View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Archivio istituzionale della ricerca - Università degli Studi di Venezia Ca' Foscari

Firewall Management With FireWall Synthesizer

Chiara Bodei[1], Pierpaolo Degano[1], Riccardo Focardi[2],
Letterio Galletta[1], Mauro Tempesta[2], and Lorenzo Veronese[2]

[1] Dipartimento di Informatica, Università di Pisa, Italy
{chiara,degano,galletta}@di.unipi.it
[2] DAIS, Università Ca' Foscari Venezia, Italy
{focardi,tempesta}@unive.it, 852058@stud.unive.it

Abstract

Firewalls are notoriously hard to configure and maintain. Policies are written in low-level, system-specific languages where rules are inspected and enforced along non-trivial control flow paths. Moreover, firewalls are tightly related to Network Address Translation (NAT) since filters need to be specified taking into account the possible translations of packet addresses, further complicating the task of network administrators. To simplify this job, we propose FireWall Synthesizer (FWS), a tool that decompiles real firewall configurations from different systems into an abstract specification. This representation highlights the meaning of a configuration, i.e., the allowed connections with possible address translations. We show the usage of FWS in analyzing and maintaining a configuration on a simple (yet realistic) scenario and we discuss how the tool scales on real-world policies.

1 Introduction

Firewalls are fundamental mechanisms for the protection of computer networks. The effectiveness of a firewall system crucially depends on the correctness of its configuration, since an oversight in the policy may have dramatic effects on the security or the functionality of the entire network.

Firewall management is a complex task also for skilled system administrators. A typical configuration consists of a large number of rules and it is often hard to understand their effect on the overall firewall behavior. Moreover, from time to time, configurations must be adjusted to reflect the updates of the desired security policies. Since rules interact with each other, maintenance must be carefully carried out to avoid introducing unintended behaviors and subtle bugs that could expose the network to external threats.

Typically, policies are written in low-level configuration languages that are specific to the firewall system in use and support non-trivial control flow constructs, such as jumps and gotos. Network administrators must also keep in mind how packets are processed by the network stack of the operating system when writing their policy. Further complications come from Network Address Translation (NAT), the mechanism for translating addresses and performing port redirection, which modifies the packets while they traverse the firewall.

To simplify the work of system administrators, we have studied the problem of *decompiling* real configurations into abstract specifications that represent the set of the permitted connections. The abstract version of a firewall policy exposes the *meaning* of the configuration while discarding all its low-level details, so that it is easier to verify whether the intended security policy is correctly implemented. In addition, by comparing two abstract specifications, an administrator can spot the differences between the corresponding real configurations and check whether the updates have the desired effect on the firewall behavior.

Our tool FireWall Synthesizer (FWS) supports the above activities. It decompiles a real configuration into a human-readable form and analyzes it checking if it meets a given security

CEUR-WS.org/Vol-2058/paper-16.pdf

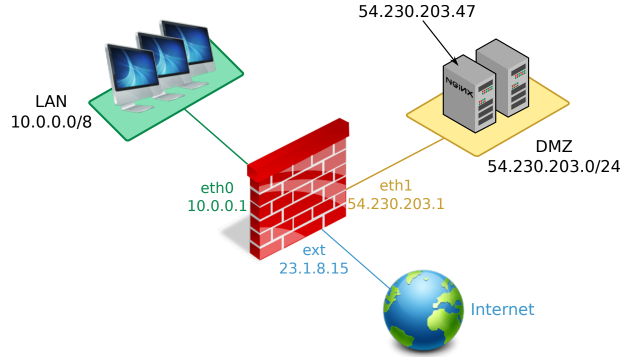

Figure 1: Network setup of our case study.

goal. Differently from the other tools and techniques proposed in the literature (see *e.g.*, [1, 3]), the novelty of FWS is that it does not target a specific firewall system. Indeed, FWS analyzes configurations for the most used firewall tools in Linux and Unix [5, 2, 4] and provides partial support for Cisco IOS routers. Furthermore, FWS can be easily extended to support new firewall systems by providing the corresponding front-ends. At [6] you can download FWS and the examples shown below. A full research paper describing the tool internals, the intermediate language, the algorithm and the underlying theory is presently submitted for presentation at an international conference.

2 FWS in Action

This section illustrates how FWS is used to analyze and manage real firewall configurations. For lack of space, we only consider the case of `iptables` but our approach can be directly applied to other systems. Besides synthesizing abstract specifications of the firewall policies, possibly projected using the query language provided by FWS, we show how the analyses implemented in the tool can be exploited while performing maintenance operations. These analyses include:

- *Reachability*: is a certain host reachable from another one, possibly through NAT?
- *Policy implication and equivalence*: are the packets accepted by one configuration at least/exactly those accepted by a different configuration?
- *Policy difference*: what are the packets accepted by one configuration and denied by another one?
- *Related rules*: which rules affect the destiny of the packets satisfying a given query?

