Abstract. Policies are used extensively in managing the security of large computer infrastructure systems. Many large organizations and several government entities such as the National Institute for Standards and Technology (NIST) and the North American Electric Reliability Corporation (NERC) define security policies to specify the allowed configurations of the systems under their watch. The goal of such policies is to help reduce the vulnerability of the infrastructure to attacks, misconfiguration and operator error. To that end, these policies specify allowed interconnections between systems, firewall configurations, software settings, and levels of redundancy in the system’s components. Ensuring compliance to such policies through frequent monitoring can reduce the time span during which these systems are vulnerable to attacks.

However, faults and attacks can make the underlying information used for validating compliance erroneous or incomplete. A compromised system could feed false information about its state to the compliance monitoring system. In this paper we introduce the concept of robustness of compliance. We show that systems which are compliant to security policies can exhibit different level of resilience to false information and we provide an algorithm for quantitatively computing a measure of robustness based on the concept of distance from violation. Intuitively, our algorithm computes an estimation of the amount of false information that needs to be provided to a compliance monitoring system for making an infrastructure appear compliant even when the underlying system is not compliant. Our experiments demonstrate that our approach is viable in large networks.

1 Introduction

Determining the compliance level of a system to a set of infrastructure policies is an important component in the overall assessment of the security of an organization. Infrastructure policies aim to prevent safety and security problems by specifying the correct operational conditions of systems within certain prescribed safety thresholds. Such policies are specified by organizations such as the National Institute of Standards and Technology (NIST) [10] to define the proper security configuration of enterprise systems, and by organizations managing critical infrastructure systems such as power grids or airport computer networks [6]
to define safe, secure and reliable operations of the critical infrastructure. The North American Electric Reliability Corporation (NERC) defines infrastructure security policies for the power grid [8] which mandate that all devices critical to the operation of the grid need to be placed within a hardened electronic security perimeter defined by firewalls. Non-compliance with these policies might expose vulnerable devices to remote exploitation. In airport systems, infrastructure policies might specify that an aircraft should be able to access airline servers for downloading software updates only when it is parked at the gate securely. Violation of this policy are potential indicators of erroneous configurations of the system which can reduce the safety margin of operations.

Generally, policy violations are detected during periodic auditing or through the use of management systems that monitor the system’s conditions. Policy monitoring systems [7] are designed to detect operation of a system in conditions that are not compliant with policies. Such conditions are detected by integrating incoming information from a variety of network monitoring devices and other monitoring software designed to observe specific characteristics of the system such as running programs, or the position of a device in a pervasive space. The integration of such information composes a view of the system’s state relevant to the policy compliance process. However, the state of the system observed by the monitoring process might not be consistent with the actual state of the infrastructure: errors in the detection of security-relevant events and malicious information provided by compromised devices can inject false state data into the compliance validation process and conceal that the infrastructure is actually operating in unsafe conditions.

In this paper we introduce techniques for assessing the robustness of an infrastructure to errors in the monitoring process. The evaluation of robustness permits the detection of situations where the infrastructure is potentially operating in unsafe conditions, even if it appears to operate within the tolerance limits defined by policies. When such situations are detected, we produce a report that specifies the exact conditions required to trigger the violation. Administrators can then verify these possible violations and take corrective actions.

We consider two types of errors in the monitoring process: non-malicious omissions or delay of events caused by availability problems in the devices that observe the system’s state, and malicious injection of false information about the system’s state from compromised devices. The analysis of the robustness to the omission or falsification of data is performed by computing sequences of events such as the installation of new software, the discovery of a new vulnerability, or the change of a firewall configuration that can result in a policy violation. We identify the critical sequences that, if not detected, would hide the fact that the infrastructure is operating in critical conditions. Using this information we can actively check the state of the system to verify that events did not occur, and we can preemptively configure the system to prevent such events from occurring in the future. The analysis of the resilience of the monitoring infrastructure to compromise of devices identifies a set of critical devices that if compromised, would allow an attacker to conceal the fact that the system is operating in unsafe
conditions. Administrators can use this information to develop a gameplan that acquires these critical pieces of information redundantly through independent channels to increase the system’s trust level.

