Monitor Placement for Large-Scale Systems

Nirupama Talele
Penn State University
IST,University Park
State College, PA-16802
nrt123@psu.edu

Jason Teutsch
Penn State University
IST,University Park
State College, PA-16802
teutsch@cse.psu.edu

Robert Erbacher
Army Research Lab
2800 Powder Mill Rd
Adelphi, MD-20783
robert.f.erbacher.civ@mail.mil

Trent Jaeger
Penn State University
IST,University Park
State College, PA-16802
tjaeger@cse.psu.edu

ABSTRACT

System administrators employ network monitors, such as traffic analyzers, network intrusion prevention systems, and firewalls, to protect the network's hosts from remote adversaries. The problem is that vulnerabilities are caused primarily by errors in the host software and/or configuration, but modern hosts are too complex for system administrators to understand, limiting monitoring to known attacks. Researchers have proposed automated methods to compute network monitor placements, but these methods also fail to model attack paths within hosts and/or fail to scale beyond tens of hosts. In this paper, we propose a method to compute network monitor placements that leverages commonality in available access control policies across hosts to compute network monitor placement for large-scale systems. We introduce an equivalence property, called *flow equivalence*, which reduces the size of the placement problem to be proportional to the number of unique host configurations. This process enables us to solve mediation placement problems for thousands of hosts with access control policies containing of thousands of rules in seconds (less than 125 for a network of 9500 hosts). Our method enables administrators to place network monitors in large-scale networks automatically, leveraging the actual host configuration, to detect and prevent network-borne threats.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Control*

Keywords

Monitor Placement; Information flow graph scalability; Large scale systems

SACMAT'14, June 25–27, 2014, London, Ontario, Canada.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2939-2/14/06 ...$15.00.
http://dx.doi.org/10.1145/2613087.2613107 .

1. INTRODUCTION

System administrators are responsible for the security of all the hosts in their networks. They aim to prevent the software on their hosts from being compromised and to protect critical organizational data from leakage and/or unauthorized modification. System administrators often leverage *network monitoring* in the form of firewalls [9], traffic analysis tools [3], and network intrusion prevention systems [34] to block attacks. Such tools examine packets destined for networked processes in the host and packets produced by those processes to detect malicious input data and leaked secret data, respectively.

The problem for system administrators is to determine how to leverage network monitoring to protect their hosts effectively. There are two challenges. First, system administrators must determine where to place network monitoring. A naive approach would be to monitor at each networked device, but monitoring incurs a cost, both towards deployment (e.g., configuring monitoring) and in terms of performance (e.g., overhead of monitoring). Second, system administrators must determine which rules to enforce at each monitor. While system administrators often leverage community knowledge (e.g., Snort rule bases) about known malicious behavior to detect or block attacks at monitors, these rules may not pertain to that monitoring location or be specific to different configurations, and therefore miss some attacks. Also, system administrators must be careful when modifying such rule bases to avoid introducing false positives.

Researchers have explored methods to both reason about adversary attack paths and to place monitoring to cover all known paths. Attack trees [21] and attack graphs [33, 29, 1] model possible actions of adversaries that may lead to the compromise of a valuable resource. However, building either attack trees or attack graphs currently requires knowledge about the likely vulnerabilities on individual hosts, which may be incomplete (i.e., previously-unknown vulnerabilities may be missed) and brittle (i.e., vulnerabilities may be patched). Alternatively, researchers have developed methods to place security monitoring to block or limit adversary access to prevent attacks based on classical problems [27, 30, 17]. These methods focus on only one layer of the system, such as the network, a single host, or a single program because the size of the graphs becomes prohibitive. A recent work that reasons about data flows in distributed systems only handles systems with tens of hosts [23]. As a result, such methods are not usable for organizations with several networks containing many hosts.

In this paper, our goal is to develop a method that enables the placement of network monitoring for the actual threats present in an organization-wide deployment. This work is motivated by Talele *et al.* who build summaries of individual hosts to improve scalability to networks of tens of hosts with fine-grained access control policies [37], such as the SELinux reference policy that contains over 50,000 rules [28]. We identify several insights that enable additional, significant improvements in scalability. First, many hosts are launched from the same OS distribution, which today come pre-configured with an access control policy, consequently all hosts running that OS distribution have the *same access control policy*. Second, many hosts assume the same "role" in an organization, such as network (e.g., DHCP or DNS) server, web server, database, web client, often resulting in the *same information flows per host*. Third, we find that network connections among hosts are often equivalent from a security standpoint, in which case we obtain the *same threats for hosts with equivalent network connections*. Using these insights, we define three equivalence relations for hosts that enable merging of equivalent host graphs. Further, we prove that one equivalence property, called *flow equivalence*, reduces the size of the placement problem to be proportional to the number of unique host configurations, rather than the number of hosts.

We use these equivalence properties to examine how to produce network monitor placements in large-scale, heterogeneous networks. First, we develop a method for computing merged graphs from network configurations, host access control policies, and target applications of the hosts. We show that solutions can be produced for the merged graphs that are equivalent to those that would be computed from the original graph using standard algorithms. Second, we demonstrate the impact of our approach on a heterogeneous network configuration [25], finding that it works well for all types of networks except ad hoc (i.e., where connections cannot be predicted). In this example, merging enables a significant reduction in the sizes of graphs, from millions of nodes to a few thousand, enabling network monitor placements to be computed in 1-2 minutes, where the entire process of generating the merged systems and computing placements takes 10-40 minutes. While we do not produce specific monitor placement code, we produce the placement of network monitoring and the associated security requirements, which correspond approximately to network monitoring rules. Generating specific network monitor rules is future work.

In this work, we make the following contributions.

- We define three equivalence relations among hosts, called *concrete*, *label*, and *flow equivalence*, that enable all hosts in an equivalence class to be represented by a single *merged* host in a mediator placement problem without loss of information flow semantics.

- We show that by using these equivalence relations the size of a mediator placement problem is dependent on the number of unique host configurations, not the number of hosts. Thus, networks with significant redundancy among host configurations will see significant benefits.

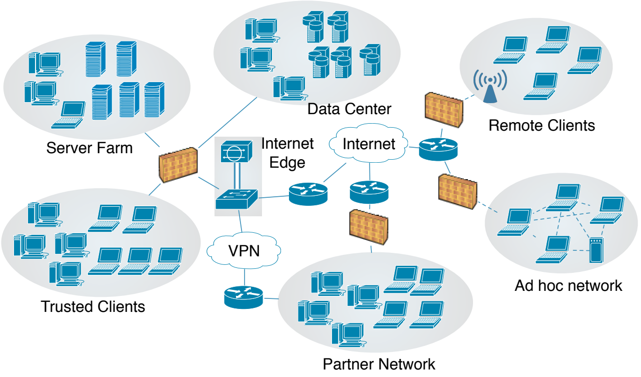

Figure 1: Example Organization's System: Containing six networks, two server and four client networks (one is wireless and one is ad hoc)

- We use the method to show that graphs representing the information flows in networks containing nearly 10,000 hosts can be compressed from millions of nodes and edges to just a few thousand without any loss of information flow semantics. Using this method, monitor placement for such networks can be computed in slightly over 2 minutes.

