Architecting the Lumeta Firewall Analyzer

Avishai Wool
Lumeta Corporation,
220 Davidson Ave, Somerset NJ 08873
E-mail: `yash@acm.org`.

Abstract

Practically every corporation that is connected to the Internet has at least one firewall, and often many more. However, the protection that these firewalls provide is only as good as the policy they are configured to implement. Therefore, testing, auditing, or reverse-engineering existing firewall configurations should be important components of every corporation's network security practice. Unfortunately, this is easier said than done. Firewall configuration files are written in notoriously hard to read languages, using vendor-specific GUIs. A tool that is sorely missing in the arsenal of firewall administrators and auditors is one that will allow them to analyze the policy on a firewall.

The first passive, analytical, firewall analysis system was the Fang prototype system [MWZ00]. This was the starting point for the new Lumeta Firewall Analyzer (LFA) system. LFA improves upon Fang in many ways. The most significant improvements are that human interaction is limited to providing the firewall configuration, and that LFA automatically issues the "interesting" queries and displays the outputs of all of them, in a way that highlights the risks without cluttering the high-level view. This solves a major usability problem we found with Fang, namely, that users do not know which queries to issue.

The input to the LFA consists of the firewall's routing table, and the firewall's configuration files. The LFA parses these various low-level, vendor-specific, files, and simulates the firewall's behavior against all the packets it could possibly receive. The simulation is done completely offline, without sending any packets. The administrator gets a comprehensive report showing which types of traffic the firewall allows to enter from the Internet into the customer's intranet and which types of traffic are allowed out of the intranet. The LFA's report is presented as a set of explicit web pages, which are rich with links and cross references to further detail (allowing for easy drill-down). This paper describes the design and architecture of the LFA.

1 Introduction

1.1 Background

Firewalls are the cornerstones of corporate intranet security. Once a firewall is acquired, a security/systems administrator has to configure and manage it to realize an appropriate security policy for the particular needs of the company. This is a crucial task; quoting [RGR97]: "The single most important factor of your firewall's security is how you configure it".

Even understanding the deployed firewall policy can be a daunting task. Administrators today have no easy way of answering questions such as "can I telnet from here to there?", or "from which machines can our DMZ be reached, and with which services?", or "what will be the effect of adding this rule to the firewall?". These are basic questions that administrators need to answer regularly in order to perform their jobs, and sometimes more importantly, in order to explain the policy and its consequences to their management. There are several reasons why this task is difficult, for instance:

(i) Firewall configuration languages tend to be arcane, very low level, and highly vendor specific.

(ii) Vendor-supplied GUIs require their users to click through several windows in order to fully understand even a single rule: at a minimum, the user needs to check the IP addresses of the source and destination fields, and the protocols and ports underlying the service field.

(iii) Firewall rule-bases are sensitive to rule order. Several rules may match a particular packet, and usually the first matching rule is applied – so changing the rule order, or inserting a correct rule in the wrong place, may lead to unexpected behavior and possible security breaches.

(iv) Alternating PASS and DROP rules create rule-bases that have complex interactions between different rules. What policy such a rule-base is enforcing is hard for humans to comprehend when there are more than a handful of rules.

A tool that is sorely missing in the arsenal of firewall administrators and auditors is one that will allow them to analyze, test, debug, or reverse-engineer the policy on a firewall. Such a tool needs to be exhaustive in its coverage, be high level, and be convenient to use. This paper describes the evolution and architecture of the Lumeta Firewall Analyzer (LFA), a second generation system that addresses the analysis needs of firewall administrators, security consultants, and auditors.

1.2 The Fang System

The first passive, analytical, firewall analysis system was the Fang prototype system [MWZ00]. Fang read all the vendor-specific configuration files, and built an internal representation of the implied policy. It provided a graphical user interface (GUI) for posing queries of the form "does the policy allow service S from A to B?". Fang would then simulate the firewall's policy against the query, and display the results back onto the user's screen.

Before Fang could be used, it needed to have an instantiated model of the firewall connectivity, which contained details like how many interfaces the firewall has, which subnets are connected to each interface, and where the Internet is situated with respect to the firewall. Therefore, before querying the firewall policy, a Fang user needed to write a firewall connectivity description file. The language used to describe the firewall connectivity was derived from the Firmato MDL language [BMNW99].

The core of Fang's query engine was a combination of a graph algorithm and a rule-base simulator. It took as input a user query consisting of source and destination host-groups (arbitrary sets of IP addresses, up to a wildcard "all possible IP addresses"), and a service group (up to a wildcard "all possible services"). It would then simulate the behavior of the firewall's rule-base on all the packets described by the query, and compute which portions of the original query would manage to reach from source to destination: Perhaps only a subset of the queried services are allowed, and only between subsets of the specified source and destination host-groups.

