Int. J. Inf. Secur. (2005)
DOI 10.1007/s10207-005-0074-z

REGULAR CONTRIBUTION

Alain Mayer · Avishai Wool · Elisha Ziskind

Offline firewall analysis

Published online: 16 June 2005

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 are 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 allows them to analyze the policy on a firewall.

To alleviate some of these difficulties, we designed and implemented two generations of novel firewall analysis tools, which allow the administrator to easily discover and test the global firewall policy. Our tools use a minimal description of the network topology, and directly parse the various vendor-specific low-level configuration files. A key feature of our tools is that they are passive: no packets are sent, and the analysis is performed offline, on a machine that is separate from the firewall itself. A typical question our tools can answer is "from which machines can our DMZ be reached, and with which services?." Thus, our tools complement existing vulnerability analyzers and port scanners, as they can be used before a policy is actually deployed, and they operate on a more understandable level of abstraction. This paper describes the design and architecture of these tools, their evolution from a research prototype to a commercial product, and the lessons we have learned along the way.

Keywords Network Security · Internet Protocol (IP) · Packet Filtering · Security anagment

Parts of this paper appeared, in preliminary form, in the *21st IEEE Symposium on Security & Privacy, Oakland, CA, May 2000* and in the *10th USENIX Security Symposium, Washington, DC, 2001.*

A. Mayer
CenterRun Inc., 900 Island Drive, Redwood City, CA 94065, USA
E-mail: alain_mayer@hotmail.com

A. Wool (✉)
School of Electrical Engineering, Tel Aviv University,
Ramat Aviv 69978, Israel
E-mail: yash@acm.org

E. Ziskind
Department of Computer Science, Princeton University,
Princeton, NJ, USA
E-mail: eziskind@cs.princeton.edu

1 Introduction

1.1 Motivation

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 [29]: "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, including the following:

(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.

(v) Packets may have multiple paths from source to destination, each path crossing several filtering devices. To answer a query the administrator would need to check the rules on all of these.

A tool that is sorely missing in the arsenal of firewall administrators and auditors is one that allows 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 of such a system through two generations, the Fang research prototype, and its successor, the Firewell Analyzer (FA).[1]

1.2 Contributions

The design goals we had in mind when we built our firewall analysis tools are as follows:

- *Use an adequate level of abstraction*: The administrator should be able to interact with the tool on an adequate level of abstraction, i.e., on the same level at which the corporate security policy is defined or expressed. In a large network, the tool should allow to quickly focus on the important security aspects of testing.
- *Be comprehensive*: A partial or statistical analysis is not good enough. A firewall with even a single badly written rule is useless if an attacker discovers the combination of IP addresses and port numbers that can get through.
- *Do no harm*: Policy analysis should be possible without having to change or tinker with actual network configurations, which in turn might make the network vulnerable to attacks.
- *Be passive*: Policy analysis should not involve sending packets, and should complement the capabilities of existing active test tools.

Our first generation offline firewall analysis tool was the Bell Labs Fang research prototype (for Firewall ANalysis enGine). Early users of Fang provided us with significant feedback that led to the development of the second generation tool, called the Firewall Analyzer (FA).

The heart of our tools is a query engine, which handles the main computational parts of the analysis. The query engine parses the relevant vendor-specific configuration files, and builds an internal representation of the implied policy and network topology. It provides an application programming interface (API) which exports functions to compute the answers to firewall analysis queries. The simplest queries have the form "does the policy allow service s from a to b?." However, the query engine also accepts aggregate queries, where s may be a set of services (up to a wildcard "all possible services"), and a and b may be arbitrary sets of IP addresses (up to a wildcard "all possible addresses").

Given a query, the query engine simulates the behavior of the various firewalls, taking into account the network topology, and computes which portions of the original query would manage to reach from source to destination; perhaps only a subset of the services are allowed, and only between subsets of the specified source and destination host groups. The query engine is able to simulate spoofing attacks; the API allows the user to specify where the packets are to be injected into the network—which may be *different* from the real location of the source host group. The query engine can also take into account firewall rules that perform network address translation (NAT).

The Fang research prototype interacted with the administrator through a query-and-answer session, using a simple GUI we developed. This GUI allowed the administrator compose a query through a collection of menus, activated the query engine API, and finally displayed the results of the query. Fang's GUI allowed us experiment with the query engine, and, more importantly, allowed us collect feedback from users. This feedback was the basis for the development of FA.

The most important lesson we learned from Fang is that users often *do not know what to query*. Therefore, the biggest improvement in FA is that human interaction is limited to providing the firewall configuration, and the analysis is fully automated from that point on. Instead of a manually written network topology file, FA accepts the firewall's routing table. Instead of the point-and-click interface, FA automatically issues *all* the "interesting" queries. Instead of the naive query output display we had in Fang, the FA's report is presented as a set of explicit web pages, which are rich with links and cross references to further detail. Finally, FA supports several major firewall vendors' products. Beyond the fact that we needed several front-end parsers, we had to bridge various semantic discrepancies that exist between the products and FA's model of a generic firewall.

Parts of this paper appeared, in preliminary form, in [26] and [33].

Organization: In Sect. 2 we discuss the components of firewall policies, and introduce some terminology. In Sect. 3 we give some details of the inner workings of the query engine. Section 4 shows Fang's GUI. In Sect. 5 we describe the components of the FA architecture and the design decisions that led us to this architecture, and in Sect. 6 we describe the algorithm for converting a routing table into a topology file. In Sect. 7 we provide an annotated example of how the FA works. In Sect. 8 we discuss some related work, and we conclude in Sect. 9.

[1] Formerly called the "Lumeta Firewall Analyzer," now developed and sold by Algorithmic Security Inc.

2 Background and terminology

2.1 The components of a firewall policy

A firewall is typically placed on a gateway, separating the corporate intranet from the public Internet. This gateway may be either a dedicated machine or a router. Firewalls are configured via a *rule-base*. In the case of a firewall guarding a single homogeneous intranet (e.g., a small company LAN), a single rule-base instructs the firewall which inbound sessions (packets) to let pass and which to block. Similarly, the rule-base specifies which outbound sessions are allowed. The administrator needs to implement the high-level corporate security policy using this low-level rule-base.

A medium- or large-sized company, and any company which has an e-commerce web presence, usually has more than a single firewall; its firewalls divide the company's intranets into multiple *zones*, such as the demilitarized-zone (DMZ), corporate net, human resources, etc. In this case, the security policy is typically realized by multiple rule-bases, located on the various gateways that connect the different zones to each other. Thus, the interplay between these rule-bases determines which sessions will be allowed through.

A typical firewall's configuration tool allows the security administrator to define various host groups (collections of IP addresses) and service groups (groups of protocols and corresponding port-numbers at the hosts which form the endpoints). A single rule typically includes a *source*, a *destination*, a *service-group*, and an appropriate *action*. The source and destination are host groups, and the action is either "pass" or "drop" (the packets of the corresponding session).[2] In most firewalls, the rule-base is order-sensitive: The firewall checks if the first rule in the rule-base applies to a new session. If so, the packets are either dropped or let through according to the action of the first rule. Otherwise, the firewall checks if the second rule applies, and so forth.

2.2 Terminology

Firewall terminology varies slightly from vendor to vendor, so we need to precisely define the terms we use as follows:

Gateways: These are the IP packet filtering devices. Typically, IP packet filtering occurs on a dedicated firewall machine. However, most routers have packet filtering capabilities as well. For our purposes in this paper, any packet filtering device that works at layers 3 (IP) and layer 4 (`tcp`/`udp`/`icmp` etc.) is considered to be a Gateway. We sometimes use the terms "firewall" and "gateway" interchangeably.

Interfaces: Typically, a gateway has multiple network connections. Each connection goes through an *interface*. We assume that each interface has a packet filtering rule-base associated with it (this is more general than assuming only a single rule-base per gateway). Normally each interface has its own unique IP address.

Zones: The gateways partition the IP address space into disjoint *zones*. Precisely, a zone z is a maximal set of IP addresses such that packets sent between any two addresses in z do not pass through any filtering gateway. Most zones correspond to a corporation's subnet(s), usually with one big "Internet" zone corresponding to the portion of the IP address space that is not used by the corporation. Note that zones are required to be disjoint: each IP address can only appear in a single zone.[3]

Service: This is the combination of a protocol-base (e.g., `tcp`, `udp`, etc.) and the port numbers on both the source and destination sides. For instance, the service `telnet` is defined as `tcp` with destination port 23 and any source port. A service group is simply a set of services.

Hosts: A host is specified by a single IP address. A host group is a set of IP address, which may be specified as a range of IP addresses, or as a subnet (using an IP address and netmask), or as a list of hosts, ranges, and subnets.

3 The query engine: under the hood

The query engine is implemented as a library of C functions, which exports an API. Before processing queries, the query engine reads a network topology description file, and the firewall rule-bases. Natively, the query engine supports the configuration file syntax of the Lucent VPN Firewall Brick[4] [25]. Figure 1 shows the data flow through the query engine, as it is used by Fang.

3.1 The topology in the query engine

Before the query engine can be used, it needs to have an instantiated model of the network topology. This is provided in the form of a network topology description file. The language we use to describe the network topology is a subset of Firmato's MDL language. We refer the reader to [2] for more details about MDL. In the Fang prototype, the users were required to manually create this file. The FA creates this file automatically, based on the routing table.

