Packet Filtering and Stateful Firewalls

Avishai Wool, Ph.D.,
School of Electrical Engineering, Tel Aviv University

Outline

Keywords:

Packet filtering, Stateful filtering, Firewalls, Packet Matching, Packet Classification, Firewall Configuration, Firewall Policy, Firewall Useability

Abstract

The firewall is a crucial component in the defense mechanisms of every network that is connected to the Internet. In this chapter we shall first survey what firewalls are, what it means for them to be "stateful", and why this is important to the organization's security. Then we shall go into some more advanced material: How firewalls work "under the hood", how poorly configured typical firewalls are, why spoofing attacks are so difficult to combat at the firewall, and finally we discuss some recent research directions concerning firewall management.

1. Introduction

The Internet is like a system of roads that transport packets of data from one computer network to another, using the TCP (Transmission Control Protocol)/Internet Protocol (IP) protocol suite. However, not all IP traffic is welcome everywhere. Most organizations need to control the traffic that crosses into and out of their networks: to prevent attacks against their computer systems, to prevent attacks originating from their network against other organizations, to prevent attacks originating from inside of the organization against other parts of the organization [insider threat - i.e. an employee in Finance trying to get into the Human Resources Department network], and to conform with various policy choices. The firewall is the primary control point for these tasks. It is typically the first filtering device that sees IP packets that attempt to enter an organization's network from the outside, and it is typically the last device to see an exiting packet. It acts like the security guard at the entrance to a building: It is the firewall's job, using the **policy** it is configured to use, to make a filtering decision on every packet that crosses it: either to let it **pass**, or to **drop** it.

In order for the firewall to perform its job, several conflicting issues need to be addressed:

1. The firewall must be able to **observe all the traffic**: Obviously, a firewall cannot control traffic that bypasses it.
2. The firewall must be able to **differentiate** between various types of traffic, and apply the appropriate filtering decisions.
3. The firewall must **work fast** enough so that it does not become a bottleneck.

Observe all traffic

To address the first issue, a firewall needs to be placed at a **choke point** on the network, through which all controlled traffic has to travel. In a typical setup the firewall is connected to the communication line going to the organization's Internet Service Provider (ISP). If the organization is large, with multiple Internet connections, there should be a firewall attached to each connection. The only way to ensure that unauthorized traffic cannot bypass the firewall is by an administrative policy, that forbids adding Internet connections without authorization.

Differentiate

The firewall must be able to distinguish between packets that conform to the corporate policy (and should be let through), and the rest (which should be dropped). For this, the policy needs to be encoded in a language that the firewall's software can understand. This encoding is usually done via a sequence of **rules** that describe various cases. The sequence of rules is usually called a **rule-base**. A rule-base has to have a policy decision for every possible type of packet the firewall will ever encounter: this is usually done by listing all the special cases, followed by a catch-all default rule. The default rule should implement a safe stance of "what is not explicitly allowed should be dropped": almost all firewalls have such a default "out of the box", but some exceptions do exits – in which case the administrator needs to add a safe default rule manually.

Each rule makes reference to values that distinguish one packet from another. The common **match fields** in firewall rules refer to a packet's source and destination IP addresses, protocol, and source and destination port numbers. These are fields that appear in a packet's IP header, or in the TCP/UDP headers as appropriate. However, some firewalls look deeper into the packet, and are able to interpret the semantics of specific protocols like SMTP (email), HTTP (web browsing) etc. Most firewalls are also capable of filtering based on a packet's **direction**: which network interface card the packet is crossing, and whether the packet is crossing the interface from the network into the firewall ("inbound") or vice versa ("outbound").

Most firewalls implement **first-match semantics**: This means that the firewall tries to match a packet against all the rules in the rule-base, in sequence. When if it finds the first rule that that matches all the match fields in the packet, it stops, and makes the filtering decision according to that rule. Therefore, more specific rules need to be listed higher up in the rule-base.

Work fast

Packet matching in firewalls involves matching on many fields from the TCP, UDP, and IP packet header . With available bandwidth increasing rapidly, very efficient matching algorithms need to be deployed in modern firewalls, to ensure that the firewall does not become a bottleneck. There is an inherent tradeoff between speed and differentiation capabilities: a firewall that looks deeper into each packet spends more time on it. The standard solution, that practically all firewalls adopt, is to start with the basic five fields (protocol number, source and destination IP addresses, and source and destination port numbers). The advantage of these five fields is that they are in fixed positions in the packet headers, their contents are standardized and simple, and they describe a packet fairly well: the IP addresses show where the packet is going and where it's coming from, and the combination of protocol and port numbers show what the packet is trying to achieve. The combination of protocol and port numbers is called a **service**: From a firewall's perspective, the HTTP service is defined by protocol=TCP, destination port=80, and any source port. Beyond these five basic fields, there is a great variability between vendors and configuration choices, since looking deeper into the packets is service-dependent and often computationally expensive. Common solutions offload the service dependent filtering to additional dedicated machines that are situated behind (or next to) the firewall: e.g., an e-mail filter that scans for viruses, an http filter that strips Java applets and mal-formed URLs, etc.

2. Basic Packet Filtering

What is it?

Most first generation firewalls used **basic packet filtering**. Basic packet filtering means that

1. The firewall keeps no **state**. The filtering decision is made separately for every packet, and does not take into account any earlier decisions made on related packets.

2. The filtering decision is based only of the five basic fields: Source and Destination IP addresses, Protocol, and Source and Destination Port numbers (for protocols that have port numbers).

The typical actions that a basic packet filter can take are:

1. Pass: let the packet through

2. Drop: do not forward the packet. No indication is sent back to the sender.

3. Reject: Same as Drop, except that a special ICMP packet is sent back to the sender informing it that the packet was filtered.

In addition, it is usually possible to **log** the action: write an entry in a log file with the details of the packet, the decision, the rule that caused the decision, etc. Logging is very useful for debugging, and for identifying concentrated attacks. However, logging should be used with care, as it can be very expensive: disk-write rates are orders of magnitude slower than network transmission rates. When the network is subjected to a Denial-of-Service (DoS) attack, and the firewall is bombarded with packets that are being dropped, the firewall itself may crash under the load if it attempts to log too many of the packets.

