IP Firewalls

an overview of the principles

0. Foreword

WHY: These notes were born out of some discussions and lectures with technical security personnel. The main topics which we discussed were not subtle aspects of configuring some particular firewall of some vendor, but the general aspects and the overall strategy to adopt when creating a firewall configuration. Thus it was more on understanding the principles behind the technology than the technology itself. Needless to say, not always the firewalls do what they should if they would follow these principles, but having clear the principles makes it much easier to understand the exceptions (or the violations).

WHAT: My aim in these notes is to give an overall description of what happens when one introduces a firewall in between a data flow, and to deduce from this some hopefully useful guidelines on how to plan and prepare the configuration of a firewall.

HOW: I will describe a simple firewall which filters some IP traffic using the most relevant features (in my personal opinion) of TCP/IP. The important point for me is not to enter in the complicated details of the protocols, but to give the overall picture of what one should expect to happen. At the end I will give a very preliminary example on how to proceed to create a configuration using a Linux box with iptables.

NOTICE: the terms TCP connection, datagram, segment and packet are used in a somewhat loose way to make the discussion more informal and easier to follow. Most of what I have written comes directly from my notes of the lectures that I gave, where it was more important to deliver the message than to be technically perfectly precise. In a future version I hope to be able to add a few diagrams to make the reading even more intuitive.

Table of Contents

1. TCP/IP Background

Sadly to say (write), this is basic stuff that one can find in many books, but sooner or later I'll add here a very short summary: encapsulation, OSI pile, TCP/IP stack, headers, TCP connections.

In these notes I will introduce a term which generally is not used and someone will say at least quite improper if not completely wrong. I will talk about **IP connections.** Obviously IP is a connectionless protocol which doesn't have any means to insure the establishing and control of a connection, it does not even guarantee that a single packet/datagram is delivered to the intended recipient! So what do I mean by **IP connection**? The concept is very simple: when a sender sends a packet/datagram to a destination, most of the time the receiver will send something back, it can be an answer, an error message or just an acknowledgement. Thus by IP connection what I mean is the fact that very often sending a packet generates a reply. Someone (the firewall) in between the two ends of the *connection*, having seen the first packet passing by, will expect a reply to come back, and will be able to associate the reply, if it comes, to the first packet. Since this is not a precise protocol, the way of associating a reply with a first packet is mainly based on time, that is if within some time from the passing of the first packet there is a reply, this reply is associated with the original packet. If instead a timeout passes, then the *connection* is closed and if a packet goes back this is not considered to be a reply to the first. Only for TCP (and other connection-oriented protocols) it is possible to be more precise about which packets belong to a connection and which do not, but for all other connectionless protocols the definition of IP connection is based only on the time of the optional reply.

2. Types of Firewalls

In these notes a firewall is a network device which should be able to filter the data stream and allow to pass through it only traffic which is deemed to be not harmful. There can be other kinds of firewalls, for example you can have a firewall on an host to directly protect the host from external traffic, and you should do this too on all your hosts which can receive outside traffic (directly or indirectly). Anyway, all firewalls work with the same idea, there is a data stream and the firewall is put in the middle of it to check what passes and to allow only the good data through.

In this notes I will concentrate only on what happens in the firewall, and the setup is the following. There is an host **A** which communicates (sends some packets) to the host **B**, which in case can reply to **A**. In between them there is the firewall **F** which filters all the traffic between **A** and **B.**

Depending on the principle on which it works, I can roughly organize the existing IP firewalls in 4 groups:

1. **Packet Filtering**: each and all packets passing through the firewall are tested against a

single list of rules, and depending on the rule which a packet matches, a different action, usually `accept`, `drop` or `reject`, is taken;
2. **Stateful Inspection**: the first packet of an IP connection is treated as for the previous firewall, but once an `IP connection` is established, all subsequent packets belonging to that connection are automatically accepted, or examined in a different, and faster, way;
3. **Application Layer Gateway**: these firewalls are also capable of understanding the payload of each packet at the application level, and thus to see if the payload contains harmful data;
4. **Proxy**: these firewalls for each application layer protocol of interest, behave like a server to the original client, and a client to the final server, in this way there are no direct connections between the original client and final server, but both are connected directly only to the Proxy in between.