Note that we use the term "topology" somewhat loosely. The query engine does not need to be aware of every router and switch in the network, and is indifferent to the routing scheme that is used. We only care about devices that have

[2] Other actions are usually allowed, such as writing a log record, performing network address translation (NAT), activating deeper payload inspection, or initiating IPSec encryption. We focus only on the basic pass/drop actions, for sake of brevity.

[3] Requiring zones to be disjoint is, occasionally, a significant constraint: There are cases in which the same IP address is used more than once within an organization, e.g., as a result of mergers and acquisitions. Connectivity between such "clones" can only be established using complex NAT rules, but sometimes this is easier to accomplish than renumbering IP addresses. In such cases each part of the network needs to be analyzed separately.

[4] Formerly known as the Lucent Managed Firewall (LMF). We continue using the acronym LMF since the query engine's parser recognizes it as a keyword.

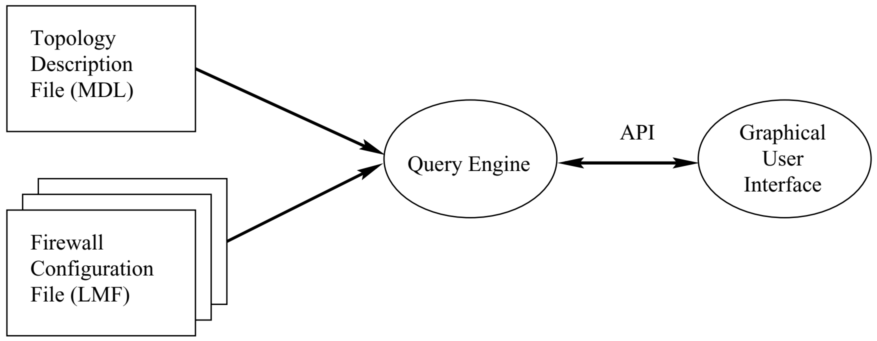

Topology Description File (MDL)
Firewall Configuration File (LMF)
Query Engine
API
Graphical User Interface

Fig. 1 Data flow through the query engine

packet filtering rule-bases installed on them, and about the zones these devices define (recall Sect. 2.2). At this level of granularity network topology is quite stable; the network topology file only needs to be touched if the routing tables on the *firewalls* are changed. Mundane events like a change in internal routing or adding a new network device do not invalidate an existing network topology file. Furthermore, firewalls seldom participate in dynamic routing protocols such as BGP or OSPF (cf. [31]), and in the vast majority of cases we have seen, routes on firewalls are statically defined. Thus, writing or updating the network topology file is a rare event.

As a part of the network topology file, the user specifies the names of the firewall configuration files that contain the rule-bases for all the gateway interfaces. After reading the network topology file, the query engine parses each of these configuration files in turn, and populates its internal rule-base data structures for each device.

The network topology is modeled as follows. The network is partitioned into Zones, which are connected through Gateways. A Gateway has an Interface for each adjacent Zone. Each Interface either has its own IP address (and is considered a Host for some purposes), or is declared to be invisible (using the `INVIS` keyword) if the firewall operates as a bridge. Packets leaving and entering a Zone can be filtered by the Gateway on the corresponding Interface; packets sent and received within the same Zone cannot, simply because they do not pass through any Gateway. Therefore, from the query engine's perspective, there exists a path between any two hosts in the same Zone; any and all filtering is performed by the Interfaces. Zones consist of host groups. Host groups are typically further subdivided into a hierarchy of smaller host groups or single hosts.

3.2 The internal model

3.2.1 Naming

In our model, all the objects (host groups, service groups, gateways, interfaces, zones) have names. This allows us to have a high level of abstraction when interacting with the user. Meaningful names are more expressive than raw IP addresses and port numbers. The query engine obtains these names from the rule-base files: the LMF configuration syntax supports naming.[5] Additionally, the query engine uses a table of "well-known" named service definitions.

Since each device and interface is assumed to have been configured independently, there may be name conflicts. Furthermore, there may be name conflicts between the global "well known" service definitions and the local definitions. For instance, the administrator may have defined the name `http` to signify `tcp` on port 80 on one gateway, while on another gateway she may use the same name to mean `tcp` on ports 80, 8000, and 8080. To support this level of naming, the query engine maintains a separate symbol table context per interface (i.e., per rule-base). If the same name appears in different contexts *with different meanings*, the query engine will use and export all the variants, prefixed with the interface name. Otherwise, if all the variants are identical, the name will appear only once with no prefix.

3.2.2 Rule-bases

The query engine accepts the rule-bases as files written in the LMF configuration language. Each rule-base is associated with an Interface. The query engine parses each of these files into a table of logical rules in memory. The record structure of the rule table (simplified for ease of explanation) consists of the following:

Name	Description
Source	The host group listing the source IP addresses
Destination	The host group listing the destination IP addresses
Service	The service group listing the protocols and port numbers
Direction	IN or OUT or BOTH
Action	PASS or DROP

The details of rule-base semantics differ among different vendors, therefore we need to explain the semantics of

[5] Some vendor products, notably Cisco's IOS and PIX, do not provide good support for names. Therefore, the FA front-ends that translate the Cisco configuration files into the LMF language create names for the various objects. See Sect. 5.6.

the query engine's firewall model. When packets are filtered, the rules in the rule-base are examined in their order until a match occurs (i.e., we use "first-match" semantics). The Source, Destination and Service fields are compared to the corresponding fields in the packet. The direction specifies whether the rule applies to packets entering (IN) or leaving (OUT) the gateway on which this interface sits (i.e., the rules are gateway-centric).[6] The wildcard direction (BOTH) indicates that the rule applies to both directions. If a match occurs, the corresponding action (DROP or PASS) is performed.

The internal model also supports NAT. In the query engine's firewall model, NAT actions are performed at a per-rule granularity. Hence, the `rule` structure has additional fields describing which header fields should be translated, and how. We omit the details. Per-rule NAT is the approach taken by the Lucent VPN Firewall Brick, and by one of Check Point's NAT modes. However, Check Point also has a per-host-group mode of NAT in which a host group can have a valid IP address, and a translated address, and the firewall implicitly creates translation rules. Cisco products perform NAT globally, for the whole firewall, rather than per rule. Therefore, significant parts of FA's front-ends (Sect. 5.4) are concerned with converting the various vendors' NAT definitions into their query engine equivalent.

3.2.3 Queries

A central object in the query engine is a *query*. A query is a triplet, consisting of the following fields:

Name	Description
Source	The host group listing the source IP addresses
Destination	The host group listing the destination IP addresses
Service	The service group listing the protocols and port numbers

The semantics of such a query are "which IP addresses within the Source host group can send packets, with services from the Service group, to which IP addresses in the Destination host group?."

Recall that host groups and service groups may be wildcards, i.e., any element of the query triplet can be the "*" (wildcard), meaning "any." So the question "which machines can use the company's web-servers?" can be expressed by the query `(*, web_servers, http_services)`, assuming that the host group `web_servers` and the service group `http_services` are defined.

It is usually the case that not all the packets described by a query can reach the destination. Through the operations of the various rule-bases, some packets may be dropped. Therefore, the query engine's answer to such a query is a refined *list* of "sub-queries." This is a list of query triplets, where the value in each field in the sub-queries is a subset of the corresponding field in the original query.[7] The semantics of the answer are that for each sub-query in the result, the corresponding source host group can indeed send the service to the destination host group.

3.2.4 The gateway–zone graph

The query engine's search algorithm, described in the next section, is based on a graph algorithm, where the graph is defined by the network topology. For this purpose we use the following auxiliary graph, which we call the gateway–zone graph (see Fig. 2 for an example). The same type of graph was also used by the localization method of [17] and by the Rule Assignment and DIrection Setting (RADIS) Algorithm of [2].

Definition 1 Let the *gateway–zone graph* be a bi-partite graph $H = ((G \cup Z), \mathcal{I})$ whose vertices consist of the set of gateways G and the set of zones Z. The set of interfaces $\mathcal{I}$ forms the edges: H contains an edge $i = (g, z)$ connecting a gateway $g \in G$ to a zone $z \in Z$ iff g has an interface i whose adjacent-zone is z.

There is a `node` for each gateway and zone. For zone nodes, the node is attached to the set of subnets which the zone consists of; for gateway nodes, the node lists the `interfaces` that belong to the gateway, along with their IP addresses and adjacent zones.

Note that IP addresses of an interface appears both as part of the gateway node's host group, and as part of the adjacent zone. This implies that queries that test access to the gateway itself (i.e., to one of its interfaces' IP addresses) have to be flagged as such and preformed separately. The reason is that, otherwise, the query engine would not have been able to determine which of the two occurrences is referred to by the query.

3.3 The query engine algorithm

The query engine consists of a graph algorithm and a rule-base simulator. It takes as input a query (recall Sect. 3.2.3), and uses the gateway–zone graph data structures. The algorithm simulates the behavior of all the packets described by the query as they traverse the network.

3.3.1 Simulating a rule-base

The basic step of the algorithm is propagating a query over an edge in the gateway–zone graph, which represents a

[6] The semantics of the direction field, in all firewall configuration languages we are aware of, are arbitrary and particularly confusing to users. See [35] for a discussion.

