10

Filtering Services

The decision about what services to filter is based on a desired policy. Nonetheless, some general rules are prudent for most policies. In this chapter, we present our philosophy about these. They are not to be viewed as hard-and-fast rules, but rather as suggestions, or perhaps as a template policy to be customized. This chapter discusses *what* to filter and *why*. The *how* is covered in Chapter 11. The astute reader will note that the services discussed here are a small subset of the ones from Chapter 2. Rather than discuss every possible service, we focus on the more interesting ones, with an eye toward pedagogy.

In this chapter, when we describe a service, we include a summary about how to handle it from a security point of view. It looks something like the following:

protocol	**out**	**in**	**comment**
PROT	x	y	*optional comment*

In this table, legal values for *x* and *y* are as follows:

allow	let it through
block	don't let it through
filter	an application-level proxy should make the decision
tunnel	block the port for PROT, but allow users to tunnel it with a more secure protocol

The *out* column refers to the decision about outbound traffic for port PROT. For TCP packets, "outbound" is straightforward; it refers to connections initiated from the inside. "Inbound" refers to connections initiated from the outside.

The meaning is less clear for UDP, because the protocol itself is connectionless. Furthermore, some of the protocols of interest are *not* simple query/response services. For query/response services, we thus speak of an "inbound query," which elicits an "outbound response"; similarly, "outbound queries" elicit "inbound responses." For protocols that do not fit this model, we can speak only of inbound and outbound packets.

10.1 Reasonable Services to Filter

10.1.1 DNS

DNS represents a dilemma for the network administrator. We need information from the outside, but we don't trust the outside. Thus, when we get host name-to-IP address mappings from the outside, it is best not to base any security-related decisions on them. To be more precise, we absolutely must not trust such information for internal purposes, though we may have to rely on it for something like sending sensitive e-mail to external partners.

This has some consequences. Although under some circumstances it might be okay to do name-based authentication for internal machines, it is never acceptable for external machines. We must also ensure that no other internal-to-internal trust relationship depends on any information learned from the outside.

The basic threat is simple: Outsiders *can* contaminate the DNS cache, notably by including extraneous information in their responses. The details are explained in [Bellovin, 1995]. The rules for outbound DNS queries can be summarized as follows:

protocol	outbound query	inbound response	comment
DNS	allow	filter	*block internal info*

The best way to filter DNS is to use a DNS proxy that does two things [Cheswick and Bellovin, 1996]. First, it redirects queries for internal information to internal DNS servers. Second, it censors inbound responses to ensure that no putatively internal information is returned. This is most likely to occur in the Additional Information or Authoritative Server sections of the response, but could occur anywhere. Nevertheless, one simple rule covers all cases: If it was not in the request, we do not want to know it. (Note that a query for internal information will never be sent to external servers, and hence should never be returned in response to our query.)

Inbound queries are simpler: Put your DNS server in the DMZ. For that matter, you can (and often should) out-source it;[1] as a matter of operational correctness, you should have at least two DNS servers for each zone, and they should be as far apart as possible [Elz *et al.*, 1997]. Do you operate your own machines in widely separated parts of the Internet?

You should be especially certain that you don't have them all on the same LAN. (There are security reasons, too—what if someone DDoS's your link? Make them work harder!) The rules are thus quite simple:

protocol	outbound response	inbound query	comment
DNS	allow	DMZ	

Dealing with the DNS is one of the more difficult problems in setting up a firewall, especially if you use a simple packet filter. It is utterly vital that the gateway machine use it, but it poses many risks.

