Limiting Data Exposure in Monitoring Multi-domain Policy Conformance*

Mirko Montanari, Jun Ho Huh, Rakesh B. Bobba, Roy H. Campbell
{mmontan2, jhhuh, rbobba, rhc}@illinois.edu

University of Illinois at Urbana-Champaign

Abstract. In hybrid- or multi-cloud systems, security information and event management systems often work with abstract level information provided by the service providers. Privacy and confidentiality requirements discourage sharing of the raw data. With access to only the partial information, detecting anomalies and policy violations becomes much more difficult in those environments.
This paper proposes a mechanism for detecting undesirable events over the composition of multiple independent systems that have constraints in sharing information because of security and privacy concerns. Our approach complements other privacy-preserving event-sharing methods by focusing on discrete events such as system and network configuration changes. We use *logic-based policies* to define undesirable event sequences, and use *multi-party computation* to share event details that are needed for detecting violations. Further, through experimental evaluation, we show that our technique reduces the information shared between systems by more than half, and we show that the low performance of multi-party computation can be balanced out with concurrency—demonstrating an event rate acceptable for verification of configuration changes as well as other complex conditions.

1 Introduction

Monitoring of complex systems for configuration errors, security breaches, or regulation compliance requires a large amount of information to be collected (usually in the form of audit logs) and analyzed. In distributed environments like clouds or cloud-of-clouds [2], this monitoring may require logs to be shared across multiple security domains to detect particular security events. However, some of those logs might contain sensitive information about customers or might have commercial value. Without the necessary confidentiality and privacy guarantees, most organizations will be reluctant to share such privileged logs with others.

* This material is based on work supported in part by a grant from The Boeing Company, and by a grant from 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.

Being mindful of such concerns, this paper proposes a solution that facilitates integration of events across multiple security domains while providing the necessary confidentiality guarantees. With our solution, cloud users or cloud providers can detect problems in their own systems (e.g., a virtual machine running in an IaaS cloud), even if some parts of the infrastructure are being controlled by different organizations. We rely on the definition of "invariants," which capture the correct or desirable operations of the systems of interest. *Only* those events that have the potential to identify violations of those invariants are ever shared, minimizing the amount of events that need to be shared in the first place.

Privacy-preserving techniques that rely on data aggregation and anonymization [23, 27, 17] have been proposed in the past. While those techniques can be effective on policies that use numeric data and thresholds [3, 6], many conditions that are of interest to network administrators require the capability to analyze *discrete events* (e.g., configuration changes in servers or network devices, changes in user information, or component failures). To the best of our knowledge, there is no good solution for aggregating or anonymizing discrete events without losing the details necessary for validating policies.

The proposed technique, on the other hand, detects complex, inter-domain policy violations through careful selection of the events to share across the organization boundaries. We introduce a distributed algorithm that coordinates the interaction among a dynamic set of monitoring servers to guarantee the detection of all policy violations, and we use a cryptographic mechanism called *privacy preserving secure two-party computation* [9] to figure out which events are relevant to a violation, and share only them across the domain boundaries.

We show that parallelism in the event correlation problem makes secure two-party computation practical in our case. Additionally, the lack of a central server where information is analyzed makes our technique suitable for multi-cloud systems or cloud-broker-based systems where new resources and security domains are continuously being added at runtime. Our performance evaluation confirms that our technique is indeed capable of significantly reducing the amount of information that needs to be shared, and that it can handle event loads of popular configuration management systems.

The contributions of this paper are summarized as follows:

1. We describe a policy-based algorithm for detecting violations of invariants across multiple security domains, and provide a proof of correctness of that algorithm.
2. We propose a mechanism for generating a policy-dependent implementation of the two-party computation algorithm using an efficient implementation of garbled circuits [10]. This algorithm identifies the information that needs to be shared while preserving privacy.
3. We demonstrate that parallelism of the event correlation problem can lead to a practical deployment of the two-party computation algorithm.
4. Our experimental evaluation shows that our solution is practical and can reduce significantly the amount of information that needs to be shared across multiple domains.

The rest of the paper is structured as follows. Section 2 provides an overview of related work in the area of multi-domain monitoring and secure information sharing. Section 3 discusses our own monitoring architecture and distributed event correlation protocol. Results from the experimental evaluation are presented in Section 4. Finally, Section 5 presents our conclusions and future work.

2 Related Work

Collaboration between organizations for detecting attacks and other security problems has been an important research topic for several years. The interdependencies between systems that we find in today's cloud computing environments increase the need for such collaboration. However, sharing of information presents several security problems. Monitoring data can provide insights into an organization's computing infrastructure which may give a competitive advantage to rival organizations. Such data can also provide attackers with information about possible vulnerabilities. For that reason, a significant amount of work has been focused on reduction of information sharing, while still permitting the detection of complex event patterns.

Several past techniques focus on hiding log information through anonymization [23, 27, 17]. Lincoln et al. [17], in particular, introduce a technique for removing critical data from network traces. SEPIA [3] provides a threshold-based mechanism for sharing aggregated data about network traffic. Denker et al. [6] use a selective downgrade of GPS data for sharing location data. However, such techniques generally apply to numeric information, and the summarization is strongly dependent on the semantics of the information. Our technique focuses on discrete data that cannot be summarized without loss of the ability to detect policy violations correctly, such as configuration changes, failures of specific machines, or vulnerability information.

Montanari et al. [20] introduce a protocol for the validation of policies across two organizations. The authors use explicit meta-data about the completeness of the information collected from each monitoring system to decide which events to share. Our work does not require explicit metadata, and our approach is applicable to multiple security domains. Additionally, the use of a secure two-party computation protocol further reduces the amount of shared events.