The rest of the paper is as follows. In Section 2, we identify the challenges and goals in network monitor placement. In Section 3, we review the formal model for reasoning about network monitor placement and highlight the challenges in solving the problem. In Section 4, we outline the proposed approach to producing network monitor placements in large-scale networks and define three equivalence relations for merging host information flow models. In Section 5, we outline a method for computing network monitor placements that leverages the merging offered by these equivalence classes. In Section 6, we evaluate the method analytically and experimentally. In Section 7, we compare our approach to related work. We conclude by summarizing our approach and results in Section 8.

2. EXAMPLE SCENARIO

Figure 1 shows an example of an organization's computing system. In this example, the organization deploys a set of web applications across six networks, two server networks and four client networks. While the exact deployments vary, the server-side deployments of the web applications generally utilize edge servers (e.g., firewalls, load balancers, etc.) that forward requests to one or more web servers (e.g., Apache, IIS), which may then leverage application servers (e.g., Tomcat) to implement the core application functionality by retrieving the necessary data from database servers. The server-side deployments are almost exclusively wired networks with a well defined network topology and extensive connectivity among the server layers.

On the other hand, the clients may access server applications through more varied network configurations. Client networks may be wired or wireless, and the wireless networks may be 802.11, cellular, or ad hoc (e.g., MANETs). Clients on organization networks are protected by firewalls and may be isolated by technologies, such as VLANs, but otherwise the structure of such networks is relatively flat.

For clients outside organizational networks, organizations often offer their employees services to access internal applications (e.g., VPNs).

Modern organizations often control the configuration of their internal hosts. For convenient management, it is often common for server hosts performing the same task to be configured identically. For example, all web servers may be configured using the same OS distribution and many may support the same web applications to enable load balancing. In addition, while organizational clients may be deployed on a variety of platforms, including traditional hosts (e.g., desktops and laptops) and a variety of new devices (e.g., phones, tablets, etc.), organizations often control the applications (and versions) that run on these devices to ease management as well. In many cases, client users are not allowed to download new applications to organizational machines.

System administrators are concerned with a variety of threats to the confidentiality, integrity, and availability of their application processing throughout their organization. Threats may originate externally, from other internal networks, and from hosts within the same network. While external hosts may be treated as fully untrusted, our particular interest is in tracking attacks that may be propagated from within organizational networks. A common problem is that unprivileged processes on one host are compromised and then used as a stepping stone to more advanced attacks compromising security-critical hosts that impact all hosts on the network (e.g., a Windows domain controller). Thus, we aim to account for data flows among processes on hosts, as well as data flows among hosts across entire organizations.

In this work, we propose to develop automated methods to place network monitors to log and/or block unsafe communications for large-scale (organizational) networks. Conceptually, the goal is to produce a minimal cost monitor placement that blocks all access to vulnerabilities (i.e., no false negatives), that does not block any legitimate functionality (i.e., no false positives), and does not include any spurious monitoring (i.e., no unnecessary overhead). In practice, such goals are ideal, but experience has shown that false positives must be prevented, while spurious monitoring and false negatives must be minimized.

3. MEDIATOR PLACEMENT PROBLEM

The problem of determining where to place network monitoring to block or log all possible attack paths is an instance of the *mediator placement problem* [18, 30]. The mediator placement problem aims to resolve all information flow errors, as defined in an information flow model, such as the one below (adopted from Talele *et al.* [37]).

Definition 1. *An* information flow model, $\mathcal{I} = (\mathcal{G}, \mathcal{L}, \mathcal{M})$, *consists of the following concepts:*

1. *A* directed data flow graph $G = (V, E)$ *consisting of a set of nodes* V *connected by edges* E*.*

2. *A* lattice $\mathcal{L} = \{L, \preceq\}$*. For any two labels* $l_i, l_j \in L$, $l_i \preceq l_j$ *means that* l_i *'can flow to'* l_j*.*

3. *A* label mapping function $M : V \rightarrow \mathcal{P}^L$ *where* $\mathcal{P}^L$ *is the power set of* L *(i.e., each node is mapped either to a set of labels in* L *or to* $\emptyset$*).*

4. *The lattice imposes security constraints on the information flows enabled by the data flow graph. Each pair* $u, v \in V s.t.\ [u \hookrightarrow_G v \wedge (\exists l_u \in M(u), l_v \in M(v).\ l_u \not\preceq_{\mathcal{L}} l_v)]$*, where* $\hookrightarrow_G$ *means there is a path from* u *(source) to* v *(sink) in* G*, represents an* information flow error*.*

In this model, the possible data flows are edges that propagate labels representing the security requirements on system data among nodes that represent system resources (subjects and objects). The lattice of labels represents the legal flows of labeled data that every operation of the system must satisfy. While lattice policies are traditionally associated with multilevel security [5, 6], more general policies are possible, such as policies constructed from sets of individual security requirements [19, 42] that we will leverage in this paper. When data with incompatible labels reach the same node, an information flow error results. It has been shown that information flow errors [24, 16, 32, 8] can be found automatically using such a model.

A solution that resolves all information flow errors mediates all paths to those errors by imposing security requirements (i.e., labels required by the sink). An *edge mediator* (or simply, mediator), $R = ((u, v), l)$, where $(u, v) \in E$ is an edge and $l \in L$ is the label of the data propagated on that edge. In general, the *mediator placement problem* is to find the minimal cost placement of mediators that resolve all errors in an information flow model.

Researchers have explored methods for solving the mediator placement problem to monitor security in networks [27], hosts [23, 30], and individual programs [18, 13, 20]. These techniques convert the operations authorized by network policies, network topology, host policies, and program code, respectively, into *data flow graphs.* They then identify threats and security requirements of the system, define the legal information flows as a *lattice* of labels representing these threats and security requirements, and define a *label mapping function* to associate the threats and security requirements with their sources and sinks, respectively. The security requirements at sinks are mostly domain-specific, and may be added by OS distributors and/or system administrators. Researchers have demonstrated that the mediator placement problem can be reduced to well-known graph problems, such as directed multicut (i.e., graph cut for multiple pairs of terminals) and vertex cover. Although these problems are NP-complete, several greedy algorithms are available (e.g., union the solution to individual cut problems). In fact, the equivalence between such problems has been shown formally [18].

The main limitation of the above approaches is scalability. Organizations may consist of thousands of hosts. In addition, each host may run many processes each with complex interactions. The policy that governs how Linux processes may legally communicate contains tens of thousands of rules [28]. Finally, individual programs also implement complex data flows. As a result, most prior methods for solving mediator placement problems only reason about one level of the system, such as the network [27], hosts [23, 30], or individual programs [13, 20, 18]. When researchers consider all these layers, the problem was limited to a small number of machines [23]. Talele *et al.* proposed a method whereby summaries of individual hosts are produced [37], yet only problems consisting of tens of hosts could be solved. Our goal is to develop methods for reasoning about organizational networks in their entirety.