[7] The answer triplets actually contain a few additional fields that deal with NAT: e.g., we maintain the original source host group when a rule replaces it with a NAT host group. We omit the details.

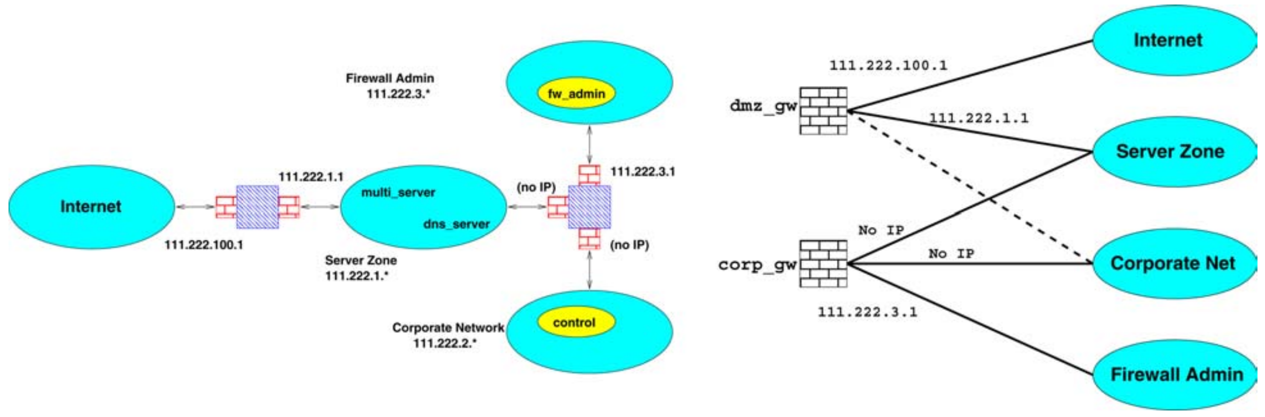


Fig. 2 A simple network topology diagram on the left, and its matching gateway–zone graph $H = ((G \cup Z), \mathcal{I})$ on the right. Interface edges shown as solid lines. G consists of two gateways, Z consists of four zones. The edges in $\mathcal{I}$ are labeled by the IP addresses of the interfaces they represent. Adding a third interface to dmz_gw, whose adjacent zone is the corporate net, would add the dashed line to the gateway–zone graph and create a cycle

firewall interface. This models the effect of the rule-base that is attached to the interface on the packets described by the query. Typically, only portions of the query can cross any given edge, since some of the packets would be dropped by the interface. Therefore, after crossing an edge, the query may need to be broken up into a set of more refined queries, that represent only those packets that would have been allowed through. For instance, the original query may have been (corporate_net, internet, *), but the rule-base only allows outgoing http and smtp, so the set of queries that reaches the other side of the edge is now (corporate_net, internet, http), (corporate_net, internet, smtp).

The fundamental operations the simulation uses are host group and service group intersection and difference operations. Each host group or service group consists of one or more ranges (of IP addresses, or of port numbers, respectively). Thus a query, or a rule, can be viewed as a collection of hyper-cubes in four-dimensional space (it does not make sense to have a *range* of values in the protocol field, hence the hyper-cubes are not five-dimensional). Computing the intersection or difference between such cubes is straightforward.

Recall that all the objects in the internal model have names. This includes the host groups and service groups produced by a rule operating on a query. However, the resulting groups do not always have pre-defined "nice" names. For instance, a rule that drops all netbios traffic may produce a service group consisting of "all services except tcp on destination ports 137, 138, and 139," which may not have a user-defined name. Therefore, the algorithm checks whether there exists a definition that fits each created group. If an appropriate definition is found, it is used. Otherwise, the algorithm dynamically creates a new one. If the rule was a PASS rule (intersection action), the name is (group1&group2). For drop rules (difference action) the name is (group1^group2). Multiple rules acting on a query can produce fairly long names, like (((all_tcp^serv1)^serv2)^serv3), so beyond some point they are truncated.

3.3.2 Searching the gateway–zone graph

Initially, the user's query is attached to the node in the gateway–zone graph which contains the source host group.[8] Then, the algorithm attempts to propagate the query over all the edges (interfaces) that touch the current node, by simulating the effect of the rule-bases associated with each edge. It then continues in the same manner, propagating the query further until it searches the entire graph. The search does not attempt to only reach the destination, and does not terminate when the destination is reached. Rather, it continues searching until no additional sub-queries cross into any node in the gateway–zone graph.

A feature of the search algorithm is that it floods the gateway–zone graph with the query, and lets the query attempt to cross all possible interfaces out of every gateway it reaches. Note that some nodes may be visited more than once, while other will not be visited at all. This is because the algorithm backtracks over all possible paths the query can take through the network. If the query can reach a node v (whether a zone or a gateway) via different paths, the new query that is attached to v is the union of the query results reaching v on each of the possible paths.

At first sight, one may wonder why an algorithm like Breadth-First Search (BFS) or Depth-First Search (DFS) does not suffice, and why we need to visit the same node multiple times. The reason is that cycles may exist in the network, and hence, packets may have multiple ways to reach their destination. Since we do not model the routing pre-

[8] Recall that the zones are required to be disjoint, so every IP address in the source host group appears in exactly one zone. If the source host group is not contained in a single zone (e.g., when the wildcard, *, is used), the source host group is broken up into disjoint host groups, each of which is contained in a zone. A separate graph search is performed for each host group.

cisely, the algorithm cannot assume that the packets reaching a zone z via some particular path from source s are the *only* such packets that could reach z, and we need to try all the other paths too.

An advantage of this approach is that it allows us to handle NAT relatively easily: If a rule-base on some interface causes part of a query to be translated, then the resulting sub-query simply has its source or destination host group replaced by its NAT-ed counterpart. Note that NAT can change the routing of packets if it translates the destination IP address. This is another reason why we cannot just search for the path between source and destination, and subsequently "push" the query along that path: The path itself might be changed by the rule-bases that operate on the query.

We follow [2, 17] and make one assumption about the routing.

Assumption 1 *Packets are never routed in cycles.*

In particular, the algorithm ignores the possibility of "U-turns": We assume that if the query crossed interface i from gateway g into zone z, there is no need for it to try and cross the same i back into the gateway g, since packets will not be routed this way. Note, though, that such "bounce routing" may occur, for instance, as a side-effect of network address translation, in which case the search algorithm may miss valid traffic paths. However, bounce routing is very rare in practice, and when it does occur it is typically the result of a routing mis-configuration.

The search is also slightly optimized to not cross an edge if it is clear that no new packets will be allowed through that have not been already allowed by other paths.

After the search has terminated, we need to collect the results. This simply involves looking at the node or nodes that contain the destination host group and picking out those queries that have reached their correct destination.

3.3.3 Complexity

In the worst case, the complexity of the algorithm is exponential in the size of the gateway–zone graph. However, this worst case can only occur in very dense graphs. Typical gateway–zone graphs are very sparse, since firewalls are normally placed at strategic choke points in the network, and the most common gateway–zone graph topology is a tree. On a tree topology the algorithm is essentially a DFS, i.e., its time complexity is linear in the size of the graph. Furthermore, since we only model zones that are separated by firewalls, gateway–zone graphs tend to be quite small—the largest gateway–zone graphs we have seen were trees with under 30 zones.

The running time for having a query cross an interface i is clearly linear in the number of rules r_i on the interface. Furthermore, the resulting number of sub-queries that are formed as a result of crossing an interface can also be linear in r_i: A PASS rule can produce one result sub-query, and a DROP rule or a rule with a NAT action can produce up to 16 result sub-queries.[9] Therefore, crossing a sequence of d interfaces, all with rule-bases of r rules, incurs a time and space complexity of $O(r^d)$.

From a theoretical viewpoint, the exponential dependence on the graph size is more worrisome than the polynomial dependence on the number of rules. In reality, though, it's the other way around. As we noted earlier, the graphs are usually very small and have paths of length 2 or 3. However, we have encountered large rule-bases with many thousands of rules. For such rule-bases, and especially when the number of DROP rules is high, we have observed a significant slowdown of the algorithm. On a 550-MHz Pentium III with 256 MB RAM, with a mid-sized configuration (a four-interface Cisco PIX firewall with 500–1500 rules), a query takes approximately 0.25–0.39 s to complete.[10] However, in a handful of cases we have seen queries taking close to 15 s: one pathological example was a Check Point firewall rule-set with ≈4000 rules, 2600 of which were NAT rules.

3.4 Rule locators

One of the lessons we learned from the Fang prototype was that it is important to track which rules are responsible for a query reaching its destination. This serves two purposes: First, it tells the user which rules need to be modified if the query result indicates a problem. Second, it increases the user's confidence in the simulation process, as the user is able to look at the "culprit" rule and convince herself that the simulation produced a correct result. For this purpose, the query structure has a `rule_locators` array, that lists all the rules that affected the query along its path through the gateway–zone graph.

Determining which rules affected a query is not always clear-cut. Consider, for example, the following rule-base.