Huh and Lyle [12] propose a trusted computing based approach to enable "blind log analysis," allowing different organizations to freely share raw log data with the guarantees that their raw data will not be revealed to other organizations. A trustworthy log reconciliation service is attested and verified, providing assurance that all the log reconciliation and analysis is performed blindly inside a protected virtual machine. Only the fully processed, privacy-preserving analysis results are made available to other organizations. In contrast, in our approach, only the information required for detecting violations leaves the security domain, reducing the amount of information that need to be shared in the first place. We reduce the reliance on remote software for protecting the confidentiality of data, and we do not rely on the capabilities of remote attestation [5].

Techniques focusing on integrating data in a central server for analysis have also been proposed. Australia's Commonwealth Scientic and Industrial Research Organisation (CSIRO) has developed a Privacy-Preserving Analytics (PPA) software for analyzing sensitive healthcare data with confidentiality guarantees [22]. PPA performs analysis on the original raw data but modifies the output delivered to researchers to ensure that no individual unit record is disclosed. Huh and Martin [13] propose the concept of a "blind analysis server," an attested and verified remote server which allows privileged data analysis to be performed securely and privately. In work closely related to ours, Lee et al. [16] introduce a framework that allows a group of organizations to share encrypted logs with a central auditor. The auditor analyzes the encrypted logs and detects attacks or other policy violations. Our work improves on such approaches in two major ways. First, we remove the need to store centrally the logs collected from all organizations. While having a central authority is feasible in certain situations, in cloud and cloud-of-clouds systems, organizations can integrate resources across multiple providers and provide them to different clients at runtime. Having all entities involved push out their logs to a single central location is impractical and not desirable. Our approach uses a distributed mechanism for correlating data, and security domains interact directly only when they require information about specific external resources.

Other approaches rely on explicit confidentiality policies to define which events to share across organization boundaries. Singh et al. [26] introduce a system defining explicit confidentiality policies on classes of events. Similarly, Evans et al. [7] propose a solution in which access control is enforced by tagging events with labels. Rigid confidentiality policies proposed in those works would not be fully compatible with our multi-domain scenario. As information should be shared only when needed, fixed policies are either too open or too strict. Open policies unnecessarily increase the information shared, while strict policies might make the system unable to guarantee the detection of all policy violations.

3 Multi-domain Event Sharing for Compliance

The increasing complexity of managing and securing large systems has driven the development of policy-based approaches to address the problem [8]. In such approaches, policies or "invariants" identify correct or desirable conditions of the system. Administrators define rules that identify violations of such policies and indicate misconfigurations, vulnerabilities to known attacks, or non-compliance to best-practices that reduce the risk of zero-day attacks. Monitoring systems continuously collect information about the infrastructure to detect violations.

Examples of policy-based approaches can be found in different domains. For example, it is possible to define policies to monitor for compliance with regulatory requirements such as the Payment Card Data Security Standard (PCI-DSS) [24] or the Federal Information Security Management Act (FISMA) [25]. Both regulations mandate a minimum level of security configuration in an infrastructure. PCI-DSS applies to companies handling credit card data, while FISMA

Table 1. Example of a multi-domain policy and events required for the detection of a violation. For each event, we list its source and the information it carries.

Policy example	Events	Source	Res	Description
Not run a critical service on a physical server that is sending malicious traffic	`criticalService`	Private Infrastructure	P, I	Critical service P is running on instance I
	`instanceAssigned`	Cloud Provider	I, S	VM Instance I launched on S
	`badTraffic`	Cloud Provider	S	Malicious traffic detected from S

applies to information systems in the U.S. federal government. Policies define known types of misconfigurations or error situations, and are used to identify quickly the presence of a problem before an attacker can exploit it.

In many modern cloud systems, multiple security domains interact to provide the desired services. For example, in a hybrid cloud environment, services provided by the infrastructure of an organization are integrated with services managed by a cloud provider. In intercloud systems [2], or in systems based on cloud brokers [18], multiple cloud services are integrated to provide a service to cloud users. Such services are provided by a variety of cloud providers, and their selection might depend on dynamic conditions not known beforehand. In such settings, multiple independent monitoring systems acquire information about the infrastructure. Monitoring systems in the private infrastructure acquire application-level information from software running on the local infrastructure and on cloud instances. Monitoring servers managed by the cloud provider acquire information about the physical location of virtual machine instances, the colocation of virtual machines with other customers, and the load of the infrastructure. In these settings, conditions to detect violations can be complex, and analyzing each organization's information independently is not sufficient to detect violations. However, sharing monitoring information outside an organization is often undesirable. Information about the infrastructure configurations provides details about security postures or information about the internals of an organization to competitors.

We can find several examples of policies in enterprise and cloud systems. For example, a policy can specify that a critical service should not depend on services running on machines that process a lot of external traffic; another policy might require a computer joining a private network not to run a certain set of services. In both cases, it might be desirable not to reveal the entire state of a system at once, but only when certain conditions occur. For the purpose of explaining our approach, we use the following example throughout the paper.