1.3 Contributions

To test Fang's usability and the value it provided, we collected feedback from beta testers. This feedback raised issues we needed to address. The new LFA architecture introduces several new features that address these issues:

- The user does not need to write the firewall connectivity file any more. LFA has a new front-end module that takes a formatted routing table and automatically creates the firewall connectivity file.
- Using a GUI as an input mechanism turned out to be difficult for users. Instead, LFA is now a batch process, that simulates the firewall policy against practically every possible packet.
- A crucial part of the batch processing is the automatic selection of queries. Our choice of queries needs to ensure comprehensive coverage, to highlight any risks, and to make sense to users without overwhelming them with minutiae.
- The LFA output is now formatted as a collection of web pages (HTML). This format gives us the ability to present the output at many levels of abstraction and from multiple viewpoints, allowing easy drill-down to details without cluttering the high level view.
- We needed to support more firewall vendors. For this purpose, LFA now uses an intermediate firewall configuration language, to which we convert the various vendors' configurations.

Organization: In Section 2 we describe the components of the LFA architecture and the design decisions that led us to this architecture. In Section 3 we discuss some related work. In Section 4 we provide an annotated example of how the LFA works. We conclude in Section 5.

2 The LFA Architecture

The main contribution of the Fang prototype was its core query engine. The combination of its internal firewall connectivity model, data structures, and efficient algorithms, demonstrated that it is feasible to analytically simulate a firewall's policy offline. However, from the beta-testers' feedback we got, it became apparent that the software architecture needed to be revisited in order to take the core technology from a prototype into a product. In the next sections we describe the problems that we identified in the Fang prototype, and their solutions within the LFA.

2.1 Describing the Firewall Connectivity

As we mentioned above, before using Fang the user needed to write a firewall connectivity description file, using the Firmato MDL language [BMNW99]. For every network interface card (NIC) on the firewall, the firewall connectivity description file contains a list of IP address ranges that are located behind that NIC. These lists are required to be disjoint: each IP address is allowed to appear only once. This requirement is fundamental to the simulation process: For every possible packet, Fang needs to know which firewall interfaces the packet would cross on its path from source to destination—and thereby, which firewall rule-bases would be applied to it.

The need to write a firewall connectivity file caused two problems. First, the user had to learn the syntax and semantics of the MDL language, which takes time and effort. Second, and more important, the information that is needed to describe the firewall connectivity is not readily available to firewall administrators in a suitable format. This information is typically only encoded in the firewall's routing table. However, routing table entries are usually not disjoint: It is common to have many overlapping routing table entries that cover the same IP address. The semantics of a routing table determine which route entry is used for a given IP address: it is the most specific one, i.e., the entry for the smallest subnet that contains the given IP address is the one that determines the route to that IP address. The task of accessing the routing table, and manually converting it into lists of disjoint IP address ranges, turned out to be difficult and error prone.

To solve both problems, the Lumeta Firewall Analyzer introduced a a new front-end module, called `route2hos`, that mechanically converts a routing table into a Firmato MDL firewall connectivity file. All that is required of the user is to provide the firewall's routing table (in the form of the output of the `netstat` command on Unix systems).

The `route2hos` module uses an engine that implements the routing table semantics. In other words, for a given IP address, it is able to determine over which NIC a packet with this address as its destination would be routed. By judiciously using this engine against the subnets listed in the routing table, `route2hos` is able to create the disjoint lists of IP address ranges that the Fang query engine requires. The output of `route2hos` is the firewall connectivity description file, in the MDL language.

As part of the processing done by `route2hos`, it produces definitions for two special host groups, called Inside and Outside. The Outside host group consists of all the IP addresses that get routed via the default interface, according to the firewall's routing table. This host group typically includes the Internet, and any of the corporation's subnets that are external to the firewall. The Inside host group is everything else. These two host groups are later used in the query processing (see below).

2.2 What to Query?

The Fang prototype had a graphical user interface which allowed the user to enter queries of their choice. However, during beta testing we discovered that users do not know which queries they need to try. They were not sure which services are risky, nor which host groups needed to be checked. Furthermore, on a reasonably configured firewall, most queries return uninteresting results, e.g.: "is telnet allowed into my network?"; "No"; etc. This causes users to lose interest and leads to a partial simulation of the policy. Most importantly, the queries that are likely to find the problems in the rule-base are often precisely those queries that the user does not know to try.

To solve these problems, the Lumeta Firewall Analyzer takes the burden of choosing the queries off the user's shoulders. It does this by querying *everything*. In fact, we completely eliminated the GUI as an input mechanism in the LFA, and replaced it by a batch process, which repeatedly calls Fang's query engine.