	Source	Destination	Service	Action
1)	*	*	`netbios`	*DROP*
2)	Inside	Outside	*	*PASS*
3)	Outside	`web_server`	`http`	*PASS*
4)	*	*	*	*DROP*

If the query is (Inside, Outside, `*`) then, obviously, both rules 1 and 2 affect the result. However, if the query is (Outside, Inside, `*`), rule 3 certainly affects the result, but does rule 1 affect the result? On one hand, rule 1 did match part of the query, and its action dropped the `netbios` service. On the other hand, the query result would have been the same (Outside, `web_server`, `http`) even if rule 1 was omitted. In our experience, users say that rule 1 does *not* affect the result of the query, and they do not want to see rule 1 listed in the rule locator. Therefore, the query engine

[9] In one dimension, removing a range from a larger range produces two sub-ranges. A rule has at most four dimensions (source and destination IP addresses, source and destination port numbers), so a single DROP rule can break up the original query into $2^4 = 16$ sub-queries.

[10] The pre-processing and post-processing overhead is included in the time measurements and amortized over all the queries.

uses a heuristic to decide whether to append the number of a matching rule to the rule locator array, or to replace all previously listed DROP rules by the current matching rule.

Note that a rule locator is part of the query structure. This implies that the query engine only keeps track of the parts of the query that *do* manage to cross the current interface. The query engine does not track which rules dropped the other parts of the query.[11] This is an intentional part of the query engine design, since it is primarily a security assessment tool—and packets that are dropped do not pose a threat to the network's security. However, more detailed tracking of DROP rules would help a firewall administrator that is trying to understand why some needed service is being dropped. Adding this support to the query engine is left for future work.

3.5 Spoofing

It is well known that the source IP address on an IPv4 packet is not authenticated. Therefore, source addresses may be spoofed (forged) by attackers in an attempt to circumvent the firewall's security policy (cf. [38]). For instance, consider the very common firewall rule "From IP addresses in `MyNet`, to anywhere, any service is allowed." Assuming that `MyNet` is behind the firewall, this rule is supposed to allow all Outbound traffic from hosts in `MyNet`. However, an attacker on the Internet may spoof a packet's source address to be inside `MyNet`, and set the destination address to some IP address behind the firewall. Such a spoofed packet would clearly match the above rule, and be allowed to enter. Obviously, the attacker will not see any return traffic, but damage has already been done: this is enough to mount a DoS[12] attack against hosts behind the firewall, and is sometimes enough to hijack a `tcp` session [3].

Most firewall vendors provide mechanisms to combat spoofing attacks, based on the direction that a packet is traveling: which interface it is crossing, and whether it is entering or leaving the firewall [35]. These anti-spoofing mechanisms essentially produce rules (sometime implicit or hidden rules) that are associated with the firewall interfaces, and ensure that packets only travel in directions that are consistent with the location of their claimed source address. Both Check Point FireWall-1 and Cisco PIX provide such capabilities.

A small extension to the basic algorithm allows testing for spoofing attacks. In addition to the source, destination and service parameters that define a query, we add an optional fourth parameter which specifies the zone from which the packets originate. When this fourth parameter is defined, the source host group is then understood to describe the fake (spoofed) source addresses. Processing such a query is identical to that described earlier, except that instead of starting from the gateway–zone graph nodes that contain the fake source host group, the algorithm starts at the originating zone's node. A typical spoofing query would look like `(my_net, my_net, *)` originating from zone `Internet`. Such a query tests which packets, whose source IP address is spoofed to be internal, would be able to cross into `my_net` if they reach the firewall from the Internet zone.

[11] The only exception is when a DROP rule "clips" the query but part of the query still manages to pass.

[12] Denial of Service.

4 The Fang research prototype

4.1 Overview

Fang has a graphical user interface (GUI) which was developed using Qt [7, 27], a C++ class library for writing portable GUI applications (see Fig. 3).

After launching Fang, the user needs to read in the MDL network topology file (via `File`→`Open` menu), to initialize the query engine. The MDL network topology file tells the query engine where the device configuration files of each firewall/filter device can be found. After all the files have been parsed, the query engine exports the names of all the host groups and service groups it found in the configuration. Given these names, the Fang GUI creates its drop-down menus, and is ready to accept user queries.

The user forms a query by choosing each element of the query triplet from a drop-down menu offering the choice of all the host groups or service groups that were defined in the configuration files. After clicking on the Submit button, the answer is presented as a list of triplets in the box below. Each result triplet can be expanded to offer more detailed information via clicking on the [+] icon (see Fig. 4).

Figure 3 shows the GUI with the spoofing option turned on. When the option is turned off, the rightmost drop-down menu disappears.

4.2 Examples using Fang

Figure 4 shows the results for the query: "What services can be initiated from the corporate zone into the DMZ?" against a simple firewall configuration. We can see that `ftp`, `http`, and `dns` are available from all the hosts in the corporate zone, and some additional services are allowed from a special host called `control`, that is located inside the corporate zone. Note that only the names of the host groups and service groups are displayed, but more detail, like actual IP address and port numbers, is available by expanding an entry.

Figure 5 illustrates how to check for spoofing. In it, we set the originating zone to be the Internet, and let the spoofed source address be arbitrary. In this example we can see that, on the analyzed configuration, any Internet host can create a packet with a spoofed source address of `I_dmz_in`, and such a packet will be able to reach the `fw_admin` machine. Such a situation can arise when the firewall filters traffic based upon source IP addresses without taking the direction of traffic flow into account (cf. [34]).

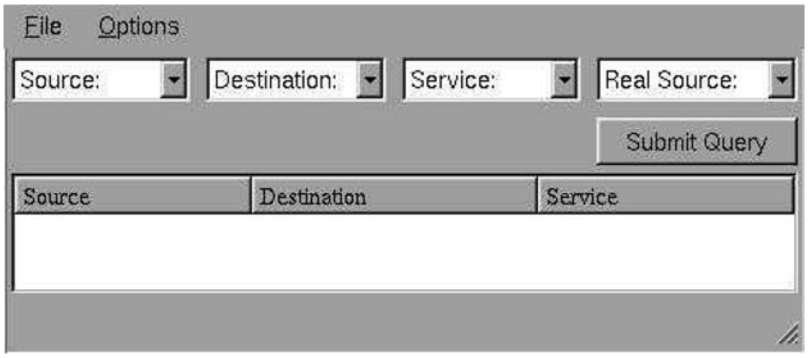
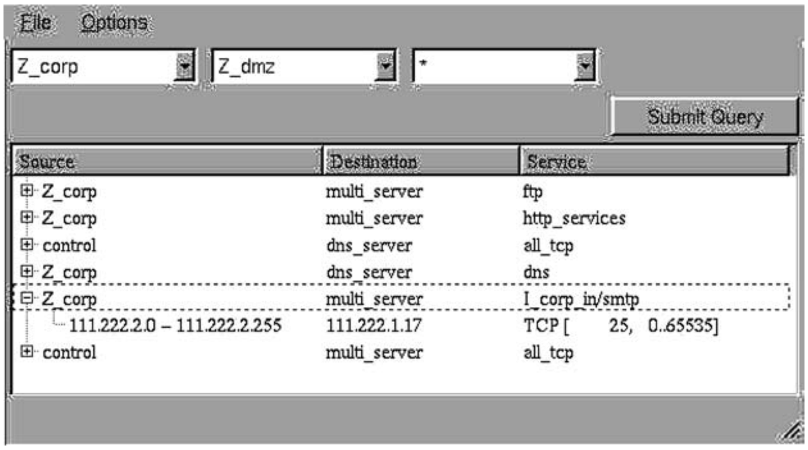
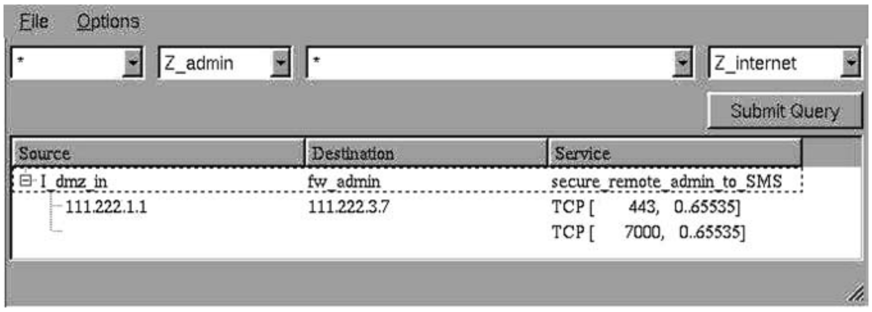


Fig. 3 GUI for Fang



Source	Destination	Service
⊞ Z_corp	multi_server	ftp
⊞ Z_corp	multi_server	http_services
⊞ control	dns_server	all_tcp
⊞ Z_corp	dns_server	dns
⊟ Z_corp	multi_server	I_corp_in/smtp
111.222.2.0 – 111.222.2.255	111.222.1.17	TCP [25, 0..65535]
⊞ control	multi_server	all_tcp

Fig. 4 Fang results for a simple query



Source	Destination	Service
⊟ I_dmz_in	fw_admin	secure_remote_admin_to_SMS
111.222.1.1	111.222.3.7	TCP [443, 0..65535]
		TCP [7000, 0..65535]

Fig. 5 Fang results for a spoof-attack query

5 The firewall analyzer

5.1 Overview

The main contribution of the Fang prototype was its query engine. The combination of its internal network topology model, data structures, and efficient algorithms, demonstrated that it is feasible to analyze 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. The feedback raised the following issues we needed to address:

- Before Fang could be used, it needed to have an instantiated model of the network topology. Therefore, before querying the firewall policy, a Fang user needed to write a network topology description file using the Firmato MDL language [2]. Users found this to be a difficult and error-prone procedure.
- Using a GUI as a query input mechanism turned out to be inefficient for users. Furthermore, users did not know

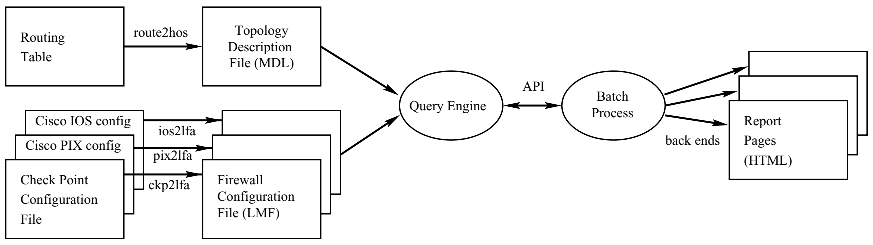


Fig. 6 Data flow through the FA

which queries to issue, and lost interest after the queries they did issue exposed no surprises.