Example: An administrator defines a policy requiring that a critical server not be run on a physical server that is sending malicious or unwanted traffic (*e.g.* unexpected port scans). Violation of such a policy can be detected by identifying three events, as shown in Table 1. Such events are generated by different information sources: deployment systems in the cloud provider indicate that a new instance has been created; SNMP agents on virtual machines generate informa-

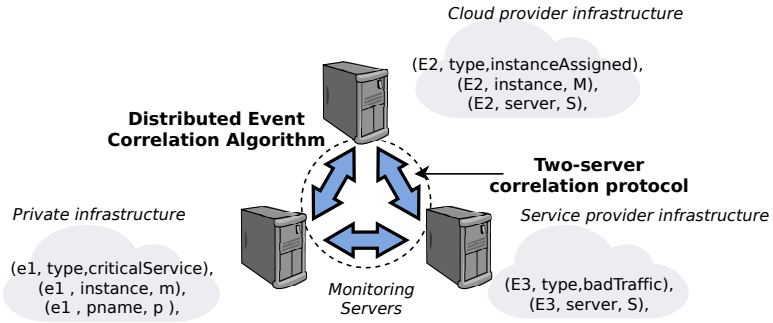

Fig. 1. Architecture of our monitoring system. Multiple monitoring servers are placed in different security domains. Servers communicate to detect violations of policies.

tion about running programs; and network monitoring systems detect malicious traffic from physical servers. Because information sources are in different organizations, detecting violations requires sharing data across domains.

We design a monitoring architecture (see Fig. 1) that supports such data sharing while minimizing the amount of data shared. The monitoring servers within each organization have a copy of the shared policy and collect information about the local infrastructure. The collected information is used to detect local violations without requiring any communication with other servers. Additionally, each server verifiers if the local information could potentially create a multi-organization policy violation if certain conditions are present in other domains. In such a case, the server uses our distributed event correlation algorithm to check the presence of the condition on remote monitoring servers without revealing information about the local infrastructure, unless a violation is actually detected. We use a distributed naming system to identify monitoring systems potentially containing other portions of the event sequence that could cause a violation. When policies are complex, our policy rewrite algorithm splits the policy into a sequence of simpler conditions that can be checked by communicating with a single monitoring server at each step. The policy rewrite is performed independently on each server, and only local information is used to identify the remote servers with which the local server needs to communicate to continue the processing. We show that, using such local actions, our algorithm identifies the same policy violations found by integrated events in a single server.

3.1 Policy Analysis

Violations of policies are rare events. For that reason, monitoring systems collect a large amount of information that does not contribute to violations. Our correlation algorithm identifies which events might contribute to violation of cross-domain policies and shares only those events with other domains.

To perform such an analysis, we take advantage of the semantic structure of the data in infrastructure monitoring systems. An infrastructure monitoring

Constraint	Description
precedes	$x^+ < y^-$
meets	$x^+ == y^-$
overlaps	$x^- < y^- < x^+ < y^+$
during	$y^- < x^- < x^+ < y^+$
starts	$x^- == y^-, x^+ < y^+$
finishes	$x^+ == y^+, x^- > y^-$
equals	$x^- == y^-, x^+ == y^+$

Fig. 2. Temporal policy constraints. x^- is the starting time of an event x, and x^+ is its end time.

$1 : (E_1, \text{type}, \texttt{criticalService}),$
$2 : (E_1, \text{instance}, I), (E_1, \text{pname}, P),$
$3 : (E_2, \text{type}, \texttt{instanceAssigned}),$
$4 : (E_2, \text{instance}, I), (E_2, \text{server}, S),$
$5 : (E_3, \text{type}, \texttt{badTraffic}), (E_3, \text{server}, S),$
$6 : [E_1 \text{during} E_2] \wedge$
$7 : ([E_2 \text{overlaps} E_3] \vee [E_2 \text{during} E_3])$
$8 :\rightarrow (v_1, \text{violation}, I)$

Fig. 3. Policy requiring that a critical service not be run on a physical server that is sending malicious traffic.

system is an event-based system that collects information about the state of a set of entities, such as computer systems, users, software programs, or network connections. We define these entities as "resources." A violation of a policy is the presence of a particular state in a set of related resources (e.g., in Table 1, the resources are a VM instance I, a software P, and a physical machine M). Our system finds violations through the identification of a sequence of events that corresponds to incorrect or invalid changes in the state of such resources.

We represent each event as multiple logic statements. Without loss of generality, we use the Resource Definition Framework (RDF). In RDF, each statement is a tuple $(id, property, value)$ composed of three parts. The first part is an event ID, which identifies uniquely the event throughout the system. The second part indicates the name of an event *property* and represents the type of information provided by the statement (e.g., the property `instance` indicates id of an virtual instance). The last part contains the value for the given property. An event is composed of multiple statements having the same event ID. In our example, we represent an event with ID e_1 of type `criticalService` providing information that a VM instance identified with m is running a program p as follows:

$$\begin{aligned} &1 : (e_1, \text{eventType}, \texttt{criticalService}), (e_1, \text{pname}, p), (e_1, \text{instance}, m), \\ &2 : (e_1, \text{startTime}, t_s), (e_1, \text{endTime}, t_e), \end{aligned} \tag{1}$$

A policy identifies a sequence of events by expressing conditions over the logic statements in each event. The condition is represented with a rule expressed using Datalog with negation (we assume typical Datalog stratification and safeness conditions on the rule [4]). In addition, we use seven constraints (and their negation) to represent in interval temporal logic (ITL) all possible temporal relations between events [1]. The constraints are listed in Fig. 2. Using such relations, we can express constraints such as the fact that an event e_1 happens before another event e_2, or that an event e_1 happens while the event e_2 is happening. Using such a language, we represent directly policies expressed as conjunctions and negations of events. We represent disjunctions by creating multiple rules, each representing an alternative in the selection of the events that can occur.

```
1: function SubPolicy(V, P, PR)
2:     for all e1, e2 ∈ P sharing variable V do
3:         VL = vars(e1) ∪ vars(e2) // we select all variables used in the two events
4:         PR = create a rule "e1 ∧ e2 ∧ [timec(e1, e2)] → partial_i(VL)";
5:         // We create P' by removing e1, e2 from P and replace them with partial_i
6:         P = (P \ e1 \ e2) ∪ partial_i(VL)
7:         if size of P' is 2 then return P'
8:         else
9:             if P' has no shared variable then
10:                Take two events e_k, e_n ∈ P and generate a new random resource r
11:                Add a property to e_k and e_n to connect them to r
12:            end if
13:            V' = choose a shared variable
14:            return SubPolicy(V', P, PR)
15:        end if
16:    end for
17: end function
```

Fig. 4. Policy rewrite algorithm pseudocode.