Of course, today many commercial firewalls have features from all four groups.

In the next section I will describe these 4 groups, focusing mainly on the Stateful Inspection Firewalls.

2.1 Packet Filtering Firewalls

I will consider only IP firewall, that is firewalls which filter IP traffic. I will consider mostly the Network (i.e. IP) and Transport (i.e. TCP, UDP) layers. The following are the data which I will consider out of all fields of the two headers:

1. Transport Protocol
2. Source IP address
3. Destination IP address
4. Source Port number (or equivalent data)
5. Destination Port number (or equivalent data)
6. Transport Flags (if any)
7. Timestamp

The first 3 are part of the IP headers, 4 to 6 of the Transport protocol headers whereas the Timestamp is added by the firewall and marks the moment at which the packet is received by it.

The points 1 to 5 of the previous list uniquely characterize a TCP/IP (public) connection, no two connections with the same first 5 numbers can exist.

A Packet Filtering Firewall works as follows: there is a list of rules which select a packet based on the numbers 1 to 5 of the previous list. So each rule says that if a packet carries data for a particular Transport Protocol, from a certain Source to a Destination IP address and ports, then apply to this packet this `action`. The first rule which matches a packet is applied, and the packet breaks out of the matching of the rules. Usually the possible actions are to **A**ccept a packet, by which a packet is allowed to pass through the firewall, to **D**eny or **D**rop the packet, by which the packet is silently discarded, or to **R**eject a packet, by which the packet is discarded but an error message is sent back to the sender.

In most firewalls of this kind, it is the user who chooses the order of the rules, some firewalls instead create themselves the order based on various criteria.

It is important to notice that all packets go through the same list of rules (the only exception can be for firewalls that allow to specify different set of rules for different interfaces, for which the statement is true interface by interface). In particular, if **A** sends a packet to **B** and **B** replies with a packet, there must be a rule for accepting the packet which goes from **A** to **B** and another rule to accept the packet from **B** to **A** since the two packets have exchanged the Source and Destination addresses. Very often this situation leads system administrators to allow all packets originating from **B** to go to **A**, which can lead to security problems.

Finally it should be noted that the Source Port number for **A** is very often a not registered and currently not used port with number between 1024 and 65535, in practice more or less a random number in this range. This guarantees that not two connections have the same numbers, but at the same time it makes it often impossible to filter on the Source Port of the original sender. Thus when **A** sends a packet, it is often not possible to filter on the Source Port number, and when **B** sends back a reply, it is often not possible to filter on the Destination Port number.

2.2 Stateful Inspection Firewalls

A Stateful Inspection firewall builds on top of a Packet Filtering firewall, by recognizing that once a packet has been accepted, most times one has to expect a packet as a reply. Thus if **A** sends a packet to **B**, then the firewall has to expect that **B** will reply to **A**, and if the firewall has allowed the first packet, it should allow also the reply. In this case we treat in a very different way the first packet of an IP connection, from all the other packets belonging to that connection:

1. The first packet of an IP connection goes through a list of Filtering Rules which can be made specific only for first-packets, notice that these rules will apply in our example only to packets from **A** to **B**, and that there will be no rules of this kind for the packets from **B** to **A**;
2. Once a first-packet is Accepted, the firewall writes in the **Connections Table** all the 7 numbers (see previous section) which characterize the IP connection;
3. When a packet arrives at the firewall, the first thing it does is to check if the packet belongs to a known IP connection as recorded in the Connections Table: if it does it can apply some special rules or just Accept the packet; notice that both the packets from **A** to **B** and the replies of **B** to **A** belong to the same connection, so that a packet is tested in the Connections Table with respected to the Source and Destination addresses direct and inverted. If there is no match, then this is a first-packet and rule 1 above applies.

To enter in more details in what the firewall does (or should do), I will consider the three most common Protocols: UDP, ICMP and TCP.

