Multi-Organization Policy-based Monitoring

Mirko Montanari, Lucas T. Cook, Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
{mmontan2, ltcook2, rhc}@illinois.edu

***Abstract*—The monitoring of modern large scale infrastructure systems often relies on complex event processing (CEP) rules to detect security and performance problems. For example, the continuous monitoring of compliance to regulatory requirements such as PCI-DSS and NERC CIP requires analyzing events to identify if specific conditions over the configurations of devices occur. In multi-organization systems, detecting these problems often requires integrating events generated by different organizations. As events provide information about the infrastructure' internal structure, organizations are interested in reducing the amount of information shared with external entities.**

This paper analyses the problem of detecting policy violations in network infrastructure systems managed by two organizations (e.g., a cloud user and a cloud provider). We focus on CEP monitoring systems and we introduce two protocols for selecting the events to share between the two organizations to ensure the detection of all possible policy violations. Our experimental evaluation shows that reciprocal information sharing between the two organizations significantly reduces the amount of information to transfer. In our SNMP monitoring test case, we obtain a 80% reduction in the information shared by any single organization.

I. Introduction

Policy-based event monitoring is often used in the management of large computer systems. Policies can identify undesirable conditions that should be addressed by network administrators. For example, in the area of network security, regulatory policies such as NERC CIP [1], FISMA [2], PCI-DSS [3] include requirements about the configurations of network systems that ensure a minimum level of security. A monitoring system can verify that a system is compliant to such policies by integrating events containing information about the system's operations. When a network infrastructure is managed by multiple organizations, the detection of such undesirable conditions requires integrating events that organizations might be reticent to share with external entities [4]. As multi-organization systems are common in today's infrastructure (e.g., cloud or critical infrastructure systems), reducing the number of events to share can foster the adoption of compliance monitoring and potentially increase the minimum level of security in large systems.

This paper analyses the problem of sharing information between two organizations to check the compliance of an infrastructure to complex policies. We represent the problem of compliance monitoring using Complex Event Processing (CEP). In CEP, simple events describing the state of the system are analyzed using rules and converted into complex events which can represent policy violations. In multi-organization infrastructures, these complex events need to be detected on event streams generated by different organizations. Using our approach we can detect such complex events while minimizing the information shared between the two organizations.

Modern multi-organization systems already have a limited ability of validating the compliance of their infrastructure to policies. For example, in cloud computing, cloud providers and cloud users share the burden of ensuring compliance to security policies [5]. Cloud providers such as Amazon AWS provide the ability of building services that are compliant to regulations such as FISMA and PCI-DSS. Similarly, in the area of critical infrastructure systems in the US, the regional power companies that compose the power grid need to show compliance to NERC CIP regulatory requirements. However, in both cases, policies are simple and designed to be validated independently by each organization. As the security requirements become more complex, it is unclear that such a separation is possible. Using our approach, administrators can automatically identify which policies can be validated by each organization and which events need to be shared to validate the remaining policies.

We use a logic-based approach [13] to represent policies as complex events. We use information about the *completeness* of the information collected by the monitoring systems of each organization to identify the events to share for detecting all policy violations. We show that increasing the amount of information shared by one organization reduces the amount of information shared by the other. We define different information sharing strategies suited for different situations: an asymmetric *pull* strategy suited for when an organization is subordinate to the other and willing to share events unconditionally; and a symmetric *push-pull* strategy for when the two organizations are peers and willing to reveal an event only if it can be shown that such an event is important for the overall compliance.

The contribution of the paper can be summarized as follows.

1) We define the problem of validating the compliance of a multi-organization infrastructure to event-based policies.
2) We refine the concept of *need-to-know* and suit it to our scenario of policy compliance monitoring.
3) We introduce two information sharing strategies for validating compliance based on *monitoring completeness* and on reciprocal information sharing.
4) We evaluate our information sharing strategies using SNMP monitoring data and we show an 80% reduction in the information shared by any single organization.

The rest of the paper is structured as follows. Section II

describes related research efforts in similar areas. Section III defines our problem of multi-organization infrastructure policy compliance. Section IV describes our information sharing strategies. Section V shows our experimental evaluation. Finally, Section VI concludes our work.

II. Related Work

The interaction between devices and services managed by different organizations is a common aspect in computing. The problem of sharing events between organizations has been analyzed in the context of pub/sub systems. Most previous work defines event confidentiality through explicit access control policies. Using these policies, pub/sub systems decide if an event can be forwarded to another organization.

In particular, Singh et al. [7] and He et al. [8] focus on the healthcare domain. Singh et al. [7] introduce a system where events are sent to other domains only when certain conditions on the recipient are satisfied. Such an approach enables the specification of explicit "need-to-know" policies in event systems. For example, a pharmacist can receive events about the existence of a prescription without receiving information about the symptoms of the disease. In our case we do not have an explicit "need-to-know" policy. Our approach provides methods for computing such "need-to-know" from a policy representing a complex event to detect. He et al. [8] are interested in hiding specific *private patterns* from a stream of events. They analyze the complexity of algorithms that maximize the amount of public events published without revealing private information. We are interested in computing the information to share for validating policies.

In general CEP systems, Evans et al. [9] propose to tag events with labels and use such labels to enforce access control. The labels represent an explicit access control policy that is not available in our setting. Denker et al. [10] analyzed the tradeoff between need-to-protect and need-to-share through the application of downgrading of data. Their model focuses on continuous data provided by GPS systems. However, their quantitative downgrading model is not suited for systems where events are discrete. We provide methods for reducing the amount of information shared while providing a complete validation of policies over discrete events.