1. Some people don't believe in out-sourcing such things. We're tempted to ask if they run their own fiber, too. Your ISP—with whom you have a business and contractual relationship—can do far worse things by playing with your traffic than by playing with your DNS. To be sure, you may want to run the primary server yourself, if only for ease of updates, and the advent of DNSsec will make that more necessary.

```
fleeble.com.       IN      SOA     foo.fleeble.com. root.foo.fleeble.com. (
                           200204011 ;serial
                           3600     ;refresh
                           900      ;retry
                           604800   ;expire
                           86400 )  ;minim

fleeble.com.       IN      NS      foo.fleeble.com.
fleeble.com.       IN      NS      x.trusted.edu.
foo.fleeble.com.   IN      A       200.2.3.4

foo.fleeble.com.   IN      MX      0 foo.fleeble.com.
*.fleeble.com.     IN      MX      0 foo.fleeble.com.
fleeble.com.       IN      MX      0 foo.fleeble.com.

ftp.fleeble.com.   IN      CNAME   foo.fleeble.com.
```

Figure 10.1: A minimal DNS zone. The inverse mapping tree is similarly small. Note the use of an alias for the FTP server. The secondary server (X.TRUSTED.EDU) is a sensitive site; any hacker who corrupted it, perhaps via a site that it trusts, could capture much of your inbound mail and intercept many incoming *ssh* calls. Note also that we do not give X's IP address; that must reside in the TRUSTED.EDU zone.

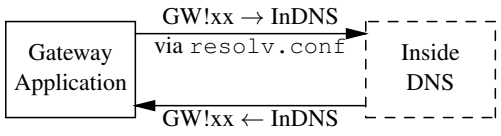
What tack you take depends on the nature of your firewall. If you run a circuit or application gateway, there is no need to use the external DNS internally. The information you advertise to the outside world can be minimal (see Figure 10.1). It lists the name server machines themselves (FOO.FLEEBLE.COM and X.TRUSTED.EDU), the FTP and mail relay machine (FOO.FLEEBLE.COM again), and it says that all mail for any host in the FLEEBLE.COM domain should be routed to the relay.

Of course, the inside machines can use the DNS if you choose; this depends on the number of hosts and system administrators you have. If you do, you must run an isolated internal DNS with its own pseudo-root. We have done that, but we were careful to follow all of the necessary conventions for the "real" DNS. It is possible to live internally with static host tables, but the details vary a lot; every operating system is different. Even the location of the `hosts` file can change. It's usually `/etc/hosts` on UNIX systems, but it can be `\windows\hosts`, `\winnt\hosts`, `\windows\drivers\etc\hosts`, and so on, on various Microsoft platforms.

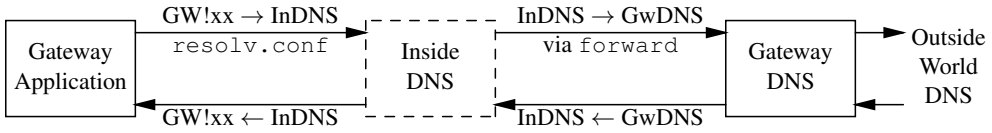
At one level, dynamic packet filters can handle DNS as properly as they can any other UDP-based protocol. But application-level filtering is necessary to deal with the attack mentioned above.

Inside hosts need to use the DNS to reach outside sites. In some messages to the Firewalls mailing list, Chapman has described a scheme that works today because of the way most UNIX system name servers happen to be implemented. But it is not guaranteed to work with all systems.

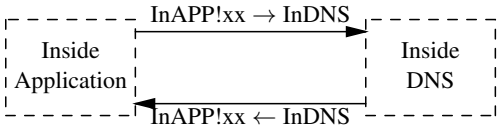
His approach (see Figure 10.2) is to run name servers for the domain on both the gateway machine and on some inside machine. The latter has the real information; the gateway's name server has the sort of minimal file shown in Figure 10.1. Thus, outside machines have no access to sensitive internal information.