2.2.1 UDP

Assume that **A** sends a UDP packet to **B**. **F** is in the middle and receives this packet, it is a first-packet and, by looking at the rules, **F** can do 3 things with it:

1. Drop the packet;
2. Reject the packet: in this case the packet is discarded and an ICMP error message, usually of the type `Port Unreachable`, is sent back to the sender (in the next section I will say more on ICMP packets);
3. Accept the packet and forward it to **B**: in this case the 7 numbers which characterize the IP connection started by this packet are written in the Connections Table.

Only in case 3. **B** will receive a packet. From the point of view of **F**, now **B** can do the following things:

1. **B** can send back an UDP reply to **A**: in this case **F** receives this packets, checks if it belongs to a known connection and since it does, it forwards the packet to **A**;
2. **B** can send back an ICMP error message to **A**: in this case **F** receives this packet, checks if it belongs to a known connection and since it does in the sense that it is a packet *Related* to a known connection, it forwards the packet to **A** and deletes the entry from the table;
3. **B** can send back nothing: in this case **F** waits some time and then deletes the connection from the Connections Table (it is here that the Timestamp in the Connections Table plays its role).

When should **F** delete the entry in the Connections Table for point 1? Since UDP is not a connection oriented protocol, a firewall which does not understand the payload of UDP packets cannot know if more UDP packets belonging to this connection are going to be exchanged or not. Thus most firewalls update the timestamp every time a packet of a UDP connection passes, and then just wait for the timeout to expire, in case that another packet of this connection is sent to or from **A**, from or to **B**.

Finally, how can **F** know that in case 2. an ICMP error message belongs to a particular UDP connection? It is very simple, since in the data portion of the ICMP error message there are the headers (or part of the headers) of the original packet which generated the error. By looking at the data portion of the ICMP message, the firewall can deduce to which connection the ICMP packet belongs to.

2.2.2 ICMP

There are two kinds of ICMP packets. The first are used to test the network and the most famous are the echo-request/echo-reply couple. In practice from the point of view of the firewall they behave as UDP packets, just that for one request packet from **A** to **B** there is at most one reply packet from **B** to **A**. **B** can reply to **A** only with one echo-reply ICMP packet, or with nothing. To avoid loops, it is indeed forbidden to send ICMP error messages if the packet which generates the error is an ICMP packet. Thus for ICMP echo-request/reply packets a Stateful Inspection firewall can allow the echo-request, put its data in the Connections Table, and allow the echo-reply packet back or close the connection after the timeout. After the echo-reply has passed, the connection can be removed from the Connections Table since for each echo-request **B** can send only one echo-reply.

The other kind of ICMP packets are error messages. These packets are generated by the TCP/IP stack (or the firewall code itself) in response to a packet which tries to do something not allowed. For example if a packet tries to connect to a service, that is a port, to which no daemon (program) is listening, the TCP/IP stack can reply to the sender with a ICMP port-unreachable message. This is to prevent the sender to keep trying if in case the first packet has gone lost, or something temporary has happened. As already said, in the data portion of the ICMP error message there are the headers (or part of the headers) of the original packet which generated the error. By looking at the data portion of the ICMP message, the original sender **A** can deduce which program has generated the original packet and report to it the error, and the firewall can deduce to which connection the ICMP packet belongs to.

2.2.3 TCP

TCP is a connection oriented protocol, that is ... is very complicated. Thus I are going to consider only the basic features which are of interest from the point of view of the firewall **F.** TCP connections are divided in 3 phases: the establishment of the connection in both directions, the exchange of the data, and the closing of the connection on both sides. To keep track of the status of the TCP connection, TCP uses some flag bits. The flag bits which will be important for us are:

1. SYN: this bit is used to start the connection, notice that a TCP connection must be established on both sides, so that first **A** sends a packet with the SYN bit up, and to this **B** replies with a packet also with the SYN bit up to request the connection in the opposite direction (SYN means synchronize);
2. ACK: this bit acknowledges the receipt of one or more packets, which ones are acknowledged has to do with the window scaling of TCP in which I do not want to go here, the important point is to know what this bit means;
3. FIN: if this bit is up, the side which has sent this packet is requesting to close the connection in this direction, that is the one which has sent this packet does not have anything else to send;
4. RST: this bit requests the reset or truncation of the connection, in other words the connection must stop immediately in both directions.