Other approaches in the area of security focused on defining data anonymization strategies for performing a collaborative section of attacks across several organizations. For example, Lincoln et al. [11] introduce a technique for removing critical data from network traces. However, these techniques are not easily applicable to general CEP systems as they are strongly related to the semantic of the data.

The idea to use knowledge about the completeness of information for validating distributed queries locally was first introduced by Denecker et al. [12]. Our approach builds on top of such technique and extends it to queries that can be partially answered locally and partially on a remote dataset.

III. Multi-Organization Policy Compliance

The management of large-scale infrastructure systems often relies on policies to define undesirable conditions of the systems. For example, in the area of security, PCI-DSS policies define that critical services need to be protected by firewalls; other policies might require anti-virus software to be installed on every machine. More complex policies might restrict the types of network services provided by a device under certain network conditions. When such conditions are identified, corrective actions are taken by operators to reduce the exposure of the system to potential attacks.

CEP systems use events to represent infrastructure monitoring information. Events are generated by different sources such as SNMP monitoring agents running on devices, intrusion detection systems (IDS), or applications. We use complex events to represent policy violations: a violation is detected when a specific sequence of events occurs in the system. For example, we consider a monitoring system receiving events carrying information about firewall configurations, about connections of devices to networks, and about which devices are critical in the current systems' configuration. By analyzing event sequences, we can detect when one of such critical devices is connected to a network which is not protected by a firewall.

When multiple organizations are involved in the management of an infrastructure, complex policies might require integrating knowledge about events that occur in a portion of the infrastructure managed by the other organization. For example, a regulatory policy can specify that access to machines storing restricted data should be allowed only to personnel meeting certain criteria (e.g., export-controlled information in the U.S. can be accessed only by citizens or U.S. residents). In a virtualized cloud environment, admin access to the Host VM storing the restricted data should be restricted as well. Detecting violations to such a policy is possible only if cloud providers and users share information about their operations. In this paper, we provide a general mechanism to identify which events should be shared across organizations for identifying policy violations.

A. Event Model

We represent events and rules using a logic-based approach [13]. For infrastructure monitoring, we are interested in reconstructing a view of the system's operations. We use Datalog predicates to represent events. We represent long-lived states of the system (e.g., the fact that a connection exists between two machines for a certain time) by associating a time interval to each event: the start time and the end time of the interval are the last two parameters of predicate. Events can have an unspecified end time if they provide information about states still holding when the event is generated.

We represent policy violations as complex events and we define them using deductive rules expressed in Datalog with negation. We store events into a Datalog KB, and we identify complex events and violations by computing a fixpoint model of the KB. For example, a policy can specify that a violation occurs when a device running a critical service connects to a server that is vulnerable. A monitoring system can collect events about the software running on a device (e.g, $\texttt{runs}(host_1, pid_1, apache)$), which

software is critical (e.g., $\texttt{critical}(apache)$), the network connections that are open by the given programs (e.g., $\texttt{connects}(host_2, pid_2, host_1, 80)$), and the service running on specific ports (e.g., $\texttt{binds}(host_1, pid_1, 80)$). As these conditions are defined upon system states that need to hold at the same time, we consider only events for which the end time is not set. When the system enters in such a state, violations can be detected immediately. We represent this by omitting the time parameters in the event predicates. The overall rule is represented as follows.

$$\texttt{violation}(X, S_X) \leftarrow \texttt{runs}(X, P_X, S_X), \texttt{critical}(S_X), \texttt{connects}(X, P_X, Y, PORT), \texttt{binds}(Y, P_Y, PORT), \texttt{runs}(Y, P_Y, S_Y), \texttt{vulnerable}(S_Y) \quad (1)$$

More general temporal constraints are represented using Allen operators [14] and time windows. The KB stores all events which are potentially relevant according to the time constraints of the rules [6]. For example, if a rule has a 5 minute time window, events older than 5 minutes are discarded. Time relations are represented by predicates which parameters are the timestamps of the events. For example, we can be interested in generating a complex event indicating the name of the software binding to a given port. We have two events, one indicating that a software sw is running with PID pid on $host$ from time t_1 to time t_2 (i.e., $\texttt{runs}(host, pid, sw, t_1, t_2)$ and one indicating that a program with PID pid is binding to a port $port$ on the same $host$ (i.e., $\texttt{binds}(host, pid, port, t_3, t_4)$). We specify that the second event needs to occur *during* the first event using the predicate $\texttt{during}(t_1, t_2, t_3, t_4)$. In the rule processing, we translate it into the set of constraints $t_1 < t_3, t_4 < t_2$.

In our multi-organization scenario, we consider two organizations A and B. We indicate with K_A and K_B the sets of events collected by the monitoring system of each organization. We separate the rules defining complex events into two sets. First, a set R that contains rules defining complex events. Second, a set V that specifies complex events associated with *violations* of a policy. We indicate with K a KB containing the events, and we define a set K^+ as the fixpoint model of K with the rules R and V. If we indicate with $K = K_A \cup K_B$ the set of events collected by both systems, we say that a violation v exists in a system if $K^+ \vdash v$. If v exists, we say that the infrastructure is not compliant to the policy defining v.