Clearly, it is impossible to simulate all the packet combinations one by one. Enumerating all the possible combinations of source and destination IP addresses (32 bits each), protocol (8 bits), and source and destination port numbers (16 bits each), gives rise to an enumeration space of 2^{104}.

There are two facts that allow LFA to circumvent this combinatorial explosion: (i) the Fang query engine processes aggregated queries very efficiently, and (ii) after the `route2hos` processing the LFA knows which IP addresses are external to the firewall (this is the Outside host group). Combining these two facts, LFA can issue the query "list the types of traffic that can enter from the Outside to the Inside using any service". We denote such a query by

$$\text{Outside} \rightarrow \text{Inside} : *.$$

The result is a list of `(src, dest, srv)` tuples describing the allowed incoming traffic, in which the IP addresses of `src` are contained in the Outside host group, the IP addresses of `dest` are contained in Inside, and the service is `srv`. Similarly, LFA can make the outgoing query "$\text{Inside} \rightarrow \text{Outside} : *$", switching the roles of Inside and Outside.

After experimenting with the approach we just outlined, we discovered that users had difficulty in interpreting its results. For instance, suppose the firewall has a rather typical rule of the form "from anywhere, to my-server, allow any service". The query "$\text{Outside} \rightarrow \text{Inside} : *$" would produce the response "$\text{Outside} \rightarrow$ my-server : $*$". This response does not convey to the user that "$*$" (any service) includes quite a few high-risk services that should probably not be allowed—if this fact was obvious to the user, he would not have written such a rule in the first place! Users found the results much easier to interpret if instead of presenting a blanket response saying "any service" is allowed, we presented them with a long list of individual services that are allowed.

Therefore, the LFA in fact does not make the query "$\text{Outside} \rightarrow \text{Inside} : *$". Instead it issues a set of focused queries: "$\text{Outside} \rightarrow \text{Inside}$: `dns`"; "$\text{Outside} \rightarrow \text{Inside}$: `netbios`"; etc., and similarly for outgoing traffic. The list of services that are queried in this way is made of two parts: a list of well known services, plus a list containing every specific service that appears in some rule on the firewall. We have found that querying individual services this way makes the query results, and the risks they entail, much more explicit. The user has two possible cues indicating risk: (1) If a rule is wide open, there will be a very long list of individual services appearing in the query results (more services == more risk); (2) The user will see services he may either recognize as dangerous, or not recognize at all (making them worrisome).

Note, however, that by querying individual services this way, LFA may miss some services. A service that is not on the LFA's list of "known services", and does not appear explicitly on any rule, will not be queried.

To ensure this does not happen, LFA performs two additional sets of queries. In these queries, the queried service is the "all service" wildcard "*". However, following the same philosophy from before, we attempt to make the queries specific, in a different way. For incoming traffic, LFA makes queries of the form "$\text{Outside} \rightarrow$ internal-host-group : $*$", where "internal-host-group" goes over every internal host group.[1] LFA then goes over the internal host groups again, making outbound queries of the form "internal-host-group $\rightarrow \text{Outside} : *$".

The results of all these queries are organized into four reports, called "Analysis by service: Incoming", "Analysis by service: Outgoing", "Analysis by host group: Incoming", and "Analysis by host group: Outgoing". This organization offers the user the opportunity to look at the firewall configuration from different viewpoints, while providing a comprehensive coverage of the traffic the firewall may encounter.

2.3 Supporting Multiple Vendors

The core query engine uses a model of the firewall rulebase, which is generic and vendor-independent. However, in order to instantiate this model, the Lumeta Firewall Analyzer (LFA) needs to be able to parse the vendor-specific configuration files, and if necessary, to convert the vendor's firewall semantics into their equivalent in the LFA model. The Fang prototype provided native support (within the C code implementing the core query engine) only for the Lucent Managed Firewall [LMF99] configuration file syntax.

When we started adding support for other vendors (notably Check Point's and Cisco's products), we decided not to include additional parsers for these vendors' languages within the core. Instead, we opted for an architecture centered around an intermediate language. We chose to write a separate front-end conversion utility for each supported vendor. We chose to write these utilities using the Perl programming language. The front-ends would take the vendor's files and translate them into the LFA's intermediate language. We had three options for an intermediate language. We could base it on an access-control-list language, or on one of Check Point's languages, or on the Lucent Managed Firewall (LMF) language.

Access-control-list languages such as Cisco's IOS [IOS00] and PIX [PIX97] configuration languages, or the Linux `ipchains` (cf. [Rus00]) script language, do not support named host groups, and a rule's source and destination are restricted to be CIDR-block subnets. Therefore, an access-control-list language was deemed too low-level for our purposes; converting other firewall configuration languages to it would lose information and

---

[1] A host group is considered to be internal if it has a non-empty intersection with in the Inside host group.

greatly increase the configuration size.[2]

Check Point [Che97] uses two separate languages in the configuration of their FW-1 product: the INSPECT language, and the language within the `*.W/*.C` policy files. The INSPECT language does support IP ranges but does not support naming, so it was deemed too low level. The `.W` language does support naming, groups, and ranges, however, it has the opposite problem: it is too expressive. It contains many irrelevant details, such as the colors in which to render the icons on screen, and has a syntax that is much harder to parse or to synthesize.

