Automatic Verification of Distributed and Layered Security Policy Implementations

Sankalp Singh, William H. Sanders, David M. Nicol, and Mouna Seri

Information Trust Institute
University of Illinois at Urbana-Champaign
Urbana, IL, USA
{sankalp,whs,dmnicol,seri}@uiuc.edu

Abstract. Access control has long been the linchpin of intrusion prevention. Modern networked systems that are intended to be secure have a global policy, usually implicit, that specifies the overall system-level objectives with respect to access to various resources. The policy indicates both what is inadmissible, so that the intrusion attempts from within and without the network may be prevented, and what accesses must be allowed, so that the essential functionality of the system is not compromised. This policy is implemented through the configuration of myriad local devices and mechanisms, including router-based and host-based firewalls and discretionary or mandatory OS-based access control mechanisms (e.g., SELinux). The complex interactions among these distributed and layered mechanisms can mask problems and lead to subtle errors. In this paper, we introduce a framework for performing a comprehensive security analysis of an automatically obtained snapshot of the access control policy implementation to check for compliance against a (potentially partial) specification of the global access policy. The framework has been implemented as the Access Policy Tool. APT helps to increase confidence in the efficacy of the intrusion prevention mechanisms in place by allowing for reasoning about the security policy at a high level of abstraction. We describe our analysis techniques and demonstrate their efficiency, scalability, and extensibility by using APT for a variety of test cases.

Keywords: Security Assessment, Access Control, Policy, Firewalls, SCADA and Process Control Systems

1 Introduction

Networked systems are used in a large number of settings, including several critical infrastructure systems, such as chemical plants; electric power generation, transmission, and distribution facilities; water distribution networks; and waste water treatment facilities. The emerging scenarios and the likely trends for the future of critical networked systems demand that the problem of securing these systems receive immediate attention, especially in the area of controlling access to the critical elements of the system over communication networks. Given the mission-critical nature of a significant number of large networked information systems, it is extremely important to ensure their protection against cyber-attacks, which, in a worst-case scenario, could result in loss of life, or in massive financial losses through loss of data, actual physical destruction, misuse, or theft.

A modern networked system includes a variety of devices and mechanisms to control access to its resources. These access control mechanisms include, but are not limited to, router-based dedicated firewalls; host-based firewalls, which could be based in software or hardware; operating-system-based mechanisms, such as the mandatory access control in NSA's SELinux; and middleware-based mechanisms, such as the Java Security Manager, that provide for specification and enforcement of fine-granularity access control policies for Java programs.

The importance of correctly implementing access control for effective intrusion prevention cannot be overestimated. A survey of the SANS top 20 vulnerabilities [1] shows that a significant number of them are defended against by appropriate configurations of access control policy. To defend systems against the most critical known threats, one has to be able to validate security policy implementation. However, distributed and layered mechanisms, such as those listed above, can interact in complex ways that can lead to subtle errors and mask problems. It can be difficult to discern the global picture that emerges from the local configurations of these myriad access control elements. As a result, it is not surprising that misconfigurations of these mechanisms are a major source of security vulnerabilities. In fact, a recent study suggests that most firewalls (the most popular access control mechanism) suffer from misconfigurations [2]. It is important for the administrators of computer networks to have ways to make sure that high-level specifications of such system access constraints are reflected in the actual configurations of the access control mechanisms spread throughout the system. Furthermore, if the implementation of policy (device configurations) is not in compliance with the specification, a diagnosis to locate the root causes of the problem is critical.

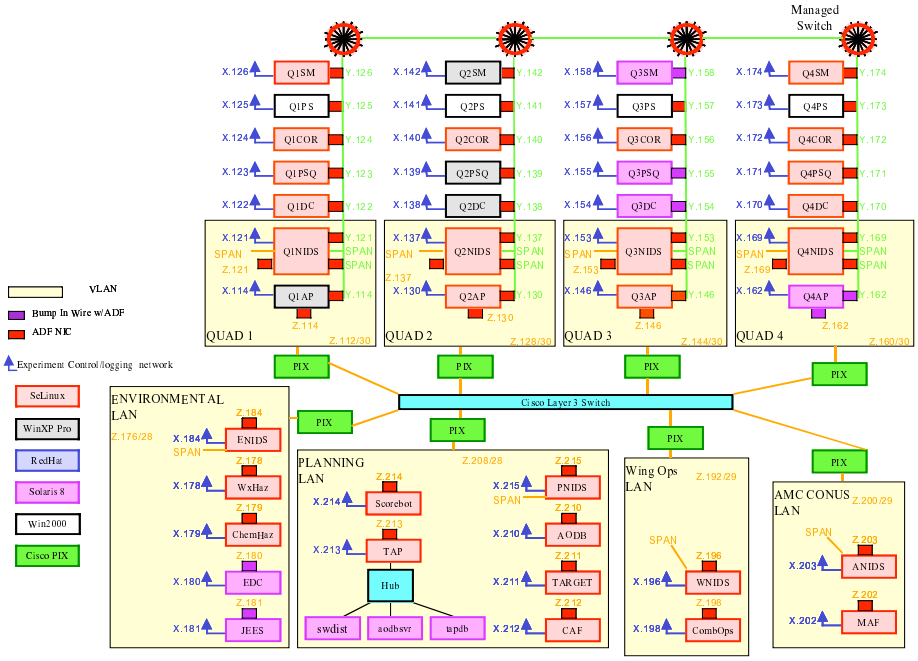

Fig. 1. Representative Network

This paper describes the Access Policy Tool (APT), a tool that we have developed in answer to the needs described above. APT analyzes the security policy *implementation* for conformance with the global security policy *specification*. It integrates policy rules (i.e., configuration information) from a large variety of sources typically found in a modern network and provides for a detailed offline analysis, as well as dynamic online analysis of incremental configuration changes for detection of policy implementation holes during operation. It ensures scalability with increasing network size and complexity via a statistical analysis mode. For increased usability and ease of information management, the tool includes a graphical front-end.

2 Access Control in Networks

Access control has long been the linchpin of system security; modern systems have multiple access control methodologies, different security models, and separate configurations for each methodology and each device. Together these all form the access control *implementation*. However, only very limited technology exists to answer crucial questions about precisely what security posture is

produced, how the different access control policies interact, and whether the implementation is in compliance with an overall statement of global access control policy, among others.