Applying reasoning independently in each organization KB leads to *incomplete* and *incorrect* results: necessary events for the detection of a violation or events which presence is necessary for compliance might be stored in the remote KB and not considered by the local reasoning process. The lack of completeness and correctness are a consequence of the closed-world assumption used by Datalog reasoning. In our case, an event not in K_A can still be true in the system and stored in K_B. Guaranteeing correctness and completeness requires identifying a set of events E that can be transferred from B to A to ensure that the evaluation of the policy is complete.

Our model is based on a few simplifying assumptions. We assume that the two organizations agree on the policies to use for creating complex events and that the same events and policies are used in both organizations. We assume that organizations behave according to the protocols and do not provide false information (i.e., honest-but-curious attack model). Additionally, we assume that clocks are synchronized so that event timestamps are comparable across organizations. The first assumption generally holds for policies specified by regulatory agencies, or when knowledge of policies themselves do not provide competitive advantage to the other party. The second assumption generally holds in organizations that are collaborating for providing a service. Periodic auditing of the infrastructure can be used to verify compliance to the protocol at a later time. The third assumption generally holds if clocks are synchronized to an external source (e.g., NTP) and if we consider policies which are not strictly dependent on causality and event ordering. In our experience, infrastructure security policies specified in PCI-DSS or FISMA do not require strict synchronization between events.

B. Policy Violations of Local Resources

An organization is generally interested in detecting policy violations that relate to its own resources. For example, in the case of a simple policy requiring a host-based IDS to be running on each machine, an organization might be interested in receiving violations only for the machines that it manages. We associate each resource in the system to a domain. We assume that each resource has a unique name (i.e., a URI). We define a set U containing all resources in the overall system, and its subsets D_A and D_B representing the sets of resources owned or managed by each organization.

An organization defines the violations of interest by specifying—in the policy itself—the domains over which variables are quantified. For example, we can add domains to the policy in Eq. 1 to specify that organization A is interested in receiving all violations about its own machines as follows:

$$\forall X \in D_A, \forall P_X, S_X, Y, P_Y, S_Y, PORT \in U \quad \texttt{violation}(X, S_X) \leftarrow \texttt{runs}(X, P_X, S_X), \texttt{critical}(S_X), \texttt{connects}(X, P_X, Y, PORT), \texttt{binds}(Y, P_Y, PORT), \texttt{runs}(Y, P_Y, S_Y), \texttt{vulnerable}(S_Y), \quad (2)$$

where X and Y are hosts; P_X and P_Y are PIDs of processes; S_X and S_Y are identifiers of software packages; and $PORT$ is a network port number. We identify the interest in violations involving resources of organization A by restricting the value of the variable X to D_A. However, we cannot restrict the domain of other variables. For example, restricting the domain of Y to D_A would make the policy identify only violations that involve connections between two hosts within A. By restricting only the domain of the variables in the head of the rule (i.e., in the violation statement), we ensure that we find all violations in K^+ that relate to resources in D_A.

C. Need-to-Know Events

For an organization A and a violation v, our approach is based on identifing a set of events $E \in K_B$ so that $K_A \cup E \cup R \cup V \vdash v \Leftrightarrow K^+ \vdash v$. That is, E contains the events that should be shared by K_B for detecting the violation. When this set of events is minimal, we call it a *need-to-know event set*.

Events in such a set might carry less information than the original events in K_B and still be useful in determining the presence of a violation. In particular, we can mask the name of some resources mentioned in the statement. An event relates n resources with each other under a specific relation. For example, the event $\texttt{runs}(host_1, pid_1, apache)$ relates the resource $host_1$ with the PID pid_1 and with the software $apache$. Often, we can detect a violation by knowing that a relation exists between a resource and an undefined resource with specific characteristics. For example, we can consider a rule $\texttt{violation}(P) \leftarrow \texttt{runs}(H, PID, P)$, $\texttt{vulnerable}(P)$ and assume that A maintains a list of vulnerable programs, while B provides the actual services. If A knows $\texttt{vulnerable}(apache)$, the only information missing from B is the knowledge about the existence of at least one host that runs such a service. As organization A is only interested in knowing which vulnerable programs are run and not the actual name of the host running them (i.e., the host name is not part of the parameters of `violation`), we can remove the host name from the events passed to A.

In first order logic we represent the existence of a host running $apache$ using existential quantification as in $\exists H, P : runs(H, P, apache)$. This statement expresses the fact that `runs` is true for some value of H, P. Multiple pieces of partial information can be related to each other to express that two events are related to the same resource. For example, we can express that some critical server runs apache using $\exists H, P : critical(H) \wedge runs(H, P, apache)$. We define *knowledge units* to be existentially quantified expressions in the form $\exists V_1, \ldots, V_n : e_1 \wedge \ldots \wedge e_k$. In our Datalog framework, we represent knowledge units using Skolemization: we substitute the existential quantified variables with unique constants. As Skolemization preserves satisfiability, if a violation exists, we can find it in the Skolemized version of the system.

IV. Information Sharing Algorithm

For ensuring the completeness and the correctness of the monitoring process, an organization needs to ensure that the need-to-know events are among the events shared with the other organization. However, identifying such a minimal set is challenging, as it depends on the effect of each event on the compliance state of the other organization. We have a tradeoff in sharing: the more an organization shares information about its events, the better the other organization can reduce the events to share by better identifying the need-to-know events.

Without knowledge about any of other organization's events, all events relevant to the policies need to be shared. On the other side, a complete knowledge of the other organization's events permits to select and share only the events that are part of the minimal need-to-know set. We introduce two intermediate approaches. Our first strategy uses the knowledge about the completeness of the information collected by the monitoring system of A to provide a first reduction in the amount of information that B needs to share. Our second strategy uses reciprocal information sharing from A to B to further reduce the number of events shared by B.