Fig. 3 shows the formal representation of the example policy in Table 1. The policy uses the same variable names in the values of the properties `instance` and `server` to define an equality relation between such properties in events E_1, E_2 and E_2, E_3. Temporal conditions are expressed within square brackets, as shown in lines 6-7. The policy is violated if the three events satisfy all conditions.

3.2 Rule Rewrite

Our distributed correlation algorithm splits across multiple servers the process of building subsequences of events that may violate the policy. Each monitoring server *registers* to manage a set of resources in a distributed naming service, and constructs the sequences of events related to them. Servers receive events generated within the security domain about the resources for which they are registered. We use a distributed naming registry based on Zookeeper [14] to maintain an assignment between resources and monitoring servers within each organization, and we expose such an assignment to external organizations through a DNS-based interface: a server obtains the monitoring servers managing a resource through the resolution of a name containing a short hash of the resource.

Monitoring servers identify violations by analyzing events related to the resources they manage and by connecting them with events stored in other monitoring servers. To identify explicitly the relation between resources, we rewrite each policy into an equivalent set of rules called *resource-based rules*.

As resources involved in a policy violation are related to each other, some events contributing to a violation carry information about two or more of the resources involved[1]. Because such a relation between events and resources exists,

[1] Such a condition is common among monitoring policies: if no relation exists between events, any occurrence of certain unrelated events could create a violation.

we can split a policy into a set of rules, each composed of two events which carry information about the same resource. As one of the two events also carry information about some additional resource (otherwise the two resources involved would be unrelated to the other resources), we connect resource-based rules together in the following way. The consequence of a resource-based rule is a new logic statement related to the additional resource; such a statement is used in other rules. In the processing of events, the first rule identifies two matching events, and creates a statement indicating that such a match is found. The next rule takes such a statement and integrates it with an additional event; such a process is repeated until all events in the violation sequence are matched.

Fig. 4 describes a greedy version of the algorithm we use in the policy rewrite. Intuitively, the algorithm takes a policy P and selects two events with a resource in common (line 2). A new resource-based rule PR is created (line 4) based on the two events and the time constraints *timec* involving both of them. We replace the two events in the original policy with the new statement $partial_i$ (line 6). If the resulting policy P is composed of two events, the algorithm is complete (line 7), otherwise the execution continues recursively (line 14). If the remaining policy does not have any common variable (i.e., events are unrelated), a new shared resource is created and added to the events (lines 9-12).

As an example, we apply the rewrite algorithm to our policy of Fig. 3. We consider three resources I, S, and P, where the VM instance I is assigned to the server S, and I runs the program P. Our rewrite splits the policy into two resource-based rules. The first rule integrates the events `badTraffic` and `instanceAssigned` related to the physical server S. Because the physical server is managed by the cloud provider, such integration is performed on the cloud provider system. The consequence of such a rule is a statement $partial_1$ that contains a reference to the instance I. We obtained the following resource-based rule, where the variables E_isT and E_ieT represent the start time and end time of the events.

$$\begin{array}{l} 1 : (E_2, \text{type}, \text{instanceAssigned}), (E_2, \text{instance}, I), (E_2, \text{server}, S), \\ 2 : (E_2, \text{startTime}, E_2sT), (E_2, \text{endTime}, E_2eT), \\ 3 : (E_3, \text{type}, \texttt{badTraffic}), (E_3, \text{server}, S), \\ 4 : (E_3, \text{startTime}, E_3sT), (E_3, \text{endTime}, E_3eT), \\ 5 : ([E_2\text{overlaps}E_3] \vee [E_2\text{during}E_3]) \\ 6 :\rightarrow partial_1(I, S, E_2sT, E_2eT, E_3sT, E_3eT) \end{array} \tag{2}$$

In the second step, we consider the statement $partial_1$ related to VM instance I, and we integrate the remaining events `criticalService` generated by the private infrastructure. As the new statement contains all information from the selected events, all temporal constraints and event conditions can still be applied. The result is as follows.

$$\begin{array}{l} 1 : partial_1(I, S, E_2sT, E_2eT, E_3sT, E_3eT), \\ 2 : (E_1, \text{type}, \texttt{criticalService}), (E_1, \text{instance}, I), (E_1, \text{pname}, P), \\ 3 : (E_1, \text{startTime}, sE_1sT), (E_1, \text{endTime}, sE_1eT), \\ 4 : [E_1\text{during}E_2] \rightarrow (v_1, \text{violation}, I) \end{array} \tag{3}$$

Because part of the information about I is contained in the cloud provider and part in the private infrastructure, the two monitoring servers need to communicate for integrating the two events. The next section describes our mechanism for correlating the two events while revealing information only if a match exists.

The correctness of the rewrite algorithm is shown below in Lemma 1.