To better appreciate the issue, let's examine some of the access control components of a distributed system. Firewalls are critical assets in the protection of a network. A firewall is configured through its rules (collectively referred to as a *rule-set*), which it can use to mold and shape the traffic that crosses it. A firewall matches the incident network traffic against its rules, using traffic characteristics such as the source of origin, intended destination, and communication protocol, and either forbids or allows the traffic to pass through depending on the action indicated by the matching rule. A typical setting usually contains a distributed firewall implementation, wherein traffic may need to pass through more than one firewall to transit the network. Firewalls can be used to divide the network into secure zones that limit user and application access between zones. For example, Figure 1 shows a real-life network we have studied (developed as a part of a large DARPA project), composed of multiple network zones (in boxes) isolated by devices that enforce access control policy. Of particular interest is the fact that there are eight separate zones, and over 50 different policy enforcement devices (including SELinux on some of the hosts).

A system can also have host-based firewalls, implemented in either software or hardware. Software-based firewalls, such as iptables [3] in Linux and several commercial ones available for Windows, are implemented in the host operating system. Hardware-based firewalls, such as 3Com's Embedded Firewall (EFW) PCI card [4], are implemented on the network interface card (NIC) itself, and as a result, those firewalls are tamper-resistant to cyber-attackers who might gain control of the operating system on a host.

There are published guidelines (e.g., the National Infrastructure Security Coordination Center (NISCC) guide to good practices in firewall deployment [5]) to facilitate the development of unique rule-sets for the different firewalls at an operating site, but they are fairly generic and have to be customized by network system administrators to the specific needs of their sites.

However, access control is more complex than just firewall rules. In security-conscious settings, hardened versions of traditional operating systems have been adopted. These include SELinux [6] (developed by the NSA), which is a secure version of the Linux operating system. Similar functionality for Windows operating systems can be provided through third-party software, such as the Cisco Security Agent [7]. Such software can provide mandatory mechanisms such as role-based access control, type checking, and multi-level security models.

One can use a global policy to specify at a system level the overall objectives with respect to access control. It specifies both what connectivity between roles and devices is inadmissible, and what connectivity must be supported. The rules are stated in terms of sets of roles and devices, rather than individual ones. For example, "An account manager in the sales network zone must be able to use sftp to forward any file in the Monthly Sales directory to his Reports directory, on the sales server found in the management network zone." Clearly, multiple access control policies are at play here; it is also clear that the rules can be stated in a form that allows a program analyzing access control configurations to determine whether the rule is satisfied.

The critical problem this work addresses is that networked systems only check fine-grained local access policy rules at single devices, not global policy. Global access policy implementations may or may not comply with global policy requirements. Without a way to check that compliance, serious security vulnerabilities can and do exist in real implementations of critical systems, causing them to fail in potentially harmful ways when attacked.

3 Related Work

The complexity of implementing an access control policy through configuration of a large number of distributed devices and the risk of conflicts among these devices have spawned a significant amount of work. However, the majority of the work focuses on internal consistency among rules of a single firewall [8–13]. A conflict detection tool for distributed firewall systems has been described in [14], but it only checks for certain kinds of syntactic errors, such as overlapping rules, and not for semantic errors with respect to a specification of the intent. Furthermore, none of the work takes into account the collective fine-grained access control provided by the network-based mechanisms (e.g., firewalls) and host-based mechanisms (e.g., SELinux policies). Also, none of the above work has been applied to and optimized for specific classes of networks.

Another vector of research effort focuses on automatic generation of consistent policy implementation (at devices) from formal description of global policy [15, 16]. Cisco provides CiscoWorks [17], a suite of products that enable top-down firewall policy management that includes configuration management, a policy manager that generates policies from high-level specifications, and log/audit analysis. Those are appropriate for new, vendor-homogeneous systems, but do not address existing implementations or heterogeneous systems. Furthermore, they require specification of detailed and exhaustive global security policy. For

most current network system administrators, the ability to check existing configurations against policy specifications that indicate intended high-level behavior would likely be more useful.

Vendor-neutral tools have recently appeared that consider access control in a distributed system, including the Skybox firewall compliance auditor [18] and Red Seal SRM [19]. Both tools use network topology and routing information to determine potential network flows. Red Seal SRM uses vulnerability databases and specification of software running on hosts to compute risk measures, while Skybox determines when firewall rules allow violation of a more abstractly stated global policy. In comparison, our approach incorporates sophisticated statistical analysis for highly improved scalability and also naturally admits integration of higher-level layers of access control policy (e.g., SELinux policies, and/or role-based access in trusted networks).

The problem of checking implementation against specification is also known as "model checking" [20]. This line of research has a rich history in the context of proving the correctness of both hardware and software. The key abstraction is a finite-state machine, and the key problem is that the size of the state-space explodes with the complexity of the system. As we consider integration of higher layers of access control mechanisms, the finite state machine view may be appropriate for them. However, while it is possible to map *network* access control into a classical model-checking framework, we aren't convinced of the value of doing so. Much of the attention in model-checking is paid to compact representation of the state-space; we already have a representation of the problem (i.e., the rule-graph, to be described), which is a graph whose number of nodes is linear in the number of rules. Furthermore, our analysis takes advantage of the problem domain rather than a general state-space, with an approach that lends itself naturally to optimizations for scalability. The research on "attack graphs," largely an application of model checking, is of some relevance when considering the scalability issue, particularly when compared to the statistical analysis that we have developed. Ritchey and Ammann [21] use model checking to identify a *single* violating path in an attack graph. Sheyner et al. [22] provide a more comprehensive analysis, but suffer from the state-space explosion problem, since the entire attack graph needs to be analyzed to provide the relevant metrics, thereby severely limiting the scalability. Ou et al. [23, 24] have used a prolog-based approach, rather than the traditional model checking, for attack graph generation and analysis; however, their solution does not seem to be able to scale to large networks that are also deep (i.e., graphs with potentially long

