Specification of Network Access Policy and Verification of Compliance Through Passive Monitoring

By

SCOTT ANDREW MILLER

B.S. (Bowling Green State University) 1996

THESIS
Submitted in partial satisfaction of the requirements for the degree of
MASTER OF SCIENCE

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES
of the
UNIVERSITY OF CALIFORNIA
DAVIS

Approved:

Professor Biswanath Mukherjee Co-Chair

Professor Ronald A. Olsson Co-Chair

Professor Matthew A. Bishop

Committee in Charge

1999

Contents

Acknowledgements

Chapter 1

Introduction

Protecting corporate resources has been complicated by the use of inter-networked, off-the-shelf computer systems. Many common off-the-shelf computer systems are delivered with services that have security vulnerabilities (as do many common applications added by users). The severities of these vulnerabilities range from a local user being able to tie up computational resources to an unauthorized remote user being able to obtain complete control over the system. A number of these vulnerabilities are publicized on well known web sites[1], and are often accompanied by programs that exploit them. As such, even a user with moderate skills can obtain the ability to compromise vulnerable systems. In a strongly connected corporate intranet environment, any host that is running a vulnerable service is exposed to compromise by any employee or visitor who has access to the network. In large corporations (which may span continents), the set of people with this amount of access can be frighteningly large and diverse. There has also been a proliferation of network scanning software, which allows users to quickly find all hosts in a network that are providing known vulnerable services. Thus, there is a significant danger that a vulnerable target can be detected by a wrongdoer and an actual compromise can occur. One approach to reducing this risk is the specification and enforcement of an internal network access policy, which is the subject of this research.

A network access policy is a formal statement of how hosts are allowed (and not allowed) to interact with one another. By specifying an internal network access policy, site security officers can make and promulgate formal decisions regarding the restriction of insecure applications and the

[1] Three of the more popular sites are: Rootshell (http://www.rootshell.com), AntiOnline (http://www.antionline.com), and Exploit World (http://www.insecure.org).

limitation of access made available to untrusted hosts. These restrictions on how the network is utilized can reduce the set of potential malefactors with access to resources.

To illustrate the security issues and benefits of an access policy, consider a typical corporate intranet, as shown in Figure 1.1. This corporation has a number of assets contained in its various departments, but for our example the proprietary information that is created and managed by the various research groups raises the most concern. The corporate intranet is strongly connected using the Internet Protocol (IP). The intranet is connected to the Internet, but utilizes a restrictive firewall to limit all outside access to essentially email and outgoing file transfers. With few exceptions, every permanent employee is given a computer with access to the intranet, and a number of IP addresses are left open for visitors and contractors with laptops. A modem bank allows employees to dial in over conventional phone lines, which includes its own user authentication system. In addition, within departments there is a large amount of trust between hosts[2]. All of these properties are commonly present in corporate environments. While the firewall excludes the majority of unsavory Internet users, consider the number of different ways a malefactor could gain access to this intranet. Within the large corporation there is a large number employees given access to the network, some of whom might be willing to compromise the corporation for personal benefit. Anyone with a conventional phone can call into the modem bank. While the modem bank incorporates some authentication, it can still be an open access point for external users, depending on the strength of authentication and diligence of administration in maintaining passwords, deleting old accounts, and training users not to divulge their account information. The frequency of visitors and contractors entering in the various departments may also be a concern.

The proprietary information maintained in the research groups need not be exposed to all of these threats. A network access policy can leverage knowledge of the needs of the research groups to remove access points for malefactors without loss of productivity. For example, the system administrator responsible for Research Group 1 is aware of the real software and network access needs of the group. A few users in Research Group 2 are working together on a project and require the ability to login and transfer files from Group 1. All of the corporate hosts require email and DNS services. Leveraging this knowledge, the administrator could specify a policy that stipulates that Research Group 1 be made entirely inaccessible to all other hosts, with the exception of the DNS and SMTP servers and a few trusted hosts in Research Group 2 that are allowed to use only secure

[2]For example, it is common to use the "r" applications (rsh, rlogin, etc.), which allow users to remotely access a host from any of a listed set of hosts. Frequently, the entire set of hosts within a subnetwork or department will be included on these lists. For this reason, and the extensive use of broadcast networking medium and cleartext authentication, a malefactor who compromises one host in a department can easily gain access to the entire set.

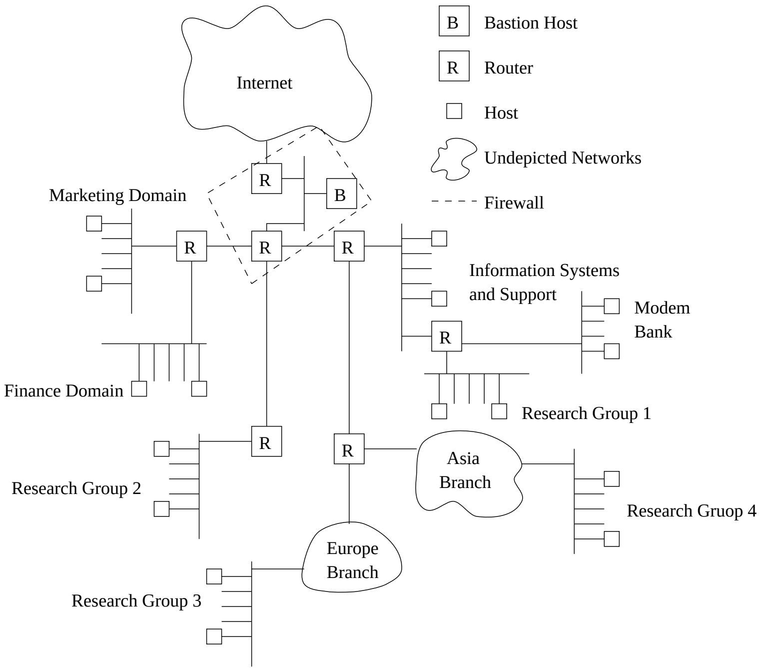

Figure 1.1: Corporate intranet topology.

login and file transfer services. In addition, the majority of hosts in Research Group 1 are personal (client) hosts, and do not need to provide those network services. The administrator can further restrict access to the couple of servers that Research Group 2 requires.

If this policy is enforced, the security of Research Group 2 becomes more manageable for three reasons. First, a large portion of the potential malefactors has been denied direct access to the area. They will first need to compromise one of the trusted hosts, a host in Research Group 2, or the DNS or SMTP servers. Second, with access to the client hosts cut off, a malefactor within Research Group 2 cannot exploit vulnerable applications that may have been shipped with these systems or were installed (possibly unwittingly) by their users. Finally, by limiting the network services provided to external hosts, the administrator reduces the amount of danger associated with an oversight in the work required to maintain the security of all of the services (following reports of newly discovered vulnerabilities, applying patches, etc.). The detriment of enforcing this policy is that users will be prevented from behaving outside of the imposed boundaries. There may be some forms of useful activities of which the administrator was unaware when specifying the policy. The issues of utilizing network access policies include: the need for feedback regarding potential misconfiguration, the lack of authentication in IP, and the complexity of network usage. Currently, there is no easy to use mechanism for specifying and enforcing intranet access policies. In this investigation, we will present methods and mechanisms that aid in accomplishing this task beyond the ability of currently available tools.

1.1 Network Monitoring

A network monitor is a computer system that is capable of observing the communications between other computer systems in the network. A number of network monitoring techniques have been adopted to increase the security of corporate intranets. General network monitoring tools allow an administrator to easily summarize and view network traffic[3]. Network intrusion detection systems automate the process of discovering miscreants.

Network intrusion detection systems are categorized as employing one of two types of approaches: anomaly detection and misuse detection[4]. An example anomaly detection system is Network Security Monitor (NSM) [10]. NSM utilizes passive network monitoring and statistical methods to profile the normal behavior of the network. To detect intrusions, NSM then passively monitors the network looking for traffic that significantly deviates from the profile. The assumption is that intrusive behavior is likely to be dissimilar to typical, legitimate network usage. Current systems employing anomaly detection tend to generate many false positives and reports are often difficult to interpret. The latter approach utilizes signatures to detect specific attacks. The strength of the misuse detection approach is a low false-positive rate. The weakness is that they are inherently limited to classes of attacks that are known, while new vulnerabilities in common computer systems are constantly being discovered [12]. In addition, systems that rely on detecting specific patterns in the data portion of network packets are vulnerable to insertion and evasion tactics [16]. Some examples of network misuse detection systems are DIDS [18] and GrIDS [19]. The approach that seems to be the most relevant to our work is GrIDS. By utilizing a hierarchy of monitors and abstracting data that is passed up the hierarchy, GrIDS is able to detect wide-spread attacks across large networks while remaining scalable. GrIDS was shown to behave well in environments where traffic was generally compartmentalized. In environments with a large amount of cross-domain traffic, GrIDS may cause some detriment to network performance.