Lemma 1. *Given a set of rules R generated through the application of Algorithm 4 to a policy P, a set of events $e_1, \ldots, e_n$ creates a violation of P iff it creates a violation of the set of rules R.*

Proof Sketch The rewrite of the formula P creates a tree structure. Each rule is a node in the tree. A node A is a child of a node B if the consequence of the rule A (i.e., the $partial_i$ statement) is a condition in the parent node B. In our example, Eq 2 is a child of Eq 3 because the consequence $partial_1$ of the first rule appears as a condition in the body of the second rule. We prove by induction on such a tree. If the height of the tree is 1, then the condition is trivially satisfied as we have only one rule and $r = P$. Assuming that the height of the tree is n, we prove that the condition is satisfied for $n + 1$. We consider a node A in the n tree. In the $n + 1$ tree, the node A is replaced with a node A' with a child B. B is obtained by considering two events e_1, e_2 from the rule in A and by creating a new rule r_j having such events as body and a new statement $partial_i$ as conclusion. The node A' is created by replacing events e_1, e_2 in A with the $partial_i$ statement. Because the events satisfy all rules at heigh n, the rule r_j is satisfied as the conditions on events e_1, e_2 were satisfied in the tree n. Hence, the statement $partial_i$ is also satisfied. Because the statement $partial$ maintains all the information about matched events, all conditions that were not taken and placed in r_j can still be validated in the original node. Hence, no constraints have been eliminated in this process, and all events that satisfy the rules also satisfy the original policy.

3.3 Event Correlation

The resource-based rules split the identification of the violation in a set of two-event correlations. Such processing is performed independently in each monitoring server: when an event matching a part of the resource-based rule is received, the server triggers a process for identifying the presence of an event matching the remaining part of the rule, even if the event is stored in another domain. The server uses the naming system to identify all servers containing events related to the common resource. For each server, it uses our matching protocol to identify if events matching the remaining portion of the rule are present. If found, the local event is shared. Based on the received event, the remote server repeats the described process and interacts with other servers until a violation is found, or no event matches the resource-based rule.

Because information about a single resource can be spread across multiple domains (e.g., for the same host, a domain might provide network information and another domain provide system information), servers in different domains

can be registered for the same resource. Additionally, because events carry information about multiple resources, a server contains information about resources for which it is not registered. Our algorithm uses a *subscription* process to keep track of data about resources for which the server is not registered. When a server s receives an event relevant to resource r that matches a rule, it contacts the registered server for r to search for matching events. Even when matching events are not found, the registered server maintains a reference to s, as it is known that it contains events relevant to r. When new events are received, the registered server contacts s for correlation. Additionally, when another server s' requests a correlation for r, the address of s is shared, so that s' can correlate its events with s directly.

Theorem 1. *If events $e_1, \ldots, e_n$ satisfy all conditions of a policy, the distributed protocol identifies the presence of a violation.*

Proof Sketch: We assume that there exists a sequence of common resources that connects all events, i.e., $r_1, \ldots, r_n$ such that $\forall e_i \exists r_k, e_j : r_k \in e_i, r_k \in e_j$. If such sequence does not exist, our algorithm introduces new common resources to connect all events. Lemma 1 shows the equivalence between the resource-based rules and the policy. Hence, we show that if two events matching a resource-based rule exist, our distributed algorithm identifies them. By construction, such events have a resource r in common. Hence, given an event e, we need to ensure that a server finds all events e' that share the same resource r. We have three cases.

1. Both events e and e' are received by servers registered for the resource r. According to our algorithm, when an event is received, the server interacts with all servers registered for such a resource. As both servers are registered, the last event received would interact with the server storing the first event.
2. Event e is received by a server s registered for r, and event e' is received by a non-registered server s'. If the servers receive the events in the sequence (e, e'), the arrival of e' triggers a lookup on the naming registry, leading to the identification of the server containing e. If the sequence is (e', e), the server of e is identified: however the correlation protocol returns false, as e is not present yet, and the event e' is not shared. In that case, s saves the reference for s'. When the event e is received, s runs the correlation protocol with s' again and identifies the event e'.
3. Both events e and e' are received by two non-registered servers s and s'. In such a case, the first event triggers a lookup in the naming system, leading to the identification of a server s_r registered for the resource r. The correlation process creates the reference to s in the server. Receiving the event e' triggers the same process. This time, s_r saves the reference to s' and returns the reference to s. The correlation process between s and s' identifies the matching events.

3.4 Privacy-Preserving Matching Protocol

The privacy-preserving matching protocol is initiated between two servers. One peer, called the *gc-client*, initiates the process by picking a resource-based rule

1 : $partial_1(I, E_2sT, E_2eT)$,
2 : $(E_1, \text{instance}, I)$,
5 : $[E_1 \text{during} E_2]$

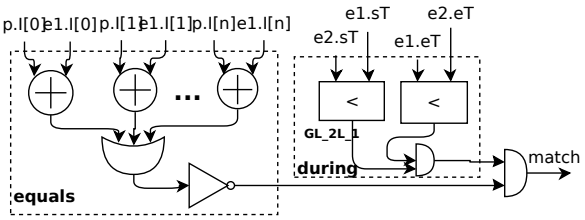

Fig. 5. (left) Simplified resource-based rule containing only constraints requiring input from both events; (right) circuit blocks implementing the resource-based rule.

and an event e for which it wants to find a match. The other peer, called the *gc-server*, considers all local events satisfying the local condition of the rule and, for each event e', executes a *two-event matching* protocol. To speed up the process, the system executes the two-event matching for all pairs (e, e') in parallel.