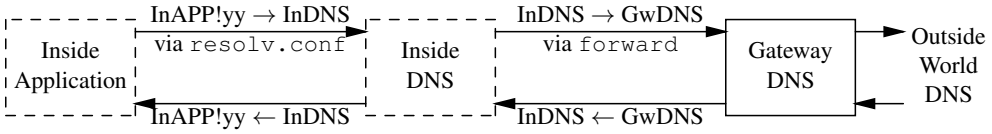
(a) Gateway application calling inside machine



(b) Gateway application calling outside machine



(c) Inside application calling inside machine



(d) Inside application calling outside machine

Figure 10.2: Passing DNS through a packet filter. The packet filter separates the gateway machine GW from the inside machines; the latter are always shown as dashed boxes. Note that all incoming packets through the firewall—that is, all arrows from solid boxes to dashed ones—are from GW to the inside DNS server INDNS, which lives on a fixed port. The query always starts out in the left-most box; in scenario (b), the query goes back out through the firewall, as noted in the text.

The tricky parts are as follows:

1. Permitting the gateway itself to resolve internal names (for mail delivery, for example)
2. Permitting inside machines to resolve external names
3. Providing a way for the necessary UDP packets to cross the firewall

The first part is handled by creating a `/etc/resolv.conf` file on the gateway that points to the internal DNS server. That file tells *application programs* on the gateway, but not the name server itself, where to go to resolve queries. Thus, for example, whenever *mail* wants to find an IP address, it will ask the inside server.

Name server processes pay no attention to `/etc/resolv.conf` files. They simply use the tree-structured namespace and their knowledge of the root name servers to process all requests. Queries for names they do not know are thus properly resolved.

The second problem involves queries for external names sent to the internal name server. Of course, this server doesn't know about outside machines. Rather than talk to the real servers directly (we cannot permit that, because we can't get the replies through the firewall safely), the inside server has a `forwarder` entry pointing to the gateway in its configuration file. This line denotes which server should be queried for any names not known locally. Thus, if asked about an inside machine, it responds directly; if asked about an outside machine, it passes the query to the gateway's name server.

Note the curious path taken by a request for an outside name by a process running on the gateway machine. It first goes to the inside server, which *can't* know the answer unless it's cached. It then hops back across the firewall to the outside machine's own server, and thence eventually to the distant DNS server that really knows the answer. The reply travels the same twisty path.

The reason that the inside and outside servers can talk through the packet filter is that DNS servers use a constant port number when sending their queries. On older versions, it's port 53; newer ones let you configure the port number. This solves the third problem.

One "ı" has been left undotted. If an inside machine opens a connection to some external site, that site will probably want to look up its host name. The gateway's DNS server does not have that information, however, and this sort of failure will cause many sites to reject the connection. For example, a number of FTP sites require that the caller's IP address be listed in the DNS. Chapman suggests using a wildcard `PTR` record:

```
*.3.2.127.in-addr.arpa.    IN   PTR    UNKNOWN.fleeble.com.
```

which will at least offer some answer to the query. But if the external site performs a DNS cross-check, as described in Section 2.2.2, it will fail. Again, many outside sites will reject connections if this occurs. UNKNOWN.FLEEBLE.COM has no IP addresses corresponding to the actual inside machine's address. To deal with that, a more complete fiction is necessary. One suggestion we've heard is to return a special-format host name for any address in your domain:

```
42.3.2.127.in-addr.arpa.  IN  PTR  pseudo-127-2-3-42.fleeble.com.
```

When a query is made for an `A` record for names of this form, the appropriate record can be synthesized. (Note that underscores are illegal characters in domain names, though many people use them.)

10.1.2 Web

Unless you want a revolution on your hands, allow outbound HTTP queries. At the same time, it is a good idea to use proxy filtering to scan for hostile applets and viruses. Depending on your security policies, you may want to block some ActiveX controls as well [Bellovin *et al.*, 2000]. However, note that scanning for viruses at the firewall can be quite challenging [Martin *et al.*, 1997]. Do not place these filters in a place that breaks caching.