The language we chose to base our intermediate language on was the LMF configuration language. The basic LMF language is relatively easy to parse and to synthesize, yet contains higher-level constructs such as service groups and host groups, named user-defined services, named host groups, and arbitrary ranges of IP addresses.

Since we only use the language internally, within the LFA, there was no reason to maintain strict compatibility with the real LMF language. Therefore we only used some of the LMF language components and ignored others. Furthermore, we did need to extend the LMF language to incorporate features which LMF itself does not support, such as negated host groups.[3]

2.4 Presentation of Results

In addition to letting the user specify her query, the Fang GUI also displayed the query output to the user. The GUI had a basic mode showing the names of the sources, destinations, and services in the resulting `(src, dest, srv)` tuple. The user had the ability to expand each tuple to show the IP addresses and port numbers (all the components expanded simultaneously). However, beta testers felt that these two display modes were too limiting.

When we discarded the GUI, we needed an alternative mechanism to view the query results. Our choice was to use an HTML-based display. We updated the core query engine so it will dump all its findings into several formatted plain-text output files. Then we created a collection of Perl back-end utilities that convert the output files into a set of web pages.

The back-ends create four support web pages:

Original rules. This page shows the rule-base in a format that is as close as possible to the format used by the vendor's management tools.

Expanded rules. This page shows the rule-base after conversion into the LFA intermediate language.

Services. This page shows a table of all the service definitions (protocols and port numbers), with the containment relationships[4] between services. A service has a hyperlink to every service group containing it, and to every service it contains.

Host groups. This page shows a table of the definitions (IP addresses) of all the host groups encountered in the firewall rule-base, with the containment relationships between host groups represented by hyperlinks.

In addition to the support pages, the back-ends create web pages for the four query reports we mentioned in Section 2.2: Analysis by service (Incoming and Outgoing), and Analysis by host group (Incoming and Outgoing). Each query result tuple is linked to the appropriate entries in the Host groups and Services pages, with a direct link to the Expanded rules page pointing to the rule allowing the traffic through. A typical LFA report contains hundreds of such hyperlinks (depending on the complexity of the rule-base).

Besides the extensive navigation capability offered by the various links, we added a JavaScript-based navigation bar, and JavaScript scrolling functions that highlight the table entries in the Rules, Services, and Host Groups tables.

An advantage of such a web-based display is that it does not impose a reading order on the user, and allows easy access to any level of detail the user desires to view. The query result pages just show the names, and the user can choose whether to drill down on each component.

Section 4 contains excerpts from some of the produced web pages.

2.5 Naming Things

An important part of the Lumeta Firewall Analysis involves assigning names to services and host groups.

---

[2] A single IP address range may need multiple CIDR block subnets to cover it, the worst case being the range 0.0.0.1–255.255.255.254, which requires 62 separate CIDR blocks.

[3] A negated host group is shorthand for the IP addresses that are not contained in the host group.

[4] A service group s_1 contains service s_2 if the s_2's protocol is one of s_1's protocols, and s_2's port numbers are contained in the range of s_1's port numbers.

For services and service groups, we use several sources of naming information. First, the LFA has a fairly long list of “well known” service definitions. So if the firewall rule-base contains a rule that refers to `tcp` on port 443, LFA displays it as `https`. Second, most firewalls have built-in named definitions which we use. Finally, for firewalls that support user-defined services, we read those names in.

If the name and definition we get from two sources both match, we only show the service once. However, sometimes there are mismatches: e.g., Check Point has a predefined service called `icmp-proto`, which has the same definition as an LFA-defined service called `ALL_ICMP`. In such cases we incorporate both names into the reports. Another type of mismatch is when the same name is used with different definitions. For instance, there is an LFA-defined service called `traceroute`, which is defined as `udp` with a port range of 32000–53000. Check Point has a predefined service with the same name but defined with a port range of 33001–65535. To avoid ambiguity, we prefix the service name with the source of the definition.

For host groups, we rely on the naming information that the firewall provides, which consists of user-defined names. If the firewall does not support host group names (as is the case, e.g., for Cisco IOS [IOS00] access-control-lists), we use the IP addresses themselves as the name. In addition, in all cases, LFA attempts to supplement the host group names with DNS lookups where possible. A reverse DNS lookup is performed for every individual IP address that appears anywhere in the rule-base. For subnets, LFA uses a heuristic to pick a representative IP address in the subnet, and looks up that IP address’ name.

