated with the different values of the variable H, we perform the same process and we integrate the partial events $\texttt{partial}_N$, $\texttt{partials}_S$, and the event `conn`. Using our algorithm, brokers share data only if such an interaction is necessary for detecting a possible violation.

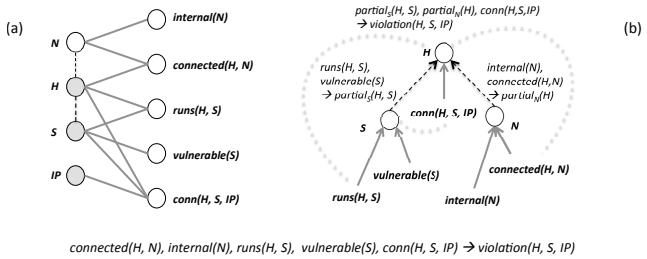

Fig. 2. (a) Policy graph. Dashed lines are added during the conversion process; (b) Policy tree generated by choosing H as root. Dotted lines are graph edges removed during the conversion process.

1) Basic Algorithm: The process of rule rewriting starts with the identification of the resources used in the rule. We represent this information in a *policy graph*. The policy graph is a bipartite graph where one set of nodes represents variables and constants used in the rule, and the other represents predicates. We create an edge between a predicate node and a variable node if the predicate contains the variable. A policy graph for an example policy is shown in Fig. 2(a).

The policy graph shows the relation between variables and events. If the graph is not connected we modify the policy by adding a common fictitious resource that connects the two parts of the policy. However, we believe that unconnected polices are rare as they would be the equivalent of a SQL join of two tables without a "where" condition.

Using the graph, a policy is rewritten as a resource ruleset in two steps. First, we choose an order for the correlation steps by computing a spanning tree of the graph that we call *policy tree*. Then, we generate the resource ruleset from the tree. As the root of the spanning tree represents the final step of the correlation process, we choose the root to be one of the resources that appear in the final violation notification message, so that information about such a resource is readily available for creating the notification. Fig. 2(b) provides an example of the conversion of a policy graph to a policy tree. The steps necessary for such a conversion are as follows.

1) We remote variable nodes connected to only one predicate node. In Fig. 2(a), we remove the node IP.
2) We explicitly represent the relations between resources by adding edges between variable nodes. We create a new edge between variable nodes of distance 2 (i.e., we ignore predicate nodes and we connect directly neighboring variable nodes). In our example of Fig. 2(a), we mark such edges with dashed lines.
3) We compute the spanning tree by selecting a root and by removing redundant edges. For each predicate node, we maintain only one edge. We select the edge to ensure that each variable node is connected to at least one event containing a reference to the variable node's parent. In Fig. 2(b), we mark the removed edges with dotted lines.

This transformation connects each predicate node (i.e., events) to a variable representing one of the resources contained in the event. Events are forwarded to the broker managing the resource represented by the value of the variable.

For example, we can take a predicate $\texttt{connected}(H, N)$ with an edge to N. An event $\texttt{connected}(host_1, net_1)$ is forwarded to the broker managing net_1. For the rest of the section we indicate with $\mathcal{N}$ the resource associated with the value of the variable N.

A broker managing a resource $\mathcal{N}$ can perform a partial matching of the policy. For each variable node, we create a rule of the resource ruleset by considering the incoming edges. At the lower heights of the tree, all incoming edges of a variable node are connected to predicate nodes. These predicate nodes all share the same variable H. For example, the body of the rule associated with the variable node N in Fig. 2(b) is $\texttt{internal}(N)$, $\texttt{connected}(H, N)$. For a violation of the policy to exist, the variable N must have the same value on all predicates contained in the policy. As the broker managing $\mathcal{N}$ receives all events that share the same value for the variable, the partial matching is complete. The head of the partial rule is a predicate that has the variables used in the partial rule as parameters. In this case, the result is encoded with $\texttt{partial}_N(H, N)$. This predicate is treated as a new event, and it is forwarded up in the tree. In our example, the event is sent to the broker $\mathcal{H}$. Step 3 in the conversion of the graph to the tree ensures that the broker always has at least one event that carries both the information about the current resource and about the resource one level higher in the tree.

A variable node can have incoming edges connected to other variable nodes. These edges are considered as connected to events representing the partial execution. In our case, the node H is associated to the rule $\texttt{partial}_S(H, S)$, $\texttt{conn}(H, S, IP)$, $\texttt{partial}_N(H, N)$. If the variable node is the root, the rule is directly associated with the presence of a violation. When multiple policies are present, each policy uses a different name for the partial execution predicates.