General network monitoring tools such as Network Flight Recorder [17] and SNIF [1] have

[3]We will use the terms communications and traffic to refer to any instances of communication between hosts through a network.

[4]Their predecessor, host-based intrusion detection systems employ similar techniques and are also categorized as such [7, 13, 11].

been used by system administrators for security purposes. These systems provide a useful interface to network traffic and perform some forms of data summarization. Many of them provide a simple, rule-based language for specifying conditions under which an alert should be generated. This mechanism could be used to specify and validate a network policy, but these systems lack a systematic way of promulgating such a policy and ensuring coverage.

1.2 Firewalls

Firewalls have become one of the most popular corporate network security mechanisms. Numerous publications discuss the various approaches, including the well known compendiums [4], and [3]. A number of commercial implementations have become widely available. Here we present an overview of current firewall approaches.

The basic functionality of a firewall is to enforce an access policy between two networks. Firewalls are typically categorized into two types: filtering routers and network proxies, but many implementations provide the functionality of both types. The most simple firewall is a filtering router. This device acts as a router, but is given the additional ability to drop packets based upon a set of specified rules. Each rule is a tuple with the attributes: direction, source, destination, source port, destination port, protocol, and action (allow or deny). When a packet traverses the firewall, the rules are checked sequentially. The first rule that matches the packet's direction, source, destination, source port, destination port, and protocol has its action effected. This dependence on ordering to resolve conflicts can lead to problems. A conflict may be a result of a misconfiguration, in which case it ought to be reported rather than handled silently. If the conflict is intentional, i.e., the administrator knowingly specifies overlapping allow and deny sets, then the policy may be difficult to read. For example, consider a large network policy that is managed by a number of individuals. The overlapping rules may be far apart within the specification file; i.e., the rules do not appear together within the specification editor. As such, interpreting the code of the later rules will be misread by a user if they are unaware of the preceding rules. In our approach, we show how a language based on set notation can better handle this situation.

Current implementations of firewalls have embellished this basic design[5]. Some eliminate the port and protocol attributes and instead use a service attribute to summarize those attributes. In addition, most implementations now incorporate two filtering routers and a bastion host be-

[5] At least one commercial firewall implementation [6], has incorporated some simple network intrusion detection techniques (misuse detection in the given reference).

tween them (as depicted in Figure 1.1). The purpose of the bastion host is to act as a proxy for allowed applications. The bastion host actually processes the packets and interprets the data at the application layer, which provides additional assurance that the application is not being misused. The two filtering routers typically enforce policies that ensure that all communication must pass through the bastion host, and only over the well-known ports of the applications that the bastion host is proxying. Typically, a firewall is used to create a secured perimeter around a trusted area (the inside). There are no constraints applied to the inside area. As portrayed in Figure 1.1, it is common for the entire corporate intranet to be considered as inside. Using firewalls to enforce an intranet-wide policy, as opposed to one regarding a single interface, is a difficult task. While some current implementations provide remote administration and centralized policy specification, the local policies for each firewall must be specified independently. One problem with using current firewall implementations to enforce an intranet access policy is address spoofing, i.e., where one host pretends to be another by forging the IP address on packets that it sends. Figure 1.2 depicts a configuration in which a host h can bypass the security constraints of the firewall F to access the host d by spoofing the host s. Since F cannot distinguish packets sent by h from packets sent by s, it cannot prevent the packets sent by h to reach d. Considering that similar configurations would occur frequently in the corporate network, the use of internal firewalls would be rather susceptible to address spoofing. The severity of this problem varies depending on the application involved. Some applications require a full connection (e.g., telnet) to be established (making them difficult to utilize without seeing the returning packets), while others (e.g., Network File System) can be exploited with a single packet. The problem can be alleviated with additional firewalls. For the example in Figure 1.2, a firewall placed between the router and the domain with h could block all packets emerging from that domain with a source address of a host not contained in that domain. Such packets are considered intrinsically illegitimate as they could not have feasibly been routed through the firewall by legitimate means. Such a firewall configuration would currently need to be done by hand. We provide an algorithm to automate this process.

Our work adopts some of the mechanisms employed by firewalls and alleviates some of the problems. In our approach, we incorporate the summarization of services, as well as provide the detailed port attributes. In addition to providing centralized administration, we present algorithms that determine if the configuration will enforce the entire policy, even if malefactors attempt to spoof the addresses of privileged hosts. Our approach also provides constructs that enable multiple users to more easily work together in specifying the policy.

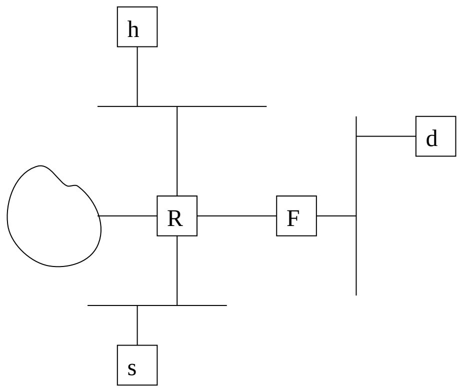

Figure 1.2: The firewall F cannot distinguish packets sent by s from those sent by h.

1.3 Guttman's Approach to Network Access Policies

An approach was recently described in [9] that utilizes firewalls to enforce a global policy. The contributions of that work were a language which allowed users to specify network access policies without worrying about the various paths that packets could take and algorithms for determining local (firewall) policies that enforce the global policy and prevent some amount of spoofing. A formal analysis of the algorithms was not provided in [9], but the results of an empirical study were given which showed that the algorithms ran in a reasonable amount of time (on the order of tens of seconds) for corporations with few distinct areas; the largest graph presented in [9] contained 11 areas.

Some distinctions between the work in [9] and our work are the following:

- Our network model incorporates routing information. We show how this information can be utilized to guarantee that no host can violate a stipulated policy without being detected. This includes when one host spoofs another host. The methods presented in [9] utilize a more general network model and can only prevent a small subset of policy violation through address spoofing.

- Our model of network connections provides a more generic, expandable framework. Where the model presented in [9] is tied to the attributes of a packet filtering firewall, the framework presented in this work can utilize most any type of attribute describing network connections.

- The work in [9] provides a method for validating arbitrary firewall configurations, which we show is NP-hard in our general model.

- We provide polynomial methods for validating that a configuration of policy enforcement mechanisms is capable of detecting all policy violations. However, we require that each enforcement mechanism is aware of the entire policy, whereas in [9] each mechanism could enforce a subset of the policy. The tradeoff is between scalability (in our work) and flexibility (in [9]).

- Our policy language contains constructs to provide multi-party specification and the appropriate controls.

1.4 Contributions of This Study

In this study, we examine the benefits and problems of network access policy specification. We present solutions and mechanisms to aid in policy specification and enforcement. For this work, we have developed Snapl, a language for specifying network access policies which addresses many of the issues of access policy specification in a large corporate environment. We have implemented a prototype enforcement mechanism that verifies compliance to a Snapl policy through passive monitoring. That is, we have augmented a network monitor to verify that observed traffic complies with a given policy. We have also implemented a learning algorithm that aids in policy specification by automatically generating policies based upon observed traffic. Those mechanisms are described in this document. We also present a formal, abstract model of network access policy enforcement. Within this framework, we describe problems of enforcement that arise due to address spoofing and the fact that connections between a pair of hosts could follow a number of different paths through the network. To guarantee policy enforcement, monitors must be placed to block all possible paths and detect all spoofing that could be used to bypass the policy. We present algorithms that determine if a configuration of network monitors (placed throughout the intranet) is capable of enforcing a given policy in face of those issues. Lastly, we discuss some limitations of guaranteeing policy enforcement.