2.6 Check Point-Specific Features

The Lumeta Firewall Analyzer (LFA) front-end `ckp2lfa`, that converts Check Point FW-1 configurations into the LFA intermediate language, has to deal with several Check Point-specific features.

Global properties These are properties which are accessed through a separate tab in Check Point’s management module, and are not seen in the rules table shown in the Check Point GUI. Some of the properties control remote management access to the firewall itself, `dns` access through the firewall, and `icmp` access. Depending on their setting, these properties in fact create implicit rules that are inserted into the rule-base at certain positions. The `ckp2lfa` front-end converts these FW-1 properties into explicit rules, and places them in their appropriate position in the rule base (First/Before-Last/Last).

Object groups Check Point FW-1 allows network objects (i.e., host groups) to be defined as groups of other objects, which themselves may be groups, thus creating a containment hierarchy of groups. If the hierarchy is complicated enough, FW-1 users sometimes lose track of what IP addresses the group actually consists of, which leads to all kinds of configuration errors. The `ckp2lfa` front-end flattens out the hierarchy, by computing the explicit list of IP addresses that belong to such a group object. This flattening does not lose information: one of the features of the LFA query engine is that it computes the host group containment relationships from the IP addresses, regardless of whether a host group was defined as a group or not.

Negated objects Check Point FW-1 allows the firewall administrator to define rules which refer to IP addresses “not in” a host group, or to services “not in” a service group. The `ckp2lfa` front-end converts the implicit definition into an explicit one, by computing all the IP addresses that do not belong to the negated host group.

3 Related Work

3.1 Active Vulnerability Testing

A number of vulnerability testing tools are available in the market today. Some are commercial, from vendors such as Cisco [CSS00] and ISS [ISS00], others are free such as Fyodor’s `nmap` [Fyo00]. These tools physically connect to the intranet, and probe the network, thereby testing the deployed routing and firewall policies. These tools are *active*: they send packets on the network and diagnose the packets they receive in return. As such, they suffer from several restrictions:

(i) If the intranet is large, with many thousands of machines, testing all of them using an active vulnerability tester is prohibitively slow. Certainly, an active test tool cannot check against every possible combination of source and destination IP address, port numbers and protocols. Hence, users are forced to select which machines should be tested, and hope that the untested machines are secure. Unfortunately, it only takes one vulnerable machine to allow a penetration.

(ii) Vulnerability testing tools can only catch one type of firewall configuration error: allowing unauthorized packets through. They do not catch the other type of error: inadvertently blocking authorized packets. This second type of error is typically detected by a "deploy and wait for complaints" strategy, which is disruptive to the network users and may cut off critical business applications.

(iii) Active testing is always after-the-fact. Detecting a problem after the new policy has been deployed is dangerous (the network is vulnerable until the problem is detected and a safe policy is deployed), costly (deploying policy in a large network is a time consuming and error prone job), and disruptive to users. Having the ability to cold-test the policy before deploying it is a big improvement.

(iv) An active vulnerability tester sends packets, and detects problems by examining the return packets it gets or doesn't get. Therefore, it is inherently unable to test network's vulnerability to spoofing attacks: If the tester would spoof the source IP address on the packets it sends, it would never receive any return packets, and will have no indication whether the spoofed packets reach their destination or not.

(v) An active tester can only test from its physical location in the network topology. A problem that is specific to a path through the network that does not involve the host on which the active tool is running will go undetected.

3.2 Distributed Firewalls

Recently there has been a renewed interest in firewall research, focusing on Bellovin's idea of a distributed firewall [Bel99]. A working prototype has been developed under OpenBSD [IKBS00]. The basic idea is to make every host into a firewall that filters traffic to and from itself. This trend is growing in the commercial world as well: personal firewalls for PCs, such as Zone Labs [Zon00] and BlackICE [Bla00], are becoming more common, as high-bandwidth, always-on, Internet connections like DSL and Cable become more widespread.

The main advantages of a distributed firewall are that (i) since the filtering is at the endpoint, it can be based on more detailed information (such as the binary executable that is sending or receiving the packets); and (ii) there is no bandwidth bottleneck at the perimeter firewall. The main difficulties with a distributed firewall are (i) the need for a central policy to control the filtering, and (ii) the need to ensure that *every* device in the network is protected, including infrastructure devices like routers and printers.

It is this author's opinion that a distributed firewall architecture will augment, rather than replace, the perimeter firewall. The conventional firewall will remain as an enterprise network's first line of defense. The fact that one can put a lock on every office door does not make the guard at the building entrance unnecessary: there is still valuable stuff in the hallways, and not everyone uses the lock properly. When a widely deployed distributed firewall system becomes available, it will most likely be used as a second line of defense, behind the perimeter firewall. The perimeter firewall will continue to protect all the infrastructure that is not controlled by the new architecture, to defend against denial-of-service attacks, and to ensure central control.

