9

Kinds of Firewalls

fire wall *noun*: A fireproof wall used as a barrier to prevent the spread of a fire.

—AMERICAN HERITAGE DICTIONARY

Some people define a firewall as a specific box designed to filter Internet traffic—something you buy or build. But you may already have a firewall. Most routers incorporate simple packet filters; depending on your security, such a filter may be all you need. If nothing else, a router can be part of a total firewall system—firewalls need not be one simple box.

We think a firewall is any device, software, or arrangement or equipment that limits network access. It can be a box that you buy or build, or a software layer in something else. Today, firewalls come "for free" inside many devices: routers, modems, wireless base stations, and IP switches, to name a few. Software firewalls are available for (or included with) all popular operating systems. They may be a *client shim* (a software layer) inside a PC running Windows, or a set of filtering rules implemented in a UNIX kernel.

The quality of all of these firewalls can be quite good: The technology has progressed nicely since the dawn of the Internet. You can buy fine devices, and you can build them using free software. When you pay for a firewall, you may get fancier interfaces or more thorough application-level filtering. You may also get customer support, which is not available for the roll-your-own varieties of firewalls.

Firewalls can filter at a number of different levels in a network protocol stack. There are three main categories: *packet filtering*, *circuit gateway*s, and *application gateway*s. Each of these is characterized by the protocol level it controls, from lowest to highest, but these categories get blurred, as you will see. For example, a packet filter runs at the IP level, but may peek inside for TCP information, which is at the circuit level. Commonly, more than one of these is used at the same time. As noted earlier, mail is often routed through an application gateway even when no security firewall is used. There is also a fourth type of firewall—a *dynamic packet filter* is a combination of a packet filter and a circuit-level gateway, and it often has application layer semantics as well.

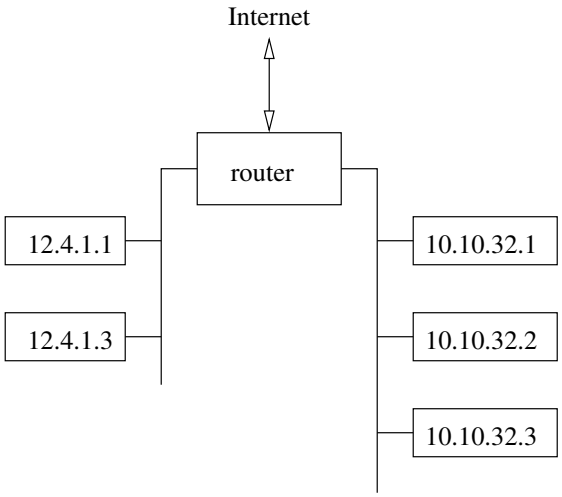

Figure 9.1: A simple home or business network. The hosts on the right have RFC 1918 private addresses, which are unreachable from the Internet. The hosts on the left are reachable. The hosts can talk to each other as well. To attack a host on the right, one of the left-hand hosts has to be subverted. In a sense, the router acts as a firewall, though the only filtering rules might be route entries.

There are other arrangements that can limit network access. Consider the network shown in Figure 9.1. This network has two branches: One contains highly attack-resistant hosts, the other has systems either highly susceptible to attack or with no need to access the Internet (*e.g.,* network printers). Hosts on the first net have routable Internet addresses; those on the second have RFC 1918 addressing. The nets can talk to each other, but people on the Internet can reach only the announced hosts—no addressing is available to reach the second network, unless one can bounce packets off the accessible hosts, or compromise one of them. (In some environments, it's possible to achieve the same effect without even using a router, by having two networks share the same wire.)

9.1 Packet Filters

Packet filters can provide a cheap and useful level of gateway security. Used by themselves, they are cheap: the filtering abilities come with the router software. Because you probably need a router to connect to the Internet in the first place, there is no extra charge. Even if the router belongs to your network service provider, they may be willing to install any filters you wish.

Packet filters work by dropping packets based on their source or destination addresses or port numbers. Little or no context is kept; decisions are made based solely on the contents of the

current packet. Depending on the type of router, filtering may be done at the incoming interface, the outgoing interface, or both. The administrator makes a list of the acceptable machines and services and a stoplist of unacceptable machines or services. It is easy to permit or deny access at the host or network level with a packet filter. For example, one can permit any IP access between host A and B, or deny any access to B from any machine but A.

Packet filters work well for blocking spoofed packets, either incoming or outgoing. Your ISP can ensure that you emit only packets with valid source addresses (this is called *ingress filtering* by the ISP [Ferguson and Senie, 2000].) You can ensure that incoming packets do not have a source address of your own network address space, or have loopback addresses. You can also apply *egress filtering*: making sure that your site doesn't emit any packets with inappropriate addresses. These rules can become prohibitive if your address space is large and complex.

Most security policies require finer control than packet filters can provide. For example, one might want to allow any host to connect to machine A, but only to send or receive mail. Other services may or may not be permitted. Packet filtering allows some control at this level, but it is a dangerous and error-prone process. To do it right, one needs intimate knowledge of TCP and UDP port utilization on a number of operating systems.

> *This is one of the reasons we do not like packet filters very much. As Chapman [1992] has shown, if you get these tables wrong, you may inadvertently let in the Bad Guys.*
>
> *In fact, though we proofread our sample rules extensively and carefully in the first edition of this book, we still had a mistake in them. They are very hard to get right unless the policy to be enforced is very simple.*

Even with a perfectly implemented filter, some compromises can be dangerous. We discuss these later.

Configuring a packet filter is a three-step process. First, of course, one must know what should and should not be permitted. That is, one must have a security policy, as explained in Section 1.2. Second, the allowable types of packets must be specified formally, in terms of logical expressions on packet fields. Finally—and this can be remarkably difficult—the expressions must be rewritten in whatever syntax your vendor supports.

An example is helpful. Suppose that one part of your security policy allowed inbound mail (SMTP, port 25), but only to your gateway machine. However, mail from some particular site SPIGOT is to be blocked, because they host spammers. A filter that implemented such a ruleset might look like the following:

action	ourhost	port	theirhost	port	comment
block	*	*	SPIGOT	*	*we don't trust these people*
allow	OUR-GW	25	*	*	*connection to our SMTP port*

The rules are applied in order from top to bottom. Packets not explicitly allowed by a filter rule are rejected. That is, every ruleset is followed by an implicit rule reading as follows:

action	ourhost	port	theirhost	port	comment
block	*	*	*	*	*default*

This fits with our general philosophy: all that is not expressly permitted is prohibited.