1.5 Organization

In Chapter 2, we discuss policy specification, the difficulties thereof, and how Snapl addresses some of these difficulties. We also discuss the known limitations of Snapl. In Chapter 3, we present our formal model of intranets and policy enforcement. In Chapter 4, we present the polynomial algorithms for determining policy coverage. We also discuss the limitations of policy coverage determination and present the related proof of NP-hardness. Then in Chapter 5, we describe the implementation of an automated policy learning mechanism, which precipitated from experience in a real corporate environment. Finally in Chapter 6, we conclude this study by assessing the success of our work and discuss ways to extend this research.

Chapter 2

Policy Specification

Many aspects of network access policies can be well expressed in set notation. Consider this example policy: the dedicated FTP servers are the only hosts that should provide the FTP service, that should be the only service that they provide, and this service should only be available to our internal machines. Three attributes in this example that are well expressed by sets are the set of dedicated FTP servers, the set of server ports that are associated with the FTP application, and the set of internal hosts. A user can define these sets with all of the well-known set operators. In addition, these sets can be combined with other sets with the same operators. For example, additional constraints might be placed on research group FTP servers (such as a limit on the number of bytes transferred in a day). Figure 2.1 gives definitions for these sets in Snapl, our language for the specification of network access policies. These are examples of constant definitions, which can be used for policy specification. The first term indicates the data type of the constant which is followed by the name of the constant and then the value. The notation of the set operations are fully described in the following section.

```
IPAddress FTP_servers = 128.120.56.77 + 128.120.56.121 + 128.120.62.133 +
  128.120.60.141
Port FTP_ports = [20-21]
IPAddress internal_hosts = 128.120.*.*
IPAddress research_group_one = 128.120.56.*
IPAddress research_group_two = 128.120.[59-60].*
IPAddress constrained_FTP_servers = FTP_servers *
   (research_group_one + research_group_two)
```

Figure 2.1: Set definitions in Snapl.

The set notation is also useful for describing allowed and disallowed behaviors. We can consider a policy specification as a partitioning of the space of all possible network usages. The partitioning defines that activity which is allowed, that which is denied, and that which is unclassified. The notion of sets is also useful because, after a user defines the sets allowed and denied, we can explicitly compute the set of unclassified activity and the set of inconsistencies. Both of these sets are of concern for a user specifying policies. Reports of unclassified activity indicate that the user may have accidentally omitted specifications. A non-null set of inconsistencies may be a result of misconfiguration. For example, a simple and common error such as providing an incorrect IP address could result in one of the FTP servers in Figure 2.1 being allowed to also serve as a web sever. The user would benefit from being aware of occurrences of either case. A more formal description of our model of policy specification follows in the next section.

2.1 Model of Network Access Policies

We define network activity in terms of end-to-end connections. We describe connections with a finite set of attributes. We use the term *connections*[1], as opposed to *packets*, because we wish to maintain a general model which, in application, may incorporate attributes describing multiple packets. There is much freedom in the types of attributes that can be employed in our model. The only restrictions on the attributes are the following: they must be representable by a number (our current implementation requires numbers with size on the order of 32 bits); for a single connection, each attribute can be bound to at most one value; and they must be observable by a single network monitor. In addition, we require that the first two attributes be the source and destination addresses. Some example attributes are: the ordinary TCP/IP field values (source, destination, source port, destination port, protocol, ACK and SYN flags), timestamp values (hour, day of week, etc.), boolean values describing the application layer (does the content match any of a particular set of strings, such as "/bin/sh"?), and state attributes (number of packets seen from the source within the last time frame, average size of the packets, etc.).

Each possible connection can then be represented by a tuple with values for each attribute, which we call a *connection record*. An example connection record, for a simple set of attributes (used by some firewalls), would be: `(source:128.120.56.7, dest:128.120.59.3, service:SMTP)`. We

[1] The term *connection* used here is a broader, more general definition than a connection defined by TCP/IP [20]. A connection (by our definition) can be a single TCP/IP connection or can be an application-specific defined collection of packets. For example, a connection for the network application *ping* could be the set of ICMP packets sent between two hosts, beginning with the first detected packet and ending with an absence of icmp packets for a defined length of time.

```
BaseClass allowed(Report:NoReport)
BaseClass denied(Report:DeniedReport)
ComplexClass unclassified(Report:UnclassifiedReport) =
   !allowed * !denied
```

Figure 2.2: Standard class specifications.

can then describe sets of connections by the Cartesian product of sets of attribute values. For example, we can describe all accesses to the SMTP service on the server 128.120.59.3 with $U \times \{128.120.59.3\} \times \{SMTP\}$, where U is the set of all values in the attribute domain (all possible IP addresses in this case).

We define a network access policy to have three sets of connection records: *allowed*, *denied*, and *unclassified*, where the allowed and denied sets are defined through policy specification and the unclassified set is inferred as the complement of the union of those sets. Regarding enforcement, we will allow all connections in the allowed set and deny (or report, if utilizing passive enforcement) all connections that are denied or unclassified. For the purposes of describing our formal model, we define the set P to be the allowed set and $\overline{P}$ to be the denied and unclassified sets (the sets of activity that must be identified by the enforcement mechanisms).

2.2 Snapl: Our Specification Language

In this section, we briefly describe our network access policy specification language and show how it addresses difficulties in the specification process. Policy specification is organized into two types of classes: *base classes* and *complex classes*. The former class is defined by specifying its components explicitly (via *connection descriptors*, described later). The latter is defined by a set expression with base classes as literals. This distinction allows the user to specify the negation (!), intersection (*), and union (+) of whole classes. Each class can be associated with a type of action. When network activity that is contained in a class is observed, the associated action is effected. Figure 2.2 contains Snapl specifications for the standard classes, which, in this case, are associated with the actions of generating appropriate reports. These classes can be extended and additional classes can be defined, as shown in the following section.

A *connection descriptor* is a variable-length, attribute value set tuple that specifies a set of possible connections. The tuple can contain set values for any or all of the attributes. The set of connections specified by the descriptor is the Cartesian product of those values, with the universal set (U) assigned to all unspecified attributes. Connection descriptors are used to define base classes; e.g. `allowed(dest:128.120.56.77, service:DNS)` would expand `allowed` to `allowed` $\cup\, U \times \{128.120.56.77\} \times \{DNS\}$ and would allow any client to access 128.120.56.77 on the DNS port.

Attribute value sets can be specified with all of the standard set operators: union (`+`), intersection (`*`), complement (`!`), and difference (`-`). Order of operation is standard and can be overridden with parentheses. Attribute value literals, such as IP addresses, can be specified in the standard ways including ranges and wild cards.

Snapl reports inconsistencies (`allowed` $\cap$ `denied` $\neq \emptyset$) during compilation and enforces the more restrictive policy. In comparison, traditional firewall languages would handle inconsistencies by enforcing the rules that appear first in the specification, potentially leading to misconfigurations [2].

2.3 Merging Policies

One problem in specifying an intranet-wide access policy is that it is unlikely that any manager will be aware of the network usage needs of all users. As such, it will require a collective effort to put the policy together.

Snapl language constructs allow portions of the policy to be specified independently and then merged together with minimal side-effects. The main enabling construct is the order-independent nature of the syntax. Since classes are defined by the union of connection descriptors (a commutative operation), the order of their appearance is irrelevant. When connection descriptors are merged, they can simply be appended.

The class structure can be used to enforce the partitioning of a specification by using separate base classes for each partition. These classes are then intersected with their respective domains before being merged into the final class. Figure 2.3 shows an example of this structure. Alice and Bob are each responsible for an area in the intranet, and may specify any policies involving their hosts. Connection descriptors in the files `alice.policy` and `bob.policy` populate the base classes. A simple check can be made to ensure that neither Alice nor Bob specified connection descriptors for any of the other's classes.

```
BaseClass alice_domain(Report:NoReport)
BaseClass bob_domain(Report:NoReport)
BaseClass allowed_by_alice(Report:NoReport)
BaseClass allowed_by_bob(Report:NoReport)
BaseClass denied_by_alice(Report:NoReport)
BaseClass denied_by_bob(Report:NoReport)

ComplexClass allowed(Report:NoReport) =
  (alice_domain * allowed_by_alice) + (bob_domain * allowed_by_bob)

ComplexClass denied(Report:DeniedReport) =
  (alice_domain * denied_by_alice) + (bob_domain * denied_by_bob)

ComplexClass unclass(Report:UnclassifiedReport) =
  !allowed * !denied

alice_domain(source:128.120.[56-57].*)
alice_domain(dest:128.120.[56-57].*)
bob_domain(source:128.120.62.*)
bob_domain(dest:128.120.62.*)

Include "alice.policy"
Include "bob.policy"
```

Figure 2.3: Example class structure for merging policies.