We use garbled circuits [9] to implement the two-event matching protocol. Garbled circuits are a cryptographic mechanism for performing secure two-party computation. Without the use of cryptography, one server needs to acquire data about both events to validate all constraints (temporal and others) specified in the rule. However, such an approach would reveal a large amount of information to the other party, as all relevant events need to be stored on one server. Using secure two-party computation, each party provides part of the input data and collaborates with the other party through a distributed protocol to determine if two events satisfy the constraints of the rule. The data provided by each party remains hidden, and only the result of the computation is known to both. In the last several years, garbled circuits have been shown to be one of the most efficient methods for performing secure two-party computation [10].

Garbled circuit protocols require expressing the computation as a binary circuit. Our system encodes events into binary strings and generates a combinatorial circuit based on the conditions of each resource-based rule. Circuits are created through the connection of sub-circuit blocks that depend on the type of constraints specified in the policy. Connections are performed through AND, OR, or NOT gates. We consider only constraints that require input from both events, as other constraints are validated locally.

The sub-circuit blocks in our implementation cover all temporal constraints in Fig. 2, in addition to *equality*, and *less-than* constraints. More circuits can be created for other types of constraints. For the resource-based rule in Eq. 3, the transformation maintains the equality constraint between the values of the variable I used in the event and the statement, and the *during* temporal constraint, as shown in Fig. 5 (left). Fig. 5 (right) also shows the encoding of the policy.

Our system uses a recent implementation of the garbled circuit protocol [10] to execute the circuit. The gc-client sends the ID of the rule to check, and interacts with the server to construct the garbled circuit. The gc-server uses an Oblivious Transfer (OT) protocol [15] to ask the gc-client to encrypt its input, and uses such data to execute the encrypted circuit locally. The encrypted output

of the circuit is sent to the client for decryption. If a match is found, the gc-client sends the matched event unencrypted. The gc-server adds the event in its local storage, which might trigger other two-server event correlation protocols.

The system executes the privacy-preserving matching protocol for each pair of events independently from the others. As garbled circuit computation requires several communication round trips for the exchange of data, parallelization can significantly improve throughput because of better utilization of the CPU.

In our interactions, the gc-server returns to the gc-client the number of events satisfying the local conditions to determine the number of times the privacy-preserving matching protocol is executed. If the number reveals information about the state of the infrastructure, the monitoring server can report any number larger than the given value, so that the number of events cannot be used to make any inference. Once the local events are exhausted, additional computations are performed with invalid values to ensure that no matching is possible.

3.5 Privacy Analysis and Limitations

From a privacy perspective, the security property of the two-event matching protocol shows that, for two-event policies, we share only events that create violations. This is the minimum level of information sharing that we can have between two organizations [20]. However, when the complexity of the policy increases and multiple resource-based rules are needed, sequences of events matching a single resource-based rule need to be shared to process the next resource-based rule. In such a situation, we limit information sharing by first selecting, when possible, resource-based rules that are rarely matched.

The interaction between monitoring servers leaks additional information that can be used to make inferences on the state of the remote party, even if no explicit sharing occurs. The request for a two-party correlation reveals the hash of the common resource and the policy involved. The hash is intentionally kept short, so that conflicts and false positives are possible, making the identification of the resource ID hard. The presence of an interaction, even if leads to no matched events, can still reveal that a subsequence of events matching a portion of the policy is present on the server. To counter such an inference, we add spurious requests with random events to ensure that such knowledge cannot be inferred.

The implementation of secure two-party computation used in our system relies on the assumption of an honest-but-curious attacker [9]. Such an attack model assumes that the two parties interact according to the protocol, and do not provide false information about their own systems. In the interaction between organizations, additional mechanisms can ensure that false information is not provided. The periodic auditing currently performed for ensuring compliance to regulations could also validate recoded logs of the interactions. Such logs create an audit trail that could dissuade organizations from providing false information. In addition, techniques have been proposed to validate the received information through independent information sources [21] to ensure its correctness. Moreover, progress has been made in building secure two-party computation that applies to semi-honest adversaries [11]. Such advance can be integrated to our solution.

4 Experimental Evaluation

Our evaluation measured the reduction in the amount of information exchanged when our event-correlation method was used, and the event rate obtained with our two-event matching protocol. We implemented the system in Java and used a garbled circuit protocol implementation by Huang et al. [10], with modifications performed to improve significantly parallelism. We ran our experiments on Amazon EC2 m1.large instances (7.5 Gb memory, 4 compute units), an instance type which computation capabilities fall in the middle of the EC2 spectrum.

4.1 Reduction of Event Sharing

We measured the reduction in the information shared between domains. Resources were partitioned across servers, and events were distributed randomly. Information sharing occurs when events about the same resource are stored in different domains. As the occurrence of such a condition is policy- and event-dependent, we evaluated our solution in different points in the space by changing three critical parameters. The first one is the frequency at which two events in different domains create a partial policy violation. The second parameter is the fraction of the infrastructure under the control of each organization. The third parameter is the number of security domains managing the infrastructure.

We measured the performance of our encrypted communication (`encr`) with rules of different complexity (`2-event rule, 4-event rule`). We compared it with a clear-text solution (`clear-txt`) that sends events related to a resource to the monitoring servers managing it (even if they are in a different domain) [19], and with the minimum need-to-know information (`min`).