- The Lucent VPN Firewall Brick had a fairly small market share, and we needed to add support for the market leading firewall vendors: Check Point and Cisco.
- The output display mechanism in Fang was not flexible enough. Users wanted to see high-level results, but also to have the ability to drill down to low-level details.

To address these issues, the FA architecture introduced the following new features:

- The user no longer needs to write the network topology file. FA has a new front-end module that takes a formatted routing table and automatically creates the network topology file.
- Instead of a GUI-based program, FA 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 FA output is now formatted as a collection of HTML web pages, with some JavaScript code for easier navigation of the reports.
- We added support for Check Point FireWall-1, Cisco PIX, and Cisco IOS router access lists. The FA uses an intermediate firewall configuration language, to which we convert the various vendors' configurations.

The details of the new features, with some examples, are described in the following sections. Figure 6 illustrates the data flow through the various FA modules.

5.2 Describing the network topology

As we mentioned earlier, before issuing any queries, a Fang user needed to write a network topology description file, using the Firmato MDL language [2]. The network topology description file contains a definition of every zone in the network (recall Sect. 2.2). The zones 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, the query engine 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 network topology file caused two problems for Fang users. First, they 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 network topology 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 rarely 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 FA introduced a new front-end module, called `route2hos`, that mechanically converts a routing table of a single firewall into a Firmato MDL network topology file. All that is required from the user is to provide the firewall's routing table. The `route2hos` module is able to parse the routing table formats produced by several flavors of the Unix `netstat` command, the Microsoft Windows `route print` command, and the routing tables that appear in Cisco's PIX and IOS configuration languages. Section 6 describes how `route2hos` works.

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.[13] This host group typically includes the Internet, and any of the corporation's subnets that are external to the firewall. The Inside host group consists of all other IP addresses. These two host groups are later used in the query processing.

Remark As we saw in Sect. 3.3.2, the query engine's search algorithms and the Firmato MDL language are capable of handling topologies that are much more complicated than a single firewall and its adjacent zones: The query engine is able to simulate several interconnected firewalls at once, and

[13] For Cisco PIX configurations, the Outside host group consists of the IP addresses that are routed via the interface with PIX trust level 0, regardless of whether that interface is the default route or not.

show their combined effect in its output. However, the current version of `route2hos` hides this advanced capability, as it is able to handle a single routing table at a time. Simulating multiple firewalls in a way that allows for simple user input is left for future work.

5.3 What to query?

As we saw in Sect. 4, the Fang prototype had a GUI which allowed users 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 rulebase are often precisely those queries that the user does not know to try.

To solve these problems, the 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 FA, and replaced it by a batch process, which repeatedly calls the 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 FA to circumvent this combinatorial explosion: (i) the query engine processes aggregate queries very efficiently, and (ii) after the `route2hos` processing the FA knows which IP addresses are external to the firewall (this is the Outside host group). Combining these two facts, FA can issue the query "list the types of traffic that can enter from the Outside to the Inside using any service," which is expressed by the query triplet `(`Outside`, `Inside`, *)`. The result is a list of `(src, dest, srv)` result sub-queries 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, FA can make the outgoing query `(`Inside`, `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 `(`Outside`, `Inside`, *)` would produce the response `(`Outside`, 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 were obvious to the user, perhaps 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 FA in fact does not make the query `(`Outside`, `Inside`, *)`. Instead it issues a set of focused queries: `(`Outside`, `Inside`, dns)`; `(`Outside`, `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, FA may miss some services. A service that is not on the FA's list of "known services," and does not appear explicitly on any rule, will not be queried.

To ensure this does not happen, FA 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, FA makes queries of the form `(`Outside`, internal-host-group, *)`, where "`internal-host-group`" goes over every internal host group.[14] FA then goes over the internal host groups again, making outbound queries of the form `(internal-host-group, `Outside`, *)`.

Most modern firewalls have more than two interfaces. The networks attached to these interfaces have various levels of trust, since they include connections to business partner networks, DMZs, etc. Therefore, in addition to traffic that crosses the network perimeter (Outside to Inside or vice versa), the FA queries "internal" traffic. This is done by issuing a set of by-service queries of the form `(`Inside`, `Inside`, srv)`, and another set of by-host-group queries of the form `(`Inside`, internal-host-group, *)`.

Finally, it is very important to analyze the access to the firewall itself. If the firewall itself is not adequately protected then the whole network behind it is vulnerable. This is done by issuing queries of the form `(zone, fw, *)`, where `zone` goes over all possible zones (the Outside zone plus all the internal zones), and `fw` is a host group consisting of the firewall's interfaces (recall Sect. 3.2.4).

The results of all these queries are organized into seven report pages: a "Access to the firewall" report, and two sets of three reports each, one set organized by service, and the

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

other organized by host group. Each set of reports contains an Incoming report, an Outgoing report, and an Internal report.

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.

Remark As we saw in Sect. 3.5, the query engine is able to analyze possible spoofing vulnerabilities. However, at the time of writing, the FA batch process does not utilize this advanced capability. Adding support for spoofing queries is left as future work.

5.4 Supporting multiple vendors

The query engine uses a model of the firewall rule-base, which is generic and vendor-independent. However, in order to instantiate this model, the FA 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 FA model. The query engine provides native support (within the C code) only for the Lucent VPN Firewall Brick [25] 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 query engine. Instead, we opted for an architecture centered around an intermediate language. We chose to write a separate front-end conversion utility, written in the Perl programming language, for each supported vendor. The front-ends would take the vendor's files and translate them into the FA'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 VPN Firewall Brick (a.k.a. LMF) language.

Access-control-list languages such as Cisco's IOS [20] and PIX [5] configuration languages, or the Linux `ipchains` (cf. [30]) 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 greatly increase the configuration size.[15]

Check Point (cf. [32]) uses two separate languages in the configuration of their FireWall-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 FA, 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.[16]

5.5 Presentation of results

As we saw in Sect. 4, the Fang GUI also displays the query output to the user. The GUI has a basic mode showing the names of the sources, destinations, and services in the resulting `(src, dest, srv)` tuple. The user has 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 query engine so it will output 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 FA intermediate language.

Services. This page shows a table of all the service definitions (protocols and port numbers), with the containment relationships[17] 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 seven query reports we mentioned in Sect. 5.3:

[15] 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.

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

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

Access to the firewall, Analysis by service (Incoming, Outgoing, and Internal), and Analysis by host group (Incoming, Outgoing, and Internal). Each query result triplet 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(s) allowing the traffic through, as reported by the rule locator structures (Sect. 3.4). A typical FA report contains hundreds or thousands 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. Furthermore, the report itself can easily be packaged and viewed on a variety of platforms without the need for any specialized report-viewer software. Section 7 contains excerpts from some of the produced web pages.

5.6 Naming things

As we mentioned in Sect. 3.2.1, the query engine attaches a name to every object. For services and service groups, we use several sources of naming information. First, the FA 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, FA displays it as `https`. Second, most firewalls have built-in name definitions which we use. Finally, for firewalls that support user-defined services, we read those names in. All these definitions are converted by the front-end modules into the LMF language for the query engine to parse.

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 [20] access-control-lists), we use the IP addresses themselves as the name. In addition, in all cases, FA 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, FA uses a heuristic to pick a representative IP address in the subnet, and looks up that IP address’ name.

5.7 Check point-specific features

The Firewall Analyzer (FA) front-end `ckp2fa`, that converts Check Point FW-1 configurations into the FA 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 `ckp2fa` 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 `ckp2fa` 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 FA 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 `ckp2fa` 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.

Note, though, that Check Point FW-1 supports additional features that FA does not handle, or handles partially. For instance, FW-1 supports the notion of a named “User Group” and lets the administrator specify such a group as a source in a VPN-type rule. The members of a User Group are authenticated by some means (like a password or hardware token). Since the FA deals only with IP addresses, it simply assigns a fake IP address (like 0.0.0.1) to the User Group. Other Check Point features that are ignored include “resources” (which are external proxies for handling specific protocols) and layer 7 inspection (“SmartDefense”).