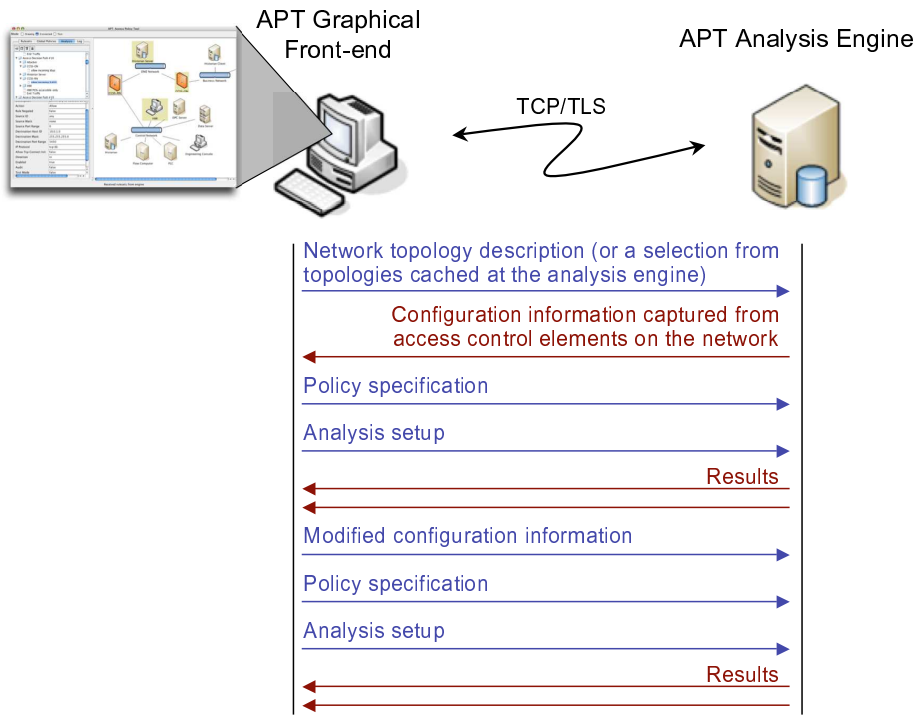

Fig. 2. APT Architecture

paths), or to calculate generic metrics that are functions of paths rather than edges in the graph.

4 APT Architecture

As shown in Figure 2, APT has two independent components:

- a graphical front-end (or management console), written in Java Swing, that system administrators can run from their workstations and use to provide information about the basic network topology, to enter specifications of the global access policy (or subsets thereof), and to set up analyses; and
- an analysis engine, written in C++, which captures the system state and analyzes that information with respect to policy specifications.

As indicated in Figure 2, the two components of the tool communicate securely over the network using TCP/TLS protocols. This segregation of functionality makes it possible to use the analysis engine as an appliance that can be

plugged into a suitable spot in the network, from which it can establish secure connections to the access control elements present on the network and capture the relevant configuration information. The actual interface between the tool and the user (the front-end) can then be freely placed where it is convenient for the user.

As shown in the figure, the user first provides a description of the network topology using the graphical front-end. That can be done using the easy-to-use drawing tools included with the front-end, or via text files that can be read by the front-end. The information to provide includes all the relevant access control elements (firewalls, proxy servers, wireless access points); details for each element, such as its IP address(es) and parameters that the analysis engine can use to establish secure connections to the element to capture its configuration; and all the network interactions between the listed elements. Entities can be grouped together to indicate sub-networks and LANs, and properties can be specified for such groupings. The front-end converts the visual topology representation provided by the user to an internal XML representation. We use XML as our underlying language for internal representation for most of the information in the tool, which provides the added benefit that we may interface third-party tools and applications simply by providing wrappers that output the XML conforming to our schemas. It is also possible to browse and select from the list of topology descriptions previously cached remotely at the analysis engine. In either case, once the topology is decided, the information is sent over the network to the analysis engine. The analysis engine uses that information to securely obtain the snapshot of the access control policy implementation, in the form of configuration files from the various devices indicated in the topology description (described in more detail in Section 5). It also indexes and caches the topology description and captured configuration information (both of which are encrypted before storage for added security). The captured information from all the different sources is then converted to XML (a unified schema for firewall rules and another for OS-based mechanisms), and the resultant XML is sent back to the front-end, where the user can then highlight various elements in the topology diagram and look at their current configurations (e.g., rule-sets for a firewall).

The user then specifies the global access constraints, which are a subset of the possible comprehensive access policy. They indicate some intended behavior against which the user wishes to check the policy implementation for compliance. The user uses a graphical front-end to specify constraints for the various elements or groupings of elements, and the tool considers their conjunction. The individual

constraints can express positive as well as negative assertions about the nature of traffic, roles, user-classes, or applications that can access a particular set of resources (hosts, specific files or applications, and so forth). In other words, we can express things that should never happen as well as things that must be allowed. Once specified, the constraints are converted into an internal XML representation. This further allows the freedom to use third-party technologies, such as a variety of modal logics, to specify the global access constraints.

The next step for the user is to set up the analysis. The user can select either an exhaustive analysis or a statistical analysis, following which the front-end sends the policy specification and setup information for the analysis to the analysis engine. The analysis engine sends back the results as it obtains them, giving the user the option to abort the analysis at any stage and do post-analysis on the partial results. The user can manipulate, filter, or navigate through the results, if any violations are found, to visualize the problem and diagnose the key misconfigurations behind the violations. A variety of post-analysis techniques are available in the front-end to help the user identify the likely root causes of the violations. Once the user has some possibilities in mind, hypothetical changes can be analyzed using the front-end; the user can make proposed changes to the configuration of various elements (e.g., modify, delete, or reorder certain rules), and the new information will be sent to the analysis engine, which then performs a quick re-analysis to see if the (hypothetically) modified configuration information now conforms to the specification of the access policy. This feature can also be used to check planned changes in configuration for compliance quickly before they are actually rolled out.

5 Analysis Engine Internals

Figure 3 gives an operational overview of the APT analysis engine. The front-end supplies the analysis engine with information about the network topology and the parameters for establishing secure network connections with the access control elements of the topology. Examples of such parameters include keys for establishing VPN connections to Cisco PIX firewalls and a guest username/password for establishing SSH connections to Linux/SELinux hosts. Using the supplied information, the analysis engine can establish connections and obtain the relevant configuration information, hence capturing a snapshot of the access policy implementation. The configuration information can take many forms depending on the mechanisms and devices being used to enforce access control. It may consist of custom rule descriptions for different kinds of firewalls, SELinux type

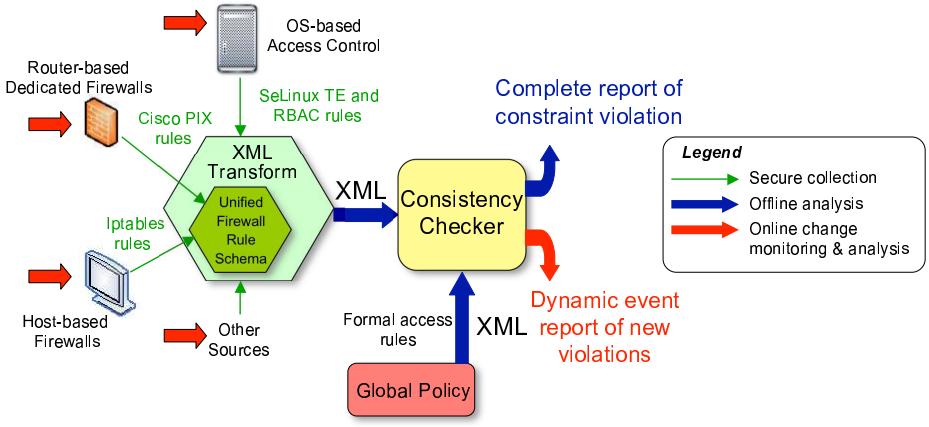

Fig. 3. Operational Overview of the Analysis Engine

and role transition policies, or Java Security Manager policies. APT integrates policy rules from a large variety of such sources. We have developed a unified XML schema that captures the essence of the union of all the classes of access control mechanisms that one is likely to encounter in a modern IT network. The analysis engine includes modules that convert the policy rules (configuration information) from the different sources, with rules from each source in their own custom language, into XML conforming to our unified schema. That allows for easy extensibility as support for new access control mechanisms and devices can be added by simply writing the translation modules for those devices.