During the execution of this algorithm there is no single broker managing the resolution for an entire policy: depending on the resources mentioned in the events, different brokers are in charge of the different steps of the aggregation and of the final identification of violations.

2) Remove unnecessary information: We remove unnecessary data from partial execution statements to reduce the amount of information sent to other brokers. If a variable is not used anywhere else in the rule, we can drop it from a partial statement without affecting the equivalence to the original policy. For each variable node V, we build a set of variables we call *shared set*. The shared set is constructed by removing the subtree of V from the policy tree and by taking all variables used in the remaining tree. We drop from the partial execution statements of V the variables that do not appear in the shared set.

3) Distribute information about critical resources: A pure resource-based distribution of events offers little protection against attackers interested in acquiring information about a specific critical resource in the system. An attacker can acquire these events by identifying the broker managing such a resource and by compromising it. Our algorithm provides a protection against this type of attack by spreading events about critical resources across multiple brokers. As different types of events about the same resource are generally used in different policies, we add a policy-dependent prefix in the selection of the broker. When a resource is identified as critical, instead of relying only on the resource name r for the identification of the broker $H(r)$, we add a prefix p_i that depends on the policy in which the resource is used. Different types of events about a critical resource are sent to different brokers $H(p_i|r)$.

We consider critical all resources representing data about the monitoring servers. In this way, We limit the possibility that an attacker could use information collected from the compromise of a monitoring server to compromise another server.

C. Correctness argument

Our algorithm is correct and complete if it identifies the same set of violations that would be identified if all events were integrated in a global KB. We show the equivalence in two steps. First, we show that the processing of each rule in the resource ruleset on a broker is equivalent to the processing of the same resource-ruleset rule in a global knowledge base. Second, we show that the combination of the resource ruleset is equivalent to the original rule.

All predicates in a resource-ruleset rule share a common variable V. For any instantiation $V = v$, the rule is triggered only if events have the same value v for the variable V. Our algorithm ensures that if a predicate p is part of the resource-ruleset rule for a variable V, all events having $V = v$ are sent to the same broker. Hence, such a broker can identify all inference for which $V = v$. As the same rule is repeated in all brokers, we obtain the same result for every value of V.

Second, we show the equivalence between the resource ruleset and the original rule by noticing that the rewrite algorithm is, in fact, a simple logical manipulation of the rule. The rule-generation algorithm can be seen as a set of rewriting steps that preserve the equivalence. We choose a variable V_1 and create a rule $rule_{V_1}$ that has a body containing all the predicates $\texttt{pred}_i(\overline{A_i})$ (where $\overline{A_i}$ is a set of variables) such that $V_1 \in \overline{A_i}$. The head of the rule is a new predicate $\texttt{partial}_{v_1}(\overline{A_{V_1}})$ where $\overline{A_{V_1}} = \{\cup \overline{A_i} | V_1 \in \overline{A_i}\}$. We remove the selected predicates from the body of the original rule and we substitute them with the $\texttt{partial}_v(\overline{A_v})$. For each $rule_v$, because the predicate $partial_v$ is unique for the rule, we have that a particular assignment of $\texttt{partial}_{V_1}(\overline{A_{V_1}})$ is possible if and only if all predicates $\texttt{pred}_i(\overline{A_i})$ in the rule body are also true. Hence, at the end of the rewrite process, the initial rule has been rewritten as $\texttt{partial}_{V_1}(\overline{A_{V_1}})$, $\ldots$, $\texttt{partial}_{V_k}(\overline{A_k}) \rightarrow \texttt{violation}(\overline{A})$. Because each of the $\texttt{partial}_v$ is true if one only if the body of its rule is true, and $\overline{A_{V_1}}, \ldots, \overline{A_{V_k}}$ still represent the entire set of variables, we have that $\texttt{violation}(\overline{A})$ is generated if and only if it is generated by the original rule.