The process of detecting violations is as follows. An organization A interested in detecting a set of violations V_A analyses its policies and its past events to create a set of persistent queries P over the stream of events of the other organization. Events in B matching queries are continuously sent back to A. Collectively, these events form a set of events E' that is guaranteed to contain E.

A. Completeness of Local Information

The monitoring system of an organization focuses on acquiring events from a specific set of resources: the resources under the control of the organization. For such resources, we might be able to acquire *complete* information. For example, a monitoring system generating events $\texttt{overloaded}(H)$ with $H \in D_A$ is complete if it is monitoring the state of all hosts in D_A. In such a case, if a policy needs to match events $\texttt{overloaded}(H)$ with $H \in D_A$, all the relevant events can be found on the local knowledge base K_A. Hence, for complete statements, the closed-world assumption of Datalog holds in the local KB and reasoning based on them is correct and complete.

We model the completeness of knowledge acquired by the monitoring system with a *completeness KB* (CKB). A CKB describes patterns of events about which the local monitoring system ensures that we have complete knowledge. A CKB depends on the structure of the local monitoring system and it is composed of two types of statements: simple completeness statements and conditional completeness statements. An example of CKB is shown in Fig. 1.

A *simple completeness statement* is a pattern defining events for which we have complete local knowledge. If a policy is looking for a certain pattern of events and such a pattern can be described by a simple completeness statement, then all events matching it can be found in the local KB. A simple completeness statement is expressed by a statement and by the domains of its variables. We indicate it with the syntax $\forall V_i \in D : st(V_1, \ldots, V_n)$. A statement $b_i(U_1, \ldots, U_n)$ matches a simple completeness statement $st(V_1, \ldots, V_n)$ when $b_i = st$ and for all $i, 1 \leq i \leq n$, $domain(U_i) \subseteq domain(V_i)$. For example, the simple completeness statement for the `overloaded` event above is $\forall H \in D_A : \texttt{overloaded}(H)$.

A *conditional completeness statement* provides a restricted notion of completeness of an event pattern. For example, we can define a set of events $\texttt{critical}(P)$ indicating that a particular software P is currently critical to the organization operations. By using simple completeness statements, we would be able to express that, given any program P, a monitoring system has generated events indicating if such program is critical to the system as a whole. However, most monitoring systems would be able to decide only which

$\forall X \in D_A : \texttt{overloaded}(X)$
$\forall X \in D_A, P, PID \in U : \texttt{runs}(X, PID, P)$
$\forall X \in D_A, PID, P \in U :$
$\texttt{critical}(P) \leftarrow \texttt{runs}(X, PID, P)$

Fig. 1. Example of completeness KB for organization A containing simple and conditional completeness statements

programs are critical to the local organization, and might not be aware of the critical programs for the other organization. We represent such restriction of knowledge by using *conditional completeness statements*. In our example, we express that a monitoring system generates such `critical` events only for the programs that are currently running on its devices by specifying the following: $\forall X \in D_A, \forall P \in D : \texttt{critical}(P) \leftarrow \texttt{runs}(X, P)$. Such statement indicates that, given a policy containing an event pattern $\texttt{critical}(P)$, we can consider the pattern local only if the values of P are restricted to the programs running on the machines in the domain D_A. In general, we represent conditional completeness statements as $\forall V_i \in D : st \leftarrow c_1, \ldots, c_m$.

Given a policy, our information sharing strategies start by performing a *completeness analysis* to identify the information provided by the local monitoring system. The completeness analysis takes a policy $v \leftarrow p_1, \ldots, p_n$, a completeness knowledge base CKB, and determines which statements p_i can be found completely on the local knowledge base K_i. We call such statements *local-complete*. We indicate the set of local-complete statements with S_L, and use S_R to indicate the others. A non-empty S_R indicates that events affecting the result of the policy compliance process might be stored in K_B.

For example, the following policy can be answered completely in the knowledge base subject to the CKB in Fig. 1.

$$\begin{array}{ll} \forall M \in D_1, \forall S \in D & \\ \texttt{violation}(M, S) & \leftarrow \texttt{overloaded}(M), \\ & \texttt{runs}(M, PID, S), \texttt{critical}(S). \end{array} \quad (3)$$

The statements `overloaded` and `runs` are complete because of the simple completeness conditions in the CKB. The statement `critical` is complete because of the conditional completeness statement $\texttt{critical}(\ldots) \leftarrow \texttt{runs}(\ldots)$. In this case, $S_R = \emptyset$ and $S_L = \{\texttt{overloaded}(M), \texttt{runs}(M, PID, S), \texttt{critical}(S)\}$.

If the set S_R is not empty we need to acquire all relevant events from K_B. We introduce two strategies for defining the set of persistent queries that provide a complete and correct detection of all policy violations. The first strategy, called *asymmetric pull strategy*, creates a set of queries P from the set S_R. Such a set of persistent queries is independent from the events contained in K_A and does not provide B any information about events. The second strategy, a *symmetric push-pull strategy* determines the queries in P using both S_R and the current state K_A of the system. Added or removed events in K_A can add or remove queries. Using the symmetric strategy, A can reduce the amount of events requested from B at the cost of revealing some of its internal state.