Note carefully the distinction between the first ruleset, and the one following, which is intended to implement the policy "any inside host can send mail to the outside":

action	ourhost	port	theirhost	port	comment
allow	*	*	*	25	*connection to their SMTP port*

The call may come from any port on an inside machine, but will be directed to port 25 on the outside. This ruleset seems simple and obvious. It is also wrong.

The problem is that the restriction we have defined is based solely on the outside host's port number. While port 25 is indeed the normal mail port, there is no way we can control that on a foreign host. An enemy can access any internal machine and port by originating his or her call from port 25 on the outside machine.

A better rule would be to permit *outgoing* calls to port 25. That is, we want to permit our hosts to make calls to someone else's port 25, so that we know what's going on: mail delivery. An incoming call *from* port 25 implements some service of the caller's choosing. Fortunately, the distinction between incoming and outgoing calls can be made in a simple packet filter if we expand our notation a bit.

A TCP conversation consists of packets flowing in two directions. Even if all of the data is flowing one way, acknowledgment packets and control packets must flow the other way. We can accomplish what we want by paying attention to the direction of the packet, and by looking at some of the control fields. In particular, an initial open request packet in TCP does not have the `ACK` bit set in the header; all other TCP packets do. (Strictly speaking, that is not true. Some packets will have just the reset (`RST`) bit set. This is an uncommon case, which we do not discuss further, except to note that one should generally allow naked `RST` packets through one's filters.) Thus, packets with `ACK` set are part of an ongoing conversation; packets without it represent connection establishment messages, which we will permit only from internal hosts. The idea is that an outsider cannot initiate a connection, but can continue one. One must believe that an inside kernel will reject a continuation packet for a TCP session that has not been initiated. To date, this is a fair assumption. Thus, we can write our ruleset as follows, keying our rules by the source and destination fields, rather than the more nebulous "OURHOST" and "THEIRHOST":

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	25		*our packets to their SMTP port*
allow	*	25	*	*	`ACK`	*their replies*

The notation "{our hosts}" describes a set of machines, any one of which is eligible. In a real packet filter, you could either list the machines explicitly or specify a group of machines, probably by the network number portion of the IP address, e.g., something like 10.2.42.0/24.

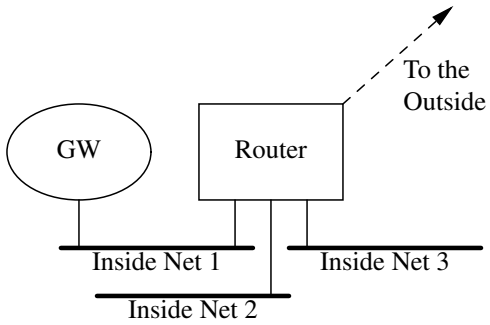

Figure 9.2: A firewall router with multiple internal networks.

9.1.1 Network Topology and Address-Spoofing

For reasons of economy, it is sometimes desirable to use a single router both as a firewall and to route internal-to-internal traffic. Consider the network shown in Figure 9.2. There are four networks, one external and three internal. Net 1, the DMZ net, is inhabited solely by a gateway machine GW. The intended policies are as follows:

- Very limited connections are permitted through the router between GW and the outside world.
- Very limited, but possibly different, connections are permitted between GW and anything on NET 2 or NET 3. This is protection against GW being compromised.
- Anything can pass between NET 2 or NET 3.
- Outgoing calls only are allowed between NET 2 or NET 3 and the external link.

What sorts of filter rules should be specified? This situation is very difficult if only output filtering is done. First, a rule permitting open access to NET 2 must rely on a source address belonging to NET 3. Second, nothing prevents an attacker from sending in packets from the outside that claim to be from an internal machine. Vital information—that legitimate NET 3 packets can only arrive via one particular wire—has been ignored.

Address-spoofing attacks like this are difficult to mount, but are by no means out of the question. Simpleminded attacks using IP source routing are almost foolproof, unless your firewall filters out these packets. But there are more sophisticated attacks as well. A number of these are described in [Bellovin, 1989]. Detecting them is virtually impossible unless source-address filtering and logging are done.

Such measures do not eliminate all possible attacks via address-spoofing. An attacker can still impersonate a host that is trusted but not on an internal network. One should not trust hosts outside of one's administrative control.

Assume, then, that filtering takes place on input, and that we wish to allow any outgoing call, but permit incoming calls only for mail, and only to our gateway GW. The ruleset for the external interface should read as follows:

action	src	port	dest	port	flags	comment
block	{NET 1}	*	*	*		*block forgeries*
block	{NET 2}	*	*	*		
block	{NET 3}	*	*	*		
allow	*	*	GW	25		*legal calls to us*
allow	*	*	{NET 2}	*	ACK	*replies to our calls*
allow	*	*	{NET 3}	*	ACK	

That is, prevent address forgery, and permit incoming packets if they are to the mailer on the gateway machine, or if they are part of an ongoing conversation initiated by any internal host. Anything else will be rejected.

Note one detail: Our rule specifies the destination host GW, rather than the more general "something on NET 1." If there is only one gateway machine, there is no reason to permit open access to that network. If several hosts collectively formed the gateway, one might opt for simplicity, rather than this slightly tighter security; conversely, if the different machines serve different roles, one might prefer to limit the connectivity to each gateway host to the services it is intended to handle.

The ruleset on the router's interface to NET 1 should be only slightly less restrictive than this one. Choices here depend on one's stance. It certainly makes sense to bar unrestricted internal calls, even from the gateway machine. Some would opt for mail delivery only. We opt for more caution; our gateway machine will speak directly only to other machines running particularly trusted mail server software. Ideally, this would be a different mail server than the gateway uses. One such machine is an internal gateway. The truly paranoid do not permit even this. Rather, a relay machine will call out to GW to pick up any waiting mail. At most, a notification is sent by GW to the relay machine. The intent here is to guard against common-mode failures: If a gateway running our mail software can be subverted that way, internal hosts running the same software can (probably) be compromised in the same fashion.

Our version of the ruleset for the NET 1 interface reads as follows:

action	src	port	dest	port	flags	comment
allow	GW	*	{*partners*}	25		*mail relay*
allow	GW	*	{NET 2}	*	ACK	*replies to inside calls*
allow	GW	*	{NET 3}	*	ACK	
block	GW	*	{NET 2}	*		*stop other calls from* GW
block	GW	*	{NET 3}	*		
allow	GW	*	*	*		*let* GW *call the world*

Again, we prevent spoofing, because the rules all specify GW; only the gateway machine is supposed to be on that net, so nothing else should be permitted to send packets.