First, we analyzed how the frequency with which events create partial policy violations affects the amount of information shared. We created events so that each pair of events in a policy has a given probability of referring to the same resource, and we randomly distributed them across domains. We show the results in Fig. 6. Our system significantly outperformed the baseline (i.e., `clear-txt`) solution and, for two-event polices, the fraction remained equal to the theoretical minimum (`min`). As we measure events shared over total events, the theoretical minimum number of events for the 4-event rules is smaller than the one for the 2-event rule: in the optimal case, a single interaction can summarize information about multiple events and it is counted as one event.

The distribution of resources across domains affects the fraction of events shared, as shown in Fig. 7. To test the system under less than ideal conditions, we created events that partially matched policies with a probability of 75% and we distributed them randomly to each server. The highest information sharing occurred when each organization had half of the resources, while the amount of event shared is reduced when more resources are managed by a single domain.

We measured the fraction of events stored at each server under different conditions (see Fig. 8). We considered both complete events and events that can be inferred from the presence of partial statements. Increasing the number of

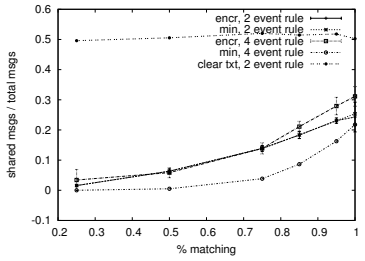

Fig. 6. Probability of matching events affects the fraction of events shared.

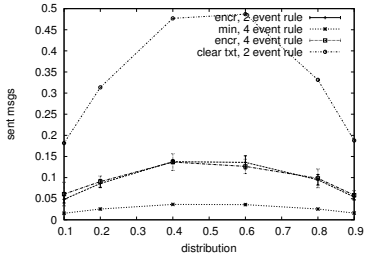

Fig. 7. Fraction of resources allocated to a monitoring server. 2 servers

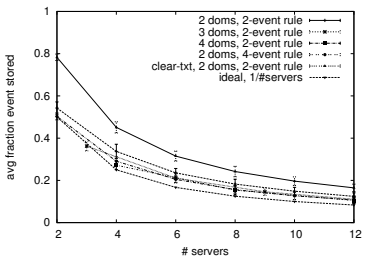

Fig. 8. Average fraction of events known to each server.

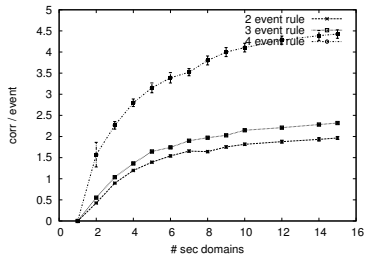

Fig. 9. Server load, multiple security domains. One server per domain.

security domains reduced the average number of events in each server. In all cases, our system provided a significant improvement over a clear-text solution.

To summarize, our experiment showed that while the performance of the system depends on the conditions of the policy and the frequency of matching events, our solution still outperforms a baseline solution. In many cases, the amount of information shared is close to the minimum possible. Best conditions occur when events in different domains creating a policy violations are not frequent, and when a significant fraction of interacting resources are stored within the same security domains, so that most violations can be found locally.

4.2 Performance Evaluation

To evaluate computation overhead, we measured the average number of secure two-party computations performed by each server, as shown in Fig. 9. We varied the number of domains. We considered the ratio between two-event correlations and events received. We saw that increasing the number of security domains increases the number of two-event correlations performed, as it increases the likelihood that interacting resources are managed by external servers.

We measured the ability of parallelizing and of distributing the computation by measuring the average number of computation per-server, as shown in Fig. 10. We injected a constant number of events into the system and we measured the ratio between the average number of two-event correlations performed on each

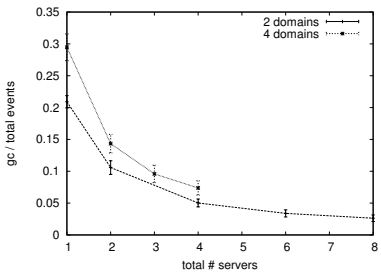

Fig. 10. Distribution of load with the increase of the number of servers within each domain.

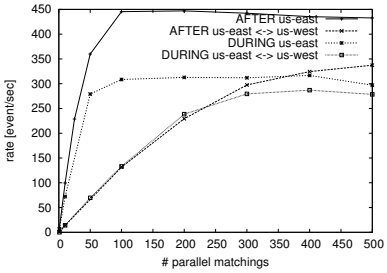

Fig. 11. Delay in the processing of an event as a function of the level of concurrency in the server.

server and the total number of events. When we increase the number of monitoring servers within each domain, the average number of per-server two-event correlations decreases as resources are distributed across the multiple servers.

To demonstrate the practical feasibility of our system, we measured the event rate achievable using our prototype implementation on an Amazon EC2 deployment. We used 64 bits for representing the resource name in binary form, and 32 bits for representing each event timestamp. Such values are sufficient to reduce collisions and to maintain low circuit complexity. Because the performance of garbled circuit protocols depends on the round-trip communication delays, we measured the performance between two servers within the same geographical region, and between servers in different regions. The former represents conditions found when monitoring servers are co-located within the same provider.

We measured the throughput in event correlation per second and we show the results in Fig. 11. The remote dataset was split into groups of 100 events, and the processing of each group occurred in parallel. When using multiple threads, we increased the rate up to 400 correlations per second on a single server.

We evaluated the effects of the policy constraints on the system's throughput. We evaluated two circuits: one checking for an equivalence between properties and for a constraint *after*; and a more complex circuit checking for an equivalence and a constraint *during*. The first one had an input size of 192 bits: 64 bits for each property name and 64 bits for two timestamps. The second circuit used 256 bits: 64 bits for each property name, and 128 bits for the four timestamps. The complex circuit reduced the throughput by about 30%.