4. MERGING REDUNDANT HOSTS

The key to placing network monitors in large-scale networks is removing the redundancy from instances of the information flow model of Definition 1. In this section, we leverage the insight that there is potentially a significant amount of redundancy among hosts. The commonality assumption is based on the understanding of various information available on the corporate and university networks studied. Using this insight, we propose three, progressively more ambitious equivalence relationships among hosts, *concrete*, *label*, and *flow equivalence*, that enable the merging of hosts that satisfy those relations.

4.1 Redundant Host Information Flows

We make the observation that in a distributed system, the system is composed from a set of interconnected hosts. Thus, we distinguish the contributions to the system information flow model of each host as a *host information flow model* (HIFM), where a HIFM for host i is defined as $\mathcal{I}_i = (G_i, \mathcal{L}, M_i)$, where $G_i = (V_i, E_i)$, security lattice $\mathcal{L}$ and label mappings $M_i : V_i \rightarrow \mathcal{P}^L$.

Viewing a system's single host at a time also requires a distinction of input and output between hosts. Input nodes are the nodes in a HIFM's data-flow graph that only receive input from nodes outside the data-flow graph, and output nodes are nodes in a HIFM's data-flow graph that only send output to nodes outside the data-flow graph. Formally, an edge (u, v) is an *input edge* for an HIFM data-flow graph G_i if $v \in V_i$ and $u \notin V_i$. v is then said to be an *input node* for host graph i, and for all edges (u, v) for an input node v imply that $u \notin V_i$. The set of input nodes of G_i are $I_i \subseteq V_i$. Second, an edge (v, u) is an *output edge* for G_i if $v \in V_i$ and $u \notin V_i$. v is then said to be an *output node* for G_i, and for all edges (v, u) for an output node v imply that $u \notin V_i$. The set of output nodes of G_i are $O_i \subseteq V_i$. The combination of input and output edges and nodes are called *I/O edges* and *I/O nodes*, respectively.

Our goal is to identify HIFMs that are equivalent with respect to the mediator placement problem. Intuitively, two HIFMs are equivalent if any equivalent mediator placement will either resolve all the information flow errors in both HIFMs or will fail to resolve at least one information flow error. If so, we find that we can *merge* the two HIFMs into one merged HIFM that represents all the information flows of both, reducing the size of the data-flow graph by removing one host sub graph. In some cases, some effort must be undertaken to ensure that the outputs produced by the merged HIFM is equivalent to that of the individual nodes. We discuss these requirements below, but detail the merging methods for each equivalence relation in Section 5.3.

We find that there is a significant redundancy among hosts in conventional systems because many hosts are now deployed from the same image. For example, many organizations produce a master image for hosts with specific roles in the organization, such as application-specific servers (e.g., web server and database) and employee clients. Using a single master image makes it easier to install hosts and also gives administrators more control over the security of the hosts.

From our perspective, the main impact of the use of master images is that the security policies of several hosts may be identical, potentially resulting in the same information flows. In modern systems, many OS distributions include a mandatory access control (MAC) policy [28, 4, 35, 41]. Researchers have previously shown that MAC policies define the possible data flows on a host [38, 16]. If the images are the same, then they will implement the same firewall policy dictating the I/O of the host. For hosts deployed for the same purpose (e.g., a generic client host or a specific server application), then the firewall policies will often allow only the same I/O. As a result, we find that many hosts implement the same data flows. We define the *data-flow equivalence* relation below, which we will use as a foundation for the later equivalence classes used for merging below.

Definition 2. *Two HIFMs* $\mathcal{I}_1 = (G_1, \mathcal{L}, M_1)$ *and* $\mathcal{I}_2 = (G_2, \mathcal{L}, M_2)$ *are said to be* data-flow equivalent $\mathcal{I}_1 \equiv_{df} \mathcal{I}_2$ *if:*

1. **Same data-flow graph***: There is a graph isomorphism between* G_1 *and* G_2*. Implied by the graph isomorphism is a bijection* $f : V_1 \rightarrow V_2$*, which maps nodes from graph* G_1 *to* G_2*. A node in one graph that is bijectively-mapped to a node in the other graph is said to* correspond.
2. **Corresponding inputs***: For every input node* $i \in I_1$ *in* G_1 *the corresponding node* $f(i) = j$ *that is an input node in* G_2*, such that* $j \in I_2$*.*
3. **Corresponding outputs***: For every output node* $o \in O_1$ *in* G_1 *the corresponding node* $f(o) = p$ *that is an output node in* G_2*, such that* $p \in O_2$*.*

It is easy to see that the data-flow equivalence relation reflexive, symmetric, and transitive, so it is an equivalence relation.

The key insight in this paper is that if many hosts have the same data flows, then many will face different versions of essentially the same mediator placement problem. While different HIFMs with the same data flow graphs may still have different label mappings in general, we find that in many cases installations that are configured for same application can apply the same label mapping. In addition, if the labels are distinct, applications dictate that label mappings be applied on the same subjects and objects in the MAC policy (i.e., the same nodes in the data flow graph). The rest of this section, we leverage this idea to define three equivalence relations for HIFMs that imply that the hosts have the same impact on the solution of the mediator placement problem.

4.2 Concrete Equivalence

We begin by defining a basic equivalence relation between HIFMs that serves as a foundation for the more subtle equivalence relations defined later. Intuitively, the idea is that if two hosts have the same data-flow graphs, label mapping functions, and the same input and output connections, then they will produce the same information flows along all corresponding edges. We call this *concrete equivalence* because the HIFM of the two hosts must be identical.

Definition 3. *Two HIFMs* $\mathcal{I}_1 = (G_1, \mathcal{L}, M_1)$ *and* $\mathcal{I}_2 = (G_2, \mathcal{L}, M_2)$ *are said to satisfy* concrete equivalence $\mathcal{I}_1 \equiv_c \mathcal{I}_2$ *if:*

1. $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *satisfy data-flow equivalence.*
2. **Corresponding input edges***: If* $i_1 \in I_1$*,* $(u, i_1) \in E$ *and* $f(i_1) = i_2$*, then* $(u, i_2) \in E$*.*
3. **Corresponding mappings***: If* $M_1(v_1) = L$ *and* $f(v_1) = v_2$*, then* $M_2(v_2) = L$*.*

4. **Corresponding output edges**: *If $o_1 \in O_1$, $(o_1, v) \in E$ and $f(o_1) = o_2$, then $(o_2, v) \in E$.*

This definition places many restrictions on equivalent hosts: two hosts have the same data flow graph, I/O nodes and edges, and, label mapping function. That is these are hosts that enforce the same security policies (same data flow), configured within the same network to the same other hosts (same I/O), and are applied to the same application security requirements (same label mapping). Clearly, in this case, these two hosts will have the same information flow error paths (since all the paths are the same).