2.4 Limitations of Snapl

Snapl is a simple domain specific language. While it can specify expressive restrictions on end-to-end communications and provides useful tools for abstraction and composition, it cannot be used to specify all types of policies that might be employable in a network environment. Here we describe a few of these limitations.

Snapl cannot freely aggregate reports of network activity. The language can employ aggregation to a certain degree by the addition of "state" attributes. For example, the attribute `NumBytesFromSource` could be added, it would represent the number of bytes sent by the source address over a past fixed period of time. However, such a state attribute cannot be created within the policy. Such an attribute needs to be implemented within the monitoring mechanism and language explicitly. In addition, the expressiveness of state attributes are limited and cannot be added arbitrarily due to scaling concerns; i.e., an arbitrarily large attribute set in Snapl would result in long compile times.

Snapl does not handle multicast addressing intuitively. Multicast and broadcast addresses are treated as any other address. For example, the address 128.120.56.255 is treated as a unique host address from 128.120.56.33, whereas 128.120.56.33 is really a subset of 128.120.56.255. Multicast addresses are handled likewise. An officer specifying the policy will have to specify policies for both types of addresses separately from the hosts associated with them. In some cases, it will be beneficial to treat broadcast addresses differently from the set of hosts within the corresponding subnet. For instance, it might be wise to specify a policy disallowing the use of broadcast addresses for most applications as they have been exploited for denial-of-service attacks. In other cases, the user might leave a hole or specify an inconsistent policy; i.e., a policy might be specified for a set of hosts not to use certain applications, but not for its associated broadcast (or multicast) address, allowing a miscreant to access an illegal service on a host by using the broadcast address.

2.5 Verifying Policy Compliance Through Network Monitoring

One benefit of using a relatively domain-constrained language is that we can provide some optimizations when verifying compliance to the policy. When monitoring connections for policy breaches, efficiency is important. If the rate at which we can verify connections is lower than the highest observed rate of connections, then the guarantee of policy compliance might not hold due to buffer overflows. As a result, we prefer monitoring efficiency at the cost of an intensive initial set up.

The policy specification is compiled down to a decision table data structure with attribute value sets represented as intervals. Utilizing a binary search, the matching algorithm determines all connection descriptor matches against an attribute value with $O(\lg n)$ comparisons where n is the number of specified distinct attribute values. Bitwise AND's are then used to determine all connection descriptor and base class matches. This requires $O(d/\alpha)$ AND's where d is the number of connection descriptors and α is the maximum width of a bitwise AND. As a result, matching an observed packet requires $O(a \lg n + ad/\alpha)$ or just $O(ad/\alpha)$ simple operations when n is on the same order as d, and a (the number of attributes) is constant. Complex class matches are then computed as defined. By distinguishing between base and complex classes, we prevent a large polynomial explosion of connection descriptors that would result in representing a negated base class. The trade-off is that complex class matches need to be computed separately.

Using this improved matching procedure, we have augmented a network intrusion detection application to verify that observed connections comply with the specified policy.

Chapter 3

Network Model

The network model that we have adopted was designed to be directly applicable to IP, but is general enough to describe any destination-based routing protocol. The model is similar to the one presented in [9] in that it uses the notion of *areas* to abstract the network. That is, instead of modeling each host in the network, we partition the hosts into areas with the constraint that all interfaces between areas must be specified. Areas can represent sets of hosts ranging in size from individuals, to broadcast domains, to whole intranets. This abstraction is essential for reducing the complexity of policy specification and enforcement. For example, the security officer may be unaware of the topology or configuration of certain areas in the network, e.g., the Asia and Europe branches in Figure 1.1. These sections can be merged together in the abstract topology, so long as the policy does not require them to be separate. We do, however, require a consistent host-labeling scheme in that, given any host label, we can deduce the area to which it belongs. For our application, a host's label is its IP address. The importance of the consistent labeling scheme is to determine connectivity between hosts. We assume that there is connectivity between all hosts in the same area and that if two areas are connected, then all hosts contained in these areas are also connected. Lastly, we incorporate the expected flow of packets through the network which will be used to detect misrouted packets or packets that have spoofed source addresses.

3.1 The Abstract Network Topology

We define several graphs to represent the network topology. N_{phys} is an undirected graph that identifies all areas in the model and their physical connectivity (the interfaces between areas).

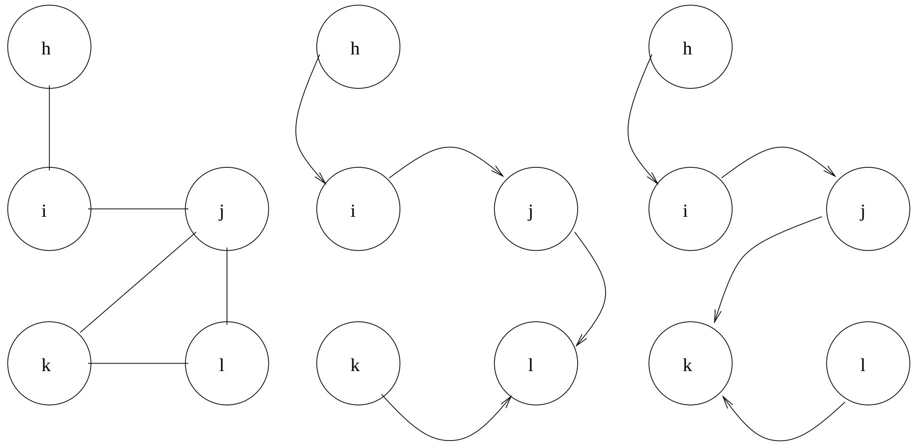

Figure 3.1: Example network graphs: N_{phys}, F_k, and F_l.

Each node in N_{phys} is labeled with an area. Each edge in N_{phys} represents physical connectivity between the two nodes on which it is incident; i.e., packets could potentially be directly routed between the two indicated areas.

N_{flow} is a set of directed graphs that collectively represent the expected flow of network traffic. For each area $d \in A$, where A is the partition of the hosts in the network, there exists a directed graph $F_d \in N_{flow}$, where F_d represents the flow of all packets that have a destination address contained in d. Each node in F_d is labeled with an area. An edge in F_d that is incident from a and incident to b indicates that a packet found in a with a destination address contained in d will be next routed to b [1]. A node in F_d with multiple edges incident from it indicates that packets could potentially be routed in multiple directions. This could be attributed to unstable routing tables or that an area is large, with many routes emerging from various subsections. Note that, for each $d \in A$, $F_d \subseteq N_{phys}$. Figure 3.1 depicts an example of a physical network topology graph and the flow graphs for packets destined to the areas k and l.

3.2 Monitor Placement

We define several sets regarding monitors. I is a set of ordered pairs of areas that represents interfaces between areas that can be monitored. That is, based upon the actual hardware configuration that connects a and b, if we can place a monitoring device such that it can observe traffic going from a to b, then $(a, b) \in I$. Note that a single device may contribute several elements

[1] Our dependency upon destination-based routing becomes apparent at this point as the source takes no part in how we represent flow.

to I. For example, in most situations, a monitor situated between two areas will be able to observe traffic going in both directions. Thus, when this is the case: $(a, b) \in I \Leftrightarrow (b, a) \in I$.

M is the set of ordered pairs of areas that represents interfaces between areas that are monitored. When we introduce methods for finding optimal placements of monitors, we will be populating this set. All monitors are assumed to enforce the entire policy. That is, if $(a, b) \in M$, then any connection which traverses from a to b with corresponding connection record r will be denied if $r \in \overline{P}$. In addition, monitors use the routing information from N_{flow} to deny *intrinsically illegitimate traffic*. When a connection is observed traversing a link where it could not have been routed according to N_{flow}, it is considered intrinsically illegitimate and is denied if a monitor has been placed on that link.