Lets follow what happens in a simple[1] TCP connection between **A**, which requests first the connection, and **B**.

1. **A** sends a packet with only the SYN flag up among the 4 flags just mentioned;

2. **F** can do 3 things as usual:

   1. Drop the packet;
   2. Reject the packet with a TCP reset packet, in this case the firewall sends back a packet with the RST flag up as if it was **B** telling **A** that the service requested is not running on **B**;
   3. Accept the packet, forward it to **B** and add an entry in the Connections Table; <u>notice</u> that at this moment for **F** the IP connection is established, whereas for **A** and **B** the TCP connection is not yet established since the 3-way handshake is not finished;

   in the first two cases no packet arrives to **B** and thus nothing else passes, otherwise the communication can continue;

3. **B** receives the SYN packet and it too can do 3 things:

   1. Drop the packet;
   2. Reject the packet with a TCP reset (i.e. with the RST flag up) packet, in this case **B** tells **A** that the service requested is not running on **B**;
   3. Reply to **A** sending a packet with the SYN and ACK flags up: the ACK flag is up to acknowledge the receipt of the first SYN packet and the SYN flag is up to request the opening of the connection in the opposite direction (no other flags are up);

   in the last two cases a packet is sent back and **F** will receive something;

4. if **F** receives a packet with the RST flag up for a TCP connection which exists in the Connections Table, it must forward it to the other host and cancel the connection from the Connections Table; otherwise **F** now receives from **B** a packet with the SYN and ACK flags up corresponding to a IP connection which exists in the Connections Table, and it allows it to pass;

5. **A** receives the SYN-ACK packet from **B** and sends back a packet with only an ACK to acknowledge the receipt and opening of the connection in the other direction.

From now on, **A** and **B** exchange packets only with the ACK flag up among the 4 flags mentioned before, and the firewall let the packets pass. To gracefully end the connection, let say that **B**, which has finished first, sends a packet with the ACK-FIN flags up, to which **A**, replies with a packet with the ACK flag up. Then when **A** has finished too, it too sends a packet with ACK-FIN flags up. Finally **B** sends the last packet with the ACK flag up. When the firewall **F** sees the two packets in opposite directions with the FIN flag up, and the last packet with only the ACK flag, it can delete the connection from the Connections Table.

A fundamental observation is that all packets of a TCP connection, except the first one, have the ACK flag up. Moreover, various combination of flags are not allowed, some never, other in some moment of the connection. I can list the following important combinations of flags:

1 There are more complicated scenarios.

1. The SYN flag is up only in the first two packets, one in each direction, of a TCP connection;
2. The first packet of a TCP connection must have the SYN flag up and the ACK and RST flags down; it can have the FIN flag up, but this is used very rarely and for most application the FIN flag is down in the first packet of a TCP connection;
3. the ACK flag must be up in each packet after the first.

Since a Stateful Inspection firewall treats differently the first packet of a IP connection from all the other, I can summarize the rules for the firewall as follows:

1. **F** should allow TCP packets with specified source and destination addresses and ports with only the SYN flag up out of the SYN,ACK,RST (and almost always also FIN) flags, as first packet establishing an IP connection (of course only if we want to allow this TCP connection through the firewall);
2. **F** should then allow all packets of the established IP connection in both direction; a good firewall should check if the TCP flags which are up are allowed; moreover when it sees that the RST or FIN flags are up, it should close the connection, that is delete it from the Connections Table; Moreover **F** should manage the timestamp and the timeouts in a manner consistent with the timeouts of the TCP protocol (which is not a completely trivial task).

Thus it is quite easy to identify the first packet of a TCP connection, and to allow it to pass through the firewall. After that, it should be the firewall which follows the connection and allows or denies the packets.

Following what TCP does is not an easy task: packets can arrive out of order, or be retransmitted, various flags in various combination can mean different things depending when they are set during a connection, and so on. Sometimes it is not only a technical difficulty of implementing an algorithm which follows the TCP connection, but also that security decisions must be taken. For example, one could decide to have a firewall to drop packets which arrive out of order because this could be a sign of someone who is trying to inject packets in a connection. What would happen is that slowly, very slowly, packets will be retransmitted, hopefully in the correct order (but SACK, FACK, DSACK etc. would actually make this more difficult to happen). In practice by dropping packets which arrive out of order we could end up to reduce tremendously the speed of the connection or to block it completely. On the other side, illegal or not correct TCP flags combinations are often used by attackers to scan (map) the internal network and hosts, and thus are the preliminary step of a true attack. A firewall should then drop these packets.