Basic packet filtering can still be found in free firewalls like `iptables`/`netfilter` [Net] and `IPFilter` [IPF] in their standard configuration, and also in routers that implement access-lists. However, such "pure" packet filters are becoming rare: `netfilter` can be used with the `conntrack` module which is stateful, and Cisco provides add-on modules that make their access control lists stateful as well.

Limitations of basic packet filtering

Most of the useful Internet services are bi-directional: data is sent from one computer to another, and responses (**return traffic**) are sent back. This is always the case for TCP-based services – at the very least, every TCP packet must be acknowledged by its recipient. For the sake of simplicity, in this section we shall focus on TCP traffic. Obviously, for a bi-directional service to work correctly, both traffic directions have to be allowed to cross the firewall.

The combination of packets going in both directions is called a TCP **flow**. There is always one endpoint (computer) that initiates a flow (sends the first packet) – this endpoint is called the **client,** and the other is called the **server**. A TCP flow is characterized by four attributes: the IP addresses of the two endpoints, and the two ports being used. However, the specification of a TCP service typically only identifies the destination port. For example, a telnet connection is almost always on TCP port 23. So a typical telnet *server* listens on port 23. However, a telnet *client* may use any port number on its endpoint – and typically, this port number is chosen dynamically at run time, and is essentially unpredictable. If the firewall is a basic packet filter, the un-predictability of the client's port number makes it almost impossible to let the flow cross the firewall without introducing risky side effects.

To illustrate the problem, let us continue with the telnet example. Suppose we want to allow a telnet flow between a client at IP address 1.1.1.1 and a server at 2.2.2.2. Assume further that the client is inside the organization's network, and the server is outside. Client-to-server packets look like this:

Source IP=1.1.1.1, Destination IP=2.2.2.2, Source Port=NNN, Destination Port=23

where NNN is an arbitrary number selected by the client. On the other hand, return traffic, from the server to the client, swaps IP addresses and port numbers, and looks like this:

Source IP=2.2.2.2, Destination IP=1.1.1.1, Source Port=23, Destination Port=NNN

(for the same port number NNN). Now consider this flow from the firewall's point of view. The person writing the rules to allow this flow knows the endpoints' IP addresses, and knows that the telnet port is 23. But she does not know the client's source port (NNN), which is selected dynamically at run time. So to allow both directions of traffic through the firewall, she needs to write two basic filtering rules:

1. Permit TCP packets from 1.1.1.1, to 2.2.2.2, with Any source port and destination port 23.

2. Permit TCP packets from 2.2.2.2, to 1.1.1.1, with source port 23 and Any destination port.

Unfortunately, rule 2, which matches packets based on their source port, is extremely risky. Remember that the source port is under the packet sender's control. An attacker on machine 2.2.2.2 can craft packets with source port 23, destination IP 1.1.1.1, and a destination port of his choice. Packets crafted this way will cross the firewall because they match the second telnet rule – and then access a *non-telnet* port! Thus, we see that basic filtering rules that are intended to allow outbound telnet, also allow inbound TCP on *any* destination port as a side effect.

This type of risk is unavoidable for a basic packet filter. Since the firewall does not keep state, it doesn't "remember" whether a telnet flow is already established, and what source port number the client selected. Thus, the firewall has to rely on source-port filtering – which, as we just saw, is unreliable and risky.

Therefore, essentially all modern firewalls go beyond basic packet filtering, and are **stateful**. Stateless filtering can still be found, e.g., in routers, but these devices should not be perceived as "real" firewalls and should only be used in relatively safe environments, such as within an organization's internal network, to separate departments from each other.

3. Stateful Packet Filtering

What is it?

To address the limitations of basic packet filtering, practically all the firewalls on the market nowadays are stateful. This means that the firewall keeps track of established flows, and all the packets that belong to an existing flow, in both directions, are allowed to cross the firewall.

To do this, the firewall keeps an entry, in a **cache,** for each open flow. When the first packet of a new flow is seen by the firewall (this is the so-called SYN packet in a TCP flow), the firewall matches it against the rule-base. If there is a rule that allows the packet across, the firewall inserts a new entry into the cache. This entry includes both endpoints' IP addresses, and, importantly, both port numbers. The port number information was not fully known when the administrator wrote the rules – however, when the flow is being set up, both port numbers are known, since they are listed in the packet's TCP header.

When a subsequent packet reaches the firewall, the firewall checks whether an entry for the flow it belongs to already exists in the cache. If the flow is listed in the cache – the packet is allowed through immediately. If no such flow exists, then the packet is matched against the rule-base, and is handled accordingly.

When the flow is torn down (triggered by a so-called FIN packet), the firewall removes the cache entry, thereby blocking the flow. Typically the firewall also has a timeout value: if a flow becomes inactive for too long, the firewall evicts the entry from the cache and blocks the flow.

Therefore, we see that the first packet of a flow effectively opens a hole in the firewall, and the cache mechanism allows the return traffic to go through this hole.

Advantages of stateful filtering

The main advantage of a stateful firewall is that it is inherently more secure. The firewall administrator no longer needs to write broad (and insecure) source-filtering rules to allow the return traffic: She only needs to list the attributes of the flow's first packet in the rule-base, and the cache mechanism takes care of the rest. The hole that is created is only large enough for this particular flow, and does not introduce side effects. Furthermore, the hole is temporary: Once the flow is torn down, the hole is closed.

An additional benefit is that the rule-base is shorter: A single rule is enough to describe a flow, whereas a basic packet filter needs 2 rules for the same flow. Writing and managing fewer rules mean less work for the administrator and fewer chances to make mistakes.

Finally, this mechanism provides a significant performance boost. With appropriate data structures, a cache lookup can be made very efficiently, e.g., by using a hash-table or search tree. Matching a packet against the rule-base is usually a much slower operation (although, as we shall see in the Matching Algorithms section, it does not have to be slow).

Limitations of stateful filtering

An established flow is allowed through the firewall only as long as it has an entry in the cache. Therefore, if the cache entry is removed while the flow is still active – all remaining traffic will be dropped, and the connection will break. This can happen in two common situations:

- Cache table overflow: the cache table grows dynamically, as more connections are activated concurrently. A firewall protecting a busy network may need to keep hundreds of thousands of established flows in its cache. If the firewall does not have sufficient memory and the cache overflows its capacity, then the firewall will start evicting cache entries, causing connections to drop. Note that causing a cache table overflow can be a type of "Denial of Service (DoS)" attack: If an attacker floods the firewall with connections, the firewall would drop legitimate network connections.]