If we are using routers that support only output filtering, the recommended topology looks very much like the schematic diagram shown in Figure 9.3. We now need two routers to accomplish the tasks that one router was able to do earlier (see Figure 9.4). At point (a) we use the ruleset that protects against compromised gateways; at point (b) we use the ruleset that guards against address forgery and restricts access to only the gateway machine. We do not have to change the rules even

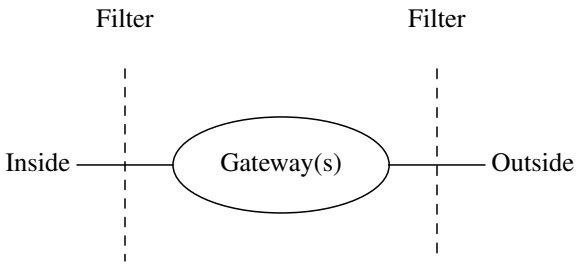

Figure 9.3: Schematic of a firewall.

slightly. Assuming that packets generated by the router itself are not filtered, in a two-port router an input filter on one port is exactly equivalent to an output filter on the other port.

Input filters do permit the router to deflect packets aimed at it. Consider the following rule:

action	src	port	dest	port	flags	comment
block	*	*	ROUTER	*		*prevent router access*

This rejects all nonbroadcast packets destined for the firewall router itself. This rule is probably too strong. One almost certainly needs to permit incoming routing messages. It may also be useful to enable responses to various diagnostic messages that can be sent from the router. Our general rule holds, though: If you do not need it, eliminate it.

One more point bears mentioning if you are using routers that do not provide input filters. The external link on a firewall router is often a simple serial line to a network provider's router. If you are willing to trust the provider, filtering can be done on the output side of that router, thus permitting use of the topology shown in Figure 9.2. But caution is needed: The provider's router probably serves many customers, and hence is subject to more frequent configuration changes.

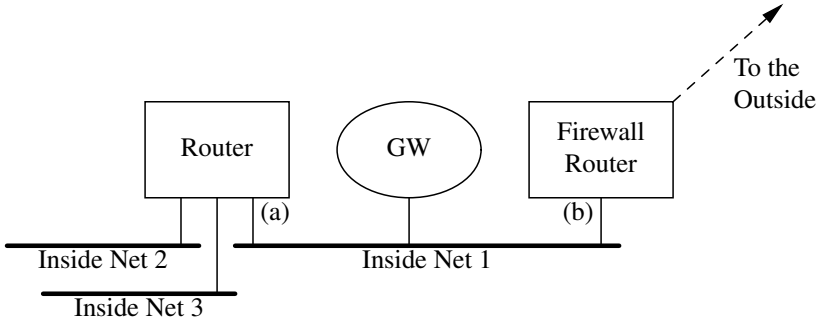

Figure 9.4: A firewall with output-filtering routers.

When Routes Leak

Once upon a time, one of us accidentally tried a *telnet* to the outside from his workstation. It shouldn't have worked, but it did. While the machine did have an Ethernet port connected to the gateway LAN, for monitoring purposes, the transmit leads were cut. How did the packets reach their destination?

It took a lot of investigating before we figured out the answer. We even wondered if there was some sort of inductive coupling across the severed wire ends, but moving them around didn't make the problem go away.

Eventually, we realized the sobering truth: Another router had been connected to the gateway LAN, in support of various experiments. It was improperly configured, and emitted a "default" route entry to the inside. This route propagated throughout our internal networks, providing the monitoring station with a path to the outside.

And the return path? Well, the monitor was, as usual, listening in promiscuous mode to all network traffic. When the acknowledgment packets arrived to be logged, they were processed as well.

The incident could have been avoided if the internal network was monitored for spurious default routes, or if our monitoring machine did not have an IP address that was advertised to the outside world.

The chances of an accident are correspondingly higher. Furthermore, the usefulness of the network provider's router relies on the line being a simple point-to-point link; if you are connected via a multipoint technology, such as X.25, frame relay, or ATM, it may not work.

9.1.2 Routing Filters

It is important to filter routing information. The reason is simple: If a node is completely unreachable, it may as well be disconnected from the net. Its safety is almost that good. (But not quite—if an intermediate host that can reach it is also reachable from the Internet and is compromised, the allegedly unreachable host can be hit next.) To that end, routers need to be able to control what routes they advertise over various interfaces.

Consider again the topology shown in Figure 9.2. Assume this time that hosts on NET 2 and NET 3 are not allowed to speak directly to the outside. They are connected to the router so that they can talk to each other and to the gateway host on NET 1. In that case, the router should not advertise paths to NET 2 or NET 3 on its link to the outside world. Nor should it re-advertise any routes that it learned of by listening on the internal links. The router's configuration mechanisms must be sophisticated enough to support this. (Given the principles presented here, how should the outbound route filter be configured? Answer: Advertise NET 1 only, and ignore the problem

of figuring out everything that should not leak. The best choice is to use RFC 1918 addresses [Rekhter *et al.*, 1996], but this question is more complicated than it appears; see below.)

There is one situation in which "unreachable" hosts can be reached: If the client employs IP source routing. Some routers allow you to disable that feature: if possible, do it. The reason is not just to prevent some hosts from being contacted. An attacker can use source routing to do address-spoofing [Bellovin, 1989]. Caution is indicated: There are bugs in the way some routers and systems block source routing. For that matter, there are bugs in the way many hosts handle source routing; an attacker is as likely to crash your machine as to penetrate it.

If you block source routing—and in general we recommend that you do—you may need to block it at your border routers, rather than in your backbone. Apart from the speed demands on backbone routers, if you have a complex topology (e.g., if you're an ISP or a large company), your network operations folk might need to use source routing to see how *ping* and *traceroute* behave from different places on the net.

Filters must also be applied to routes learned from the outside. This is to guard against *subversion by route confusion*. That is, suppose that an attacker knows that HOST A on NET 1 trusts HOST Z on NET 100. If a fraudulent route to NET 100 is injected into the network, with a better metric than the legitimate route, HOST A can be tricked into believing that the path to HOST Z passes through the attacker's machine. This allows for easy impersonation of the real HOST Z by the attacker.

To some extent, packet filters obviate the need for route filters. If *rlogin* requests are not permitted through the firewall, it does not matter if the route to HOST Z is false—the fraudulent *rlogin* request will not be permitted to pass. But injection of false routes can still be used to subvert legitimate communication between the gateway machine and internal hosts.

