Firewall Analysis with Policy-Based Host Classification

Robert Marmorstein and Phil Kearns – The College of William and Mary

ABSTRACT

For administrators of large systems, testing and debugging a firewall policy is a difficult process. The size and complexity of many firewall policies make manual inspection of the rule set tedious and error-prone. The complex interaction of conflicting rules can conceal serious errors that compromise the security of the network or interrupt the delivery of important services. Most existing tools for verifying the policy require the user to provide a detailed set of test cases or queries, which can sometimes be as difficult as verifying the policy by hand. Deriving a sufficiently comprehensive set of tests requires a detailed knowledge of potential vulnerabilities and a familiarity with the mechanics of the firewall. It also requires a significant investment of time and other resources. In this work, we present a fully automatic technique for identifying significant anomalies in a firewall policy. Our technique employs a novel system for classifying the hosts of a network into classes based on an equivalence structure, which is calculated from the firewall policy. This ''policy-based classification of network hosts'' substantially reduces the difficulty of identifying potential errors in the configuration of a firewall or group of connected firewalls and can be combined with existing firewall verification techniques to improve their effectiveness in detecting errors.

Introduction

A comprehensive firewall policy can protect a network against many serious internal and external threats. For installations with many hosts, however, the process of maintaining a restrictive policy can be prohibitively difficult. Firewall rules often interact in confusing ways. As the topology of the network evolves and users request new services, rules must be added and removed from the policy. These modifications make the policy increasingly complex and increase the likelihood of introducing serious errors into the firewall policy. For this reason, many system administrators avoid restrictive firewall policies and rely other security measures to keep the network secure [17]. This is unfortunate, since a properly configured firewall can play a major role in defense-in-depth and in protecting legacy or unpatched systems.

One reason firewalls are so difficult to manage is that slight differences in the rule set can cause dramatic changes in the behavior of the firewall. For instance, on *iptables* firewalls [5], the filtering policy is specified using ordered chains of rules. In each chain, the first rule that matches a packet is used to determine the fate of the packet. Reversing two rules can introduce an error that is difficult to detect, but significantly modifies the behavior of the firewall. Other firewall systems have similarly deceptive semantics. For instance, ipfilters firewalls use a ''last-match'' policy to determine the fate of the packet [16]. Features such as network address translation and stateful filtering can also create opportunities for introducing difficult-to-detect errors.

While the techniques we describe in this work can be applied to any packet filtering system, we use *iptables* rules in our examples. An *iptables* firewall has three built in chains. The INPUT chain is used to process packets that are destined for the firewall host itself. The OUTPUT chain is used for packets generated by the firewall host. The FORWARD chain manages packets that pass through the host to other machines. The policy language also provides the ability to define new chains which can called from existing chains by making them the target of a firewall rule.

Each of the built-in chains consists of a default policy and an ordered list of rules. When deciding whether to accept or drop a packet, the firewall considers the rules of the appropriate chain in order until a matching rule is found.

Each firewall rule consists of a set of matches and a target. The matches specify criteria that determine which packets should be processed by the rule. The target tells the firewall what to do with a packet that matches. The target ACCEPT means to allow the packet through the firewall. The targets DROP and REJECT mean to discard the packet. The user can also instruct the firewall to pass the packet on to a user-defined chain for more processing or LOG the packet to a file.

Figure 1 gives an example of a simple *iptables* policy. The INPUT chain of the firewall allows SSH traffic to reach the firewall host itself. The default policy of DROP ensures that all other traffic to the host will be discarded. Outgoing traffic from the firewall, however, is allowed to pass. Although the OUTPUT chain contains no rules, the default policy of ACCEPT allows the firewall host to send any packet to any host.

The FORWARD chain of the firewall protects an internal network 192.168.1.0/24 against threats from the outside world. Only SSH and HTTP connections to internal machines are allowed. All other traffic is blocked. HTTP traffic will only be accepted if it is bound for the web server (host 192.168.1.3).

Even a simple policy such as this one can be difficult to maintain. Suppose, for instance, that a new wireless network is brought online as subnet 192.168.2.0/24. To better protect the existing internal network, the system administrator may add a rule to the firewall policy that blocks SSH traffic from the new network. Figure 2 shows one way the system administrator might modify the policy.

Rule 3 of the new FORWARD chain is intended to prevent hosts on the wireless network from connecting to protected machines. Unfortunately, the order of Rules 1 and 3 creates a hole through which SSH traffic can enter the protected network. As described above, the firewall processes the rules from top to bottom and applies the first rule that matches. When an untrusted machine from 192.168.2.0/24 tries to access a protected machine on the 192.168.1.0/24 subnet, Rule 1 will grant access and Rule 3 will never even come into play. A correct policy, in which the rules are ordered properly is given in Figure 3.

In a simple three rule policy, errors like this can be easily detected and corrected. Many firewalls, however, have dozens or even hundreds of rules. Debugging these larger policies can be extremely difficult.