Our approach is based on analyzing the state of the system to determine its distance from a state violating a policy. We express a policy as a set of conditions over the state, and we define the distance from a violation as the number of conditions in the policy which are not satisfied in the current state. We call this set the robustness set. By analyzing the robustness sets, we compute the minimal set of actions that would lead to a violation and the minimum number of devices that must be compromised to conceal a violation from detection.

The contributions of this paper include:

1. The definition of the concept of distance from violations as a way to measure the robustness of a system to errors in the monitoring process
2. Two techniques for computing the robustness of the state of the infrastructure to errors and to attacks directed toward the monitoring process.
3. An evaluation of the scalability of our techniques

The rest of the paper is structured as follows. Section 2 describes work related to the analysis of system robustness. Section 3 introduces our formalism for specifying the configuration of a system and for specifying policies. Section 4 introduces the concepts of robustness of compliance and the algorithms for computing critical events and critical devices. Section 5 provides an experimental evaluation of the performance of our algorithm. Finally, Section 6 concludes our work and discusses future directions.

2 Related Work

The use of infrastructure policies for the specification of the security of a system has been proposed by several authors. Fenz and Ekelhart [3] analyze how we can formalize policy requirements and mechanisms using an ontology to show the compliance of a system design. Anwar and Campbell [1] formalize policy requirements using logic and validate the system design for compliance. Previous work [7] introduced a scalable dynamic monitoring of security configurations to detect policy violations at runtime. This work extends this line of research by analyzing the consequences of errors in the monitoring process.

Security metrics have been presented as methods for analyzing the security state of an infrastructure system. For example, an attack tree analysis [9] provides an analysis of the security of a configuration by identifying possible attack vectors and their effects on the system. In our policy-based compliance analysis we express policies which can rely on the computation of logic attack trees. Additionally, this paper focuses on potential errors in the monitoring process. An algorithm for efficiently computing the security consequences of changes in the configurations of a systems have been proposed [11]. Such analysis is designed to compare different choices at design time and does not take into account errors in the monitoring process.
Probabilistic attack trees aim at providing a probabilistic estimation of the security of a network. Xie et al. [12] propose to analyze the security of a system by modeling the uncertainties about the difficulty in exploiting vulnerabilities and about the runtime behavior of the attacker using a Bayesian network. Our work considers general policy violations which are not restricted to security attacks as they can represent safety and reliability requirements. Additionally, we identify events and devices critical to the compliance of the system. This enables administrators to take preemptive actions and correct security problems before an attack occurs.

Recent work in the context of access control by Lee and Yu [4] provides a framework to reason about partial proof of access control authorizations. Among the feature of the framework, it provides the ability to identify users which are “almost” authorized to perform an action on a resource, even if their authorization is not complete. While the concept of “distance from violation” is similar, the access control framework is not directly applicable at policy compliance. Additionally, we provide algorithms for the computation of critical events and critical devices.

3 Policy Compliance Monitoring

Policy compliance is an essential security and regulatory requirement for infrastructure systems. Policies codify a set of rules that can help to prevent undesirable conditions which could lead to undesired consequences such as system compromises or reduced safety margins for critical systems. We focus on a set of policies that regulate valid security configurations and operational conditions of the computing infrastructure of large organizations and critical infrastructure systems. Such policies are defined by several government bodies and by many large organizations. For instance, NIST provides a set of policies which the computing infrastructure of government organizations must comply with [10]. These policies control the type of software that can run on machines, mandate the installation of anti-virus software or the use of VPN connections, and regulate the use of USB memory drives. NERC is another government entity which specifies policies about the configuration of power grid systems. For example, one NERC policy requires that all remote access point to the system must be monitored and logged at all times; and all critical systems need to be placed within an enclosed electronic security perimeter. Policies are not restricted to security configurations. Domain-specific policies can be used to specify valid conditions of specific types of systems. For example, a policy in the context of airport infrastructure systems could require aircrafts to access remote airline applications only when they are parked at the gate [6].