As with any sort of address-based filtering, route filtering becomes difficult or impossible in the presence of complex topologies. For example, a company with several locations could not use a commercial data network as a backup to a leased-line network if route filtering were in place; the legitimate backup routes would be rejected as bogus. To be sure, although one could argue that public networks should not be used for sensitive traffic, few companies build their own phone networks. But the risks here are too great; an encrypted tunnel is a better solution.

Some people take route filtering a step further: They deliberately use unofficial IP addresses inside their firewalls, perhaps addresses belonging to someone else [Rekhter *et al.*, 1996]. That way, packets aimed at them will go elsewhere. This is called *route squatting*.

In fact, it is difficult to choose non-announced address spaces in general. True, RFC 1918 provides large blocks of address space for just this purpose, but these options tend to backfire in the long run. Address collisions are almost inevitable when companies merge or set up private IP links, which happens a lot. If foreign addresses are chosen, it becomes difficult to distinguish an intentionally chosen foreign address from one that is there unexpectedly. This can complicate analysis of intranet problems.

As for picking RFC 1918 addresses, we suggest that you pick small blocks in unpopular address ranges (see Figure 13.3). For example, if a company has four divisions, it is common to divide net 10 into four huge sections. Allocating smaller chunks—perhaps from, for example, 10.210.0.0/16—would lessen the chance of collisions.

action	src	port	dest	port	flags	comment
allow	SECONDARY	*	OUR-DNS	53		allow our secondary nameserver access
block	*	*	*	53		no other DNS zone transfers
allow	*	*	*	53	UDP	permit UDP DNS queries
allow	NTP.OUTSIDE	123	NTP.INSIDE	123	UDP	ntp time access
block	*	*	*	69	UDP	no access to our tftpd
block	*	*	*	87		the link service is often misused
block	*	*	*	111		no TCP RPC and ...
block	*	*	*	111	UDP	no UDP RPC and no...
block	*	*	*	2049	UDP	NFS. This is hardly a guarantee
block	*	*	*	2049		TCP NFS is coming: exclude it
block	*	*	*	512		no incoming ”r” commands ...
block	*	*	*	513		...
block	*	*	*	514		...
block	*	*	*	515		no external lpr
block	*	*	*	540		uucpd
block	*	*	*	6000-6100		no incoming X
allow	*	*	ADMINNET	443		encrypted access to transcript mgr
block	*	*	ADMINNET	*		nothing else
block	PCLAB-NET	*	*	*		anon. students in pclab can’t go outside
block	PCLAB-NET	*	*	*	UDP	... not even with TFTP and the like!
allow	*	*	*	*		all other TCP is OK
block	*	*	*	*	UDP	suppress other UDP for now

Figure 9.5: Some filtering rules for a university. Rules without explicit protocol flags refer to TCP. The last rule, blocking all other UDP service, is debatable for a university.

9.1.3 Sample Configurations

Obviously, we cannot give you the exact packet filter for your site, because we don’t know what your policies are, but we can offer some reasonable samples that may serve as a starting point. The samples in Figures 9.5 and 9.6 are derived in part from CERT recommendations.

A university tends to have an open policy about Internet connections. Still, they should block some common services, such as NFS and TFTP. There is no need to export these services to the world. In addition, perhaps there’s a PC lab in a dorm that has been the source of some trouble, so they don’t let them access the Internet. (They have to go through one of the main systems that require an account. This provides some accountability.) Finally, there is to be no access to the administrative computers except for access to a transcript manager. That service, on port 443 (https), uses strong authentication and encryption.

Conversely, a small company or even a home network with an Internet connection might wish to shut out most incoming Internet access, while preserving most outgoing connectivity. A gateway machine receives incoming mail and provides name service for the company’s machines. Figure 9.6 shows a sample filter set. (We show incoming *telnet*, too; you may not want that.) If the company’s e-mail and DNS servers are run by its ISP, those rules can be simplified even more.

Remember that we consider packet filters inadequate, especially when filtering at the port level. In the university case especially, they only slow down an external hacker, but would probably not stop one.

action	src	port	dest	port	flags	comment
allow	*	*	MAILGATE	25		inbound mail access
allow	*	*	MAILGATE	53	UDP	access to our DNS
allow	SECONDARY	*	MAILGATE	53		secondary nameserver access
allow	*	*	MAILGATE	23		incoming telnet access
allow	NTP.OUTSIDE	123	NTP.INSIDE	123	UDP	external time source
allow	INSIDE-NET	*	*	*		outgoing TCP packets are OK
allow	*	*	INSIDE-NET	*	ACK	return ACK packets are OK
block	*	*	*	*		nothing else is OK
block	*	*	*	*	UDP	block other UDP, too

Figure 9.6: Some filtering rules for a small company. Rules without explicit protocol flags refer to TCP.

9.1.4 Packet-Filtering Performance

You do pay a performance penalty for packet filtering. Routers are generally optimized to shuffle packets quickly. The packet filters take time and can defeat optimization efforts, but packet filters are usually installed at the edge of an administrative domain. The router is connected by (at best) a *DS1* (T1) line (1.544 Mb/sec) to the Internet. Usually this serial link is the bottleneck: The CPU in the router has plenty of time to check a few tables.

Although the biggest performance hit may come from doing any filtering at all, the total degradation depends on the number of rules applied at any point. It is better to have one rule specifying a network than to have several rules enumerating different hosts on that network. Choosing this optimization requires that they all accept the same restrictions; whether or not that is feasible depends on the configuration of the various gateway hosts. You may be able to speed things up by ordering the rules so that the most common types of traffic are processed first. (But be careful; correctness is much more important than speed. Test before you discard rules; your router is probably faster than you think.) As always, there are trade-offs.

You may also have performance problems if you use a two-router configuration. In such cases, the inside router may be passing traffic between several internal networks as well. Degradation here is not acceptable.

9.2 Application-Level Filtering

A packet filter doesn't need to understand much about the traffic it is limiting. It looks at the source and destination addresses, and may peek into the UDP or TCP port numbers and flags.

Application-level filters deal with the details of the particular service they are checking, and are usually more complex than packet filters. Rather than using a general-purpose mechanism to allow many different kinds of traffic to flow, special-purpose code can be used for each desired application. For example, an application-level filter for mail will understand RFC 822 headers, MIME-formatted attachments, and may well be able to identify virus-infected software. These filters usually are store-and-forward.