The firewall should not allow incoming HTTP traffic, except to your official Web servers. Of course, your Web servers should be in the DMZ. Packets to port 80 on an internal machine should be tossed out. These days, most of them are generated automatically by worms seeking new targets. The rule is as follows:

protocol	out	in	comment
Web	allow	block	*Put Web server in DMZ*

An alternative ruleset, if you require insiders to use an internal Web proxy, is to permit only it to talk directly to the world. In this case, the rule looks as follows:

protocol	out	in	comment
Web	filter	block	*Put Web server in DMZ*

You should probably treat port 443 the same way as port 80.

10.1.3 FTP

FTP is a tricky protocol. Because by default FTP uses `PORT` mode, which requires a separate, incoming connection, many stateful firewalls open a hole allowing incoming connections to an internal machine. This has been shown to be perilous [Martin *et al.*, 1997]. A better idea is to require `PASV` FTP for outbound connections [Bellovin, 1994]. Most browsers run in passive mode (though some require that an option be set), so this should not be a problem. Do not allow inbound FTP connections, and place the FTP server in the DMZ. The rule is as follows:

protocol	out	in	comment
FTP	passive	block	*Put FTP server in DMZ*

In order to handle `PORT` mode, even dynamic packet filters need an application proxy. Some of them try to get away with looking at just one packet at a time, rather than reassembling the TCP stream. The technical term for this behavior is "a very bad idea." Looking at single packets can break things, if the sender has split data across multiple packets. There have even been reports of exploitable vulnerabilities in such setups.[2]

10.1.4 TCP

Is it a good idea to allow incoming and outgoing TCP connections? As a general rule, you have to trust insiders. If you cannot trust them, then you have a people problem, which is much more serious than a networking problem. To quote Ranum's Law, "You can't solve people problems with software."

2. See `http://www.kb.cert.org/vuls/id/328867`.

Because insiders are trusted, is it okay to allow outgoing TCP connections? Not completely. Although the insiders might be trusted, it is not always certain that the code they are running is behaving properly. Applets running on users' machines are considered insiders. Signed applets can be granted privileges by naïve users; these allow the applets to talk to the file system and connect to arbitrary places on the network. (Many organizations train their users to click "OK" to use payroll and other systems.) The TCP connections originating from these applets come from the inside.

There are other ways that bad things can originate from the inside. Assume that the mail filter is weeding out viruses and worms. That only works if users obtain their mail via POP3 or IMAP. If mail is read through a Web-based server, such as Hotmail or Hushmail, there is little to prevent the poor user from infection via these vectors. Once hit, the inside machine may generate problematic outgoing TCP connections. (Imagine a dual-mode worm: When it can, it spreads by direct attacks on vulnerable systems, but it also e-mails copies of itself to users behind firewalls. Your imagination won't be stretched very far; these worms exist.)

We don't really know what to do about this. Disallowing outgoing TCP is Draconian, and represents a restriction that is probably too strong. Conversely, highly sensitive government sites may have confidentiality requirements on their data that justify such a policy. The rest of us can probably live with the risk. Besides, clever malware can exploit application-level proxies in the same way.

Incoming TCP connections should not be allowed. If there is a strong need for access to an internal machine from the outside, this should be handled via a dedicated proxy, often from a machine on the DMZ. If possible, use cryptographically enhanced services such as *ssh*. It is also best to limit the sets of machines that can be reached; and, if possible, the set of machines that can initiate access. The filtering rule for TCP can be summarized as follows:

protocol	out	in	comment
TCP	allow	block	*Generally trust insiders*

10.1.5 NTP

There are now cheap, extremely accurate time devices available based on the Global Positioning System and other radio sources. If these are not used, there are time sources on the Internet. You should limit access to selected, trusted external servers.

If you have a close relationship with the outside time server, you may want to use NTP's built-in authentication mechanisms. It is also common to run an external NTP server of your own and use the firewall to restrict insiders' access to that server alone.

protocol	out	in	comment
NTP	allow	allow	*Specific hosts only*

Note that NTP is not a query/response protocol.