Existing Tools

The difficulty of testing the firewall can be reduced by using firewall analysis tools. Existing tools fall into three general categories: active testing tools, passive testing tools, and structure analysis tools. Active testing tools check for specific vulnerabilities by transmitting packets over the wire. Passive testing tools perform an off-line analysis to answer user queries about the policy. Structure analysis tools

INPUT (Default DROP)				
#	**Target**	**Source Address**	**Destination Address**	**Options**
1	ACCEPT	Anywhere	Anywhere	TCP dpt:22
OUTPUT (Default ACCEPT)				
#	**Target**	**Source Address**	**Destination Address**	**Options**
FORWARD (Default DROP)				
#	**Target**	**Source Address**	**Destination Address**	**Options**
1	ACCEPT	Anywhere	192.168.1.0/24	TCP dpt:22
2	ACCEPT	Anywhere	192.168.1.3	TCP dpt:80

Figure 1: A simple *iptables* firewall.

FORWARD (Default DROP)				
#	**Target**	**Source Address**	**Destination Address**	**Options**
1	ACCEPT	Anywhere	192.168.1.0/24	TCP dpt:22
2	ACCEPT	Anywhere	192.168.1.3	TCP dpt:80
3	DROP	192.168.2.0/24	192.168.1.0/24	TCP dpt:22

Figure 2: Adding a rule to the chain.

FORWARD (Default DROP)				
#	**Target**	**Source Address**	**Destination Address**	**Options**
1	DROP	192.168.2.0/24	192.168.1.0/24	TCP dpt:22
2	ACCEPT	Anywhere	192.168.1.0/24	TCP dpt:22
3	ACCEPT	Anywhere	192.168.1.3	TCP dpt:80

Figure 3: A correct policy.

FORWARD (Default DROP)				
#	**Target**	**Source Address**	**Destination Address**	**Options**
1	ACCEPT	192.168.1.*	Anywhere	TCP dpt:80
2	ACCEPT	Anywhere	Anywhere	state ESTABLISHED

Figure 4: Forwarding chain of a stateful firewall.

identify poor practices such as duplicate rules in the specification of the firewall policy.

Active Testing Tools

Port scanners such as nmap [7] and hping [3] can be used to simultaneously test the firewall and important servers. Vulnerability testing tools [6, 10, 9, 2] also allow the system administrator to detect common firewall errors. These tools are ''active tools'' in the sense that they test the security of the network by transmitting packets through the firewall. Although this has the advantage of testing both firewall and host, it has the disadvantage of only detecting errors when the host is available. If a host is down for maintenance or powered off by a user, these tests cannot be used to verify the firewall policy. Furthermore, because active tools consume a significant amount of bandwidth and CPU time, they can only examine a small subset of the possible packets a firewall might encounter. To thoroughly test every possible combination of source address, destination address, and network port is infeasible.

Active tools require the user to decide which behaviors to test. For instance, a port scanner requires a list of hosts to scan. Scanning all ports on a few important servers will often catch the most critical vulnerabilities, but it is often helpful to also scan individual workstations for less obvious errors. To check for as many vulnerabilities as possible, the user must carefully craft a testing pattern that balances running time against the number of hosts to scan, the number of ports to check, and the number of spoofed source addresses to employ. Vulnerability scanners such as Nessus [10] also require a significant amount of user input. These tools make use of a database of predesigned tests. While well-known vulnerabilities can usually be caught using the scripts provided with the scanner, creating new tests requires learning a sophisticated scripting language.

Passive Testing Tools

One alternative to active testing is off-line analysis of the firewall policy using a query engine. Passive tools [15, 18, 11, 4] use efficient data structures to analyze a representation of the firewall policy without transmitting any packets over the network.

Passive tools can be much faster than active tools and do not interfere with other network activities. In theory, passive tools can test every possible behavior of the firewall. In practice, however, such a test produces too much unstructured output to be useful. Since the decision of which behaviors are desirable and which are undesirable must be made by the user, testing all eventualities would produce output for every possible packet seen by the network. Since there are 255^4 possible source addresses and the same number of destination addresses, there are billions of packets to consider – an overwhelming amount of output.

To avoid this problem, the user must carefully construct a set of queries that test for specific vulnerabilities. While it is often easier to construct these tests than to inspect the rule set manually, it can be difficult to create queries that test enough interesting behaviors and produce useful output. As with active testing tools, there is no way to guarantee that all important behaviors have been tested. If the system administrator fails to provide a test for an important threat, the testing software cannot detect that the firewall is vulnerable. Errors that are difficult to catch by manually inspecting the rule set are also likely to be overlooked by the user when creating queries for a passive tool and escape detection.

Subtleties in the syntax of the query language can also cause the query engine to generate unexpected results. In previous work, we introduced ITVal [12, 14, 13], a passive analysis tool for *iptables* firewalls. Our tool uses decision diagrams to store a representation of the rule set and allows the user to perform queries such as ''From which hosts is SMTP allowed to the mail server?''

```
QUERY DADDY FOR TCP 80 AND
 NOT FROM 192.168.1.*;
# Addresses: *.*.*.*
```

Figure 5: An example ITVal query.