While policy compliance cannot guarantee security or detect all unsafe situations, compliance to a well-designed set of policies is considered a first step toward securing a system against attack that would be avoidable had proper security measures been taken, and can be used to detect erroneous operating conditions before the system enters an unsafe state.
Periodic auditing and monitoring systems [7] analyze the infrastructure and detect policy violations. Once violations are detected, manual or automatic attempts to correct the situation are undertaken. However, in several situations it is desirable to have an “early warning” when a system is operating in conditions close to a violation. For example, a system might be operating in conditions where a single action from any user could lead to a critical policy violation. Availability problems in the devices detecting such actions might make the monitoring system unaware that the system is operating outside policy specifications. Detecting such situations allows preemptive reconfiguration of the system to disable dangerous actions to reduce the risk of entering into a critical state.

Additionally, incorrect information provided by a single unprotected device can hide a dangerous operating condition from detection. By detecting such critical pieces of information, we can guide the process of information acquisition to increase the trustworthiness of information.

Our architecture is composed of a set of software agents or devices that provide observations about the state of the system relevant to policy validation to one (or more) monitoring servers, as shown in Figure 1. For example, some critical portions of the system’s state can include the list of running programs on each host, the list of active network connections between hosts, firewall rules, and application-level information such as the list of authorized users of web-application applications. Network management software such as WBEM or SNMP are able to acquire part of this information. Other pieces of information are acquired through network monitoring tools or specific software agents. The
definition of policies needs to be coupled with the definition of software sensors or devices that can observe the information useful for its validation.

Determination of the relevant policy compliance events to detect depends on the type of policies that the monitoring system is tracking. For security configuration monitoring, such events can be the installation of new programs by users or administrator, changes in firewall configurations, or the connection of new devices. All these events can alter the security stance of the system, and some of them might make the system violate the conditions specified by the policies. For example, a change in a firewall configuration that makes a vulnerable program accessible from the Internet would violate policies which specify that it should not be possible to compromise critical systems by exploiting known vulnerabilities.

3.1 Definition of Policies

We represent the state of the system as a set of facts expressed using Datalog. For example, we encode the statement that a machine $m_1$ is running a program $p$ that binds to port 80 using a Datalog ground predicate (fact) $\text{binds}(m_1, p, 80)$. These facts about the system are generated through observations of the actual state performed by entities called sensors. A sensor could be a hardware device or a software program. For example, the fact above can be generated by software running on the machine $m_1$, or by port scanning software running on a machine $m_2$ that port scanned $m_1$. For the purpose of policy compliance, we are not required to observe and represent the entire state of the system. We only need to observe the state relevant to the process of policy compliance. Events evolve the state of the system over time. The compliance monitoring system integrates events over time and reconstructs a representation of the ephemeral state.

Policies are expressed using Datalog rules. Datalog rules represent implications defined over the state of the infrastructure. The conditions of these implications are specified by a conjunction of predicates with variables. Predicates are matched (i.e., unified) with the facts representing the state, and if the condition is true a new fact is added to the state of the system. Upper-case characters indicate variables and lowercase characters represent entities in the system. For example, we can consider a simple rule which specifies that critical servers should not run applications with known vulnerabilities unless an exception is specified. By acquiring information about the running program on each machine, annotations about the importance of each server, and information about vulnerabilities, we represent this rule by specifying that we have a violation if a critical server provides a vulnerable service as following:

$$
\text{type}(S, \text{server}), \text{type}(A, \text{service}),
\text{provides}(S, A), \text{criticality}(S, \text{high}), \text{hasVuln}(A, V),
\neg \text{hasException}(S, E) \rightarrow \text{violation}(r1, V).
$$

The last predicate, called the consequence of the rule, is specified by a predicate which has variables that appear in the condition (body) of the rule. The
consequence can represent a violation as in our example, or it can represent a new fact which can be used by other rules for validating compliance. The compliance of a system to these policies is checked by considering the current state of the system and the inference rules. A policy violation is present in the system if a violation fact is present in the minimal model.