B. Asymmetric Pull Strategy

The first strategy uses the set S_R to acquire from K_B all events relevant to the policy. If the statements in $S_R = \{p_1, \ldots, p_n\}$ are simple events, creating a set of queries $P = \{(p_1), \ldots, (p_n)\}$ is sufficient for ensure completeness and correctness: all events matching any of the statements relevant for the policy that cannot be found completely on K_A are sent back to A. Such a process guarantees that all events potentially relevant to the process are known by A.

When the statements in S_R are complex events, we need to analyze recursively the policy to identify all simple and complex events that can contribute to it. In our proofs, we take advantage of the connection between complex event processing and relational algebra formalized by Bry et al. [13]. Given a knowledge base K, we use the symbol $\sigma_{p_i}(K)$ to indicate a knowledge base obtained by selecting only statements in K for which there exists a substitution of variables that unify with the given statement p_i. Given a set of statements B, we define the *extended set* B' by adding to the set B the statements d_j contained in the rules $h \leftarrow d_1, \ldots, d_m$ where a $p_i \in B$ unifies with h. We continue until we analyzed all statements in B'.

Lemma 1. *We take a violation v, a rule $v \leftarrow p_1, \ldots, p_n$, a knowledge base K containing ground statements and the sets of rules R. We consider a set $B = p_1, \ldots, p_n$ and its extended set B'. We have that $K \vdash v \Leftrightarrow \cup_{b_i \in B'} \sigma_{b_i}(K) \vdash v$.*

Proof Sketch: Proving ($\Leftarrow$) is simple: $\bigcup_{b_i} \sigma_{b_i}(K)$ is a subset of K, so if it can prove v then K can prove v. The other direction can be proven by contradiction: assume that there is an event not in B' that is necessary in the proof of v. This event is either part of the policy or generated to contribute eventually to the policy. However, events of this type are included in B' by definition. ■

Theorem 1. *Given two sets K_A, K_B, a rule $V = \{v \leftarrow p_1, \ldots p_n\}$ with a body containing $S_L = \{p_1, \ldots, p_{r-1}\}$ statements and $S_R = \{p_r, \ldots, p_n\}$ statements. Let S'_R be the extension of S_R on the rules R.*

For every v we have that $(K_A \cup (\bigcup_{b_i \in S'_R} \sigma_{b_i}(K_B)) \cup R) \vdash v \Leftrightarrow K^+ \vdash v$.

Proof Sketch: For $P = S_L \cup S'_R$, the lemma implies $(\bigcup_{p_i \in P}(\sigma_{p_i}(K_A \cup K_B)) \cup R) \vdash v \Leftrightarrow K^+ \vdash v$. By definition of completeness, for each $p_i \in S_L$, we have that $\sigma_{p_i}(K_A \cup K_B) = \sigma_{p_i}(K_A)$. Hence, we can rewrite the first part of the expression as $\bigcup_{p_i \in P} \sigma_{p_i}(K_A) \bigcup_{p_i \in S'_R} \sigma_{p_i}(K_B) \wedge R$. Because K_A is a superset of $\bigcup_{p_i \in P} K_A$, we can rewrite the entire expression and obtain $(K_A \cup (\bigcup_{p_i \in S'_R}(K_B)) \cup R) \vdash v \Leftrightarrow K^+ \vdash v$ ■

C. Symmetric Push-Pull Strategy

The second strategy uses the set S_R and the events in K_A to create a set of persistent queries P selecting a narrower set of events E'. This narrower selection is obtained by providing to B some limited information about K_A so that only events that are potentially relevant to a violation are delivered to A.

Intuitively, the strategy is based on selecting the events matching on K_A the local part of the policy to determine

a set of specific "missing events" that, if present in K_B, would create a violation. In particular, we consider the set $S_L = \{p_1, \ldots, p_{r-1}\}$ as a query $L = (p_1 \wedge \ldots \wedge p_{r-1})$. For each set of events matching the query, we substitute the values of the variables (substitution γ_i) in the non-local statements S_R. After this process, we call statements in $\gamma_i(S_R)$ that now have at least a ground parameter *boundary statements*. These statements are submitted as persistent queries to organization B. Events matching these queries are returned to A. A then adds the new values for the variables to each substitution γ_i and repeats the process until all statements are considered.

1) We take a policy $v \leftarrow p_1, \ldots, p_n$. We assume that all p_i are simple events. We will see below how to generalize it to complex events.
2) Starting from S_R and S_L of the policy, we compute the set S_B of statements in S_R that share at least one variable with statements in S_L. S_B is the *boundary set* which represents the remote information about which the local statements have partial knowledge. If we indicate with X_L the variables used in S_L and with X_R the variables used in S_R, we can determine the set of shared variables $X_B = X_L \cap X_R$. If $X_B = \emptyset$ but $S_R \neq \emptyset$, no variables are shared and we revert to the pull algorithm.
3) We construct a local query on K_A by taking the conjunction of the statements in S_L and projecting the result on the variables X_B. We substitute the variables in $S_B = \{p_1, \ldots, p_b\}$ with the substitution γ_i and we create a set of queries $P = \{(\gamma_i(p_1)), \ldots, (\gamma_i(p_b))\}$. The queries P are submitted to organization B.
4) When new events are receive from B, we add the results in the knowledge base and create a new CKB' where we add a conditional completeness condition $\bigwedge_{l_i \in S_L} l_i \rightarrow p_i$ for each $p_i \in S_B$ where $p_{l,i} \in S_L$. We repeat the algorithm with the new CKB' until $S_R = \emptyset$

If an event p_i is a complex event, we add an additional step to the process. We consider the rule heads that unify with p_i, and we apply the algorithm recursively to the respective rules.