In APT's offline analysis mode, it is not necessary to tightly integrate the analysis engine with the system being analyzed, and if needed, the analysis engine can be directed to read in the configuration snapshot that has been collected offline a priori and placed locally on the file system. In the online analysis mode, the analysis engine either periodically checks with access control devices for any changes in configuration, or is specifically directed towards those changes by the user via the front-end.

5.1 Consistency Checker

The consistency checker forms the core of the analysis engine. It takes as input the XML representations of the collective configuration information from the various devices and of the formal specification of the access constraints provided by the front-end, and sets up an analysis based on the parameters supplied by the user (again, using the front-end). As described in Section 4, the output, in the form of the list of violations, is sent back to the front-end. In the online

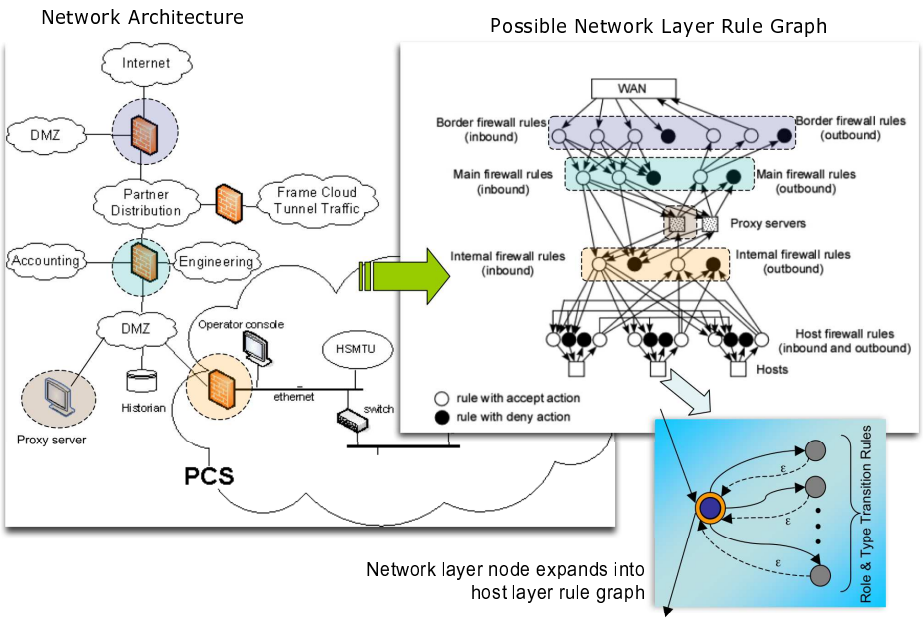

Fig. 4. Constructing Rule Graphs

mode, if an intrusion detection and alert correlation system has been deployed on the network, the results may also be forwarded to it in the form of an alert.

The network topology and the configuration information (policy rules) from the access control devices are internally represented by the consistency checker using a specially constructed data structure called a *multi-layered rule graph*. Figure 4 shows a simplified representation of a possible two-layered rule graph for the network shown on the left side of the figure. The rule graph captures the network interconnectivity and data flow among the policy enforcement rules. In the top-level, or the *network-layer*, rule graph, there are some nodes that correspond to devices that accept traffic and other nodes that correspond to rules in devices through which traffic passes. A firewall (dedicated or host-based) is represented by as many nodes as it has rules, with both inbound and outbound rules being presented; a host (terminal or proxy) is represented by just one node. A node representing rule r_i in firewall F_m directs an arc to a node representing rule r_j in firewall F_n if the action (access control decision) associated with the rule r_i is to accept, and a network packet that is thus accepted can be received (in accordance with the network topology) by firewall F_n. A node representing a proxy host (that allows for traffic to pass through it) has both incident and outgoing arcs.

Nodes representing hosts may further expand into a lower-level rule graph; Figure 4 shows a potential *host-layer rule graph* (modeled as a directed graph with labeled edges) representing the SELinux policy rules for one of the hosts in the network-layer rule graph. The arcs in this rule graph change the security contexts and/or the permissions of various objects present on the host.

The overall strategy used by the consistency checker is to analyze paths from traffic sources to terminal nodes in the rule graph. A terminal node may be a rule associated with a "deny" action, a destination, or an exit from the system. A path describes a possible sequence of access policy decisions applied to all traffic that traverses the corresponding devices in the network. The analysis approach is built around the observation that knowing which rules in the system make decisions about a piece of traffic tells us something about the attributes of that piece of traffic. A network packet arriving from the Internet may in principle carry any source address, any destination address, any protocol, and any set of user privileges. If the packet is accepted by a particular rule in the border firewall, then we can infer that its attributes do not satisfy the preconditions of the earlier rules in the firewall's rule-set, but do satisfy the preconditions of the rule that it matched. Every time a rule recognizes traffic, it refines the attributes of the traffic to reflect the constraining influence of the rules against which the traffic was tested at the firewall. Given a path in the rule graph, the consistency checker can compute successive refinements of the attribute set of traffic, user classes, and object permissions that can possibly traverse that path. At each stage along a path, the current attribute set is matched against the formal specification of the access constraints to confirm whether it is in violation.

Exhaustive Analysis: In the exhaustive analysis mode, the consistency checker analyzes the multi-layered rule graph for the system in its entirety, considering each piece of configuration information collected. It then produces a complete list of sequences of access decisions (e.g., firewall rules or SELinux transition rules) that can result in traffic that violates one or more of the global access constraints. The analysis can be directed to use a specific starting point as the initial source of the traffic (e.g., a particular host or the WAN/Internet); if a starting point is not specified, all the hosts in the topology are considered starting points in turn.

The data structures for rule graphs and traffic attribute sets, and the algorithms to manipulate them, are very critical to the analysis performance. We use a number of optimizations for analysis speed-up. They include the use of multidimensional interval trees for representing the network traffic component

of the attribute sets and custom set data structures for efficient representation of the discrete component of the attribute sets (e.g., security contexts or object permissions). We also use intelligent caching of results of analyses of sub-paths to minimize repeated computations.