Policies are associated to a cost of violation. This value represents the expected impact on the infrastructure of a policy violation. For example, the cost of violating a policy that would allow the compromise of a substation in the power grid can be computed by estimating the cost of a potential blackout in the area powered by the substation [2].

4 Robustness of Compliance

Policies define the operating conditions within which the system should operate. Our robustness analysis considers these policies and the current state of the infrastructure to identify two pieces of information: the critical sequences of events which if not detected, would make the system operate in violations of policies, and the set of devices which, if compromised, would hide violations from detection.

The determination of these two sets of critical information is based upon computation of the distance of the state of the system from a violation. This distance is expressed as the set of predicates in the rule that would make the system violate a policy if satisfied. We express this distance through the definition of a robustness set. Based on the robustness set, we analyze the robustness to omitted events and robustness to compromised devices.

4.1 Robustness set

A basic step for both the determination of the set of actions which lead to violations and the set of critical devices is the evaluation of the distance from non-compliance of the infrastructure state.

In our framework, policies are represented as rules. The body of the rule expresses all the conditions that need to be verified for inferring that there is a violation of a particular policy in the system. When a system is compliant to a policy, one or more of such conditions do not occur in the system’s state. Intuitively, this set of conditions which do not occur constitutes an indication of the distance of the system from the violation of the policy. Given a set of facts matching a portion of the rule body, we define the robustness set as the set of predicates for which no match is found in the infrastructure configuration. Each rule has several robustness sets that depend on which facts and predicates in the body of the rule are considered. For example, a rule $p_1(X, Y), p_2(Y, Z) → violation$ with a system configuration $p_1(a_1, b_1), p_1(a_2, b_2)$ has two robustness sets: $p_2(b_1, Z)$ if we use the substitution $X/a_1, Y/b_1$, and $p_2(b_2, Z)$ if we use the substitution $X/a_2, Y/b_2$. A set of facts matching one of the robustness sets (e.g., $p_2(b_1, z_1)$) would make the entire body of the rule true.
and trigger a violation. In a more realistic example, consider a system whose state is described by $\text{computer}(c_1), \text{computer}(c_2), \text{run}(c_1, p_1)$ and $\text{run}(c_1, p_2)$ with a policy $\text{computer}(X), \text{run}(X, P), \text{hasVulnerability}(P, V) \rightarrow \text{violation}$ resolves to four robustness sets: $\{\text{hasVulnerability}(p_1, V)\}, \{\text{hasVulnerability}(p_2, V)\}, \{\text{run}(c_1, P), \text{hasVulnerability}(P, V)\}, \{\text{run}(c_2, P), \text{hasVulnerability}(P, V)\}$. Such robustness set represents the cases in which a new vulnerability is discovered in $p_1$ or $p_2$, or in which a new vulnerable program is run on $c_1$ or $c_2$.

We associate robustness sets to each policy by considering all substitutions that match at least one predicate of the rule body with a fact in the state. The minimal robustness set is the set with the least amount of free variables.

Robustness sets are associated with a policy by unifying different portions of the body of the rule with facts in the system state. We consider each predicate in the body of the rule and we find all facts in the knowledge base that unify with such a predicate. For all matching facts, we substitute the variables in the rule body with the values in the fact. The partially matched rule is added to the list of robustness sets and to a candidate list of rules. We continue the analysis by removing one of the partially matched candidate sets from the candidate list and considering all its predicates with facts in the system. The algorithm for associating robustness sets to each policy is shown in Figure 2. Our techniques do not require to generate all robustness sets. Optimizations are used to reduce the sets to consider.