10.1.6 SMTP/Mail

There are two common reasons to restrict outbound SMTP traffic. In the old days of the Internet, badly formatted e-mail messages were common, and an outgoing filter could clean up or reject

incorrect message formats. You may also wish to check outgoing mail for viruses, strange attachments, or even corporate secrets. An alarm for a virus in outgoing mail may be your first clue that a virus is running around your intranet. Mail programs have been notorious for security problems, so be sure to keep up with the latest security alerts and patches for your mail software. Scan for viruses and perhaps other active content, and filter or discard attachments. (If you do the latter, you may want to also build a moat around your house and office. Moat monsters are optional.)

Some organizations try to scan outbound mail for secrets and *dirty words*, a military term for phrases that secret texts are likely to contain. This is a difficult proposition at best; apart from Ranum's Law considerations, there is the whole problem of natural language recognition. Unless you work for a company that is legally required to do such things (some U.S. brokerage firms fall into this category)—or live in a country that "needs" to do such things—it's probably not worth trying.

ISPs have another reason to block outgoing SMTP service, even if they block nothing else. Spammers find open hosts ("open relays") or use dial-up access and send thousands of unwanted e-mail messages from them. Proactive ISPs suppress this activity by blocking outgoing SMTP service. This is a reasonable policy for services that have messy user populations. Of course, legitimate users may be blocked from accessing their home SMTP servers. They could use a tunnel, SMTP AUTH (see Section 3.1.1), or "SMTP after POP" (see Section 3.1.3).

If none of these issues is a concern, then outbound SMTP can be allowed, unfiltered. The rule is as follows:

protocol	out	in	comment
SMTP	allow	filter	

10.1.7 POP3/IMAP

Inbound POP3 and IMAP are used by outsiders attempting to get mail that is on the inside. These protocols should be blocked. There are probably passwords flowing in the clear; there is almost certainly sensitive internal content that shouldn't be exposed to prying eyes. Even the APOP protocol, which uses challenge/response, is vulnerable to dictionary attacks. If you want to provide mail access to the outside, do it with a tunnel; most mail clients and servers now support these protocols over SSL. But even this permits online password-guessing attacks.

Should internal users be allowed to access external POP3/IMAP servers? From a security standpoint, this is not a great idea. In addition to the password exposure problem, you have to worry about malicious content. Sure, users can then tunnel around you using *ssh*, but if the policy forbids external e-mail access, then those are misbehaving users who can be dealt with in other ways. If you do decide to allow queries to external POP3/IMAP servers, do it through an application-level proxy that scans for viruses, worms, and other executables. (Add a spam filter, too, as an incentive to use it.) The rule looks as follows:

protocol	out	in	comment
POP3/IMAP	filter	tunnel	*Block active content*

Attachments: Can't Live With 'Em, Can't Live Without 'Em

It used to be that typical e-mail contained a two-line ASCII sentence, *e.g.,* "The meeting has been moved to 2:30." E-mail now usually contains attachments, specially formatted files glued into the message.

Unless you are one of the few people who has a life that does not involve interaction with people who use Windows, you probably have to handle attachments. An attachment used to mean some kind of a romantic relationship with another human being. Today, it is a MIME-encoded thing that is often associated with some Microsoft Office application; at the very least, it's the same text in both ASCII and HTML, the latter adorned with embedded images (and Web bugs) as well.

The bloat aside—that same one-line e-mail message is 19 KB as a Word file—there are security implications as well. These Office applications can contain embedded programs; such programs are prominent vectors for worms and viruses. (Besides, the file formats themselves can leak information. When using UNIX tools to view Word files, we've seen not just information that the sender had thought was deleted, but the contents of *other* documents that were open at the same time!)

There is also a mismatch between MIME semantics and those of some operating systems, *i.e.*, Windows. Here are some MIME headers embedded in a copy of Klez some worm thoughtfully sent us:

```
Content-Type: audio/x-wav;
   name=EASYvolume[1].exe
Content-Transfer-Encoding: base64
```

The `Content-Type` field implies what application *should* be used to process the data, presumably some sort of audio program in this case, but Windows uses the filename—and thus treats the attached data as an executable program and runs it. This is bad.