VI. Experimental Evaluation

We measure the ability of our event distribution to protect the system's information against the three types of attackers identified in Section IV. We show that our resource-based

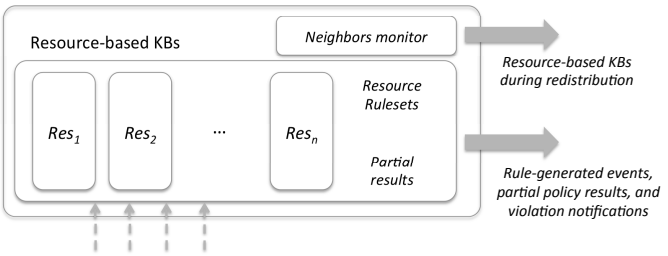

Fig. 3. Internal structure of our broker implementation. It receives events and adds them to the appropriate resource-based KB. The Datalog resource rulesets are added to the KB obtained by the union of the resource-based KBs. The events generated by the rules are forwarded to the appropriate brokers. In case of a changes in the availability of other brokers, the neighbors monitor sends and receives resource-based KBs from other brokers.

mapping provides a better protection of event-confidentiality than other types of event distributions, and that our technique only introduces a delay limited within a few seconds in the detection of violations.

A. Implementation

We implement the policy compliance algorithm in our distributed monitoring system. Our agent-based system provides monitoring of SNMP data, running processes, network connections, and logged users. Information is integrated in a set of monitoring servers communicating using a 1-hop DHT. The hash of the resource name is used for mapping resources to brokers. The system is implemented in Java. Rules are distributed to the servers before starting the process.

Each broker analyses the policies and creates resource rulesets and queries. Events received by brokers are placed in a resource-KB depending on the resources contained in the event. Rulesets and queries are applied on the KB obtained by taking the union of all resource-KBs. Once inference is performed, the monitoring server applies the queries defined from the rules and creates the set of events to send to other brokers. Such events contain the partial executions of the policies or specify the detection of a violation. If a violation is detected, the broker forwards it to the brokers subscribed to receiving violation notifications about the rule or about the resources involved (e.g., the monitoring server in the sub-organization managing the resources).

Mapping between resources and brokers can change over time (e.g., because we add a new broker). A component, the neighbors' monitor, detects when a resource becomes mapped to another broker and moves the proper part of the knowledge base to such a broker.

Failures of a broker are handled by remapping automatically resources to new brokers. Such a process is managed by the neighbors monitor. However, when a broker fails, its current knowledge base is lost. For the cases in which events are correlated within a limited time window, a failure would only affect temporarily the ability of detection violations until the time window is passed. After such a time, the new broker would have received all relevant events and the event correlation becomes complete again. For the cases in which correlation requires to store longer term information, devices are configured to send periodically such information about the state to brokers. For example, SNMP information is generally long-lived. A device could periodically (e.g., every 10 minutes) send again its entire state to the brokers. A proper handling of timestamps (such as the one described in Walzer et al. [24]) ensures that policy violations that occurred during the downtime are still detected, even if with a delay. An architectural description of a broker is shown in Fig. 3.

The evaluation of our system requires having policies and events to correlate. To evaluate our system in a wide range of realistic conditions, we generate events from publicly available data traces and from monitoring a set of systems in our research infrastructure. The first dataset we use is a network trace collected during the three days of the 2010 Network Warfare Competition [27]. We use Snort[1] to analyze the trace and generate 60152 security-relevant events carrying information such as type of alert, source IP, destination IP, protocols, and ports. The second dataset is composed of syslog data collected by monitoring the wireless network infrastructure of Dartmouth college [28]. It is composed of 30 million syslog entries describing the state of the wireless access points. It reports events such as association of wireless devices to access points, interactions between access points, and errors. The third dataset contains SNMP data about configurations of hosts, network connections, running services, and running programs. We collected this dataset by monitoring using SNMP the state of different types of machines: servers, development desktops, and laptop computers. We choose the datasets to show the applicability of our technique to different data used in policy compliance: network traffic data, network management data, and security management data.

B. Event Dataset Analysis

We start our evaluation by analyzing the characteristics of the events generated in network management and intrusion detection. Such an analysis shows that a resource-based mapping can provide a more uniform distribution of data than a mapping based on event-type.

First, we analyze our datasets to characterize the distribution of events and of resources. We identify event types and resource types. We have 24 types of events in the IDS dataset, 42 in the wireless management dataset, and 14 in the SNMP dataset. Each dataset has several types of resources. We identify source ip, dest ip, source port, dest port, and ICMP type as different resources for the IDS dataset[2]. We identify access point IDs and MAC addresses for the wireless dataset, and we identify IP, programs, network ports, and services as resources for the SNMP motoring dataset.