Finally, note that the requirement for output equivalence can be relaxed. HIFMs that satisfy concrete equivalence requirements 1-3 above will always produce data of the same label at the corresponding output nodes. Thus, we can merge two HIFMs with different output edges by unioning these edges to the merged HIFM. HIFMs that satisfy only concrete equivalence requirements 1-3 are said to satisfy *concrete input equivalence.*

Example: Suppose two web servers have the same input connections (from edge servers and databases) and output connections (with databases and edge servers). If they also enforce the same MAC policy, then they satisfy concrete equivalence and can be represented by a single host sub graph. If the web servers had different output connections, then could still be merged because they satisfy concrete input equivalence.

4.3 Label Equivalence

While concrete equivalence will enable merging of hosts in the same network, it is not suitable for merging hosts in different networks. In that case, the hosts do not have the same concrete connections in the network topology. We find that two HIFMs that receive data of the same label at corresponding input nodes, also produce equivalent information flow errors. We call this *label equivalence.*

Definition 4. *Two HIFMs $\mathcal{I}_1 = (G_1, \mathcal{L}, M_1)$ and $\mathcal{I}_2 = (G_2, \mathcal{L}, M_2)$ are said to satisfy* label equivalence $\mathcal{I}_1 \equiv_l \mathcal{I}_2$ *if:*

1. *$\mathcal{I}_1$ and $\mathcal{I}_2$ satisfy data-flow equivalence.*
2. **Equivalent input mappings**: *If $M_1(i_1) = L$ and $f(i_1) = i_2$, then $M_2(i_2) = L$.*
3. **Corresponding mappings**: *If $M_1(v_1) = L'$ and $f(v_1) = v_2$, then $M_2(v_2) = L'$.*

In this case the main difference between concrete input equivalence is that we replace the corresponding input edges by equivalent input label mappings (requirement 2). This implies that if the labels of the data received at the input node is known and is same at each input, then the two HIFMs satisfy information-flow equivalence. In this case, the input labels, data flows, and label mappings are the same, so the information flow error paths will be the same.

Example: To understand when this would be applicable consider the following case. Suppose clients in two different offices depend on the same services (DHCP, DNS, etc.) administered by the same trusted party and are limited to the same set of web applications. In this case, the labels of the data that can be received by these clients could be determined in advance. Assuming that the two clients further enforce the same MAC policy and host firewall policy (e.g., use the same OS distribution) and run the same internal applications over the same data (i.e., same label mappings), then these clients satisfy label equivalence.

4.4 Flow Equivalence

While label equivalence abstracts hosts from their specific network connections, hosts have to enforce exactly the same security requirements (label mappings) against exactly the same threats (input labels). However, in many cases, the same programs may be used for different deployments, where we know that may face threats at the same input location, but the exact nature of the threat may vary (i.e., input label may differ). In addition, the exact security requirements that a program may need to enforce may also vary although the program must still defend itself against threats from the same paths (i.e., label mapping may differ). In this section, we show that HIFMs can be equivalent even if the specific label mappings do not match; instead, only the sources and sinks of information flow errors must match.

The intuition is that external threats are received at input nodes, and the question is simply whether two hosts have information flow errors at the same sinks, regardless of the specific labels mapped to input nodes or sinks. If so, the hosts' HIFMs still have information flow errors along the same path (see Definition 1, item 4).

Conceptually, the key insight is that information flow errors are not borne of specific labels, but of the paths that lead to errors. If the corresponding paths lead to errors, then corresponding mediator placements will resolve those errors. The actual label of the mediator can be determined later. Based on this insight, we define *flow equivalence* between HIFMs.

Definition 5. *Two HIFMs $\mathcal{I}_1 = (G_1, \mathcal{L}, M_1)$ and $\mathcal{I}_2 = (G_2, \mathcal{L}, M_2)$ are said to satisfy* flow equivalence $\mathcal{I}_1 \equiv_f \mathcal{I}_2$ *if:*

1. *$\mathcal{I}_1$ and $\mathcal{I}_2$ satisfy data-flow equivalence.*
2. **Corresponding info flow errors**: *If node $u_1 \in V_1$ is a source of an information flow error at $v_1 \in V_1$ if and only if the corresponding node $f(u_1) = u_2 \in V_2$ is a source of an information flow error at the corresponding node $f(v_1) = v_2 \in V_2$.*

Flow equivalence is the first equivalence class for which some non-trivial computation is necessary to validate equivalence. In theory, computing all the sources of constraint violations could be expensive[1], but there are several factors that mitigate this expense. First, we only need to focus on input nodes as sources, as other errors could be computed in advance. In practice, input nodes form a small fraction of the number of nodes in a host's data-flow graph. Second, we compute the paths after the host graphs are summarized, which already eliminated spurious paths [37].

Example: Suppose two web servers implement two different web applications on behalf of their clients. The labels of the application data are l_1 and l_2, respectively. Note that the web servers must protect their application data from untrusted clients, whose label c is below (recall that we are focusing on integrity) that of the application data $c < l_1$ and $c < l_2$. If the two web servers are deployed using the same OS distribution and web server, the untrusted clients will submit requests through corresponding input nodes in the two servers (sources), leading to information flow errors (as $c < l_1$ and $c < l_2$) in the corresponding web applications

[1] Information flow errors can be detected in linear time in the worst case [31], but identifying all sources that may cause all errors is $O(|V| * |E|)$ in worst-case.

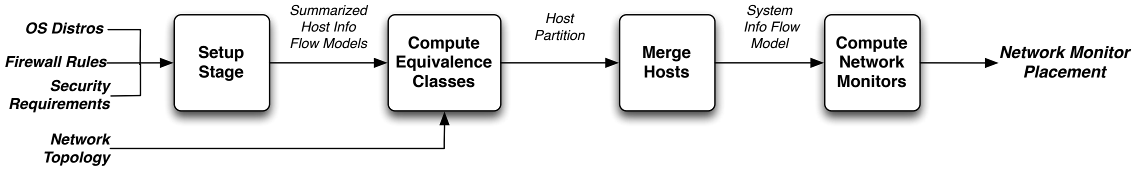

Figure 2: Network Monitor Placement Method

(sinks). Since such information flow errors in both servers can be resolved anywhere along the corresponding paths, the corresponding mediator placement can resolve both errors. That is, if the merged HIFM mediator is $((u, v)l)$ this can be mapped to respective mediators $((u_1, v_1)l_1)$ and $((u_2, v_2)l_2)$ for the web servers. Thus for networks with diverse configurations, we can still perform merging, since the flow equivalence is not based on the concrete connections and labels. Our method boils down the minimal required constraint for information flow error equivalence. In hosts where we don't have information flow error equivalence, merging is not feasible in those cases and have to be solved separately.

Reduction in mediation placement problem size. One key question is what the impact of flow equivalence is on the size of the mediation placement problem. In theory, the size of the merged network is the same as the number of flow-equivalence classes of hosts, as each host in an equivalence class can be represented by a single HIFM. In practice, flow equivalence dictate that any hosts with the same MAC policy (OS distribution), firewall policy, and target application(s) has the potential of being merged. Thus, as described above, all web servers hosted on the same OS distribution are candidates for equivalence.