4.2 Robustness to event omissions

The detection of events that affect the security state of the system might not be complete. Faults can temporarily disconnect monitoring devices from the network, and software or configuration errors can make the system miss important events. For this reason, the representation of the system’s state collected by the
monitoring system might not be consistent with the actual state of the system: even when it appears that the system is operating in safe (i.e., policy-compliant) conditions, the actual infrastructure might in fact be in violation of the policy because a single critical event has been missed. Our technique takes the state collected at the monitoring server and finds such critical sequences of events. Using this knowledge, administrators can verify that such events did not occur in the past, and they can reconfigure the system so that such events cannot occur in the future.

**Actions** The determination of the critical sequences of events is performed by modeling events relevant to policy compliance and by analyzing the robustness sets of each policy. From the robustness set, we identify the sequences that lead to violations. Events that change the configuration of the system are represented as actions. Actions are hypothetical modifications of the state of the system and we represent them by adding or removing facts from the overall state. For example, to represent the action of running a new program \( p_1 \) on machine \( m_1 \), we add a fact \( \text{run}(m_1, p_1) \) to the state of the system. We call such facts added or removed temporarily to the state the consequences of the action. Often, actions can be applied only if certain conditions are verified in the system. For example, we can run a new program only if the machine \( m_1 \) is active. We represent this fact using preconditions. A precondition is a set of conditions about the state that needs to be satisfied for the action to be possible.

We represent actions as a list of preconditions and a list of consequences. The preconditions are separated by the consequences with the operator \( \Rightarrow \). The action of adding a new fact is represented by using by prefixing \( + \) before the predicate, while the removal of a fact is represented by prefixing \( - \) before the predicate. For example, the action of running a new program is represented as \( \text{computer}(X), \text{program}(Y) \Rightarrow +\text{run}(X, Y) \).

Similarly, we can express the fact that an unknown vulnerability is discovered in an installed program using an action as follows:

\[
\text{computer}(X), \text{software}(SW), \text{hasInstalled}(X, SW), \\
vulnerability(V) \Rightarrow +\text{hasVul}(SW, V),
\]

(2)

Actions represent the security events that might be not detected by the monitoring system. The missed detection of events can occur for several reasons. For example, malfunctioning or maintenance on a network monitoring device might cause network packets to be not analyzed for a short period of time. Events occurring during the downtime period might have not been detected.

Events happen with different frequencies in the system. Running a new vulnerable program is likely to happen on a user machine, but less likely to happen on a controlled server. A short downtime of a sensor has a larger chance to miss a frequent event rather then a very rare event. Because many different sequences of actions can lead to violations, we are interested in finding only the most probable sequences. We represent the likelihood of an event to occur by associating
actions with a score $s$. To simplify the composition of the score of different actions, we assume that they are independent. We define the robustness to changes of a policy given a configuration as the composition of the score of the actions that can lead to a violation of the policy. For example, the actions that represent the discovery of a previously unknown vulnerability can be associated with a score $s$ that represents an estimation that a new vulnerability is discovered in a program within a specified time interval. Scores for each action are computed by observing the frequency with which events occur in the system.

**Analysis** Once the possible actions are specified, the robustness to events of a policy is computed by finding the sequences of actions that lead to a violation. We call these sequences *critical action sequences*. A naive computation of these sequences can be performed by applying all possible sequences of actions to the state and identifying the ones that lead to a violation of the policy. However, even when we limit the length of sequences by removing the ones whose score is below a predefined threshold, the unguided search remains computationally intractable as the size of the network increases.

We use the robustness set to guide the search for the critical action sequences. The robustness sets provide the minimal sets of predicates that should be removed or added to the system for a policy to be violated. We reduce the search space by considering only actions that affect the robustness set of the policy, and we reduce the length of the sequences of actions by ignoring sequences whose probability goes below a predetermined threshold.

The sequence of critical events for the policy is found by analyzing the robustness sets in order of increasing size (i.e., we consider first the robustness sets with a small amount of free variables). We consider the actions whose consequences match with one of the predicates in the robustness set and we call this set $A'$. We search within the set $A'$ for all sequences of actions that lead to a violation, but we stop if the combined score of the actions goes below a set threshold. The threshold represents the maximum number of events that we assume can be lost at any point in time. If a sequence is found, we add it to the list of critical sequences. If sequence over the threshold is found, the robustness set is added to a set $D$ of strong robustness sets. A critical sequence for the set must also contain a critical sequence for the strong robustness set. Since this critical sequence could not be found in this case, any set that contains a strong robustness set has no critical sequence. This observation allows skipping the analysis of a large number of robustness sets.