We show that the distribution of the number of messages is more flat if we distribute messages by resource, while it is far from uniform when we distribute messages by type (i.e., each resource has a similar number of messages related to them,

[1] http://www.snort.org

[2] We consider source and destination IPs and ports as different resources to better understand the dataset characteristics.

IDS Resource	Cardinality	With type	Avg Msgs	stddev
Event type	24	24	4.17%	12.0%
Src port	180	357	0.556%	1.97%
Dst port	249	704	0.402%	1.31%
Src IP	75	742	1.33%	4.64%
Dst IP	171	1299	0.585%	1.93%
ICMP type	6	108	16.7%	29.9%
Wireless Resource	**Cardinality**	**With type**	**Avg Msgs**	**stddev**
Event type	42	42	6.67%	17.2%
Access point ID	544	8944	0.184%	0.451%
MAC address	9251	59473	0.010%	0.100%
SNMP Resource	**Cardinality**	**With type**	**Avg Msgs**	**stddev**
Event type	14	14	7.14%	24.8%
Host IP	4145	4147	0.024%	0.457%
Program	493	493	0.20%	1.50%
Network Port	20770	24222	0.00480%	0.20%
Services	56	56	1.79%	4.51%

TABLE I
RESOURCES AND THEIR DISTRIBUTIONS IN THE DATASETS. WE CONSIDER THE AVERAGE NUMBER OF MESSAGES FOR EACH RESOURCE AND ITS STDDEV. THE VALUES ARE NORMALIZED BY THE TOTAL NUMBER OF MESSAGES DEALING WITH THE SPECIFIED TYPE OF RESOURCE.

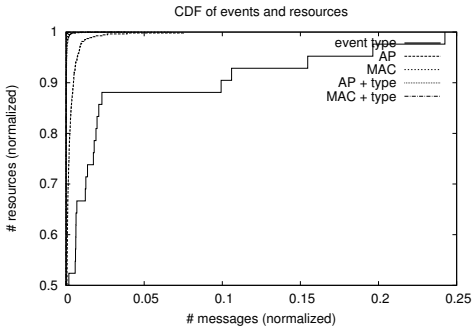

Fig. 4. Event distribution for the Dartmouth dataset. We show the fraction of resources (y-axis) receiving at least the fraction "x" of the messages.

while there are type of events that are much more frequent than other types). We compute the standard deviation of the normalized distribution of the number of messages by type and by resource. For each event type and resource, we compute its fraction of the total messages. We compute the stddev of this distribution and we represent the value as a percentage. This value is much larger when messages are aggregated by event types, while it remains low (in most cases) when events are aggregated by resource. We summarize this data in Table I.

A graphical representation of the CDF of this message distribution for the Dartmouth dataset is shown in Fig. 4. The y-axis of the graph represents the fraction of resources (or message type) that are mentioned in at least the fraction of messages specified in the x-axis. We see that we have a large number of messages for each event type, while most resources are mentioned in a small fraction of the messages. Hence, a distribution of messages by resource could provide a more uniform distribution of information.

C. Deployment on EC2

We run the system on EC2 spot instances to test the system in a real distributed environment. One of the instances generates events according to the distributions specified in our datasets to model a stream of events from a real system. We performed several experiments by changing the number of policies, the length of the policies, and the number of brokers. Each data point is the average of at least 5 executions. Each time new policies have been generated to consider different variations of rules and events.

We compare our system with systems presenting a different distribution of events. Centralized solutions do not provide any protection against the attack, as all information is compromised as soon as the main server is compromised. For this reason, we do not consider it in our evaluation. Instead, we consider a distribution of events based on event-type.

Distribution based on event type is used in pub-sub policy-based event systems for *filtering* events (e.g., [29]). The processing of a policy is distributed across brokers by processing the policy in groups of two event types. For example, a policy in the form $A \wedge B \wedge C \rightarrow violation$ is split into two parts: $A \wedge B \rightarrow partial_{AB}$ and $partial_{AB} \wedge C \rightarrow violation$. A broker b_1 receives events of type A and B and send to a broker b_2 the resulting events of of type $partial_{AB}$. Broker b_2 integrates the events $partial_{AB}$ with events of type C to identify all policy violations. While at first glance this splitting of the rule is similar to the one we describe in this paper, this process does not consider in the mapping the value that the variables assumes and groups events only based on their predicates (i.e., their type).