4.5 Leveraging Network Mediation

Merging individual HIFMs based on flow equivalence provides the potential for the greatest reduction of the three relations, but it does not take into account the protection that can be provided by the network nodes on paths to hosts. Suppose that a network node provides the data to a set of web server hosts, if the web servers all satisfy flow equivalence, then a mediator may be placed in each of the web server hosts at the corresponding edges. However, since the network node is on the data flow path to the web server hosts, we could place a single mediator at the network node obviating the need for the per host mediation. A naive application of flow equivalence will lead to a larger number of mediators by not utilizing network nodes effectively.

The fundamental problem is that flow equivalence enables us to merge a set of hosts that are in different networks. If one of the hosts is in a network that lacks the ability to do network mediation, then mediators will be placed inside the merged host and be applied to *all* hosts represented by the merged host, including those hosts that have network nodes capable of the required mediation. As a result, we will produce a worse solution than we would without the merging.

Fortunately, there is a simple solution to this problem. Instead of merging based solely on hosts, we can include the nodes for the network devices that may provide mediation in the HIFM's data flow graph. Thus, if the network deployments differ in their mediation, then the hosts will not be merged, enabling utilization of network mediation for all hosts where it is possible.

Of course, the network nodes must be capable of performing whatever mediation is required. In practice, a network node must be capable of changing the security requirements of the network data to satisfy the needs of the sink. As such security requirements are expressed in terms of labels, likewise mediation capabilities can be defined in terms of the labels that can be achieved by nodes. In this work, we associate each network node and host with a lattice label defining the LUB of mediation, called the *capability label*, where each capability label is in the set of lattice labels of the system information flow model. In theory, the node can mediate to any label dominated by its capability label.

5. PLACING NETWORK MONITORS

In this section, we discuss how to use the equivalence relations defined above to compute a network monitor placement from host information flow models and network topologies for the system. Figure 2 shows the steps in our proposed method. First, we describe a setup stage that produces the host information flow models from security policies and network topologies, using an existing method to produce summaries. Second, we partition the set of summarized HIFMs into equivalence classes using the three types of equivalence relations from the previous section. Third, given a set of equivalence classes and the network topologies for the system, we produce a merged system information flow model. Fourth, we leverage known methods for solving the mediator placement problem for the merged system information flow graph.

5.1 Setup Stage: Host Summaries

Our method produces network monitor placements from host information flow models and network topologies. However, since hosts are not configured directly as host information flow models, these have to be produced. Fortunately, researchers have developed several automated methods for computing elements of the information flow model. While some manual configuration of host information flow models may still be required, the task can be significantly reduced. In addition, host information flow models of modern OS distributions may be quite complex themselves, so we leverage previously proposed methods for producing summarized host information flow models, which we call *host summaries.*

Host information flow models consist of a data flow graph, lattice, and label mapping function as specified in Definition 1, but automated techniques are available to generate each of the above elements. First, modern OS distributions now provide pre-configured software packages, host firewall policies, and mandatory access control (MAC) policies from which data flow graphs can be constructed[2]. One issue is connecting the data flows between network nodes and the host processes, but some OS distributions (e.g., RedHat) leverage labeled networking [10] (e.g., Secmark [22]) and researchers have explored methods to relate access control policies to system call sites [15]. Network topologies express flows among network nodes.

Second, instead of using lattices to express traditional multilevel security policies [5, 6], we envision using lattices to represent security requirements as sets of labels, as in Decentralized Information Flow Control [14] (DIFC). Security requirements are predicates on nodes that must be satisfied to prevent compromise. For example, one security requirement would be a limit for the number of allowed entries in an HTTP Range query at the web server. Such a requirement can be encoded as a label, where only data satisfying that requirement may be assigned that label. Our approach is agnostic to the source of security requirements. Some requirements may be derived from known vulnerabilities and others from software testing.

In general, all web servers may want to enforce the HTTP Range requirement highlighted above, so that label can be mapped to any web server in any deployment. Thus, we can automatically assign this label mapping to any host targeted as a web server deployment. However, some security requirements may be deployment-specific. For example, many organizations deploy their own custom software on OS distributions, such as web applications. Fortunately, MAC policies support such customizations. For example, system administrators use the `mod_selinux` module for Apache to generate separate web application processes with distinct permissions. However, the system administrators (or web application developers) will have to assign specific labels for their web application, if the web application has any special data requirements. This is the main manual effort in setup.

Finally, researchers have found that host information flow models themselves can be large, with thousands of nodes and edges. This observation inspired Talele *et al.* to produce *host summaries* that retain only the nodes and edges necessary to preserve the attack path semantics of the original host information flow model [37]. Such summarization is analogous to building function summaries for static program analysis [7]. For some server host configurations, they found that they could reduce the number of nodes by 65-80% and the number of edges by approximately 85%.

5.2 Compute Equivalence Classes

Given the HIFM summaries computed above, we aim to partition these summaries into equivalence classes using the equivalence relations defined in Section 4 and then merge the equivalent hosts. The challenge is that not all the equivalence classes are the same from a merging perspective. Concrete equivalence requires the fewest graph changes, followed by label equivalence, and finally flow equivalence. Thus, we want to design a method that prefers concrete equivalence to others, where possible, but still enables subsequent merging using label and flow equivalence.

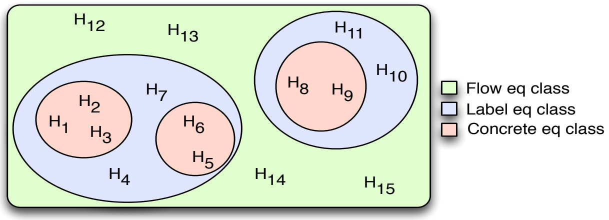

Figure 3: Equivalence Class Dominance: Flow Equivalence creates classes that are a superset of Label Equivalence, which in turn creates classes that are a superset of Concrete Equivalence

We find that the three proposed equivalence relations satisfy a set-dominance relation themselves.

Definition 6. *If two hosts information flow models $\mathcal{I}_1$ and $\mathcal{I}_2$ are* concrete-equivalent $\mathcal{I}_1 \equiv_c \mathcal{I}_2$ *(i.e., they belong to the same concrete equivalence class), then they are also* label-equivalent $\mathcal{I}_1 \equiv_l \mathcal{I}_2$ *and* flow-equivalent $\mathcal{I}_1 \equiv_f \mathcal{I}_2$. *Similarly, if two hosts are label-equivalent then they are also flow-equivalent.*

By definition concrete equivalence implies that equivalent hosts have equivalent information flow models and connect to the same external nodes (input and output). As a result, they are guaranteed to receive input data of the same label, which along with the equivalent information flow models satisfies label equivalence. Further, hosts that satisfy label equivalence must violate constraints at the same sinks since they have the same input labels, label mapping functions, and lattice. Also, the corresponding sources will contribute atoms (data of offending labels) that violate constraints at those sinks for the same reason. Since label-equivalent hosts also have equivalent host information flow graphs, they satisfy flow equivalence as well. Concrete-equivalent are also flow-equivalent as can be seen.