Statistical Analysis: When the networked system (considered together with the relevant components of the IT network) being analyzed is large and deep, the sheer combinatorics of the possible interactions among the access control devices and mechanisms may render a comprehensive exhaustive analysis computationally impossible. The underlying rule graph would simply contain too many paths to allow an exhaustive exploration, especially when the actual paths of interest (i.e., the violations) form a very small subset of the set of all possible paths through the rule graph.

To handle this challenge, APT incorporates an advanced statistical analysis mode that produces a sample (likely incomplete) set of policy violations, and a quantitative estimate of the remainder. The latter may take the form of estimates of the total number of violations, the fraction of all traffic that violates global policy, or the probability that there are no violations given that none were discovered after the analysis was performed for a specified amount of time.

The tool obtains statistically valid estimates of such quantitative measures without actually exploring the entire rule graph. It does so through repeated and random exploration of an extended version of the rule graph to sample a few paths, using mathematically formulated heuristics to guide the choice at each step of the exploration towards the likely sources of policy violation. The desired set of metrics is calculated for each sampled path, and then we use "importance sampling" [25] to remove the bias introduced by the guidance heuristic and obtain unbiased estimators of the metrics. As the analysis continues, the user can watch the progress of the analysis, including the convergence of the chosen set of metrics using the graphical front-end. The process can be continued until a user-specified relative error bound is reached or a user-specified fraction of the rule graph has been explored. The user can also abort the analysis at any time.

The key idea of importance sampling can be illustrated by a simple example. Let X be a random variable with probability density function p, $A \subset \mathbb{R}$, and $\gamma = \Pr\{X \in A\}$. Define the *indicator random variable* $1_{\{X \in A\}}$ as follows: sample from X's distribution and observe x, if $x \in A$ set $1_{\{X \in A\}} = 1$, otherwise set $1_{\{X \in A\}} = 0$. It follows that $\gamma = E_p[1_{\{X \in A\}}]$, with the expectation taken with respect to p. The naive Monte Carlo approach for estimating γ would be to generate N samples, $X_1, X_2, \ldots, X_N$, using the density p, and then use

$\hat{\gamma}_N = \frac{1}{N}\sum_{i=1}^{N} 1_{\{X_i \in A\}}$ as an unbiased estimator of γ ($E_p[\hat{\gamma}_N] = \gamma$). The width of the confidence interval around $\hat{\gamma}_N$ will be proportional to the standard deviation $\sqrt{\gamma(1-\gamma)/N}$. For the estimate to be meaningful, the magnitude of $\hat{\gamma}_N$ needs to be significantly larger than the window of uncertainty around it, i.e., the ratio of sampling standard deviation to mean (*relative error*) needs to be small. If A corresponds to a rare event, i.e., γ is very small, the relative error is approximately $1/\sqrt{\gamma N}$, and obviously a very large N is needed to minimize the relative error.

Importance sampling achieves small relative error using significantly smaller N than this standard approach calls for. We define another probability density function $p'(x)$, with $p'(x) > 0$ for all $x \in A$ for which $p(x) > 0$. Then, $\gamma = E_{p'}[1_{\{X \in A\}} L_{p'}(X)]$, where $L_{p'}$ is the likelihood ratio, i.e., $L_{p'}(x) = p(x)/p'(x)$, and the subscript in the expectation denotes sampling using density p'. We use p' to generate N samples $X_1^{'}, X_2^{'}, \ldots, X_N^{'}$, and $\hat{\hat{\gamma}}_N = \frac{1}{N}\sum_{i=1}^{N} 1_{\{X_i^{'} \in A\}} L_{p'}(X_i^{'})$ is an unbiased estimator of γ. The sampling with a different density is usually referred to as a "change of measure" and p' is called the "importance sampling density." It has be shown [25] that to reduce the variance of the estimator $\hat{\hat{\gamma}}_N$, we need to make the likelihood ratio $p(x)/p'(x)$ small on the set A. Since p is a given, this means that we should choose a p' that biases towards $x \in A$. The aim then is that through use of importance sampling, the relative error associated with N samples will be smaller than it would be for N samples from a normal Monte Carlo scheme. This implies greater accuracy for the estimator, meaning significantly better results for the same number of samples.

An identical discussion applies for estimating $\alpha = E_p[W(X)1_{\{X \in A\}}]$, where W is some function of the random variable.

Importance sampling has a long history in contexts as diverse as estimations of integrals in high dimensions, chemistry, communications, reliability, and economics. We have used it ourselves to estimate measures of an attacker's success against a system with known vulnerabilities [26]. It is a general technique, but the challenge in using it lies in applying the general theory to a specific context.

We have formalized the approach for use in APT, developing heuristics for change of measure that exploit domain knowledge. It must be understood that all randomness in this formulation comes from the randomness of sampling, not from the system itself. If we choose a path through a rule graph randomly, we can say something about the probability of the traffic description at the end of the path having certain properties or metrics, but the probability is with respect to the randomness used in choosing the path. The path chosen is like the random variable X in the example above, and the event of the path violating the global

policy is like an indicator $1_{\{X \in A\}}$. Probability measure p reflects the random selection of rules that define the path.

We now describe the formulation for estimating the total number of paths through the rule graph that end in deviation from global policy. In the interest of space, we will focus our discussion on network layer rule graphs, and later provide a brief description into how the analysis applies to multi-layered rule graphs. The first step is to identify a useful sample space, or set of events that can occur during sampling. Let $\mathcal{S}$ be the set of all rule-graph nodes. Let $\mathcal{S}_0$ denote the set of root nodes, i.e., the traffic sources. A *sample sequence* denotes any infinite sequence $\{d_i \in \mathcal{S},\, i > 0\}$. There is no supposition of any kind of structure except that each element of the sequence is drawn from $\mathcal{S}$. We use the term *valid prefixes* to describe legitimate complete paths in the rule-graph, starting in $\mathcal{S}_0$ and terminating in a final policy decision (e.g., firewall denies, or a host accepts). "Legitimate" here means that there exists a packet with attributes such that the firewall rules force it to traverse that path.

We say that a valid prefix is *interesting* if the corresponding path conflicts with global policy, and let $\mathcal{V}$ be the set of all interesting prefixes. Let the sample space, Ω, be the set of all sample sequences. We define a probability measure $\mathcal{P}$ on Ω such that all sample sequences are equally likely. We can enumerate the nodes in $\mathcal{S}$, transform each index into a single digit in the base $|\mathcal{S}|$ number system, and define a random variable X on Ω that maps a sample sequence to a real number $(0.d_1 d_2 d_3 \ldots)_{|\mathcal{S}|}$, where d_i are the base $|\mathcal{S}|$ digits representing the nodes. Hence, X provides a one-to-one correspondence between Ω and $[0, 1]$. Therefore, under $\mathcal{P}$, X is uniformly distributed over $[0, 1]$, and has a probability density function $p(x) = 1$ if $x \in [0, 1]$ and 0 otherwise. Henceforth, we drop the subscript $|\mathcal{S}|$, and all real numbers are in base $|\mathcal{S}|$.