To the best of our knowledge, current pub-sub policy-based systems do not address the problem of limiting the knowledge maintained at each broker. Allocating rules and events to brokers in this solution is a challenging problem. A solution that minimizes the overall number of events would require mapping rules containing similar set of predicate types on the same brokers. However, by doing such an allocation, the maximum knowledge on brokers becomes large, as each different rule might require a slightly different set of predicates. To provide a fair comparison, we perform an explicit allocation of type-based policies to brokers so that the knowledge in each broker is reduced. We allocate in the same broker partial rules that manage the same events, and we balance the remaining rules across brokers to balance the maximum knowledge.

The evaluation of our solution needs to analyze the behavior of the system with a wide range of policies. While there are already a limited number of policies specified in regulatory documents, the way to map these abstract policies in rules that rely on information acquired from the system depends on the organization. An evaluation that focuses only on these policies would be limited in evaluating the system for the future types of complex policies. For this reason, we evaluate our system using a set of semantically meaningful but randomly generated polices. We use the semantic relations between the events in our datasets to construct policies that preserve these relations. We create such policies by defining a graph that relates events and resources: nodes in the graph are the resources; edges of the graph are the events generated in our datasets. Each

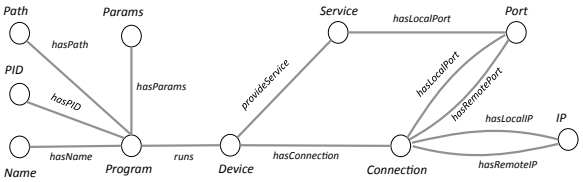

Fig. 5. Semantic relations between resources and events. We use this graph to generate random policies that preserve a valid meaning.

policy is constructed by a random walk through the graph. The semantic graphs we used in the evaluation of the SNMP data is shown in Fig. 5. We use the SNMP dataset as it provides the most diverse set of events for the construction of complex policies.

We quantify the amount of information obtained by an attacker using the number of events, as each event represents an atomic piece of information about the system. The distributed execution of the rules, both in the resource-based solution and in the type-based solution, creates partial results in form of events. These events might not carry complete information about the system's condition. For example, we can consider a resource ruleset $p_1(A,B), p_2(B,C) \rightarrow partial_B(C)$ and $partial_B(C), p_3(C,D) \rightarrow violation(C)$. The knowledge of the event $partial_B(r)$ can be used to infer that there exist two events in the form $p_1(u_a,u_b), p_2(u_b,r)$. However, the exact values for the resources u_a and u_b are unknown. Even if such information is partial, it might still be useful to an attacker. In our evaluation we consider this type of inference. As we do not know the usefulness of partial events, we consider the events that can be inferred by the knowledge of $partial_B$ as complete events. Hence, we might be overestimating the knowledge acquired by an attacker. This type of "backward-inference" is measured by inverting the direction of all the rules in the resource ruleset. In our example, we measure the number of events in a knowledge base containing the rules $partial_B(C) \rightarrow p_1(u_a,u_b), p_2(u_b,C)$ and $violation(C) \rightarrow p_3(C,u_d), partial_B(C)$. In this KB, the knowledge of the event $violation(C)$ allows the attacker to infer three other events. We only count these events if they are not already known by the attacker because of other information contained in the same KB.

We evaluate the ability of the system to distribute information across the brokers. For estimating the protection provided by the system against a MAX_ALL attacker we measure the number of events stored in each broker. We see that brokers, on average, store less than 10% of the events. At the most, we have one broker that stores about 17% of the events. The effects of increasing the number of rules are limited: the information about resources is reused across the execution of multiple rules. The effects of inference are also limited. By performing inference we can obtain a few additional predicates about resources in the system. We see that a type-based solution requires storing a larger number of events in brokers for all cases. Fig. 6 shows the normalized number of events stored in each broker when the number of rules increases.

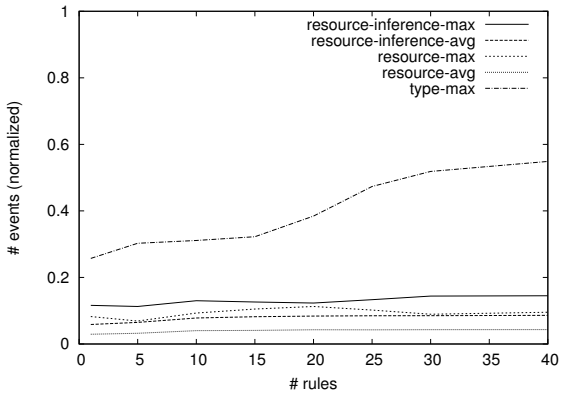

Fig. 6. Average and maximum number of events in each broker. 20 brokers.