We evaluated the effect of colocation of servers on event throughput. We considered servers co-located in the us-east region, and servers placed in us-east and us-west. Without concurrency, co-located servers for the *after* constraints obtained a rate of 9.5 event/s. When servers are located in different regions, the rate was 1.4 event/s. However, increasing the concurrency significantly increased the event rate. With 500 concurrent executions, co-located servers obtained a rate of 435 event/s, while servers in different regions obtained 337 event/s, with a reduction by about 22%. The *before* constraint had similar results.

In summary, our experiments demonstrated that our system is capable of performing hundreds of correlations per second on a single server, and that multiple servers can run in parallel to further improve the performance. This kind of event rate makes our system practical for monitoring system configuration changes and detecting complex attacks. It would scale to, for instance, taking a few seconds to validate the effects of a local configuration change on a remote infrastructure that consists of thousands of servers.

5 Conclusion

This paper introduced a distributed monitoring architecture for detecting violations of policies in multi-domain systems. The system uses secure two-party computation to reduce the amount of confidential information shared outside each security domain: information is shared only after verifying that it can potentially contribute to a violation. Our analysis and experimental evaluation show that the performance of our technique is adequate for configurations and other discrete operational state. Our approach is complementary to techniques that can process a larger amount of numerical data through aggregation and anonymization, such as network traffic information. We show that our technique limits the information shared to a minimal need-to-know for simple policies, and can significantly reduce the amount of information shared for complex policies.

Future work should introduce more optimizations in complex policies by changing the order of correlations so that the frequency of events and the willingness of the organization to share are taken into account. Additionally, reducing information stored in other domains can increase the security of the overall system, as security breaches in one of domains would provide little information to attackers about other systems. However, more work is needed to extended the system beyond the honest-but-curious attack model. For example, redundancy would provide mechanisms for recognizing compromised monitoring servers.

References

1. J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
2. D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow. Blueprint for the intercloud-protocols and formats for cloud computing interoperability. In *ICIW'09*, pages 328–336. IEEE, 2009.
3. M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. *USENIX Sec*, 2010.
4. S Ceri, G Gottlob, and L Tanca. What you always wanted to know about Datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166, 1989.
5. David Grawrock. *The Intel Safer Computing Initiative*, chapter 1–2, pages 3–31. Intel Press, 2006.

6. Grit Denker, Ashish Gehani, Minyoung Kim, and David Hanz. Policy-Based Data Downgrading: Toward a Semantic Framework and Automated Tools to Balance Need-to-Protect and Need-to-Share Policies. In *IEEE POLICY*, 2010.
7. David Evans and David Eyers. Efficient Policy Checking across Administrative Domains. *IEEE POLICY*, 2010.
8. C. Giblin, S. Müller, and B. Pfitzmann. From regulatory policies to event monitoring rules: Towards model-driven compliance automation. *IBM Research Zurich, Report RZ*, 3662, 2006.
9. O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, volume 2. Cambridge university press, 2004.
10. Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.
11. Y. Huang, J. Katz, and D. Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium on Security and Privacy*, 2012.
12. Jun Ho Huh and John Lyle. Trustworthy Log Reconciliation for Distributed Virtual Organisations. In *Trust '09*. Springer, 2009.
13. Jun Ho Huh and Andrew Martin. Towards a Trustable Virtual Organisation. In *IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 425–431. IEEE, August 2009.
14. P. Hunt, M. Konar, F.P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX ATC*, volume 10, 2010.
15. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. *Advances in Cryptology-CRYPTO 2003*, pages 145–161, 2003.
16. A.J. Lee, P. Tabriz, and N. Borisov. A privacy-preserving interdomain audit framework. In *WPES*. ACM, 2006.
17. Patrick Lincoln, P. Porras, and V. Shmatikov. Privacy-preserving sharing and correction of security alerts. In *USENIX Security Symposium*, 2004.
18. F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf. Nist cloud computing reference architecture. *NIST Special Publication*, 500:292, 2011.
19. M. Montanari and R.H. Campbell. Confidentiality of event data in policy-based monitoring. In *Dependable Systems and Networks (DSN), 2012*. IEEE, 2012.
20. M. Montanari, L.T. Cook, and R.H. Campbell. Multi-organization policy-based monitoring. In *IEEE POLICY, 2012*, 2012.
21. M. Montanari, J.H. Huh, D. Dagit, R.B. Bobba, and R.H. Campbell. Evidence of log integrity in policy-based security monitoring. In *Dependable Systems and Networks Workshops (DSN-W), 2012*. IEEE, 2012.
22. Christine M O'Keefe. Privacy and the use of health data - reducing disclosure risk. In *Health Informatics*, 2008.
23. Ruoming Pang. A high-level programming environment for packet trace anonymization and transformation. In *ACM SIGCOMM*, Germany, 2003.
24. Payment Card Industry (PCI) Security Standard Council. Data security standard version 1.1, 2006.
25. R. Ross, S. Katzke, A. Johnson, M. Swanson, G. Stoneburner, G. Rogers, and A. Lee. Recommended security controls for federal information systems (final public draft; nist sp 800-53), 2005.
26. Jatinder Singh, Luis Vargas, Jean Bacon, and Ken Moody. Policy-Based Information Sharing in Publish/Subscribe Middleware. *IEEE POLICY*, 2008.
27. Adam Slagell, Kiran Lakkaraju, and Katherine Luo. Flaim: A multi-level anonymization framework for computer and network logs. In *LISA*, 2006.