6 Converting a routing table into a topology file

The information about which IP address is located behind which of the firewall’s network interface cards (NIC) is encoded in the firewall’s routing table. However, routing table entries are rarely disjoint: It is common to have many overlapping routing table entries that cover the same IP address. The “best-route” 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. Therefore, we need to convert the information from the routing table’s “best-route” semantics into disjoint zones.

6.1 Routing table basics

A full explanation of IP routing is beyond the scope of this paper. The interested reader is referred to [21, 31]. Below we only touch upon the points that are relevant to our discussion.

In IPv4, every IP gateway maintains its routing information in the form of a *routing table*. Each entry in the table is called a *route*, and describes how the gateway should deal with packets destined to a given range of IP addresses. The range of addresses described by a route is always a subnet, specified as a CIDR[18] block; i.e., it is specified as an IP address, with a *netmask* that indicates which bits are "don't-care" bits.

The routing table distinguishes between routes to *directly connected* subnets, and all other routes. Directly connected subnets are subnets that the gateway is connected to via one of its interfaces: the IP address of that interface belongs to the subnet. The gateway can communicate with IP addresses on a directly connected subnet using layer 2 protocols. To communicate with other IP addresses, the gateway has to send its packets via some other gateway. Thus, for directly connected subnets, the routing table lists the NIC which is connected to the subnet. For other subnets the routing table lists the IP address of the next-hop gateway.

When a routing decision is being made for a given IP address a, the gateway needs to search the routing table for the route leading to a. Note that multiple subnets that contain a are often present in the routing table. The choice among these candidate routes is made by "best-route" semantics: the smallest subnet (in terms of how many IP addresses belong to it) is selected.

A routing table usually (but not always) has a special route called the *default route*. This is the route that is taken if none of the other routes applies to the IP address a. In some implementations the default route is identified by a special keyword such as "default." In other implementations its subnet is simply identified by the IP address 0.0.0.0 with netmask 0.0.0.0 (i.e., all the bits are "don't-care" bits). A typical gateway has its default route pointing toward the public Internet. However, a default route may be intentionally missing if the routing policy at the gateway allows no traffic to the Internet. See Fig. 7 for an example.

6.2 The algorithm

The main goal of the conversion algorithm is to compute the zone connected to each of the gateway's interfaces: i.e., list all the IP addresses that are routed over each NIC. Recall that the zones are required to be disjoint. In addition, if a default route exists in the routing table, then the algorithm uses the zone to which the default route leads to define the Outside host group. The algorithm is a one-dimensional "line sweep" algorithm (cf. [8]).

[18] Classless InterDomain Routing.

The algorithm uses the notion of *critical points* which are defined as follows:

Definition 2 For a subnet s, let $low(s)$ denote the first (lowest) IP address in s, and let $high(s)$ denote the last (highest) IP address in s.

Definition 3 A *critical point* in a given routing table is any IP address a that meets one of the following criteria:

1. $a = 0.0.0.0$.
2. $a = low(s)$ for some route.
3. $a = high(s) + 1$ for some route.

In the example of Fig. 7 there are six critical points: 0.0.0.0, 132.66.0.0 (lowest in the 1st and 2nd routes), 132.66.1.0, (immediately follows the highest IP address of the second route), and similarly 132.67.0.0, 172.16.200.0, and 172.16.201.0.

A key observation is that if one sweeps over all possible IP addresses, starting from 0.0.0.0 in increasing order, then the routing decision at IP address a can only differ from the decision at $a - 1$ if a is a critical point of the routing table. Based on this observation, the algorithm consists of the following steps:

1. Input: The gateway's routing table R, containing $r = |R|$ routes.
2. Identify the directly connected subnets.
3. Associate an NIC with every route in the routing table.
4. Identify and sort the routing table's critical points.
5. Determine the routing decision at every critical point a, and the NIC leading to a.
6. Output: the disjoint zones.

For the purpose of calculating the time complexity of the algorithm, in the following subsections we assume that the number of NICs is constant and is negligible in comparison to the number of routes r.

6.2.1 Identify the directly connected subnets

In this step the algorithm builds a lookup table D indexed by the gateway's NICs. D lists the directly connected subnets attached to each NIC. A directly connected subnet is marked as such in the routing table R (e.g., by a keyword such as "directly connected" or an equivalent marker). The complexity of this step is $O(r)$.

6.2.2 Associate an NIC with every route

In this step the algorithm builds an Annotated Routing Table T in which every route includes a field listing the interface over which it leads. For a directly connected subnet $s1$, $T(s1)$ lists the NIC that connects the gateway to $s1$. For a non-directly connected subnet $s2$, let $g(s2)$ denote the next-hop gateway associated with $s2$ in R. Let i denote the NIC for which the subnet $D(i)$ contains $g(s2)$. Then $T(s2) = i$. Note that in the example of Fig. 7 this step is vacuous since the Linux operating system already listed a NIC for each route. This is not the case for other routing table formats. The complexity of this step is $O(r)$.

```
Destination    Gateway       Genmask          Flags   Iface
132.66.0.0     0.0.0.0       255.255.0.0      U       eth0
132.66.0.0     0.0.0.0       255.255.255.0    U       eth1
172.16.200.0   0.0.0.0       255.255.255.0    U       vmnet8
0.0.0.0        132.66.48.1   0.0.0.0          UG      eth0
```

Fig. 7 An example of a routing table (output of the Linux `netstat -rn` command). The gateway in this example has three NICs (eth0, eth1, and vmnet8). In this output format, directly connected subnets are identified by "0.0.0.0" in the second (Gateway) column. The last row shows the default route

6.2.3 Identify and sort the critical points

In this step the algorithm builds a list C of unique critical points (according to Definition 3) that is sorted in increasing order of IP addresses. An easy way to create C is to list all the critical points, sort them, and then eliminate all the duplicates: clearly the time complexity of this step is $O(r \log r)$.

6.2.4 Output the disjoint zones

In this step the algorithm uses a module *RoutingDecision*(T, a) that computes the best-match routing decision for an IP address a using the annotated routing table T. Specifically, if module *RoutingDecision*(T, a) returns i it means that a packet destined for a is routed over interface i according to routing table T. There are many possible search data structure for the *RoutingDecision* module, whose search time is $O(\log r)$ or better.

The algorithm performs a "sweep" over the critical points. For each critical point, it computes the routing decision. If the result differs from that at the previous critical point, the currently open range is output, and a new range is opened. Precisely, the sweep algorithm works as follows:

```
$low = undef$; $current\_nic = undef$
For all $a$ in $C$ (in ascending order) do
    $i = RoutingDecision(T, a)$
    if ($i \neq current\_nic$) then
        Output the range $[low, a-1]$ as behind interface
            $current\_nic$
        $low = a$
        $current\_nic = i$
    endif
enddo
Output the range $[low, 255.255.255.255]$ as behind
    interface $current\_nic$
```

Since the sweep algorithm issues at most two calls to *RoutingDecision* per route in the routing table, its time complexity is $O(r \log r)$ (or better, according to the data structure used in the *RoutingDecision* module).

6.2.5 Creating the inside and outside host groups

After the disjoint zones behind all the NICs are computed, we can build the Inside and Outside host groups (recall Sect. 5.2). These host groups are computed as follows: Let i_d be the NIC to which the routing table's default route points. Then the Outside host group contains the IP addresses in the zone behind i_d, and the Inside host group contains all other IP addresses.

Note that if the routing table does not contain a default route, which occurs occasionally, either intentionally or due to configuration error, then the Outside host group is undefined and the firewall analysis cannot proceed. In such cases the user needs to manually identify the Outside and Inside host groups.

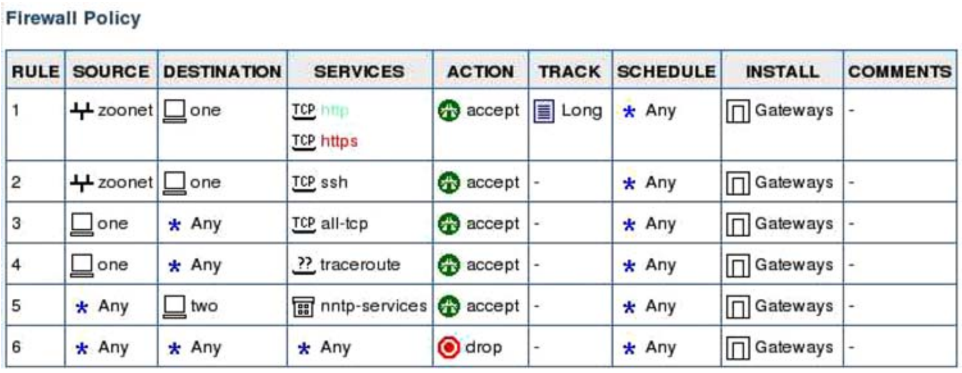
7 An example

In this section we show an annotated example which illustrates the flow of data through the various components of the FA. 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 FA 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 FA. The report excerpt shown here is from the lab machine. The full web-based sample report is available on-line from [1].

In Fig. 8 we see a web page showing a Check Point FW-1 rule-base. This is the FA'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 [36].

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 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`.