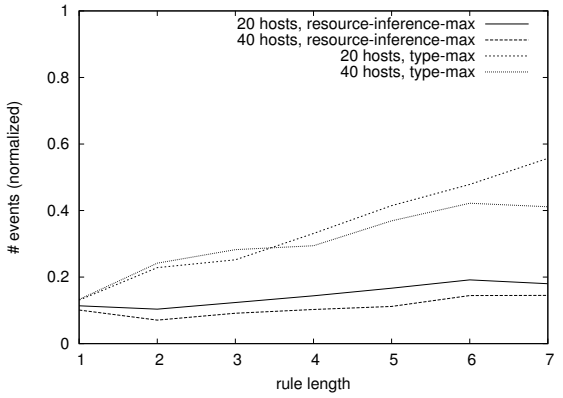

Fig. 7. Events in each broker as a function of the length of the rule. 20 rules.

We measure the normalized maximum number of events stored in a broker with the increase of the length of the rules. Having longer rules creates the need to match a larger number of predicate combinations. This increases the number of events to store both in the resource-based solution and in the type-based solution. However, the number of events in the resource-based solution remains significantly lower. For all cases, the maximum is obtained as the average maximum amount of events across multiple executions. These results are shown in Fig. 7.

Another measure of the protection provided toward a MAX_ALL attacker is the cumulative effects of the compromises of multiple brokers. We consider brokers in decreasing number of events to consider the worst case. We see that our resource-based approach distributes the load so that compromises of multiple brokers still have limited effects. These results are shown in Fig. 8.

Next, we measure the effects of our event distribution against an attacker interested in knowing all events of a specific type (i.e., a MAX_EVENT_TYPE attacker). We randomly select an event type and declare it "type-critical". We measure the maximum fraction of "type-critical events" that an attacker obtains when compromising a broker. As the event-type distribution uses event-type for its distribution, our

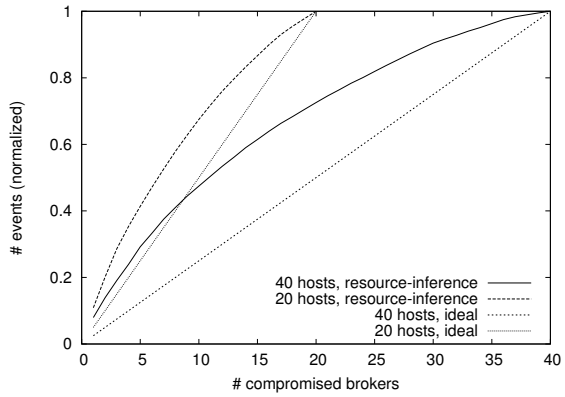

Fig. 8. Fraction of events obtained when multiple brokers are compromised. 20 rules. The brokers with most events are compromised first.

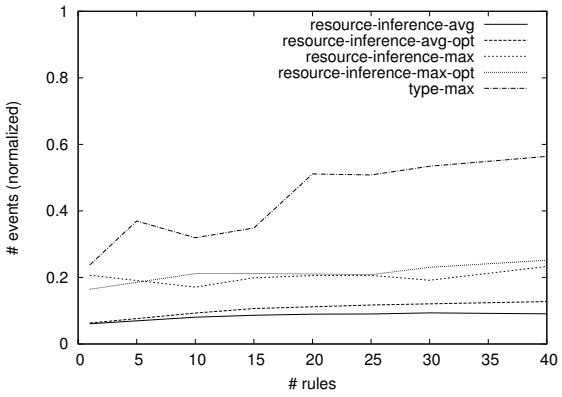

Fig. 10. Average and maximum fraction of *resource-critical* events contained in each broker. 20 brokers.

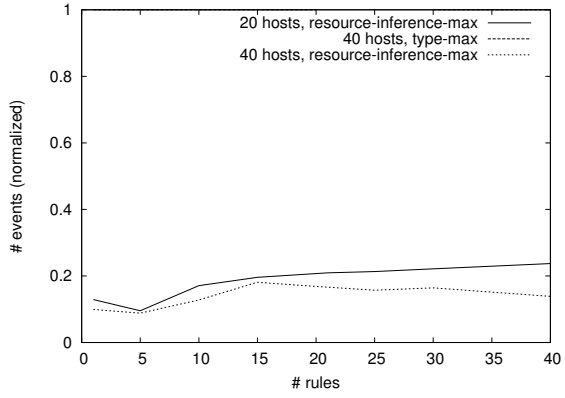

Fig. 9. Average and maximum fraction of *type-critical* events contained in each broker.