The Venn diagram shown in Figure 3 demonstrates this subsumption relationship among three equivalence classes. As Section 5.3 shows, flow equivalence is the most expensive case to merge, so this subsumption relation is helpful because we can merge concrete and label cases to reduce the cost associated with merging for flow equivalence. As a result, our method checks for concrete equivalence, followed by label, and lastly flow equivalence. Also, we avoid checking for equivalence for obviously distinct cases, such as those hosts with different OS distributions and different applications with label mappings.

5.3 Merge Hosts

Merge operation uses the equivalence classes produced in previous section and leverages the hierarchy of equivalence properties to execute the merge. Figure 4 shows a method for merging the summarized HIFMs (simply HIFMs in this section) for the three equivalence properties. We merge from finest (concrete) to coarsest (flow) equivalence classes.

First, concrete equivalence classes consist of HIFMs that have the same input and output links, so merging these HIFMs is straightforward. We produce one representative HIFM for each class and eliminate the rest. The solution produced for the merged HIFM will be applicable to all members of the equivalence class. In Figure 3, we see that

[2] The typical method is to create a node for each subject and object and edges as follows: for each authorized read-like operation by subject u upon object v create edge (v, u) and for each authorized write-like operation by subject u upon object v create edge (u, v).

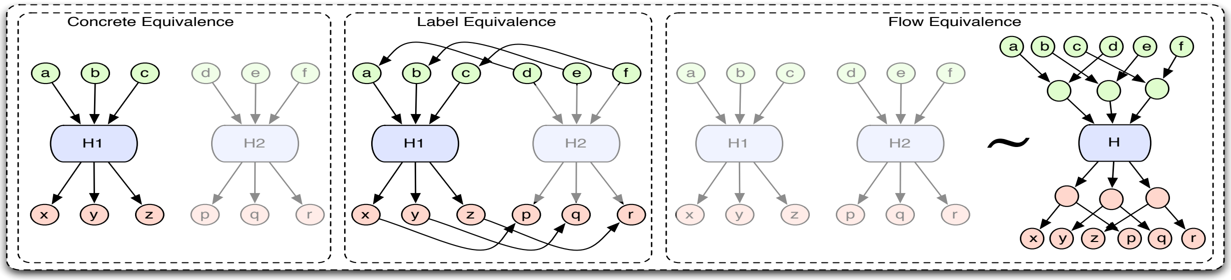

Figure 4: Host Merging for Concrete, Label, and Flow Equivalence (left to right)

HIFMs H_1, H_2 and H_3 can be represented using H_1, thus we keep one HIFM and discard the others. If we have two HIFMs that satisfy only concrete input equivalence, we union the output edges (not shown).

Figure 4 shows the merging operation performed for label-equivalent HIFMs. For label equivalence, we again create one merged HIFM for each equivalence class, but we have to address the problem that the network links are different. Since the HIFMs all receive data of the same label, it is sufficient to union the I/O edges of the individual HIFMs into the merged HIFM. Again referring to Figure 3, we have H_1 and H_5 as representatives of two concrete equivalence classes contained in the same label equivalence class with two additional HIFMs H_4 and H_7. We then perform the merge operation on these four HIFMs to create one merged HIFM. The I/O edges for all four HIFMs are unioned and are added to the corresponding nodes of the merged HIFM in this case.

We merge HIFMs of the same flow equivalence classes as represented in Figure 4. If HIFMs are flow-equivalent, then they have the same data-flow graphs and hence each of the HIFMs can be represented using any one of the HIFM's data-flow graphs. However, since each HIFM's input nodes may expect data of different labels, we augment the data-flow graph with a second layer of dummy input nodes for each unique input label, which we call *constraint nodes* because they require satisfaction of a label mapping. Each input edge to one of the merged HIFMs is connected to one of the new input constraint nodes instead, enabling detection of information flow errors should the input not comply with the expected constraints from their label mappings.

A similar approach is taken to handle the output nodes and their connections[3] as represented in Figure 4. In this case, we create an output constraint node for each combination of corresponding output node and expected label for that node to transmit output of an expected output label along the output edges. In order to create these output constraint nodes, we need to predict the expected output labels correctly. Normally, this is not a problem, as the target application is responsible for most outputs and must obey specific security requirements (i.e., label mappings). However, some data may simply "flow through" the HIFM, so we cannot predict the label associated to that data. We remove HIFMs from the merge if we cannot predict the labels of the data at all output nodes.

We then have to construct the label mapping function for the merged HIFM based on the HIFMs that satisfy flow equivalence. Since flow equivalence requires that all the label mappings for each of the HIFMs merged result in the same error paths, we can use any one HIFM as a template to produce the merged HIFM's label mapping function. Thus, we create a dummy lattice corresponding to the labels used in one HIFM and their information flow relationships and map those to the corresponding nodes in the equivalent HIFMs. Input and output constraint nodes "translate" between the dummy labels and the actual labels to maintain the correct input and output information flows.

5.4 Compute Network Monitor Placement

In the last step, we compute a network monitor placement that satisfies the system's security requirements. This resultant network monitors may be chosen to enforce secrecy, integrity, and/or availability requirements. Finally, the security requirements must be converted into equivalent rules for the network monitors to enforce those requirements.

We compute network monitor placements by solving the mediator placement problem for the system's information flow model. Recall from Section 3 that the mediator placement problem can be formulated as a graph problem. We solve the directed multicut problem [11] using a greedy algorithm that unions solutions to individual cut problems.

Since security requirements are simply sets of individual requirements, we can represent requirements for integrity, secrecy, and availability independently. In general, such requirements may not be orthogonal, however. An integrity requirement to filter data may cause a denial of service. In order to prevent conflicts, we separate the security requirements into those that are known unsafe (to block) or not known to be safe (to log). Thus, if the integrity requirement above is for a known unsafe case, then blocking it denies an adversary and prevents a likely compromise.

6. EVALUATION

In this section we aim to evaluate the two claims made in this paper. First, we examine how the concrete, label, and flow equivalence relations enable reductions in the size of the information flow model. We find that one host information flow model (HIFM) per distinct host configuration can represent large systems, thus considerably reducing the sizes of mediator placement problems. Second, we examine the variation in the cost of computing mediator placement while keeping the number of equivalence class constant but increasing the host count. We observe that the results substantiate our claim that the size of a mediator placement

[3] Note that we must precompute the label on each output constraint node in order to propagate data of the expected label as the original host would have.

Table 1: Example Network from Figure 1: Network types: ”wired”=regular wired office network with routers and switches; ”wireless”=wireless network with access point acting as monitor; ”adhoc”=network with no specific access point. Network protected values: ”yes”= network devices can mediate to any label; ”limited capability”=can only mediate some errors; ”no”=no mediation on the network device.

Network	Client	Admin Client	Web Server 1	Web Server 2	DB Server	DNS Server	Network Type	Network Devices	Network Protected	Total
Trusted Clients	400	400	100	-	-	1	wired	router, IDPS, firewall	yes	901
Server Farm	-	100	400	400	-	1	wired	router, IDPS, firewall	yes	901
Data Center	-	-	100	100	600	1	wired	router, IDPS, firewall	yes	801
Partner Clients	150	150	-	-	-	1	wired	router, firewall	limited capability	301
Remote Clients	300	100	-	-	-	1	wireless	router, access point	limited capability	401
Adhoc Network	300	-	-	-	-	-	ad hoc	network connected host	no	300
Total	1150	750	600	500	600	5				3605

problem depends on the number of distinct HIFMs and not the number of hosts.