The ITVal query given Figure 5 might be used to discover which servers provide web access to hosts outside the network. The ''DADDY'' subject tells ITVal to list the destination addresses of these machines. The query condition ''FOR TCP 80'' specifies a match against all HTTP packets while the condition ''NOT FROM 192.168.1.*'' excludes internal hosts from consideration. (For a more detailed treatment of the syntax of ITVal queries, we refer the user to [14].) For many firewalls, the query in Figure 5 will work as expected. However, for a stateful firewall, such as the *iptables* rule set of Figure 4, it is likely that this query will generate many false positives.

When ITVal processes the query against the stateful firewall, it will report that any host can send web traffic through the firewall. This surprising result is technically true. The rule on line 2 of Figure 4 allows

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	10.239.202.38	Anywhere	dpt tcp:25
2	ACCEPT	10.239.202.0/24	10.239.202.38	dpt tcp:25

Figure 6: Controlling mail with a packet filter.

arbitrary access on established connections. A more careful examination of the rule set, however, reveals that only machines on the internal network can initiate new connections to the web server. A more precise query that examines only new connections is given in Figure 7. The new query correctly reports that only internal hosts can initiate HTTP connections.

```
QUERY DADDY FOR TCP 80 AND
 NOT FROM 192.168.1.*
 AND IN NEW;
# Addresses: 192.168.1.*
```

Figure 7: A better query for stateful firewalls.

Structure Analysis Tools

Although query-based testing tools can be a significant help to the system administrator, they are limited by the user's ability to construct a comprehensive set of useful queries. It is difficult to tell whether a set of queries tests every important behavior of the firewall. Furthermore, testing techniques often generate too much output for the user to easily distinguish dangerous vulnerabilities from desired behavior.

Another approach to firewall analysis is to look for errors in the structure of the policy specification. Structure analysis tools [1, 8] detect problems such as duplicate or conflicting rules. Although these tools do not directly identify vulnerabilities, they often uncover fundamental weaknesses in the policy that can produce more significant errors. Some of these tools also generate a simpler version of the policy that removes these structural weaknesses. The generated policy is often easier to inspect manually than the original policy.

One significant advantage of structure analysis tools is that they can be fully automatic. The only input the user must provide is the firewall policy itself. The tool builds a list of anomalies and outputs a report or a restructured version of the policy. Unfortunately, there are many types of vulnerabilities that cannot be detected using these tools. For instance, allowing mail traffic from the outside world to certain workstations could be undesirable behavior on some networks. A structure analysis tool would not detect a problem of that nature unless the rule that permitted the flow of such traffic also conflicted with another rule or violated the structural criteria in some other way.

Host Classification

The "Lumeta Firewall Analyzer" [18], a commercially available tool derived from FANG [15], combines some of the advantages of a structure analysis tool with the flexibility of a passive analysis tool. Lumeta automatically generates a comprehensive set of queries by using routing information to classify hosts into groups [18]. This reduces the amount of output since results can be provided on a per-subnet or per-zone basis rather than a per-host basis. It also removes the burden of designing good queries from the user.

The idea of classifying hosts into groups allows a query engine to provide much simpler output and addresses the problem of creating good queries. Using the topology of the network to classify hosts, however, has the drawback that hosts with very different properties, but that have similar addresses, are grouped together.

Consider, for instance, the filtering policy shown in Figure 6. This simple policy restricts outgoing mail from an internal network 10.239.202.0/24. Outgoing traffic is only allowed from the mail server, host 10.239.202.38. Other hosts on the subnet are allowed to send mail to the mail server, but cannot send mail to each other or to the outside world. Incoming SMTP mail traffic from the outside world is also dropped unless it is destined for the mail server.

A classification based on the network topology would break the network into two groups: the set of hosts on 10.239.202.0/24 and the set of all other hosts. However, the mail server is a different type of host from the other machines on the network. As a result, queries about mail traffic will return imprecise results. For instance, the answer to the query "Can a host in 10.239.202.0/24 send mail to the outside world?" will be "yes" since the mail server is allowed to forward mail through the firewall. The query "Can all hosts in 10.239.202.0/24 send mail to the outside world?" will be "no" since the client machines are not allowed to send mail to anywhere but the mail server.

Neither of these queries accurately describes the fundamental organization of the network: a special mail server which can send mail to the outside world and a set of clients which cannot. To improve the precision of these queries, we must use a different classification scheme that allows us to group hosts by their function as well as by their placement in the network topology.

Policy-Based Host Classification

Hosts on a network play a variety of roles. Some hosts are workstations. Some are database servers. Some are web servers. Some provide multiple services. Each type of host is often treated very differently by the firewall. Hosts of the same type, however, are usually treated very similarly. This means that the firewall implicitly classifies hosts into various groups

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	DROP	192.168.2.0/24	192.168.1.0/24	
2	ACCEPT	Anywhere	192.168.1.1	dpt tcp:80

Figure 8: A simple network with four host classes.

based on their function. Sometimes the implicit classification of the firewall policy is not quite as straightforward as simply sorting hosts by the services they provide. For instance, the network may have a web server that provides service exclusively to hosts inside the network as well as a general purpose web server that anyone can access. The filtering policy for these two systems could be drastically different even though they are both web servers.