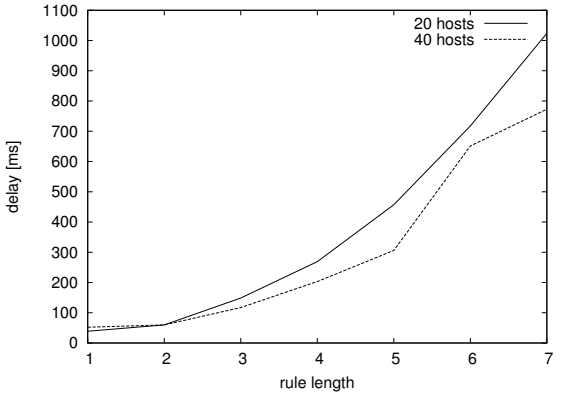

Fig. 11. Violation detection delay introduced by our algorithm. 20 rules.

technique provides a better protection against this type of attack. These results are shown in Fig. 9. We see that for a type-based distribution of events the totality of events of a specific type can be found on a specific node. Hence, the value is 1 for all cases. In our resource-based distribution, the number of type-critical events stored in each broker is limited.

We measure the effects of our approach against an attacker interested in obtaining information about a specific set of resources (i.e., a MAX_EVENT_CRITICAL attacker). We randomly marked 10% of the resources as critical. We define as "resource-critical" the events that relate to one (or more) of such critical resources. Our goal is to minimize the amount of events related to critical resources acquired by the attacker. We measure the max value of critical events stored in each broker for our resource-based distribution and for an event-type distribution. We show that the resource-based distribution limits the number of critical events stored in each broker. These results are shown in Fig. 10.

However, for attackers targeting a specific resource, a pure resource-based mapping algorithm provides weak guarantees: an attacker only need to compromise a specific broker to access all information about the resource. A better strategy for deployment can take advantage of a mix-mechanism for distributing data. This method distributes events about a critical resource to multiple brokers instead of concentrating them in a single broker. However, it also increases the overall distribution of information, as it reduces the reuse of events across multiple rules. We show the effects of this distribution in Fig. 10 with the suffix *-opt*.

The multi-step validation has limited negative effects on the performance of the system. The distributed correlation process adds a delay in the detection of problems. The result of the processing in one broker needs to be forwarded to other nodes before a complete detection is performed. We measure the average delay in the detection introduced by our system. We see that the delay introduced is within a second even for long rules. The slightly lower delay in the 40-host solution is created by a lower average communication delay in the 40-host network configuration of our EC2 deployment. We show this results in Fig. 11. The load of receiving event messages is distributed across servers in a way proportional to the amount of events stored in each broker.

In summary, our solution provides a better distribution of information and, hence, a better protection against attacks toward the confidentiality of the monitoring system than other previous solutions for the distribution of information.

VII. Discussion and Future Work

While our algorithm achieves a good distribution of information across hosts, we highlight a few limitations of our approach and possible future extensions.

In the evaluation we do not consider inference based on the semantic of the data. For example, the presence of an event might imply with high probability the presence of a particular condition in another resource. However, our approach of limiting the overall exposure of information would limit also the amount of data available for such an inference.

The algorithm focuses on the confidentiality of the event data and does not consider the integrity and the availability consequences of attacks. Protection against these attacks can be included by adding redundancy in the aggregation process and by the use of the log audit techniques presented in the related work section. The addition of redundancy would increase by a constant factor the number of events contained in each monitoring host. Reducing the data necessary for policy validation is still important to reduce the amount of data to replicate. Our future work will investigate the tradeoff between confidentiality, availability, and integrity in monitoring.

Additionally, future work will look at techniques such as privacy-preserving set intersection to reduce the amount of partially processed information stored in monitoring servers. These techniques allow matching events between monitoring hosts without revealing partially processed data.

VIII. Conclusions

We introduce an algorithm for performing policy-based large-scale event correlation that maintains the knowledge about the state of the system distributed across a large number of machines. We show that this distribution of information increases the security of the system toward attackers interested in exploiting the monitoring system to acquire information about the infrastructure's state. We consider attackers interested in all knowledge about the system, attackers interested in knowledge about a particular type of event, and attackers interested in a specific critical resources. We perform experiments by injecting events taken from real datasets in an implementation of the monitoring system running on machines in a EC2 cluster. We show that our system provides a better protection against all these types of attackers.