2.2.4 Weaknesses

A Stateful Inspection firewall has, among others, two basic weaknesses which are worth mentioning here:

1. the firewall makes most of its decisions based on the **Source IP address and Source port**, thus it must trust that these data are correct; it is practically impossible to understand if the source addresses and port written in the headers of a packet are fake, that is the host which sent the packet is not the one which has that IP address. There are cases when this is fine, for example when a host is NATted by a firewall or a router: in this case the source address and port which the firewall **F** receives are not the ones of the original host, but the one of the device which does the NAT/PAT translation. The security problems arise when a malicious host sends a packet writing in the headers of the packet the address of another legitimate host (this is called *Spoofing*). Thus the firewall thinks that the packet comes from the legitimate host, whereas instead it comes from another. Of course the malicious host will not receive the reply packets back, which are sent to the legitimate host, but this could be enough to pass through the firewall and do damage.[2] Routing can help a little to minimize this problem, since it tells us which classes of IP addresses are behind each interface, so that if a packet arrives on an interface with a source IP address which belongs to a class which is behind another interface, something is wrong and most probably (if our routing is set up correctly !!!) this source IP address is spoofed. For TCP connections the strict checking of the TCP flags of each packet can also help to prevent the injection in an active connection of forged packets by a malicious host.
2. The firewall does not look in the data portion of the packets, but only in the headers, thus it cannot know if the data contains some malicious code. The simplest example is the email viruses: if a Stateful Inspection firewall allows the transit of emails, it can not know if in an email there is or not a virus. Another problem is with application protocols like ftp in which after the first connection is made on predefined ports, new connections are opened always between the same hosts on ports which are agreed upon in the first connection. Thus if the firewall does not look in the contents of the data, it cannot know that the two hosts want to start a new connection, *related* to the original ftp connection, using different ports, and this new connection should be allowed too if the first has been allowed.

The first problem is very difficult to solve with today technologies. In practice one would like to identify with certainty the source of each packet which arrives at the firewall. This can be, and in same cases is, done for the internal network by using authentication schemes for users or machines, it is in general practically impossible for the Internet. One case in which this identification is done with some or partial success, is in E-commerce where Secure Web Servers are identified with their SSL Certificates which are issued by a Certificate Authority. Other encrypted communications, as in the case of IPSEC VPN, permit (depending on the configuration) to exactly identify the remote host.

2.3 Application Layer Gateways

We can keep building on top of the previous kind of firewall, by adding to the Stateful Inspection features also the ability to look into the data portion of the packets. It must be noted that at this point the firewall gets a <u>completely different structure</u>. Up to now, all rules are valid for all IP packets, with some *minor* modifications due to the difference bewteen the

2 This technique is also used to attack the legitimate host but making it look like the attack comes from our **B** host behind the firewall, for example for Denial of Service attacks.

various Transport protocols (TCP, UDP etc.). There are not too many Transport protocols and they are all reasonably similar in their fundamental structure.

Instead *Application Protocols*, like ftp, http, dns, smtp etc., even if usually adopt TCP or/and UDP as Transport protocol, can be really very different. For this reason most of the ALGs (Application Layer Gateways) are built in a modular form, to allow the possibility of adding a new module which understands a particular new protocol, or to activate just the modules for the protocols of interest. Thus we can have modules for ftp (active, passive, behind NAT or PAT), http, dns, H323 etc. and even modules for private protocols which are not public standards. The porpuse of these modules can be different:

1. the simplest modules are those for the protocols which require more dynamical connections to be opened, like ftp or H323, after the first one is allowed on the well known ports. These modules can just restrict themselves to look at some particular data in the packets (for example the `PORT` command in ftp) and from these learn the details of the new connection that is going to be opened. Once this happens they will insert in the Connections Table a new entry for this *related* connection, even if no rule exists which specifies these ports, and they will then allow the new connection to pass through the firewall. In this way, ports are opened and closed just when they are needed, and only after a first allowed connection has been established. This allows for example to have a firewall with all connections from outside denied by default, but if a H323 connection from the inside to the outside is allowed, then automatically all needed ports are opened also for the incoming connections from the outside related to this one.
2. A little more complicated are modules which look at some particular dangerous commands or data for a protocol. The prime example is the http module which filters out (delete or just comment out) Java applets inserted web pages. The module just looks for the command which starts a Java applet, and ignores everything else in the payload of an http packet.
3. In the future there will be modules which will check that the contents of a packet strictly adhere to the Application protocol to which they belong. This would prevent attacker to insert illegal data, from the point of view of the Application protocol, in the packets. The problem with this is that not always all applications adhere strictly to the protocols, there can be vendor extensions or just bad implementations, and such a module would just prevent these connections even if they are not malicious.

One can imagine to have more advanced modules to do a real check of the contents of each packet, or even reassembling packets to understand exactly the possible consequences of the data on the destination host. Unfortunately one must keep in mind that for a firewall it is very expensive in terms or resources (RAM, CPU etc.) to analyse the payload of each or most packets. Thus the more complicated and sophisticated these modules are, the more powerful the hardware of the firewall must be. Moreover, the more checks the firewall must do, the longer delay it adds in the communication, even more if packets must be reassembled in files and so on. At this point we should strike a balance between what these modules can do, and what the hardware allows to do in such a way to introduce delays not noticeable in real time communications. Thus we could divide the Application protocols in 2 classes:

1. the ones which have timing constraints of the order of the (hundred of) milliseconds;
2. the ones for which delays of seconds or even minutes are not so important.

To the first class belong some interactive protocols like teleconferencing (H323), remote logins etc., whereas to the second class belong protocols from web surfing (which requires delays introduced by a firewall not more than a few seconds) to e-mail, where delays even of a few minutes do not really matter. The first class is usually addressed with ALG modules, whereas for the second class one can adopt Proxies.

2.4 Proxy Firewalls

For Application protocols which are not too sensitive to delays, the more secure solution is to introduce an external host, parallel to the firewall which would fully filter all communications between the internal and external hosts. Thus **F** will send all packets of a certain kind, for example to or from port 80 TCP (i.e. http), to an host **G** which is directly connected to it.[3] Lets look at how this works in a simple example:

1. **A** sends a http request (first packet) to **B** on TCP port 80 of **B** from port 2345 of **A.**
2. **F** receives this packet and from the rules for first-packets knows that the packet is allowed and that it should be *redirected* to **G** on TCP port 8080. (Alternatively the user on **A** could have defined **G** as a http-proxy, and then **A** would send the first packet directly to **G;** a rule for this kind of first-packets should also be present in **F.**)
3. **G** receives the packet and behaves as if it was **B**, that is the final host to which the packet was addressed; thus **G** establishes a TCP connection with **A** posing as the final web server; this implies that the program running on **G** and listening on port TCP 8080 must emulate a full Web (httpd) server.
4. **G** sends a http request (first-packet) to **B** on TCP port 80 of **B** from port 5345 of **G**. What **G** is doing is to behave now as a Web client (a Web browser) and it forwards the request just arrived from **A** to **B**, but this time the request is coming from **G** as source address.
5. **F** receives this new first-packet, and has a rule which allows **G** to connect to **B** on port TCP 80, thus a new connection is established from **G** to **B**.
6. **B** sends the requested Web page to **G** thinking that **G** is a Web browser; **G** then sends the Web page to **A**, which thinks that **G** is a final Web server.

This complicated setup accomplishes the following things:

1. **A** can connect only with **G** and believes that **G** is a Web server.
2. **G** receives all requests of Web pages from **A** and can decide which to honour and which to deny.
3. **G** asks all Web pages for **A** to the final Web Servers (**B** in this example) and receives all pages; **G** can reassemble and examine all pages before sending them to **A**; With respect to **B**, **G** behaves as a Web browser (a client).
4. **A** and **B** can never establish a direct TCP connection, they connect directly only with **G**.