Case study

We consider the simple (yet realistic) network setup shown in Figure 1:

- the LAN network is assigned the set of private IP addresses 10.0.0.0/8;
- servers and production machines, including the HTTPS web server with address 54.230.203.47 are placed in the demilitarized zone (54.230.203.0/24).

The firewall has three interfaces: `eth0` connected to the LAN with IP 10.0.0.1, `eth1` connected to the DMZ with IP 54.230.203.1 and `ext` connected to the Internet with public IP 23.1.8.15.

We want to enforce the following policy on the traffic passing through the firewall:

```
1  *nat
2  # ACCEPT policy in nat chains
3  :PREROUTING ACCEPT [0:0]
4  :INPUT ACCEPT [0:0]
5  :OUTPUT ACCEPT [0:0]
6  :POSTROUTING ACCEPT [0:0]
7
8  # R3 - Apply SNAT on connections from the LAN towards the Internet
9  -A POSTROUTING -s 10.0.0.0/8 -o ext -j MASQUERADE
10
11 COMMIT
12
13 *filter
14 # DROP policy in filtering chains
15 :INPUT DROP [0:0]
16 :FORWARD DROP [0:0]
17 :OUTPUT DROP [0:0]
18
19 # Allow established packets
20 -A FORWARD -m state --state ESTABLISHED -j ACCEPT
21 # R2 - LAN hosts can connect to DMZ
22 -A FORWARD -s 10.0.0.0/8 -d 54.230.203.0/24 -j ACCEPT
23 # R3 - LAN hosts can connect to the Internet over HTTP/HTTPS
24 -A FORWARD -s 10.0.0.0/8 -o ext -p tcp --dport 80 -j ACCEPT
25 -A FORWARD -s 10.0.0.0/8 -o ext -p tcp --dport 443 -j ACCEPT
26 # R1 - Any host can connect to the HTTPS server in the DMZ
27 -A FORWARD -d 54.230.203.47 -p tcp --dport 443 -j ACCEPT
28
29 COMMIT
```

Figure 2: Firewall configuration in `iptables`.

R1 Hosts from the Internet can connect to the HTTPS server in the DMZ;
R2 LAN hosts can freely connect to any host in the DMZ;
R3 LAN hosts can connect to the Internet over HTTP and HTTPS (with source NAT).

Analysis of an `iptables` configuration

Figure 2 shows a commented `iptables` configuration (in the `iptables-save` format) that implements the above security policy. In the rules we use both IP addresses and network interfaces: the mapping between interfaces and addresses is provided to FWS with a separate file. Now we use FWS for verifying that the configuration meets the requirements above. For instance, we can check R1 and R2 simultaneously by asking FWS the query

```
dstIp == 54.230.203.0/24 && state == NEW
```

where `dstIp` is the destination address of the packet entering the firewall and `state` is used to distinguish between new and established connections. The operator `==` constrains a variable to be equal to a value or inside a certain interval, while `&&` is the logical and. In particular, the query checks whether new connections are allowed towards hosts in the subnet 54.230.203.0/24.

The output of the tool is in Table 1a, where `*` denotes any value. The first line of the table states that any host can communicate over HTTPS (port 443) with the server 54.230.203.47 (R1); whereas the second line says that any machine in the LAN can freely communicate with the hosts in the DMZ (R2).

Src IP	Src Port	Dst IP	Dst Port	Protocol	State
*	*	54.230.203.47	443	tcp	NEW
10.0.0.0/8	*	54.230.203.0/24	*	*	NEW

(a) Requirements R1 and R2

Src IP	Src Port	SNAT IP	Dst IP	Dst Port	Protocol	State
10.0.0.0/8	*	23.1.8.15	* \ { 10.0.0.0/8 54.230.203.0/24 127.0.0.0/8 }	80 443	tcp	NEW

(b) Requirement R3

Table 1: Analysis of the `iptables` policy.

We now check requirement R3. To do that, we ask which packets from the LAN can reach hosts that are not in the DMZ:

```
srcIp == 10.0.0.0/8 && not(dstIp == 54.230.203.0/24) && state == NEW
```

The answer to the query is in Table 1b: Internet hosts can be reached over HTTP and HTTPS (ports 80 and 443) and the source IP address of outgoing packets is rewritten to 23.1.8.15, the external address of the firewall. In the fourth column we use \ (set difference) on addresses to denote all addresses except those of the LAN, DMZ and loopback interface (the Internet).