Chapter 4

Graph Algorithms for Policy Enforcement

In this chapter, we present a series of graph algorithms that can be used to determine when intrinsically illegitimate traffic can be detected and to determine if a given policy is covered by a placement of monitors. The basis for most of these algorithms is determining if there is connectivity between areas, i.e., is there a path between area s and area d? This basic problem is rather simple and a straight-forward algorithm (breadth-first search) is presented in [5] that has a running time of $O(n + e)$ where n is the number of vertices (areas) and e is the number of edges (network interfaces between areas).

4.1 Blocking Intrinsically Illegitimate Traffic

Given N_{flow}, we describe a polynomial algorithm to determine the set of legitimately routed traffic that could flow through any link (a, b). We define $L(a, b)$ to be the set of area pairs that could be the sources and destinations of connections legitimately routed through the link (a, b). That is, $(s, d) \in L(a, b)$ entails that the set of connections $(s, d, *)$ could be legitimately routed through the link (a, b), where $(s, d, *)$, a notation we will use often, is equivalent to $s \times d \times U \times ... \times U$. Defined in terms of our model: $L(a, b)$ is the set of pairs (s, d) for all areas s and d such that a path from s to d traverses the edge (a, b) in the graph F_d.

A breadth-first search can be used to find all edges through which paths from s to d could traverse. Label each edge incident from s, (s, b) in F_d for all nodes b with "(s, d)", and add (s, d) to the set $L(s, b)$. For each b that has not been previously labeled with "(s, d)", label b with "(s, d)" and label all edges (b, c) in F_d for all c with "(s, d)", and add (s, d) to the set $L(b, c)$. The algorithm continues as such until no new edges or nodes have been labeled. Since edges are only examined before the node that it is incident from is labeled, each edge is only examined once. Each node is only checked for labels when a new edge is found incident to it. Thus, the algorithm runs on the order of $O(n + e)$, where n and e are the number of nodes and edges in F_d, respectively.

To find all possible legitimate traffic, this process is executed for all n^2 pairs of areas s and d. With n areas and e total edges in N_{flow}, the entire procedure runs on the order $O(n^3 + n^2 e)$.

4.2 Verifying Coverage When Monitors Enforce the Entire Policy

We say that a policy is *covered* if no host is able to establish an illegal connection without traversing through a link monitored by a device that enforces against that connection. If each monitor enforces the entire policy, and we assume that no host will spoof another, then if all paths from s to d in N_{phys} include a monitored link, then the policy is covered for $(s, d, *)$. If for all areas s and d, where $(s, d, *) \cap \overline{P} \neq \emptyset$, there does not exist a path from s to d lacking a monitored link, then the policy is covered. A breadth-first search through N_{phys} (without the links contained in M) can be performed for each of the pairs (s, d) to determine if such a path exists. As shown previously, this can be accomplished in $O(n^3 + n^2 e)$ operations.

By using N_{phys} to determine connectivity, we cover the case that a malefactor will use source routing to bypass the routing controls. If it is certain that all of the relevant routers have been configured to prevent source routing, and that the flow diagrams represent a superset of the possible routing decisions, then F_d can be used to determine connectivity. This will, in some cases, reduce the set of monitors necessary for policy coverage.

4.2.1 Considering Spoofing

In some situations, a host will be able to obtain unauthorized privileges by pretending to be another host. In our model of policy, a host h can obtain unauthorized privileges by pretending to be the host s in a connection with the host d when $(a3_s \times a4_s \times ...ak_s) \not\supseteq (a3_h \times a4_h \times ...ak_h)$ where ai_s is the ith attribute value set in the Cartesian product $(s, d, *) \cap P$ and ai_h is the ith attribute value set in $(h, d, *) \cap P$. That is, the types of connections that are permitted between h

and d is not a superset of the connections permitted between s and d. We define S to be the set of all tuples of areas (h, s, d) such that the above conditions hold. That is, $(h, s, d) \in S$ implies that if it is possible for h to spoof connections to d as the host s, then h will be able to bypass policy enforcement based upon source address. Since our policy enforcement devices use host addresses for authorization, we must rely on the blocking of intrinsically illegitimate traffic to prevent this type of policy violation. Fortunately, we can prevent much spoofing by placing monitors on the interfaces of networks containing hosts with limited access privileges. We are, however, bound to only stopping cross-area spoofing. That is, if two hosts reside in the same area, then we have no mechanism to detect when one may spoof another. When partitioning the hosts in the network, this limitation should be kept in mind. To prevent spoofing within an area, that area will need to be partitioned, separating the hosts with different privileges.

In Chapter 4.1, we showed how we can determine the set of intrinsically illegitimate traffic at each link in the graph. Any monitor placed upon a link can be configured to block all such traffic. For our model, we assume that all monitors are configured to do so. Given a monitor placement M, we can then determine all of the links over which spoofed connections could traverse.

Create a new graph G with vertices and edges from N_{phys}. Additional *spoof links* are added to the graph that represent links over which a host could spoof an address and establish an illegal connection, whether or not a monitor is present on that link. Spoof links are labeled with a set of pairs of areas (h, d) with the meaning that h can establish an illegal connection with d over that link while impersonating some other host. More formally, the set of spoof link labels for each link (a, b) is defined to be $\{(h, d) \mid (h, s, d) \in S \wedge (s, d) \in L(a, b)\}$. That is, there is a spoof link along each edge wherever a connection between s and d could be legitimately routed to that link and a host in h could obtain unauthorized privileges by pretending to be s. We then define a path from s to d in G to be any series of consecutive edges including unlabeled physical edges and spoof links that are labeled with (s, d). If a monitor is placed on the link (a, b), it does not interrupt a path from s to d that traverses spoof links from a to b. By defining G in this way, we model the fact that a host can establish unauthorized connections through a monitored link if it spoofs another host with additional privileges and that connection could have been legitimately routed across that link. Figure 4.1 depicts the graph G for our previous example, with a policy P that yields the set:

$$S = \{(h, j, k), (h, j, l), (h, k, l), (h, l, k), (i, j, k), (i, k, j), (i, k, l), (i, l, k)\}$$

That is, h and i have the same privileges when accessing k and l, which is less than j has, entailing that if h or i can spoof as j when accessing k or l, they may establish illegal connections. In addition,

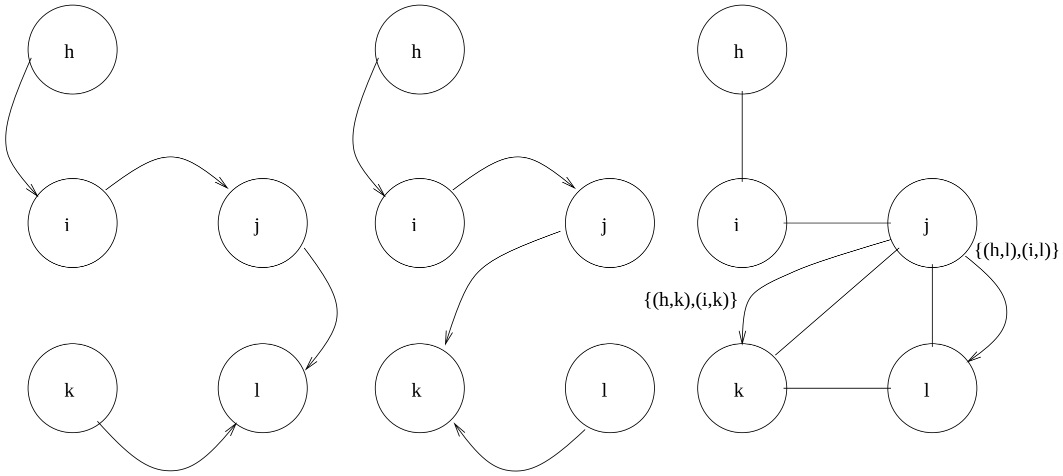

Figure 4.1: Example graphs F_k, F_l, and G.

it can be seen that $L(j,k) = \{(h,k),(i,k),(j,k)\}$, and $L(j,l) = \{(h,l),(i,l),(j,l)\}$. Thus, the spoof link (j,k) is added to G, and labeled with $\{(h,k),(i,k)\}$ and likewise (j,l) is added with the label set $\{(h,l),(i,l)\}$. As a result, monitors placed on the edges (j,k) and (j,l) would not be sufficient to enforce the policy, as hosts in either h or i would be able to spoof hosts in j to obtain additional privileges.