The rule set in Figure 8 prevents hosts on an untrusted network 192.168.2.0/24 from accessing systems on a protected network 192.168.1.0/24. Rule 1 divides the set of hosts into three groups. One group consists of hosts in the untrusted network. The second group contains hosts from the protected network. The third group contains all other hosts on the Internet. Rule 2 refines this classification by further restricting which services are available to the web server. This distinguishes the web server from other trusted machines, creating a fourth group containing only the web server.

This classification scheme has many advantages over a topological classification. An error in the firewall policy will often cause the firewall to treat similar hosts differently or to treat different hosts alike. This means that a classification scheme based on the structure of the firewall policy can be used to directly detect many kinds of errors. Furthermore, classifying hosts according to their treatment by the firewall produces groups of hosts that can be used to increase the precision of query-based testing techniques.

Calculating Host Classes

There are several possible ways to use the structure of the firewall policy to create classes of hosts. A naive approach is to search through the firewall policy and record every group of addresses that is explicitly mentioned as a host class.

Unfortunately, the naive approach will generate overlapping classes. For instance, the host 192.168.1.1 from Figure 8 would be represented twice: once in its own class and once in the class containing all hosts from the 192.168.1.0/24 subnet. This is undesirable because it decreases the precision we can obtain in our queries. A host that appears in two classes is fundamentally different from the other hosts in those classes and should be categorized separately in order to obtain accurate results.

The algorithm in Figure 9 reduces the amount of overlap by splitting overlapping classes into smaller pieces. The algorithm examines every host and set of addresses mentioned explicitly in the rule set. Each new range of addresses is added to a set of potential classes, *C*.

```
set CalculateClasses(Policy P):
1 set C = {0.0.0.0/0}
2 for each rule r in P:
3 for each addr_range S in r:
4 C = InsertAddr(C, S)
5 return C

set InsertAddr(set C, addr_range S):
1 for each element T of C:
2 I = IntersectAddress(S, t)
2 if I is empty:
3 C = SetAdd(C, S)
4 return C
5 C = SetDelete(C, T)
6 C = InsertAddr(C, S-I)
7 C = InsertAddr(C, I)
8 C = InsertAddr(C, T-I)
9 return C
```

Figure 9: A naive algorithm for computing host classes.

If a new set of addresses overlaps with an existing class, we break both classes into three non-overlapping pieces and replace both original classes with the result. When we have considered every address of every rule, the elements of *C* describe a set of classes that can be used to analyze the behavior of the firewall.

This approach yields an approximation of the firewall designer's view of the network. Addresses that are explicitly mentioned usually correspond to important components that the designer intended to control. Unfortunately, the technique does not give a perfect picture of the actual behavior of the firewall. For instance, the firewall rule set in Figure 10 seems at first glance to have three groups. The algorithm will create a group for subnet 192.168.2.0/24 and for subnet 192.168.3.0/24. It will also create a group representing "all other addresses."

In reality, hosts on the 192.168.3.0/24 subnet are treated no differently from hosts in the "all other addresses" group, because Rule 3 of the firewall policy is an unreachable rule. Because all packets from 192.168.2.0/24 will be dropped in Rule 1, no packet can ever match Rule 3. This is probably an error in the firewall configuration, but the naive algorithm will happily report that, as expected, 192.168.3.0/24 is a special class and the user will not detect the error.

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	DROP	192.168.2.0/24	Anywhere	
2	ACCEPT	Anywhere	192.168.2.0/24	
3	ACCEPT	192.168.2.0/24	192.168.3.0/24	

Figure 10: Rule set with a shadowed network.

To correct this problem we need to more carefully define the concept of a host class. We do this by constructing an equivalence relation over the set of all network hosts. The equivalence classes determined by this relation will give us a precise and complete characterization of the policy that we can use for performing vulnerability analysis.

Structure-Based Classification

Every firewall policy can be described as a function, F, that maps the set of all network packets to the set $\{ACCEPT, DROP\}$ of filtering decisions. For a specific packet p, we say $F(p) = ACCEPT$ if the packet would be accepted by the firewall and $F(p) = DROP$ if the packet would be dropped by the firewall.

We define an equivalence relation, $\equiv_{SD}$, as follows: let x and y be any two hosts. We say that $x \equiv_D y$ if and only if for any two packets p from x and q from y that differ only by source address, $F(p) = F(q)$. Similarly, $x \equiv_D y$ if and only if $F(p) = F(q)$ for any two packets p from x and q from y that differ only by destination address. If $x \equiv_S y$ and $x \equiv_D y$, then we say that $x \equiv_{SD} y$.

Informally, two hosts are source equivalent if replacing the source address of a packet from one host with the source address of the other does not affect the filtering decision of the firewall. They are destination equivalent if replacing the destination address does not affect the filtering decision. If they are both source and destination equivalent, we say that they are equivalent under the relation $\equiv_{SD}$. The relation $\equiv_{SD}$ is derived directly from the function F, which describes the filtering policy of the firewall and can be computed without any other input from the user.

It can be shown that $\equiv_{SD}$ is an equivalence relation, since it is reflexive, transitive, and symmetric. This means that $\equiv_{SD}$ partitions the set of network hosts into equivalence classes. In other words, a packet from a host in a particular equivalence class will only be accepted if identical packets from other hosts in the class would also be accepted.