3 On some firewalls the Proxy program could run on the same hardware device as the firewall itself, the physical division in two hosts is for a more clear understanding of the data flow and it is also the suggested implementation for security and reliability reasons.

Since **G** implements both a http client and a server, it is able to address all possible issues of the http protocol and it is then able to apply all possible kinds of filters and security checks on the Web pages which are requested by **A**. Moreover, **B** never even knows that **A** exists.

Finally **G** can also *cache* the most requested pages from **A**, and serve them to **A** without getting them again from **B**.

Besides http Proxy, the other very well known one is the electronic mail (smtp protocol) anti-virus proxies. In this case the firewall will forward to a smtp proxy all emails, and the smtp proxy checks every message for the presence of viruses. In the case of smtp it is very easy to implement a proxy, since for how the protocol is defined, any smtp server can be considered to be a proxy in the sense I are using here. Moreover the firewall does not even need to do any packet redirection if the DNS records are configured appropriately.

2.5 Examples

After these general notes I am going to give an example just to end with something practical. I do not want to give another set of the infinite collection of the *My favourite firewalls rules*, but I should give you something anyway. I will give rules for a Stateful Inspection firewall with some simple Application Layer modules.

2.5.1 The Strategy

To write some rules one has to start from some point of view. Here is a simple starting point:

1. Classify the traffic which is allowed to pass through the firewall, do it interface by interface considering all <u>incoming</u> traffic to the firewall and only for the first packet of each connection.
2. Adopt the point of view that all traffic which is not explicitly allowed is denied.
3. Write rules which start by accepting all established connections
4. then filter all incoming traffic (first packets) depending on the incoming interface (if the firewall has different rules for different interfaces, do this independently interface by interface)
5. if needed, filter also the traffic depending on the outgoing interface (this could be needed in complicated firewalls with many interfaces, simple firewalls with 3 interfaces rarely need this); in any case specify as much as possible all rules, restricting them to the most.

Adopting this approach it is usually possible to write rules for a firewall in a compact and understandable way. Moreover, the lower the number of rules and the simpler their organization, the easiest it will be to have efficient and correct policies, and the easier will be to understand what is happening to the traffic through the firewall.

2.5.2 A simple example with Linux iptables

Consider a Linux firewall with iptables and 2 interfaces, eth0 towards the inside and eth1 towards the Internet. A part of the configuration of the FORWARD chain could be like the following (comments are after each group of instructions):

```
iptables -P FORWARD DROP
```

This sets the default policy of the chain to DROP.

```
modprobe ip_conntrack_ftp
modprobe ip_nat_ftp
```

These start the (ALG) modules which allows the automatic opening of ftp ports .

```
iptables -N logdrop
iptables -A logdrop -m limit --limit 10/s --limit-burst 60
           -j LOG --log-ip-options --log-tcp-options
           --log-prefix IPTlogdrop
iptables -A logdrop -m limit --limit 10/s --limit-burst 60
           -p tcp -j REJECT --reject-with tcp-reset
iptables -A logdrop -m limit --limit 10/s --limit-burst 60
           -p ! icmp -j REJECT
           --reject-with icmp-port-unreachable
iptables -A logdrop -j DROP
```

These create an auxiliary chain which sends to syslog all packets to which it is denied to pass through the firewall. Then for TCP packets a tcp reset is sent back, for all other not icmp packet a icmp port unreachable is sent back. All packets are anyway discarded. All reject and log rules are rate limited to prevent to fill up the logs of the log server, and to act as a indirect Deny of Service agent. (If you fear that your firewall could be used for an indirect DoS, do not use the two Reject rules in this list.)

```
iptables -A FORWARD -m state --state INVALID -j logdrop
```

This rules drops (or rejects) all packets which are classified as INVALID by iptables. Among them there are also some illegal TCP flags combinations, or packets with impossible IP addresses. If you have also the tcp-window-tracking patch from patch-o-matic, all packets with illegal combination of TCP flags, or out of the tcp window, are marked as INVALID and dropped by this rule.

```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED
           -j ACCEPT
```