Maintaining existing configurations

Suppose to add a new computer with IP address 10.13.3.7 to the LAN. Differently from other machines, we allow access to the hosts on the Internet only over HTTPS. The other requirements on traffic should be preserved. To this purpose, we can add the following rule to the `FORWARD` chain, which drops connections to port 80 from host 10.13.3.7:

```
-A FORWARD -s 10.13.3.7 -p tcp --dport 80 -j DROP
```

However, we must be careful about where to place this rule in order to fulfill the desired requirement and avoid to unintentionally block legal traffic.

If we place the new rule at the end of the `FORWARD` chain, the *policy equivalence* analysis implemented in FWS reports that the new policy is equivalent to the previous version. Then, we use the *related rules* analysis to understand which rules are relevant for processing new HTTP packets. The output of this analysis includes the filtering rules at lines 22 and 24 of the `FORWARD` chain (see Figure 2):

```
-A FORWARD -s 10.0.0.0/8 -d 54.230.203.0/24 -j ACCEPT
-A FORWARD -s 10.0.0.0/8 -o ext -p tcp --dport 80 -j ACCEPT
```

If we add the new rule before the line 21, FWS reports that the policy is not equivalent to the previous one. In order to check the impact of our changes we run the *policy difference* analysis projected over the HTTP traffic using the query `protocol == tcp && dstPort == 80`. The output is in Table 2. In the first column, `+` (resp. `-`) stands for lines that appear (resp. disappear) in the synthesis after the updates. Now the host 10.13.3.7 cannot connect to the Internet over HTTP, as desired (lower part of Table 2). However, our update also prevents

+/-	Src IP	Src Port	Dst IP	Dst Port	Protocol	State
+	10.0.0.0/8 \ {10.31.3.37}	*	54.230.203.0/24	80	tcp	NEW
-	10.0.0.0/8	*	54.230.203.0/24	80	tcp	NEW

<table>
<tr><th>+/-</th><th>Src IP</th><th>Src Port</th><th>SNAT IP</th><th>Dst IP</th><th>Dst Port</th><th>Protocol</th><th>State</th></tr>
<tr><td>+</td><td>10.0.0.0/8 \ {
10.31.3.37
}</td><td>*</td><td>23.1.8.15</td><td>* \ {
10.0.0.0/8
54.230.203.0/24
127.0.0.0/8
}</td><td>80</td><td>tcp</td><td>NEW</td></tr>
<tr><td>-</td><td>10.0.0.0/8</td><td>*</td><td>23.1.8.15</td><td>* \ {
10.0.0.0/8
54.230.203.0/24
127.0.0.0/8
}</td><td>80</td><td>tcp</td><td>NEW</td></tr>
</table>

Table 2: Policy differences after the wrong update.

communications over HTTP with machines on the DMZ, thus violating the requirement R2 (upper part of Table 2).

The correct placement of the new rule is between the lines 22 and 23, i.e. the rule for the requirement R2 and those for the requirement R3. In this way we allow HTTP traffic from 10.13.3.7 only to the DMZ. If we repeat the analysis, we see that now the only difference is just in the HTTP traffic towards the Internet, as desired.

3 Conclusions

We have shown how our tool FWS can be used to verify the compliance of a firewall policy with the desired functionality and security requirements of a network in a simple yet realistic case study. Moreover, we have discussed how the various analyses implemented by the tool can to simplify maintenance operations.

In order to evaluate the effectiveness of FWS on real-world scenarios we have tested on several `iptables` policies collected by the authors of [1]. We have measured the time needed by FWS to synthesize policies of size ranging from 15 to 263 rules and most of the analyses terminate in less than 3 seconds. The 3 most complex cases of size 77, 90 and 263 rules terminate in 28, 25 and 355 seconds, respectively. We stress that the reported times are those required to perform a *complete* synthesis of a policy, whereas in a typical usage the user will run the analyses on a specific set of hosts or services using the query language of FWS.

References

[1] Cornelius Diekmann, Julius Michaelis, Maximilian P. L. Haslbeck, and Georg Carle. Verified iptables Firewall Analysis. In *Proceedings of the 15th IFIP Networking Conference, Vienna, Austria, May 17-19, 2016*, pages 252–260, 2016.

[2] The IPFW Firewall. https://www.freebsd.org/doc/handbook/firewalls-ipfw.html.

[3] Robert M. Marmorstein. *Formal Analysis of Firewall Policies.* PhD thesis, College of William and Mary, Williamsburg, VA, May 2008.

[4] Packet Filter (PF). https://www.openbsd.org/faq/pf/.

[5] The Netfilter Project. https://www.netfilter.org/.