This means that if one host in the class has a vulnerability, all hosts in the class are vulnerable. On the other hand, if that host is adequately protected by the firewall, then all the others are too. This guarantee makes the equivalence class paradigm much more useful than the naive classification algorithm or a classification based on topology for generating precise queries.

Implementation

In previous work [14], we described an algorithm for representing the rule set of an *iptables* firewall as a multi-way decision diagram (MDD). We can use the reduction properties of the MDD to compute the equivalence classes of the firewall.

An MDD is a directed acyclic graph that can efficiently represent a function over a large set of vectors. An MDD representation of the rule set of a firewall can be obtained by converting the rule set into a function over the set of all network packets. An example MDD is depicted in Figure 11. Each of the non-terminal levels of the MDD corresponds to one field of a packet. For space and simplicity, we have greatly simplified the example by considering only a few of the fields. In practice, the MDD is much larger and contains levels for such values as the protocol, the source port, the destination port, and the TCP flags.

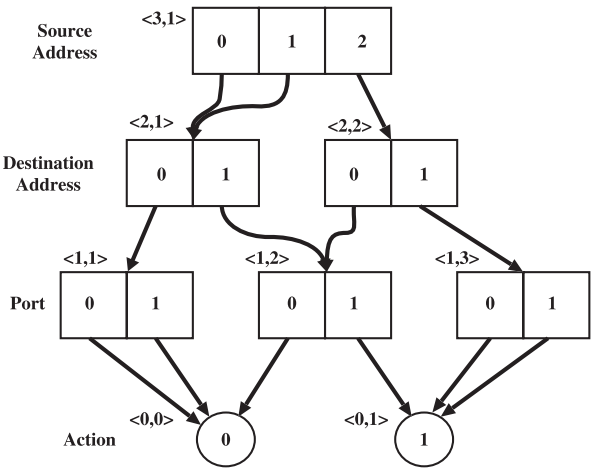


Figure 11: A simplified rule set MDD.

The top level of the MDD in Figure 11 corresponds to the source address of a packet, while the second level corresponds to the destination address of the packet. The bottom level is a special terminal level representing the action that the firewall should take on a packet. The integer value 0 means to drop the packet. The integer value 1 means to accept the packet.

Each path through the MDD represents one filtering rule. Each node represents a set of related packets. We use the notation $< k, i >$ to denote the i^{th} node of level k. Each arc at level k represents a choice of value for attribute k of the packet. For example, the first arc of node $< 3, 1 >$ represents a source address value of 0 and node $< 2, 1 >$ of Figure 11 represents the set of all packets with source address 0 or 1. The path $< 3, 1 >$ to $< 2, 2 >$ to $< 1, 2 >$ to $< 0, 1 >$ represents the rule that any packet with source address 2, destination address 0, and destination port 1 will be accepted by the firewall.

1. Construct the MDD representation of each firewall chain.
2. For each chain:
3. Reorder the levels of the chain MDD so that source address is on top.
4. Record the source equivalence classes.
5. Reorder the levels of the chain MDD so that destination address is on top.
6. Record the destination equivalence classes.
7. Merge the source and destination classes of all three chains together.

Figure 12: Outline of the equivalence class computation algorithm.

In ITVal, we use reduced-ordered MDDs in which duplicate nodes, with all arcs the same, are not allowed. This requirement means that each node at level k represents an equivalence class over the set of attributes K through $k+1$, where level K is the top level of the MDD. For instance, node $< 2, 1 >$ represents the source equivalence class containing addresses 0 and 1. Node $< 2, 2 >$ represents the class containing source address 2. By reordering the levels of the MDD, we can calculate equivalence classes over first the source address and then the destination address. We can use these intermediate classes to construct classes of hosts that are equivalent under the $\equiv_{SD}$ relation. An outline of the class generation procedure is given in Figure 12. A more detailed illustration of the algorithm is given in Figure 13.

In step 1, we generate an MDD representation for each of the three built-in chains. The MDD representation takes into consideration network address translation and other packet mangling rules. We then consider each chain in turn. In steps 3 and 4, we compute a list of source equivalent addresses. To do this, we first use a level swapping algorithm to bring the levels encoding source address to the top of the graph. The reduction properties of the MDD now guarantee that each node at the level immediately below the source address levels represents an equivalence class with respect to source address. Each path from the root node to a node at that level represents one element of the equivalence class associated with that node. Step 4 extracts these equivalence classes and stores them in a new MDD.

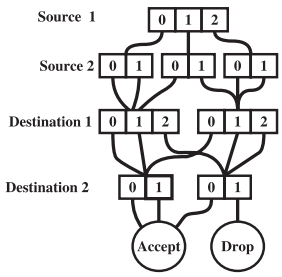
In steps 5 and 6 we perform an identical operation to collect a list of equivalence classes with respect to destination address. When we have considered source and destination address in every chain, we now merge the various classes together using MDD union, intersection and difference operators. Finally, we print the result.

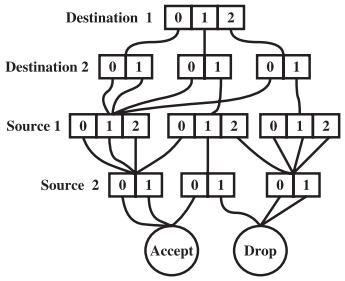
Error Detection

The information provided by the host classification algorithm can be extremely useful for detecting errors in the firewall policy. The list of classes is usually much shorter and simpler than the rule set, so it is easier for a system administrator to examine.