4 An Example

In this section we show an annotated example which illustrates the flow of data through the various components of the LFA. This example is based upon a firewall rule-base that was installed on a real firewall protecting a production network of a large enterprise. Using the LFA report, the firewall's administrators were able to correct a major security risk that was present in their firewall configuration. For demonstration purposes, we recreated the key elements of that risky configuration onto a lab machine, and ran the resulting files through the LFA. The report excerpt shown here is from the lab machine. The full web-based sample report is available online from [Lum01].

In Figure 1 we see a web page showing a Check Point FW-1 rule-base. This is the LFA's starting point. The only processing that was done to create this page was to convert Check Point's configuration files into HTML, rendered in a format that is quite close to that of FW-1's management module (down to the level of user-defined colors for various objects). The conversion utility we used is an improved version of the `fwrules50` program [XOS$^+$00].

At a cursory glance, the rule-base looks rather simple, protecting two machines (called `one` and `two`). Machine `one` seems to be a web server, and machine `two` seems to be a Usenet (nntp) news server. The policy is

Figure 1: The original rule-base, rendered in HTML.

quite lax on outbound services (rule 3 allows all types of `tcp` outbound), but seems quite reasonable for inbound connections, allowing only `http`, `https`, `ssh`, and `nntp`.

In Figure 2 we can see an HTML rendering, produced for the same rule-base, of the LFA's intermediate language, as discussed in Section 2.3. The figure shows the results of the Check-Point-to-LFA front-end conversion utility, `ckp2lfa`, post-processed into an HTML-based report (called the Expanded Rules report) by the back-end utilities.

We can see that the rule-base now has several additional rules. These rules are derived from Check Point "properties", which are controlled through a separate tab in Check Point's management module. The properties that are selected by the administrator create implicit rules that are inserted into the rule-base at certain positions. One of the tasks of the `ckp2lfa` front-end is to convert all these implicit rules into their explicit equivalents, and insert them in their correct positions in the rule-base.

Figure 2 shows the effects of properties that govern DNS and ICMP traffic, and of the property that controls remote management access to the firewall itself. After `ckp2lfa` converts the implicit rules into explicit ones, we can see that rules 1, 3, and 10, are wide open (allowing traffic from anywhere to anywhere). Unfortunately, these rules represent the effects of Check Point FW-1's default settings. Based on client configuration files we have seen, leaving these properties at their default setting seems to be a common mistake among FW-1 administrators.

Another piece of information that is clear after the `ckp2lfa` conversion is that the firewall is actually performing Network Address Translation (NAT) on the address on machine `one`: Rules 4–8 show that machine `one` has both a valid (routable) IP address and a private IP address. The firewall translates between the two addresses based on the direction of the packets.

The next step in the processing is the the `route2hos` front-end, which converts the firewall's routing table into a Firmato MDL firewall connectivity file. Instead of showing the firewall connectivity file itself, in Figure 3 we show a graphical representation of the firewall connectivity, which is derived from the MDL firewall connectivity file using the graph visualization tool `dot` [GKNV93] [Dot01]. We emphasize that Figure 3 is completely machine-generated, with no manual tweaking. The figure shows the IP addresses behind each of the firewall's three internal interfaces. We can see that interface `if_2` is connected to an RFC 1918 private IP address subnet, with a single routable IP address added (this is the valid IP address of machine `one`, which is NATed). The rest of the IP address space, including all of the Internet, is behind interface `if_0`.

Once the Check Point configuration files have been converted to the LFA intermediate language, and the routing table has been converted into an MDL network firewall connectivity file, the LFA proceeds to simulate the configured policy. This is done by the core query engine (Section 2.2). The output of the core engine is then rendered in HTML by the back-end utilities, which also create all the cross-links between various components of the report.

In Figure 4 we see a portion of the "Analysis by service: Incoming" HTML-based report, which is one of the four reports that LFA creates. The figure shows the results of the query "$\mathrm{Outside} \rightarrow \mathrm{Inside} : \mathtt{netbios}$", meaning "Can `netbios` traffic cross the firewall from the Outside to the Inside?".

Somewhat surprisingly, the report shows that `netbios` traffic is allowed from anywhere on the Outside, to machine `two`. The figure shows the user-defined name ("`two`") alongside the result of a reverse `dns` lookup on the IP address of that machine (recall Section 2.5). We can see in the figure that the culprit rule which allows `netbios` traffic through is rule number 9. All the underlined values shown in Figure 4 are hyperlinks. Clicking on the "9" link brings the user to the Expanded Rules report (recall Figure 2), with rule 9 highlighted. Looking back at Figure 2, we see that rule 9 indeed refers to machine `two`, however, the service listed is called `nntp_services`, not `netbios`.