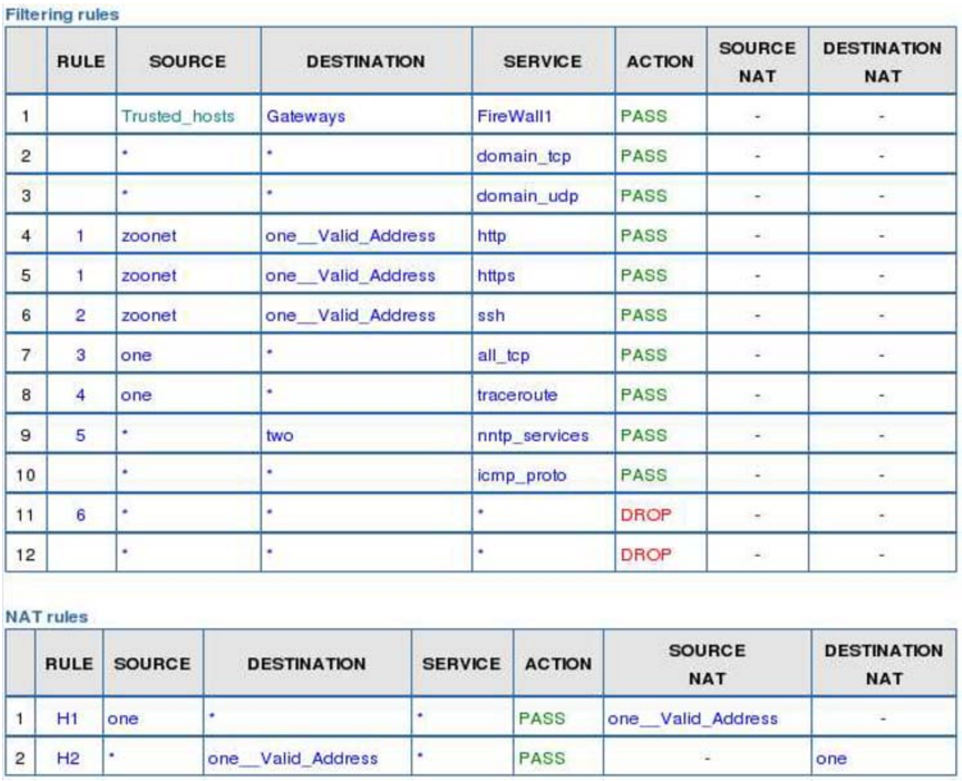
In Fig. 9 we can see an HTML rendering, produced for the same rule-base, of the FA's intermediate language, as discussed in Sect. 5.4. The figure shows the results of the Check-Point-to-FA front-end conversion utility, `ckp2fa`, post-processed into an HTML-based report (called the Expanded Rules report) by the back-end utilities.



Firewall Policy

RULE	SOURCE	DESTINATION	SERVICES	ACTION	TRACK	SCHEDULE	INSTALL	COMMENTS
1	zoonet	one	TCP http TCP https	accept	Long	Any	Gateways	-
2	zoonet	one	TCP ssh	accept	-	Any	Gateways	-
3	one	Any	TCP all-tcp	accept	-	Any	Gateways	-
4	one	Any	?? traceroute	accept	-	Any	Gateways	-
5	Any	two	nntp-services	accept	-	Any	Gateways	-
6	Any	Any	Any	drop	-	Any	Gateways	-

Fig. 8 The original rule-base, rendered in HTML



Filtering rules

	RULE	SOURCE	DESTINATION	SERVICE	ACTION	SOURCE NAT	DESTINATION NAT
1		Trusted_hosts	Gateways	FireWall1	PASS	-	-
2		*	*	domain_tcp	PASS	-	-
3		*	*	domain_udp	PASS	-	-
4	1	zoonet	one__Valid_Address	http	PASS	-	-
5	1	zoonet	one__Valid_Address	https	PASS	-	-
6	2	zoonet	one__Valid_Address	ssh	PASS	-	-
7	3	one	*	all_tcp	PASS	-	-
8	4	one	*	traceroute	PASS	-	-
9	5	*	two	nntp_services	PASS	-	-
10		*	*	icmp_proto	PASS	-	-
11	6	*	*	*	DROP	-	-
12		*	*	*	DROP	-	-

NAT rules

	RULE	SOURCE	DESTINATION	SERVICE	ACTION	SOURCE NAT	DESTINATION NAT
1	H1	one	*	*	PASS	one__Valid_Address	-
2	H2	*	one__Valid_Address	*	PASS	-	one

Fig. 9 The expanded rule-base, after conversion to the FA's intermediate language

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 `ckp2fa` 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 9 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 `ckp2fa` converts the implicit rules into explicit ones, we can see that rules 2, 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 [34].

Another piece of information that is clear after the `ckp2fa` conversion is that the firewall is actually performing NAT on the address on machine `one`: The additional NAT rule table shows 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 route2hos front-end, which converts the firewall's routing table into a Firmato MDL network topology file. Instead of showing the network topology file itself, in Fig. 10 we show a graphical representation of the network topology, which is derived from the MDL network topology file using the graph visualization tool dot [14, 16]. We emphasize that Fig. 10 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 FA intermediate language, and the routing table has been converted into an MDL network topology file, the FA proceeds to simulate the configured policy. This is done by the query engine (Sect. 5.3). The output of the query 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 Fig. 11 we see a portion of the "Analysis by service: Incoming" HTML-based report, which is one of the seven reports that FA creates. The figure shows the results of the query (Outside, Inside,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 Sect. 5.6). 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 Fig. 11 are hyperlinks. Clicking on the "9" link brings the user to the Expanded Rules report (recall Fig. 9), with rule 9 highlighted. Looking back at Fig. 9, 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 (Fig. 9) brings the user to the Services report, the relevant portion of which is shown in Fig. 12. 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 technique known as "firewalking," and is usually done using source port 53 (dns) which is very often open [15].

Remarks:

- A manual inspection of the rule-base shown in Fig. 8, even by an expert auditor, is very likely to miss the vulnerability that the FA 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 FA, 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.

8 Related work

There are many firewall products in the market, from vendors such as Check Point [32], Cisco [5, 20], Lucent Technologies [25], NetScreen, and Network Associates, just to mention a few (See [12] or [22] for lists of vendors). Additionally, there are many books on firewall technology and on how to build your own firewall (e.g., [6, 38]). While most of the firewall offerings include configuration tools with varying degrees of sophistication, none of these vendors seems to focus on firewall and security policy analysis tools.

Note that the issue of testing a firewall product (see, e.g., [28]) is different from our goals. We assume that all deployed filtering devices work properly and we are interested in testing the configuration of these devices.

Guttman's work on filtering postures [17, 18] introduced a Lisp-like language which is used to define a filtering policy. Also, a method for localizing the policy to the different interfaces of a filtering router is given (where the local policies are again expressed in the same Lisp-like formalism). This formalism allows making and verifying statements about the policy.

Hazelhurst et al. [19] suggested algorithms that represent a firewall's policy using ordered binary decision diagrams. Their method can be used in two ways: to improve firewall performance, and algorithms to validate the rule sets.

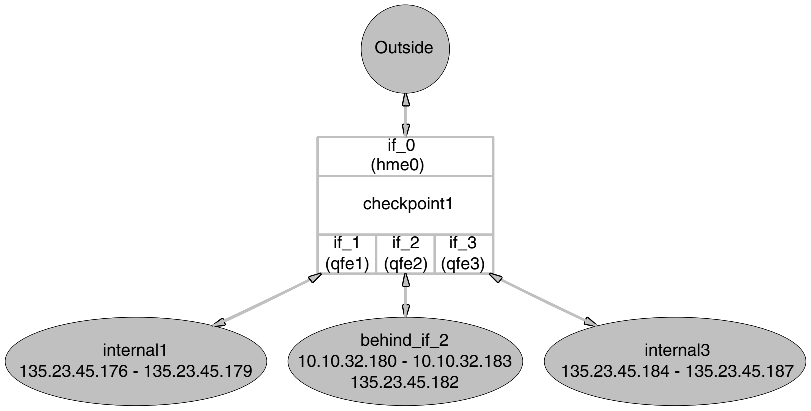
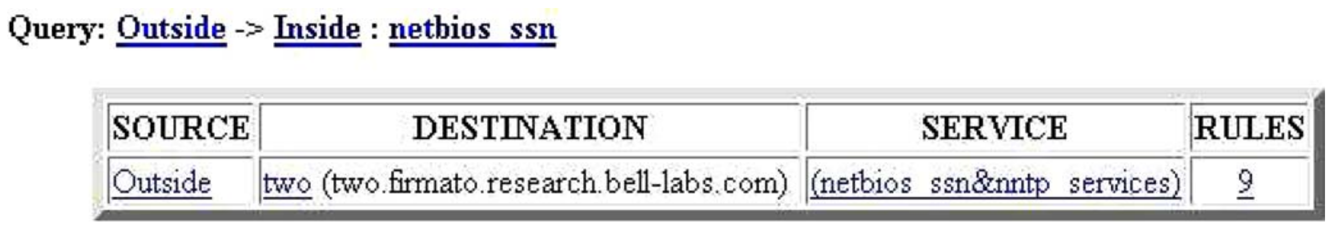


Fig. 10 A diagram of the firewall's network connectivity, derived from the network topology description file