Attachments themselves are not evil—family pictures and PGP messages are sent as attachments—but the stuff some people attach to messages these days is terrible. A large financial company once monitored all attachments coming from outside of their intranet for a week. They found that not one had a business purpose, so they instituted a company policy that discarded all incoming attachments. As a result, when the Melissa worm struck, they were largely unaffected. The policy, while Draconian, may not be as unreasonable as it seems. At the very least, an "Evil Stuff" check should be made, with "evil" defined as "anything not on the 'Approved' list." Then, if you can get away with it, exclude *all* executable content.

Attachments are here to stay, and they're a good way to e-mail non-ASCII files when you need to. They are the way the world does business. You can't live with them; you can't live without them.

10.1.8 *ssh*

One of the principles of computer security is to trust as little as possible. *Ssh* is one of the things we trust. As with Mail, it is thus crucial to keep up with bugs and patches. *Ssh* has indeed had some serious security problems in the past. *Ssh* is reasonable to allow through the firewall because it implements cryptographic authentication and encryption, and is the best way we know of to allow access through a firewall.

Depending on your internal trust policies, you may want to terminate incoming *ssh* connections at the firewall. Here you can do strong, centralized authentication. It's also attractive to pretend that doing so prevents people or malicious programs from creating back doors, but it's just that: a pretense. If you permit outbound TCP, it's easy to create back doors, and *ssh*'s port-forwarding just lets Bad Guys do it a bit more easily, from the command line. The rule for *ssh* is as follows:

protocol	out	in	comment
ssh	allow	allow	*Stay current on patches*

10.2 Digging for Worms

E-mail isn't the only way that viruses and worms spread, but it's one of the most common. If your user population runs susceptible software (i.e., Windows), you really need to filter incoming e-mail. If you want to be a good citizen of the Net, you'll filter outgoing e-mail, too.

One approach, of course, is to screen each piece of incoming mail on each desktop. That's a good idea, even if you adopt other measures as well; defense in depth generally pays off. But desktops are often behind in their updates, and getting new pattern files to them *now* can be difficult.

Fortunately, it's not hard to install a centralized filter for malware. Use MX records to ensure that all inbound e-mail goes to a central place. Make sure that you include a wildcard MX record, too, for both your inside and your outside DNS:

```
example.com.        IN MX      10 mail-gw.example.com
*.example.com.      IN MX      10 mail-gw.example.com
```

It's a good idea to use a different brand of virus scanner for your gateway than for your desktop; all virus scanners are subject to false negatives. Many goods ones are out there, both commercial and open source. If you can, obtain your central scanner from the vendor who delivers new patterns rapidly during times of plague and helminthiasis [Reynolds, 1989].

In some cases, you may want to add your own patterns. There are some legal worms—spam, actually—but "legal" because the users consented to their spread by not decrypting the legalese in the license. Antivirus companies have been hesitant to block them, given that they are, technically, legal, but you're under no obligation to allow them inside your organization.

Outgoing e-mail should be scanned, too. There's no convenient analog to MX records; if you can't rely on your users to configure their mailers correctly, you can "encourage" them by blocking outbound connections to TCP port 25. That will also help guard against worms that do their own

SMTP. If you run a DNS proxy of some sort, you can configure it to make your outbound mail gateway the MX server for the entire Internet:

```
*.                      IN MX      10 mail-gw.example.com
```

Just make sure that you filter out any more-specific inbound records.

Some antivirus software annoys as much as it protects. A number of packages, if they detect a virus on a piece of incoming e-mail, will send an alert to the sender and all other recipients of that piece of e-mail. It seems civic-minded enough, but isn't as big a help as it appears. For one thing, many worms used forged sender addresses; notifying the putative sender does no good whatsoever. Moreover, notifying other recipients has bad scaling properties when one of the addressees is a mass mailing list.