Application gateways have another advantage that in some environments is quite critical: It is easy to log and control *all* incoming and outgoing traffic. Mail can be checked for *dirty words*, indications that proprietary or restricted data is passing the gateway. Web queries can be checked for conformance with company policies, and dangerous mail attachments can be stripped off.

Electronic mail is usually passed through an application-level gateway, regardless of what technology is chosen for the rest of the firewall. Indeed, mail gateways are valuable for their other properties, even without a firewall. Users can keep the same address, regardless of which machine they are using at the time. Internal machine names can be stripped off, hiding possibly valuable data (see Section 2.2.2). Traffic analysis and even content analysis and recording can be performed to look for information leaks.

Note that the mechanisms just described are intended to guard against attack from the outside. A clever insider who wanted to import virus-laden files certainly would not be stopped by them, but it is not a firewall's job to worry about that class of problem.

The principal disadvantage of application-level gateways is the need for a specialized user program or variant user interface for most services provided. In practice, this means that only the most important services will be supported. Proprietary protocols become quite a problem: How do you filter something that is not publicly defined? Moreover, use of such gateways is easiest with applications or client software that make provision for redirection, such as mail, Web access, or FTP.

Some commercial firewalls include a large suite of application-level gateways. By signing appropriate nondisclosure agreements with major vendors, they can add support for numerous proprietary protocols. But this is a mixed blessing. While it's good to have better filtering for these protocols, do you really want many strange and wondrous extra programs—the per-application gateways—running on your firewall? Often, the real answer is to ask whether these protocols should be passed through at all. In many cases, the best spot for such things is on an extranet firewall, one that is restricting traffic already known to be from semi-authorized parties.

9.3 Circuit-Level Gateways

Circuit-level gateways work at the TCP level. TCP connections are relayed through a computer that essentially acts as a wire. The relay computer runs a program that copies bytes between two connections, while perhaps logging or caching the contents. In this scheme, when a client wishes to connect to a server, it connects to a relay host and possibly supplies connection information through a simple protocol. The relay host, in turn, connects to the server. The name and IP address of the client is usually not available to the server.

IP packets do not flow from end to end: the relay host works above that level. All the IP tricks and problems involving fragments, firewalking probes, and so on, are terminated at the relay host, which may be better equipped to handle pathological IP streams. The other side of the relay host emits normal, well-behaved TCP/IP packets. Circuit-level gateways can bridge two networks that do not share any IP connectivity or DNS processing.

Circuit relays are generally used to create specific connections between isolated networks. Since early in the Internet's history, many company intranets were separated from the Internet at the circuit level. Figure 9.7 shows a typical configuration.

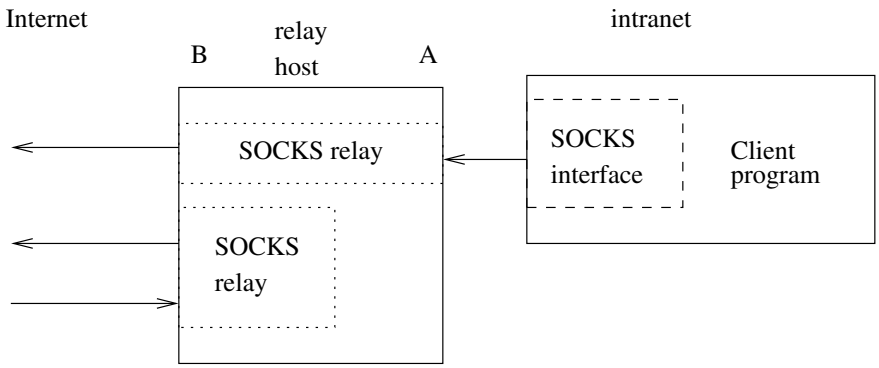

Figure 9.7: A typical SOCKS connection through interface A, and a rogue connection through the external interface, B.

In some cases, a circuit connection is made automatically, as part of the gateway architecture. A particular TCP service might be relayed from an external host to an internal database machine. The Internet offers many versions of simple programs to perform this function: look for names such as "*tcprelay*."

In other cases, the connection service needs to be told the desired destination. In this case, there is a little protocol between the caller and the gateway. This protocol describes the desired destination and service, and the gateway returns error information if appropriate. The first such service was described in [Cheswick, 1990] and was based on work by Howard Trickey and Dave Presotto. David and Michelle Koblas [1992] implemented SOCKS, which is now widely deployed. Most important Internet clients know the SOCKS protocol and can be configured to use SOCKS relay hosts.

In general, these relay services do not examine the bytes as they flow through. They may log the number of bytes and the TCP destination, and these logs can be useful. For example, we recently heard of a popular external site that had been penetrated. The Bad Guys had been collecting passwords for over a month. If any of our users used these systems, we could warn them. A quick *grep* through the logs spotted a single unfortunate (and grateful) user.

Any general circuit gateway (including SOCKS) is going to involve the gateway machine listening on some port, to implement FTP data connections. There is a subtle problem with the notion of a circuit gateway: Uncooperative inside users can easily subvert the intent of the gateway designer by advertising unauthorized services. It is unlikely that, for instance, port 25 could be used that way, as the gateway machine is probably using it for its own incoming mail processing, but there are other dangers. What about an unprotected *telnet* service on a nonstandard port? An NFS server? A multiplayer game? Logging can catch some of these abuses, but probably not all. It's wise to combine the circuit gateway with a packet filter that blocks many inbound ports.

Circuit gateways by design launder IP connections: The source IP address is not available to the server. Relay requests are expected to arrive as shown at interface A in Figure 9.7. If the

service is also provided on interface B, external users can launder connections through this host. There are hacking tools used to scan for open relay servers.

Clearly, some controls are necessary. These can take various forms, including a time limit on how long such ports will last (and a delay before they may be reused), a requirement for a list of permissible outside callers to the port, and even user authentication on the setup request from the inside client. Obviously, the exact criteria depend on your stance.

Application and circuit gateways are well suited for some UDP applications. The client programs must be modified to create a virtual circuit to some sort of proxy process; the existence of the circuit provides sufficient context to allow secure passage through the filters. The actual destination and source addresses are sent in-line. However, services that require specific local port numbers are still problematic.

9.4 Dynamic Packet Filters

Dynamic packet filters are the most common sort of firewall technology. They are for folks who want everything: good protection and full transparency. The intent is to permit virtually all client software to operate, without change, while still giving network administrators full control over traffic.