The set of all sample sequences with the interesting prefix $(0.d_1 d_2 \ldots d_l)$ (i.e., all sample sequences whose first l elements are identically $d_1, d_2, \ldots, d_l$) form a closed interval of length $|\mathcal{S}|^{-l}$ starting at $(0.d_1 d_2 \ldots d_l)$. For every interesting prefix v, the corresponding interval is denoted by I_v, and its length by l_v. Notice that if $v, w \in \mathcal{V}$, then $I_v \cap I_w = \emptyset$. It can be seen that $\mathcal{A} = \bigcup_{v \in \mathcal{V}} I_v$ forms a finite union of disjoint intervals. Hence, the probability of a uniform sample $u \in [0, 1]$ also having $u \in \mathcal{A}$ is the sum of lengths of the constituent intervals. Due to the sheer size of Ω, this total probability would be very small, and membership in $\mathcal{V}$ would be a rare event under uniform sampling on $[0, 1]$.

The objective of this formulation is the construction of an unbiased estimator of the total number of interesting prefixes (i.e., $|\mathcal{V}| = \mathcal{T}$). We define a function $W : [0, 1] \to \mathbb{Z}_+$ that takes the value $|\mathcal{S}|^l$ for all points in an interval corre-

sponding to an interesting prefix of length l, and is zero elsewhere. Therefore, we have

$$\begin{aligned} E_p[W(X)] \equiv E_p[W(X)1_{\{X\in\mathcal{A}\}}] &= \int_{-\infty}^{\infty} W(x)1_{\{x\in\mathcal{A}\}}p(x)\,dx \\ &= \int_{x\in\mathcal{A}} W(x)p(x)\,dx = \sum_{v\in\mathcal{V}}\left(\int_{x\in I_v} W(x)p(x)\,dx\right) \\ &= \sum_{v\in\mathcal{V}}\left(\int_{x\in I_v} |\mathcal{S}|^{l_v}\,dx\right) = \sum_{v\in\mathcal{V}}\left(|\mathcal{S}|^{l_v}\int_{x\in I_v} dx\right) \\ &= \sum_{v\in\mathcal{V}} |\mathcal{S}|^{l_v}|\mathcal{S}|^{-l_v} = |\mathcal{V}| = T, \end{aligned} \tag{1}$$

where the introduction of summation is possible because $\mathcal{A}$ is a finite union of disjoint I_v's. Hence, we estimate $|\mathcal{V}|$ by estimating $E_p[W(X)]$. As described earlier, the standard unbiased estimator for T would be $\hat{T}_N = \frac{1}{N}\sum_{i=1}^{N} W(X_i)$, where $X_1, X_2, \ldots, X_N$ are N random samples drawn from $[0,1]$ (or, equivalently, from Ω) with density $p(x)$. However, this estimation would be extremely inefficient, as most samples would not correspond to valid prefixes, let alone interesting ones. The value of this formulation is that *given* x that corresponds to a valid prefix, the computation of the probability mass it represents under uniform sampling is straightforward. Each sample under the importance sampling approach randomly generates a path $d_1, d_2, d_3, \ldots$ through the rule graph, stopping when a final policy action is applied. The path chosen automatically corresponds to a valid prefix. The probability distribution used to select d_i given $d_1, d_2, \ldots, d_{i-1}$ is the biased sampling strategy. Whatever that strategy is, $q_{v,i}$ denotes the probability of selecting d_i, given the prior selections $d_1, d_2, \ldots, d_{i-1}$, understanding that $v = (d_1, d_2, \ldots, d_i)$. Mapping a selected sequence back into a real number x gives rise to a new probability density function p' on $[0,1]$, driven by our biasing strategy. As we have shown in [26], this is a valid change of measure, and the value of $p'(x)$ for $x \in I_v$ for some interesting prefix v is

$$p'(x) = \frac{\prod_{i=1}^{l_v} q_{v,i}}{\text{length of } I_v} = \frac{\prod_{i=1}^{l_v} q_{v,i}}{|\mathcal{S}|^{-l_v}} = |\mathcal{S}|^{l_v}\prod_{i=1}^{l_v} q_{v,i} \quad \text{for all } x \in I_v. \tag{2}$$

Therefore, using importance sampling, we sample N valid prefixes according to our biasing strategy, map each into base $|\mathcal{S}|$ numbers $X_1, X_2, \ldots, X_N$, and construct the estimator $(1/N)\sum_{i=1}^{N}(p(X_i)/p'(X_i))W(X_i)$. Despite the large numbers symbolically expressed above (e.g., $|\mathcal{S}|^{l_v}$), the actual computations used in estimating $E_p[W(X)]$ can be done involving much smaller values, avoiding the risk of round-off and precision errors.

As systems being analyzed grow in size and sophistication, we anticipate that APT's statistical analysis mode will become increasingly relevant, providing the tool with superior scalability. We previously demonstrated the efficacy of our techniques in our work on fast model-based penetration testing [26], in which we analyzed a model of a networked system and attacker behavior to quantify the system's security posture. We were able to analyze system models with 2^{1700} states, which is a dramatic improvement over the state-of-the-art (2^{229} states) [22].

6 Experimental Evaluation

In this section, we demonstrate the efficacy of APT by using it to verify the access control policy implementation in a variety of settings. We begin with test-cases that are relatively small in size, such as the networking system typically found in a process control setting, and show how the exhaustive analysis can be used to weed out misconfigurations of access control devices. We then demonstrate APT's scalability through the use of statistical analysis for analyzing a much larger system that is beyond the capability of the exhaustive analysis. In all the experimental evaluations described below, APT's analysis engine was running on an AMD AthlonXP-64 3700+ machine with 2GB of RAM.

Figure 5 shows a testbed developed at the Sandia National Labs to represent a networking infrastructure at a typical Oil and Natural Gas (ONG) operator. The focus here is on the protection of the process control network. The network contains two dedicated Cisco PIX firewalls, each with 15 rules. Out of the 17 hosts in the network, 5 have host-based firewalls (iptables in Linux). All the hosts run the stock versions of the Windows or Linux operating systems. The rulesets for the various firewalls were populated based on the settings found at a number of actual ONG sites.