- Time-out too short: As we noted above, the cache entry is removed when a flow is intentionally torn down – and also if it is inactive for more than a certain time. If the timeout period is set too short, then the cache entry may be evicted simply because the flow was quiet for too long. Firewalls typically have sensible default values for their timeout parameters, but occasionally a connection is dropped for this reason: a typical example is a telnet or ssh connection, which sends no traffic if the person at the terminal stops typing. For this reason, well designed ssh clients use the protocol's "NOOP" packets to keep the connection alive even if the user stops typing. This is a sensible feature for any protocol that needs to cross a firewall.

Dynamic port selection

As we noted before, a firewall identifies a service based on the port numbers it uses. Implicit in this mechanism is the assumption that the port numbers, at least on the server side, are *fixed* and known in advance to the firewall administrator. Unfortunately, this assumption is not universally true.

There exist services, which we shall call **multi-channel services**, which dynamically select both the server and client ports numbers. Most streaming audio services work this way, and so does the very common FTP (file-transfer protocol). A typical multi-channel service has a control channel that uses a fixed port number (e.g., TCP port 21 for FTP). Within the application-layer protocol running over the control channel, the communicating hosts dynamically agree on a port number for a data-channel, over which they transfer the bulk of their data.

To support such services, the firewall is forced to parse the application-layer protocols, keep additional state for such connections, and dynamically open the ports that were negotiated. This usually means that the firewall vendor needs to add special program code to handle each multi-channel service it wishes to support: it is not something that the firewall administrator can add on her own by adding rules to the rule-base.

4. Matching Algorithms

This section discusses the internal algorithmics of firewalls, and assumes some mathematical background. Readers who are primarily interested in using and configuring firewalls may skip this section.

As we saw already, most modern firewalls are stateful. This means that after the first packet in a network flow is allowed to cross the firewall, all subsequent packets belonging to that flow, and especially the return traffic, is also allowed through the firewall. This statefulness is commonly implemented by two separate search mechanisms: (i) a slow algorithm that implements the ``first match'' semantics and compares a packet to all the rules, and (ii) a fast state lookup mechanism that checks whether a packet belongs to an existing open flow.

In many firewalls, the slow algorithm is a naive linear search of the rule-base, while the state lookup mechanism uses a hash-table or a search-tree. This is the case for the open-source firewalls `pf` [Pf] and `iptables` / `netfilter` [Net]. Note that this two-part design works best on long TCP connections, for which the fast state lookup mechanism handles most of the packets. However, connectionless UDP and ICMP traffic, and short TCP flows (like those produced by HTTP V1.0), only activate the ``slow'' algorithm, making it a significant bottleneck. The rest of this section explores how this “slow” packet matching algorithm.

Here is a mathematical way of looking at the firewall packet-matching problem: Given a packet, find the first rule that matches it on d fields from its header. Every rule consists of set of ranges $[l_i,r_i]$ for $i=1,\ldots,d$, where each range corresponds to a field in a packet header. The field values are in $0 <= l_i,r_i <= U_i$, where $U_i = 2^{32}-1$ for 32-bit IP addresses, $U_i = 65535$ for 16-bit port numbers, and $U_i = 255$ for 8-bit ICMP message type or code. Note that we are assuming that the field values are all numeric, which is reasonable for header fields but less so for data within the packet payload. The number of fields to be matched (d) in this representation is typically $d=4$ (for TCP, UDP, and ICMP protocols) or $d=2$ if the matching is limited to IP addresses when the protocol does not have (or does not expose) any port numbers (e.g., when filtering encrypted IPSec traffic)

Viewed this way, we can give the packet-matching problem a geometric interpretation. Each packet is a point in d-dimensional space: each header field corresponds to a dimension. Each rule is now a d-dimensional “box”, and we have N such boxes (rules), that may overlap each other, with a higher priority given to boxes (rules) which are listed first. Under this interpretation, the matching problem is now: find the highest priority box that contains a given d-dimensional point.

It is difficult for people to visualize 4-dimensional space. But in 2 dimensions the analogy is quite natural. Think of a plane, with the X-axis corresponding to the source IP address, and the Y-axis corresponding to the destination IP address. In this view, a rule is a rectangle: all points whose X value is in some range and whose Y value is in some other range. If one of the fields is a “don’t-care”, we just end up with a very wide (or very tall) rectangle. A rule-base with N rules now becomes a collection of overlapping rectangles. When 2 rectangles overlap, one hides parts of the other. Now think of all the “PASS” rules as having a white color, and all the “DROP” rules as having a black color. Viewed from above, the full set of N rectangles subdivides the plane into a patchwork of rectangles, and rectangle fragments (that are just smaller rectangles) – some white and some black. Given a particular point, with some X/Y coordinates (=source/destination IP addresses), which rectangle does it belong to, and what color is that rectangle? This is called a **point-location** problem.

Besides being elegant, this geometric view has great algorithmic importance. Computer scientists working in the field of computational geometry have studied point-location problems for several decades, and have come up with some very efficient algorithms. Some algorithms are limited to 2 dimensions, but there are algorithms that deal with arbitrary d-dimensional problems as well.

Usually, these algorithms have two parts: The first part preprocesses the colored patchwork plane into a data structure that supports fast searches. This is done once, when the rule-base is created. The size of the search data structure often grows rather large. The second part is a very fast lookup through the data structure. The search is done on every point lookup query (i.e., for every packet to be matched). Usually we do not care so much about how long the preprocessing takes, as long as it is “within reason”. What we do care about are (1) getting blazingly-fast lookups, because the lookup rate has to be as fast as the rate of arriving packets; and (2) the size of the search data structure: if it is too large to fit in main memory, the filtering rate will drop rapidly because of disk accesses or page faults.

