ROBERT MARMORSTEIN AND
PHIL KEARNS

debugging a firewall policy with policy mapping

Robert Marmorstein will graduate from the College of William and Mary this summer with a Ph.D. in Computer Science. When he is not actively researching ways to manage and analyze firewalls, he spends his time avoiding grues in the Great Underground Empire.

rmmarm@cs.wm.edu

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.

kearns@cs.wm.edu

IF YOU MANAGE A LARGE NETWORK, chances are good that lurking somewhere in your firewall policy is an error that could compromise the security of your network. Firewalls are subject to many different kinds of configuration errors. Although many of these glitches are relatively harmless, some can seriously compromise your policy's correctness and reliability. Inserting a rule into the policy twice may cause a negligible performance hit but usually does not affect the correctness or security of the policy. However, a typo in a critical rule may give an untrusted host access to your important servers. Until recently, debugging a firewall policy with more than a few rules was a challenging and tedious task. In the past few years, however, new tools for analyzing the policy have made finding firewall gremlins much easier. With these tools, you can find and repair many common firewall problems quickly and easily.

ITVal is a framework we designed for testing iptables-based Linux firewalls. It was originally intended to provide an open-source alternative to existing commercial testing tools such as the Fang firewall analysis engine [7, 9] and the Internet Security Scanner [3]. Like those tools, it relied either on the user's ability to create logical queries describing the desired behavior of the firewall or on a pregenerated set of generic tests. We quickly discovered, however, that devising meaningful and effective queries required nearly the same effort as inspecting the firewall by hand. To address this problem, we came up with a new method of debugging the firewall rule set.

Our new technique is easily usable by any system administrator. It doesn't need extensive preparation of queries, cases, or tests. In fact, the only input requirement is a textual representation of the policy (easily generated by the `iptables -L -v -n` command). As output, it produces a map of the network that can be used for detecting anomalies in the policy.

The policy map divides hosts into groups based on their interaction with the firewall. If the firewall treats two hosts the same, they will belong to the same group. If they are treated differently, they belong to different groups. A correct policy will

usually divide the hosts of a network into logical, easily identifiable groups based on their function within the network. One group might be the "all workstations" group. If the network distinguishes between different kinds of workstations, there may, instead, be separate groups for "Solaris workstations" and "Linux workstations." Other groups might be the "Web servers" group and the "wireless hosts" group.

It is significant that the map generated by the component of ITVal described in this article does not rely upon user input. The map is generated by processing only the iptables rules set that defines the firewall policy. In a very real sense, it is merely a different representation of the set of iptables rules, but we contend that it makes the difficult task of finding some errors in firewall configuration much easier.

Since you probably have a good intuitive idea of the various types of hosts on your network, it is relatively easy to check that the firewall policy map represents a correct policy. The policy map probably won't reveal every error in your policy, but it will make many of the most significant errors instantly visible. A quick glance at the policy map will reveal significant bugs in the firewall policy that query-based tools could not easily uncover.

Debugging a Firewall

To create a policy map, you need to create a few important input files. The first step is to dump a copy of your firewall policy to disk. If you have root access, this can be done simply by typing `iptables -L -n -v > myRules` at the command line. If your firewall uses packet mangling, you also need to dump the NAT table to disk with `iptables -t NAT -L -n -v > myNatRules`.

You also need to provide a query file, myQuery, containing the single statement `QUERY CLASSES;`. This tells ITVal to generate the policy map for the input firewall.

You can now use the command `ITVal -F myRules -N myNatRules -q myQuery` to generate the policy map. ITVal will create a list of the various host groups in the firewall policy and display them on stdout.

The firewall policy in Table 1 protects subnet 128.30.40.0/24 from the outside world. The network has a few key servers: a mail server (128.30.40.10) and two identical Web servers (128.30.40.11–12). Hosts on the network also talk to two special machines outside the network: a name server (128.30.1.128) and a network time server (64.15.175.5). The policy contains several errors, which can easily be detected by inspecting the policy map.