This rule accepts all established IP connections as listed in the Connections Table, the icmp error messages related to the allowed connections, and also all *related* ftp connections with the help of the modules we have previously loaded. Most of the traffic of the firewall will be accepted by this rule. From now on only first packets are examined.

```
iptables -A FORWARD -i eth0 -o eth1 -s 192.168.1.0/24
           -p tcp -m multiport
           --destination-port 21,22,53,80,443,993
           --tcp-flags SYN,ACK,FIN,RST SYN -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -s 192.168.1.0/24
```

```
                      -p udp -m multiport --destination-port 53,123
                      -j ACCEPT
          iptables -A FORWARD -i eth0 -o eth1 -s 192.168.1.0/24
                      -p icmp --icmp-type echo-request -j ACCEPT
```

These 3 rules accept all first packets for the allowed connection which are going out from our network to the Internet. Notice that ftp (TCP port 21) is allowed, the other ports used by ftp will be opened automatically by the modules. Our security policy allows the internal users to connect to the listed services to any external server, so no filter on the destination IP addresses is set.

```
          iptables -A FORWARD -j logdrop
```

This rule simply says that all other packets are to be dropped (or rejected).

```
          iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth1 -j SNAT
                      --to-source 1.2.3.4
```

This rule says to PAT all internal IP addresses in the private class 192.168.1.0/24 to the external internet address 1.2.3.4.

These rules are only a first attempt to build a firewall, many issues have not been considered like the INPUT and OUTPUT chain of the Linux firewall, or a more detailed logging or accounting. For example it is suggested to filter all illegal source addresses, the rules for doing so could be the following:

```
          iptables -A FORWARD -s 127.0.0.0/8 -j logdrop
          iptables -A FORWARD -s 0.0.0.0/8 -j logdrop
          iptables -A FORWARD -s 10.0.0.0/8 -j logdrop
          iptables -A FORWARD -s 172.16.0.0/12 -j logdrop
          iptables -A FORWARD -i ! eth0 -s 192.168.0.0/16 -j logdrop
          iptables -A FORWARD -s 224.0.0.0/3 -j logdrop
          iptables -A FORWARD -s 192.168.1.254/32 -j logdrop
          iptables -A FORWARD -s 1.2.3.4/32 -j logdrop
```

which should be inserted just after the Accept ESTABLISHED,RELATED rule. A similar rule should be added also to filter the outgoing packets for illegal destination addresses, since a destination of 10.4.5.6 would be routed to the Internet, but it is obviously not a valid address.

3. Firewalls and other Security Devices

Firewalls do not live alone and do not solve alone all security problems of an ICT deployment. Actually quite the contrary, they are a very little, but important, tool in the full security system. To make a firewall more effective, other tools should be used together with it. In particular Intrusion Detection/Prevention Systems and cryptography/encrypted connections should be adopted.

Whereas a firewall is very good in filtering the traffic, an IDS/IPS should recognize patterns in the traffic which can indicate that an attack is starting or under way. An Intrusion D/P System can analyse off-line the traffic which is reaching the firewall, and in almost real time correlate patterns or discover harmful payloads in the packets. This analysis requires often quite some hardware resources and some time on the scale of the millisecond delays that at most a firewall can introduce, which makes it impossible to be done by the firewall. What instead an I[D/P]S can do is to alert an operator or to automatically change the rules of the firewall when it realizes that an attack is under way.

Encrypted connections can help the firewall, but at the same time introduce other problems. Establishing an encrypted connection between two remote networks secures the communications between them, and also adds a strong authentication since the establishment of the encrypted tunnel requires it. Thus an encrypted tunnel is good traffic, from the point of view of the firewall, because at least the two endpoints are who they claim to be (man-in-the-middle attacks notwithstanding). On the other side the firewall must trust ultimately both endpoints, since being the traffic encrypted the firewall has no way to look at the contents of the packets, and doesn't have even a vague idea of which kind of application protocols are passing in the tunnel, or of the original payload. So if one of the two is cheating, the firewall cannot do anything about it.

To minimize the security risks, one has to plan a reasonable complete system, taking in consideration not only the role of the firewall, but also that of all other components of the network.