One successful application of geometric algorithms to the firewall packet-matching problem can be found in [RW04]. As we mentioned above, stateful firewalls have a slow algorithm, that compares a packet to each rule in sequence, and then a fast state-lookup algorithm for open flows. In [RW04] the authors show that ``slow'' algorithms do not need to be slow. They demonstrated an algorithm (called GEM, for Geometric Efficient Matching) that does lookups in O(log N) time, and has an O(N^4) search data structure size. More importantly, though, is that they implemented the algorithm as an extension to the Linux `iptables` firewall, and tested it. They showed that, e.g., on 5000-rule rule-bases, the data structure only reached some 13MB, which easily fits into the main memory of modern computers. They report that the GEM algorithm sustained a matching rate of over 30,000 packets-per-second, with 10,000 rules, without losing packets, on a standard 2.4GHz PC running RedHat Linux 9.0. This is at least 10 times faster than the rate achieved by the regular `iptables.` The interested reader is referred to [RW04] and the references therein for further details.

5. Common Configuration Errors

The protection that firewalls provide is only as good as the policy they are configured to implement. Once a company acquires a firewall, a systems administrator must configure and manage it according to a security policy that meets the company’s needs. Configuration is a crucial task, probably the most important factor in the security a firewall provides [RGR97].

Network security experts generally consider corporate firewalls to be poorly configured, as witnessed in professionally oriented mailing lists such as Firewall Wizards [FWW04]. This assessment is indirectly affirmed by the success of recent worms and viruses like Blaster [Bla03] and Sapphire [Sap04], which a well-configured firewall could easily have blocked.

In this section we focus on rule sets for Check Point's FireWall-1 product (www.checkpoint.com), and specifically, on 12 possible misconfigurations that would allow access beyond a typical corporation's network security policy.

Rule-set complexity

Firewall administrators are intuitively able to classify a rule set as "complicated" or "simple". To quantify this intuition into a concrete measure of complexity, [Wool04a] suggests the following measure of rule-set complexity:

$$RC = \text{Rules} + \text{Objects} + \text{Interfaces}(\text{Interfaces}-1)/2,$$

where RC denotes rule complexity, Rules denotes the raw number of rules in the rule set, Objects denotes the number of network objects, and Interfaces denotes the number of interfaces on the firewall. The rationale for this measure is as follows: counting the number of rules is obvious; Adding the number of objects is useful because a Check Point rule can refer to a great number of network objects (subnets and groups of these) which significantly increase its complexity; Finally, if the firewall has *i* interfaces then the number of different interface-to-interface paths through the firewall is $i(i-1)/2$, and the number of paths increases the complexity of the rule-base.

Which Configuration Errors to count?

To quantify the quality of a firewall configuration, we must define what constitutes a configuration error. In general, the definition is subjective, since an acceptable policy for one corporation could be completely unacceptable for another. [Wool04a] counted as errors only those configurations that represented violations of well-established industry practices and guidelines [ZCC00,CBR03]. Note that these are not the only dangerous configuration mistakes an administrator may make – many others are possible. The mis-configurations listed here should only serve as a *baseline* of things to avoid, rather than an exhaustive list. The following 12 items counted as configuration errors:

1. *No stealth rule*. To protect the firewall itself from unauthorized access, it is common to have a "stealth" rule of the form: "From anywhere, to the firewall, with any service, drop." The absence of such a rule to hide the firewall counted as a configuration error.

2-4. *Check Point implicit rules*. Besides the regular user-written rules, the Check Point FireWall-1 GUI has several checkboxes that produce implicit rules. These rules control both the Internet's domain name service (DNS), separately over TCP and UDP, and the Internet control message protocol (ICMP). However, the implicit rules are very broad, basically allowing the service at hand from anywhere to anywhere. Since DNS is one of the most attacked services [SANS03] writing narrow, explicit rules for it is more secure. Likewise, with any-to-any ICMP, attackers can scan the internal net and propagate worms like Nachi/Welchia. Each of the three possible implicit rules—DNS-TCP, DNS-UDP, and ICMP—counted as one error.

5. *Insecure firewall management*. Access to the firewall over insecure, unencrypted, and poorly authenticated protocols—such as telnet, ftp, or x11—counted as one error.

6. *Too many management machines*. Firewalls should be managed from a small number of machines. Allowing management sessions from more than five machines was counted as a configuration error. While this threshold is somewhat subjective, most experts agree that it is reasonable.

7. *External management machines*. An error was counted if machines outside the network's perimeter could manage the firewall. The preferred way for administrators to manage the firewall from home is from the "inside" through a virtual private network.

8. *NetBIOS service*. NetBIOS is a set of services that Microsoft Windows operating systems use to support network functions such as file and printer sharing. These frequently attacked services are very insecure [SANS03]. Allowing any NetBIOS service to cross the firewall in any direction counted as an error.

9. *Portmapper/Remote Procedure Call service*. The portmapper daemon assigns TCP ports to implement RPC services, a Unix mechanism that has a long history of being insecure. Among other services, RPCs include the Network File System protocol, which potentially exposes all the organization's file system. Allowing traffic to the portmapper (TCP or UDP on port 111) counted as an error.

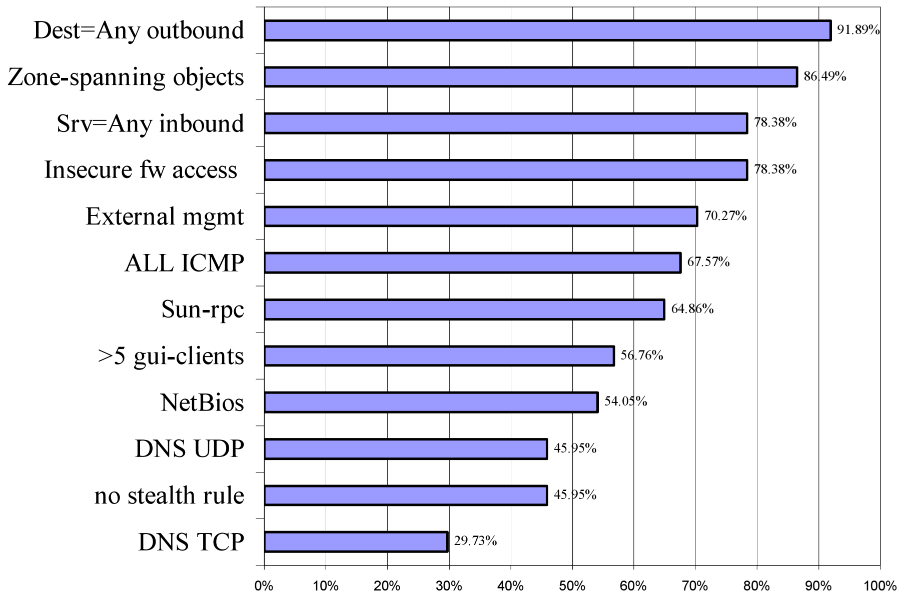
10. *Zone-spanning objects*. A Check Point network object is a named definition of a set of IP addresses. Zone-spanning objects include addresses that reside on more than one "side" of the firewall—for example, some IP addresses internal to the firewall and others external. Note that a firewall with more than two interfaces has more than just an inside and an outside – each interface defines another "side". Zone spanning objects cause many unintended consequences when used in firewall rules. For example, when administrators write a rule, they usually assume that the object is either internal or external, and this assumption affects how they write the rule. Zone-spanning objects break this dichotomy—with disastrous results. Any use of zone-spanning objects in the rule set counted as an error.