Clicking on the `nntp_services` link from the Expanded Rules report (Figure 2) brings the user to the Services report, the relevant portion of which is shown in Figure 5. We can see that the definition of `nntp_services` has two components: one with `tcp` on destination port 119 (this is the correct definition), and one with `tcp` on *source* port 119. The latter definition is very risky and is the cause for `netbios` (and, indeed, any other `tcp` service) being allowed through the firewall. This is since the choice of source port is completely under the control of the sender of the packet. There is nothing to prevent an attacker from setting the source port to 119 and the destination port to 139 (`netbios`): the firewall would let the packet through based on its source port, and allow it to access the netbios port on the target machine. This is actually part of a hacking

Figure 2: The expanded rule-base, after conversion to the LFA's intermediate language.

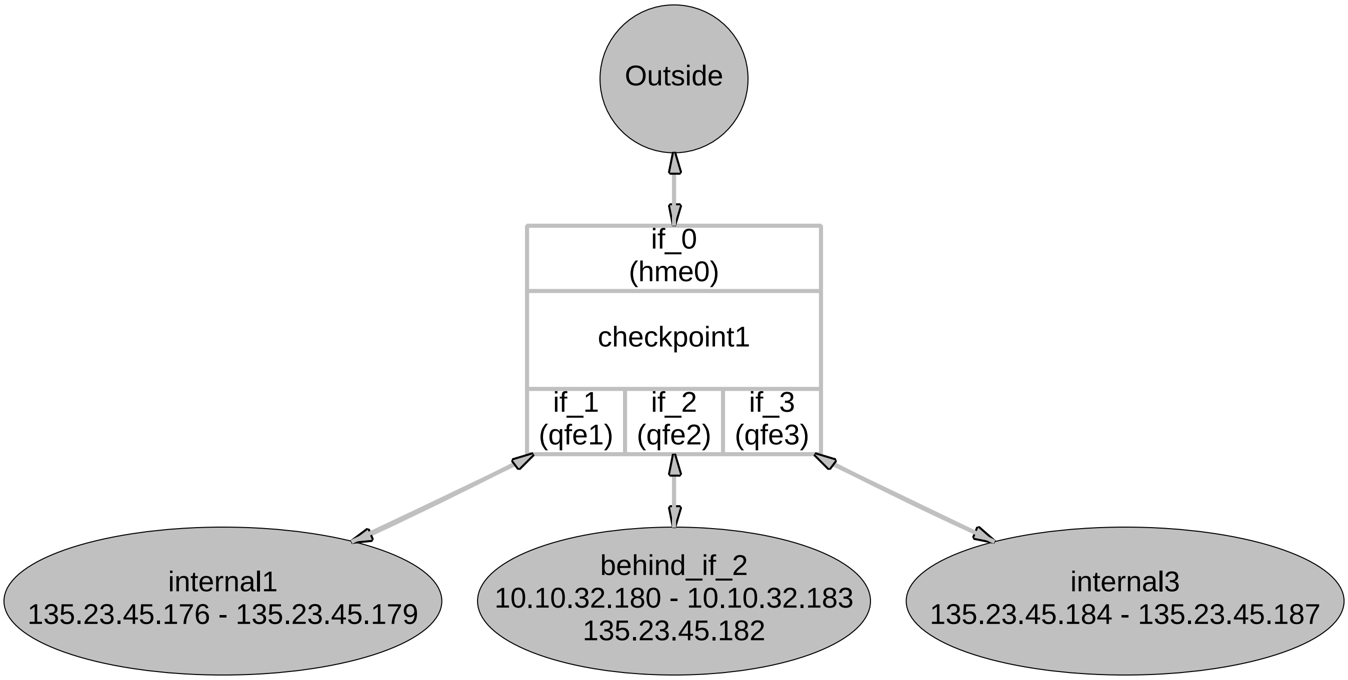

Figure 3: A diagram of the firewall's network connectivity, derived from the firewall connectivity description file.

Figure 4: An excerpt from the “Analysis by service: Incoming” report, showing the results of the `netbios` query.

Figure 5: An excerpt from the Services report, with the `nntp_services` service highlighted.

technique known as “firewalking”, and is usually done using source port 53 (`dns`) which is very often open [GS98].