Acknowledgment

This work was partially supported by the Boeing Trusted Software Center at the Information Trust Institute, University of Illinois at Urbana-Champaign. 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.

References

[1] C. Tankard and D. Pathways, "Persistent threats and how to monitor and deter them," in *Network Security*, Science Direct, vol. 2011, no. 8, pp. 16–19, 2011.

[2] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Network & Distributed System Security Symposium (NDSS)*, 2008.

[3] "Splunk," 2011. http://www.splunk.com/

[4] "Bro Intrusion Detection System," 2011. http://bro-ids.org/

[5] "Simple Event Correlator," 2011. http://simple-evcorr.sourceforge.net/

[6] M. Vallentin, R. Sommer, J. Lee, and C. Leres, "The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware," in *Recent Advances in Intrusion Detection Symposium (RAID)*, 2007.

[7] "DigiNotar reports security incident," 2011. http://www.vasco.com/company/press_room/news_archive/2011/news_diginotar_reports_security_incident.aspx

[8] "Security Content Automation Protocol Validated Products," 2011. http://nvd.nist.gov/scapproducts.cfm

[9] Payment Card Industry Security Standards Council, "Payment Card Industry (PCI) Data Security Standard," Tech. Rep. October, 2010. http://en.scientificcommons.org/8858188

[10] "Federal Information Security Management Act (FISMA) Implementation Project," 2011. http://csrc.nist.gov/groups/SMA/fisma/index.html

[11] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about Datalog(and never dared to ask)," *IEEE Transactions on Knowledge and Data Engineering*, vol. I, no. 1, 1989.

[12] Y. Deswarte, L. Blain, J.-C. Fabre, "Intrusion tolerance in distributed computing systems," *IEEE Symposium on Research in Security and Privacy*, 1991.

[13] "Syslog-ng Logging System," 2011. http://www.balabit.com/network-security/syslog-ng/

[14] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines," in *USENIX Security Symposium*, 1998.

[15] D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Transactions on Storage*, vol. 5, no. 1, pp. 1–21, Mar. 2009.

[16] P. Lincoln, P. Porras, and V. Shmatikov, "Privacy-preserving sharing and correction of security alerts," in *USENIX Security Symposium*. 2004.

[17] D. Xu and P. Ning, "Privacy-Preserving Alert Correlation : A Concept Hierarchy Based Approach," in *Annual Computer Security Applications Conference (ACSAC)*, 2005.

[18] B. Ribeiro, W. Chen, G. Miklau, and D. Towsley, "Analyzing privacy in enterprise packet trace anonymization," in *Network & Distributed Systems Security Symposium (NDSS)* , 2008.

[19] J. King, K. Lakkaraju, and A. Slagell, "A taxonomy and adversarial model for attacks against network log anonymization," in *ACM Symposium on Applied Computing (SAC)*. 2009.

[20] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Network & Distributed Systems Security Symposium (NDSS)*, 2003.

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

[22] M. Montanari and R. Campbell, "Attack-resilient compliance monitoring for large distributed infrastructure systems," in *IEEE Conference on Network and System Security*. 2011.

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

[24] K. Walzer, T. Breddin, and M. Groch, "Relative temporal constraints in the Rete algorithm for complex event detection," *International conference on Distributed Event-Based Systems (DEBS)*, 2008.

[25] X. Ou and S. Govindavajhala, "Mulval: A logic-based network security analyzer, " in *USENIX Security Symposium*, 2005.

[26] A. Adi and O. Etzion, "Amit-the situation manager," *The International Journal on Very Large Data Bases*, vol. 13, no. 2, pp. 177–203, 2004.

[27] B. Sangster, T. O'Connor, T. Cook, R. Fanelli, E. Dean, W. Adams, C. Morrell, and G. Conti, "Toward instrumenting network warfare competitions to generate labeled datasets," in *Workshop on Cyber security experimentation and Test (CSET)*. 2009.

[28] D. Kotz, T. Henderson, I. Abyzov, and J. Yeo, "CRAWDAD data set dartmouth/campus (v. 2009-09-09)," http://crawdad.cs.dartmouth.edu/dartmouth/campus. 2009.

[29] E. Fidler, H. Jacobsen, and G. Li, "The PADRES distributed publish/subscribe system," *Feature Interactions in Telecommunications and Software Systems*, 2005.