11. *"Any" service on inbound rules*. Allowing "Any" service to enter the network is a gross mistake, since "Any" includes numerous high-risk services, including NetBIOS and RPC services. Allowing such access was counted as an error.

12. *"Any" destination on outbound rules*. Because internal users typically have unrestricted access to the Internet, outbound rules commonly allow a destination of "Any." Unfortunately, firewalls commonly have more than two network interfaces. The typical usage of a third interface is to attach a DeMilitarized Zone (DMZ): a subnet dedicated to the corporation's externally-visible servers. In such cases, free Internet access also gives internal users free access to the servers in the DMZ Worse, it often allows the DMZ servers free access to the internal network, because the predefined "Any" network object is inherently zone-spanning (See the Direction-based Filtering section). Therefore, allowing such access counted as an error.

Item 12 is probably the most subjective error counted. It is possible to safely use a destination of "Any" by carefully adding other rules that restrict the unwanted access. Nevertheless, finding "destination = Any" outbound rules in a firewall audit should, at least, raise a warning flag.

Results and Analysis



• Figure 1 Distribution of configuration errors.

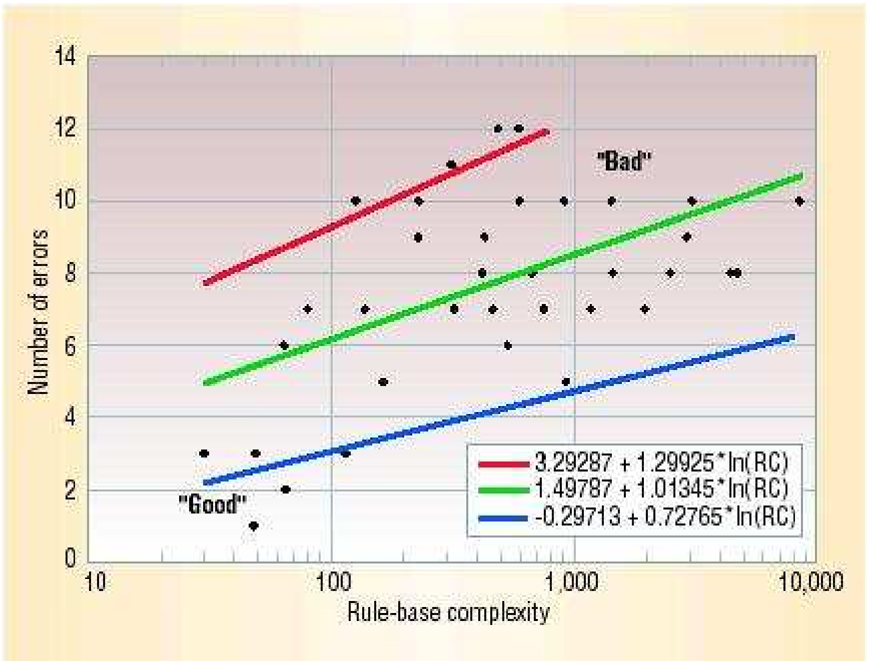
Figure 1 shows the raw distribution of configuration errors reported in [Wool04a], based on a survey of real corporate firewall configurations. The results can only be characterized as dismal. Most of the errors appeared in most of the firewalls studied: in fact, nine of the 12 errors appeared in more than half the firewalls.

Even if we discount the two most frequent errors (items 10 and 12), which may be somewhat controversial, the results show that almost 80 percent of firewalls allow both the "Any" service on inbound rules (item 11) and insecure access to the firewalls (item 5). These are gross mistakes by any account. In fact, only one of the firewalls exhibited only a single misconfiguration. All the other firewalls could have been easily penetrated by both unsophisticated attackers and mindless automatic worms.

Complexity matters: Small is beautiful

The study in [Wool04a] examined several possible factors that influence the distribution of configuration errors. The study found that the firewall's operating system was mostly irrelevant, and that later firewall software versions had slightly fewer mistakes. However, the most significant factor was shown to be the firewall rule-set's complexity, as quantified by the RC measure.

Figure 2 shows a scatter plot of the number of errors versus RC. The RC measure showed a wide range in complexity values. The average RC was 1,121, the lowest value was 30, and the highest was an astonishing 8,521. While the plot is fairly sparse, the empty lower-right quadrant indicates that there are no good high-complexity rule sets. The only reasonably well-configured firewalls—three errors or less—are very simple, with RC values under 100. However, a small and simple rule set is no guarantee of a good configuration. The figure shows that a small rule set can be configured quite badly: there are two configurations with RC values under 100 but with six or more errors.



• Figure 2. Number of errors as a function of rule-set complexity. The RC scale is logarithmic. The middle green line represents the least-squares fit, and the red and blue lines represent one standard deviation above and below the least-squares fit line.

In fact, the RC measure yields a crude but fairly accurate prediction of the number of configuration errors: A linear regression shows that a rule set of complexity RC is predicted to have about ln(RC) + 1.5 errors – this is the formula for the central green line in Figure 2 .

The conclusion to draw here is obvious: Limiting a firewall's rule-set complexity as defined by RC is safer. Instead of connecting yet another subnet to the main firewall and adding more rules and more objects, it's preferable to install a new, dedicated firewall to protect only that new subnet. Complex firewall rule sets are apparently too complex for administrators to manage effectively.