The global access constraints were defined, using APT's graphical front-end, to emphasize the tightly controlled isolation of the process control network from the rest of the IT network. In particular, they specified that the hosts in the process control network could access each other; however, only one (the PCN Historian) could be accessed from outside the process control network. The only ways the PCN Historian could be accessed were via the DMZ Historian, by a "sysadmin" class user, and via a host in the corporate network, by a "manager" class user. APT's graphical front-end facilitated easy specification of these constraints, which were subsequently translated into the underlying XML representation of the global policy.

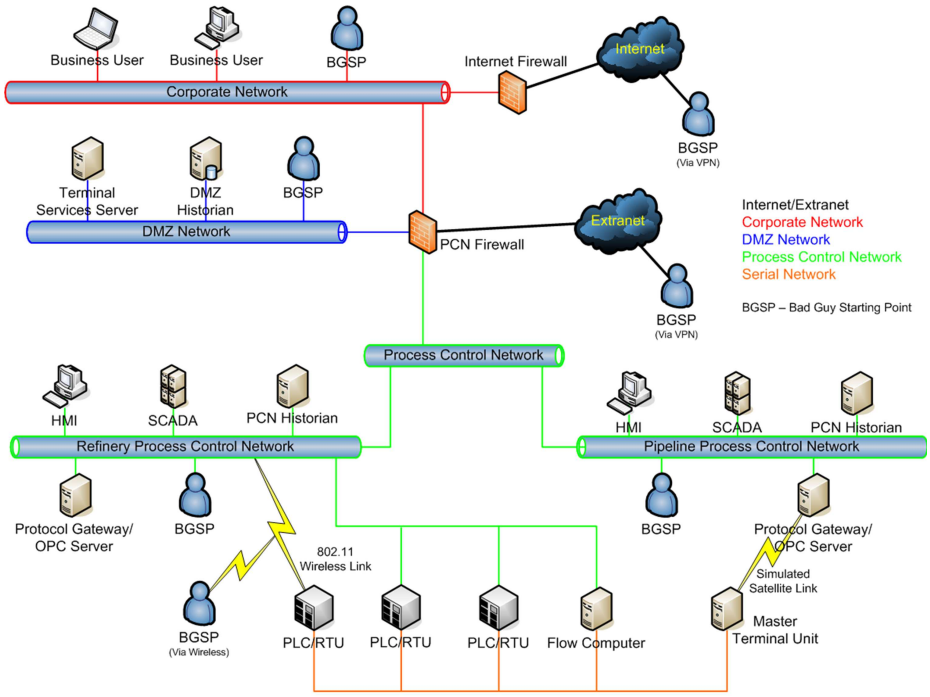

Fig. 5. A Representative Process Control Testbed (courtesy Sandia Nat. Lab.)

APT's analysis engine, using the information about the network topology provided by the front-end, automatically and securely captured the configuration snapshot of the system. An exhaustive analysis of the system was then performed. The analysis identified a total of 83 paths (sequences of firewall rules) in 10 seconds. The tool was then instructed to perform a post-analysis on the results to further pinpoint the root causes of the violations. The post-analysis, based on frequency of occurrence in the list of violations, identified four rules, two in each of the two Cisco firewalls acting in series, as the likely culprit. It further identified the attributes of the traffic that they were allowing to pass and also indicated the subset of the global access constraints that the traffic was violating. Using the graphical front-end, we were able to test hypothetical modifications to the highlighted rules and perform quick re-evaluation to discover the appropriate changes that resulted in zero violations of the policy specification. As is evident from this experiment, APT's exhaustive analysis functionality was very useful for a reasonably small network. A complete analysis of the testbed indicated that the rule graph for the system had about 230,000 paths. Hence, the number of violations not only helps identify and correct misconfigurations,

but also, when viewed in the context of the total number of ways traffic could travers the system, can serve as a quantitative measure of the system's security posture.

We also used APT to analyze the sample scripts used in the evaluation of the "Firewall Policy Advisor" by Al-Shaer and Hamed [12–14]. We captured all the misconfigurations in their sample scripts [14]. Note that APT's functionality is a superset, since our analysis is not limited to the analysis of relations between just two firewalls, and we can check for inconsistencies (due to the use of multi-dimensional interval trees) as well as explicit policy violations.

To explore the scalability of the tool, we tested it on the example setup shown in Figure 1. This testbed represents an intrusion-tolerant publish-subscribe system developed as part of a DARPA-funded research effort. The system contains 1) over 40 hosts, of which 29 are running SELinux (each with 4 or more process domains) and all of which have hardware NIC-based firewalls (with more than 20 rules per firewall), and 2) 8 Cisco PIX firewalls. As can be imagined, the configuration of the large number of access control elements in this system in adherence to the global security policy is an extremely complicated task.

The global access constraints were set to emphasize tightly controlled access to the "System Manager" group of machines (the top row of 4 machines in the figure). These machines could access one another in very specific ways (to run Byzantine fault-tolerance algorithms) and could be accessed by only a very small set of other machines, and only by processes from specific domains on those machines. Again, APT's graphical front-end and underlying XML schema were used for the specification of the global access constraints.

We then deliberately misconfigured a rule in the hardware NIC-based firewall for one of the system manager machines, allowing for traffic that would result in the violation of the global policy. The exhaustive analysis was able to identify the resulting 263 violations and pinpoint the root cause in about 140 seconds (focusing primarily on the firewall rules, and not on the host-based access control mechanisms). Note that the total number of paths in the rule-graph for this system is huge, which explains the time taken by the analysis.

We then made the problem more complex by introducing problems in two firewalls, a Cisco PIX dedicated firewall and the hardware NIC-based firewall for one of the system manager machines, such that the violation only occurred in the access decision sequences that included both rules. The exhaustive analysis, again limited to firewall rules, identified 155 violations and the two modified rules in about 200 seconds. However, when the host-based access control mechanisms were also included in the analysis, the exhaustive analysis could not complete,

even after running for more than 3 hours, setting the stage for a demonstration of increased scalability provided by the statistical analysis.

As described earlier, APT's statistical analysis can provide a sample set of violations and an estimate of the remainder. For that purpose, we use the total number of violations as the quantitative estimate of the remainder. The biasing heuristic we use for guiding importance sampling is based on shortest distance, i.e., it assigns higher weight to those access decisions that would guide the traffic closest (in a network topology sense) to the hosts that are included in the specification of the global policy. The stopping criterion for the sampling was 500,000 samples or 5% relative error with 95% confidence, whichever was achieved first.

Statistical analysis on the Figure 1 testbed for the one-rule misconfiguration example described above resulted in an answer in under 10 seconds (with the relative error convergence being the stopping criterion). The analysis estimated 255 violations, which is within 3% of the exact answer obtained from exhaustive analysis.