The correctness and completeness of this process can be shown with a few considerations. First, if we cannot find a substitution γ_i satisfying S_L on K_A, we have that $K^+ \nvdash v$. This comes from the local completeness of S_L: events in S_L can be found only in K_A or they do not exist in K. If we have a substitution γ_i, we can consider it a partial match of a rule. If we can find events matching the statements $p_j \in S_R$ in K_B that are compatible with the substitution γ_i, then we have found a set of events matching the condition of the rule. By submitting the persistent queries we obtain such a result: either the event occurs and it is delivered to A and added to a K'_A, or it does not occur. In either case, K'_A is now complete in respect to the new events. As we send queries for all γ_i, we can add a new completeness statement $\bigwedge_{l_i \in S_L} l_i \rightarrow p_j$.

Lemma 2. *Given a policy $v \leftarrow p_1, \ldots, p_n$, two sets $S_L = \{p_1, \ldots, p_{r-1}\}$ and $S_R = \{p_r, \ldots, p_n\}$, a set $S_B = \{p_r, \ldots, p_b\}$, a completeness KB CKB, and a set of substitution γ_i obtained by performing the query $p_1 \wedge \ldots \wedge p_{r-1}$ on K_A. We have that $K'_A = K_A \bigcup_{p_j \in S_B} \bigcup_i \sigma_{\gamma_i(p_j)}(K_B)$ is complete with respect with $CKB' = CKB \cup (\bigwedge_{l_i \in S_L} l_i \rightarrow p_j)$.*

Proof Sketch: For p_j to be conditionally complete in K'_A we need to ensure that for all events matching p_j for which there are a set of connected events $\bigwedge_{l_i \in S_L} l_i$, we have that $K'_A \vdash p_j \Leftrightarrow K \vdash p_j$. First, we show that if $K \vdash p_j \Rightarrow K'_A \vdash p_j$. We have that for all γ_i such that $\gamma_i(S_L) \subset K_A$, we submit a query to K_B and we obtain $\gamma_i(p_j)$. In this way we obtain all p_j for which the condition $\bigwedge_{l_i \in S_L} l_i$ are true. Similarly, $K'_A \vdash p_j \Rightarrow K \vdash p_j$ because the set of queries can also identify the lack of the event. ■

Theorem 2. *Given a policy $v \leftarrow p_1, \ldots, p_n$, two sets $S_L = \{p_1, \ldots, p_{r-1}\}$ and $S_R = \{p_r, \ldots, p_n\}$, and a completeness KB CKB, the push-pull protocol is complete, correct, and terminates.*

Proof Sketch: The core of the proof proceeds by induction. Because of space restrictions, we provide only a sketch. Each step of the protocol creates a set S_B^i and acquires the set of events $\bigcup_{p_j \in S_B^i} \bigcup_k \sigma_{\gamma_k(p_j)}(K_B)$ from K_B. From Lemma 2, the new knowledge base K_A^i is complete for $CKB^i = CKB^{i-1} \cup (\bigwedge_{l_k \in S_L} l_k \rightarrow p_j)$. By the conditional completeness, $S_L^i = S_L^{i-1} \cup S_B^{i-1}$ (computed with the new CKB^i), which is used to compute a new S_R^i and S_B^i. By induction, S_L^i will increase in size by some nonzero amount for each i since S_B^i contains disjoint events by definition. Similarly, S_R^i will decrease in size. If $S_B^i \neq \emptyset$ during the protocol, eventually $S_R^i = \emptyset$ and the validation of compliance is performed locally on K_A^m, which is complete and correct by definition. If at any point $S_B^i = \emptyset$, the K_A^i is obtained by running the pull algorithm, which is known to be complete from Theorem 1. Since the universe of events is finite, S_L^i is finite and the protocol will eventually terminate. ■

V. Experimental Evaluation

We perform a set of experiments to measure the ability of our information sharing strategies to reduce the data sent between the two organizations. We quantify the data by measuring the amount of basic information units shared. An information unit is a piece of information that associates a resource with a predicate. For example, an event `computer`($host_1$) associates the resource $host_1$ with the predicate `computer`. We count this event as one information unit. A predicate `hasIP`($host_1$, ip_1) relates the resources $host_1$ and ip_1 to `hasIP` and we consider it composed of two information units. This measure is representative of the information shared and does not depend on the specific definition of the event parameters.

We consider an event dataset collected by monitoring the SNMP state of 10 research hosts for 30 days. These hosts include a mix of laptops, development machines, and a web server. We consider 14 types of messages providing information about resources in the system. The dataset includes information about 500 distinct running programs (associated to 20000 PIDs), 70000 distinct network connections, 50 distinct network services, and 4100 IP addresses. We scale the

dataset to represent a larger set of machines by constructing probabilistic models of the events. The sequence of events in each organization is created by generating events which parameter values are taken from such distributions. Events are timestamped and added to each knowledge base.

As the type of policies that can be specified in a real system can vary widely depending on the interests of the administrators, we evaluate the performance of our approach using a wide range of policies. We randomly generate valid policies which correlate types of events that are semantically related to each other by sharing the values of some variables. We ensure that the shared values are semantically meaningful (e.g., the value for an IP in an event is correlated to the value of IP on another event) by constructing a graph where resource types are nodes, and event types are edges. Starting from a resource type node, we randomly select an event and add it to the policy. We continue until we reach a predefined length of the policy. If we reencounter the same resource type node multiple times, we randomly decide if the event needs to refer to the previous resource (i.e., use the same variable name), or if we expect the event to refer to a different resource (i.e., different variable names).