Remarks:

- A manual inspection of the rule-base shown in Figure 1, even by an expert auditor, is very likely to miss the vulnerability that the LFA demonstrated. The service name listed in the rule (`nntp_services`) makes sense. Even if the auditor is diligent enough to dig deeper and check the definition of the service, she would find that the port number (119) is in fact correct. It is just in the wrong column, half an inch away from being perfect.
- Similarly, a firewall probe by an active vulnerability test tool would probably also miss the vulnerability. Unlike LFA, such a tool inherently cannot test every possible combination of IP addresses and port numbers, and it would have no special reason to test the particular combination of source port 119 and destination port 139.
- We believe that the reason for the mistake in the definition of `nntp_services` is that the firewall administrator who created it was not fully aware of the implications of stateful inspection, and was probably used to configuring stateless packet filters, such as router access-control-lists. A stateful firewall (like Check Point FW-1) will automatically allow the returning packets of an open `tcp` session. A stateless access-control-list requires a separate rule for the returning packets, in which the filtering is done based on the source port (since the destination port is selected dynamically). The erroneous component of the `nntp_services` definition looks precisely like a stateless rule allowing the returning packets through the firewall.

5 Conclusions

The Lumeta Firewall Analyzer (LFA) is a novel, multi-vendor tool that simulates and analyzes the policy enforced by a firewall. The LFA takes the firewall’s configuration files and routing table, parses them, and simulates the firewall’s behavior against all the possible packets it could receive. The result is an explicit, cross-linked, HTML-based report showing all the types of traffic allowed in from the Internet, and all the types of traffic allowed out.

Acknowledgments

A project such as LFA is a team effort. Many people have contributed to the evolution of LFA, whether in ideas, algorithms, or code. I gratefully acknowledge the contributions of Yair Bartal, Sudip Bhattachariya, Steve Branigan, Hal Burch, Diane Burley-McGlue, Bill Cheswick, Terry Lieb, Tom Limoncelli, Ryan Martin, Alain Mayer, Kobi Nissim, Karl Siil, Bruce Wilner, and Elisha Ziskind.

References

[Bel99] S. M. Bellovin. Distributed firewalls. *;login:*, pages 39–47, November 1999.

[Bla00] BlackICE Defender. Network ICE, 2000. `http://www.networkice.com/products/blackice_defender.html/`.

[BMNW99] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. *Firmato*: A novel firewall management toolkit. In *Proc. 20th IEEE Symp. on Security and Privacy*, pages 17–31, Oakland, CA, May 1999.

[Che97] Check Point FireWall-1, version 3.0. White paper, June 1997. `http://www.checkpoint.com/products/whitepapers/wp30.pdf`.

[CSS00] Cisco secure scanner 2.0, 2000. `http://www.cisco.com/warp/public/cc/pd/sqsw/nesn/index.shtml`.

[Dot01] Graphviz - open source graph drawing software. version 1.7, 2001. `http://www.research.att.com/sw/tools/graphviz/`.

[Fyo00] Fyodor. NMAP - the network mapper, 2000. `http://www.insecure.org/nmap/`.

[GKNV93] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

[GS98] D. Goldsmith and M. Schiffman. Firewalking: A traceroute-like analysis of ip packet responses to determine gateway access control lists. White paper, Cambridge Technology Partners, 1998. `http://www.packetfactory.net/firewalk/`.

[IKBS00] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a distributed firewall. In *Proc. 7th ACM Conf. Computer and Communications Security (CCS)*, Athens, Greece, November 2000.

[IOS00] Cisco IOS firewall feature set, 2000. `http://www.cisco.com/univercd/cc/td/doc/pcat/iofwfts1.htm`.

[ISS00] Internet security systems: Internet scanner, 2000. `http://documents.iss.net/literature/InternetScanner/is_ps.pdf`.

[LMF99] Lucent managed firewall, version 3.0, 1999. `http://www.lucent.com/iss/html/technical.html`.

[Lum01] Lumeta firewall analyzer, 2001. `http://www.lumeta.com/solution_firewall.html`.

[MWZ00] A. Mayer, A. Wool, and E. Ziskind. *Fang*: A firewall analysis engine. In *Proc. IEEE Symp. on Security and Privacy*, pages 177–187, Oakland, CA, May 2000.

[PIX97] Cisco's PIX firewall series and stateful firewall security. White paper, 1997. `http://www.cisco.com/warp/public/cc/pd/fw/sqfw500/tech/nat_wp.pdf`.

[RGR97] A. Rubin, D. Geer, and M. Ranum. *Web Security Sourcebook.* Wiley Computer Publishing, 1997.

[Rus00] R. Russell. Linux IPCHAINS-HOWTO, v1.0.8, July 2000. `http://www.linuxdoc.org/HOWTO/IPCHAINS-HOWTO.html`.

[XOS$^+$00] W. Xu, S. O'Neal, J. Schoonover, S. Moser, F. Lamar, and G. Grasboeck. fwrules50, 2000. Available from `http://www.phoneboy.com/fw1/`.

[Zon00] ZoneAlarm 2.1.44. Zone Labs, 2000. `http://www.zonelabs.com/`.