By combining the scores of the critical sequences we obtain a measure of the robustness of each policy to critical events. The integration of this score with the cost of violation provides an estimation of risk. Additionally, the critical sequences can be used to reconfigure the system and prevent such events from happening, or for checking if these events have already happened.

For example, we can consider a policy in an airport infrastructure system stating that a network device should not be used near specific restricted areas around the aircrafts during refueling. When the aircraft is not refueling and no
devices are located in the restricted areas, the system needs two events for entering in a policy-violating state (i.e., moving a network device in the restricted area and starting refueling operation). When a network device enters the restricted area, then only one event is sufficient for the policy to be violated (i.e., starting the refueling operations). The computation of the robustness to event omission would detect this condition and notify in advance the refueling process that it cannot start until the network devices are out of the restricted areas.

4.3 Robustness to compromised devices

The second component of the robustness of a configuration measures its resilience to the compromise of devices. An infrastructure system operating outside the conditions specified by the policies is potentially operating in unsafe conditions. Malware can be used on compromised devices to provide false information to the monitoring system and hide the presence of a violation. In this condition, unless the information about the state of the system is obtained independently through other sensors, the monitoring system would be unable to detect a violation.

Our technique computes a minimal set of devices that is necessary to compromise in order to hide violations from detection. We rely on the robustness sets to identify how close the system is to a violation and whether information required to show the presence of a violation is provided by a single device. Such devices are called critical devices because, if compromised they would enable an attacker to conceal a violation of the policy. For assuring that the system is operating within the policies, the information collected by critical devices should be verified independently through other sensors, and the security state of the critical devices should be verified to ensure that the sensor has not been compromised.

The identification of critical devices is performed by considering the data that are potentially generated by each device. We compare this information with the robustness sets of each policy. If the entire robustness set can be generated by a single device, the device must be critical.

Sensors and Facts

Sensors observe the state of the system and represent it as Datalog facts. We can associate each ground predicate \( \text{pred}(a_1, \ldots, a_n) \) with the sensor sources \( s \) producing it. We indicate this association with an additional argument \( s \), as in \( \text{pred}(a_1, \ldots, a_n, s) \).

Each sensor generates a specific set of facts about the system. A network monitoring device provides information about network flows and communication patterns. A software agent running on a machine provides information about running programs and users. Given a device, we represent the set of predicates that it can potentially generate in a source base. A source base is a set of ground and non-ground predicates that unify with all set of facts generated by the sensor. For example, a software sensor \( s \) that observes the software running on a machine \( m_1 \) generates a fact \( \text{run}(m_1, p_i, s) \) for each program \( p_i \) running on \( m_1 \). The source base \( G_s \) associated with the sensor contains the predicate \( G_s = \{ \text{run}(m_1, P, s) \} \). In another example, if a similar sensor \( s' \) collects information from two different machines, the source base is \( G_s = \{ \text{run}(m_1, P, s'), \text{run}(m_2, P, s') \} \).
Devices can provide redundant information about the system. For example, in an airport system, the location of an aircraft is provided by GPS devices located on the aircraft itself and by the ground radar of the airport. We indicate such redundancy by saying that two sensors are equivalent.

**Analysis**  The analysis of robustness determines if compromising one sensor is sufficient for hiding a violation. A device is critical for a robustness set if it is possible to unify the set with the source base, i.e., if all predicates in the robustness set can be matched with predicates in the source base.

Generally, we can assume that monitoring systems are aware of sensors that are currently employed in the system. For performing the analysis, we take each robustness set (in order of increasing size) and we check if it is possible to match it with each of the source bases. If the match is possible, then the device associated with the source base is critical. The analysis provides a list of the critical devices associated with each policy. Such critical devices can be analyzed to assure that they are not compromised or under the control of malicious users.