We perform the evaluation on the network shown in Figure 1, which is described elsewhere [25] and covers several aspects of a typical corporate network. The details of the different host configurations and network properties for this experiment network are provided in Table 1. The columns specify distinct host data flow configurations representing different applications, the network architectures, the network devices in each network, and the mediation capability for the available network devices. Each host system enforces SELinux MAC policies [28]. The networks also include other servers such as DHCP and DNS servers. The network communication for each host is defined by the network topology and the firewall iptable rules enforced in the hosts and network.

6.1 Information Flow Model Merging Results

Table 2 shows the experimental results of merging in four organizational networks with variations in the unique host configurations and the network configuration. The fourth row describes the example network detailed in Table 1. The first column in Table 2 shows the total number of hosts in the sample network, followed by how many unique host data flows (MAC/firewall policies/applications) are given for each host. Each host can have multiple label mapping functions represented in third column. These configurations of unique host data flows and their various mapping functions generate unique host configurations identified in the next column. The distribution of these unique host configurations among the number of subnets forms the basis of the merging capability of the system.

The columns for concrete and label equivalence classes in Table 2 show the equivalence classes computed in each of the subnets of the system and then the number of classes for the whole system. The label equivalence classes will always be greater than or similar to the unique host configurations in the system. The next column shows the flow equivalence classes computed across all the networks in the system. The flow equivalence is independent of the actual labels mapped to the hosts as discussed earlier, which enables compression to lower number of equivalence classes than the unique host configurations. We discussed in Section 4.5 that mediator placement solutions may be degraded when the network mediation capability is not considered. The final column shows the merging possible when accounting for network mediation capabilities using flow equivalence. As expected, the number of equivalence class increases, but the number of mediators required decreases, as the table shows.

Table 3 shows an example of the impact of merging on the total number of nodes and edges in a system-wide information flow model for the network detailed in Table 1. We see a reduction of three orders of magnitude, even relative to the summarized hosts [37], using the proposed method, reducing the number of nodes from millions to approximately 3500. As the graph cut method has a worst-case cost $O(|E|f)$, where f is the maximum flow in the graph, such a reduction will have a significant impact on compute time.

Table 3: Reduction in the Data Flow Graph

	Whole Network	Summarized	Merged hosts
Nodes	9 million	1.5 million	3540
Edges	19 million	3.8 million	20819

By computing mediator placement solutions for each merged system we show that flow-equivalent merges that include the network devices capable of mediation reduce the size of the placement solution. For instance, the network in the fourth row in Table 2 with 18 flow-equivalent hosts results in a solution requiring 3213 host mediators using the method described in Section 5.4. The mediator placement computed from the 22 hosts including network mediation results in only 2709 host mediators and 4 network mediators. Thus the 4 network nodes can reduce the host mediation necessary by over 100 mediators per network mediator.

6.2 Performance Analysis

Table 4 shows the compute times for each step in the process of computing monitor placement for a network system. The experiments were performed on a 2.80GHz intel dual core machine running Linux kernel 2.6.31. The first two columns in the table show the average time required for computing the data flow graph and the summaries for one

Table 2: Equivalence Analysis Results: *-300 ad hoc network hosts

Host Count	Unique Data Flows	Unique Mappings	Unique Host Configs	Subnets in System	Concrete Eq (Hosts per Subnet/Sys)	Label Eq (Hosts per Subnet/Sys)	Flow Eq (Hosts per System)	Host Mediators	Flow Eq+Net (Hosts per System)	Host+Network Mediators
3600	5	2	11	5	2.8/14	2.6/13	9	1644	11	1319+3
6000	5	4	15	5	5.2/26	3.6/18	13	2326	14	1768+3
9500	5	30	120	5	26/130	24.3/121	112	20416	118	16718+3
3600	5	6	21	6	53.3/320*	4/24	18	3213	22	2709+4

Table 4: Compute Times for the Method Steps

Network Size	Build host model per host (sec)	Summarize host per host (sec)	Compute Equivalence (min)	Merge hosts (min)	Compute Placement (min)
3600	3.5	25	5.03	3.43	1.25
6000	3.5	25	11.45	6.43	1.47
9500	3.5	25	21.36	12.27	2.06

host, these steps can be performed independently of each other and hence can be parallelized easily. The computation of the equivalence classes and merge operation is performed on the entire information flow model using flow-equivalence accounting for network mediation. The computation has a worst case complexity of $O(n^2)$ where n is the number of total hosts in the system. The computation for data flow equivalence is optimized in cases where the policy applied to the host is known to be same. Once the equivalence classes have been identified, the processing of each HIFM can be done independently from the others and then merged together. Computing mediator placements for an individual HIFM given the host summary takes about 4-6 seconds on average.

7. RELATED WORK

The research in the network and system security has mostly followed a parallel path in finding and fixing the security vulnerabilities. We have various policy based system security approaches [28, 37] which address the security requirement in the system. Network security typically consists of placing the network intrusion detection systems [12, 36] and efficiently tracking the traffic between hosts. These methods do not take into account internal hosts processes to identify the actual nature of data being transmitted. The attack graph [33, 29] and attack tree [21] approaches bring the network and the host states closer to identify an exploit, but they rely on previous knowledge of the vulnerability. As the size of the system increases the approach faces the problem of state space explosion and soon becomes intractable. There has been much work done in the area of performing efficient attack graph based analysis [27, 1, 40, 26, 2]. These techniques are mostly based on various heuristics and knowledge of previous attacks to determine the attack behavior. The work in [40] does vulnerability analysis based on the topology, temporal property and received alerts to predict possible future alerts. The method used in [1] is an extension of above method and employs a temporal abstraction of the attack graph to determine relevant sequences in order to perform scalable detection. The work in [26] also employs forensic analysis of attack strategies in order to predict and defend future attacks. Rather than basing the detection on earlier attack strategies and heuristics we proactively model the host data flow to block all possible attack paths by providing complete mediation. There have been efforts to assure complete mediation while identifying the optimal placement using classical approaches like vertex cover [27] and graph cut problems [30, 17]. These efforts have been mostly either in the context of host or network mediation, but not for both.

Another work on network reduction [39] for the purpose of efficient analysis, is based on reducing the number of network nodes by unifying them such that the key network protocol correctness properties are not violated. The work takes an Border Gateway Protocol (BGP) instance of the protocol and utilizes the Stable Paths Problem(SPP) to identify the nodes that can be unified to reduce the network size prior to performing analysis for anomaly diagnosis. The method displays a similar idea on reduction of the graph for analysis and is specific to network and does not deal with host data flow connections.