	Target	Source	Destination	Port
1	ACCEPT	Anywhere	128.30.40.0/24	DNS
2	ACCEPT	128.30.40.0/24	Anywhere	DNS
3	ACCEPT	128.30.40.0/24	64.15.175.5	NTP
4	ACCEPT	128.30.40.0/24	128.30.40.0/24	
5	DROP	!128.30.40.0/24	128.30.40.13	
6	ACCEPT	128.30.40.0/24	218.30.40.10	SMTP
7	ACCEPT	128.30.40.0/24	128.30.40.10	IMAP
8	ACCEPT	128.30.40.0/24	128.30.40.12	SSH
9	ACCEPT	Anywhere	128.30.40.11	HTTP
10	ACCEPT	Anywhere	128.30.40.12	HTTP

TABLE 1: A BUGGY FIREWALL POLICY

This policy is intended to enforce a few important rules. We want to allow mail traffic only between trusted clients and the mail server. (This is unre-

alistic, since the mail server will also need to send and receive messages from the outside world, but it makes the example much simpler.) We also want both Web servers to allow HTTP connections from any host and SSH connections from clients on the trusted subnet. Furthermore, any of our systems should be able to access domain name service and the network time service, but only from appropriate servers. Using ITVal, we can generate a map of this firewall policy. The policy map for the firewall in Table 1 looks like this:

```
QUERY CLASSES; There are 6 host classes:
Class1: #Untrusted Hosts and DNS Server
        <Everything not explicitly listed in the other classes>
Class2: #NTP Server
        64.15.175.5
Class3: #Trusted Network, including the Mail Server
        128.30.40.[0–10]
        128.30.40.[14–255]
Class4: #Web Servers
        128.30.40.[11–12]
Class5: #Anomalous Host 128.30.40.13
        128.30.40.13
Class6: #Anomalous Host 218.30.40.10
        218.30.40.10
```

The policy map consists of a list of various classes of hosts, which correspond to the different types of systems on the network. Each class consists of a set of hosts with some common properties. Two hosts belong to the same class if and only if any packet sent or received by one of them is treated the same as if it had come from (or to) any of the others. If two hosts are in different classes, there is some essential difference between them in the firewall policy that distinguishes them from each other.

In the listing given here, each element of a class is given as an address or a range of addresses. For instance, in class 4, the element 128.30.40.[11–12] represents a set containing the addresses 128.30.40.11 and 128.30.40.12. A brief inspection of the various classes enables you to easily identify their members. For instance, class 2 corresponds to the external NTP server. Class 3 represents the trusted hosts of the network. Class 4 represents the Web servers of the network. Classes 5 and 6 are anomalous. Class 5 consists of a decommissioned print server that no longer belongs on the network. Class 6 is an artificial class caused by an error in the firewall. All other hosts belong to class 1. For easier reference, we have added comments to the output that identify each class.

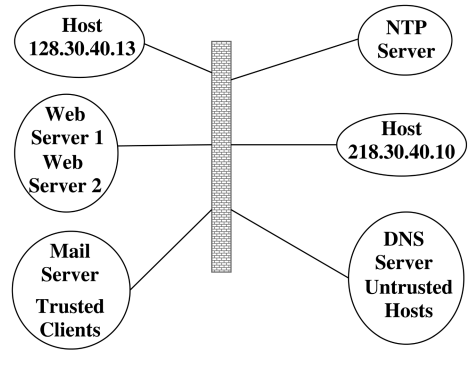

FIGURE 1: AN ITVAL NETWORK MAP

Figure 1 is a graphical depiction of the policy map. By inspecting the map for anomalies, you can detect several important policy errors.

Catching Typos

In a policy of more than a few dozen rules, there is a good chance that one of the rules contains a typo. In the sample policy, a typo on line 6 prevents SMTP traffic from reaching the mail server. The policy map immediately highlights this error. Since 218.30.40.10 doesn't correspond to any of the important servers, the existence of class 5 is a clear indication of an error in the policy. A search through the rule set for the address 218.30.40.10 uncovers the typo and allows us to patch the problem by transposing the first two digits of the address. Repairing the problem gives us a new policy map. The new map looks like this:

```
QUERY CLASSES; There are 5 host classes:
Class1: #Untrusted Hosts and DNS Server
        <Everything not explicitly listed in the other classes>
Class2: #NTP Server
        64.15.175.5
Class3: #Trusted Network, including the Mail Server
        128.30.40.[0–10]
        128.30.40.[14–255]
Class4: #Web Servers
        128.30.40.[11–12]
Class5: #Anomalous Host 128.30.40.13
        128.30.40.13
```

Detecting Outdated Rules

The new policy contains yet another anomalous class. Host 128.30.40.13 in class 5 of the new policy map does not correspond to any of our important servers. Why has it been distinguished as a special class?