[6] FireWall Synthesizer (FWS): Tool and Examples. https://github.com/secgroup/fws.

Appendix: Using FWS

In this section we describe how using our tool to replicate the results of Section 2. FWS can be downloaded from [6], and installed following the instruction inside `README.MD` file.

Once installed, the command `fws` is available on the command line. The syntax is

```
fws FRONTEND ACTIONS OPTIONS
```

where `FRONTEND` specifies which the firewall system we want to decompile: the supported firewalls are `cisco`, `ipfw`, `iptables`, `pf`. The second argument `ACTION` defines the operation to perform on the configuration. Currently, `fws` implements the following:

`synthesis`	Synthesize a specification
`implication`	Check for policy implication
`equivalence`	Check for policy equivalence
`diff`	Synthesize differences between two firewalls
`convert`	Convert a configuration to the generic language
`query`	Display the rules that affect the selected packets

The thirds argument allows the user to provide `fws` with input file, the query to execute, the format of the output (*e.g.*, ASCII, LaTeX), etc. The options we are using in the following are:

`-i`	Interfaces file
`-f`	Main configuration file
`-s`	Configuration to compare with the main one
`-q`	Query to check on the main configuration

Additionally, some useful shorthands are available to project the analysis on specific subsets of flows that packets can traverse:

`--loopback`	Flows with local source and local destination
`--input`	Flows with remote source and local destination
`--output`	Flows with local source and remote destination
`--forward`	Flows with remote source and remote destination
`--nat`	Flows with NATs
`--filter`	Flows without NATs

In the rest of this section we assume that the current directory contains the files of the directory `examples/itasec18` of the archive at [6]. To decompile the `iptables` configuration of Figure 1 we invoke `fws` as follows

```
$ fws iptables synthesis -i interfaces -f iptables.txt
```

where the file `iptables.txt` contains the configuration and the file `interface` the information about each network interface. More precisely, it includes a line for each network interface; each line contains the name of the interface, the address range of the sub-network it communicates with and its IP address. For example, the interface file for our case study contains the following

`lo`	127.0.0.0/8	127.0.0.1
`eth0`	10.0.0.0/8	10.0.0.1
`eth1`	54.230.203.0/24	54.230.20
`ext`	0.0.0.0/0	23.1.8.15

For example, the interface `eth0` communicates with hosts in the sub-network 10.0.0.0/8 and its IP address is 10.0.0.1.

The content of Tables 1a and 1b are obtained by the following invocations of `fws`:

+/-	Src IP	Src Port	SNAT IP	Dst IP	Dst Port	Protocol	State

+	10.0.0.0/8 \ {
	10.31.3.37
	}
		*	23.1.8.15	* \ {
				10.0.0.0/8
				54.230.203.0/24
				127.0.0.0/8
				}
					80	tcp	NEW

-	10.0.0.0/8	*	23.1.8.15	* \ {
				10.0.0.0/8
				54.230.203.0/24
				127.0.0.0/8
				}
					80	tcp	NEW

Table 3: Policy differences after the correct update.

```
$ fws  iptables synthesis -f iptables.txt -i interfaces --forward \
     -q dstIp == 54.230.203.0/24 && state == NEW

$ fws  iptables synthesis -f iptables.txt -i interfaces --forward \
     -q srcIp == 10.0.0.0/8 && not(dstIp == 54.230.203.0/24) && state == NEW
```

To check that the policy obtained by adding the rule to drop connections to port 80 from host 10.13.3.7 at the end of the `FORWARD` chain is equivalent to the original one, we can run the command

```
$ fws iptables equivalence -i interfaces -f iptables.txt \
    -s iptables_wrong_update_1.txt
```

To get the rules that are relevant in processing HTTP packets of new connections we launch `fws` as follows:

```
$ fws iptables query -i interfaces -f iptables.txt \
   -q "protocol == tcp && dstPort == 80 && state == NEW"
```

The output of the tool is the following:

```
-t nat -A POSTROUTING -s 10.0.0.0/8 -o ext -j MASQUERADE
-t filter -A FORWARD -s 10.0.0.0/8 -d 54.230.203.0/24 -j ACCEPT
-t filter -A FORWARD -s 10.0.0.0/8 -o ext -p tcp --dport 80 -j ACCEPT
```

Finally, we can compute the difference between the original configuration in Figure 1 and the new version concerning the connections to port 80 running:

```
$ fws iptables diff -i interfaces -f iptables.txt \
    -s iptables_correct_update.txt \
    -q "protocol == tcp && dstPort == 80" --forward
```

The output of the tool, reported in Table 3, shows that our modification affects only connections to the Internet, as expected.