At one level, a dynamic packet filter behaves like an ordinary packet filter. Transit packets are examined; if they satisfy the usual sort of criteria, such as acceptable port numbers or addresses, they're allowed to pass through. But one more thing is done: note is made of the identity of outgoing packets, and incoming packets for the same connection are also allowed to pass through. That is, the semantics of a *connection* are captured, without any reliance on the syntax of the header. It is thus possible to handle UDP as well as TCP, despite the former's lack of an `ACK` bit.

As noted earlier, ordinary packet filters have other limitations as well. Some dynamic packet filters have additional features to deal with these.

The most glaring issue is the data channel used by FTP. It is impossible to handle this transparently without application-specific knowledge. Accordingly, connections to port 21—the FTP command channel—typically receive special treatment. The command stream is scanned; values from the `PORT` commands are used to update the filter table. The same could be done with `PASV` commands, if your packet filter restricts outgoing traffic.

Similar strategies are used for RPC, H.323, and the like. Examining the packet contents lets you regulate which internal (or external) RPC services can be invoked. In other words, we have moved out of the domain of packet filtering, and into *connection filtering*.

X11 remains problematic, as it is still a very dangerous service. If desired, though, application relays such as *xforward* [Treese and Wolman, 1993] can be replaced by a user interface to the filter's rule table. The risks of such an interface are obvious, of course; what is less obvious is that almost the same danger—that an ordinary user can permit certain incoming calls—may be present with *xforward* and the like. It is better to tunnel X11 through *ssh*.

9.4.1 Implementation Options

Conceptually, there are two primary ways to implement dynamic packet filters. The obvious way is to make changes on the fly to a conventional packet filter's ruleset. While some implementations

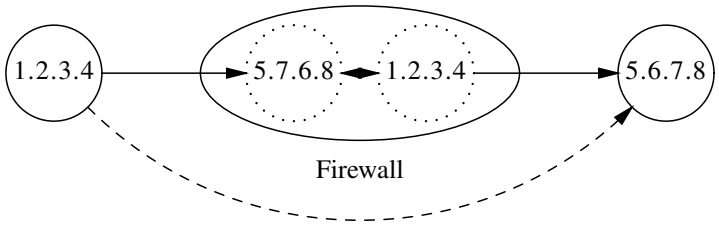

Figure 9.8: Redialing on a dynamic packet filter. The dashed arrow shows the intended connection; the solid arrows show the actual connections, to and from the relay in the firewall box. The firewall impersonates each endpoint to the other.

---

do this, we are not very happy about it. Packet filter rulesets are delicate things, and ordering matters a lot [Chapman, 1992]. It is not always clear which changes are benign and which are not.

There is another way to implement dynamic packet filters, though, one that should be equivalent while—in our opinion—offering greater *assurance* of security. Instead of touching the filter rule table, implement the dynamic aspects of the packet filter using circuit-like semantics, by terminating the connection on the firewall itself. The firewall then redials the call to the ultimate destination. Data is copied back and forth between the two calls.

To see how this works, recall that a TCP connection is characterized by the following 4-tuple:

$$\langle localhost, localport, remotehost, remoteport \rangle.$$

But *remotehost* isn't necessarily a particular machine; rather, it is any process that asserts that address. A dynamic packet filter of this design will respond as any host address at all, as far as the original caller can tell. When it dials out to the real destination, it can use the caller's IP address as its own. Again, it responds to an address not its own (see Figure 9.8). Connections are identified on the basis of not only the four standard parameters, but also network interface.

Several things fall neatly out of this design. For one thing, TCP connections just work; little or no special-case code is needed, except to copy the data (or rather, the pointers to the data) and the control flags from one endpoint to another. This is exactly the same code that would be used at application level. For another, changing the apparent host address of the source machine is a now a trivial operation; the redialed call simply has a different number in its connection control block. As we discuss in the following section, this ability is very important.

Application-level semantics, such as an FTP proxy, are also implementable very cleanly with this design. Instead of having a direct copy operation between the two internal sockets, the call from the inside is routed to a user-level daemon. This is written in exactly the same fashion as an ordinary network daemon, with one change: The local *address* of the server is wildcarded. When it calls out to the destination host, it can select which source address to use, its own or that of the original client. Figure 9.9 shows an application proxy with address renumbering.

UDP is handled in the same way as TCP, with one important exception: Because there is no in-band notion of a "close" operation in UDP, a timeout or some other heuristic, such as packet count, must be used to tear down the internal sockets.

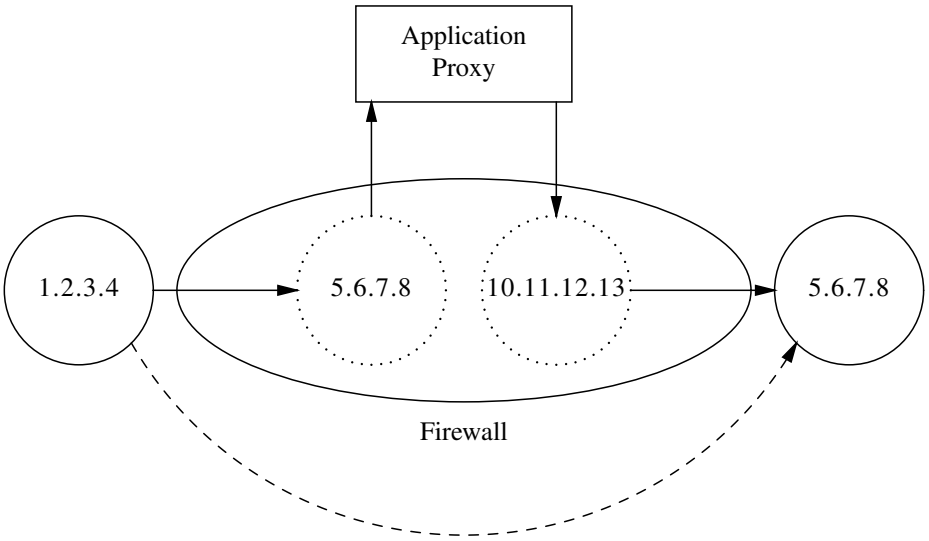

Figure 9.9: A dynamic packet filter with an application proxy. Note the change in source address.

Handling ICMP error packets is somewhat more complex; again, these are most easily processed by our dual connection model. If an ICMP packet arrives for some connection—and that is easily determinable by the usual mechanisms—a corresponding ICMP packet can be synthesized and sent back to the inside host. Non-error ICMP messages, notably `Echo Request` and `Echo Reply` packets, can be handled by setting up pseudoconnections, as is done for UDP.