A more dangerous form of annoyance is the trailer that reads something like this:

> *This piece of e-mail has been scanned, X-rayed, and screened for excessive nitrogenous compounds by ASCIIphage 2.71827, and is warranted to be free of viruses, worms, arthropods, and cyclotrimethylenetrinitramine. It is safe for consumption by humans and computers.*

A trailer like that is about equivalent to naming a file "`This is not a virus.exe`," and teaches users bad habits.

10.3 Services We Don't Like

10.3.1 UDP

Filtering TCP circuits is difficult. Filtering UDP packets while still retaining desired functionality is all but impossible. The reason lies in the essential difference between TCP and UDP: The former is a virtual circuit protocol, and as such has retained context; the latter is a datagram protocol, where each message is independent. As we saw earlier, filtering TCP requires reliance on the `ACK` bit, in order to distinguish between incoming calls and return packets from an outgoing call. But UDP has no such indicator: We are forced to rely on the source port number, which is subject to forgery.

An example will illustrate the problem. Suppose an internal host wishes to query the UDP *echo* server on some outside machine. The originating packet would carry the address

$$\langle \textit{localhost, localport, remotehost, 7} \rangle,$$

where *localport* is in the high-numbered range. But the reply would be

$$\langle \textit{remotehost, 7, localhost, localport} \rangle,$$

and the router would have no idea that *localport* was really a safe destination. An incoming packet

$$\langle \textit{remotehost, 7, localhost, 2049} \rangle$$

is probably an attempt to subvert our NFS server; and, while we could list the known dangerous destinations, we do not know what new targets will be added next week by a system administrator in the remote corners of our network. Worse yet, the RPC-based services use dynamic port numbers, sometimes in the high-numbered range. As with TCP, indirectly named services are not amenable to protection by packet filters.

A dynamic packet filter can do a better job by pairing up responses with queries. Most use a timeout to indicate that the "connection" is over. For some protocols, a simple counter will suffice: Only one response should be sent for most queries.

Barring a good dynamic packet filter, a conservative stance dictates that we ban virtually all *outgoing* UDP calls. It is not that the requests themselves are dangerous; rather, it is that we cannot trust the responses. The only exceptions are those protocols that provide a peer-to-peer relationship. A good example is NTP, the Network Time Protocol. In normal operation, messages are both from and to port 123. It is thus easy to admit replies, because they are to a fixed port number, rather than to an anonymous high-numbered port. One use of NTP—setting the clock when rebooting—will not work, because the client program will not use port 123. (Of course, a booting computer probably shouldn't ask an outsider for the time.)

The filtering rule for UDP can be summarized as follows:

protocol	out	in	comment
UDP	block	block	*Hard to distinguish spoof query from a reply*

10.3.2 H.323 and SIP

Meeting people on the Net is nice, but it's not too nice to firewalls. H.323 has several problems: It requires a complex proxy that can interpret the control messages, it requires the firewall to open additional ports (always a threat, just as with FTP), and the additional ports are UDP. SIP shares some of these attributes, but the code is a lot simpler.

Turn off inbound and outbound H.323. Use SIP for your multimedia needs. The rule is as follows:

protocol	out	in	comment
H.323	block	block	*Use the phone?*

10.3.3 RealAudio

The question to ask is if you have a strong business need to use RealAudio. If you must support it, use the TCP option. RealAudio servers, for outsider access, should be in the DMZ. The rule for filtering RealAudio is as follows:

protocol	out	in	comment
RealAudio	block	block	*If must turn on, use TCP option*

Fortunately, the RealAudio program seems to do the right thing more or less automatically.

10.3.4 SMB

Server Message Block (SMB) is a protocol that assumes a trusted environment. It provides an abstraction for sharing files and other devices. It is not the kind of thing that you want going into or out of a trust perimeter. Here is the filtering rule:

protocol	out	in	comment
SMB	block	block	

10.3.5 X Windows

Don't try to filter X Windows; tunnel it over *ssh*. Furthermore, make sure the clients are running on trustworthy machines.