To verify coverage while considering spoofing, we perform the same breadth-first searches on G as we did on N_{phys} in the previous section, with the exception that we only follow spoof edges that are labeled with the currently investigated node pair. The complexity of each search is increased with the inclusion of the spoof links. When deciding whether or not to follow a link (a,b) during a search regarding the source and destination pair (s,d), it must be determined if (s,d) is contained in the label set. Assuming that a sorted table is used to store the (n^2 in worst case) labels, a simple binary search would require $O(\lg n^2)$ comparisons. A total running time for this approach would be $O(n^3 + n^2 e \lg n^2)$.

4.3 Verifying Coverage When Monitors Enforce Subsets of the Policy

During the initial steps of our work we considered solving the problem addressed in [9] in terms of our model. That is, we hoped to remove the assumption that every monitor enforces the entire policy, and that each monitor could be configured to enforce different aspects of the policy such that collectively the global policy would be enforced. To discuss this problem in detail, we expand the definition of our model slightly. M is redefined as a function such that $M(a, b)$ is equal to the set of traffic that is denied by the monitor residing on the link (a, b). Previously, $M(a, b)$ was assumed to be $\overline{P}$ for all monitored links (a, b). We will show that verifying coverage for arbitrary graphs, policies, and monitor configurations is NP-hard. As such, providing optimizations of configurations is likely to be intractable, and thus we have not pursued such methods.

There is an essential difference in complexity between the two models. When the monitors enforce different portions of policies, then each combination of monitors can be enforcing different aspects of policies. We shall show that this leads to combinatorial complexity. When all monitors enforce the entire policy, we concern ourselves only with ensuring that no open path exists between each relevant source and destination. The method presented in [9] (albeit for a different model) enumerates all possible noncyclic paths in a graph representing the network topology. This approach is likely to be unscalable. For example, in a fully-connected graph, a path from any s to any d can have any permutation of intermediate nodes. Thus, there would be an exponential number of noncyclic paths, relative to the number of nodes.

4.4 Verifying Coverage is NP-Hard for Arbitrary Graphs and Policies

We show that the verification of policy coverage is NP-hard by reducing 3-CNF-SAT.

A 3-CNF boolean expression is satisfiable if we can assign a true value to one literal (or its negation) from each clause, without being inconsistent, i.e., we do not assign a true value to both a literal and its negation. A boolean expression is not satisfiable if it is not possible to remain consistent with any possible assignment.

An algorithm that verifies the coverage of a network access policy determines if, for any nodes s and d in N_{phys} there exists a path p in F_d, beginning at node s and ending at node d such that:

$$P \not\supseteq \bigcap_{(a,b) \in Edges(p)} \overline{M(a, b)}$$

That is, a path on which every monitor allows some traffic not contained in the defined legitimate set to go unreported. If there exists such a path, the algorithm reports that the policy has not been covered.

Below we describe a polynomial algorithm that maps 3-CNF-SAT problems into a policy coverage verification problems.

1. Order each unique literal l, such that each l has a corresponding position number i.
2. Construct N_{phys} as such:
   (a) Add a node labeled s.
   (b) For each clause c in the expression:
   - For each literal l in c:
     i. Add an edge e incident to the last created node
     ii. If l is negated, then set
     $$M(e) = U \times U \times U ... \times \{1\} \times ...U$$
     where the non-universal set corresponds to attribute $i + 2$, and the domain for this attribute is $\{1, 2, 3\}$. As such, the monitor on this edge will allow all traffic in the set
     $$U \times U \times U ... \times \{2, 3\} \times ...U$$
     iii. Otherwise, set
     $$M(e) = U \times U \times U ... \times \{3\} \times ...U$$
     allowing traffic in the set
     $$U \times U \times U ... \times \{1, 2\} \times ...U$$
     iv. Add an incidental node for each of these edges, followed by one new node incident to edges from each of these nodes.

   (c) Label the final node d.
3. Construct N_{flow} with all subgraphs $F_i = N_{phys}$ with all directed edges directed from s to d.
4. Construct P as the following union of connection descriptors:
   - $U \times U \times \{2\} \times U ... \times U$
   - $U \times U \times U \times \{2\} \times U ... \times U$
   - ...

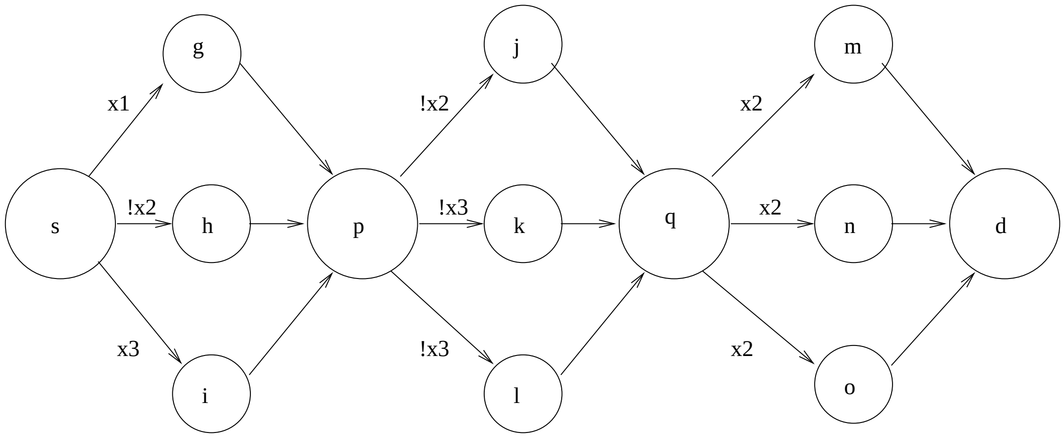

Figure 4.2: F_d graph for the 3-CNF-SAT example.

- $U \times U \times U \times U ... \times \{2\}$

Figure 4.2 depicts F_d for the expression: $(x_1 \vee !x_2 \vee x_3) \wedge (!x_2 \vee !x_3 \vee !x_3) \wedge (x_2 \vee x_2 \vee x_2)$, where ! is the boolean *negation* operator and $\vee$ and $\wedge$ are the boolean operators *or* and *and*, respectively. For this example, M is defined as:

- $M(s, g) = U \times U \times \{3\} \times U \times U$; x_1
- $M(s, h) = U \times U \times U \times \{1\} \times U$; $!x_2$
- $M(s, i) = U \times U \times U \times U \times \{3\}$; x_3
- $M(p, j) = U \times U \times U \times \{1\} \times U$; $!x_2$
- $M(p, k) = U \times U \times U \times U \times \{1\}$; $!x_3$
- $M(p, l) = U \times U \times U \times U \times \{1\}$; $!x_3$
- $M(q, m) = U \times U \times U \times \{3\} \times U$; x_2
- $M(q, n) = U \times U \times U \times \{3\} \times U$; x_2
- $M(q, o) = U \times U \times U \times \{3\} \times U$; x_2
- $M(A, B) = \emptyset$ for all other edges (A, B)

We can use the verification algorithm to determine if P is covered. If P is not covered, then the path violating coverage indicates an assignment that satisfies the boolean expression without introducing an inconsistency. The following properties prove that this is the case:

1. There is an unique path from s to d for each possible choice of literal from each clause. This follows from the fact that as a path continues from one node to the next, it may take any one of the three possible edges, each of which corresponds to a literal in the corresponding clause.

2. If an edge corresponding to an unique literal lies on a path, as does an edge corresponding to its negation, then the policy is covered for that path. Given any literal l whose attribute is in the ith position, the intersection of the monitored sets of its positive and negative sets is: $U \times U ... \times \{2\} \times ...U$ which is a subset of P. Given that a subset of any set P intersected with any other set remains a subset of P, it is the case that for all paths containing edges corresponding to both the positive and negative of a literal must be a subset of P.

3. If a path exists from s to d such that, for all unique literals, none had their positive and negative forms assigned to be true, then P is not covered. $\bigcap_{(a,b) \in Edges(p)} \overline{M(a,b)}$ where all edges in p lack the pair of positive and negative representations of a unique literal will be a single Cartesian product such that each attribute value set will be U if the corresponding literal has not been assigned at all, $\{2,3\}$ if the negation is assigned, and $\{1,2\}$ if the positive is assigned. Since none of the attribute value sets is a subset of $\{2\}$, the tuple is not a subset of P and thus, P is not covered.