We create a completeness KB that is consistent with the information collected by our SNMP-based monitoring system. This completeness KB is shown in Table I. The KB of each organization is populated by generating events according to the dataset distribution. We track each resource used in the events and assign it to either D_1 or D_2. The completeness KB allows us to distribute events so that each KB contains the events that would be collected by a monitoring system described by such a completeness KB.

Domain	**Event**
$M \in D_A$	$\texttt{TCPService}(M, S)$
$M \in D_A$	$\texttt{port}(S, P) \leftarrow \texttt{TCPService}(M, S)$
$M \in D_A$	$\texttt{UDPService}(M, S)$
$M \in D_A$	$\texttt{port}(S, P) \leftarrow \texttt{UDPService}(M, S)$
$M \in D_A$	$\texttt{TCPConn}(M, C)$
$M \in D_A$	$\texttt{LocalPort}(C, PORT) \leftarrow \texttt{hasTCPConn}(M, C)$
$M \in D_A$	$\texttt{RemotePort}(C, PORT) \leftarrow \texttt{hasTCPConn}(M, C)$
$M \in D_A$	$\texttt{LocalIP}(C, IP) \leftarrow \texttt{hasTCPConn}(M, C)$
$M \in D_A$	$\texttt{RemoteIP}(C, IP) \leftarrow \texttt{hasTCPConn}(M, C)$
$M \in D_A$	$\texttt{software}(C, SW) \leftarrow \texttt{hasTCPConn}(M, C)$
$M \in D_A$	$\texttt{connState}(C, ST) \leftarrow \texttt{hasTCPConn}(M, C)$
...	...

TABLE I
PORTION OF THE CKB OF OUR SNMP-BACKED MONITORING SYSTEM.

First, we show that our solution limits the overall exchange of information. Without the use of the completeness KB, organization A acquires all events in organization B which are relevant to any of the predicates in the rule. The first experiment measures the fraction of information shared with the increase of the length of the rule. The fraction of information shared is measured as the ratio of information shared over the information relevant to the rule (i.e., events which names appear in the rule). Its results are shown in Fig. 2. When we increase the length of the rule, the fraction of information

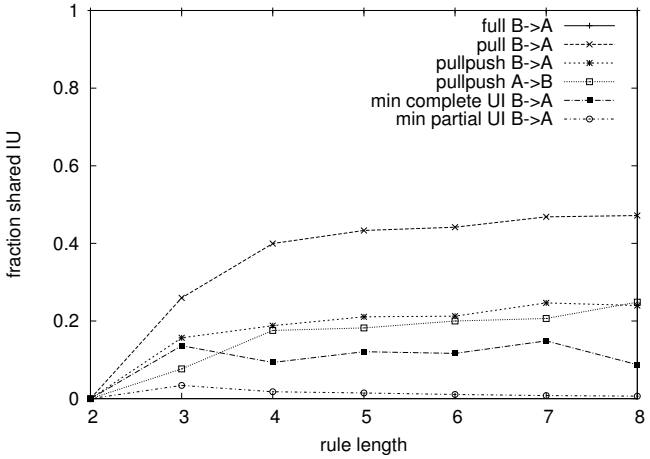

Fig. 2. Information shared with the increase in the length of the rule.

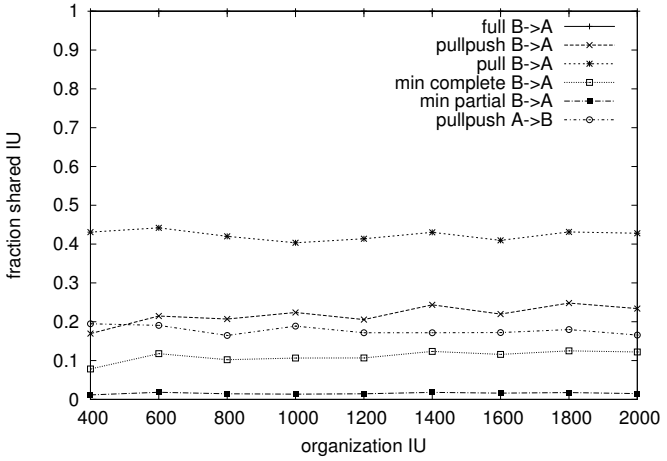

Fig. 3. Information shared with the increase in the number of events generated by the two organizations. We fix the rule size to 5.

shared remains almost constant in all cases. For small rule lengths, most of the information is found locally in organization A. Using the completeness KB and a pull strategy, we can reduce the number of events that need to be transferred by not requiring information about the local portion of the rule. In our SNMP case, this approach approximately halves the number of events shared by organization B. The push-pull strategy further reduces the amount of information to share. In the SNMP case, we reduce the information sharing to about 20% of the information shared in a full sharing strategy (i.e., no completeness KB). To obtain this reduction, organization A needs to share to organization B about the same amount of information. The optimal amount of information shared from B to A can be obtained by transferring all data from organization A. We see that the minimal number of complete events that needs to be shared to identify all violations is about 10% of the relevant information. If we share only partial events (i.e., single information units), we can further reduce this amount to about 1% of the relevant information. In this case, it