Detecting Remotely Accessible Services

Simple errors such as typos and transpositions can often be detected by the presence of a strange and unexpected class of hosts. The policy in Figure 14 is





Source Class 0: 0.0, 0.1, 1.0 Source Class 1: 1.1, 2.0, 2.1	Destination Class 0: 0.0,0.1,1.0,2.0 Destination Class 1: 1.1 Destination Class 2: 2.1	Host Class 0: 0.0,0.1,1.0 Host Class 1: 1.1 Host Class 2: 2.0 Host Class 3: 2.1
In the initial MDD, the two source address fields are at the top, so we do not need to reorder the levels. The level below the source addresses is the "Destination 1" level. Each node at that level defines a source equivalence class. To find the members of that class, we simply enumerate all paths from the root node to the node at that level.	We now reorder the MDD so that the destination address is at the top, followed by the source address levels. The accept and drop nodes remains at the bottom. Each node at level "Source 1" now defines a destination equivalence class. The members of each class can be found by collecting all paths from the root to the nodes at that level	We construct the final host classes by merging and splitting the source and destination classes. For each source class and destination class, we create three new sets. One set is constructed from the intersection of the two classes. Another consists of addresses in the destination class, but not in the source class. The last set contains the source addresses not in the destination class. Empty classes are discarded.

Figure 13: Step by step construction of the equivalence classes.

intended to protect networks 192.168.1.0/24, 192.168.2.0/24, and 192.168.3.0/24 by restricting access from the outside world. Because 192.168.1.0/24 contains several hosts with important financial information, outgoing traffic from that subnet should also be restricted. Mail traffic from the other subnets is allowed only to the mail server (host 192.168.2.20) to prevent compromised machines from becoming spam relays. The rule set also allows arbitrary web access to a group of web application servers located on the 192.168.3.0/24 subnet. The policy contains several errors, including a typo in Rule 3 that allows remote access to a protected service.

The equivalence classes of this example network are listed in Figure 16. There are five classes of hosts identified by the algorithm. Class 0 represents the group of web servers. Class 1 represents a strange class of hosts that exists because of the typo in rule 3. The strange, unexpected class makes the effect of the typo immediately obvious to the administrator.

```
Class 0: 192.168.3.*
Class 1: 168.192.1.*
Class 2: 192.168.1.*
         192.168.2.[0-19]
         192.168.2.[21-255]
Class 3: 192.168.2.20
Class 4: [0.0.0.0]-[168.192.0.255]
         [168.192.2.0]-[192.168.0.255]
         [192.168.4.0]-[255.255.255.255]
```

Figure 16: Equivalence classes for Figure 14.

Class 2 combines the protected financial network and the unprotected 192.168.2.0 network, minus the mail server. This should also arouse the analyst's suspicion since the financial network is supposed to have much stricter protection than the unprotected subnet. The fact that they are treated the same by the firewall indicates that a serious vulnerability exists. Class 3 contains the mail server. It is in a class by itself since it requires special privileges in order to accept and relay mail. Everything else belongs to Class 4.

Using the equivalence classes to detect these errors is much easier than using query based tools. The presence of a class of hosts consisting entirely of strange addresses is a clear indication of an error in the policy. Since the tool requires no input but the policy, all the user has to do to discover the error is "fire and forget."

A small amount of work is required to interpret the results of the classification system, but compared to the effort of constructing precise queries or compiling a list of hosts for active testing, using the equivalence classes is really fairly simple. For large installations, the gain is even greater due to the number of rules required to administer a large number of hosts and the greater difficulty of specifying a comprehensive set of queries that covers all the services provided by the network.

Detecting Shadowed Rules

If a packet matches more than one rule in the policy, the firewall will use the first rule that matches. This can mean that the policy contains useless or unreachable rules. The presence of these rules usually indicates an error in the policy.

When one rule shadows another, the class list will often contain fewer classes than expected. For instance, the rule set in Figure 15 contains two rules that are shadowed by Rule 1. Rule 2 is a useless rule. Web packets from 192.168.2.0/24 to 192.168.3.0/24 are already accepted by Rule 1. Rule 3 is also unreachable. The class list for the example network is given in Figure 19.

```
Class 0: 192.168.2.*
Class 1: [0.0.0.0]-[192.168.1.255]
         [192.168.3.0]-[255.255.255.255]
```

Figure 19: Equivalence classes for Figure 15.

Notice that there are no classes for the networks 192.168.3.0/24 and 192.168.4.0/24 mentioned in rules

FORWARD (Default DROP)				
#	**Target**	**Source Address**	**Destination Address**	**Options**
1	ACCEPT	Anywhere	192.168.3.0/24	TCP dpt:80
2	ACCEPT	192.168.2.0/24	Anywhere	
3	DROP	168.192.1.0/24	Anywhere	
4	ACCEPT	Anywhere	192.168.2.20	TCP dpt:25
5	ACCEPT	192.168.1.0/24	Anywhere	

Figure 14: Rule set with errors.