It is very easy to forget to change the firewall policy after altering the network infrastructure. The sample policy contains rules for an experimental print server that has been taken offline and no longer needs special protection from external hosts. In the meantime, the server's IP address has been reused as the address of a new workstation. As a result, the firewall makes a distinction between that address and the other systems. Rule 5 of the original policy, originally designed to protect the print server from untrusted hosts, is now blocking network traffic to the new workstation. Removing the outdated rule from the policy resolves this issue and gives us the following policy map:

```
QUERY CLASSES; There are 4 host classes:
Class1: #Untrusted Hosts and the DNS Server
        <Everything not explicitly listed in other classes>
Class2: #NTP Server
        64.15.175.5
Class3: #Trusted Network, including the Mail Server
        128.30.40.[0–10]
        128.30.40.[13–255]
Class4: #Web Servers
        128.30.40.[11–12]
```

Detecting Overly Broad Rules

Sometimes the easiest way to temporarily allow hosts to access an external service is to open that service up to any external host. This is a bad practice which often leaves a network open to intrusions from untrusted hosts,

but it is very convenient when bringing a new system online for the first time. In the long run, however, it is usually much better to use a more specific rule that allows only trusted hosts to provide that service.

The DNS server does not appear in any of the classes explicitly listed in the policy map. This is because the members of class 1, the "everything else" class, have been hidden to save space. Counter to our expectations, the DNS server has been lumped into this class with all of the untrusted hosts. To permit DNS traffic only from the appropriate server, the firewall policy ought to distinguish that server from other hosts outside the network. The fact that the DNS server is grouped with untrusted hosts indicates either that DNS traffic is allowed from any external host or that all DNS traffic is blocked by the firewall.

Examining the firewall policy reveals an error in rules 1 and 2. Those rules should grant DNS access only from the DNS server (128.30.1.128), and not from other external hosts.

This error can be easily repaired by inserting the correct IP address into each rule. DNS traffic will then be permitted only to the appropriate server. Running ITVal on the new policy gives us a new policy map:

```
QUERY CLASSES; There are 5 host classes:
Class1: #Untrusted Hosts
        <Everything not explicitly listed in the other classes>
Class2: #NTP Server
        64.15.175.5
Class3: #DNS Server
        128.30.1.128
Class4: #Trusted Network, including the Mail Server
        128.30.40.[0–10]
        128.30.40.[13–255]
Class5: #Web Servers
        128.30.40.[11–12]
```

Detecting Shadowed Rules

Another common error is to create a rule in the policy that shadows other rules. Since iptables considers rules in sequential order, a rule that accepts or drops packets from an entire subnet will take precedence over a narrower rule that occurs later in the chain. These shadowed rules can be easily identified in the policy map.

In the policy map, the mail server does not have its own class. Instead it is grouped with the trusted workstations on the network. This is a good sign that one or more of the rules protecting the mail server has been shadowed. By looking through the rule set for rules corresponding to the mail server and/or the trusted hosts, you can easily see that rule 4 shadows rules 6 and 7. It turns out that the rule is unnecessary and can be deleted. Removing the rule creates a new rule set with the following policy map:

```
QUERY CLASSES; There are 7 host classes:
Class1: #Untrusted Hosts
        <Everything not explicitly listed in the other classes>
Class2: #NTP Server
        64.15.175.5
Class3: #DNS Server
        128.30.1.128
Class4: #Trusted Clients
        128.30.40.[0–9]
```

128.30.40.[13–255]
Class5: #Mail Server
128.30.40.10
Class6: #Primary Web Server
128.30.40.11
Class7: #Secondary Web Server
128.30.40.12

The removal of rule 4 introduced two new classes into the policy map. As expected, there is a new class containing the mail server. The class containing the Web servers has also changed. The erroneous rule hid some discrepancies in how the two Web servers are treated by the firewall that need to be addressed. Now that the rule has been removed, these errors have become visible.

Detecting Missing Rules

In the new policy, the primary and secondary Web servers belong to separate classes. Why is this? Rules 8, 9, and 10 allow HTTP access to both servers, but they allow SSH access only to the secondary Web server. The policy is missing a rule that would permit SSH access to the primary server.

Inserting a new rule to fix this problem gives us the policy shown in Table 2, which has the following policy map:

QUERY CLASSES; There are 6 host classes:
Class1: #Untrusted Hosts
<Everything not explicitly listed in the other classes>
Class2: #NTP Server
64.15.175.5
Class3: #DNS Server
128.30.1.128
Class4: #Trusted Clients
128.30.40.[0–9]
128.30.40.[13–255]
Class5: #Mail Server
128.30.40.10
Class6: #Web Servers
128.30.40.[11–12]