For the example with two misconfigured rules, the statistical analysis obtained an answer within 4% of the exact answer in about 10 seconds when only firewall rules are being analyzed, and in about 25 seconds when host-based access control mechanisms are also being modeled.

We tested another example, in which, in addition to the two misconfigured rules, we also introduced a misconfiguration in the SELinux policy for the host with the misconfigured host-based firewall, such that the global policy was now violated only when all three access control points were included. Again, the exhaustive analysis could not produce the complete list of violations after running for more than 3 hours, but the statistical analysis was able to provide an estimate with a 10% confidence interval (where the number of samples analyzed was the stopping criterion) in about 1 minute. Additional accuracy required a disproportionate increase in the time required for the analysis. To obtain an estimate with a 5% confidence interval, it was necessary to run the analysis for about 5 minutes.

Hence, we can see that statistical analysis allows APT to analyze fairly complex systems within reasonably short periods of time. The tool's performance can likely be improved further with the use of better biasing heuristics, which is an avenue that we will continue to explore in the future.

7 Conclusion

Deployment of a large number of distributed and layered access control mechanisms, such as firewalls, is the staple solution to the problem of enforcing security policy. Hence, it is very important to ensure that all the access control mechanisms work collectively in harmony, and that their complex interactions do not mask subtle errors and thus introduce security vulnerabilities. In this paper, we have described the Access Policy Tool, which provides an easy way for network system administrators and security experts to analyze their security policy implementations (configurations of various devices on their networks) for conformance with global security specifications. It features ease of information management through a graphical front-end, offline as well as online analysis of the compliance of a configuration with the desired or intended behavior, and scalability through the use of statistical techniques for the estimation of security posture. Those capabilities allow users to gain confidence in an implementation before actual deployment, make it possible to reason about policy at an increased level of abstraction, and permit system administrators to detect policy holes created during operational use of a system.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0524695 and the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the views of the National Science Foundation, or the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

The authors would also like to thank Jenny Applequist for her editorial assistance.

References

1. SANS Institute: SANS Top-20 2007 Security Risks (2007 Annual Update). http://www.sans.org/top20/ (Apr 2008)
2. Wool, A.: A Quantitative Study of Firewall Configuration Errors. Computer **37**(6) (Jun 2004) 62–67
3. Netfilter.org: The Netfilter.org "iptables" Project. http://www.netfilter.org/projects/iptables/index.html (Apr 2008)

4. 3Com Corporation: 3Com Embedded Firewall Solution. http://www.3com.com/other/pdfs/products/en_US/400741.pdf (Apr 2008)
5. Centre for the Protection of National Infrastructure: NISCC Good Practice Guide on Firewall Deployment for SCADA and Process Control Networks. http://www.cpni.gov.uk/Docs/re-20050223-00157.pdf (Apr 2008)
6. National Security Agency: Security-Enhanced Linux. http://www.nsa.gov/selinux/index.cfm (Apr 2008)
7. Cisco Systems: Cisco Security Agent. http://www.cisco.com/en/US/products/sw/secursw/ps5057/index.html (Apr 2008)
8. Gouda, M.G., Liu, A.X.: Firewall Design: Consistency, Completeness, and Compactness. In: Proc of DSN 2004, Tokyo, Japan (Mar 2004) 320–327
9. Hari, A., Suri, S., Parulkar, G.M.: Detecting and Resolving Packet Filter Conflicts. In: Proc of IEEE INFOCOM (Vol. 3), Tel Aviv, Israel (Mar 2000) 1203–1212
10. Eppstein, D., Muthukrishnan, S.: Internet Packet Filter Management and Rectangle Geometry. In: Proc of ACM-SIAM Symp on Discrete Algorithms (SODA 2001), Society for Industrial and Applied Mathematics (2001) 827–835
11. Boboescu, F., Varghese, G.: Fast and Scalable Conflict Detection for Packet Classifiers. Computer Networks **42**(6) (2003) 717–735
12. Al-Shaer, E., Hamed, H.: Firewall Policy Advisor for Anomaly Detection and Rule Editing. In: Proc of IEEE/IFIP 8th Intl Symp on Integrated Network Mgmt (IM 2003), Colorado Springs, CO (Mar 2003) 17–30
13. Al-Shaer, E., Hamed, H.: Management and Translation of Filtering Security Policies. In: Proc of the 38th IEEE Intl Conf on Communications (ICC 2003), Anchorage, AK (May 2003) 256–260
14. Al-Shaer, E., Hamed, H.: Discovery of Policy Anomalies in Distributed Firewalls. In: Proc of IEEE INFOCOM (Vol. 4), Hong Kong (Mar 2004) 2605–2616
15. Guttman, J.D.: Filtering Postures: Local Enforcement for Global Policies. In: Proc of IEEE Symp on Security and Privacy, Oakland, CA (May 1997) 120–129
16. Bartal, Y., Mayer, A., Nissim, K., Wool, A.: Firmato: A Novel Firewall Management Toolkit. ACM Trans on Computer Sys **22**(4) (2004) 381–420
17. Cisco Systems: CiscoWorks Management Center for Firewalls. http://www.cisco.com/en/US/products/sw/cscowork/ps3992/index.html (Apr 2008)
18. Skybox Security: Automated Firewall Management Software. http://www.skyboxsecurity.com/products/FCA.html (Apr 2008)
19. RedSeal Systems: RedSeal Security Risk Manager. http://www.redseal.net/Product-Overview.shtml (Apr 2008)
20. Clark, E., Grumberg, O., Peled, D.: Model Checking. The MIT Press, Cambridge, MA (2000)
21. Ritchey, R.W., Ammann, P.: Using Model Checking to Analyze Network Vulnerabilities. In: Proc of IEEE Symp on Security and Privacy, Oakland, CA (2000) 156–165

22. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated Generation and Analysis of Attack Graphs. In: Proc of IEEE Symp on Security and Privacy, Oakland, CA (2002) 273–284
23. Ou, X., Govindavajhala, S., Appel, A.W.: MulVAL: A Logic-Based Network Security Analyzer. In: Proc of 14th USENIX Security Symp, Baltimore, MD (2005) 113–128
24. Ou, X., Boyer, W.F., McQueen, M.A.: A Scalable Approach to Attack Graph Generation. In: Proc of 13th ACM Conf on Computer and Communications Security (CCS 2006), Alexandria, VA (2006) 336–345
25. Heidelberger, P.: Fast Simulation of Rare Events in Queueing and Reliability Models. ACM Trans on Modeling and Computer Simulations **5**(1) (1995) 43–85
26. Singh, S., Lyons, J., Nicol, D.M.: Fast Model-Based Penetration Testing. In: Proc of 2004 Winter Simulation Conf (WSC'04), Washington, DC (Dec 2004) 309–317