6. Direction-based filtering

Background

Beyond the standard header fields, most firewalls are also capable of filtering based on a packet's **direction**: which network interface card the packet is crossing, and whether the packet is crossing the interface from the network into the firewall or vice versa. We call these last capabilities **direction-based filtering**.

Taking a packet's direction into account in filtering rules is extremely useful: It lets the firewall administrator protect against source address spoofing, write effective egress filtering rules, and avoid unpleasant side effects when referring to subnets that span the firewall.

Unfortunately, the firewall's definition of a packet's direction is different from what users normally assume. If interface `eth0` connects the firewall to the internal network, then, from a user's perspective, "inbound on `eth0`" is actually "Outbound" traffic. This discrepancy makes it very confusing for firewall administrators to use the packet direction correctly, and creates a significant usability problem.

Most firewall vendors (exemplified by Cisco and Lucent) seem to be unaware of the usability issues related to direction-based filtering. These vendors simply expose the raw and confusing direction-based filtering functionality to the firewall administrators. A notable exception is Check Point. In order to avoid the usability problem, Check Point chooses to keep its management interface simple, and hide the direction-based filtering functionality in such a way that most users are essentially unable to use it (indeed, many users do not even know that a Check Point FireWall-1 can perform direction-based filtering).

As we saw earlier in this chapter, many firewalls are enforcing poorly written rule-sets, and in particular, direction-based filtering is often misconfigured or entirely unused. We suspect that direction-based filtering is under-utilized in great part due to the usability problem associated with the vendor's configuration tools.

Why Use Direction-Based Filtering

Anti-Spoofing

It is well known that the source IP address on an IPv4 packet is not authenticated. Therefore, a packet's source address may be spoofed (forged) by attackers in an attempt to circumvent the firewall's security policy (cf. [ZCC00]). 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 (denial of service) attack against hosts behind the firewall, and is sometimes enough to hijack a `TCP` session [Bel89].

The main defense against such spoofing attacks, at the perimeter firewall, is based on direction-based filtering. Legitimate packets with source IP addresses that belong to `MyNet` should only enter the firewall from its internal interface. Therefore, giving the rule a direction would cause spoofed packets not to match, and to be dropped by a subsequent rule.

Egress filtering

The primary goal of a firewall is to protect the network behind it. However, it is also important to filter egress traffic – traffic that exits the network. Otherwise, the network may become a launching point of attacks, and in particular, DoS attacks, against other organizations on the Internet, or against other zones in the internal network. During such attacks, the attacking host often sends spoofed packets, to conceal its true location. Such an attack can come from a compromised host on the internal network, or from any other network that is routing through the internal network (e.g., a business partner). A well-configured firewall can prevent most DoS attacks originating behind it. This is called egress-filtering.

Again, since the problem at hand is rooted in source address spoofing, the most effective way to combat it in IPv4 is by direction-based filtering. The solution outlined in the previous section essentially works as an egress-filtering rule too: if the *only* packets that are allowed to enter the firewall via its internal interface (on their way Out) are those packets with source address in `MyNet`, then the firewall will drop all the spoofed DoS attack packets. This forces the attacker to use legitimate source addresses, and makes the attack host easier to trace back.

Note that when we talked about Anti-Spoofing, we dealt with protecting the internal network. Egress filtering deals with protecting other networks from being attacked from the internal network. But since both problems are manifestations of source address spoofing, the solution is very similar and utilizes direction-based filtering.

See [SAN00,Edm02] for recommendations on how to write effective egress filtering rules for various types of firewalls. The same recommendations also appear in RFC 2827 [FS00]. Interestingly, the language in the RFC speaks of **ingress** filtering, because it is written for an audience of Internet Service Providers to whom source-spoofed traffic is inbound. This is another illustration of the confusion surrounding packet directions (more on this in the next sections).

Zone-spanning

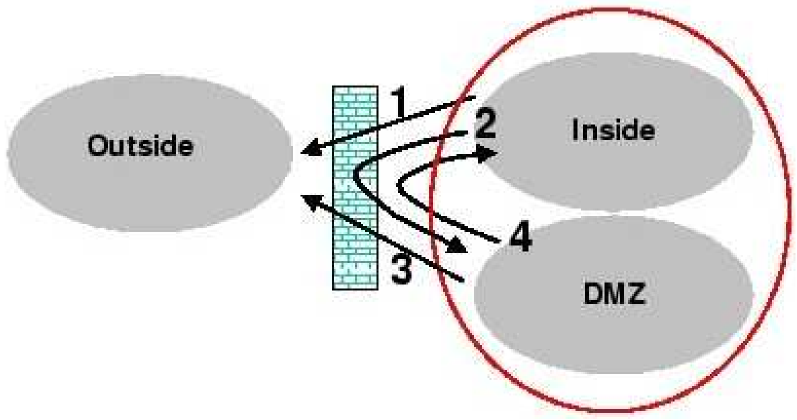


Figure 3: The side effects of zone spanning definitions when the rule is "From `MyNet` to Anywhere with Any service", and `MyNet` includes both the Inside and the DMZ subnets. Arrow 1 indicates the intended traffic, arrows 2 and 4 indicate side-effects. Arrow 3 could be either intended or a side-effect, however current "best practices" suggest not to allow unrestricted Outbound traffic from the DMZ.

Firewall administrators often define objects that span more than one zone ("side") of the firewall. A typical case is to define the `MyNet` group so that it contains both the internal net and the DMZ (see Figure 3). When the `MyNet` group is zone-spanning, the rule "from MyNet to Anywhere with Any Service" has some unintended side-effects, besides the spoofing vulnerability we discussed before. Specifically, the rule now allows all traffic between the DMZ and the internal network, in both directions, because both subnets belong to `MyNet`, and both subnets obviously belong to "Any". This defeats the whole purpose of having a DMZ, since a compromised machine in the DMZ has full access to the internal network. Additionally, the rule also allows unrestricted Outbound traffic to originate from the servers in the DMZ, which is not considered prudent (cf. [Wool02]).

The best way to avoid such side-effects is to completely eradicate zone-spanning definitions, at least in "pass" rules. Unfortunately, zone-spanning definitions are often convenient and intuitive, so avoiding them may be difficult. In particular, the "Any" built-in definition is zone-spanning.