Query: Outside -> Inside : netbios_ssn

SOURCE	DESTINATION	SERVICE	RULES
Outside	two (two.firmato.research.bell-labs.com)	(netbios_ssn&nntp_services)	9

Fig. 11 An excerpt from the "Analysis by service: Incoming" report, showing the results of the `netbios` query

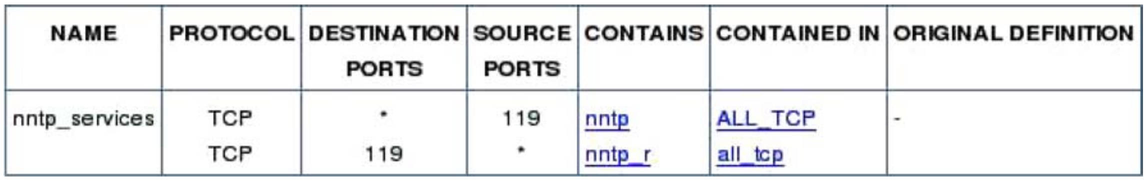

NAME	PROTOCOL	DESTINATION PORTS	SOURCE PORTS	CONTAINS	CONTAINED IN	ORIGINAL DEFINITION
nntp_services	TCP	*	119	nntp	ALL_TCP	-
	TCP	119	*	nntp_r	all_tcp	

Fig. 12 An excerpt from the Services report, with the `nntp_services` service highlighted

Another passive firewall analysis system is [9], which is based on constraint logic programming. Users express the network topology and firewall configuration as logic statements, and queries are resolved using a generic inference engine. At present, the system is in the "proof-of-concept" stage, only supporting Cisco router access lists and with a minimal text-based user interface.

8.1 Active scanners

A number of vulnerability testing tools are available in the market today. Some are commercial, from vendors such as ISS [23], others are free such as Satan [10, 11] or `nmap` [13]. 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 as follows:

- 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.
- Vulnerability testing tools can only catch one type of firewall configuration error, allowing unauthorized packets through. They do not catch the second 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.
- 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.

- An active vulnerability tool 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 tool spoofs the source IP address on the packets it sends, it will never receive any return packets, and will have no indication whether the spoofed packets reach their destination or not.
- 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.
- An active tester can detect a hole only if a host is using the destination IP address, and that host is powered up and listening on the probed port. If the host is down or not listening, the vulnerability will go undetected.

On the other hand, active scanners provide additional data that passive analysis does not. Such data includes answers to questions such as: Is a host actually listening on a port? Is the host running a vulnerable version of the application? Are the login passwords easy to crack? Is the host reachable with the actual routing scheme? Thus, offline analysis and active testing in fact complement each other. A prudent methodology is to run the offline analysis, and based on its results, target the most exposed hosts for active vulnerability testing.

8.2 Distributed firewalls

Recently there has been a renewed interest in firewall research, focusing the idea of a distributed firewall. The basic idea is to make every host into a firewall that filters traffic to and from itself. This type of firewall is already very popular commercially: personal firewalls for PCs, such as Zone Alarm [37] and BlackICE [24], have proliferated with the success of high-bandwidth, always-on, Internet connections like DSL and Cable. The case for distributed firewalls was argued effectively by Bellovin [4].

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.

We believe 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.

9 Conclusion

We have described the design and implementation of two generations of passive firewall analysis tools: the Fang research prototype, and the Firewall Analyzer (FA) product. We have shown their inner workings and how they evolved from prototype to product over a period of 4 years. The result is a novel, multi-vendor tool that simulates and analyzes the policy enforced by a firewall. The FA 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.

Acknowledgements Many people have contributed to the evolution of Fang and FA, whether in ideas, algorithms, or code. We gratefully acknowledge the contributions of Yair Bartal, Yuval Baron, Sudip Bhattachariya, Steve Branigan, Hal Burch, Diane Burley-McGlue, Bill Cheswick, Tom Limoncelli, Ryan Martin, Kobbi Nissim, Yossi Rubinroit, Yaniv Shapira, Karl Siil, and Bruce Wilner.

References

1. Algorithmic Security's Firewall Analyzer (2004) http://www.algosec.com/Products/
2. Bartal, Y., Mayer, A., Nissim, K., Wool, A.: Firmato: A novel firewall management toolkit. ACM Trans. Comput. Syst. **22**(4), 381–420 (2004)
3. Bellovin, S.M.: Security problems in the TCP/IP protocol suite. Comput. Commun. Rev. **19**(2), 32–48 (1989)
4. Bellovin, S.M.: Distributed firewalls. login: The Magazine of USENIX & SAGE, pp. 39–47 (1999)
5. Chapman, D.W., Fox, A.: Cisco Secure PIX Firewalls. Cisco Press, Indiana (2001)
6. Cheswick, W.R., Bellovin, S.M., Rubin, A.: Firewalls and Internet Security: Repelling the Wily Hacker, 2nd edn. Addison-Wesley, Reading, MA (2003)
7. Dalheimer, M.K.: Programming With Qt. O'Reilly & Associates, Inc., California (1999)
8. De Berg, M., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Berlin Heidelberg New York (2000)
9. Eronen, P., Zitting, J.: An expert system for analyzing firewall rules. In: Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001), pp. 100–107. Copenhagen, Denmark (November 2001); Technical Report IMM-TR-2001-14, Technical University of Denmark
10. Farmer, D., Venema, W.: Improving the security of your site by breaking into it. (1993) http://www.fish.com/security/admin-guide-to-cracking.html
11. Freiss, M.: Protecting Networks with SATAN. O'Reilly & Associates, Inc., California (1998)
12. Fulmer, C.: Firewall product overview. (2002) http://www.thegild.com/firewall/
13. Fyodor: NMAP – the network mapper. (2000) http://www.insecure.org/nmap/

14. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.-P.: A technique for drawing directed graphs. IEEE Trans. Softw. Eng. **19**(3), 214–230 (1993)
15. Goldsmith, D., Schiffman, M.: 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/
16. Graphviz – open source graph drawing software (2001) version 1.7, http://www.research.att.com/sw/tools/graphviz/
17. Guttman, J.D.: Filtering postures: Local enforcement for global policies. In: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA. IEEE, Piscataway, NJ (1997)
18. Guttman, J.D.: Security goals: Packet trajectories and strand spaces. In: Foundations of Security Analysis and Design (FOSAD). Lecture Notes in Computer Science, vol. 2171. Springer, Berlin Heidelberg New York (2001)
19. Hazelhurst, S., Attar, A., Sinnappan, R.: Algorithms for improving the dependability of firewall and filter rule lists. In: Workshop on Dependability of IP Applications, Platforms and Networks, pp. 576–585. IEEE Computer Society Press, Los Alamitos, CA, (2000). Published in Proceedings of International Conference on Dependable Systems and Networks
20. Held, G., Hundley, K.: Cisco Access Lists. McGraw-Hill, New York (1999)
21. Huitema, C.: Routing in the Internet. Prentice-Hall, Englewood Cliffs, NJ (1995)
22. ICSA Labs Certified firewall products. (2003) http://www.icsalabs.com/html/communities/firewalls/ certification/rxvendors/index.shtml
23. Internet Security Systems Internet Scanner (2000) http://documents.iss.net/literature/InternetScanner/is_ps.pdf
24. Internet Security Systems BlackICE Defender (2003) http://blackice.iss.net/
25. Lucent VPN firewall brick (2002) http://www.lucent.com/security
26. Mayer, A., Wool, A., Ziskind, E.: Fang: A firewall analysis engine. In: Proceedings of IEEE Symposium on Security and Privacy, pp. 177–187. Oakland, CA. IEEE, Piscataway, NJ (2000)
27. Qt: online reference documentation, version 2.0.1. (1999) Troll Tech http://www.troll.no/qt/
28. Ranum, M.: On the topic of firewall testing (1995) http://www.ranum.com/pubs/fwtest/
29. Rubin, A., Geer, D., Ranum, M.: Web Security Sourcebook. Wiley Computer Publishing, New York (1997)
30. Russell, R.: (2000) Linux IPCHAINS-HOWTO, v1.0.8, http://www.tldp.org/HOWTO/IPCHAINS-HOWTO.html
31. Stevens, W.R.: TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley, Reading, MA (1994)
32. Welch-Abernathy, D.D.: Essential Checkpoint Firewall-1: An Installation, Configuration, and Troubleshooting Guide. Addison-Wesley, Reading, MA (2002)
33. Wool, A.: Architecting the Lumeta firewall analyzer. In: Proceedings of 10th USENIX Security Symposium, pp. 85–97. Washington, DC (2001). USENIX
34. Wool, A.: A quantitative study of firewall configuration errors. IEEE Computer **37**(6), 62–67 (2004)
35. Wool, A.: The use and usability of direction-based filtering in firewalls. Comput. Security **23**(6), 459–468 (2004)
36. Xu, W., O'Neal, S., Schoonover, J., Moser, S., Lamar, F., Grasboeck, G.: (2000) fwrules50, Available from http://www.phoneboy.com/fw1/
37. ZoneAlarm (2003) 3.7.143. Zone Labs, http://www.zonelabs.om/
38. Zwicky, E.D., Cooper, S., Chapman, D.B.: Building Internet Firewalls, 2nd edn. O'Reilly & Associates, Inc., California (2000)