FORWARD (Default DROP)				
#	**Target**	**Source Address**	**Destination Address**	**Options**
1	ACCEPT	192.168.2.0/24	Anywhere	TCP dpt:80
2	ACCEPT	192.168.2.0/24	192.168.3.0/24	TCP dpt:80
3	DROP	192.168.2.0/24	192.168.4.0/24	TCP dpt:80

Figure 15: Rule set with shadowed rules.

2 and 3. When the system administrator discovers that the policy produces fewer classes than expected, she will examine the policy more closely and detect the error. Shadowed rules often indicate that a rule contains an incorrect address. For instance, rules 2 and 3 may have become shadowed by a typo in Rule 1. This could happen if the source address in Rule 1 was supposed to be 192.168.3.0/24, but was typed incorrectly.

Detecting Outdated Services

Host classification can solve real world problems. One of our firewalls originally supported a wireless network on subnet 192.168.4.0/24. When wireless service was transferred to another network, we neglected to update the firewall rules. A portion of our rule set looked something like Figure 17. A quick analysis using host classification immediately identified subnet 192.168.4.0 as a host group, enabling us to correct the problem. This error would have been very difficult to detect using query-based analysis tools. Without apriori knowledge of the error, we had no reason to create a query testing for service on that subnet. Active analysis tools like Nessus would have detected no vulnerabilities, since no hosts were available on that subnet. Using host classification, however, we were able to immediately identify a serious weakness in our policy.

Using Equivalence Classes with Other Tools

While a system administrator can detect many important vulnerabilities simply by studying the host equivalence classes of a firewall policy, even greater gains can be achieved by combining the equivalence class analysis with active and passive testing techniques. To combine the analysis with other testing paradigms, we can use the equivalence classes to determine which systems to test. By taking one or two systems from each equivalence class, we can increase the probability that we have tested all the important behaviors of the firewall.

The filtering policy in Figure 18 secures the mail service on an internal network 192.168.2.0/24. Mail from the internal network can only be sent to the mail server, host 10.239.202.13 . The mail server is allowed to distribute mail to both internal and external hosts. All other mail traffic should be dropped. Unfortunately, a copy and paste error created rule 3 of the policy, which allows mail traffic from a workstation, host 192.168.2.3 to escape the network. If that workstation is compromised, an intruder can set up a spam relay on that host and transmit thousands of unauthorized messages through the firewall.

```
Class 0: 10.239.202.13
Class 1: 192.168.2.3
Class 2: 192.168.2.[0-2]
         192.168.2.[4-255]
Class 3: [0.0.0.0]-[10.239.202.13]
         [10.239.202.14-192.168.1.255]
         [192.168.3.0-255.255.255.255]
```

Figure 20: Equivalence classes for Figure 18.

The system administrator can easily detect this problem by combining host classification with a passive testing tool. The host classes for the example network are listed in Figure 20. By taking a source address from each of these groups and matching it with a destination address from each of the groups, we can construct the set of 16 ITVal queries listed in Figure 21.

Notice that while the list of classes does not immediately cause concern, the query results show that 192.168.2.3 can access port 25 on host 192.168.3.0, which is in the ‘‘outside world’’ class of hosts. This strategy can identify serious problems without producing an overwhelming amount of input. Using the query tools by themselves would either have produced an enormous amount of data or required a large time investment in writing queries. Combining classification with passive testing limits the scope of

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	192.168.2.0/24	Anywhere	TCP dpt:22
2	ACCEPT	192.168.4.0/24	Anywhere	TCP dpt:8080
3	DROP	192.168.4.0/24	192.168.2.0/24	TCP dpt:25

Figure 17: Rule set with outdated rules.

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	Anywhere	10.239.202.13	TCP dpt:25
2	ACCEPT	10.239.202.13	Anywhere	TCP dpt:25
3	ACCEPT	192.168.2.3	Anywhere	TCP dpt:25
4	ACCEPT	192.168.2.3	Anywhere	TCP dpt:22
5	DROP	192.168.2.0/24	Anywhere	TCP dpt:25

Figure 18: Rule set for preventing spam relays.

the query to the important distinctions between hosts and requires very little work from the user.

Author Biographies

Robert Marmorstein plans to graduate with a Ph.D. from the College of William and Mary in the summer of 2007. When he is not actively researching ways to manage and analyze firewalls, he spends his time avoiding grues in the Great Underground Empire and tinkering with his collection of UNIX-based systems.

Phil Kearns is an Associate Professor of Computer Science at the College of William and Mary. His research interests lie in the general area of computer systems.

Conclusion

Policy-based host classification has several significant advantages over existing firewall analysis techniques. Examining the classes implicitly defined by the firewall policy allows a system administrator to detect many kinds of firewall errors and anomalies. When combined with active or passive testing tools, the technique can be even more powerful. Using the equivalence classes significantly decreases the amount of the work required to verify the policy and is a step toward a fully automatic firewall analysis solution. The latest version of ITVal, provides a special "CLASSES" query that displays a list of the host equivalence classes. The tool and some examples of its use can be found at http://itval.sourceforge.net .