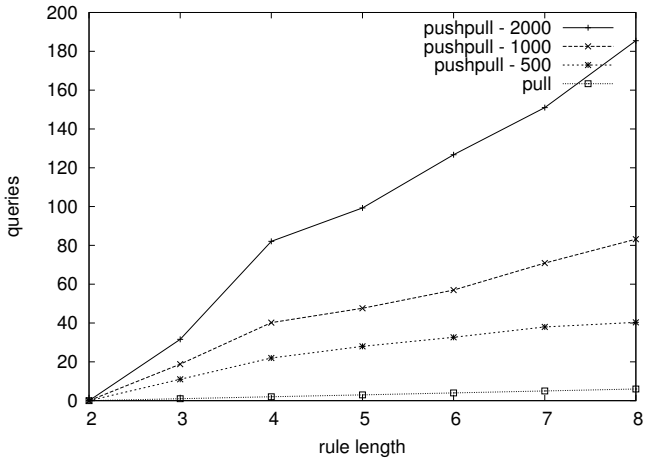

Fig. 4. Number of persistent queries placed on organization B.

is sufficient for organization B to share enough information to pinpoint which resource in A is in violation, without revealing any of its own events that contribute to the actual violation.

The second experiment measures the fraction of information shared with the increase in the amount of events considered in the organizations. We consider a rule of length 5, and we see that the fraction of information that needs to be transferred remains constant and consistent with the previous set of experiments. This data is shown in Fig. 3.

Next, we evaluate the overhead introduced by running our information-sharing algorithms. We measure the average amount of persistent queries that are placed on the organization B event stream for selecting the events to share with organization A. For the case of the pull strategy, the number of queries is proportional only to the length of portion of the rule that cannot be evaluated locally. For the case of the push-pull strategy, the number of queries depends also on the size of the events in organization A that we consider. In the push-pull case, the amount of queries is limited to a few hundred. This data is shown in Fig. 4.

In summary, our experiments show that our techniques can significantly reduce the number of events to be transferred across the two organizations without significantly increasing the overall load on the system.

VI. Conclusion and Future Work

We introduce two solutions to the problem of validating compliance of multi-organization systems to infrastructure security polices. Our solutions are based on specifying the completeness of the information collected by the local monitoring systems of each organization. Our experiments using SNMP data show that an approach that requires reciprocal sharing of events obtains a reduction of 80% in the amount of information shared by one organization when about 20% of the information is shared by the other.

Future work should extend our approach in several ways. First, we assume that organizations are honest-but-curious actors. While accountability can enforce such a behavior, future work could extend our results to a more general adversary model. Second, we assume that organizations are willing to share all events that are relevant to the policies. However, in some cases, organizations might prefer sharing only events not critical to the organization security. Future work should incorporate such sharing constraints in the framework. Finally, future work should extend our approach to the general case of n organizations.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This work was partially supported by a research grant provided by the Boeing Company. 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] North American Electric Reliability Corporation, "NERC CIP 002-009," NERC Tech. Rep., 2007. Available: http://www.nerc.com/page.php?cid=2—20

[2] National Institute of Standard and Technology, "Federal Information Security Management Act (FISMA) Implementation Project." Available: http://csrc.nist.gov/groups/SMA/fisma/index.html

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

[4] J. King, K. Lakkaraju, and A. Slagell, "A taxonomy and adversarial model for attacks against network log anonymization," in *ACM symposium on Applied Computing*. ACM, 2009, pp. 1286–1293.

[5] Amazon Web Services, "Amazon Web Services : Risk and Compliance White Paper," *Amazon AWS Whitepapers*, December, 2011. http://aws.amazon.com/whitepapers/overview-of-security-processes-2/

[6] K. Walzer, T. Breddin, and M. Groch, "Relative temporal constraints in the Rete algorithm for complex event detection," *Proceedings of the second international conference on Distributed event-based systems - DEBS '08*, p. 147, 2008.

[7] J. Singh, L. Vargas, J. Bacon, and K. Moody, "Policy-Based Information Sharing in Publish/Subscribe Middleware," *2008 IEEE Workshop on Policies for Distributed Systems and Networks*, pp. 137–144, Jun. 2008.

[8] Y. He, S. Barman, D. Wang, and J. Naughton, "On the complexity of privacy-preserving complex event processing," in *Proceedings of the 30th symposium on Principles of database systems of data*. ACM, 2011, pp. 165–174.

[9] D. Evans and D. Eyers, "Efficient Policy Checking across Administrative Domains," *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY)*. IEEE, 2010, pp. 146–153.

[10] G. Denker, A. Gehani, M. Kim, and D. Hanz, "Policy-Based Data Downgrading: Toward a Semantic Framework and Automated Tools to Balance Need-to-Protect and Need-to-Share Policies," in *IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY), 2010* . IEEE, 2010, pp. 120–128.

[11] P. Lincoln, P. Porras, and V. Shmatikov, "Privacy-preserving sharing and correction of security alerts," in *USENIX Security Symposium*. USENIX Association, 2004, pp. 17–17.

[12] M. Denecker, A. Cortés-Calabuig, M. Bruynooghes, and O. Arieli, "Towards a logical reconstruction of a theory for locally closed databases," *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 3, p. 22, 2010.

[13] F. Bry and M. Eckert, "Towards formal foundations of event queries and rules," in *Workshop on Event-Driven Architecture, Processing and Systems, held at the International Conference on Very Large Data Bases (VLDB)*, 2007.

[14] J.F. Allen, "Maintaining knowledge about temporal intervals, " *Communications of the ACM*, 26(11), 832-843. 1983