4. Since every possible combination of literal assignments is represented by a path, every path that does not represent a valid assignment results in policy coverage, and every path that does represent a valid assignment results in not policy coverage, then if the policy is not covered, then there must exist a valid assignment, and if the policy is covered, then there must not exist a valid assignment.

For the above example, we examine the path p pertaining to $x_1, !x_2, x_2$, $\bigcap_{(a,b) \in Edges(p)} \overline{M(a,b)} = U \times U \times \{1,2\} \times \{2\} \times U \subseteq P$. As such, the policy is covered for that path, indicating that this is an illegitimate assignment. We examine the path p pertaining to $x_1, !x_3, x_2$, $\bigcap_{(a,b) \in Edges(p)} \overline{M(a,b)} = U \times U \times \{1,2\} \times \{1,2\} \times \{2,3\} \nsubseteq P$. As such, the policy is not covered for that path, indicating that this assignment satisfies the expression.

Chapter 5

Learning Policies Based on Observed Traffic

After deploying our prototype of the Snapl compiler and network monitoring enforcement mechanism, we began to work with a corporate network to specify an access policy for their intranet. We discovered that some aspects of the network usage were unknown to the security group with whom we were interacting. As a result, there were large amounts of unclassified network activity reports. To aid in policy specification and log management, we decided to employ a machine learning algorithm that would automatically generate policies to cover these reports.

We chose to deploy the efficient concept learning algorithm, called Find-S in [14], to generate allowed connection descriptors based on observed traffic. The algorithm finds a maximally specific hypothesis that covers the observed traffic, for some defined *hypothesis space*, described in the next section. A new class, learnClass, is used to control which aspects of the policy will be inferred. Any reports of unclassified activity that are contained in learnClass are passed to the policy learning module which updates the policy. This enables an officer to "lock down" policies that are believed to be complete, simply by removing them from learnClass. Figure 5.1 shows the learnClass definition and usage. As depicted, only unclassified network activity will be reported to the learning mechanism. Learned policies are kept separate from specified ones so that an administrator can review the summarized behavior of the network and eventually update the specified policy.

```
BaseClass allowed(Report:NoReport)
BaseClass denied(Report:DeniedReport)
BaseClass learnClass(Report:NoReport)

ComplexClass unclassified(Report:UnclassifiedReport) =
                !allowed * !denied * !learnClass

ComplexClass learn(Report:LearnReport) =
                !allowed * !denied * learnClass

learnClass(dest:128.56.79.* + 128.56.82.*)

# All learned policies are stored in the external source file: learned.policy
Include "learned.policy"
```

Figure 5.1: Example use of machine learning in policy specification.

5.1 Hypothesis Space

A *policy hypothesis* is a set of connection descriptors that covers (is a superset of) some set of observed network activity. The *hypothesis space* is the set of all possible policy hypotheses and is ordered from most specific to least specific, i.e. from no connections to all possible connections.

To drive the explanation of these concepts, we present the following example. We would like to learn what hosts are acting as servers, what applications they are offering, and what hosts are utilizing them. To accomplish this task, we consider the attributes source, destination, and service, only, where the source of a connection is the initiator of the connection and the destination is the provider of the service. The hypothesis space ranges from `(dest:Null, source:Null, service:Null)` to `(dest:U, source:U, service:U)`. To provide generalization in the concept learning algorithm we limit the range for each of these attributes to specific sets. These are defined by hierarchies of attribute value concepts. An attribute value concept hierarchy is defined for each attribute. Figure 5.1 depicts a hierarchy for the client attribute that might be used in a typical corporation. The leaf nodes represent individual IP addresses and are the most specific client concept (aside from Null). Each parent node represents a concept that is the super set of all of its children and is considered less specific. In this case, we have partitioned the client hypothesis space according to the network topology, but any meaningful partitioning scheme could have been used.

The concept learning algorithm uses these hierarchies to state the maximally specific concept depicting what it has observed. If the monitor observes a single client, then that individual host (the corresponding leaf node) is the maximally specific applicable concept. If an additional

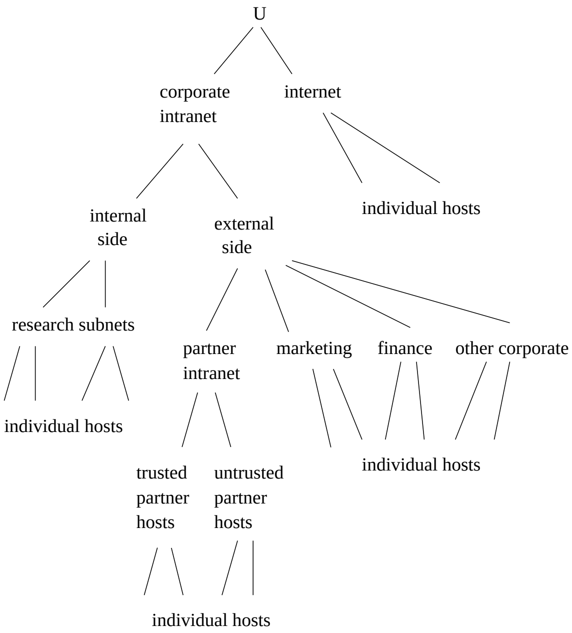

Figure 5.2: Concept hierarchy for clients.

client is observed, then the maximally specific applicable concept would be the closest common parent to the observed leaf nodes. For example, if a trusted partner host is observed acting as a client, followed by an untrusted partner host, then the maximally specific client hypothesis would be "partner intranet".

When a new connection record is reported, the learned policy is updated as follows. If a connection descriptor in the policy can be generalized to include the new record, then the concepts in the descriptor are updated. Otherwise, a new connection descriptor is added to the policy. This descriptor includes all of the leaf node concepts corresponding to the new record. For example, if the current learned policy is `(dest:128.120.56.88, source:128.120.57.82, service:25)` and the connection record `(dest:128.120.56.88, source:128.120.57.85, service:25)` is observed, then the new policy would be expanded to `(dest:128.120.56.88, source:128.120.57.*, service:25)` where 128.120.57.* is the closest parent node of the two observed clients.

A concept hierarchy need not be a tree that generalizes to U. We also utilize two additional types of attribute concepts: *flat* and *combinatorial*. A flat attribute concept cannot be generalized. To implement our example policy hypothesis, we consider the source attribute (server) to be flat. When a connection record is reported for a previously unobserved server, then a new connection

```
allowed(dest:128.120.56.88,
        source:trustedPartner,
        service:22)
allowed(dest:128.120.56.99,
        source:external,
        service:80 + 8080 + 23)
```

Figure 5.3: Examples of learned policies.

descriptor is created for that server. This way, we can distinguish which services, and sources (clients) are associated with each individual server. A combinatorial attribute concept includes all possible combinations of values, i.e., we incrementally generalize the hypothesis as new leaf node concepts are observed. For our example, we consider the `service` attribute to be combinatorial, because we are interested in each service that a server is offering.

Figure 5.3 depicts some sample learned policies. The first connection descriptor allows all trusted partners to use `ssh` on the specified server. It was learned by observing two or more hosts in the `trustedPartner` set accessing that port on the server. The second connection descriptor shows that one of the web servers was observed offering a telnet connection, and that hosts from around the corporation have been accessing this server; this is not a good sign!

Because of its simplicity, Find-S may be a surprising choice for a concept learning algorithm. However, it has features that make it quite applicable to Snapl. It is efficient. It operates with unlabeled data. Its results can be easily understood by the end user. Finally, it summarizes observed data with a desired level of generalization. In practice, the learning mechanism was quite beneficial at our corporate test site. It provided a good summary of the servers that were operating in a sensitive area and the services that they were providing. This summary was generated with a minimal amount of log space and yielded insight into the types of vulnerabilities that may exist in that area.