A less draconian measure would be to set the direction on the rule, as we suggested before. For instance, if the rule is applied only to traffic leaving the firewall via its external interface, then traffic between the DMZ and Inside zones will not match the rule and will be dropped (assuming that the firewall has no other matching rules that allow the traffic and a default DENY policy rather than default ALLOW.)

Usability Problems with Direction-Based Filtering

As we saw, direction-based filtering is a highly useful technique to combat various types of spoofing attacks. Unfortunately, though, configuring firewalls to actually use direction-based filtering seems to involve a significant usability problem. This problem is caused by the clash between the user's global, network-centric, stance, and the firewall's local, device-centric, stance. There is ample anecdotal evidence supporting the existence of this difficulty on firewall mailing lists such as [FWW04], and, indirectly, in the amount of documentation devoted to explaining the use of direction-based filtering.

To a firewall administrator, IP addresses are usually split into two disjoint sets: "Inside" invariably means "my protected network" and "Outside" is the rest of the Internet. Traffic directions such a "Inbound" draw their meaning from this dichotomy. When there are other networks involved, such as DMZs, the distinctions blur somewhat, since the DMZ can be viewed as part of either the Inside or the Outside. Still, an Inbound flow of traffic is always understood to be traffic flowing from a less trusted IP address to a more trusted IP address, the latter being within the organization's perimeter.

To the firewall, however, "inbound on interface `eth0`" means "crossing interface `eth0` from the adjacent network into the firewall". This may completely contradict the user's notion of "Inbound": If `eth0` connects the firewall to the internal, protected, network, then "inbound on `eth0`" is actually "Outbound" traffic (see Figure 4). Furthermore, the *same* Outbound traffic is both "inbound on `eth0`" and "outbound on `eth1`", assuming that `eth1` is the external interface. This discrepancy makes it very confusing for firewall administrators to use the packet direction correctly.

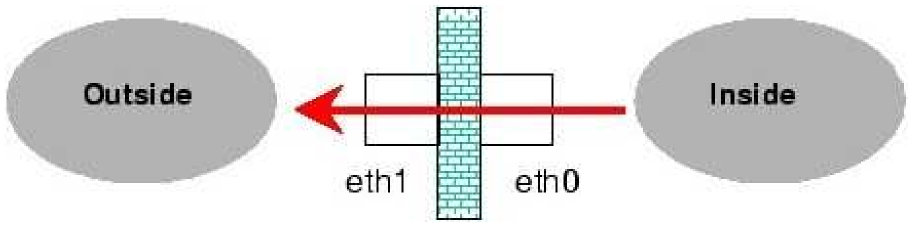


Figure 4: A traffic flow that it Outbound, and is both inbound on `eth0` and outbound on `eth1`.

Note that the difficulty is not merely syntactic or specific to a particular firewall vendor's configuration language. A typical firewall is not aware of the levels of trust given to the networks attached to each of its interfaces. Absent of such global knowledge, the *only* way to specify a direction for traffic flowing through the firewall is device-centric, per interface. This implies that, *unavoidably*, the definitions in the firewall configuration language will often clash with the users' understanding.

[Wool04b] includes a detailed critique of several vendor's approaches to direction-based filtering, and suggests possible improvements.

7. Advanced Firewall Management

Higher-level configuration

As we have seen already, firewall configuration is a difficult task for humans to do well, as evidenced by the poor state of firewall configurations observed in [Wool04a]. A significant part of the problem is that the configuration is done at a very low level of abstraction, dealing with IP addresses (or groups) and services, with no higher level of abstraction available. Since the syntax and semantics of the rules and their ordering depend on the firewall product/vendor, this is akin to the dark ages of software, where programs were written in assembly language and thus the programmer had to know all the idiosyncrasies of the target processor.

These problems get even worse for medium- or large-sized companies, which use more than a single firewall, or use routers for internal packet filtering tasks. These filtering devices divide the company's intranets into multiple **zones**. In this case, the security policy is typically realized by *multiple* rule-bases, located on multiple firewalls that connect the different zones to each other. Thus, the interplay between these rule-bases must be carefully examined so as not to introduce security holes. It is easy to see how rapidly the complexity of designing and managing these rules grows, as intranets get more complex.

The *Firmato* prototype [BMNW04] was built to address these difficulties. *Firmato* is a multi-vendor firewall rule compiler. Its input comprises of a security policy, and a description of the network topology on which the policy is to be enforced. This information is written in *Firmato*'s model description language (MDL). The compiler parses this input, transforms the data through several compilation phases, and produces firewall rules in the supported vendor's configuration languages. *Firmato* has the following distinguishing properties:

- Separate the security policy design from the firewall/router vendor specifics. This allows a security administrator to focus on designing an appropriate policy without worrying about firewall rule complexity, rule ordering, and other low-level configuration issues. It also enables a unified management of network components from different vendors and a much easier transition when a company switches vendors.

- Separate the security policy design from the actual network topology. This enables the administrator to maintain a consistent policy in the face of intranet topology changes. Furthermore, this modularization also allows the administrator to reuse the same policy at multiple corporate sites with different network details, or to allow smaller companies to use default/exemplary policies designed by experts.

- Generate the firewall configuration files automatically from the security policy, simultaneously, for multiple gateways. This reduces the probability of security holes introduced by hard-to-detect errors in firewall-specific configuration files.

- Automatically assign directions to all the rules, through the use of a *routing-aware* algorithm. This implies that anti-spoofing is turned on all the time, and requires no user intervention. Therefore, the *Firmato* approach side-steps the whole usability issue associated with direction-based filtering, by letting the compiler compute the correct direction for each rule.

In [BMNW04] the authors describe their modeling framework and algorithms. They report that a prototype system was built, and supported several commercially available firewall products. This prototype was used to control an operational firewall for several months. It seems that the *Firmato* approach is an important step toward streamlining the process of configuring and managing firewalls, especially in complex, multi-firewall installations.

Firewall Analysis

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:

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

2. 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.

3. 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.

4. 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.

5. 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.

The Fang research prototype [MWZ00] was designed and built to address these difficulties. This work was subsequently commercialized into the Firewall Analyzer [Wool01,Algo04], which currently support several of the leading firewall vendors. The design goals for these systems were as follows:

1. Use an adequate level of abstraction: The administrator should be able to interact with the tool on the level at which the corporate security policy is defined or expressed. In a large network, the tool should allow the administrator to quickly focus on important security aspects of testing.