For example, we consider an airport scenario and a policy specifying that aircrafts must be at the gate for updating their software systems. The sensors providing information about the location of the aircraft are the aircraft GPS and the control tower, while information about the fact that the aircraft is updating software is provided by the airline servers. In this case, when the aircraft is at the gate we obtain a robustness set including statements requiring the state of the software update. Hence, the airline servers are critical devices in these conditions. On the other side, when the aircraft is updating, the aircraft and the control tower are not critical devices: a compromise of one of the two devices would be detected by checking the inconsistency of the location information.

Different policies can be associated with different costs of violations. We can rank devices according to the importance of the policies by considering the sum of the costs of violations of all the policies for which the devices are critical.

## 5 Experimental Evaluation

In the experimental evaluation of our technique, we measure its applicability in realistic scenarios by creating simulated configurations of network systems and evaluating the time required for the analysis. We implemented the system in Java 1.5 using Jena [5]. We ran the experiments on 10 networks of 50 to 500 nodes and the corresponding results are the average timings over 10 execution runs on each of the networks on a 2.8 GHz Quad-Core Intel Xeon system.

Our first experiment measures the scalability of the algorithm for computing robustness to event omission. For our experiments we evaluated the robustness of an enterprise network. The complete network architecture is composed of a variable number of hosts organized in four networks: the public Internet, a DMZ area, the enterprise network, and a SCADA network. Interconnections between networks are protected by firewalls that only allow traffic to specific services (e.g. http or ssh). Each host runs services that are might have software vulnerabilities.
Our policies specify that there is a violation if it is possible to compromise a host from the Internet by exploiting a vulnerability.

We classify configurations of the system into three classes: violation, robust to one event, robust to two or more events. We compare our algorithm for the computation of the critical events with a baseline obtained by applying each possible event and evaluating the presence of violations. We consider three types of events: discovery of a new software vulnerability in an installed software packages (e.g., a zero-day attack), change in the connection between devices to networks (e.g., a device under the control of the attacker connected to different networks), and running a new software on a device. Our algorithm outperforms the baseline for large networks: we are able to identify critical events within a few seconds even in networks composed of hundred of devices. These results are shown in Figure 3.

The second experiment evaluates the computation of robustness to device compromises. While this algorithm is similar to the computation of event omissions, we cannot easily discard robustness sets which are not relevant for the analysis in the event-omission algorithm. This fact increases the time required for the analysis. In these conditions, a near real-time computation of the critical devices can be performed only on networks of about 100 hosts. This computation takes an average of 2.87 seconds. For larger networks, the computation needs to be performed offline. For a 300-host network, the computation of the critical devices requires an average of 111 seconds.

6 Conclusion

Compliance-monitoring system cannot always provide a complete view of the security of the system. Undetected events or compromises of devices can hide from monitoring the fact that a system is operating in conditions which are not compliant with policies and, hence, potentially unsafe.
This work introduced techniques for analyzing the robustness of the security configuration of a system to undetected events and compromised devices. We identify and quantify critical sequences of events that need to be detected to avoid operating outside compliance, and we identify critical information that needs to be acquired from devices for preventing attackers from hiding unsafe conditions of the infrastructure. The knowledge of such events and such devices can be used for evaluating risks and for reconfiguring the system to avoid entering such unsafe states. We showed that our approach can be applied to enterprise network systems and airport infrastructure systems.

Future work will introduce several optimizations in the computation of device-compromise robustness. We are going to use the source base of each sensor to guide the generation of only robustness sets relevant for the determination of the critical devices. Additionally, we are planning to integrate these techniques in a policy-monitoring system for airport infrastructure. Using the monitoring system, we will evaluate these techniques in a real scenario. Finally, future work is going to investigate the use of the critical event sequences for planning preemptive protective actions, and the use of the list of critical devices for performing online analyses of the security of such devices.

References