This work is closer to the host data flow summarization method done in [37], where they show that summaries maintain the fine grained data flow properties needed for complete mediation while making it more efficient to analyze. Though the method performed some reduction at host level for efficient analysis, it was not able to handle more than hundred hosts. We leverage their work further to address the redundancy across the network and achieve further summarization and are able to address thousands of hosts efficiently.

8. CONCLUSION

In this paper we have successfully introduced the method to model large network systems in scalable manner to enable information flow analysis. We presented three key equivalence concepts that enable us to preserve the information flow error paths in the reduced system model. The model can then be analyzed for security errors and the placement solution thus obtained can solve the security errors in the entire systems. This work considers the fine-grained flow properties in every host while scaling the analysis to huge corporate networks. The results show that this method can achieve substantial reduction in the system graphs where such reduction depends on the amount of *redundancy among host configurations and network topologies* rather than the actual number of hosts and network flows. We demonstrate how near-optimal and efficient network monitor placement can be done considering the host flows for typical configurations of large corporate networks.

9. ACKNOWLEDGMENTS

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

10. REFERENCES

[1] M. Albanese, S. Jajodia, A. Pugliese, and V. S. Subrahmanian. Scalable detection of cyber attacks. In N. Chaki and A. Cortesi, editors, *CISIM*, volume 245 of *Communications in Computer and Information Science*, pages 9–18. Springer, 2011.

[2] H. M. J. Almohri, D. Yao, L. T. Watson, and X. Ou. Security optimization of dynamic networks with probabilistic graph modeling and linear programming. Technical report, Virginia Tech, 2014.

[3] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, pages 71–82, New York, NY, USA, 2002. ACM.

[4] M. Bauer. Paranoid penguin: An introduction to novell apparmor. *Linux J.*, 2006(148):13–, Aug. 2006.

[5] D. E. Bell and L. J. LaPadula. Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report ESD-TR-75-306, Deputy for Command and Management Systems, HQ Electronic Systems Division (AFSC), March 1976.

[6] K. J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, MITRE, April 1977.

[7] A. J. Bik and H. A. Wijshoff. Implementation of fourier-motzkin elimination. In *Proceedings of the first annual Conference of the ASCI*, pages 377–386. Citeseer, 1994.

[8] H. Chen, N. Li, and Z. Mao. Analyzing and comparing the protection quality of security enhanced operating systems. In *NDSS*, 2009.

[9] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet security: repelling the wily hacker*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[10] Introduction to labeled Networking in Linux. `http://www.linuxfoundation.jp/jp_uploads/seminar20080709/paul_moore-r1.pdf`.

[11] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23:864–894, August 1994.

[12] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[13] S. Gulwani and A. Tiwari. Computing procedure summaries for interprocedural analysis. In *ESOP*, 2007.

[14] W. R. Harris, S. Jha, and T. Reps. Difc programs by automatic instrumentation. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 284–296. ACM, 2010.

[15] M. Howard, J. Pincus, and J. Wing. Measuring relative attack surfaces. In *Computer Security in the 21st Century*, pages 109–137. 2005.

[16] T. Jaeger, R. Sailer, and X. Zhang. Analyzing integrity protection in the SELinux example policy. In *USENIX Security Symposium*, Aug. 2003.

[17] D. King, S. Jha, T. Jaeger, S. Jha, and S. A. Seshia. Towards automated security mediation placement. Technical Report NAS-TR-0100-2008, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, November 2008.

[18] D. King, S. Jha, D. Muthukumaran, T. Jaeger, S. Jha, and S. A. Seshia. Automating security mediation placement. In A. D. Gordon, editor, *ESOP*, volume 6012 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2010.

[19] M. N. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information flow control for standard OS abstractions. In *Proceedings of the 21st ACM Symposium on Operating Systems Pr inciples*, pages 321–334, Oct. 2007.

[20] B. Livshits and S. Chong. Towards fully automatic placement of security sanitizers and declassifiers. In *Proceedings of the 40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 385–398, New York, NY, USA, Jan. 2013. ACM Press.

[21] S. Mauw and M. Oostdijk. Foundations of attack trees. In *International Conference on Information Security and Cryptology, ICISC 2005. LNCS 3935*, pages 186–198. Springer, 2005.

[22] J. Morris. New Secmark-based network controls for SELinux. `http://james-morris.livejournal.com/11010.html`.

[23] D. Muthukumaran, S. Rueda, N. Talele, H. Vijayakumar, T. Jaeger, J. Teutsch, and N. Edwards. Transforming commodity security policies to enforce Clark-Wilson integrity. In *ACSAC*, 2012.

[24] A. C. Myers and B. Liskov. A decentralized model for information flow control. *ACM Operating Systems Review*, 31(5):129–142, Oct. 1997.

[25] Network based intrusion detection configuration. `http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/ServerFarmSec_2.1/8_NIDS.pdf`.

[26] P. Ning and D. Xu. Learning attack strategies from intrusion alerts. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 200–209, New York, NY, USA, 2003. ACM.

[27] S. Noel and S. Jajodia. Advanced vulnerability analysis and intrusion detection through predictive attack graphs. In *Critical Issues in C4I, Armed Forces Communications and Electronics Association (AFCEA) Solutions Series.* International Journal of Command and Control, 2009.

[28] Security-enhanced linux. `http://www.nsa.gov/research/selinux/`.

[29] X. Ou, W. F. Boyer, and M. A. McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM Conference on Computer and*

Communications Security, pages 336–345, New York, NY, USA, 2006. ACM.

[30] L. Pike. Post-hoc separation policy analysis with graph algorithms. In *Workshop on Foundations of Computer Security (FCS'09). Affiliated with Logic in Computer Science (LICS)*, August 2009.

[31] J. Rehof and T. A. Mogensen. Tractable constraints in finite semilattices. *Sci. Comput. Program.*, 35(2-3):191–221, 1999.

[32] B. Sarna-Starosta and S. D. Stoller. Policy analysis for Security-Enhanced Linux. In *WITS*, April 2004.

[33] O. Sheyner, J. W. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *IEEE Symposium on Security and Privacy*, pages 273–284, 2002.

[34] Snort Intrusion Detection/Prevention System. `http://www.snort.org/`.

[35] Sun Microsystems. Trusted Solaris operating environment - a technical overview. `http://www.sun.com`.

[36] Suricata Intrusion Detection/Prevention System. `http://suricata-ids.org/`.

[37] N. Talele, J. Teutsch, T. Jaeger, and R. F. Erbacher. Using security policies to automate placement of network intrusion prevention. In *ESSoS*, pages 17–32, 2013.

[38] Tresys. SETools - Policy Analysis Tools for SELinux. Available at http://oss.tresys.com/projects/setools.

[39] A. Wang, C. L. Talcott, A. J. T. Gurney, B. T. Loo, and A. Scedrov. Reduction-based formal analysis of bgp instances. In C. Flanagan and B. König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 283–298. Springer, 2012.

[40] L. Wang, A. Liu, and S. Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15):2917–2933, Sept. 2006.

[41] R. N. M. Watson. TrustedBSD: Adding trusted operating system features to FreeBSD. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 15–28, 2001.

[42] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in HiStar. In *OSDI*, 2006.