2. 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 vulnerable combination of IP addresses and port numbers.

3. 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.

4. Be passive: Policy analysis should not involve sending packets, and should complement the capabilities of existing network scanners.

The first generation Fang prototype [MWZ00] interacted with the administrator through a query-and-answer session, using a simple GUI. This GUI let the administrator compose a query through a collection of menus, and displayed the results of the query. Fang demonstrated, for the first time, that static firewall policy analysis was possible, and an important task. However, the most important lesson learned from Fang was that users often *do not know what to query*.

Therefore, in subsequent generations of the tool, human input is limited to providing the firewall configuration, and the analysis is fully automated from that point on. Instead of a manually-written topology file, the Firewall Analyzer accepts the firewall's routing table. Instead of the point-and-click interface, automatically issue all the ``interesting'' queries. Work on this topic is still ongoing, and Improvements are added at a rapid pace. The reader is referred to [Wool01,Algo04] for further details.

Glossary

Action – The decision of a firewall rule, such as Pass, Drop, Reject, Log

Basic packet filter – A firewall or router that matches packets only according to their header fields, without keeping any state.

Direction-based filtering – The firewall’s ability to match packets based on the network interface card they are crossing and the direction (into or out of the firewall)

DMZ – De-Militarized Zone – A subnet that is accessible from outside the perimeter (usually contains public servers), and is semi-trusted.

Stateful firewall – A firewall that matches packets based on their headers and also on the state (or history), such as whether the packet belongs to an open flow.

First-match semantics – Among all the rules that match a given packet, take the action stated in the first rule.

Packet matching – The process of comparing packets to all the rules in the rule-base, and optionally to the state, and finding the action to take.

Policy – The organization's choices of which types of packets are allowed to cross into or out of the network. Encoded as a rule-base.

Rule-base – An ordered list of rules that instruct the firewall what to do. Each rule describes a set of packets, and an action for this set.

Spoofing – Forging the source IP address of a packet in order to bypass the firewall rules or to avoid detection.

Zone – A collection of subnets, which are located behind one of the firewall's network interface cards.

Zone Spanning – A subnet definition that contains IP addresses, which are located in more than one zone. Considered harmful.

Cross References

- Firewall Basics – pp. ###
- Firewall Architectures – pp. ###
- Proxy Firewalls – pp. ###

Bibliography

Books

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

[CBR03] W.R. Cheswick, S.M. Bellovin, and A. Rubin, *Firewalls and Internet Security: Repelling the Wily Hacker*, 2nd edition, Addison Wesley, 2003.

[ZCC00] E.D. Zwicky, S. Cooper, and D.B. Chapman, *Building Internet Firewalls*, 2nd edition, O'Reilly & Assoc., 2000.

[CF01] D. W. Chapman and A. Fox. Cisco Secure PIX Firewalls. Cisco Press, 2001.

[HH99] G. Held and K. Hundley. Cisco Access Lists. McGraw-Hill, 1999.

[Wel02] D. D. Welch-Abernathy. Essential Checkpoint Firewall-1: An Installation, Configuration, and Troubleshooting Guide. Addison-Wesley, 2002.

Web Resources

[Algo04] Algorithmic Security Firewall Analyzer. http://www.algosec.com.

[FWW04] Firewall Wizards. Electronic mailing list, 1997-2004. Archived at `http://honor.icsalabs.com/mailman/listinfo/firewall-wizards`.

[Ful04] Cathy Fulmer's Firewall Product Overview. http://www.thegild.com/firewall/

[Net] Linux netfilter/iptables. http://www.netfilter.org/

[Pf] OpenBSD packetfilter (pf). http://www.benzedrine.cx/pf.html

[IPF] FreeBSD IP Filter. http://coombs.anu.edu.au/~avalon/

[Bla03] CERT Coordination Center, "CERT Advisory CA-2003-20: "W32/Blaster Worm," 11 Aug. 2003; http://www.cert.org/advisories/CA-2003-20.html.

[SANS03] SANS Institute, "The Twenty Most Critical Internet Security Vulnerabilities," v. 4.0, 2003; http://www.sans.org/top20/.

[SANS00] SANS Institute, Global Incident Analysis Center. Egress filtering, v0.2. Electronic publication, `http://www.incidents.org/protect/egress.php`, February 2000.

Articles

[Bel89] S. M. Bellovin. Security problems in the TCP/IP protocol suite. Computer Communications Review, 19(2):32-48, 1989.

[BMNW04] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. ACM T. Computer Systems, November 2004. An earlier version appeared in *Proc. 20th IEEE Symp. on Security and Privacy*, 1999

[Edm02] M. T. Edmead. Egress filtering. TISC Insight, 4(1), January 2002. Electronic newsletter, `http://www.tisc2001.com/newsletters/41.html`.

[FS00] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. Internet Engineering Task Force RFC 2827, May 2000. `http://www.ietf.org/rfc/rfc2827.txt`. An earlier document is RFC 2267, January 1998.

[MWZ00] A. Mayer, A. Wool, and E. Ziskind, "Fang: A Firewall Analysis Engine," *Proc. IEEE Symp. Security and Privacy* (S&P 2000), IEEE Press, 2000, pp. 177-187.

[RW04] D. Rovniagin and A. Wool, "The Geometric Efficient Matching Algorithm for Firewalls", Proc. 23rd Convention of IEEE Israel, September 2004.

[Sap04] D. Moore et al., “The Spread of the Sapphire/Slammer Worm,” 2003; http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html.

[Woo01a] A. Wool, “Architecting the Lumeta Firewall Analyzer,” *Proc. 10th Usenix Security Symp.*, Usenix Assoc., 2001, pp. 85-97.

[Wool02] A. Wool. Combating the perils of port 80 at the firewall. ;login: The Magazine of USENIX & SAGE, 27(4):44-45, August 2002.

[Wool04a] A. Wool, “A Quantitative Study of Firewall Configuration Errors”, IEEE Computer, June 2004.

[Wool04b] A. Wool, The Use and Usability of Direction-Based Filtering in Firewalls. Computers & Security. Elsevier, 2004.