10.4 Other Services

10.4.1 IPsec, GRE, and IP over IP

Each of these protocols is designed to carry IP within some other protocol. In other words, they create new wires that bypass your firewall. Although this can be a good idea under certain carefully controlled circumstances—see Section 12.1—you *must* block random tunnels. Even for controlled ones, the only type we trust is IPsec.

10.4.2 ICMP

There have been instances of hackers abusing ICMP for denial-of-service attacks. Nonetheless, filtering out ICMP denies one useful information. At the very least, internal management hosts should be allowed to receive such messages so that they can perform network diagnostic functions. For example, *traceroute* relies on the receipt of `Time Exceeded` and `Port Invalid` ICMP packets.

Some routers can distinguish between "safe" and "unsafe" ICMP messages, or permit the filter to specify the message types explicitly. This enables more of your machines to send and respond to things like *ping* requests. Conversely, it lets an outsider map your network if you're not using a dynamic packet filter that properly matches responses to outbound packets or connections.

Some ICMP cannot be blocked. Path MTU discovery is a must-have, and the ICMP messages it uses must be allowed in or you won't be able to talk to certain sites. Fraudulent `Destination Unreachable` messages can lead to a denial-of-service attack, but letting them in can improve performance. There is a trade-off; the price of learning that a destination is unreachable is that you risk being flooded with ICMP messages and perhaps having some connections torn down.

ICMP provides all sorts of functionality versus security trade-offs. Some firewalking techniques (see Section 11.4.5) use Path MTU ICMP messages. Which do you prefer: random black holes or being firewalked?

The filtering rule for ICMP can be summarized as follows:

protocol	out	in	comment
ICMP	allow	some	*Path MTU requires it, as do other useful services*

10.5 Something New

Suppose someone comes to you and asks that the *frobozz* protocol be allowed through the firewall. What do you do? There are no simple answers, but we can describe the guidelines we use to evaluate such requests.

The first question, of course, is whether the calls are inbound or outbound. Outbound calls present many fewer problems, though of course the nature of the service can change that. But it's hard to imagine something worse than *ssh*'s remote port-forwarding in the hands of an uncooperative employee, who could easily connect port 110—POP3–on some outside machine to port 110 on an inside machine. Here, education is your best choice.

For inbound services, our answer is usually "block." Because that rarely persuades people, we generally ask a few more questions. Can the destination machine reside in the DMZ? Often, it can, but only at the cost of opening a different hole through the firewall. This is generally a good trade-off, because an attacker will have to penetrate two different protocols to breach your firewall. Conversely, it means that you have yet another possibly vulnerable machine in your DMZ, with more access to other DMZ machines. Separating the DMZ into separate subnets is a good idea, if you can afford it.

Does it use TCP or UDP? Does it use fixed ports or random ports? TCP is generally easier to control. Fixed ported are easier to identify and filter appropriately.

Does the *frobozz* protocol use encryption and cryptographic authentication? If so—and if the crypto is an off-the-shelf standard, rather than something home-brewed—we think more favorably of it. That's especially true if the crypto restricts connectivity to a few selected outside sites. We don't want to trust outsiders, but we'd rather trust a few than trust the entire Internet.

What is the software like? Has it been through a security review? Much more evil lurks in code than in protocols. We like things written in Java, because the Java language prevents buffer overflows, but it's possible to write insecure code in any language. Does the software require *root* or *Administrator* privileges? Remarkably little code *really* needs it; often, the requirement is a sign of programmer laziness.

Does the service try to emulate numerous users? If so, it requires more privileges and more passwords or other credentials; that makes it more dangerous. We especially don't like to *store* such credentials in the DMZ.

Can the application be jailed safely? How easy is it to use *chroot* to contain it? Can other outboard security mechanisms be layered on top of it?

Finally, how strong is the business case for it? (If you're at a university, read "educational mission" for "business case.") We're much more likely to approve something that's part of a product offering than, for example, the latest and greatest MP3-swapping program.