References

[1] Al-Shaer, Ehab S. and Hazem H. Hamed, "Modeling and management of firewall policies," *Transactions on Network and Service Management*, April, 2004.

[2] Barisani, Andrea, "Testing firewalls and IDS with ftester," In *Insight, Newsletter of the Internet Security Conference*, Vol. 5, 2001, http://www.tisc2001.com/newsletters/56.html.

[3] Bogaerts, Philippe, *HPING tutorial*, August, 2003, http://www.radarhack.com/dir/papers/hping2_v1.5.pdf .

[4] Eronen, Pasi and Jukka Zitting, "An expert system for analyzing firewall rules," *Proceedings of the 6th Nordic Workshop on Secure IT Systems*, 2001.

[5] Eychenne, Herve, *iptables man page*, March, 2002.

[6] Farmer, Dan and Wietse Venema, *SATAN: Security Administrator's Tool for Analyzing Networks*, 1995, http://www.fish.com/zen/satan/ .

[7] Fyodor, "The art of port scanning," *Phrack*, Vol. 7, Num. 51, September, 1997.

[8] Gouda, Mohamed G. and Alex X. Liu, "Firewall design: Consistency, completeness, and compactness," *Proceedings of the International Conference on Distributed Computing Systems*, IEEE Computer Society, March, 2004.

[9] Internet Security Systems, *Internet Scanner User Guide Version 7.0 SP 2*, 2005, http://documents.iss.net/literature/InternetScanner/IS_UG_7.0_SP2.pdf .

[10] Lampe, John, *Nessus 3.0 Advanced User Guide*, October, 2005, http://www.nessus.org .

[11] Liu, Alex X., Mohamed G. Gouda, Huibo Heidi Ma, and Anne H. Ngu, "Firewall queries," *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS-04), LNCS 3544*, Springer-Verlag, December, 2004, http://www.cs.utexas.edu/users/alex/publications/FirewallQueries/query.pdf .

[12] Marmorstein, Robert, *ITVal Project Website*, 2005, http://itval.sourceforge.net .

[13] Marmorstein, Robert and Phil Kearns, "An open source solution for testing NAT'd and nested iptables firewalls," *19th Large Installation Systems Administration Conference (LISA '05)*, pp. 103-112, December, 2005.

QUERY DPORT TO 10.239.202.13 AND FROM 10.239.202.13 AND IN NEW; **1 Port: 25**	QUERY DPORT TO 10.239.202.13 AND FROM 192.168.2.3 AND IN NEW; **2 Ports: 22 25**	QUERY DPORT TO 10.239.202.13 AND FROM 192.168.2.1 AND IN NEW; **1 Port: 25**	QUERY DPORT TO 10.239.202.13 AND FROM 192.168.3.0 AND IN NEW; **1 Port: 25**
QUERY DPORT TO 192.168.2.3 AND FROM 10.239.202.13 AND IN NEW; **1 Port: 25**	QUERY DPORT TO 192.168.2.3 AND FROM 192.168.2.3 AND IN NEW; **2 Ports: 22 25**	QUERY DPORT TO 192.168.2.3 AND FROM 192.168.2.1 AND IN NEW; **0 Ports:**	QUERY DPORT TO 192.168.2.3 AND FROM 192.168.3.0 AND IN NEW; **0 Ports:**
QUERY DPORT TO 192.168.2.1 AND FROM 10.239.202.13 AND IN NEW; **1 Port: 25**	QUERY DPORT TO 192.168.2.1 AND FROM 192.168.2.3 AND IN NEW; **2 Ports: 22 25**	QUERY DPORT TO 192.168.2.1 AND FROM 192.168.2.1 AND IN NEW; **0 Ports:**	QUERY DPORT TO 192.168.2.1 AND FROM 192.168.3.0 AND IN NEW; **0 Ports:**
QUERY DPORT TO 192.168.3.0 AND FROM 10.239.202.13 AND IN NEW; **1 Port: 25**	QUERY DPORT TO 192.168.3.0 AND FROM 192.168.2.3 AND IN NEW; **2 Ports: 22 25**	QUERY DPORT TO 192.168.3.0 AND FROM 192.168.2.1 AND IN NEW; **0 Ports:**	QUERY DPORT TO 192.168.3.0 AND FROM 192.168.3.0 AND IN NEW; **0 Ports:**

Figure 21: Queries auto-generated using host classes.

[14] Marmorstein, Robert and Phil Kearns, ''A tool for automated iptables firewall analysis,'' *FREENIX Track, 2005 USENIX Annual Technical Conference*, pp. 71-82, April, 2005.

[15] Mayer, Alain, Avishai Wool, and Elisha Ziskind, ''Fang: A firewall analysis engine,'' *Proceedings of the IEEE Symposium on Security and Privacy*, May, 2000.

[16] Quinton, Reg, *Using Solaris ipfilters*, http://ist.waterloo.ca/security/howto/2005-08-19/paper.pdf.

[17] Singer, Abe, ''Life without firewalls,'' *;login:*, Vol. 28, Num. 6, pp. 27-30, December, 2003.

[18] Wool, Avishai, ''Architecting the Lumeta firewall analyzer,'' *Proceedings of the 10th USENIX Security Symposium*, August, 2001.