We can now specify the precise behavior of a dynamic packet filter. When a packet arrives on an interface, the following per-interface tables are consulted, in order:

1. The active connection table. This points to a socket structure, which in turn implicitly indicates whether the data is to be copied to an output socket or sent to an application proxy.

2. An ordinary filter table, which can specify that the packet may pass (or perhaps be dropped) without creating local state. Some dynamic packet filters will omit this table; its existence is primarily an efficiency mechanism, as the rulesets can permit connections to be established in either direction.

3. The dynamic table, which forces the creation of the local socket structures. This table may have "drop" entries as well, in order to implement the usual search-order semantics of any address-based filter.

If the second table is null, the semantics—and most of the implementation—of this style of firewall are identical to that of a circuit or application gateway.

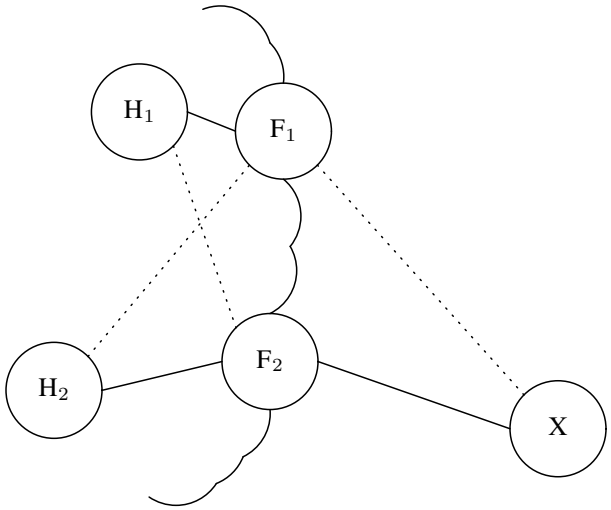

Figure 9.10: Asymmetric routes with two dynamic packet filters. Distance on the drawing is intended to be proportional to distance according to the routing protocol metrics. The solid lines show actual routes; the dotted lines show rejected routes, based on these metrics.

9.4.2 Replication and Topology

With traditional sorts of firewalls, it doesn't matter if more than one firewall is used between a pair of networks. Packet filters are stateless; with traditional circuit or application relays, the client has opened an explicit connection to the firewall, so that all conversations will pass through the same point.

Dynamic packet filters behave differently. By design, clients don't know of their existence. Instead, the boxes capture packets that happen to pass through them courtesy of the routing protocols. If the routes are asymmetric, and inbound and outbound packets pass through different boxes, one filter box will not know of conversations set up by the other. This will cause reply packets to be rejected, and the conversation to fail.

Can we avoid these asymmetric routes? Unfortunately not; in one very common case, they will be the norm, not the exception.

Consider the network topology shown in Figure 9.10, where the outside network is the Internet. In general, border routers connecting to the Internet do not (and cannot) transmit knowledge of the full Internet topology to the inside; instead, they advertise a default route. If the two firewall boxes each advertise `default`, outbound packets will go to the nearest exit point. In this case, all packets from host H_1 will leave via dynamic packet filter F_1, while those from H_2 will leave via F_2.

The problem is that the outside world knows nothing of the topology of the inside. In general, F_1 and F_2 will both claim equal-cost routes to all inside hosts, so replies will transit the firewall closest to the *outside* machine. Thus, if H_1 calls X, the outbound packets will traverse F_1, whereas the replies will pass through F_2.

Several solutions suggest themselves immediately. The first, of course, is to maintain full knowledge of the topology on both sides of the firewall, to eliminate the asymmetric routes. That doesn't work. There are too many nets on the Internet as it is; the infrastructure cannot absorb that many extra routes. Indeed, the current trend is to do more and more address aggregation, to try to stave off the table size death of the net [Fuller *et al.*, 1993]. Anyone who proposed the opposite would surely be assaulted by router vendors and network operators (though perhaps cheered on by memory manufacturers).

The opposite tack—making sure that all internal hosts have full knowledge of the Internet's topology—is conceivable, though not feasible. Only the biggest routers currently made can handle the full Internet routing tables; to deploy such monsters throughout internal nets is economically impossible for most organizations. But it won't solve the problem—the same sort of "hot potato" routing is used between ISPs, and users have no control over that.

Note, though, that full knowledge of a company's own topology is generally feasible for internal (i.e., intranet) firewalls. In such cases, the "stateful" (a horrible neologism meaning "the opposite of stateless") nature of dynamic packet filters is not a major problem.

A second general strategy for Internet connectivity is to have the multiple firewalls share state information. That is, when a connection is set up through F_1, it would inform F_2. An alternative approach would be "lazy sharing": Only check with your peers before dropping a packet or when tearing down a connection whose state was shared.

Although in principle this scheme could work (see point 3 of Section 2 of [Callon, 1996]), we are somewhat dubious. For one thing, the volume of messages may be prohibitive. Most TCP sessions are about 20 packets long [Feldmann *et al.*, 1998]. The closer a dynamic packet filter's implementation is to our idealized model, the more state must be communicated, including sequence number updates for every transit packet. This is especially true for the application proxies. For another, this sort of scheme requires even more complex code than an ordinary dynamic packet filter, and code complexity is our main reservation about such schemes in the first place. (It goes without saying, of course, that any such update messages must be cryptographically authenticated.) There is also the threat of sophisticated enemies sending packets by variant paths, to evade intrusion detection systems or to confuse the sequence numbering. This concern aside, we expect some vendors to implement such a scheme, possibly built on some sort of secure reliable multicast protocol [Reiter, 1994, 1995].

Does replication matter? It helps preserve individual TCP sessions, but most are restarted without much trouble—users click on Web links again, and mailers retry the mail transmission. VPN tunnels, which can be quite long-lived, can be restarted without any effect on the higher-level connections if restoration is fast enough. Many of the longest connections on the backbones are now peer-to-peer file transfers. These tend to be music and movie files, and are generally not vital, and may violate your security policy (or applicable laws) in any event.

For most situations, though, the best answer may be to use the address translation technique we described earlier. As before, outbound packets will pass through the gateway nearest the inside host. However, the connection from there will appear to be from the gateway machine itself, rather than from any inside machine, so packets will flow back to it. This may be suboptimal from a performance perspective, but it is simple and reliable.

What is the alternative? Install a single, reliable piece of hardware, protected by a good *uninterruptible power supply (UPS)*. Equipment should run for months without rebooting. Keep a second firewall on standby, if desired, for use if the first catches fire. At this level of reliability, Internet problems will be the major cause of outages by far.