	Target	Source	Destination	Port
1	ACCEPT	128.30.1.128	128.30.40.0/24	DNS
2	ACCEPT	128.30.40.0/24	128.30.1.128	DNS
3	ACCEPT	128.30.40.0/24	64.15.175.5	NTP
4	ACCEPT	128.30.40.0/24	128.30.40.10	SMTP
5	ACCEPT	128.30.40.0/24	128.30.40.10	IMAP
6	ACCEPT	128.30.40.0/24	128.30.40.11	SSH
7	ACCEPT	128.30.40.0/24	128.30.40.12	SSH
8	ACCEPT	Anywhere	128.30.40.11	HTTP
9	ACCEPT	Anywhere	128.30.40.12	HTTP

TABLE 2: A CORRECT FIREWALL POLICY

A graphical depiction of a map for the new policy is shown in Figure 2. Both Web servers now belong to class 6. The map now conforms to our expectations. There are separate classes for the mail server, the DNS server, and the NTP server. The Web servers are grouped into a single class. The set of trusted clients forms a group and all other systems are grouped as untrusted hosts.

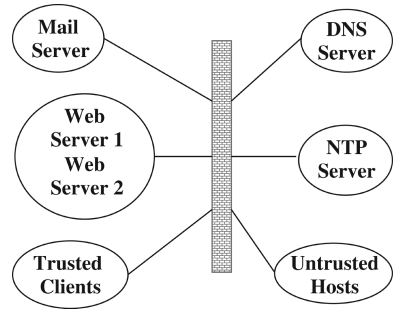

FIGURE 2: MAP OF THE NEW POLICY

Penetration Testing

In addition to making certain kinds of policy errors immediately obvious, the policy map can be used with other penetration testing techniques for more comprehensive and accurate verification. Since the firewall treats all hosts in a class the same, testing one address from each class gives complete coverage of the entire firewall policy. For instance, when using nmap [2] or hping2 [1] to search for unfiltered ports, it can be useful to use source address spoofing to test that the firewall rejects packets from a variety of sources. Instead of using a randomly selected source address, you can take one address from each class to be sure that all interesting behaviors of the firewall have been tested.

What If I Don't Use iptables?

ITVal currently only parses iptables firewalls, but the general technique of generating a policy map can be used with any type of firewall. A more technical description of policy mapping can be found in the proceedings of LISA '06 [6], which outlines the algorithm used to compute the various classes of the policy map. A quick-and-dirty approximation to the policy map can be created by listing all the addresses (and address ranges) explicitly mentioned in the firewall policy. While such a listing is far less precise and useful than the policy map, it can reveal some of the behaviors uncovered by the policy map and can provide a good sample of addresses to use during penetration testing. You might also try converting your rule set into an iptables policy using some of the scripts Bill Stearns has made available on his Web site [8] (mileage will vary!).

Conclusion

Maintaining a well-tested and tightly configured firewall is an important part of overall network security. Thanks to tools such as ITVal, it no longer needs to be an arduous task. By periodically testing whether your firewall policy conforms to your general expectations about the organization of your network, you can quickly and easily identify and repair significant firewall errors.

In addition to generating a policy map, ITVal provides many other useful tools for detecting problems in your firewall, including various ways to test for spoofing protection and to check whether viruses and Trojans can access backdoors through your firewall. More information about ITVal is available in the proceedings of Freenix '05 [5] and LISA '05 [4]. The tool itself can be downloaded from http://itval.sourceforge.net.

REFERENCES

[1] P. Bogaerts, HPING Tutorial, August 2003: http://www.radarhack.com/dir/papers/hping2_v1.5.pdf.

[2] Fyodor, "The Art of Port Scanning," *Phrack* 7, no. 51 (September 1997).

[3] 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.

[4] R. Marmorstein and P. Kearns, "An Open Source Solution for Testing NAT'd and Nested iptables Firewalls," in *19th Large Installation System Administration Conference (LISA '05)* (December 2005), pages 103–12.

[5] R. Marmorstein and P. Kearns, "A Tool for Automated iptables Firewall Analysis," in *FREENIX Track: 2005 USENIX Annual Technical Conference* (April 2005), pages 71–82.

[6] R. Marmorstein and P. Kearns, "Firewall Analysis with Policy-based Host Classification," in *20th Large Installation System Administration Conference (LISA '06)* (December 2006).

[7] A. Mayer, A. Wool, and E. Ziskin, "Fang: A Firewall Analysis Engine," in *Proceedings of the IEEE Symposium on Security and Privacy* (May 2000).

[8] B. Stearns, http://www.stearns.org/.

[9] A. Wool, "Architecting the Lumeta Firewall Analyzer," in *Proceedings of the 10th USENIX Security Symposium* (August 2001).