Chapter 6

Conclusion

We had three goals that we hoped to accomplish with this work. First, we wanted Snapl to be an intuitive, expressive language that allows administrators to specify complex, yet easily discernible policies. We believe that we have accomplished this task by utilizing an applicable and widely accepted set notation. Following the use of set notation to its fruition, we were able to incorporate constructs for specification controls within the policy itself. We also provided abstraction and composition techniques such as labeled constants, and macros. All of these constructs were incorporated in an interpreter that took only a few months for a single student research assistant to implement. We utilized the tool in practice, (within a real corporate environment) and found policy specification to be relatively easy with Snapl. Perhaps, even more useful, was the learning mechanism that was later employed. The learned policies provided an useful summary of previously unknown servers and services.

Second, we wanted to show that an intranet-wide policy can be enforced with a distributed set of monitors and that it was feasible to determine whether or not a configuration of monitors was capable of enforcing any given policy. By providing polynomial algorithms, we showed that such a mechanism is feasible and scalable. These algorithms provide further functionality than the approach presented in [9]. There, spoofing is handled with the heuristic that packets from an area a to an area b should not legitimately traverse any other area if a and b are directly connected. Rules are generated that prohibit any such routing. While this can be beneficial to security, it does not prevent spoofing in large domains (where many areas are connected, but not directly adjacent to each other) and might even block legitimate traffic. As a result, this heuristic cannot be used to provide any guarantee about policy compliance. We model routing which allows us to detect truly illegitimate traffic and can use this knowledge to provide a guarantee that the policy cannot be bypassed through address spoofing.

Finally, we wanted to determine some bounds on intranet-wide policy enforcement. We proved that determining whether or not an arbitrary configuration of monitors and local policies enforced a global policy is NP-hard. As a result, we conclude that it is unlikely for any solution to be scalable.

6.1 Future Work

The most apparent step for future work would be the implementation of the policy verification algorithms outlined in this research. The end result of this work would be a network security management tool that would help security officers to do a number of things. First, the officers can map out the physical network topology (are a number of currently available tools accomplish this job). Next, officers work with managers to discuss policy specification, balancing the need for applications with the limitation of vulnerabilities. Using Snapl, the policy is specified in parts by the various personnel and merged to generate the intranet access policy. Then, the above methods are used to aid in monitor selection and placement. The initial step is to determine if it is possible to enforce the policy, given the configuration. The officer may have places in mind that would not be feasible for monitoring and removes those interfaces from I (the set of interfaces defined in Chapter 3). The verification of coverage algorithm is executed with $M = I$ which will run in a reasonable amount of time (M is the set of monitored interfaces, also defined in Chapter 3). That is, all interfaces that can be monitored, are monitored. If the verification algorithm returns that the policy is not covered, then the personnel will need to evaluate if the set of uncovered policy is small enough to be acceptable, otherwise either the policy or area configuration is revised. When a policy is successfully, or adequately covered by a configuration of monitors, the formal policy specification can then be promulgated to the monitors. The formal policy can be reviewed by other security experts to identify potential vulnerabilities, with the knowledge that the behavior of the network is a subset of that specification.

Appendix A

Snapl Grammar

Policy ⇒ *AttributeListDeclaration StatementList*

AttributeListDeclaration ⇒ Attributes (*AttributeList*)

AttributeList ⇒ *AttributeDeclaration*

| *AttributeList* , *AttributeDeclaration*

AttributeDeclaration ⇒ *Symbol* : *AttributeType*

StatementList ⇒ *Statement*

| *StatementList Statement*

Statement ⇒ *AttributeType Symbol* = *Set*

| BaseClass *Symbol* (Report:*ReportFunction*)

| ComplexClass *Symbol* (Report:*ReportFunction*) *ClassExpression*

| *Symbol* (*ConnectionDescriptor*)

| Query (*ConnectionDescriptor*)

| Include *LiteralString*

| Macro *Symbol* (*ParameterList*) *MacroBody*

| Call *Symbol* (*ArgumentList*)

ArgumentList ⇒ *Set*

| *ArgumentList* , *Set*

ConnectionDescriptor ⇒ *SetSpecification*

| *ConnectionDescriptor* , *SetSpecification*

SetSpecification ⇒ *Symbol* : *Set*

$Set \Rightarrow Symbol$

$\mid LiteralSet$

$\mid Number$

$\mid Range$

$\mid IPAddress$

$\mid ! \; Set$

$\mid Set + Set$

$\mid Set \; * \; Set$

$\mid Set$ - Set

$\mid (\; Set \;)$

$Range \Rightarrow [\; Number$ - $Number \;]$

$IPAddress \Rightarrow IPByte \; . \; IPByte \; . \; IPByte \; . \; IPByte$

$IPByte \Rightarrow Number$

$\mid Range$

$\mid *$

$ClassExpression \Rightarrow Symbol$

$\mid ! \; ClassExpression$

$\mid ClassExpression + ClassExpression$

$\mid ClassExpression \; * \; ClassExpression$

$\mid (\; ClassExpression \;)$

$ParameterList \Rightarrow Symbol$

$ParameterList \; , \; Symbol$

$AttributeType \Rightarrow$ IPAddress | Port | Protocol | Confidence | FlowDirection | Bytes | DayOfWeek

| Day | Month | Year | Hour | Minute | Second

$ReportFunction \Rightarrow$ DeniedReport | UnclassifiedReport | NoReport | LearnReport

$LiteralSet \Rightarrow$ U | Null | UnknownApp | tcp | udp | icmp | sdir | cdir | bidir

$Number \Rightarrow [0-9]^{+}$

$Symbol \Rightarrow [a-zA-Z\_0-9]^{+}$

$LiteralString \Rightarrow$ "$[\neg$"$]^{*}$"

$MacroBody \Rightarrow \{[\neg\}]^{*}\}$

Bibliography

[1] J. Alves-Foss. An overview of SNIF: A tool for surveying network information flow. Technical report, Laboratory for Applied Logic, Department of Computer Science, University of Idaho.

[2] D. B. Chapman. Network (in)security through IP packet filtering. In *Proc. of the Third USENIX UNIX Security Symposium*, 1992.

[3] D. B. Chapman and E. D. Zwicky. *Building Internet Firewalls.* O' Reilly & Associates, Inc., 1995.

[4] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security.* Addison-Wesley Publishing Company, 1994.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw Hill, 1996.

[6] Cisco WheelGroup Corporation. NetRanger. http://www.wheelgroup.com.

[7] D. Denning. An intrusion detection model. In *Proc. 1986 IEEE Symposium on Research in Security and Privacy*, 1986.

[8] S. Garfinkel and G. Spafford. *Practical UNIX & Internet Security.* O' Reilly & Associates, Inc., 1996.

[9] J. D. Guttman. Filtering postures: Local enforcement for global policies. In *Proc. 1997 IEEE Symposium on Research in Security and Privacy*, 1997.

[10] L. T. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Proc. 1990 Symposium on Research in Security and Privacy*, 1990.

[11] K. Ilgun. Ustat: A real-time intrusion detection system for UNIX. In *Proc. 1993 IEEE Symposium on Research in Security and Privacy*, 1993.

[12] C. E. Landwehr, A. R. Bull, J. P. McDermott, and W. S. Choi. A taxonomy of computer program security flaws. *ACM Computing Surveys*, 26(3), September 1994.

[13] T. F. Lunt, A. Tamaru, R. Jagannathan, C. Jalali, H. S. Javits, A. Valdes, and P. G. Neumann. A real-time intrusion-detection expert system (IDES). Interim Progress Report Project 6784, SRI International, 1990.

[14] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[15] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, May 1994.

[16] T. H. Ptacek. Insertion, evasion, and denial of service: Eluding network intrusion detection. *Secure Networks, Inc.*, January 1998.

[17] M. J. Ranum, K. Landfield, M. Stolarchuk, M. Sienkiewicz, A. Lambeth, and E. Wall. Implementing a generalized tool for network monitoring. In *Proceedings of the 11th Systems Administration Conference*, 1997.

[18] S. R. Snapp, G. V. Dias, T. L. Goan, T. Grance, L. T. Heberlein, C. Ho, K. N. Levitt, D. Mansur, B. Mukherjee, S. E. Smaha, and J. Brentano. DIDS (distributed intrusion detection system) – motivation, architecture, and an early prototype. In *Proc. 14th National Computer Security Conference*, 1991.

[19] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, and R. Yip. GrIDS – a graph based intrusion detection system for large networks. In *Proc. 19th National Information Systems Security Conference*.

[20] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley Publishing Company, 1994.