9.4.3 The Safety of Dynamic Packet Filters

Dynamic packet filters promise to be all things to all people. They are transparent in the way packet filters are, but they don't suffer from stateless semantics or interactions between rulesets. Are they safe?

Our answer is a qualified yes. The major problem, as always, is complexity. If the implementation strategy is simple enough—which is not easy to evaluate for a typical commercial product—then the safety should be comparable to that of circuit gateways. The more shortcuts that are taken from our dual connection model, especially in the holy name of efficiency, the less happy we are.

A lot of attention must be paid to the administrative interface, the way rules—the legal connections—are configured. Although dynamic packet filters do not suffer from ruleset interactions in the way that ordinary packet filters do, there are still complicated order dependencies. Administrative interfaces that use the physical network ports as the highest-level construct are the safest, as legal connections are generally defined in terms of the physical topology.

There's one more point to consider. If your threat model includes the chance of evildoers (or evil software) on the inside trying to abuse your Internet connection, you may want to avoid dynamic packet filters. After all, they're *transparent*—ordinary TCP connections, such as the kind created by some e-mail worms, will just work. A circuit or application gateway, and in particular one that demands user authentication for outbound traffic, is much more resistant to this threat.

9.5 Distributed Firewalls

The newest form of firewall, and one not available yet in all its glory, is the *distributed firewall* [Bellovin, 1999]. With distributed firewalls, each individual host enforces the security policy; however, the policy itself is set by a central management node. Thus, rather than have a separate box on the edge of the network reject all inbound packets to port 80, a rule to reject such connection attempts is created by the administrator and shipped out to every host within its management domain. The advantages of a scheme like this are many, including the lack of a central point of failure and the ability to protect machines that aren't inside a topologically isolated space. Laptops used by road warriors are the canonical example; telecommuters' machines are another. A number of commercial products behave in approximately this fashion; it is also easy to roll your own, if you combine a high-level policy specification such as Firmato [Bartal *et al.*, 1999] with any sort of file distribution mechanism such as *rsync* or Microsoft's *Server Management System (SMS)*.

The scheme outlined here has one major disadvantage. Although it is easy to *block* things securely, it is much harder to allow in certain services selectively. Simply saying

action	ourhost	port	theirhost	port	comment
allow	(here)	25	10.2.42.0/24	*	*connection to our SMTP port*

is safe if and only if you know that the Bad Guys can't impersonate addresses on the source network, 10.2.42.0/24. If you have a router that performs anti-spoofing protection, you're reasonably safe while you're inside the protected enclave. But imposing that restriction loses one of the benefits of distributed firewalls: the ability to roam safely.

The solution is to use IPsec to identify trusted peers. The proper rule would say something like the following:

action	ourhost	port	theirhost	port	comment
allow	(here)	25	cert=*.MYMEGACORP.COM	*	

In other words, a machine is trusted if and only if it can perform the proper cryptographic authentication; its IP address is irrelevant.

9.6 What Firewalls Cannot Do

> [Product...] has been shown to be an effective decay-preventive dentifrice that can be of significant value when used as directed in a conscientiously applied program of oral hygiene and regular professional care.
>
> *American Dental Association*
> —COUNCIL ON SCIENTIFIC AFFAIRS

Although firewalls are a useful part of a network security program, they are not a panacea. When managed properly, they are useful, but they will not do everything. If firewalls are used improperly, the only thing they buy you is a false sense of security.

Firewalls are useless against attacks from the inside. An inside attack can be from a legitimate user who has turned to the dark side, or from someone who has obtained access to an internal machine by other means. Malicious code that executes on an internal machine, perhaps having arrived via an e-mail virus or by exploiting a buffer overflow on the machine, can also be viewed as an inside attacker.

Some organizations have more serious insider threat models than others. Some banks have full-fledged internal forensics departments because, after all, as Willie Sutton did not say (but is often quoted as saying), "that's where the money is." These organizations, with serious insider risk, often monitor their internal networks very carefully, and take apart peoples' machines when they suspect anything at all. They look to see what evil these people did. Military organizations have big insider risks as well. (There are oft-quoted statistics on what percentage of attacks come from the inside. The methodology behind these surveys is so bad that we don't believe any of the numbers. However, we're sure that they represent very significant threats.)

If your firewall is your sole security mechanism, and someone gets in by some other mechanism, you're in trouble. For example, if you do virus scanning only at the e-mail gateway, security

can be breached if someone brings in an infected floppy disk or downloads an executable from the Web. Any back door connection that circumvents the gateway filtering can demonstrate the limited effectiveness of firewalls. Problems processing MIME, such as buffer overflows, have led to security problems that are outside the scope of what firewalls are designed to handle.

The notion of a hard, crunchy exterior with a soft, chewy interior [Cheswick, 1990], only provides security if there is no way to get to the interior. Today, that may be unrealistic.

Insider noncooperation is a special case of the insider attack, but fundamentally, it is a people problem. We quote Ranum's Law in Chapter 10: "You can't solve people problems with software." As stated above, it is easy for users who do not want to cooperate to set up tunnels, such as IP over HTTP. IP filtering at the lower IP layer is useless at that point.

Firewalls act at some layer of the protocol stack, which means that they are not looking at anything at higher layers. If you're doing port number filtering only at the transport layer, you'll miss SMTP-level problems. If you filter SMTP, you might miss data-driven problems in mail headers; if you look at headers, you might miss viruses and Trojan horses. It is important to assess the risks of threats at each layer and to act accordingly. There are trade-offs. Higher-layer filtering is more intrusive, slower to process, and less comprehensive, because there are so many more processing options for each packet as you move up the stack.

E-mail virus scanning seems to be a win for Windows sites. If nothing else, throwing away all the virus-laden e-mail at the gateway can save a lot of bandwidth. (But a good strategy is to run one brand of virus scanner at the gateway, and another on the desktops. AV software isn't perfect.) Conversely, trying to scan FTP downloads isn't worthwhile at most sites. Data transformations, such as compression, make the task virtually impossible, especially at line speed. Deciding where to filter and how much is a question of how to balance risk versus costs. There is always a higher layer, including humans who carry out stupid instructions in e-mail. It is not easy to filter those.

Another firewall problem is that of transitive trust. You have it whether you like it or not. If A trusts B through its firewall, and B trusts C, then A trusts C, whether it wants to or not (and whether it knows it or not).

Finally, firewalls may have errors, or not work as expected. The best administration can do nothing to counter a firewall that does not operate as advertised.